Oracle® Database
ユーティリティ
19c

F16105-10(原本部品番号:E96081-18)

2023年12月

タイトルおよび著作権情報

Oracle Databaseユーティリティ, 19c

F16105-10

Copyright © 2002, 2023, Oracle and/or its affiliates.

原本著者: Douglas Williams

原本協力者: William Beauregard、Steve DiPirro、John Kalogeropoulos、Rod Payne、Rich Phillips、Mike Sakayeda、Jim Stenoish、Roy Swonger

目次

- 例一覧
- 図一覧
- 表一覧
- タイトルおよび著作権情報
- はじめに
  - 対象読者
  - ドキュメントのアクセシビリティ
  - ダイバーシティ&インクルージョン
  - 関連ドキュメント
  - 構文図
  - 表記規則
- 第I部 Oracle Data Pump
  - 1 Oracle Data Pumpの概要
    - 1.1 Oracle Data Pumpのコンポーネント
    - 1.2 Oracle Data Pumpによるデータの移動方法
      - 1.2.1 データ・ファイル・コピーを使用したデータ移動
      - 1.2.2 ダイレクト・パスを使用したデータ移動
      - 1.2.3 外部表を使用したデータ移動
      - 1.2.4 従来型パスを使用したデータ移動
      - 1.2.5 ネットワーク・リンク・インポートを使用したデータ移動
    - 1.3 CDBでのOracle Data Pumpの使用
      - 1.3.1 マルチテナント環境でのOracle Data Pumpの使用について
      - 1.3.2 データ・ポンプを使用したCDBへのデータの移動
      - 1.3.3 Oracle Data Pumpの使用によるCDB内または間のPDBの移動
    - 1.4 Oracle Data Pumpのエクスポート操作およびインポート操作に必要なロール
    - 1.5 Oracle Data Pumpジョブの実行時の処理について
      - 1.5.1 Oracle Data Pumpジョブの調整
      - 1.5.2 ジョブ内での進捗状況の追跡
      - 1.5.3 Oracle Data Pumpジョブ実行中のデータおよびメタデータのフィルタ処理
      - 1.5.4 Oracle Data Pumpジョブ実行中のメタデータの変換
      - 1.5.5 Oracle Data Pumpジョブのパフォーマンスの最大化
      - 1.5.6 Oracle Data Pumpによるデータのロードおよびアンロード
    - 1.6 Oracle Data Pumpジョブのステータスの監視方法
    - 1.7 ジョブ実行の進捗状況の監視方法
    - 1.8 Oracle Data Pumpによるファイルの割当て
      - 1.8.1 Oracle Data Pumpでのファイル割当ての理解
      - 1.8.2 ファイルの指定およびダンプ・ファイルの追加
      - 1.8.3 ダンプ・ファイル、ログ・ファイルおよびSQLファイルのデフォルトの位置
        - 1.8.3.1 ダンプ・ファイル、ログ・ファイルおよびSQLファイルのデフォルトの位置の理解
        - 1.8.3.2 Oracle RACでのOracle Data Pumpの使用方法の理解

- 1.8.3.3 Oracle Automatic Storage Managementを使用可能にした場合のディレクトリ・オブジェクトの使用方法
- 1.8.3.4 DATA_PUMP_DIRディレクトリ・オブジェクトおよびプラガブル・データベース
- 1.8.4 Oracle Data Pump Exportでの代替変数の使用
- 1.9 異なるOracle Databaseリリース間のエクスポートとインポート
- 1.10 Oracle Data Pumpを使用したSecureFilesラージ・オブジェクトのエクスポートの管理
- 1.11 Oracle Data Pumpプロセス終了コード
- 1.12 統合監査を使用してOracle Data Pumpジョブを監視する方法
- 1.13 Oracle Data Pump操作に対する暗号化されたデータのセキュリティ警告
- 1.14 Oracle Data Pumpによるタイムスタンプ・データの処理方法
- 1.14.1 TIMESTAMP WITH TIMEZONEの制限事項
- 1.14.1.1 TIMESTAMP WITH TIME ZONEの制限事項の理解
- 1.14.1.2 Oracle Data PumpによるTIMESTAMP WITH TIME ZONEデータのサポート
- 1.14.1.3 ソースおよびターゲットのタイムゾーン・ファイルのバージョン
- 1.14.2 TIMESTAMP WITH LOCAL TIME ZONEの制限事項
- 1.15 文字セットおよびグローバリゼーション・サポートに関する考慮点
- 1.15.1 DDL
- 1.15.2 シングルバイト文字セットとエクスポートおよびインポート
- 1.15.3 マルチバイト文字セットとエクスポートおよびインポート
- 1.16 データ・バインドされた照合に関連するOracle Data Pumpの動作
- 2 Oracle Data Pumpエクスポート
- 2.1 Oracle Data Pumpエクスポートとは
- 2.2 Oracle Data Pumpエクスポートの起動
- 2.2.1 Oracle Data Pumpエクスポートのインタフェース
- 2.2.2 Oracle Data Pump Exportモード
- 2.2.2.1 全体モード
- 2.2.2.2 スキーマ・モード
- 2.2.2.3 表モード
- 2.2.2.4 表領域モード
- 2.2.2.5 トランスポータブル表領域モード
- 2.2.3 Oracle Data Pumpエクスポートのネットワークに関する考慮事項
- 2.3 エクスポート操作中のフィルタ処理
- 2.3.1 Oracle Data Pumpエクスポートのデータ・フィルタ
- 2.3.2 メタデータ・フィルタ
- 2.4 データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ
- 2.4.1 データ・ポンプ・エクスポート・パラメータについて
- 2.4.2 ABORT_STEP
- 2.4.3 ACCESS_METHOD
- 2.4.4 ATTACH
- 2.4.5 CLUSTER
- 2.4.6 COMPRESSION

- 2.4.7 COMPRESSION_ALGORITHM
- 2.4.8 CONTENT
- 2.4.9 DATA_OPTIONS
- 2.4.10 DIRECTORY
- 2.4.11 DUMPFILE
- 2.4.12 ENABLE_SECURE_ROLES
- 2.4.13 ENCRYPTION
- 2.4.14 ENCRYPTION_ALGORITHM
- 2.4.15 ENCRYPTION_MODE
- 2.4.16 ENCRYPTION_PASSWORD
- 2.4.17 ENCRYPTION_PWD_PROMPT
- 2.4.18 ESTIMATE
- 2.4.19 ESTIMATE_ONLY
- 2.4.20 EXCLUDE
- 2.4.21 FILESIZE
- 2.4.22 FLASHBACK_SCN
- 2.4.23 FLASHBACK_TIME
- 2.4.24 FULL
- 2.4.25 HELP
- 2.4.26 INCLUDE
- 2.4.27 JOB_NAME
- 2.4.28 KEEP_MASTER
- 2.4.29 LOGFILE
- 2.4.30 LOGTIME
- 2.4.31 METRICS
- 2.4.32 NETWORK_LINK
- 2.4.33 NOLOGFILE
- 2.4.34 PARALLEL
- 2.4.35 PARALLEL_THRESHOLD
- 2.4.36 PARFILE
- 2.4.37 QUERY
- 2.4.38 REMAP_DATA
- 2.4.39 REUSE_DUMPFILES
- 2.4.40 SAMPLE
- 2.4.41 SCHEMAS
- 2.4.42 SERVICE_NAME
- 2.4.43 SOURCE_EDITION
- 2.4.44 STATUS
- 2.4.45 TABLES
- 2.4.46 TABLESPACES
- 2.4.47 TRANSPORT_FULL_CHECK
- 2.4.48 TRANSPORT_TABLESPACES
- 2.4.49 TRANSPORTABLE

- 2.4.50 TTS_CLOSURE_CHECK
- 2.4.51 VERSION
- 2.4.52 VIEWS_AS_TABLES
- 2.5 データ・ポンプ・エクスポートの対話方式コマンド・モードで使用可能なコマンド
  - 2.5.1 Oracle Data Pump Exportの対話方式コマンド・モードについて
  - 2.5.2 ADD_FILE
  - 2.5.3 CONTINUE_CLIENT
  - 2.5.4 EXIT_CLIENT
  - 2.5.5 FILESIZE
  - 2.5.6 HELP
  - 2.5.7 KILL_JOB
  - 2.5.8 PARALLEL
  - 2.5.9 START_JOB
  - 2.5.10 STATUS
  - 2.5.11 STOP_JOB
- 2.6 Oracle Data Pumpエクスポートの使用例
  - 2.6.1 表モード・エクスポートの実行
  - 2.6.2 選択した表および行のデータのみのアンロード
  - 2.6.3 表モード・エクスポートに必要なディスク領域の見積り
  - 2.6.4 スキーマ・モード・エクスポートの実行
  - 2.6.5 パラレル全データベース・エクスポートの実行
  - 2.6.6 対話方式モードを使用したジョブの停止および再接続
- 2.7 Oracle Data Pumpエクスポートの構文図
- 3 Oracleデータ・ポンプ・インポート
  - 3.1 Oracle Data Pumpインポートとは
  - 3.2 Oracle Data Pumpインポートの開始
    - 3.2.1 Oracle Data Pump Importのインタフェース
    - 3.2.2 Oracle Data Pump Importのモード
      - 3.2.2.1 Oracle Data Pump Importのモードについて
      - 3.2.2.2 全体インポート・モード
      - 3.2.2.3 スキーマ・モード
      - 3.2.2.4 表モード
      - 3.2.2.5 表領域モード
      - 3.2.2.6 トランスポータブル表領域モード
    - 3.2.3 Oracleデータ・ポンプ・インポートのネットワークに関する考慮事項
  - 3.3 インポート操作中のフィルタ処理
    - 3.3.1 データ・フィルタ
    - 3.3.2 メタデータ・フィルタ
  - 3.4 Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ
    - 3.4.1 インポート・コマンドライン・モードについて
    - 3.4.2 ABORT_STEP
    - 3.4.3 ACCESS_METHOD
    - 3.4.4 ATTACH

- 3.4.5 CLUSTER
- 3.4.6 CONTENT
- 3.4.7 CREDENTIAL
- 3.4.8 DATA_OPTIONS
- 3.4.9 DIRECTORY
- 3.4.10 DUMPFILE
- 3.4.11 ENABLE_SECURE_ROLES
- 3.4.12 ENCRYPTION_PASSWORD
- 3.4.13 ENCRYPTION_PWD_PROMPT
- 3.4.14 ESTIMATE
- 3.4.15 EXCLUDE
- 3.4.16 FLASHBACK_SCN
- 3.4.17 FLASHBACK_TIME
- 3.4.18 FULL
- 3.4.19 HELP
- 3.4.20 INCLUDE
- 3.4.21 JOB_NAME
- 3.4.22 KEEP_MASTER
- 3.4.23 LOGFILE
- 3.4.24 LOGTIME
- 3.4.25 MASTER_ONLY
- 3.4.26 METRICS
- 3.4.27 NETWORK_LINK
- 3.4.28 NOLOGFILE
- 3.4.29 PARALLEL
- 3.4.30 PARALLEL_THRESHOLD
- 3.4.31 PARFILE
- 3.4.32 PARTITION_OPTIONS
- 3.4.33 QUERY
- 3.4.34 REMAP_DATA
- 3.4.35 REMAP_DATAFILE
- 3.4.36 REMAP_DIRECTORY
- 3.4.37 REMAP_SCHEMA
- 3.4.38 REMAP_TABLE
- 3.4.39 REMAP_TABLESPACE
- 3.4.40 SCHEMAS
- 3.4.41 SERVICE_NAME
- 3.4.42 SKIP_UNUSABLE_INDEXES
- 3.4.43 SOURCE_EDITION
- 3.4.44 SQLFILE
- 3.4.45 STATUS
- 3.4.46 STREAMS_CONFIGURATION
- 3.4.47 TABLE_EXISTS_ACTION

- 3.4.48 REUSE_DATAFILES
- 3.4.49 TABLES
- 3.4.50 TABLESPACES
- 3.4.51 TARGET_EDITION
- 3.4.52 TRANSFORM
- 3.4.53 TRANSPORT_DATAFILES
- 3.4.54 TRANSPORT_FULL_CHECK
- 3.4.55 TRANSPORT_TABLESPACES
- 3.4.56 TRANSPORTABLE
- 3.4.57 VERSION
- 3.4.58 VIEWS_AS_TABLES (ネットワーク・インポート)

- 3.5 Oracle Data Pump Import対話方式コマンド・モードで使用可能なコマンド
  - 3.5.1 Oracle Data Pump Importの対話方式コマンド・モードについて
  - 3.5.2 CONTINUE_CLIENT
  - 3.5.3 EXIT_CLIENT
  - 3.5.4 HELP
  - 3.5.5 KILL_JOB
  - 3.5.6 PARALLEL
  - 3.5.7 START_JOB
  - 3.5.8 STATUS
  - 3.5.9 STOP_JOB
- 3.6 Oracle Data Pumpインポートの使用例
  - 3.6.1 データのみ表モード・インポートの実行
  - 3.6.2 スキーマ・モード・インポートの実行
  - 3.6.3 ネットワーク・モード・インポートの実行
  - 3.6.4 URLベースのダンプ・ファイル名でのワイルドカードの使用
- 3.7 Oracle Data Pump Importの構文図

- 4 Oracle Data Pumpのレガシー・モード
  - 4.1 Oracle Data Pumpのレガシー・モードのユースケース
  - 4.2 パラメータのマッピング
    - 4.2.1 データ・ポンプでのオリジナルのエクスポート・パラメータの使用
    - 4.2.2 データ・ポンプでのオリジナルのインポート・パラメータの使用
  - 4.3 データ・ポンプ・レガシー・モードでのファイルの場所の管理
  - 4.4 Oracle Data Pumpのログ・ファイルとエラーに対する既存のスクリプトの調整
    - 4.4.1 ログ・ファイル
    - 4.4.2 エラー・ケース
    - 4.4.3 終了状態

- 5 Oracle Data Pumpのパフォーマンス
  - 5.1 Oracle Data Pumpエクスポートおよびインポートのデータ・パフォーマンスの改善点
  - 5.2 パフォーマンス・チューニング
    - 5.2.1 リソース消費の制御
    - 5.2.2 圧縮および暗号化によるパフォーマンスへの影響
    - 5.2.3 統計のエクスポートおよびインポートにおけるメモリーの考慮事項

- 5.3 データ・ポンプ・ユーティリティのパフォーマンスに影響する初期化パラメータ
    - 5.3.1 GoldenGateレプリケーション環境におけるバッファ・キャッシュのサイズの設定
    - 5.3.2 複数ユーザーによるデータ・ポンプ・ジョブのリソース使用率の管理
- 6 Oracle Data Pump APIの使用
    - 6.1 Oracle Data Pumpのクライアント・インタフェースAPIの動作
        - 6.1.1 DBMS_DATAPUMPジョブの状態
    - 6.2 Oracle Data Pump APIを使用する場合の基本ステップ
    - 6.3 データ・ポンプAPIの使用例
- 第II部 SQL*Loader
    - 7 SQL*Loaderの使用方法の理解
    - 7.1 SQL*Loaderの機能
    - 7.2 SQL*Loaderパラメータ
    - 7.3 SQL*Loader制御ファイル
    - 7.4 SQL*Loaderによる入力データおよびデータ・ファイルの読取り方法
        - 7.4.1 固定レコード形式
        - 7.4.2 可変レコード形式およびSQL*Loader
        - 7.4.3 ストリームレコード形式およびSQL*Loader
        - 7.4.4 論理レコードおよびSQL*Loader
        - 7.4.5 データ・フィールド設定およびSQL*Loader
    - 7.5 LOBFILEおよびセカンダリ・データ・ファイル(SDF)
    - 7.6 データ変換およびデータ型の指定
    - 7.7 SQL*Loaderの廃棄レコードと拒否レコード
        - 7.7.1 SQL*Loader不良ファイル
            - 7.7.1.1 SQL*Loaderによって拒否されたレコード
            - 7.7.1.2 SQL*Loaderの操作中にOracle Databaseによって拒否されたレコード
        - 7.7.2 SQL*Loader廃棄ファイル
    - 7.8 ログ・ファイルおよびログ情報
    - 7.9 従来型パス・ロード、ダイレクト・パス・ロードおよび外部表ロード
        - 7.9.1 従来型パス・ロード
        - 7.9.2 ダイレクト・パス・ロード
        - 7.9.3 パラレル・ダイレクト・パス
        - 7.9.4 外部表ロード
        - 7.9.5 外部表およびSQL*Loaderの選択
        - 7.9.6 SQL*Loaderと外部表との処理内容の違い
            - 7.9.6.1 複数のプライマリ入力データ・ファイル
            - 7.9.6.2 構文およびデータ型
            - 7.9.6.3 バイト順序マーク
            - 7.9.6.4 デフォルトの文字セット、日付マスク、小数点区切り
            - 7.9.6.5 バックスラッシュ・エスケープ文字の使用
        - 7.9.7 オブジェクト・ストレージに格納されたデータを使用した表のロード
    - 7.10 SQL*Loaderによるオブジェクト、コレクションおよびLOBのロード
        - 7.10.1 サポートされるオブジェクト型

- 7.10.1.1 列オブジェクト
- 7.10.1.2 行オブジェクト
- 7.10.2 サポートされるコレクション型
- 7.10.2.1 ネストした表
- 7.10.2.2 VARRAY
- 7.10.3 サポートされるLOBデータ型
- 7.11 SQL*Loaderでのパーティション・オブジェクトのサポート
- 7.12 アプリケーション開発: ダイレクト・パス・ロードAPI
- 7.13 SQL*Loaderの事例
- 7.13.1 事例用ファイル
- 7.13.2 事例の実行
- 7.13.3 事例用ログ・ファイル
- 7.13.4 事例の結果確認
- 8 SQL*Loaderコマンドライン・リファレンス
- 8.1 SQL*Loaderの起動
- 8.1.1 コマンドラインでのパラメータの指定
- 8.1.2 SQL*Loaderのパラメータ指定の代替方法
- 8.1.3 ネットワークを介してデータをロードするためのSQL*Loaderの使用
- 8.2 SQL*Loaderのコマンドライン・パラメータ
- 8.2.1 BAD
- 8.2.2 BINDSIZE
- 8.2.3 COLUMNARRAYROWS
- 8.2.4 CONTROL
- 8.2.5 DATA
- 8.2.6 DATE_CACHE
- 8.2.7 DEFAULTS
- 8.2.8 DEGREE_OF_PARALLELISM
- 8.2.9 DIRECT
- 8.2.10 DIRECT_PATH_LOCK_WAIT
- 8.2.11 DISCARD
- 8.2.12 DISCARDMAX
- 8.2.13 DNFS_ENABLE
- 8.2.14 DNFS_READBUFFERS
- 8.2.15 EMPTY_LOBS_ARE_NULL
- 8.2.16 ERRORS
- 8.2.17 EXTERNAL_TABLE
- 8.2.18 FILE
- 8.2.19 HELP
- 8.2.20 LOAD
- 8.2.21 LOG
- 8.2.22 MULTITHREADING
- 8.2.23 NO_INDEX_ERRORS
- 8.2.24 PARALLEL

- 8.2.25 PARFILE
- 8.2.26 PARTITION_MEMORY
- 8.2.27 READSIZE
- 8.2.28 RESUMABLE
- 8.2.29 RESUMABLE_NAME
- 8.2.30 RESUMABLE_TIMEOUT
- 8.2.31 ROWS
- 8.2.32 SDF_PREFIX
- 8.2.33 SILENT
- 8.2.34 SKIP
- 8.2.35 SKIP_INDEX_MAINTENANCE
- 8.2.36 SKIP_UNUSABLE_INDEXES
- 8.2.37 STREAMSIZE
- 8.2.38 TRIM
- 8.2.39 USERID

- 8.3 終了コードによる結果の検査と表示

- 9 SQL*Loader制御ファイル・リファレンス
  - 9.1 制御ファイルの内容
  - 9.2 制御ファイルのコメント
  - 9.3 制御ファイル中でのコマンドライン・パラメータの指定
    - 9.3.1 OPTIONS句
    - 9.3.2 一度に評価されるデフォルト式の数の指定
  - 9.4 ファイル名およびオブジェクト名の指定
    - 9.4.1 SQLおよびSQL*Loaderの予約語と競合するファイル名
    - 9.4.2 SQL*Loader制御ファイルでのSQL文字列の指定
    - 9.4.3 オペレーティング・システムとSQL Loader制御ファイルの文字
      - 9.4.3.1 完全パスの指定
      - 9.4.3.2 バックスラッシュ・エスケープ文字
      - 9.4.3.3 移植不能文字列
      - 9.4.3.4 エスケープ文字としてのバックスラッシュの使用
      - 9.4.3.5 エスケープ文字が使用できない場合
  - 9.5 XMLType表の識別
  - 9.6 フィールド順序の指定
  - 9.7 データ・ファイルの指定
    - 9.7.1 データ・ファイルの指定方法の理解
    - 9.7.2 INFILE構文の例
    - 9.7.3 複数のデータ・ファイルの指定
  - 9.8 CSV形式のファイルの指定
  - 9.9 BEGINDATAによる制御ファイルのデータの識別
  - 9.10 データ・ファイル形式およびバッファリングの指定
  - 9.11 不良ファイルの指定
    - 9.11.1 不良ファイルの理解と指定
    - 9.11.2 不良ファイル名指定の例

- 9.11.3 LOBFILEおよびSDFを使用した不良ファイルの処理方法
- 9.11.4 拒否レコードの条件
- 9.12 廃棄ファイルの指定
  - 9.12.1 廃棄ファイルの理解と指定
  - 9.12.2 制御ファイルでの廃棄ファイルの指定
  - 9.12.3 廃棄レコード数の上限付け
  - 9.12.4 廃棄ファイル名指定の例
  - 9.12.5 廃棄レコードの条件
  - 9.12.6 LOBFILEおよびSDFを使用した廃棄ファイルの処理方法
  - 9.12.7 コマンドラインからの廃棄ファイルの指定
- 9.13 表レベルでのNULLIF句の指定
- 9.14 表レベルでの日時書式の指定
- 9.15 異なる文字コード体系の処理
  - 9.15.1 マルチバイト(アジア系言語)文字セット
  - 9.15.2 Unicode文字セット
  - 9.15.3 データベース文字セット
  - 9.15.4 データ・ファイル文字セット
  - 9.15.5 入力文字の変換
    - 9.15.5.1 VARRAYまたは主キー・ベースREFへのデータ・ロード時の考慮事項
    - 9.15.5.2 CHARACTERSETパラメータ
    - 9.15.5.3 制御ファイル文字セット
    - 9.15.5.4 文字長セマンティクス
  - 9.15.6 シフト・センシティブ文字データ
- 9.16 中断されたロード
  - 9.16.1 中断された従来型パス・ロード
  - 9.16.2 中断されたダイレクト・パス・ロード
    - 9.16.2.1 領域エラーによって中断されたロード
    - 9.16.2.2 エラーが最大数を超えたことによって中断されたロード
    - 9.16.2.3 致命的エラーによって中断されたロード
    - 9.16.2.4 [Ctrl]キーを押しながら[C]キーを押すことによって中断されたロード
  - 9.16.3 ロードの中断後の表および索引の状態
  - 9.16.4 ログ・ファイルを使用したロード状態の確認
  - 9.16.5 単一表へのロードの継続
- 9.17 物理レコードからの論理レコードの作成
  - 9.17.1 CONCATENATEを使用した論理レコードの作成
  - 9.17.2 CONTINUEIFを使用した論理レコードの作成
- 9.18 表への論理レコードのロード
  - 9.18.1 表名の指定
    - 9.18.1.1 INTO TABLE句
  - 9.18.2 表固有のロード方法
    - 9.18.2.1 INSERTを使用した空の表へのデータのロード
    - 9.18.2.2 空でない表へのデータのロード
      - 9.18.2.2.1 APPEND

- 9.18.2.2.2 REPLACE
- 9.18.2.2.3 既存の行の更新
- 9.18.2.2.4 TRUNCATE
- 9.18.3 表固有のOPTIONSパラメータ
- 9.18.4 条件に基づいたレコードのロード
- 9.18.4.1 LOBFILEおよびSDFでのWHEN句の使用
- 9.18.5 デフォルトのデータ・デリミタ(区切り記号)の指定
- 9.18.5.1 fields_spec
- 9.18.5.2 termination_spec
- 9.18.5.3 enclosure_spec
- 9.18.6 データが欠落しているショート・レコードの処理
- 9.18.6.1 TRAILING NULLCOLS句
- 9.19 索引オプション
- 9.19.1 SORTED INDEXES句
- 9.19.2 SINGLEROWオプション
- 9.20 複数のINTO TABLE句を使用するメリット
- 9.20.1 複数の論理レコードの抽出
- 9.20.1.1 デリミタに基づく相対的な位置指定
- 9.20.2 異なる入力レコード形式の区別
- 9.20.2.1 POSITIONパラメータに基づく相対的な位置指定
- 9.20.3 異なる入力行オブジェクトのサブタイプの区別
- 9.20.4 複数表へのデータのロード
- 9.20.5 複数のINTO TABLE句使用のサマリー
- 9.21 バインド配列および従来型パス・ロード
- 9.21.1 バインド配列のサイズ要件
- 9.21.2 バインド配列のパフォーマンスに関する考慮点
- 9.21.3 行数およびバインド配列サイズの指定
- 9.21.4 バインド配列サイズを確認するための計算
- 9.21.4.1 長さインジケータのサイズの決定
- 9.21.4.2 フィールド・バッファ・サイズの計算
- 9.21.5 バインド配列用のメモリー所要量の最小化
- 9.21.6 複数のINTO TABLE句に対するバインド配列サイズの計算
- 10 SQL*Loaderフィールド・リスト・リファレンス
- 10.1 フィールド・リストの内容
- 10.2 データ・フィールドの位置指定
- 10.2.1 タブを含むデータでのPOSITIONの使用
- 10.2.2 複数表のロードでのPOSITIONの使用
- 10.2.3 POSITIONを使用した例
- 10.3 列およびフィールドの指定
- 10.3.1 FILLERフィールドの指定
- 10.3.2 データ・フィールドのデータ型の指定
- 10.4 SQL*Loaderのデータ型
- 10.4.1 移植可能なデータ型と移植不能なデータ型の相違点

- 10.4.2 移植不能なデータ型
  - 10.4.2.1 移植不能なデータ型のカテゴリ
  - 10.4.2.2 INTEGER(n)
  - 10.4.2.3 SMALLINT
  - 10.4.2.4 FLOAT
  - 10.4.2.5 DOUBLE
  - 10.4.2.6 BYTEINT
  - 10.4.2.7 ZONED
  - 10.4.2.8 DECIMAL
  - 10.4.2.9 VARGRAPHIC
  - 10.4.2.10 VARCHAR
  - 10.4.2.11 VARRAW
  - 10.4.2.12 LONG VARRAW
- 10.4.3 移植可能なデータ型
  - 10.4.3.1 移植不能なデータ型のカテゴリ
  - 10.4.3.2 CHAR
  - 10.4.3.3 日時データ型および期間データ型
    - 10.4.3.3.1 日時および間隔のデータ型のカテゴリ
    - 10.4.3.3.2 DATE
    - 10.4.3.3.3 TIME
    - 10.4.3.3.4 TIMEWITHTIMEZONE
    - 10.4.3.3.5 TIMESTAMP
    - 10.4.3.3.6 TIMEWITHTIMEZONE
    - 10.4.3.3.7 TIMESTAMPWITHLOCALTIMEZONE
    - 10.4.3.3.8 INTERVALYEARTOMONTH
    - 10.4.3.3.9 INTERVALDAYTOSECOND
  - 10.4.3.4 GRAPHIC
  - 10.4.3.5 GRAPHIC EXTERNAL
  - 10.4.3.6 数値型EXTERNAL
  - 10.4.3.7 RAW
  - 10.4.3.8 VARCHARC
  - 10.4.3.9 VARRAWC
  - 10.4.3.10 システム固有のデータ型フィールド長の競合
  - 10.4.3.11 LENGTH-VALUEデータ型のフィールド長
- 10.4.4 データ型の変換
- 10.4.5 日時データ型および期間データ型のデータ型変換
- 10.4.6 デリミタの指定
  - 10.4.6.1 終了および囲みの指定に関する構文
  - 10.4.6.2 データ中のデリミタ記号
  - 10.4.6.3 デリミタ付きデータの最大長
  - 10.4.6.4 デリミタを使用した後続の空白のロード
- 10.4.7 デリミタ付きデータの処理方法
  - 10.4.7.1 TERMINATED BYのみを使用するフィールド

- 10.4.7.2 TERMINATED BYなしでENCLOSED BYを使用するフィールド
- 10.4.7.3 TERMINATED BYとともにENCLOSED BYを使用するフィールド
- 10.4.7.4 TERMINATED BYとともにOPTIONALLY ENCLOSED BYを使用するフィールド
- 10.4.8 文字データ型フィールド長の競合
- 10.4.8.1 事前にサイズが決まっているフィールド
- 10.4.8.2 デリミタ付きフィールド
- 10.4.8.3 日付フィールド・マスク
- 10.5 フィールド条件の指定
- 10.5.1 フィールド条件の構文図
- 10.5.2 フィールドとBLANKSの比較
- 10.5.3 フィールドと文字列の比較
- 10.6 WHEN、NULLIFおよびDEFAULTIF句の使用
- 10.7 WHEN、NULLIFおよびDEFAULTIF句の使用
- 10.8 異なるプラットフォーム間でのデータのロード
- 10.9 バイト順序
- 10.9.1 SQL*Loaderによるバイト順序の管理方法の理解
- 10.9.2 バイト順序の構文
- 10.9.3 バイト順序マーク(BOM)の使用
- 10.9.4 BOMの確認の抑止
- 10.10 すべてが空白のフィールドのロード
- 10.11 空白の切捨て
- 10.11.1 空白の切捨てが可能なデータ型
- 10.11.2 空白の切捨てが可能なデータ型に対するフィールド長指定
- 10.11.2.1 事前にサイズが決まっているフィールド
- 10.11.2.2 デリミタ付きフィールド
- 10.11.3 フィールドの相対的な位置指定
- 10.11.3.1 フィールドの開始位置が指定されていない場合
- 10.11.3.2 前のフィールドの終端がデリミタで指定されている場合
- 10.11.3.3 前のフィールドに囲みデリミタおよび終了デリミタの両方が含まれる場合
- 10.11.4 先頭の空白
- 10.11.4.1 前のフィールドが空白で区切られている場合
- 10.11.4.2 オプションの囲みデリミタ
- 10.11.5 後続の空白の切捨て
- 10.11.6 囲まれたフィールドの切捨て
- 10.12 空白の切捨てに対するPRESERVE BLANKSオプションの影響
- 10.13 [NO] PRESERVE BLANKSとデリミタ句の併用
- 10.14 フィールドへのSQL演算子の適用
- 10.14.1 フィールドの参照
- 10.14.2 フィールド指定でのSQL演算子の共通使用
- 10.14.3 SQL演算子の組合せ
- 10.14.4 日付マスク付きのSQL文字列の使用

- 10.14.5 書式化されたフィールドの解析
- 10.14.6 SQL文字列を使用したANYDATAデータベース型へのロード
- 10.15 SQL*Loaderを使用した入力データの生成
  - 10.15.1 ファイルを使用しないデータのロード
  - 10.15.2 列への定数値の設定
    - 10.15.2.1 CONSTANTパラメータ
  - 10.15.3 列への式の値の設定
    - 10.15.3.1 EXPRESSIONパラメータ
  - 10.15.4 データファイル・レコード番号への1列の設定
    - 10.15.4.1 RECNUMパラメータ
  - 10.15.5 列への現在の日付の設定
    - 10.15.5.1 SYSDATEパラメータ
  - 10.15.6 列への一意の順序番号の設定
    - 10.15.6.1 SEQUENCEパラメータ
  - 10.15.7 複数の表に対する順序番号の生成
    - 10.15.7.1 例: 挿入ごとの順序番号の生成
- 11 SQL*Loaderを使用したオブジェクト、LOBおよびコレクションのロード
  - 11.1 列オブジェクトのロード
    - 11.1.1 ストリーム・レコード形式への列オブジェクトのロード
    - 11.1.2 可変レコード形式への列オブジェクトのロード
    - 11.1.3 ネストした列オブジェクトのロード
    - 11.1.4 導出サブタイプを使用した列オブジェクトのロード
    - 11.1.5 オブジェクトに対するNULL値の指定
      - 11.1.5.1 NULL属性の指定
      - 11.1.5.2 アトミックNULLの指定
    - 11.1.6 ユーザー定義コンストラクタを使用した列オブジェクトのロード
  - 11.2 オブジェクト表のロード
    - 11.2.1 サブタイプを使用したオブジェクト表のロード
  - 11.3 SQL*Loaderを使用したREF列のロード
    - 11.3.1 REF句における表名の指定
    - 11.3.2 システム生成OID型のREF列
    - 11.3.3 主キーREF列
    - 11.3.4 主キーが使用可能な有効範囲なしREF列
  - 11.4 SQL*Loaderを使用したLOBのロード
    - 11.4.1 「プライマリ・データ・ファイルからのLOBデータのロード」
      - 11.4.1.1 事前に決められたサイズのフィールドのLOBデータ
      - 11.4.1.2 デリミタ付きフィールドのLOBデータ
      - 11.4.1.3 Length-Value PairフィールドのLOBデータ
    - 11.4.2 LOBFILEからのLOBデータのロード
      - 11.4.2.1 動的および静的LOBFILE指定
      - 11.4.2.2 LOBFILEからのLOBデータのロードの例
        - 11.4.2.2.1 ファイルごとに1つのLOB
        - 11.4.2.2.2 事前に決められたサイズのLOB

- 11.4.2.2.3 デリミタ付きフィールドのLOB
- 11.4.2.2.4 Length-Value Pairで指定したLOB
- 11.4.2.3 LOBFILEからLOBをロードする場合の考慮点
- 11.4.3 LLSフィールドを含むデータ・ファイルのロード
- 11.5 SQL*Loaderを使用したBFILE列のロード
- 11.6 コレクション(ネストした表およびVARRAY)のロード
- 11.6.1 ネストした表およびVARRAYでの制限事項
- 11.6.2 セカンダリ・データ・ファイル(SDF)
- 11.7 動的または静的SDF指定の選択
- 11.8 親表を子表から分割してのロード
- 11.8.1 VARRAY列ロード時のメモリーの問題
- 12 従来型パス・ロードおよびダイレクト・パス・ロード
- 12.1 データのロード方法
- 12.2 ROWID列のロード
- 12.3 従来型パス・ロード
- 12.3.1 単一パーティションの従来型パス・ロード
- 12.3.2 従来型パスを使用する場合
- 12.4 ダイレクト・パス・ロード
- 12.4.1 SQL*Loaderダイレクト・パス・ロードについて
- 12.4.2 ダイレクト・パス・ロード時のデータ変換
- 12.4.3 パーティション表またはサブパーティション表のダイレクト・パス・ロード
- 12.4.4 単一パーティションまたはサブパーティションのダイレクト・パス・ロード
- 12.4.5 ダイレクト・パス・ロードのメリット
- 12.4.6 ダイレクト・パス・ロード使用上の制限
- 12.4.7 単一パーティションのダイレクト・パス・ロードでの制限
- 12.4.8 ダイレクト・パスを使用する場合
- 12.4.9 整合性制約
- 12.4.10 ダイレクト・パスのフィールド・デフォルト
- 12.4.11 シノニムへのロード
- 12.5 ダイレクト・パス・ロードの使用
- 12.5.1 ダイレクト・パス・ロードのセットアップ
- 12.5.2 ダイレクト・パス・ロードの指定
- 12.5.3 索引の作成
- 12.5.3.1 パフォーマンスの向上
- 12.5.3.2 一時セグメント記憶域要件の計算
- 12.5.4 使用禁止状態(Index Unusable)のままの索引
- 12.5.5 データ・セーブによるデータ損失の防止
- 12.5.5.1 データ・セーブを使用したデータ損失の防止
- 12.5.5.2 ROWSパラメータの使用
- 12.5.5.3 データ・セーブとコミット
- 12.5.6 ダイレクト・パス・ロード時のデータ・リカバリ
- 12.5.6.1 メディア・リカバリおよびダイレクト・パス・ロード
- 12.5.6.2 インスタンス・リカバリおよびダイレクト・パス・ロード

- 12.5.7 LONG型データ・フィールドのロード
- 12.5.8 PIECEDとしてのデータのロード
- 12.5.9 ダイレクト・パス・モードを使用するSQL*Loader操作の監査
- 12.6 ダイレクト・パス・ロードのパフォーマンスの最適化
  - 12.6.1 ダイレクト・パス・ロードに必要な時間と領域の最小化
  - 12.6.2 高速ロードのための記憶域の事前割当て
  - 12.6.3 高速索引付けのためのデータの事前ソート
    - 12.6.3.1 データの事前ソートの利点
    - 12.6.3.2 SORTED INDEXES句
    - 12.6.3.3 未ソートのデータ
    - 12.6.3.4 複数列索引
    - 12.6.3.5 最適ソート順序の選択方法
  - 12.6.4 データ・セーブの回数の削減
  - 12.6.5 REDOログの最小限の使用
    - 12.6.5.1 アーカイブの使用禁止
    - 12.6.5.2 SQL*LoaderのUNRECOVERABLE句の指定
    - 12.6.5.3 SQL NOLOGGINGパラメータの設定
  - 12.6.6 列配列の行数およびストリーム・バッファ・サイズの指定
  - 12.6.7 DATE_CACHEの値の指定
- 12.7 複数CPUシステムのダイレクト・パス・ロードの最適化
- 12.8 索引メンテナンスの回避
- 12.9 ダイレクト・パス・ロード、整合性制約およびトリガー
  - 12.9.1 整合性制約
    - 12.9.1.1 使用可能な制約
    - 12.9.1.2 使用禁止の制約
    - 12.9.1.3 制約を使用可能に戻す方法
  - 12.9.2 挿入トリガー
    - 12.9.2.1 挿入トリガーの整合性制約への置換
    - 12.9.2.2 自動制約が使用できない場合
    - 12.9.2.3 データベース・トリガーの準備
    - 12.9.2.4 更新トリガーの使用
    - 12.9.2.5 例外処理と同じ処理の実現
    - 12.9.2.6 ストアド・プロシージャの使用
  - 12.9.3 永続的に使用禁止のトリガーおよび制約
  - 12.9.4 従来型パスの同時ロードによるパフォーマンスの向上
- 12.10 ダイレクト・パス・ロードのパフォーマンスの最適化
  - 12.10.1 SQL*Loaderのパラレル・データ・ロード・モデルについて
  - 12.10.2 従来型パスによる同時ロード
  - 12.10.3 ダイレクト・パスによるセグメント間同時処理
  - 12.10.4 ダイレクト・パスによるセグメント内同時処理
  - 12.10.5 パラレル・ダイレクト・パス・ロードの制限
  - 12.10.6 複数のSQL*Loaderセッションの初期化
  - 12.10.7 パラレル・ダイレクト・パス・ロードのパラメータ

- 12.10.7.1 FILEパラメータを使用した一時セグメントの指定
- 12.10.7.2 FILEパラメータの使用
- 12.10.7.3 STORAGEパラメータの使用方法
- 12.10.8 パラレル・ダイレクト・パス・ロード後の制約の使用可能化
- 12.10.9 主キー制約および一意キー制約
- 12.11 一般的なパフォーマンス改善のヒント
- 13 SQL*Loaderエクスプレス
- 13.1 SQL*Loaderエクスプレス・モードとは
- 13.2 SQL*Loaderエクスプレス・モードの使用方法
- 13.2.1 SQL*Loaderエクスプレス・モードで使用されるデフォルト値
- 13.2.2 SQL*Loaderエクスプレス・モードによるバイト順序の処理方法
- 13.3 SQL*Loaderエクスプレス・モードのパラメータ・リファレンス
- 13.3.1 BAD
- 13.3.2 CHARACTERSET
- 13.3.3 CSV
- 13.3.4 DATA
- 13.3.5 DATE_FORMAT
- 13.3.6 DEGREE_OF_PARALLELISM
- 13.3.7 DIRECT
- 13.3.8 DNFS_ENABLE
- 13.3.9 DNFS_READBUFFERS
- 13.3.10 ENCLOSED_BY
- 13.3.11 EXTERNAL_TABLE
- 13.3.12 FIELD_NAMES
- 13.3.13 LOAD
- 13.3.14 NULLIF
- 13.3.15 OPTIONALLY_ENCLOSED_BY
- 13.3.16 PARFILE
- 13.3.17 SILENT
- 13.3.18 TABLE
- 13.3.19 TERMINATED_BY
- 13.3.20 TIMESTAMP_FORMAT
- 13.3.21 TRIM
- 13.3.22 USERID
- 13.4 SQL*Loaderエクスプレス・モードの構文図
- 第III部 外部表
- 14 外部表の概要
- 14.1 外部表の作成方法
- 14.2 データ・ファイルおよび出力ファイルの位置
- 14.3 外部表のアクセス・パラメータ
- 14.4 外部表の使用時のデータ型の変換
- 15 ORACLE_LOADERアクセス・ドライバ
- 15.1 ORACLE_LOADERアクセス・ドライバについて

- 15.2 access_parameters句
- 15.3 record_format_info句
  - 15.3.1 record_format_info句の概要
  - 15.3.2 FIXED長
  - 15.3.3 VARIABLEサイズ
  - 15.3.4 DELIMITED BY
  - 15.3.5 XMLTAG
  - 15.3.6 CHARACTERSET
  - 15.3.7 EXTERNAL VARIABLE DATA
  - 15.3.8 PREPROCESSOR
    - 15.3.8.1 PREPROCESSOR句によるパラレル処理の使用
    - 15.3.8.2 PREPROCESSOR句を使用する際の制限事項
  - 15.3.9 LANGUAGE
  - 15.3.10 TERRITORY
  - 15.3.11 DATA IS...ENDIAN
  - 15.3.12 BYTEORDERMARK [CHECK | NOCHECK]
  - 15.3.13 STRING SIZES ARE IN
  - 15.3.14 LOAD WHEN
  - 15.3.15 BADFILE | NOBADFILE
  - 15.3.16 DISCARDFILE | NODISCARDFILE
  - 15.3.17 LOGFILE | NOLOGFILE
  - 15.3.18 SKIP
  - 15.3.19 FIELD NAMES
  - 15.3.20 READSIZE
  - 15.3.21 DISABLE_DIRECTORY_LINK_CHECK
  - 15.3.22 DATE_CACHE
  - 15.3.23 文字列
  - 15.3.24 condition_spec
  - 15.3.25 [directory object name:] [filename]
  - 15.3.26 条件
    - 15.3.26.1 range start : range end
  - 15.3.27 IO_OPTIONS句
  - 15.3.28 DNFS_DISABLE | DNFS_ENABLE
  - 15.3.29 DNFS_READBUFFERS
- 15.4 field_definitions句
  - 15.4.1 delim_spec
    - 15.4.1.1 例: 終了デリミタを含む外部表
    - 15.4.1.2 例: 囲みデリミタおよび終了デリミタを含む外部表
    - 15.4.1.3 例: オプションの囲みデリミタを含む外部表
  - 15.4.2 trim_spec
  - 15.4.3 MISSING FIELD VALUES ARE NULL
  - 15.4.4 field_list
  - 15.4.5 pos_spec句

- 15.4.5.1 start
- 15.4.5.2 *
- 15.4.5.3 increment
- 15.4.5.4 end
- 15.4.5.5 length
- 15.4.6 datatype_spec句の構文
  - 15.4.6.1 [UNSIGNED] INTEGER [EXTERNAL] [(len)]
  - 15.4.6.2 DECIMAL [EXTERNAL]およびZONED [EXTERNAL]
  - 15.4.6.3 ORACLE_DATE
  - 15.4.6.4 ORACLE_NUMBER
  - 15.4.6.5 浮動小数点数
  - 15.4.6.6 DOUBLE
  - 15.4.6.7 FLOAT [EXTERNAL]
  - 15.4.6.8 BINARY_DOUBLE
  - 15.4.6.9 BINARY_FLOAT
  - 15.4.6.10 RAW
  - 15.4.6.11 CHAR
  - 15.4.6.12 date_format_spec
    - 15.4.6.12.1 DATE
    - 15.4.6.12.2 MASK
    - 15.4.6.12.3 TIMESTAMP
    - 15.4.6.12.4 INTERVAL
  - 15.4.6.13 VARCHARおよびVARRAW
  - 15.4.6.14 VARCHARCおよびVARRAWC
- 15.4.7 init_spec句
- 15.4.8 LLS句
- 15.5 column_transforms句
  - 15.5.1 transform
    - 15.5.1.1 column_name FROM
    - 15.5.1.2 NULL
    - 15.5.1.3 CONSTANT
    - 15.5.1.4 CONCAT
    - 15.5.1.5 LOBFILE
    - 15.5.1.6 lobfile_attr_list
    - 15.5.1.7 STARTOF source_field (長さ)
- 15.6 ORACLE_LOADERアクセス・ドライバのパラレル・ロードに関する考慮点
- 15.7 ORACLE_LOADERアクセス・ドライバを使用する際のパフォーマンスのヒント
- 15.8 ORACLE_LOADERアクセス・ドライバ使用時の制限事項
- 15.9 ORACLE_LOADERアクセス・ドライバの予約語
- 16 ORACLE_DATAPUMPアクセス・ドライバ
  - 16.1 ORACLE_DATAPUMPアクセス・ドライバの使用
  - 16.2 access_parameters句
    - 16.2.1 コメント

- 16.2.2 COMPRESSION
- 16.2.3 ENCRYPTION
- 16.2.4 LOGFILE | NOLOGFILE
  - 16.2.4.1 パラレル・ロードのログ・ファイル・ネーミング
- 16.2.5 VERSION句
- 16.2.6 HADOOP_TRAILERS句
- 16.2.7 SQL ENCRYPT句の使用による影響
- 16.3 ORACLE_DATAPUMPアクセス・ドライバを使用したデータのアンロードとロード
  - 16.3.1 パラレル・ロードおよびパラレル・アンロード
  - 16.3.2 ダンプ・ファイルの結合
- 16.4 サポートされるデータ型
- 16.5 サポートされないデータ型
  - 16.5.1 BFILEデータ型のアンロードおよびロード
  - 16.5.2 LONGおよびLONG RAWデータ型のアンロード
  - 16.5.3 FINALオブジェクト型を含む列のアンロードおよびロード
  - 16.5.4 FINALオブジェクト型の表
- 16.6 ORACLE_DATAPUMPアクセス・ドライバを使用する際のパフォーマンスのヒント
- 16.7 ORACLE_DATAPUMPアクセス・ドライバ使用時の制限事項
- 16.8 ORACLE_DATAPUMPアクセス・ドライバの予約語

- 17 ORACLE_HDFSおよびORACLE_HIVEアクセス・ドライバ
  - 17.1 プロパティを指定するための構文ルール
  - 17.2 ORACLE_HDFSアクセス・パラメータ
    - 17.2.1 ORACLE_HDFSのデフォルトのパラメータ設定
    - 17.2.2 ORACLE_HDFSのオプションのパラメータ設定
  - 17.3 ORACLE_HIVEアクセス・パラメータ
    - 17.3.1 ORACLE_HIVEのデフォルトのパラメータ設定
    - 17.3.2 ORACLE_HIVEのオプションのパラメータ設定
  - 17.4 com.oracle.bigdataパラメータの説明
    - 17.4.1 com.oracle.bigdata.colmap
    - 17.4.2 com.oracle.bigdata.erroropt
    - 17.4.3 com.oracle.bigdata.fields
    - 17.4.4 com.oracle.bigdata.fileformat
    - 17.4.5 com.oracle.bigdata.log.exec
    - 17.4.6 com.oracle.bigdata.log.qc
    - 17.4.7 com.oracle.bigdata.overflow
    - 17.4.8 com.oracle.bigdata.rowformat
    - 17.4.9 com.oracle.bigdata.tablename

- 18 外部表の例
  - 18.1 ORACLE_LOADERアクセス・ドライバを使用したパーティション化された外部表の作成
  - 18.2 ORACLE_LOADERアクセス・ドライバを使用したパーティション化されたハイブリッド表の作成
  - 18.3 ORACLE_DATAPUMPアクセス・ドライバを使用したパーティション化された外部表の作成
  - 18.4 ORACLE_HDFSアクセス・ドライバを使用したパーティション化された外部表の作成
  - 18.5 ORACLE_HIVEアクセス・ドライバを使用したパーティション化された外部表の作成

- 18.6 外部表からのLOBのロード
- 18.7 外部表からのCSVファイルのロード
- 19 ORACLE_BIGDATAアクセス・ドライバ
  - 19.1 Using the ORACLE_BIGDATAアクセス・ドライバ
  - 19.2 オブジェクト・ストアの資格証明の作成方法
    - 19.2.1 DBMS_CREDENTIAL.CREATE_CREDENTIALを使用した資格証明オブジェクトの作成
    - 19.2.2 DBMS_CLOUD.CREATE_CREDENTIALを使用した資格証明オブジェクトの作成
    - 19.2.3 オブジェクト・ストレージのLocation句の定義方法
    - 19.2.4 ORACLE_BIGDATAアクセス・パラメータの理解
  - 19.3 オブジェクト・ストアのアクセス・パラメータ
    - 19.3.1 プロパティを指定するための構文ルール
    - 19.3.2 ORACLE_BIGDATAアクセス・パラメータ
    - 19.3.3 GATHER_EXTERNAL_TABLE_STATS
- 第IV部 その他のユーティリティ
  - 20 クラウド移行前アドバイザ・ツール
    - 20.1 クラウド移行前アドバイザ・ツールとは
    - 20.2 クラウド移行前アドバイザ・ツールを使用するための前提条件
    - 20.3 クラウド移行前アドバイザ・ツールのダウンロードと構成
    - 20.4 クラウド移行前アドバイザ・ツール(CPAT)の開始
    - 20.5 クラウド移行前アドバイザ・ツールの接続文字列
    - 20.6 クラウド移行前アドバイザ・ツールに必要なコマンドライン文字列
    - 20.7 FULLモードおよびSCHEMAモード
    - 20.8 クラウド移行前アドバイザ・ツール(CPAT)レポート・データの解釈
    - 20.9 コマンドライン構文およびプロパティ
      - 20.9.1 移行前アドバイザ・ツールのコマンドライン構文
      - 20.9.2 移行前アドバイザ・ツールのコマンドラインのプロパティ
        - 20.9.2.1 analysisprops
        - 20.9.2.2 connectstring
        - 20.9.2.3 excludeschemas
        - 20.9.2.4 full
        - 20.9.2.5 gettargetprops
        - 20.9.2.6 help
        - 20.9.2.7 logginglevel
        - 20.9.2.8 maxrelevantobjects
        - 20.9.2.9 maxtextdatarows
        - 20.9.2.10 migrationmethod
        - 20.9.2.11 outdir
        - 20.9.2.12 outfileprefix
        - 20.9.2.13 pdbname
        - 20.9.2.14 reportformat
        - 20.9.2.15 schemas

- 20.9.2.16 sqltext
- 20.9.2.17 sysdba
- 20.9.2.18 targetcloud
- 20.9.2.19 username
- 20.9.2.20 version
- 20.9.2.21 updatecheck

- 20.10 移行前アドバイザ・ツールのログ・ファイル構造
- 20.11 移行前アドバイザ・ツールによって実行されるチェックのリスト
  - 20.11.1 dp_has_low_streams_pool_size
  - 20.11.2 gg_enabled_replication
  - 20.11.3 gg_force_logging
  - 20.11.4 gg_has_low_streams_pool_size
  - 20.11.5 gg_not_unique_bad_col_no
  - 20.11.6 gg_not_unique_bad_col_yes
  - 20.11.7 gg_objects_not_supported
  - 20.11.8 gg_supplemental_log_data_min
  - 20.11.9 gg_tables_not_supported
  - 20.11.10 gg_tables_not_supported
  - 20.11.11 gg_user_objects_in_ggadmin_schemas
  - 20.11.12 has_absent_default_tablespace
  - 20.11.13 has_absent_temp_tablespace
  - 20.11.14 has_active_data_guard_dedicated
  - 20.11.15 has_active_data_guard_serverless
  - 20.11.16 has_basic_file_lobs
  - 20.11.17 has_clustered_tables
  - 20.11.18 has_columns_of_rowid_type
  - 20.11.19 has_columns_with_media_data_types_adb
  - 20.11.20 has_columns_with_media_data_types_default
  - 20.11.21 has_columns_with_spatial_data_types
  - 20.11.22 has_common_objects
  - 20.11.23 has_compression_disabled_for_objects
  - 20.11.24 has_csmig_schema
  - 20.11.25 has_data_in_other_tablespaces_dedicated
  - 20.11.26 has_data_in_other_tablespaces_serverless
  - 20.11.27 has_db_link_synonyms
  - 20.11.28 has_db_links
  - 20.11.29 has_dbms_credentials
  - 20.11.30 has_dbms_credentials
  - 20.11.31 has_directories
  - 20.11.32 has_enabled_scheduler_jobs
  - 20.11.33 has_external_tables_dedicated
  - 20.11.34 has_external_tables_default
  - 20.11.35 has_external_tables_serverless

- 20.11.36 has_fmw_registry_in_system
- 20.11.37 has_illegal_characters_in_comments
- 20.11.38 has_ilm_ado_policies
- 20.11.39 has_incompatible_jobs
- 20.11.40 has_index_organized_tables
- 20.11.41 has_java_objects
- 20.11.42 has_java_source
- 20.11.43 has_libraries
- 20.11.44 has_logging_off_for_partitions
- 20.11.45 has_logging_off_for_subpartitions
- 20.11.46 has_logging_off_for_tables
- 20.11.47 has_low_streams_pool_size
- 20.11.48 has_noexport_object_grants
- 20.11.49 has_parallel_indexes_enabled
- 20.11.50 has_profile_not_default
- 20.11.51 has_public_synonyms
- 20.11.52 has_refs_to_restricted_packages_dedicated
- 20.11.53 has_refs_to_restricted_packages_serverless
- 20.11.54 has_refs_to_user_objects_in_sys
- 20.11.55 has_role_privileges
- 20.11.56 has_sqlt_objects_adb
- 20.11.57 has_sqlt_objects_default
- 20.11.58 has_sys_privileges
- 20.11.59 has_tables_that_fail_with_dblink
- 20.11.60 has_tables_with_long_raw_datatype
- 20.11.61 has_tables_with_xmltype_column
- 20.11.62 has_trusted_server_entries
- 20.11.63 has_user_defined_objects_in_sys
- 20.11.64 has_users_with_10g_password_version
- 20.11.65 has_sys_privileges
- 20.11.66 has_tables_that_fail_with_dblink
- 20.11.67 has_tables_with_long_raw_datatype
- 20.11.68 has_tables_with_xmltype_column
- 20.11.69 has_trusted_server_entries
- 20.11.70 has_user_defined_objects_in_sys
- 20.11.71 has_users_with_10g_password_version
- 20.11.72 has_xmlschema_objects
- 20.11.73 has_xmltype_tables
- 20.11.74 modified_db_parameters_dedicated
- 20.11.75 modified_db_parameters_serverless
- 20.11.76 nls_character_set_conversion
- 20.11.77 nls_national_character_set
- 20.11.78 nls_nchar_ora_910

- 20.11.79 options_in_use_not_available_dedicated
- 20.11.80 options_in_use_not_available_serverless
- 20.11.81 standard_traditional_audit_adb
- 20.11.82 standard_traditional_audit_default
- 20.11.83 timezone_table_compatibility_higher_dedicated
- 20.11.84 timezone_table_compatibility_higher_default
- 20.11.85 timezone_table_compatibility_higher_serverless
- 20.11.86 unified_and_standard_traditional_audit_adb
- 20.11.87 unified_and_standard_traditional_audit_default
- 20.11.88 xdb_resource_view_has_entriesチェック
- 20.12 移行前アドバイザ・ツールを使用するためのベスト・プラクティス
  - 20.12.1 ターゲット・データベース・インスタンスでのプロパティ・ファイルの生成
  - 20.12.2 CPAT分析の焦点
  - 20.12.3 レポートのデータ量の削減
  - 20.12.4 JSONレポートの生成およびログの保存
  - 20.12.5 出力接頭辞を使用した様々な移行シナリオの記録
- 21 ADRCI: ADRコマンド・インタプリタ
  - 21.1 ADRコマンド・インタプリタ(ADRCI)ユーティリティ
  - 21.2 Oracle Database ADRCの定義
  - 21.3 ADRCIの起動とヘルプの利用
    - 21.3.1 対話方式モードでのADRCIの使用方法
    - 21.3.2 ヘルプの表示
    - 21.3.3 バッチ・モードでのADRCIの使用方法
  - 21.4 ADRCIコマンドを使用する前のADRCIホームパスの設定
  - 21.5 アラート・ログの表示
  - 21.6 トレース・ファイルの検索
  - 21.7 インシデントの表示
  - 21.8 インシデントのパッケージ化
    - 21.8.1 インシデントのパッケージ化
    - 21.8.2 インシデント・パッケージの作成
      - 21.8.2.1 論理インシデント・パッケージの作成
      - 21.8.2.2 論理インシデント・パッケージへの診断情報の追加
      - 21.8.2.3 物理インシデント・パッケージの生成
  - 21.9 ADRCIコマンド・リファレンス
    - 21.9.1 CREATE REPORT
    - 21.9.2 ECHO
    - 21.9.3 EXIT
    - 21.9.4 HOST
    - 21.9.5 IPS
      - 21.9.5.1 IPSコマンドでの<ADR_HOME>および<ADR_BASE>変数の使用
      - 21.9.5.2 IPS ADD
      - 21.9.5.3 IPS ADD FILE

    - 21.9.5.4 IPS ADD NEW INCIDENTS
    - 21.9.5.5 IPS COPY IN FILE
    - 21.9.5.6 IPS COPY OUT FILE
    - 21.9.5.7 IPS CREATE PACKAGE
    - 21.9.5.8 IPS DELETE PACKAGE
    - 21.9.5.9 IPS FINALIZE
    - 21.9.5.10 IPS GENERATE PACKAGE
    - 21.9.5.11 IPS GET MANIFEST
    - 21.9.5.12 IPS GET METADATA
    - 21.9.5.13 IPS PACK
    - 21.9.5.14 IPS REMOVE
    - 21.9.5.15 IPS REMOVE FILE
    - 21.9.5.16 IPS SET CONFIGURATION
    - 21.9.5.17 IPS SHOW CONFIGURATION
    - 21.9.5.18 IPS SHOW FILES
    - 21.9.5.19 IPS SHOW INCIDENTS
    - 21.9.5.20 IPS SHOW PACKAGE
    - 21.9.5.21 IPS UNPACK FILE
- 21.9.6 PURGE
- 21.9.7 QUIT
- 21.9.8 RUN
- 21.9.9 SELECT
    - 21.9.9.1 AVG
    - 21.9.9.2 CONCAT
    - 21.9.9.3 COUNT
    - 21.9.9.4 DECODE
    - 21.9.9.5 LENGTH
    - 21.9.9.6 MAX
    - 21.9.9.7 MIN
    - 21.9.9.8 NVL
    - 21.9.9.9 REGEXP_LIKE
    - 21.9.9.10 SUBSTR
    - 21.9.9.11 SUM
    - 21.9.9.12 TIMESTAMP_TO_CHAR
    - 21.9.9.13 TOLOWER
    - 21.9.9.14 TOUPPER
- 21.9.10 SET BASE
- 21.9.11 SET BROWSER
- 21.9.12 SET CONTROL
- 21.9.13 SET ECHO
- 21.9.14 SET EDITOR
- 21.9.15 SET HOMEPATH
- 21.9.16 SET TERMOUT

- 21.9.17 SHOW ALERT
- 21.9.18 SHOW BASE
- 21.9.19 SHOW CONTROL
- 21.9.20 SHOW HM_RUN
- 21.9.21 SHOW HOMEPATH
- 21.9.22 SHOW HOMES
- 21.9.23 SHOW INCDIR
- 21.9.24 SHOW INCIDENT
- 21.9.25 SHOW LOG
- 21.9.26 SHOW PROBLEM
- 21.9.27 SHOW REPORT
- 21.9.28 SHOW TRACEFILE
- 21.9.29 SPOOL
- 21.10 ADRCIのトラブルシューティング
- 22 DBVERIFY: オフライン・データベース検査ユーティリティ
  - 22.1 DBVERIFYを使用した単一データ・ファイルのディスク・ブロックの検査
    - 22.1.1 単一ファイルのブロックを検査する際のDBVERIFY構文
    - 22.1.2 単一ファイルのブロックを検査する際のDBVERIFYパラメータ
    - 22.1.3 単一データ・ファイルに対するDBVERIFYの出力例
  - 22.2 DBVERIFYを使用したセグメントの検査
    - 22.2.1 セグメントを検査する際のDBVERIFY構文
    - 22.2.2 単一のセグメントを検査する際のDBVERIFYパラメータ
    - 22.2.3 検査対象のセグメントに対するDBVERIFYの出力例
- 23 DBNEWIDユーティリティ
  - 23.1 DBNEWIDユーティリティとは
  - 23.2 DBIDおよびDBNAMEの変更による影響
  - 23.3 グローバル・データベース名に関する考慮点
  - 23.4 DBNEWIDを使用したCDBおよびPDB DBIDの両方の変更
  - 23.5 データベースのDBIDおよびDBNAMEの変更
    - 23.5.1 DBIDおよびデータベース名の変更
    - 23.5.2 データベースIDのみの変更
    - 23.5.3 データベース名のみの変更
    - 23.5.4 DBNEWIDのトラブルシューティング
  - 23.6 DBNEWIDの構文
    - 23.6.1 DBNEWIDパラメータ
    - 23.6.2 制限事項および使用上のノート
    - 23.6.3 Oracle Database 10gより前のリリースに対する制限事項
- 24 REDOログ・ファイル分析のためのLogMinerの使用
  - 24.1 LogMinerのメリット
  - 24.2 LogMinerの概要
    - 24.2.1 LogMiner構成
      - 24.2.1.1 LogMiner構成ファイルのオブジェクト
      - 24.2.1.2 LogMinerの構成例

- 24.2.1.3 LogMinerの要件
- 24.2.2 LogMiner操作の指定および分析するデータの取得
- 24.3 CDBのLogMinerの使用
  - 24.3.1 CDBでのLogMiner V$ビューおよびDBAビュー
  - 24.3.2 CDBでのV$LOGMNR_CONTENTSビュー
  - 24.3.3 CDBでのサプリメンタル・ロギングの有効化
  - 24.3.4 CDBでのフラット・ファイル・ディクショナリの使用
- 24.4 LogMinerディクショナリ・ファイルとREDOログ・ファイル
  - 24.4.1 LogMinerディクショナリ・オプション
    - 24.4.1.1 オンライン・カタログの使用
    - 24.4.1.2 REDOログ・ファイルへのLogMinerディクショナリの抽出
    - 24.4.1.3 フラット・ファイルへのLogMinerディクショナリの抽出
  - 24.4.2 REDOログ・ファイル・オプション
- 24.5 LogMinerの起動
- 24.6 分析するREDOデータについてのV$LOGMNR_CONTENTSの問合せ
  - 24.6.1 V$LOGMNR_CONTENTSビューへの移入方法
  - 24.6.2 列の値に基づいたV$LOGMNR_CONTENTSの問合せ
    - 24.6.2.1 MINE_VALUE関数によって返されるNULL値の意味
    - 24.6.2.2 MINE_VALUE関数およびCOLUMN_PRESENT関数の使用規則
    - 24.6.2.3 MINE_VALUEファンクションを使用してNCHAR値を取得する際の制限事項
  - 24.6.3 XMLType列およびXMLType表に基づいたV$LOGMNR_CONTENTSの問合せ
    - 24.6.3.1 XMLType列およびXMLType表に基づいたV$LOGMNR_CONTENTSの問合せ方法
    - 24.6.3.2 XMLTypeデータを使用するLogMinerの使用上の制限
    - 24.6.3.3 XMLTypeデータを作成するためのPL/SQLプロシージャの例
- 24.7 V$LOGMNR_CONTENTSに返されるデータのフィルタ処理および書式設定
  - 24.7.1 コミット済トランザクションのみの表示
  - 24.7.2 REDO破損のスキップ
  - 24.7.3 時刻によるデータのフィルタ処理
  - 24.7.4 SCNによるデータのフィルタ処理
  - 24.7.5 再実行のために再構築されたSQL文の書式設定
  - 24.7.6 返されるデータの可読性向上のための表示方法の書式設定
- 24.8 V$LOGMNR_CONTENTSに返されたDDL文の再適用
- 24.9 DBMS_LOGMNR.START_LOGMNRの複数回のコール
- 24.10 サプリメンタル・ロギング
  - 24.10.1 データベース・レベルのサプリメンタル・ロギング
    - 24.10.1.1 最小サプリメンタル・ロギング
    - 24.10.1.2 データベース・レベルの識別キー・ロギング
    - 24.10.1.3 プロシージャ型サプリメンタル・ロギング
  - 24.10.2 データベース・レベルのサプリメンタル・ロギングの無効化
  - 24.10.3 表レベルのサプリメンタル・ロギング

- 24.10.3.1 表レベルの識別キー・ロギング
- 24.10.3.2 表レベルのユーザー定義サプリメンタル・ログ・グループ
- 24.10.3.3 ユーザー定義のサプリメンタル・ログ・グループを使用する場合のノート
- 24.10.4 LogMinerディクショナリでのDDL文の追跡
- 24.10.5 DDL_DICT_TRACKINGおよびサプリメンタル・ロギングの設定
- 24.10.6 DDL_DICT_TRACKINGおよび指定された時間範囲またはSCN範囲
- 24.11 ビューでのLogMiner操作情報へのアクセス
  - 24.11.1 LogMiner操作情報を表示するためのオプション
  - 24.11.2 V$LOGMNR_LOGSの問合せ
  - 24.11.3 サプリメンタル・ロギング設定に関するビューの問合せ
  - 24.11.4 LogMinerを使用した個々のPDBの問合せ
- 24.12 一般的なLogMinerセッションのステップ
  - 24.12.1 LogMinerセッションの実行方法の理解
  - 24.12.2 一般的なLogMinerセッション・タスク1: サプリメンタル・ロギングの有効化
  - 24.12.3 一般的なLogMinerセッション・タスク2: LogMinerディクショナリの抽出
  - 24.12.4 一般的なLogMinerセッション・タスク3: 分析用のREDOログ・ファイルの指定
  - 24.12.5 LogMinerの起動
  - 24.12.6 V$LOGMNR_CONTENTSの問合せ
  - 24.12.7 一般的なLogMinerセッション・タスク6: LogMinerセッションの終了
- 24.13 LogMinerの使用例
  - 24.13.1 分析するREDOログ・ファイルの明示的指定によるマイニングの例
    - 24.13.1.1 例1: 最後にアーカイブされたREDOログ・ファイルでのすべての変更の検索
    - 24.13.1.2 例2: コミット済トランザクションへのDML文のグループ化
    - 24.13.1.3 例3: 再構築されたSQLの書式設定
    - 24.13.1.4 例4: REDOログ・ファイル内のLogMinerディクショナリの使用
    - 24.13.1.5 例5: 内部ディクショナリでのDDL文の追跡
    - 24.13.1.6 例6: 時間範囲による出力のフィルタ処理
  - 24.13.2 LogMinerのユース・ケースのシナリオ
    - 24.13.2.1 LogMinerを使用した特定のユーザーによる変更の追跡
    - 24.13.2.2 LogMinerを使用した表アクセス統計の計算
- 24.14 サポートされるデータ型、記憶域属性、およびデータベースとREDOログ・ファイルのバージョン
  - 24.14.1 サポートされるデータ型と表記憶域属性
    - 24.14.1.1 互換性要件
  - 24.14.2 サポートされないデータ型と表記憶域属性
  - 24.14.3 サポートされるデータベースとREDOログ・ファイルのバージョン
  - 24.14.4 SecureFiles LOBに関する考慮点
- 25 メタデータAPIの使用
  - 25.1 DBMS_METADATA APIを使用する理由
  - 25.2 DBMS_METADATA APIの概要
  - 25.3 オブジェクトのメタデータを取得するためのDBMS_METADATA APIの使用
    - 25.3.1 基本的なメタデータ取得の通常ステップ
    - 25.3.2 複数のオブジェクトの取得

- 25.3.3 変換の条件指定
- 25.3.4 特定のメタデータ属性へのアクセス
- 25.4 取得したオブジェクトを再作成するためのDBMS_METADATA APIの使用
- 25.5 異なるオブジェクト型のコレクションを取得するためのDBMS_METADATA APIの使用
  - 25.5.1 異種オブジェクト型の戻り値のフィルタ
- 25.6 オブジェクトのメタデータを比較するためのDBMS_METADATA_DIFF APIの使用
- 25.7 DBMS_METADATA APIのプログラム・インタフェースに関するパフォーマンスのヒント
- 25.8 DBMS_METADATA APIの使用例
  - 25.8.1 DBMS_METADATAの例で行われる処理
  - 25.8.2 GET_PAYROLL_TABLESプロシージャで生成される出力
- 25.9 DBMS_METADATAプロシージャの要約
- 25.10 DBMS_METADATA_DIFFプロシージャの要約
- 26 オリジナルのエクスポート
  - 26.1 エクスポート・ユーティリティとは
  - 26.2 エクスポート・ユーティリティを使用する前に
    - 26.2.1 エクスポート・ユーティリティを使用するための準備チェックリスト
    - 26.2.2 catexp.sqlまたはcatalog.sqlの実行
    - 26.2.3 エクスポート操作に必要な十分なディスク領域の確保
    - 26.2.4 エクスポート操作およびインポート操作のアクセス権の確認
  - 26.3 エクスポート・ユーティリティの起動
    - 26.3.1 SYSDBAでのエクスポート・ユーティリティの起動
    - 26.3.2 コマンドライン・エントリ
    - 26.3.3 パラメータ・ファイル
    - 26.3.4 対話方式モード
      - 26.3.4.1 対話方式でのエクスポートを使用する際の制限事項
    - 26.3.5 オンライン・ヘルプの利用
  - 26.4 エクスポート・モード
    - 26.4.1 表レベル・エクスポートおよびパーティション・レベル・エクスポート
      - 26.4.1.1 表レベル・エクスポート
      - 26.4.1.2 パーティション・レベル・エクスポート
  - 26.5 エクスポート・パラメータ
    - 26.5.1 BUFFER
      - 26.5.1.1 例: バッファ・サイズの計算
    - 26.5.2 COMPRESS
    - 26.5.3 CONSISTENT
    - 26.5.4 CONSTRAINTS
    - 26.5.5 DIRECT
    - 26.5.6 FEEDBACK
    - 26.5.7 FILE
    - 26.5.8 FILESIZE
    - 26.5.9 FLASHBACK_SCN
    - 26.5.10 FLASHBACK_TIME
    - 26.5.11 FULL

- 26.5.11.1 全データベースのエクスポートおよびインポートについての考慮点
- 26.5.12 GRANTS
- 26.5.13 HELP
- 26.5.14 INDEXES
- 26.5.15 LOG
- 26.5.16 OBJECT_CONSISTENT
- 26.5.17 OWNER
- 26.5.18 PARFILE
- 26.5.19 QUERY
  - 26.5.19.1 QUERYパラメータを使用する際の制限事項
- 26.5.20 RECORDLENGTH
- 26.5.21 RESUMABLE
- 26.5.22 RESUMABLE_NAME
- 26.5.23 RESUMABLE_TIMEOUT
- 26.5.24 ROWS
- 26.5.25 STATISTICS
- 26.5.26 TABLES
  - 26.5.26.1 表名の制限
- 26.5.27 TABLESPACES
- 26.5.28 TRANSPORT_TABLESPACE
- 26.5.29 TRIGGERS
- 26.5.30 TTS_FULL_CHECK
- 26.5.31 USERID (username/password)
- 26.5.32 VOLSIZE

- 26.6 エクスポート・セッションの例
  - 26.6.1 全体データベース・モードのエクスポート・セッションの例
  - 26.6.2 ユーザー・モードのエクスポート・セッションの例
  - 26.6.3 表モードのエクスポート・セッションの例
    - 26.6.3.1 例1: DBAによる2人のユーザー用の表のエクスポート
    - 26.6.3.2 例2: ユーザーによる自身が所有している表のエクスポート
    - 26.6.3.3 例3: パターン一致を使用した様々な表のエクスポート
  - 26.6.4 パーティション・レベル・エクスポートでのエクスポート・セッションの例
    - 26.6.4.1 例1: パーティションを指定しない表のエクスポート
    - 26.6.4.2 例2: パーティションを指定した表のエクスポート
    - 26.6.4.3 例3: コンポジット・パーティションのエクスポート
- 26.7 警告、エラーおよび完了メッセージ
  - 26.7.1 ログ・ファイル
  - 26.7.2 警告メッセージ
  - 26.7.3 リカバリ不能なエラー・メッセージ
  - 26.7.4 完了メッセージ
- 26.8 終了コードによる結果の検査と表示
- 26.9 従来型パス・エクスポートとダイレクト・パス・エクスポート
- 26.10 ダイレクト・パス・エクスポートの開始

- 26.10.1 ダイレクト・パス・エクスポートに関するセキュリティ上の考慮点
- 26.10.2 ダイレクト・パス・エクスポートに関するパフォーマンス上の考慮点
- 26.10.3 ダイレクト・パス・エクスポートに関する制限事項
- 26.11 オリジナルのOracle Data Pumpエクスポートのネットワーク考慮事項
  - 26.11.1 ネットワークを介してエクスポート・ファイルを転送する方法
  - 26.11.2 Oracle Netを使用したエクスポート
- 26.12 文字セットおよびグローバリゼーション・サポートに関する考慮点
  - 26.12.1 ユーザー・データ
    - 26.12.1.1 変換による文字セットのソート順への影響
  - 26.12.2 DDL
  - 26.12.3 シングルバイト文字セットとエクスポートおよびインポート
  - 26.12.4 マルチバイト文字セットとエクスポートおよびインポート
- 26.13 エクスポート・ユーティリティおよびインポート・ユーティリティでのインスタンス親和性の使用
- 26.14 データベース・オブジェクトのエクスポート時の考慮点
  - 26.14.1 エクスポート順序
  - 26.14.2 LONGおよびLOBデータ型のエクスポート
  - 26.14.3 外部関数ライブラリのエクスポート
  - 26.14.4 オフライン・ローカル管理表領域のエクスポート
  - 26.14.5 ディレクトリ別名のエクスポート
  - 26.14.6 BFILE列および属性のエクスポート
  - 26.14.7 外部表のエクスポート
  - 26.14.8 オブジェクトの型定義のエクスポート
  - 26.14.9 ネストした表のエクスポート
  - 26.14.10 AQ表のエクスポート
  - 26.14.11 シノニムのエクスポート
  - 26.14.12 Javaシノニムに関連して発生する可能性があるエクスポート・エラー
  - 26.14.13 ファイングレイン・アクセス・コントロールに対するサポート
- 26.15 トランスポータブル表領域
- 26.16 読取り専用データベースからのエクスポート
- 26.17 エクスポート・ユーティリティおよびインポート・ユーティリティを使用したデータベース移行のパーティション化
  - 26.17.1 移行をパーティション化する場合のメリット
  - 26.17.2 移行をパーティション化する場合のデメリット
  - 26.17.3 エクスポート・ユーティリティおよびインポート・ユーティリティを使用したデータベース移行のパーティション化方法
- 26.18 リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用
  - 26.18.1 リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用時の制限事項
  - 26.18.2 リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用例
- 27 オリジナルのインポート
  - 27.1 インポート・ユーティリティとは
    - 27.1.1 表オブジェクト: インポートの順序

- 27.2 インポート・ユーティリティを使用する前に
  - 27.2.1 catexp.sqlまたはcatalog.sqlの実行
  - 27.2.2 インポート操作のアクセス権の確認
    - 27.2.2.1 オブジェクトをスキーマにインポートする方法
    - 27.2.2.2 権限のインポート
    - 27.2.2.3 他のスキーマへのオブジェクトのインポート
    - 27.2.2.4 システム・オブジェクトのインポート
  - 27.2.3 処理上の制限事項
- 27.3 既存の表へのインポート
  - 27.3.1 データのインポート前に手動で表を作成する方法
  - 27.3.2 参照制約を使用禁止にする方法
  - 27.3.3 手動によるインポートの順序付け
- 27.4 インポート操作上のスキーマおよびデータベース・トリガーの影響
- 27.5 インポートの起動
  - 27.5.1 コマンドライン・エントリ
  - 27.5.2 パラメータ・ファイル
  - 27.5.3 対話方式モード
  - 27.5.4 SYSDBAでのインポート・ユーティリティの起動
  - 27.5.5 オンライン・ヘルプの利用
- 27.6 インポート・モード
- 27.7 インポート・パラメータ
  - 27.7.1 BUFFER
  - 27.7.2 COMMIT
  - 27.7.3 COMPILE
  - 27.7.4 CONSTRAINTS
  - 27.7.5 DATA_ONLY
  - 27.7.6 DATAFILES
  - 27.7.7 DESTROY
  - 27.7.8 FEEDBACK
  - 27.7.9 FILE
  - 27.7.10 FILESIZE
  - 27.7.11 FROMUSER
  - 27.7.12 FULL
    - 27.7.12.1 全データベースのエクスポートおよびインポートについての考慮点
  - 27.7.13 GRANTS
  - 27.7.14 HELP
  - 27.7.15 IGNORE
  - 27.7.16 INDEXES
  - 27.7.17 INDEXFILE
  - 27.7.18 LOG
  - 27.7.19 PARFILE
  - 27.7.20 RECORDLENGTH
  - 27.7.21 RESUMABLE

- 27.7.22 RESUMABLE_NAME
- 27.7.23 RESUMABLE_TIMEOUT
- 27.7.24 ROWS
- 27.7.25 SHOW
- 27.7.26 SKIP_UNUSABLE_INDEXES
- 27.7.27 STATISTICS
- 27.7.28 STREAMS_CONFIGURATION
- 27.7.29 STREAMS_INSTANTIATION
- 27.7.30 TABLES
  - 27.7.30.1 表名の制限
- 27.7.31 TABLESPACES
- 27.7.32 TOID_NOVALIDATE
- 27.7.33 TOUSER
- 27.7.34 TRANSPORT_TABLESPACE
- 27.7.35 TTS_OWNERS
- 27.7.36 USERID (username/password)
- 27.7.37 VOLSIZE

- 27.8 インポート・セッションの例
  - 27.8.1 特定のユーザーの表を選択してインポートする例
  - 27.8.2 別のユーザーによってエクスポートされた表をインポートする例
  - 27.8.3 あるユーザーの表を別のユーザーへインポートする例
  - 27.8.4 パーティション・レベル・インポートでのインポート・セッションの例
    - 27.8.4.1 例1: パーティション・レベル・インポート
    - 27.8.4.2 例2: コンポジット・パーティション表のパーティション・レベル・インポート
    - 27.8.4.3 例3: 別の列での表の再パーティション化
  - 27.8.5 パターン一致を使用して様々な表をインポートする例
- 27.9 終了コードによる結果の検査と表示
- 27.10 インポート中のエラー処理
  - 27.10.1 行エラー
    - 27.10.1.1 整合性制約違反
    - 27.10.1.2 無効なデータ
  - 27.10.2 データベース・オブジェクトのインポートでのエラー
    - 27.10.2.1 オブジェクトはすでに存在します
    - 27.10.2.2 順序
    - 27.10.2.3 リソース・エラー
    - 27.10.2.4 ドメイン索引のメタデータ
- 27.11 表レベル・インポートおよびパーティション・レベル・インポート
  - 27.11.1 表レベル・インポートの使用に関するガイドライン
  - 27.11.2 パーティション・レベル・インポートの使用に関するガイドライン
  - 27.11.3 パーティションと表の間のデータ移行
- 27.12 索引作成およびメンテナンスの制御
  - 27.12.1 索引作成の延期
  - 27.12.2 索引作成およびメンテナンスの制御

- 27.12.2.1 索引更新延期の例
- 27.13 オリジナルのインポート・ユーティリティでOracle Netを使用する際のネットワーク考慮事項
- 27.14 文字セットおよびグローバリゼーション・サポートに関する考慮点
  - 27.14.1 ユーザー・データ
    - 27.14.1.1 変換による文字セットのソート順への影響
  - 27.14.2 DDL
  - 27.14.3 シングルバイト文字セット
  - 27.14.4 マルチバイト文字セット
- 27.15 インスタンス親和性の使用方法
- 27.16 データベース・オブジェクトのインポートに関する考慮点
  - 27.16.1 オブジェクト識別子のインポート
  - 27.16.2 既存のオブジェクト表およびオブジェクト型の含まれている表のインポート
  - 27.16.3 ネストした表のインポート
  - 27.16.4 REFデータのインポート
  - 27.16.5 BFILE列およびディレクトリ別名のインポート
  - 27.16.6 外部関数ライブラリのインポート
  - 27.16.7 ストアド・プロシージャ、ファンクションおよびパッケージのインポート
  - 27.16.8 Javaオブジェクトのインポート
  - 27.16.9 外部表のインポート
  - 27.16.10 AQ表のインポート
  - 27.16.11 LONG列のインポート
  - 27.16.12 トリガーが存在する場合のLOB列のインポート
  - 27.16.13 ビューのインポート
  - 27.16.14 パーティション表のインポート
- 27.17 ファイングレイン・アクセス・コントロールに対するサポート
- 27.18 スナップショットおよびスナップショット・ログ
  - 27.18.1 スナップショット・ログ
  - 27.18.2 スナップショット
    - 27.18.2.1 スナップショットのインポート
    - 27.18.2.2 異なるスキーマへのスナップショットのインポート
- 27.19 トランスポータブル表領域
- 27.20 記憶域パラメータ
  - 27.20.1 OPTIMALパラメータ
  - 27.20.2 OID索引とLOB列の記憶域パラメータ
  - 27.20.3 記憶域パラメータの上書き
- 27.21 読取り専用表領域
- 27.22 表領域の削除
- 27.23 表領域を再編成する方法
- 27.24 統計情報のインポート
- 27.25 エクスポート・ユーティリティおよびインポート・ユーティリティを使用したデータベース移行のパーティション化
  - 27.25.1 移行をパーティション化する場合のメリット
  - 27.25.2 移行をパーティション化する場合のデメリット

- 27.25.3 エクスポート・ユーティリティおよびインポート・ユーティリティを使用したデータベース移行のパーティション化方法
- 27.26 インポート操作のチューニングに関する考慮点
  - 27.26.1 システム・レベル・オプションの変更
  - 27.26.2 初期化パラメータの変更
  - 27.26.3 インポート・オプションの変更
  - 27.26.4 大量のLOBデータの処理
  - 27.26.5 大量のLONGデータの処理
- 27.27 リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用
  - 27.27.1 リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用時の制限事項
  - 27.27.2 リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用例

- 第V部 付録
- A SQL*Loaderの構文図
- B SQL*Loader、エクスポートおよびインポートのInstant Client
  - B.1 ツールInstant Clientとは
  - B.2 インストールするInstant Clientの選択
  - B.3 OTNからのダウンロードによるツールInstant Clientのインストール
  - B.4 12c Clientリリース・メディアからのツールInstant Clientのインストール
  - B.5 ツールInstant Clientパッケージの構成
  - B.6 ツールInstant Clientパッケージによるデータベースへの接続
  - B.7 Instant Clientのアンインストール
- 索引

例一覧

- 1-1 PDBへの表のインポート
- 1-2 データをインポートする際の資格証明の指定
- 1-3 デフォルトの資格証明を使用したデータのインポート
- 1-4 無効なローカル・ユーザー・エラーの回避
- 2-1 例
- 2-2 表モード・エクスポートの実行
- 2-3 選択した表および行のデータのみのアンロード
- 2-4 表モード・エクスポートに必要なディスク領域の見積り
- 2-5 スキーマ・モード・エクスポートの実行
- 2-6 パラレル全体エクスポート
- 2-7 ジョブの停止と再接続
- 3-1 データのみ表モード・インポートの実行
- 3-2 スキーマ・モード・インポートの実行
- 3-3 スキーマのネットワーク・モード・インポート
- 3-4 URLベースのファイル名に使用したワイルドカード
- 6-1 簡単なスキーマ・エクスポートの実行
- 6-2 ダンプ・ファイルのインポートおよびすべてのスキーマ・オブジェクトの再マップ
- 6-3 簡単なスキーマ・エクスポート実行中の例外処理機能の使用方法
- 6-4 ダンプ・ファイルの情報の表示
- 7-1 固定レコード形式でのデータのロード
- 7-2 可変レコード形式でのデータのロード
- 7-3 ストリーム・レコード形式でのデータのロード
- 9-1 制御ファイル
- 9-2 SQL*Loader制御ファイルでのXMLType表の識別
- 9-3 PRESERVEパラメータを使用しないCONTINUEIF THIS
- 9-4 PRESERVEパラメータを使用するCONTINUEIF THIS
- 9-5 PRESERVEパラメータを使用しないCONTINUEIF NEXT
- 9-6 PRESERVEパラメータを使用するCONTINUEIF NEXT
- 10-1 サンプル制御ファイルのフィールド・リスト・セクション
- 10-2 FILLERフィールドの指定
- 10-3 DEFAULTIF句の無評価
- 10-4 DEFAULTIF句の評価
- 10-5 DEFAULTIF句を使用した位置の指定
- 10-6 DEFAULTIF句を使用したフィールド名の指定
- 11-1 ストリーム・レコード形式への列オブジェクトのロード
- 11-2 可変レコード形式への列オブジェクトのロード
- 11-3 ネストした列オブジェクトのロード
- 11-4 サブタイプを使用した列オブジェクトのロード
- 11-5 NULLIF句を使用したNULL属性の指定
- 11-6 FILLERフィールドを使用したデータのロード
- 11-7 一致するコンストラクタを使用した列オブジェクトのロード

- 11-8　一致しないコンストラクタを使用した列オブジェクトのロード
- 11-9　コンストラクタが一致しない場合のSQLを使用した列オブジェクトのロード
- 11-10　主キーOIDを使用したオブジェクト表のロード
- 11-11　OIDのロード
- 11-12　サブタイプを使用したオブジェクト表のロード
- 11-13　SQL*Loader制御ファイルでのREF句の記述
- 11-14　システム生成REF列のロード
- 11-15　主キーREF列のロード
- 11-16　複数のINTO TABLE句を使用した単一ロードの方法
- 11-17　事前に決められたサイズのフィールドのLOBデータ
- 11-18　デリミタ付きフィールドのLOBデータのロード
- 11-19　Length-Value Pairフィールドへの LOBデータのロード
- 11-20　LOBFILE当たり1つのLOBを使用したLOBデータのロード
- 11-21　事前に決められたサイズのLOBを使用したLOBデータのロード
- 11-22　デリミタ付きLOBを使用したLOBデータのロード
- 11-23　Length-Value Pairを指定したLOBを使用したLOBデータのロード
- 11-24　BFILEを使用したデータのロード: ファイル名のみを動的に指定
- 11-25　BFILEを使用したデータのロード: ファイル名およびディレクトリを動的に指定
- 11-26　VARRAYおよびネストした表のロード
- 11-27　ユーザー定義SIDを使用した親表のロード
- 11-28　ユーザー定義SIDを使用した子表のロード
- 12-1　SQL*Loaderの制御ファイルに対する日付書式の設定
- 12-2　NLS_DATE_FORMAT環境変数の設定
- 14-1　ORACLE_LOADERアクセス・ドライバの属性の指定
- 14-2　ORACLE_DATAPUMPアクセス・ドライバの属性の指定
- 14-3　ORACLE_HDFSアクセス・ドライバの属性の指定
- 14-4　ORACLE_HIVEアクセス・ドライバの属性の指定
- 17-1　ORACLE_HDFSの複数のアクセス・パラメータの設定
- 17-2　ORACLE_HIVEの複数のアクセス・パラメータの設定
- 18-1　ORACLE_LOADERを使用したパーティション化された外部表の作成
- 18-2　例
- 18-3　ORACLE_DATAPUMPアクセス・ドライバを使用したパーティション化された外部表の作成
- 18-4　ORACLE_HDFSアクセス・ドライバを使用したパーティション化された外部表の作成
- 18-5　ORACLE_HIVEアクセス・ドライバを使用したパーティション化された外部表の作成
- 18-6　外部表からのLOBのロード
- 18-7　アクセス・パラメータを使用しないCSVファイルからのデータのロード
- 18-8　データ・ファイルの日付フィールドの書式と一致しないセッションのデフォルトの日付マスク
- 18-9　データが2つのデータ・ファイルに分割されている場合
- 18-10　データが2つのファイルに分割され、最初のファイルにのみフィールド名の行がある場合
- 18-11　ファイルのフィールドの順序が表の列の順序と一致する場合
- 18-12　データ・ファイルの一部のフィールドでアクセス・パラメータのデフォルト設定が使用されない場合
- 19-1　クラウド・サービス資格証明
- 19-2　ネイティブのOracle Cloud Infrastructure資格証明

- 19-3　ネイティブのOracle Cloud Infrastructure資格証明
- 19-4　ネイティブのOracle Cloud Infrastructureオブジェクト・ストレージ
- 19-5　Oracle Cloud Infrastructureオブジェクト・ストレージ
- 19-6　ホストスタイルのURI形式
- 19-7　パススタイルのURI形式
- 19-8　Azure BLOB Storageの場所の形式
- 19-9　CSVデータ・ファイル
- 19-10　CSVデータ・ファイル
- 19-11　JSONデータ・ファイル
- 20-1　CPATプロパティ・ファイルの生成
- 23-1　データベース名とデータベースIDの変更に関するアラート・ファイル
- 24-1　SYS.DBA_LOGMNR_DICTIONARYの問合せ
- 24-2　最小サプリメンタル・ロギングの有効化
- 25-1　データ取得のためのDBMS_METADATAプログラム・インタフェースの使用
- 25-2　データ取得のためのDBMS_METADATAブラウザ・インタフェースの使用
- 25-3　複数のオブジェクトの取得
- 25-4　変換の条件指定
- 25-5　XML文書の変更
- 25-6　特定のメタデータ属性にアクセスするための解析項目の使用
- 25-7　取得したオブジェクトを再作成するための送信インタフェースの使用
- 25-8　異種オブジェクト型の取得方法
- 25-9　異種オブジェクト型の返りのフィルタ
- 25-10　オブジェクト・メタデータの比較

図一覧

- 7-1　SQL*Loaderの概要
- 10-1　フィールド変換の例
- 10-2　固定長フィールドの後の相対的な位置指定
- 10-3　デリミタ付きフィールドの後の相対的な位置指定
- 10-4　囲みデリミタの後の相対的な位置指定
- 10-5　空白で区切られたフィールド
- 10-6　オプションの囲みデリミタ付きフィールド
- 24-1　LogMinerデータベースの構成例
- 24-2　LogMinerディクショナリの選択方法

表一覧

- 1-1 Oracle Data Pumpの終了コード
- 2-1 データ・ポンプ・エクスポートの対話方式コマンド・モードでサポートされているコマンド
- 3-1 Oracle Data Pump Importの対話方式コマンド・モードでサポートされているコマンド
- 4-1 データ・ポンプ・エクスポートでのオリジナルのエクスポート・パラメータの扱い
- 4-2 データ・ポンプ・インポートでのオリジナルのインポート・パラメータの扱い
- 6-1 DBMS_DATAPUMPプロシージャを実行できる有効なジョブの状態
- 7-1 事例用ファイルおよび関連ファイル
- 8-1 SQL*Loaderの終了コード
- 9-1 INFILEキーワードのパラメータ
- 9-2 CONTINUEIF句のパラメータ
- 9-3 固定長フィールド
- 9-4 非グラフィック・フィールド
- 9-5 グラフィック・フィールド
- 9-6 可変長フィールド
- 10-1 位置指定句のパラメータ
- 10-2 日時データ型および期間データ型のデータ型変換
- 10-3 デリミタの指定に使用するパラメータ
- 10-4 フィールド条件句のパラメータ
- 10-5 空白の切捨てに関する動作のサマリー
- 10-6 列指定に使用されるパラメータ
- 17-1 プロパティの特殊文字
- 17-2 com.oracle.bigdata.log.execの変数
- 17-3 com.oracle.bigdata.log.qcの変数
- 19-1 プロパティの特殊文字
- 19-2 共通のアクセス・パラメータ
- 19-3 Avro固有のアクセス・パラメータ
- 19-4 Parquet固有のアクセス・パラメータ
- 19-5 テキスト・ファイルおよびCSVに固有のアクセス・パラメータ
- 20-1 JDBC接続文字列の例
- 20-2 移行前アドバイザ・ツール(CPAT)のチェック結果の定義
- 21-1 ADRCIバッチ操作パラメータ
- 21-2 IPS ADDコマンドの引数
- 21-3 IPS CREATE PACKAGEコマンドの引数
- 21-4 IPS PACKコマンドの引数
- 21-5 IPS REMOVEコマンドの引数
- 21-6 IPS構成パラメータ
- 21-7 PURGEコマンドのフラグ
- 21-8 SELECTコマンドのフラグ
- 21-9 SHOW ALERTコマンドのフラグ
- 21-10 SHOW ALERTのアラート・フィールド
- 21-11 状態モニターの実行に使用するフィールド

- 21-12　SHOW INCIDENTコマンドのフラグ
- 21-13　SHOW INCIDENTのインシデント・フィールド
- 21-14　SHOW LOGコマンドのフラグ
- 21-15　SHOW LOGのログ・フィールド
- 21-16　SHOW PROBLEMコマンドのフラグ
- 21-17　SHOW PROBLEMの問題フィールド
- 21-18　SHOW TRACEFILEコマンドの引数
- 21-19　SHOW TRACEFILEコマンドのフラグ
- 23-1　DBNEWIDユーティリティのパラメータ
- 25-1　複数オブジェクトの取得に使用するDBMS_METADATAプロシージャ
- 25-2　ブラウザ・インタフェースで使用するDBMS_METADATAプロシージャ
- 25-3　XMLの送信で使用するDBMS_METADATAのプロシージャおよびファンクション
- 25-4　DBMS_METADATA_DIFFのプロシージャおよびファンクション
- 26-1　各モードでエクスポートされるオブジェクト
- 26-2　2人のユーザーによる更新時のイベント順序
- 26-3　ダンプ・ファイルの最大サイズ
- 26-4　エクスポート・ユーティリティの終了コード
- 26-5　リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用
- 27-1　自分のスキーマにオブジェクトをインポートするために必要な権限
- 27-2　権限のインポートに必要な権限
- 27-3　各モードでインポートされるオブジェクト
- 27-4　インポート・ユーティリティの終了コード
- 27-5　リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用
- B-1　ツール・パッケージ内のInstant Clientファイル

はじめに

このマニュアルでは、Oracle Databaseユーティリティを使用して、データ転送、データ・メンテナンスおよびデータベース管理を行う方法について説明します。

- 対象読者
- ドキュメントのアクセシビリティ
- ダイバーシティ&インクルージョン
- 関連ドキュメント
- 構文図
- 表記規則

対象読者

このマニュアルのユーティリティは、データベース管理者(DBA)、アプリケーション・プログラマ、セキュリティ管理者、システム・オペレータを対象としています。また、次の作業を行うOracle Databaseユーザーも対象としています。

- エクスポート・ユーティリティおよびインポート・ユーティリティ(オリジナルおよびOracle Data Pump版の両方)を使用した、データのアーカイブ、Oracle Databaseのバックアップおよび異なるOracle Database間のデータ移動
- SQL*Loaderを使用したオペレーティング・システム・ファイルからOracle Database表へのデータのロード
- 外部表機能を使用した外部ソースからのデータのロード
- DBVERIFYユーティリティを使用した、オフライン・データベース上での物理データ構造に対する整合性チェックの実行
- DBNEWIDユーティリティを使用した、オペレーショナル・データベースの内部データベース識別子(DBID)およびデータベース名(DBNAME)の維持
- メタデータAPIを使用した、Oracle Databaseオブジェクトに対するメタデータの完全な表現の抽出および処理
- LogMinerユーティリティを使用した、(SQLインタフェースからの)REDOログ・ファイルの問合せおよび分析
- 自動診断リポジトリ・コマンド・インタプリタ(ADRCI)ユーティリティを使用した、Oracle Database診断データの管理

このマニュアルを使用するには、SQLおよびOracleの基礎知識が必要です。これらの情報は、『Oracle Database概要』で参照できます。また、SQL*Loaderを使用するには、オペレーティング・システムのファイル管理機能の使用方法を理解しておく必要があります。

親トピック: はじめに

ドキュメントのアクセシビリティ

オラクルのアクセシビリティについての詳細情報は、Oracle Accessibility ProgramのWebサイト(http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc)を参照してください。

Oracle Supportへのアクセス

サポートをご契約のお客様には、My Oracle Supportを通して電子支援サービスを提供しています。詳細情報は(http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info)か、聴覚に障害のあるお客様は(http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs)を参照してください。

親トピック: はじめに

ダイバーシティ&インクルージョン

Oracleはダイバーシティ&インクルージョンに積極的に取り組んでいます。Oracleは、ソート・リーダーシップと革新性を高める社員の多様性を尊重し、その価値を重んじています。従業員、お客様、パートナー様にポジティブな影響をもたらすインクルーシブな文化を醸成する私たちのイニシアティブの一環として、製品やドキュメントからインセンシティブな用語を取り除くように努めています。また、Oracle製品および業界標準が進化する中、お客様の既存の技術との互換性を維持する必要性およびサービスの継続性確保の要求にも留意しています。このような技術的な制限により、当社のインセンシティブな用語を削除する取組みは継続中であり、時間と皆様のご協力が必要となります。

親トピック: はじめに

関連ドキュメント

詳細は、Oracle Databaseのドキュメント・セットを参照してください。特に、次のドキュメントを確認してください。

- Oracle Database概要
- Oracle Database SQL言語リファレンス
- Oracle Database管理者ガイド
- Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス

また、Oracle Data Pumpのタスクに関連するMy Oracle Supportのノートを参照し、特に、ご使用のリリースに推奨されるプロアクティブ・パッチを参照してください:

Data Pump Recommended Proactive Patches For 19.10 and Above (ドキュメントID 2819284.1)

Oracle Data Pumpのパッチは、Oracle Databaseリリースの更新には含まれませんが、SQL、PL/SQLパッケージおよびOracle Data Pump用のXMLスタイルシートを含むバンドル・パッチで提供されます。ご使用のリリース用の最新のOracle Data Pumpバンドル・パッチを適用することをお薦めします。これらのパッチにはOracle Databaseのバイナリが含まれないため、Oracle Data Pumpがその時点で使用されていないかぎり、データベースの実行中にOracle Data Pumpのパッチをオンラインで適用できます。

このマニュアル内の例の一部で、Oracle Databaseのインストール時にデフォルトとしてインストールされるシード・データベースのサンプル・スキーマを使用しています。サンプル・スキーマの作成方法および使用方法の詳細は、『Oracle Databaseサンプル・スキーマ』を参照してください。

親トピック: はじめに

構文図

このマニュアルには、様々なSQL、PL/SQLまたはその他のコマンドライン構造体の構文の説明が、図形式またはバッカス正規形(BNF)で記載されています。これらの説明の解釈方法は、『Oracle Database SQL言語リファレンス』を参照してください。

親トピック: はじめに

表記規則

このドキュメントでは、次のテキスト表記規則が使用されます:

規則	意味

規則	意味
boldface	太字体は、アクションに関連付けられたグラフィカル・ユーザー・インタフェース要素や、本文または用語集で定義されている用語を示します。
italic	イタリック体は、ブック・タイトル、強調、またはユーザーが特定の値を指定するプレースホルダー変数を示します。
`monospace`	等幅体は、段落内のコマンド、URL、サンプル内のコード、画面に表示されるテキスト、またはユーザーが入力するテキストを示します。

親トピック: はじめに

第I部 Oracle Data Pump

Oracle Data Pump Export、Oracle Data Pump Import、レガシー・モード、パフォーマンスおよびOracle Data Pump API DBMS_DATAPUMPを使用したデータ移動オプションについて学習します。

- Oracle Data Pumpの概要
  この章では、Oracle Data Pumpテクノロジの概要を説明します。Oracle Data Pumpとは、データおよびメタデータをデータベース間で非常に高速に移動できるテクノロジです。
- Oracle Data Pumpエクスポート
  Oracle Data Pumpエクスポート・ユーティリティは、ダンプ・ファイル・セットと呼ばれる一連のオペレーティング・システム・ファイルにデータとメタデータをアンロードするために使用します。
- Oracleデータ・ポンプ・インポート
  Oracleデータ・ポンプ・インポートを使用すると、ターゲット・データベースにエクスポート・ダンプ・ファイル・セットをロードできます。また、中間ファイルを使用することなく、ソース・データベースからターゲット・データベースへの直接的なロードも可能です。
- Oracle Data Pumpのレガシー・モード
  Oracle Data Pumpのレガシー・モードでは、Oracle Data Pumpエクスポートとデータ・ポンプ・インポートのコマンドラインで、オリジナルのエクスポート・パラメータとインポート・パラメータを使用できます。
- Oracle Data Pumpのパフォーマンス
  Oracle Data Pumpエクスポートおよびインポートがオリジナルのエクスポートおよびインポート・ユーティリティよりも優れている点と、エクスポート操作およびインポート操作のパフォーマンスを向上させる方法について説明します。
- Oracle Data Pump APIの使用
  データ移動の操作は、Oracle Data Pump PL/SQL APIのDBMS_DATAPUMPを使用することで自動化できます。

1 Oracle Data Pumpの概要

Oracle Data Pumpテクノロジを使用すると、データおよびメタデータをデータベース間で非常に高速に移動できます。

次の項目を理解することは、Oracle Data Pumpを最大限に活用するために役立ちます。

- Oracle Data Pumpのコンポーネント
  Oracle Data Pumpは、コマンドライン・クライアントのexpdpとimpdp、DBMS_DATAPUMP PL/SQLパッケージ(データ・ポンプAPI)、およびDBMS_METADATA PL/SQLパッケージ(メタデータAPI)の3つの個別のコンポーネントで構成されています。
- Oracle Data Pumpによるデータの移動方法
  Oracle Data Pumpによりデータベース内およびデータベース外にデータを移動するには、いくつかの方法を使用できます。ユースケースに最適な方法を選択できます。
- CDBでのOracle Data Pumpの使用
  Oracle Data Pumpを使用すると、非CDBのデータベースの全部または一部をPDBに移行することも、同じCDB内または異なるCDB内のPDB間で移行することも、PDBから非CDBに移行することもできます。
- Oracle Data Pumpのエクスポート操作およびインポート操作に必要なロール
  エクスポートおよびインポートの多くの操作には、DATAPUMP_EXP_FULL_DATABASEロールとDATAPUMP_IMP_FULL_DATABASEロールが必要になります。
- Oracle Data Pumpジョブの実行中の処理
  Oracle Data Pumpジョブでは、データ・ポンプ制御ジョブ表、データ・ポンプ制御ジョブ・プロセスおよびワーカー・プロセスを使用して、作業を実行し、進行状況を追跡します。
- Oracle Data Pumpジョブの状態の監視方法
  Oracle Data Pump ExportおよびOracle Data Pump Importクライアント・ユーティリティでは、ロギング・モードまたは対話方式コマンド・モードのどちらでもジョブに接続できます。
- ジョブ実行の進行状況の監視方法
  表データの転送を監視する場合は、V$SESSION_LONGOPS動的パフォーマンス・ビューを使用するとOracle Data Pumpのジョブを監視できます。
- Oracle Data Pumpによるファイル割当て
  Oracle Data Pumpのファイルの割当て方法と処理方法は、対話方式モードのコマンドを使用して変更できます。
- 異なるOracle Databaseリリース間のエクスポートとインポート
  Oracle Data Pumpを使用して、異なるリリースのデータベース・ソフトウェア間でOracle Databaseのすべてまたは一部を移行できます。
- Oracle Data Pumpを使用したSecureFilesラージ・オブジェクトのエクスポートの管理
  SecureFilesラージ・オブジェクト(LOB)のエクスポートは、コンテンツ・タイプ、VERSIONパラメータおよびその他の変数に影響されます。
- Oracle Data Pumpプロセス終了コード
  Oracle Data Pump ExportおよびOracle Data Pump Import操作の状態をチェックするには、ログ・ファイルでプロセス終了コードを確認します。
- 統合監査を使用したOracle Data Pumpジョブの監視方法
  特定のユーザー・データベース・アクションを監視および記録するには、統合監査を使用してデータ・ポンプ・ジョブの監査を実行します。
- Oracle Data Pump操作に対する暗号化されたデータのセキュリティ警告
  Oracle Data Pumpは、暗号化されたデータが暗号化されていないデータとしてエクスポートされると警告を表示しま

す。

- Oracle Data Pumpによるタイムスタンプ・データの処理方法
  タイムスタンプ・データ型のTIMESTAMP WITH TIMEZONEおよびTIMESTAMP WITH LOCAL TIMEZONEを含むエクスポートおよびインポート・ジョブの正常な完了に影響する可能性のある要因について学習します。
- 文字セットおよびグローバリゼーション・サポートに関する考慮点
  ユーザー・データの文字セット変換とデータ定義言語(DDL)を使用したOracle Data Pumpエクスポートおよびインポートのグローバリゼーション・サポートについて説明します。
- データ・バインドされた照合に関連するOracle Data Pumpの動作
  Oracle Data Pumpでは、データ・バインドされた照合(DBC)がサポートされます。

親トピック: Oracle Data Pump

1.1 Oracle Data Pumpのコンポーネント

Oracle Data Pumpは、コマンドライン・クライアントのexpdpとimpdp、DBMS_DATAPUMP PL/SQLパッケージ(データ・ポンプAPI)、およびDBMS_METADATA PL/SQLパッケージ(メタデータAPI)の3つの個別のコンポーネントで構成されています。

Oracle Data Pumpクライアントのexpdpとimpdpは、それぞれデータ・ポンプ・エクスポート・ユーティリティとデータ・ポンプ・インポート・ユーティリティを起動します。

expdpクライアントおよびimpdpクライアントでは、コマンドラインで入力されたパラメータを使用してエクスポートおよびインポートを実行するために、PL/SQLパッケージDBMS_DATAPUMPで提供されているプロシージャを使用します。これらのパラメータは、完全なデータベースまたはデータベースのサブセットに対するデータおよびメタデータをエクスポートおよびインポート可能にします。

メタデータを移動する場合、データ・ポンプでは、PL/SQLパッケージDBMS_METADATAで提供される機能が使用されます。DBMS_METADATAパッケージは、ディクショナリのメタデータの抽出、操作および再作成に関する集中的な機能を提供します。

DBMS_DATAPUMPおよびDBMS_METADATAの2つのPL/SQLパッケージは、データ・ポンプ・クライアントとは別に使用できます。

> ノート:
>
> Oracle Data Pump エクスポートおよびインポートのすべての処理(ダンプ・ファイルの読取りおよび書込みを含む)は、指定したデータベース接続文字列によって選択されるシステム(サーバー)上で実行されます。つまり、ユーザーに権限がない場合は、データベース管理者(DBA)が、そのサーバーのファイル・システムで読取りおよび書込みが実行されるデータ・ポンプ・ファイル用のディレクトリ・オブジェクトを作成する必要があります。(セキュリティ上の理由から、承認されたユーザーのみが、ディレクトリ・オブジェクトにアクセスできるようにする必要があります。)特権ユーザーは、デフォルトのディレクトリ・オブジェクトを使用できます。

Oracle Databaseリリース18c以降では、Oracle Data Pumpを使用して、完全または部分のどちらかのエクスポートおよびインポート操作に統合監査証跡を含めることができます。ユーザー・インタフェースの変更はありません。データベースのエクスポート操作またはインポート操作を実行するときに、統合監査証跡がOracle Data Pumpのダンプ・ファイルに自動的に含まれます。DBMS_DATAPUMPパッケージおよびDBMS_METADATAパッケージの詳細は、*Oracle Database PL/SQL*パッケージおよびタイプ・リファレンスを参照してください。Oracle Data Pumpを使用した統合監査証跡のエクスポートとインポートの詳細は、『*Oracle Database*セキュリティ・ガイド』を参照してください。

関連トピック

- ダンプ・ファイル、ログ・ファイルおよびSQLファイルのデフォルトの位置の理解

- Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス
- 『Oracle Databaseセキュリティ・ガイド』

親トピック: Oracle Data Pumpの概要

1.2 Oracle Data Pumpによるデータの移動方法

Oracle Data Pumpによりデータベース内におよびデータベース外にデータを移動するには、いくつかの方法を使用できます。ユースケースに最適な方法を選択できます。

> ノート:
>
> Oracle Database 18c 以降のリリースでの UTL_FILE_DIR のサポート終了は、Oracle Data Pump に影響します。このサポート終了は、Oracle Data Pump、BFILE および外部表を含む(ただし、これに制限されない)、シンボリック・リンクを使用する以前のリリースの機能に影響する可能性があります。シンボリック・リンクで構成された影響を受ける機能を使用しようとした場合、ORA-29283: 無効なファイル操作です。: パスが symlink をトラバースしていますが発生します。シンボリック・リンクのかわりにディレクトリ・オブジェクトを使用することをお薦めします。
>
> データ・ポンプでは、無効な一意索引を持つ表は、ロードされません。データを表にロードするには、その索引を削除するかまたは再度有効にする必要があります。

- データ・ファイル・コピーを使用したデータ移動
  最も高速なデータ移動の方法は、データベースのデータ・ファイルを、データの解析や変更を行わずに、ターゲット・データベースにコピーすることです。この方法では、データ・ポンプ・エクスポートを使用して、構造的な情報(メタデータ)のみをダンプ・ファイルにアンロードします。
- ダイレクト・パスを使用したデータ移動
  ダイレクト・パスは、データ・ファイル・コピーの次に高速なデータ移動方法です。この方法では、データベースのSQLレイヤーはバイパスされ、最小限の解析のみで行の移動がダンプ・ファイル間で行われます。
- 外部表を使用したデータ移動
  データ・ファイル・コピーを選択しないときに、データの移動にダイレクト・パスが使用できない場合は、外部表メカニズムを使用できます。
- 従来型パスを使用したデータ移動
  表属性の競合が存在する場合、Oracle Data Pumpは従来型パスを使用してデータを移動します。
- ネットワーク・リンク・インポートを使用したデータ移動
  インポート操作のネットワーク・リンクの指定でNETWORK_LINKインポート・パラメータを使用した場合、デフォルトでダイレクト・パス方式が使用されます。サポートされているデータベース・リンク・タイプを確認します。

親トピック: Oracle Data Pumpの概要

1.2.1 データ・ファイル・コピーを使用したデータ移動

最も高速なデータ移動の方法は、データベースのデータ・ファイルを、データの解析や変更を行わずに、ターゲット・データベースにコピーすることです。この方法では、データ・ポンプ・エクスポートを使用して、構造的な情報(メタデータ)のみをダンプ・ファイルにアンロードします。

- トランスポータブル表領域のエクスポートの指定に、TRANSPORT_TABLESPACESパラメータが使用されている場合。指定した表領域のメタデータのみがエクスポートされます。
- 表モード・エクスポート(TABLESパラメータで指定)、全体モード・エクスポート(FULLパラメータで指定)、または全体モード・ネットワーク・インポート(FULLおよびNETWORK_LINKパラメータで指定)でTRANSPORTABLE=ALWAYSパラメータが指定されている場合。

エクスポート操作でデータ・ファイル・コピーが使用されている場合、対応するインポート・ジョブでも常にデータ・ファイル・コピーが使用されます。その後のインポート操作時に、データ・ファイルとエクスポート・ダンプ・ファイルの両方をロードする必要があります。

ノート:

トランスポータブル・インポート中、表領域は一時的に読取り/書込み可能にされ、その後、読取り専用に戻されます。一時的な設定変更は、パフォーマンスを向上させるために、Oracle Database 12c リリース 1 (12.1.0.2)で導入されました。ただし、この動作により、インポート・ジョブ・データ・ファイルの SCN が変更される場合もあることに注意してください。データ・ファイルの SCN を変更すると、将来、これらのファイルをトランスポータブル・インポートするときに問題が発生する可能性があります。

たとえば、表領域を読取りまたは書き込み可能にした後に、トランスポータブル表領域インポートが失敗した場合(読取り専用に戻した場合でも)、データ・ファイルは破損します。これらはリカバリできません。

トランスポータブル・ジョブは再開できないため、失敗したジョブを最初から再開する必要があります。破損したデータ・ファイルを削除し、破損していないバージョンをターゲット宛先にコピーする必要があります。

トランスポータブル・ジョブを実行する場合、ベスト・プラクティスは、インポート・ジョブがターゲット・システムで正常に完了するまで、ソース・システムにデータ・ファイルのコピーを保持することです。インポート・ジョブがなんらかの理由で失敗しても、コピーを保持しておくことで、破損していないデータ・ファイルのコピーを確保できます。

データ・ファイル・コピーを使用してデータを移動する場合、ソース・データベースとターゲット・データベース間の文字セットの互換性に関する制限があります。

ソース・プラットフォームとターゲット・プラットフォームのエンディアンが異なる場合、転送するデータを変換してターゲット・プラットフォームの形式にする必要があります。DBMS_FILE_TRANSFER PL/SQLパッケージまたはRMAN CONVERTコマンドを使用してデータを変換できます。

関連項目:

- RMAN CONVERTコマンドの詳細は、Oracle Databaseバックアップおよびリカバリ・リファレンスを参照してください
- データベース間で行う表領域の転送の詳細および例(データの変換方法を含む)は、Oracle Database管理者ガイドを参照してください

親トピック: Oracle Data Pumpによるデータの移動方法

1.2.2 ダイレクト・パスを使用したデータ移動

ダイレクト・パスは、データ・ファイル・コピーの次に高速なデータ移動方法です。この方法では、データベースのSQLレイヤーはバイパスされ、最小限の解析のみで行の移動がダンプ・ファイル間で行われます。

データ・ポンプでは、データをロードおよびアンロードするために、表の構造上可能なかぎり、自動的にダイレクト・パスによる方法が使用されます。たとえば、表にBFILE型の列が含まれる場合は、ダイレクト・パスを使用して表をロードできないため、かわりに外部表が使用されます。

次の項では、ロードおよびアンロードに、ダイレクト・パスを使用できない場合について説明します。

ダイレクト・パス・ロードが使用されない場合

表が次に示すいずれかの条件に該当する場合、データ・ポンプでは、その表へのデータのロードにダイレクト・パスではなく外部表が使用されます。

- CONTEXTタイプの索引ではないドメイン索引がLOB列に存在する。
- 単一パーティションのロード中に、複数パーティション表のグローバル索引が存在する。この場合、パーティション化されたオブジェクト表も含まれる。
- クラスタ内に表が存在する。
- 既存の表にアクティブなトリガーが存在する。
- 既存の表で、ファイングレイン・アクセス・コントロールが挿入モードで有効である。
- 表にBFILE列または不透明な型の列が存在する。
- 既存の表に参照整合性制約が存在する。
- 表に不透明な型が埋め込まれたVARRAY列が存在する。
- 表に暗号化された列がある。
- データのインポート先が既存の表であり、次に示す1つ以上の条件に該当する。
    - アクティブなトリガーが存在する。
    - 表がパーティション化されている。
    - ファイングレイン・アクセス・コントロールが挿入モードである。
    - 参照整合性制約が存在する。
    - 一意索引が存在する。
- サプリメンタル・ロギングが有効で、表に1つ以上のLOB列がある。
- 指定した表のデータ・ポンプ・コマンドが、QUERY、SAMPLEまたはREMAP_DATAパラメータを使用している。
- 表にTIMESTAMP WITH TIME ZONEデータ型を持つ列(VARRAY列を含む)が含まれ、かつ、タイムゾーン・データ・ファイルのバージョンがエクスポート・システムとインポート・システムの間で異なる。

ダイレクト・パス・アンロードが使用されない場合

表が次に示すいずれかの条件に該当する場合、データ・ポンプでは、その表へのデータのアンロードにダイレクト・パスではなく外部表が使用されます。

- SELECTに対するファイングレイン・アクセス・コントロールが有効である。
- 表は、キュー表である。
- 表に1つまたは複数のBFILE型または不透明な型の列、あるいは不透明な列を含むオブジェクト型が存在する。
- 表に暗号化された列がある。

- 表にアップグレードが必要な進化した型の列がある。
- 指定した表のデータ・ポンプ・コマンドが、QUERY、SAMPLEまたはREMAP_DATAパラメータを使用している。
- アンロード操作の前に、NOT NULLかつ指定されたデフォルト値を持つ列を含むように表が変更されている。

親トピック: Oracle Data Pumpによるデータの移動方法

1.2.3 外部表を使用したデータ移動

データ・ファイル・コピーを選択しないときに、データの移動にダイレクト・パスが使用できない場合は、外部表メカニズムを使用できます。

外部表によるメカニズムでは、データベース表のダンプ・ファイル・データにマップする外部表が作成されます。その後、SQLエンジンを使用してデータが移動されます。可能な場合は、データベースへのデータのコピーを高速化するために、インポート時にAPPENDヒントが使用されます。ダイレクト・パス・データと外部表データは、ダンプ・ファイル内で同様に表示されます。これらは同じであるため、Oracle Data Pumpはエクスポート時にダイレクト・パス・メカニズムを使用しても、ターゲット・データベースへのデータのインポート時には外部表を使用することがあります。同様に、Oracle Data Pumpはエクスポートに外部表を使用し、インポートにダイレクト・パスを使用することもあります。

特に、Oracle Data Pumpは次の状況で外部表を使用することがあります。

- パラレルSQL機能の使用がメリットとなる状況において、非常に大きい表およびパーティションをロードおよびアンロードする場合
- グローバル索引またはドメイン索引が定義されている表(パーティション・オブジェクト表など)をロードする場合
- トリガーがアクティブになっている表またはクラスタ表をロードする場合
- 暗号化された列が含まれている表をロードおよびアンロードする場合
- ファイングレイン・アクセス・コントロールでの挿入が使用可能な表をロードする場合
- インポート操作以外で作成された表をロードする場合(インポートを開始する前から表が存在する)

ノート:

Oracle Data Pump がデータ・アクセスに外部表を使用する場合は、ORACLE_DATAPUMP アクセス・ドライバが使用されます。ただし、データ・ポンプが外部表を使用するときに作成するファイルは、ユーザーが SQL CREATE TABLE ... ORGANIZATION EXTERNAL 文を使用して手動で外部表を作成したときに作成されるファイルとは互換性がない点に注意してください。

関連トピック

- ORACLE_DATAPUMPアクセス・ドライバ

関連項目:

APPENDヒントの使用方法の詳細は、Oracle Database SQL言語リファレンスを参照してください

親トピック: Oracle Data Pumpによるデータの移動方法

1.2.4 従来型パスを使用したデータ移動

表属性の競合が存在する場合、Oracle Data Pumpは従来型パスを使用してデータを移動します。

表属性の競合が存在すると、Oracle Data Pumpは、ダイレクト・パスと外部表のどちらを使用してもデータを表にロードできません。このような場合は従来型パスが使用されますが、パフォーマンスに影響する可能性があります。

親トピック: Oracle Data Pumpによるデータの移動方法

1.2.5 ネットワーク・リンク・インポートを使用したデータ移動

インポート操作のネットワーク・リンクの指定でNETWORK_LINKインポート・パラメータを使用した場合、デフォルトでダイレクト・パス方式が使用されます。サポートされているデータベース・リンク・タイプを確認します。

(たとえば、列の1つがBFILEであるなどの理由で)ダイレクト・パスを使用できない場合、データの移動にはSQLのINSERT SELECT文が使用されます。(Oracle Database 12cリリース2 (12.2.0.1)より前には、デフォルトでINSERT SELECT文が使用されていました。)SELECT句は、ネットワーク・リンク上のリモート・データベースからデータを取得します。INSERT句はSQLを使用してデータをターゲット・データベースに挿入します。ダンプ・ファイルは含まれません。

エクスポート操作のネットワーク・リンクの指定にNETWORK_LINKエクスポート・パラメータが使用されている場合、リモート・データベースのデータがターゲット・データベースのダンプ・ファイルに書き込まれます。(読取り専用のデータベースからのエクスポートには、NETWORK_LINKパラメータが必要です。)

リンクとは、ネットワーク接続されたリモートのデータベースを示すため、データベース・リンクおよびネットワーク・リンクという用語は、同じ意味で使用されます。

サポートされているリンク・タイプ

データ・ポンプ・エクスポートおよびデータ・ポンプ・インポートでは、次のタイプのデータベース・リンクの使用がサポートされています。

- パブリック固定ユーザー
- パブリック接続ユーザー
- パブリック共有ユーザー(リンク所有者により使用される場合のみ)
- プライベート共有ユーザー(リンク所有者により使用される場合のみ)
- プライベート固定ユーザー(リンク所有者により使用される場合のみ)

サポートされていないリンク・タイプ

データ・ポンプ・エクスポートおよびデータ・ポンプ・インポートでは、次のタイプのデータベース・リンクの使用はサポートされていません。

- プライベート接続ユーザー
- 現在のユーザー
- ネットワーク・ジョブのメタデータのパラレル・エクスポートまたはインポート。

  従来型のジョブについては、メタデータのパラレル・インポートが必要な場合は、NETWORK_LINKのかわりにダンプ・ファイルを使用してください。

関連項目:

- データベース・リンクを介したエクスポートの実行の詳細は、エクスポートの「NETWORK_LINK」パラメータを参照してください。
- データベース・リンクを介したインポートの実行の詳細は、インポートの「NETWORK_LINK」パラメータを参照してください。
- データベース・リンクおよび異なるタイプのリンクを作成する方法の詳細は、Oracle Database管理者ガイドを参照してください

親トピック: Oracle Data Pumpによるデータの移動方法

1.3 CDBでのOracle Data Pumpの使用

Oracle Data Pumpを使用すると、非CDBのデータベースの全部または一部をPDBに移行することも、同じCDB内または異なるCDB内のPDB間で移行することも、PDBから非CDBに移行することもできます。

- マルチテナント環境でのOracle Data Pumpの使用について
  一般的に、PDBでOracle Data Pumpを使用することは、非CDBでOracle Data Pumpを使用することと同じです。
- Oracle Data Pumpを使用したCDBへのデータの移動
  空のPDBを作成した後にデータをPDBに移動する場合、Oracle Data Pumpの全体モード・エクスポートおよびインポート操作を使用できます。
- Oracle Data Pumpの使用によるCDB内または間のPDBの移動
  PDBでのOracle Data Pumpのエクスポートおよびインポート操作で`ORA-65094`ユーザー・スキーマ・エラーを回避する方法について説明します。

親トピック: Oracle Data Pumpの概要

1.3.1 マルチテナント環境でのOracle Data Pumpの使用について

一般に、PDBでのOracle Data Pumpの使用は、非CDBでのOracle Data Pumpの使用と同じです。

マルチテナント・コンテナ・データベース(CDB)は、0、1または多数のユーザー作成のプラガブル・データベース(PDB)を含むOracle Databaseです。PDBは、スキーマ、スキーマ・オブジェクトおよび非スキーマ・オブジェクトのポータブル・セットで、Oracle Netクライアントからは、非CDBとして表示されます。非CDBとは、CDBではないOracle Databaseです。非CDBアーキテクチャのOracle Databaseは、Oracle Database 12cリリース1 (12.1)で非推奨になりました。Oracle Database 21c以降、非CDBアーキテクチャのデプロイメントはサポートされなくなりました。

Oracle Data Pumpを使用して、次のシナリオでデータベースの一部またはすべてを移行できます。

- 非CDBからPDB
- 同じCDBまたは異なるCDB内のPDB間
- PDBから以前のリリースの非CDBへ

ノート:

Oracle Data Pump では、CDB 全体の操作はサポートされません。CDB のルートまたはシード・データベースに接続してい

```
ORA-39357: Warning: Oracle Data Pump operations are not typically needed when connec
```

親トピック: CDBでのOracle Data Pumpの使用

1.3.2 データ・ポンプを使用したCDBへのデータの移動

空のPDBを作成した後にPDBにデータを移動する場合、Oracle Data Pumpの全体モード・エクスポートおよびインポート操作を使用できます。

データはトランスポータブル・オプションを指定してもしなくてもインポートできます。トランスポータブル・オプションを全体モード・エクスポートまたはインポートで使用する場合、全体トランスポータブル・エクスポート/インポートと呼ばれます。

トランスポータブル・オプションが使用される場合、エクスポートおよびインポートでは、トランスポータブル表領域のデータ移動と従来のデータ移動の両方を使用しますが、後者はSYSTEMやSYSAUXなどの非トランスポータブル表領域に存在する表が対象です。トランスポータブル・オプションを使用すると、エクスポート時間や(特に)インポート時間を短縮できます。トランスポータブル・オプションを使用すると、表データをアンロードおよび再ロードする必要がありません。また、ユーザー表領域の索引構造を再作成する必要がありません。

Oracle Data Pumpを使用してCDBにデータを移動する場合、次の要件に注意してください。

- マルチテナント環境を管理するには、CDB_DBAロールを持っている必要があります。
- Oracle Database 11.2.0.2以前からの全体データベース・エクスポートは、Oracle Database 12c以降(CDBまたは非CDB)にインポートされる場合があります。ただし、登録したオプションおよびコンポーネントの情報がエクスポートに含まれるように、まずソース・データベースをOracle Database 11gリリース2 (11.2.0.3以降)にアップグレードすることをお薦めします。
- 全体データベース・エクスポートまたは全体トランスポータブル・データベース・エクスポートのいずれかを使用して、Oracle Database 11gリリース2 (11.2.0.3以上)をCDB(または非CDB)に移行する場合、Oracle Data Pumpのエクスポート・パラメータを少なくともVERSION=12に設定して、Oracle Database 12c以降のリリースにインポート可能なダンプ・ファイルを生成する必要があります。VERSION=12を設定しない場合、生成したエクスポート・ファイルには、登録したデータベース・オプションおよびコンポーネントの完全な情報は含まれません。
- ネットワーク・ベースの全体トランスポータブル・インポートでは、FULL=YES、TRANSPORTABLE=ALWAYSおよびTRANSPORT_DATAFILES=datafile_nameパラメータを使用する必要があります。ソース・データベースがOracle Database 11gリリース11.2.0.3以上で、Oracle Database 12cリリース1 (12.1)より前の場合、VERSION=12パラメータも必要です。
- ファイル・ベースの全体トランスポータブル・インポートでは、TRANSPORT_DATAFILES=datafile_nameパラメータのみを使用する必要があります。データ・ポンプ・インポートでは、TRANSPORTABLE=ALWAYSおよびFULL=YESパラメータの存在が推測されます。
- Oracle Database 12cリリース2 (12.2)以降、マルチテナント・コンテナ・データベース(CDB)環境では、CDBの各PDBに対する一意のパスとして、デフォルトのOracle Data Pumpディレクトリ・オブジェクトDATA_PUMP_DIRが定義されます。この一意のパスは、CREATE PLUGGABLE DATABASE文のPATH_PREFIX句が相対パスに定義されているかどうかに関係なく、定義されます。
- Oracle Database 19c以降、impdpのcredentialパラメータにより、オブジェクト・ストア・バケットへのアクセスに必要なユーザー名とパスワードを含む資格証明オブジェクトの名前を指定します。DEFAULT_CREDENTIALという名前のPDBプロパティを使用して、デフォルトの資格証明を指定することもできます。デフォルトの資格証明を使用してimpdbを実行する場合は、ダンプ・ファイル名に接頭辞DEFAULT_CREDENTIAL:を付け、credentialパラメー

タは指定しません。

例1-1 PDBへの表のインポート

エクスポート/インポート操作に対して特定のPDBを指定するには、データ・ポンプの開始時に接続文字列に接続識別子を指定します。たとえば、pdb1というPDBにデータをインポートするには、データ・ポンプ・コマンドラインで次のように入力します。

```
impdp hr@pdb1 DIRECTORY=dpump_dir1 DUMPFILE=hr.dmp TABLES=employees
```

例1-2 データをインポートする際の資格証明の指定

この例では、次のように、DBMS_CREDENTIAL.CREATE_CREDENTIALを使用してHR_CREDという名前の資格証明を作成していることを前提としています。

```
BEGIN
  DBMS_CLOUD.CREATE_CREDENTIAL(
    credential_name => 'HR_CRED',
    username => 'atpc_user@example.com',
    password => 'password'
  );
END;
/
```

次のコマンドは、資格証明HR_CREDを指定し、オブジェクト・ストアに格納されているファイルを指定します。ファイルのURLはhttps://example.com/ostore/dnfs/myt.dmpです。

```
impdp hr@pdb1 ¥
      table_exists_action=replace ¥
      credential=HR_CRED ¥
      parallel=16 ¥
      dumpfile=https://example.com/ostore/dnfs/myt.dmp
```

例1-3 デフォルトの資格証明を使用したデータのインポート

1. 次のように、DBMS_CREDENTIAL.CREATE_CREDENTIALを使用してHR_CREDという名前の資格証明を作成します。

   ```
   BEGIN
     DBMS_CLOUD.CREATE_CREDENTIAL(
       credential_name => 'HR_CRED',
       username => 'atpc_user@example.com',
       password => 'password'
     );
   END;
   /
   ```

2. 次のように、PDBプロパティDEFAULT_CREDENTIALを設定します。

   ```
   ALTER DATABASE PROPERTY SET DEFAULT_CREDENTIAL = 'ADMIN.HR_CRED'
   ```

3. 次のコマンドは、ダンプ・ファイルの場所https://example.com/ostore/dnfs/myt.dmpに対する接頭辞としてデフォルトの資格証明を指定します。

   ```
   impdp hr@pdb1 ¥
         table_exists_action=replace ¥
         parallel=16 ¥
         dumpfile=default_credential:https://example.com/ostore/dnfs/myt.dmp
   ```

   credentialパラメータが指定されていないことに注意してください。

関連項目:

- オブジェクト・ストアのアクセスに必要なSSL認証を構成する方法を学習するには、Oracle Databaseセキュリティ・ガイドを参照してください
- Oracle Data Pumpインポートを使用してオブジェクト・ストアにファイルをロードする方法について学習するには、『Oracle Databaseユーティリティ』を参照してください

親トピック: CDBでのOracle Data Pumpの使用

1.3.3 Oracle Data Pumpの使用によるCDB内または間のPDBの移動

PDBでのOracle Data Pumpのエクスポートおよびインポート操作でORA-65094ユーザー・スキーマ・エラーを回避する方法を学習します。

PDBでのエクスポートおよびインポート操作は、共通ユーザーの処理方法を除き、非CDBに対する操作と同じです。

CDBで共通ユーザーを作成する場合、CDBのPDB内からそのユーザーの全データベースまたは特権スキーマをエクスポートすると、標準のCREATE USER C##common name DDL文がインポート時に実行されます。ただし、ユーザー名の共通ユーザー接頭辞C##が原因で、文は失敗します。次のエラー・メッセージが返された場合:

```
ORA-65094:invalid local user or role name
```

例1-4 無効なローカル・ユーザー・エラーの回避

エクスポート対象のPDBで、そのユーザーのスキーマにローカル・オブジェクトを作成してそれらをインポートする場合、同じ名前の共通ユーザーがすでにターゲットCDBインスタンスに存在することを確認するか、impdpコマンドでOracle Data PumpのインポートREMAP_SCHEMAパラメータを使用して、有効なローカル・ユーザーにスキーマを再マップします。たとえば:

```
REMAP_SCHEMA=C##common name:local user name
```

関連トピック

- 全体モード
- 全体インポート・モード
- オリジナルのOracle Data Pumpエクスポートのネットワーク考慮事項

親トピック: CDBでのOracle Data Pumpの使用

1.4 Oracle Data Pumpのエクスポート操作およびインポート操作に必要なロール

エクスポートおよびインポートの多くの操作には、DATAPUMP_EXP_FULL_DATABASEロールとDATAPUMP_IMP_FULL_DATABASEロールが必要になります。

エクスポート操作またはインポート操作を実行するときには、その操作の実行に使用しているユーザー・アカウントにDATAPUMP_EXP_FULL_DATABASEロールまたはDATAPUMP_IMP_FULL_DATABASEロール(あるいはその両方)が付与されていることが必要になる場合があります。これらのロールは、データベース作成の一環として標準スクリプトを実行するとOracle Database用に自動的に定義されます。(これらのロールの名前にはFULLという語が含まれますが、これらのロールは、実際にはFullモードのみではなく、任意のエクスポートまたはインポート・モードのすべての特権操作に適用される点に注意してください)。

DATAPUMP_EXP_FULL_DATABASEロールはエクスポート操作にのみ影響します。DATAPUMP_IMP_FULL_DATABASE

ロールは、インポート操作と、SQLFILEインポート・パラメータを使用する操作に影響します。これらのロールによって、ユーザーは次の処理を行うエクスポートおよびインポート操作を実行できます。

- 自分のスキーマ・スコープ以外での操作の実行
- 他のユーザーが開始したジョブの監視
- 権限のないユーザーが参照できないオブジェクトのエクスポート(表領域定義など)とインポート(ディレクトリ定義など)

これらは強力なロールです。管理者として、これらのロールをユーザーに付与する際は、十分に注意する必要があります。

SYSスキーマには、これらのいずれのロールも割り当てられていません。ただし、Oracle Data Pumpが実行するセキュリティ・チェックうち、これらのロールを必要とするものについてはSYSスキーマへのアクセス権が付与されます。

> ノート:
>
> ORA-39181: ファイングレイン・アクセス・コントロールのため、データの一部のみがエクスポートされましたというエラー・メッセージが表示される場合は、My Oracle Support ノート 422480.1 のファイングレイン・アクセス・コントロール・ポリシーが有効化されている表データのエクスポート時のセキュリティに関する情報を参照してください。
>
> https://support.oracle.com/rs?type=doc&id=422480.1

Oracleロールには、認証が必要なものがあります。そのようなロールをOracle Data Pumpのエクスポートおよびインポートで使用する必要がある場合は、ENABLE_SECURE_ROLESパラメータをYESに設定することで明示的に使用可能にする必要があります。

関連項目:

Oracle Databaseのインストールで事前定義されるロールの詳細は、Oracle Databaseセキュリティ・ガイドを参照してください

親トピック: Oracle Data Pumpの概要

1.5 Oracle Data Pumpジョブの実行時の処理について

Oracle Data Pumpジョブでは、データ・ポンプ制御ジョブ表、データ・ポンプ制御ジョブ・プロセスおよびワーカー・プロセスを使用して、作業を実行し、進行状況を追跡します。

- Oracle Data Pumpジョブの調整
  すべてのOracle Data Pumpエクスポートおよびインポート・ジョブを調整するために、データ・ポンプ制御プロセスが作成されます。
- ジョブ内での進捗状況の追跡
  データおよびメタデータの転送中に、ジョブ内の進捗状況の追跡に親ジョブ表が使用されます。
- Oracle Data Pumpジョブ実行中のデータおよびメタデータのフィルタ処理
  Oracle Data Pumpによってエクスポートおよびインポートされるオブジェクトのタイプをフィルタ処理する場合、EXCLUDEおよびINCLUDEパラメータを使用できます。
- Oracle Data Pumpジョブ実行中のメタデータの変換
  データベース間でデータを移動するときには、Oracle Data Pump Importのパラメータを使用してメタデータの変換

を実行できます。

- Oracle Data Pumpのジョブ・パフォーマンスの最大化
  ジョブ・パフォーマンスが向上するように、Oracle Data PumpのPARALLELパラメータを使用して、複数のジョブ・プロセスをパラレルで実行できます。
- Oracle Data Pumpによるデータのロードおよびアンロード
  データのインポートおよびエクスポート時にOracle Data Pump子プロセスがどのように動作するかを説明します。

親トピック: Oracle Data Pumpの概要

1.5.1 Oracle Data Pumpジョブの調整

すべてのOracle Data Pumpエクスポートおよびインポート・ジョブを調整するために、データ・ポンプ制御プロセスが作成されます。

Data Pump制御プロセスによって、ジョブ全体(クライアント・プロセスとの通信、ワーカー・プロセス・プールの作成および制御、ロギング操作の実行など)が制御されます。

親トピック: Oracle Data Pumpジョブの実行時の処理について

1.5.2 ジョブ内での進捗状況の追跡

データおよびメタデータの転送中に、ジョブ内の進捗状況の追跡に親ジョブ表が使用されます。

親ジョブ表は、ユーザー表としてデータベース内に実装されます。また、エクスポート・ジョブおよびインポート・ジョブ用の親ジョブ表固有の機能は、次のとおりです。

- エクスポート・ジョブの場合、親ジョブ表には、ダンプ・ファイル・セット内のデータベース・オブジェクトの位置が記録されます。エクスポート・ユーティリティでは、ジョブの継続中に、親ジョブ表の作成およびメンテナンスが行われます。エクスポート・ジョブ終了時に、親ジョブ表の内容がダンプ・ファイル・セット内のファイルに書き込まれます。
- インポート・ジョブの場合、親ジョブ表は、ダンプ・ファイル・セットからロードされ、ターゲット・データベースにインポートする必要があるオブジェクトの位置を特定する操作の順序の制御に使用されます。

親ジョブ表は、エクスポートまたはインポート操作を実行している現在のユーザーのスキーマ内に作成されます。そのため、このユーザーには、CREATE TABLEシステム権限、および親ジョブ表を作成するための十分な表領域の割当て制限が必要です。親ジョブ表は、その表を作成したジョブと同じ名前になります。したがって、Oracle Data Pumpジョブに、既存の表またはビューと同じ名前を明示的には指定できません。

すべての操作で、ジョブの再起動に親ジョブ表内の情報が使用されます。(トランスポータブル・ジョブは再開できないことに注意してください。)

親ジョブ表は、状況に応じて、次のとおり保持または削除されます。

- ジョブが正常に終了すると、親ジョブ表は削除されます。これを上書きするには、ジョブに対してOracle Data PumpのKEEP_MASTER=YESパラメータを設定します。
- ジョブが正常に完了しなかった場合、親ジョブ表は自動的に保持されます。
- 対話方式コマンドSTOP_JOBを使用してジョブを停止すると、ジョブの再起動に使用できるように親ジョブ表は保持されます。
- 対話方式コマンドKILL_JOBを使用してジョブを中断すると、親ジョブ表は削除され、ジョブを再起動できません。
- ジョブが突然終了した場合でも、親ジョブ表は保持されます。このジョブを再起動しない場合、マスター表を削除できま

す。

- ジョブが実行開始前(つまり、データベース・オブジェクトがコピーされる前)に停止すると、親ジョブ表は削除されます。

関連トピック

- JOB_NAME

親トピック: Oracle Data Pumpジョブの実行時の処理について

1.5.3 Oracle Data Pumpジョブ実行中のデータおよびメタデータのフィルタ処理

Oracle Data Pumpによってエクスポートおよびインポートされるオブジェクトのタイプをフィルタ処理する場合、EXCLUDEおよびINCLUDEパラメータを使用できます。

マスター表内では、名前や自分のスキーマなどの属性が特定のオブジェクトに割り当てられます。オブジェクトは、オブジェクトのクラス(TABLE、INDEX、DIRECTORYなど)にも属します。オブジェクトのクラスは、オブジェクトのオブジェクト型と呼ばれます。EXCLUDEおよびINCLUDEパラメータを使用して、エクスポートまたはインポートされるオブジェクト型を制限できます。オブジェクトの名前またはオブジェクトを自分のスキーマの名前に基づいて、制限するオブジェクトを指定できます。データ固有のフィルタを指定して、エクスポートおよびインポートする行を制限することもできます。

関連トピック

- エクスポート操作中のフィルタ処理
- インポート操作中のフィルタ処理

親トピック: Oracle Data Pumpジョブの実行時の処理について

1.5.4 Oracle Data Pumpジョブ実行中のメタデータの変換

データベース間でデータを移動するときには、Oracle Data Pump Importのパラメータを使用してメタデータの変換を実行できます。

多くの場合、表領域間で記憶域を再マッピングするためや、特定のオブジェクト・セットの所有者を再定義するために、メタデータの変換を実行することが役立ちます。データを移動するときに、Oracle Data Pump ImportパラメータのREMAP_DATAFILE、REMAP_SCHEMA、REMAP_TABLE、 REMAP_TABLESPACE、TRANSFORMおよびPARTITION_OPTIONSを使用して変換を実行できます。

親トピック: Oracle Data Pumpジョブの実行時の処理について

1.5.5 Oracle Data Pumpジョブのパフォーマンスの最大化

ジョブ・パフォーマンスが向上するように、Oracle Data PumpのPARALLELパラメータを使用して、複数のジョブ・プロセスをパラレルで実行できます。

PARALLELパラメータを使用すると、現状の環境で最大の効果が得られる並列度を設定できます。たとえば、本番システムへのジョブの影響を制限するために、データベース管理者は並列度を制限できます。並列度は、ジョブの実行中いつでも再設定できます。たとえば、運用時間中はPARALLELを2に設定して、特定のジョブの並列度を2に制限します。運用時間外は、並列度を8にリセットします。並列度の設定は、操作内でデータおよびメタデータ処理を実行するワーカー・プロセスにワークロードを割り当てるデータ・ポンプ制御プロセスによって強制されます。これらのワーカー・プロセスは、パラレルで動作します。並列度の設定に関する推奨事項は、エクスポートのPARALLELおよびインポートのPARALLELパラメータの説明を参照してください。

ノート:

並列度を調整する機能は、Oracle Database の Enterprise Edition でのみ使用可能です。

関連トピック

- PARALLEL
- PARALLEL

親トピック: Oracle Data Pumpジョブの実行時の処理について

1.5.6 Oracle Data Pumpによるデータのロードおよびアンロード

データのインポートおよびエクスポート時にOracle Data Pump子プロセスがどのように動作するかを学習します。

Oracle Data Pump子プロセスによって、メタデータおよび表データがアンロードおよびロードされます。エクスポートでは、トランスポータブル表領域を使用するジョブを除いて、すべてのメタデータおよびデータがパラレルでアンロードされます。インポートでは、正しい依存順序でオブジェクトを作成する必要があります。

同じタイプのオブジェクトが、複数の子プロセスを使用するのに十分なだけ存在する場合、それらのオブジェクトは複数の子プロセスによってインポートされます。一部のメタデータ・オブジェクトには相互依存性があり、その依存性を満たすために、1つの子プロセスで、これらのオブジェクトを順に作成する必要があります。子プロセスは、コマンドライン・パラメータPARALLELに指定した値と同数になるまで必要に応じて作成されます。アクティブな子プロセスの数は、ジョブの存続期間中いつでも再設定できます。ワーカー・プロセスは、Oracle Real Application Clusters(Oracle RAC)環境内の様々なノードで起動できます。

ノート:

Oracle Database の Standard Edition では、PARALLEL の値は 1 に制限されています。

非常に大きい表またはパーティションをロードまたはアンロードするタスクが子プロセスに割り当てられた場合、パラレル実行を最大限に利用できるようにするため、外部表によるアクセス方法を使用できます。その場合、子プロセスはパラレル実行コーディネータになります。実際のロードおよびアンロード作業は、Oracle Real Application Clusters(Oracle RAC)環境で使用可能なプロセスのプールから割り当てられたパラレル入出力(I/O)実行プロセスの数に分割されます。

関連トピック

- PARALLEL
- PARALLEL

親トピック: Oracle Data Pumpジョブの実行時の処理について

1.6 Oracle Data Pumpジョブのステータスの監視方法

Oracle Data Pumpエクスポートおよびインポート・クライアント・ユーティリティでは、ロギング・モードまたは対話方式コマンド・モードのどちらでもジョブに接続できます。

ロギング・モードでは、ジョブの実行中に、そのジョブの詳細な状態がリアルタイムで自動的に表示されます。表示される情報には、ジョブおよびパラメータの説明、処理されるデータ量の推定、現在の操作または処理中のアイテムの説明、ジョブで使用されるファイル、発生したエラーおよび最終的なジョブの状態(停止または完了)があります。

対話方式コマンド・モードでは、ジョブの状態をリクエストに表示できます。表示される情報には、ジョブの説明および状態、現在の操作または処理中のアイテムの説明、書込み中のファイルおよび累積的な状態があります。

ジョブの実行中にログ・ファイルを書き込むこともできます。ログ・ファイルには、ジョブの進捗状況のサマリーが記録され、ジョブの実行中に発生したエラーがリストされます。また、ジョブの完了状態も記録されます。

Oracle Data Pumpジョブについての情報(ジョブの状態など)を調べるための別の方法として、DBA_DATAPUMP_JOBSビュー、USER_DATAPUMP_JOBSビュー、またはDBA_DATAPUMP_SESSIONSビューを問い合せることもできます。詳細は、『*Oracle Database*リファレンス』を参照してください。

関連トピック

- Oracle Databaseリファレンス

親トピック: Oracle Data Pumpの概要

1.7 ジョブ実行の進捗状況の監視方法

表データの転送を監視する場合は、V$SESSION_LONGOPS動的パフォーマンス・ビューを使用するとOracle Data Pumpのジョブを監視できます。

表データを転送するOracle Data Pumpの操作(エクスポートおよびインポート)では、ジョブの進捗状況(単位は、転送された表データのMB数)を示す動的パフォーマンス・ビューV$SESSION_LONGOPSのエントリが保持されます。このエントリは、転送の推定サイズを含み、実際に転送されたデータの量が反映されるように定期的に更新されます。

パラメータCOMPRESSION、ENCRYPTION、ENCRYPTION_ALGORITHM、ENCRYPTION_MODE、ENCRYPTION_PASSWORD、QUERYおよびREMAP_DATAを使用しても、推定値の決定には反映されません。

エクスポート操作の推定値が有効かどうかは、操作開始時に要求された推定のタイプによって異なります。この値は、実際の転送量を超えた場合に必要に応じて更新されます。インポート操作の推定値は、厳密な値です。

データ・ポンプ・ジョブに関連するV$SESSION_LONGOPS列は、次のとおりです。

- USERNAME: ジョブの所有者
- OPNAME: ジョブ名
- TARGET_DESC: ジョブ操作
- SOFAR: この時点までのジョブ実行中に転送されたMB数
- TOTALWORK: ジョブ内の推定MB数
- UNITS: メガバイト(MB)
- MESSAGE: 状態メッセージ。次の形式が使用されます。

```
'job_name: operation_name : nnn out of mmm MB done'
```

親トピック: Oracle Data Pumpの概要

1.8 Oracle Data Pumpによるファイル割当て

Oracle Data Pumpによるファイルの割当て方法と処理方法は、対話方式モードのコマンドを使用することで変更できます。

- Oracle Data Pumpでのファイル割当ての理解

Oracle Data Pumpによるファイルの割当て方法と処理方法を理解すると、エクスポート・ユーティリティとインポート・ユーティリティを最大限に活用できるようになります。

- ファイルの指定およびダンプ・ファイルの追加
  エクスポート操作の場合、ダンプ・ファイルは、Oracle Data Pumpジョブの定義時およびエクスポート操作の後の段階で指定できます。
- ダンプ・ファイル、ログ・ファイルおよびSQLファイルのデフォルトの位置
  Oracle Data Pumpのファイルのデフォルトの場所、およびOracle RAC、Oracle Automatic Storage Managementおよびマルチテナント・アーキテクチャを使用している場合にこれらの場所がどのように影響を受けるかを学習します。
- Oracle Data Pump Exportでの置換変数の使用
  Oracle Data Pump Export操作で複数のダンプ・ファイルを指定する場合は、ファイル名に置換変数を含む`DUMPFILE`パラメータを使用します。

親トピック: Oracle Data Pumpの概要

1.8.1 Oracle Data Pumpでのファイル割当ての理解

Oracle Data Pumpによるファイルの割当て方法と処理方法を理解すると、エクスポート・ユーティリティとインポート・ユーティリティを最大限に活用できるようになります。

Oracle Data Pumpジョブは、次のタイプのファイルを管理します。

- ダンプ・ファイル。移動中のデータとメタデータを格納します。
- ログ・ファイル。操作に関連したメッセージを記録します。
- SQLファイル。`SQLFILE`操作の出力を記録します。`SQLFILE`操作は、Oracle Data Pumpインポートの`SQLFILE`パラメータを使用して開始します。この操作により、その他のパラメータに基づいてインポート・ユーティリティが実行するすべての`SQL DDL`が、SQLファイルに書き込まれます。
- トランスポータブル・インポート中に`DATA_FILES`パラメータにより指定されるファイル

ノート:

Oracle Data Pump ジョブによってネットワーク・ファイル・ストレージ(NFS)関連のエラーが生成される場合は、対象プラットフォームのインストレーション・ガイドを参照して適切な NFS マウント設定を調べてください。

親トピック: Oracle Data Pumpによるファイル割当て

1.8.2 ファイルの指定およびダンプ・ファイルの追加

エクスポート操作の場合、ダンプ・ファイルは、Oracle Data Pumpジョブの定義時およびエクスポート操作の後の段階で指定できます。

エクスポート操作中に領域が不足した場合は、Oracle Data Pumpエクスポート・ユーティリティの`ADD_FILE`コマンドを対話方式モードで使用して、追加ダンプ・ファイルを追加できます。

インポート操作の場合、ジョブの定義時にすべてのダンプ・ファイルを指定する必要があります。

既存のファイルは、ログ・ファイルおよびSQLファイルによって上書きされます。ダンプ・ファイルの場合は、`REUSE_DUMPFILES`エ

クスポート・パラメータを使用して、既存のダンプ・ファイルを上書きするかどうかを指定できます。

親トピック: Oracle Data Pumpによるファイル割当て

1.8.3 ダンプ・ファイル、ログ・ファイルおよびSQLファイルのデフォルトの位置

Oracle Data Pumpのファイルのデフォルトの場所、およびOracle RAC、Oracle Automatic Storage Managementおよびマルチテナント・アーキテクチャを使用している場合にこれらの場所がどのように影響を受けるかを学習します。

- ダンプ・ファイル、ログ・ファイルおよびSQLファイルのデフォルトの位置の理解
  Oracle Data Pumpは、クライアントベースではなくサーバーベースです。ダンプ・ファイル、ログ・ファイルおよびSQLファイルはサーバーベースのディレクトリ・パスとの相対でアクセスされます。
- Oracle RACでのOracle Data Pumpの使用方法の理解
  Oracle Real Application Clusters (Oracle RAC)環境でOracle Data Pumpを使用するには、いくつかのチェックを実行して、クラスタ・メンバー・ノードが使用可能であることを確認する必要があります。
- Oracle Automatic Storage Managementを使用可能にした場合のディレクトリ・オブジェクトの使用方法
  Oracle Automatic Storage Management (Oracle ASM)を使用可能にしてOracle Data Pumpエクスポートまたはインポートを使用する場合は、ダンプ・ファイル用に使用するディレクトリ・オブジェクトを定義します。
- DATA_PUMP_DIRディレクトリ・オブジェクトおよびプラガブル・データベース
  CDB内の各PDBに対する一意のパスとして、デフォルトのOracle Data Pumpディレクトリ・オブジェクト`DATA_PUMP_DIR`を定義します。

親トピック: Oracle Data Pumpによるファイル割当て

1.8.3.1 ダンプ・ファイル、ログ・ファイルおよびSQLファイルのデフォルトの位置の理解

Oracle Data Pumpは、クライアントベースではなくサーバーベースです。ダンプ・ファイル、ログ・ファイルおよびSQLファイルはサーバーベースのディレクトリ・パスとの相対でアクセスされます。

Oracle Data Pumpでは、ディレクトリ・パスをディレクトリ・オブジェクトとして指定する必要があります。ディレクトリ・オブジェクトは、ファイル・システムのディレクトリ・パスに名前をマップします。データベース管理者は、ディレクトリ・パスに関連付けられたディレクトリ・オブジェクトに承認されたユーザーのみがアクセスできるようにする必要があります。

次の例は、`/usr/apps/datafiles`にあるディレクトリにマップされる`dpump_dir1`というディレクトリ・オブジェクトを作成するSQL文を示しています。

```
SQL> CREATE DIRECTORY dpump_dir1 AS '/usr/apps/datafiles';
```

ディレクトリ・オブジェクトは、データのセキュリティおよび整合性を確保するために必要です。たとえば:

- 入力ファイルのディレクトリ・パスの場所を指定する権限が付与されたユーザーは、サーバーにはアクセス権があるが、ユーザー自身はアクセス権を持たないデータの読取りを実行できる可能性があります。
- 出力ファイルのディレクトリ・パスの場所を指定できるユーザーは、通常は削除する権限のないファイルを上書きできるようになる可能性があります。

UNIX、LinuxおよびWindowsオペレーティング・システムの場合、デフォルトのディレクトリ・オブジェクト`DATA_PUMP_DIR`は、データベースの作成時、またはデータベース・ディクショナリがアップグレードされるたびに作成されます。デフォルトでは、このディレクトリ・オブジェクトは特権ユーザーのみが使用できます。(ユーザー`SYSTEM`は、デフォルトで`DATA_PUMP_DIR`ディレクトリへの読取りおよび書込みアクセス権を持っています。)Oracleは、Oracle Databaseのアップグレード時またはパッチの適用時に、`DATA_PUMP_DIR`ディレクトリの定義を変更することがあります。

権限のないユーザーがOracle Data Pumpエクスポートまたはインポートを実行できるようにするには、まず、データベース管理者(DBA)またはCREATE ANY DIRECTORY権限を持つユーザーがディレクトリ・オブジェクトを作成しておく必要があります。

ディレクトリの作成後、ディレクトリ・オブジェクトを作成するユーザーは、そのディレクトリに対するREAD権限またはWRITE権限を他のユーザーに付与する必要があります。たとえば、dpump_dir1という名前のディレクトリで、Oracle Databaseがユーザーhrのかわりにファイルを読取りまたは書込みできるようにするには、次のコマンドをDBAが実行する必要があります。

```
SQL> GRANT READ, WRITE ON DIRECTORY dpump_dir1 TO hr;
```

ディレクトリ・オブジェクトに対するREAD権限またはWRITE権限は、それに対応するディレクトリ内にあるファイルの読取りまたは書込みをユーザーのかわりにOracle Databaseのみが実行できることを意味します。Oracle Database以外は、適切なオペレーティング・システム権限が付与されていないかぎり、該当するファイルに直接アクセスすることはできません。同様に、Oracle Databaseには、そのディレクトリ内のファイルに対する読取りおよび書込みのオペレーティング・システム権限が必要になります。

Oracle Data Pumpエクスポートおよびインポート・ユーティリティでは、次の順序でファイルの場所が判断されます。

1. ディレクトリ・オブジェクトがファイル指定の一部として指定されている場合は、そのディレクトリ・オブジェクトで指定された位置が使用されます。(ディレクトリ・オブジェクトとファイル名は、コロンで区切る必要があります。)
2. ディレクトリ・オブジェクトがファイル指定の一部として指定されていない場合は、DIRECTORYパラメータで指定されたディレクトリ・オブジェクトが使用されます。
3. ディレクトリ・オブジェクトがファイル指定の一部として指定されていない場合、およびDIRECTORYパラメータでディレクトリ・オブジェクトが指定されていない場合は、環境変数DATA_PUMP_DIRの値が使用されます。この環境変数は、データ・ポンプ・エクスポートおよびインポート・ユーティリティを実行するクライアント・システムで、オペレーティング・システム・コマンドを使用して定義します。このクライアント・ベースの環境変数には、DBAがサーバー・システムで最初に作成するサーバー・ベースのディレクトリ・オブジェクトの名前を割り当てる必要があります。たとえば、次のSQL文では、サーバー・システムにディレクトリ・オブジェクトが作成されます。このディレクトリ・オブジェクトの名前はDUMP_FILES1で、位置は '/usr/apps/dumpfiles1'です。

   ```
   SQL> CREATE DIRECTORY DUMP_FILES1 AS '/usr/apps/dumpfiles1';
   ```

   この文の実行後、cshを使用しているUNIXベースのクライアント・システムのユーザーは、環境変数DATA_PUMP_DIRに値DUMP_FILES1を割り当てることができるようになります。コマンドラインでは、DIRECTORYパラメータは省略できます。ダンプ・ファイルemployees.dmpおよびログ・ファイルexport.logは、'/usr/apps/dumpfiles1'に書き込まれます。

   ```
   %setenv DATA_PUMP_DIR DUMP_FILES1
   %expdp hr TABLES=employees DUMPFILE=employees.dmp
   ```

4. 前述の3つの条件ではディレクトリ・オブジェクトが生成されないときに、特権ユーザーが操作している場合、Oracle Data Pumpはデフォルトのサーバー・ベースのディレクトリ・オブジェクトDATA_PUMP_DIRの値の使用を試行します。このディレクトリ・オブジェクトは、データベースの作成時またはデータベース・ディクショナリのアップグレード時に自動的に作成されます。DATA_PUMP_DIRのパス定義を確認する場合は、次のSQL問合せを使用できます。

   ```
   SQL> SELECT directory_name, directory_path FROM dba_directories
   2 WHERE directory_name='DATA_PUMP_DIR';
   ```

   特権ユーザーでない場合は、DATA_PUMP_DIRディレクトリ・オブジェクトへのアクセス権限が、DBAによって事前に付与されている必要があります。

   デフォルトのDATA_PUMP_DIRディレクトリ・オブジェクトと、同じ名前のクライアント・ベースの環境変数とを混同しない

でください。

親トピック: ダンプ・ファイル、ログ・ファイルおよびSQLファイルのデフォルトの位置

1.8.3.2 Oracle RACでのOracle Data Pumpの使用方法の理解

Oracle Real Application Clusters (Oracle RAC)環境でOracle Data Pumpを使用するには、いくつかのチェックを実行して、クラスタ・メンバー・ノードが使用可能であることを確認する必要があります。

- Oracle RAC構成でOracle Data Pumpまたは外部表を使用するには、ディレクトリ・オブジェクトのパスがクラスタ・ファイル・システム上に存在するようにする必要があります。

  ディレクトリ・オブジェクトは、Oracle Data Pumpまたは外部表プロセス(あるいはその両方)を実行できるすべてのインスタンスから参照およびアクセスできる共有物理記憶域を指している必要があります。

- デフォルトのOracle Data Pumpの動作では、Oracle RAC構成内の任意のインスタンスで子プロセスを実行できます。したがって、このようなOracle RACインスタンス上の子プロセスは、ディレクトリ・オブジェクトによって定義された場所(共有ストレージ・メディアなど)に物理的にアクセスできる必要があります。この目的で使用できる共有ストレージが構成内にない場合にも並列処理が必要であれば、CLUSTER=NOパラメータを使用して、すべての子プロセスを、Oracle Data Pumpジョブが開始されたインスタンスのみに置くことができます。

- 特定の状況下では、Oracle Data Pumpはパラレル問合せの子プロセスを使用してデータをロードまたはアンロードします。Oracle RAC環境では、これらの子プロセスの実行場所はデータ・ポンプによって制御されません。したがって、これらの子プロセスは、Oracle Data PumpジョブのCLUSTERおよびSERVICE_NAMEに指定されたインスタンスに関係なく、クラスタ内の他のクラスタ・メンバー・ノードで実行できます。パラレル問合せ操作の制御は、Oracle Data Pumpとは無関係です。パラレル問合せ子プロセスがOracle Data Pumpジョブの一部として他のインスタンスで実行される場合は、ダンプ・ファイル・セットの物理記憶域へのアクセスも必要になります。

親トピック: ダンプ・ファイル、ログ・ファイルおよびSQLファイルのデフォルトの位置

1.8.3.3 Oracle Automatic Storage Managementを使用可能にした場合のディレクトリ・オブジェクトの使用方法

Oracle Automatic Storage Management (Oracle ASM)を使用可能にしてOracle Data Pumpエクスポートまたはインポートを使用する場合は、ダンプ・ファイル用に使用するディレクトリ・オブジェクトを定義します。

Oracle ASMディスク・グループ名が,オペレーティング・システムのディレクトリ・パスのかわりに使用されるように、ダンプ・ファイルに対して使用されるディレクトリ・オブジェクトを定義する必要があります。

ログ・ファイルには、オペレーティング・システムのディレクトリ・パスを指す個別のディレクトリ・オブジェクトを使用します。

たとえば、このプロシージャを使用して、Oracle ASMダンプ・ファイルのディレクトリ・オブジェクトを作成できます。

```
SQL> CREATE or REPLACE DIRECTORY dpump_dir as '+DATAFILES/';
```

このディレクトリ・オブジェクトを作成した後、ログ・ファイル用の個別のディレクトリ・オブジェクトを作成します。

```
SQL> CREATE or REPLACE DIRECTORY dpump_log as '/homedir/user1/';
```

ユーザーhrに、これらのディレクトリ・オブジェクトに対するアクセス権を付与する場合は、このユーザーに必要な権限を割り当てます。

```
SQL> GRANT READ, WRITE ON DIRECTORY dpump_dir TO hr;
SQL> GRANT READ, WRITE ON DIRECTORY dpump_log TO hr;
```

最後に、次のデータ・ポンプ・エクスポート・コマンドを使用できます。

```
> expdp hr DIRECTORY=dpump_dir DUMPFILE=hr.dmp LOGFILE=dpump_log:hr.log
```

コマンドを実行する前に、パスワードの入力を求められます。

> ノート:
>
> データ・ポンプのダンプ・ファイルを ASM とディスク・ディレクトリの間で単にコピーする場合には、DBMS_FILE_TRANSFER PL/SQL パッケージを使用できます。

関連トピック

- Oracle Database SQL言語リファレンス
- Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス

親トピック: ダンプ・ファイル、ログ・ファイルおよびSQLファイルのデフォルトの位置

1.8.3.4 DATA_PUMP_DIRディレクトリ・オブジェクトおよびプラガブル・データベース

CDB内の各PDBに対する一意のパスとして、デフォルトのOracle Data Pumpディレクトリ・オブジェクトDATA_PUMP_DIRを定義します。

Oracle Database 12c リリース2 (12.2)からは、マルチテナント・コンテナ・データベース(CDB)環境において、相対パスにCREATE PLUGGABLE DATABASEのPATH_PREFIX句が定義されているどうかに関係なく、デフォルトのOracle Data Pumpディレクトリ・オブジェクトDATA_PUMP_DIRがCDB内の各PDBの一意のパスとして定義されます。

親トピック: ダンプ・ファイル、ログ・ファイルおよびSQLファイルのデフォルトの位置

1.8.4 Oracle Data Pump Exportでの代替変数の使用

Oracle Data Pump Export操作時に複数のファイルを指定する場合は、ファイル名に含まれる置換変数を指定したDUMPFILEパラメータを使用します。

特定のファイル名をリストするかわりに(またはリストすることに加えて)、ファイル名に置換変数を使用するときには、置換変数が含まれているファイル名を**ダンプ・ファイル・テンプレート**と呼びます。

> ノート:
>
> 次の各例では、置換変数%U を使用して Oracle Data Pump での置換変数の使用方法を説明します。その他の使用可能な置換変数については、インポートまたはエクスポート・ユーティリティの DUMPFILE パラメータ・リファレンスのトピックで確認してください。

ダンプ・ファイル・テンプレートを使用すると、新しいダンプ・ファイルが必要に応じて作成されます。たとえば、置換変数%Uを使用している場合は、新しいダンプ・ファイルの作成が必要になったときに、最初の%Uには01が使用され、その後は順に02、03が使用されるようになります。PARALLELパラメータの現行の設定で指定されたすべてのプロセスをアクティブにできるように、十分な数のダンプ・ファイルが作成されます。ダンプ・ファイルのいずれかがFILESIZEパラメータで指定した最大サイズに達したために満杯になると、そのダンプ・ファイルはクローズされ、そのファイルを引き継ぐために新しく生成された名前を持つ新しいダンプ・ファイルが作成されます。

複数のダンプ・ファイル・テンプレートが用意されている場合は、これらのテンプレートは、ラウンドロビン法によるダンプ・ファイルの生

成に使用されます。たとえば、並列度が6のジョブにexpa%U、expb%UおよびexpC%Uのすべてが指定されている場合、最初に作成されるダンプ・ファイルはexpa01.dmp、expb01.dmp、expc01.dmp、expa02.dmp、expb02.dmpおよびexpc02.dmpになります。

インポート操作とSQLFILE操作の場合は、ダンプ・ファイルの仕様expa%U、expb%UおよびexpC%Uを指定すると、その操作では最初にダンプ・ファイルexpa01.dmp、expb01.dmpおよびexpc01.dmpを開こうとします。データ・ポンプ制御エクスポート表は、複数のダンプ・ファイルにまたがることができます。そのため、データ・ポンプ制御表のすべての部分が検出されるまで、置換変数を増分し、新しいファイル名(expa02.dmp、expb02.dmpおよびexpc02.dmpなど)を検索することで、ダンプ・ファイルを開き続けます。ダンプ・ファイルが存在しない場合は、エラーとなったダンプ・ファイル指定の置換変数の増加が中止されます。たとえば、expb01.dmpとexpb02.dmpが見つかっているときにexpb03.dmpが見つからない場合、それ以降expb%Uの仕様を使用したファイルは検索されなくなります。Data Pump制御表全体が検出されると、ダンプ・ファイル・セット内のすべてのダンプ・ファイルが配置されたかどうかを調べるためにその表が使用されます。

関連トピック

- DUMPFILE
- DUMPFILE

親トピック: Oracle Data Pumpによるファイル割当て

1.9 異なるOracle Databaseリリース間のエクスポートとインポート

Oracle Data Pumpを使用すると、異なるリリースのデータベース・ソフトウェア間でOracle Databaseのすべてまたは一部を移行できます。

通常、データベース・リリース間で移行する場合は、Oracle Data PumpエクスポートのVERSIONパラメータを使用します。VERSIONを使用すると、指定したバージョンと互換性のあるOracle Data Pumpのダンプ・ファイル・セットが生成されます。

VERSIONのデフォルト値はCOMPATIBLEです。この値は、エクスポートされたデータベース・オブジェクト定義がCOMPATIBLE初期化パラメータで指定されたリリースと互換性があることを示します。

アップグレードの際に、Oracle Data Pumpベースの移行のターゲット・リリースがソースより新しくなる場合は、通常、VERSIONパラメータの指定は必要ありません。ターゲット・リリースがソースより新しい場合、ソース・データベース内のすべてのオブジェクトには、これらの新しいターゲット・リリースとの互換性があります。ただし、Oracle Database 12cリリース1 (12.1.0.1)以降にインポートするための準備として、Oracle Database 11g (リリース11.2.0.3以降)全体をエクスポートする場合を除きます。この場合は、FULL=YESとともにVERSION=12を明示的に指定して、Oracle Databaseの内部コンポーネント・メタデータの完全セットを含めます。

ダウングレードの際に、Oracle Data Pumpベースの移行のターゲット・リリースがソースより古くなる場合は、VERSIONパラメータ値をターゲットと同じバージョンに設定します。ターゲット・リリース・バージョンがソース・システムのCOMPATIBLE初期化パラメータの値と同じである場合は例外です。この場合、VERSIONを指定する必要はありません。ただし、VERSIONパラメータが明示的に指定されていない場合、通常、Oracle Data Pumpインポートは現在のリリースよりも新しいOracle Databaseのリリースで作成されたダンプ・ファイル・セットを読み取ることができません。

異なるデータベース・リリース間のエクスポートとインポートを行う場合、次の点を考慮する必要があります。

- Oracle Data Pumpエクスポートで、現在のデータベース・バージョンより古いデータベース・バージョンを指定すると、その古いバージョンのデータベースにインポート可能なダンプ・ファイル・セットが作成されます。たとえば、Oracle Database 12cリリース1 (12.1.0.2)を実行していて、エクスポート時にVERSION=11.2を指定すると、Oracle Database 11g (リリース11.2)にインポートできるダンプ・ファイル・セットが作成されます。

ノート:

- Oracle Database 12c リリース 1 (12.1.0.2)以降のみで有効なデータベース権限(たとえば、表、ビュー、マテリアライズド・ビューおよびシノニムに対する READ 権限)は Oracle Database 12c リリース 1 (12.1.0.1)以降にインポートできません。これを実行しようとすると、エラーとして報告され、インポート操作が続行されます。
- Oracle Database 12c リリース 2 (12.2.0.1)よりも前のリリースにエクスポートするときに、Oracle Data Pump が 30 バイトを超えるオブジェクト名を除外することはありません。該当するオブジェクトはエクスポートされます。インポート時には、30 バイトを超える名前のオブジェクトを作成しようとするとエラーが返されます。

- 現在のOracle Databaseのリリースより古いOracle Databaseのリリースを指定すると、特定の機能とデータ型が使用できなくなることがあります。たとえば、`VERSION=10.1`を指定したときに、そのジョブに対してデータ圧縮も指定するとエラーが発生します。これは、Oracle Database 10gリリース1 (10.1)では圧縮がサポートされていないためです。別の例として、ソースOracle Databaseリリースのユーザー定義またはOracle提供の型がターゲット・データベースの型よりも新しいバージョンの場合、その型はターゲット・データベースの型のいずれのバージョンとも一致しないためロードされません。
- Oracle Data Pumpインポートは、常に、古いOracle Databaseリリースのデータベースで作成されたOracle Data Pumpのダンプ・ファイル・セットを読み込むことができます。
- ネットワーク・リンクを通じたOracle Data Pump操作の場合は、ソースとターゲットのOracle Databaseリリースのバージョンの差異が2以下であることが必要です。

  たとえば、一方のデータベースがOracle Database 12cの場合、もう一方のOracle Databaseのリリースは12c、11gまたは10gであることが必要です。Oracle Data Pumpでは、メジャー・バージョン番号(10g、11g、12cなど)のみがチェックされます。Oracle Databaseの具体的なリリース番号(12.2、12.1、11.1、11.2、10.1、10.2など)はチェックされません。
- 表統計が含まれているOracle Database 11gダンプ・ファイルをOracle Database 12cリリース1 (12.1)以降にインポートすると、Oracle ORA-39346エラーが発生する場合があります。このエラーは、Oracle Database11gのダンプ・ファイルに、メタデータとして表の統計が含まれているために発生します。Oracle Database12cリリース1 (12.1)以降のリリースでは、表の統計が表データとして提示される必要があります。インポート操作中のこのエラーは無視しても問題ありません。インポート操作が完了した後、表の統計を再収集します。

関連トピック

- VERSION
- VERSION

関連項目:

- READおよびREAD ANY TABLE権限の詳細は、『Oracle Databaseセキュリティ・ガイド』を参照してください

親トピック: Oracle Data Pumpの概要

1.10 Oracle Data Pumpを使用したSecureFilesラージ・オブジェクトのエクスポートの管理

SecureFilesラージ・オブジェクト(LOB)のエクスポートは、コンテンツ・タイプやVERSIONパラメータなどの変数による影響を受けます。

LOBは、大量のデータを保持するために設計された一連のデータ型です。Oracle Data Pumpエクスポートを使用してSecureFiles LOBをエクスポートすると、そのエクスポートの動作は、いくつかの条件に応じて異なります(エクスポートのVERSIONパラメータの値、コンテンツ・タイプ(ContentType)が存在するかどうか、LOBがアーカイブされデータがキャッシュされるかどうかなど)。

次の例に、これらの変数の様々な組合せを示します。

- 表に、ContentTypeが存在するSecureFiles LOBが含まれていて、エクスポート・ユーティリティのVERSIONパラメータが11.2.0.0.0未満の値に設定されていると、ContentTypeはエクスポートされません。
- 表に、ContentTypeが存在するSecureFiles LOBが含まれていて、エクスポート・ユーティリティのVERSIONパラメータが11.2.0.0.0以降の値に設定されていると、ContentTypeがエクスポートされ、その後のインポートでリストアされます。
- 表に、現在アーカイブされているSecureFiles LOBが含まれていて、データがキャッシュされ、エクスポート・ユーティリティのVERSIONパラメータが11.2.0.0.0未満の値に設定されていると、SecureFiles LOBデータがエクスポートされて、アーカイブ・メタデータが削除されます。この例では、VERSIONが11.1以降に設定されていると、SecureFiles LOBは標準のSecureFiles LOBになります。ただし、VERSIONが11.1未満の値に設定されていると、SecureFiles LOBはBasicFiles LOBになります。
- 表に、現在アーカイブされているSecureFiles LOBが含まれていて、データがキャッシュされず、エクスポート・ユーティリティのVERSIONパラメータが11.2.0.0.0未満の値に設定されていると、ORA-45001エラーが返されます。
- 表に、現在アーカイブされているSecureFiles LOBが含まれていて、データがキャッシュされ、エクスポート・ユーティリティのVERSIONパラメータが11.2.0.0.0以上の値に設定されていると、キャッシュされたデータとアーカイブ・メタデータの両方がエクスポートされます。

SecureFiles LOBの詳細は、『*Oracle Database SecureFiles*およびラージ・オブジェクト開発者ガイド』を参照してください。

関連トピック

- Oracle Database SecureFiles and Large Objects開発者ガイド

親トピック: Oracle Data Pumpの概要

1.11 Oracle Data Pumpプロセス終了コード

Oracle Data Pump ExportおよびOracle Data Pump Import操作の状態をチェックするには、ログ・ファイルでプロセス終了コードを確認します。

Oracle Data Pumpでは、完了してすぐにエクスポートおよびインポート操作の結果が提供されます。Oracle Data Pumpでは、結果をログ・ファイルに記録するだけでなく、結果をプロセス終了コードでレポートすることもできます。Oracle Data Pumpの終了コードを使用して、コマンドラインやスクリプトからのOracle Data Pumpジョブの結果を確認します。

表1-1 Oracle Data Pump終了コード

終了コード	意味
EX_SUCC 0	エクスポートおよびインポート・ジョブが、正常に完了しました。エラーは出力デバイスに表示されず、ログ・ファイル(ある場合)にも記録されません。
EX_SUCC_ERR 5	エクスポートまたはインポート・ジョブは正常に完了しましたが、ジョブ実行中にエラーが検出されました。エラーは出力デバイスに表示され、ログ・ファイル(ある場合)に記録されます。
EX_FAIL 1	エクスポートまたはインポート・ジョブで、次を含む 1 つ以上の致命的エラーが検出されました。 ● コマンドラインまたはコマンド構文のエラー ● エクスポートまたはインポートがリカバリできない Oracle Database のエラー ● オペレーティング・システムのエラー(malloc など) ● ジョブの開始を妨げる無効なパラメータ値(たとえば DIRECTORY パラメータで指定された無効なディレクトリ・オブジェクト) 致命的エラーは出力デバイスに表示されますが、ログ・ファイルには記録されない場合があります。これがログ・ファイルに記録されるかどうかは、次のようないくつかの要因によります。 ● ログ・ファイルをジョブの開始時に指定したか。 ● ログ・ファイルを開くのに十分なほどジョブの処理が進行していたか。

親トピック: Oracle Data Pumpの概要

1.12統合監査を使用してOracle Data Pumpジョブを監視する方法

特定のユーザー・データベース・アクションを監視および記録するには、統合監査を使用してデータ・ポンプ・ジョブの監査を実行します。

特定のユーザー・データベース・アクションを監視および記録するには、Oracle Data Pumpジョブの監査を実行できます。Oracle Data Pumpでは、すべての監査レコードが1つの場所に一元化される統合監査を使用します。統合監査を設定するには、統合監査ポリシーを作成するか、既存の監査ポリシーを変更します。監査ポリシーは、データベースにおけるユーザー動作の特定の部分を監査できる監査設定の名前付きグループです。

ポリシーを作成するには、SQL CREATE AUDIT POLICY文を使用します。監査ポリシーを作成したら、AUDIT SQL文を使用してポリシーを有効にします。

ポリシーを無効にするには、NOAUDIT SQL文を使用します。

関連項目:

- SQL CREATE AUDIT POLICY、ALTER AUDIT POLICY、AUDITおよびNOAUDIT文の詳細は、Oracle Database SQL言語リファレンスを参照してください
- Oracle Databaseで監査を使用する方法の詳細は、Oracle Databaseセキュリティ・ガイドを参照してください

親トピック: Oracle Data Pumpの概要

1.13 Oracle Data Pump操作に対する暗号化されたデータのセキュリティ警告

Oracle Data Pumpは、暗号化されたデータが暗号化されていないデータとしてエクスポートされると警告を表示します。

Oracle Data Pumpのエクスポート操作中に、エクスポート・ジョブの開始時に指定された暗号化データがOracle Data Pumpで検出されると、ORA-39173警告が表示されます。このORA-39173 ("ORA-39173: 暗号化データがダンプ・ファイル・セットに暗号化されずに格納されました。")は、監査レコードにも書き込まれます。統合監査証跡のDP_WARNINGS1列を確認すると、エクスポート操作中に発生したORA-39173エラーを表示できます。監査情報は、次のSQL文を実行することで取得します。

```
SELECT DP_WARNINGS1 FROM UNIFIED_AUDIT_TRAIL WHERE ACTION_NAME = 'EXPORT' ORDER BY 1;
```

親トピック: Oracle Data Pumpの概要

1.14 Oracle Data Pumpによるタイムスタンプ・データの処理方法

タイムスタンプ・データ型のTIMESTAMP WITH TIMEZONEおよびTIMESTAMP WITH LOCAL TIMEZONEを含むエクスポートおよびインポート・ジョブの正常な完了に影響する可能性のある要因について学習します。

> ノート:
>
> この項の情報は、Oracle Database 12c 以上で実行されている Oracle Data Pump にのみ適用されます。

- TIMESTAMP WITH TIMEZONEの制限事項
  TIMESTAMP WITH TIME ZONEデータを含むエクスポートおよびインポート・ジョブは制限されます。
- TIMESTAMP WITH LOCAL TIME ZONE制限事項
  トランスポータブル・モードを使用した表の移動は制限されます。

親トピック: Oracle Data Pumpの概要

1.14.1 TIMESTAMP WITH TIMEZONEの制限事項

TIMESTAMP WITH TIME ZONEデータを含むエクスポートおよびインポート・ジョブは制限されます。

- TIMESTAMP WITH TIME ZONEの制限事項の理解
  TIMESTAMP WITH TIME ZONEデータが含まれるエクスポート・ジョブおよびインポート・ジョブを実行するには、タイムゾーン・ファイル・データとOracle Databaseリリースに関する情報を理解する必要があります。
- TIMESTAMP WITH TIME ZONEデータのOracle Data Pumpサポート
  Oracle Data Pumpでは、様々なエクスポート・モードおよびインポート・モードでTIMESTAMP WITH TIME ZONEデータがサポートされます。

- ソースおよびターゲットのタイムゾーン・ファイルのバージョン
  ジョブが正常に完了するかどうかは、ソースとターゲットのタイムゾーン・ファイルのバージョンが一致しているかどうかによります。

親トピック: Oracle Data Pumpによるタイムスタンプ・データの処理方法

1.14.1.1 TIMESTAMP WITH TIME ZONEの制限事項の理解

`TIMESTAMP WITH TIME ZONE`データが含まれるエクスポート・ジョブおよびインポート・ジョブを実行するには、タイムゾーン・ファイル・データとOracle Databaseリリースに関する情報を理解する必要があります。

ダンプ・ファイルをインポートする場合、宛先(ターゲット)データベースのタイム・ゾーン・バージョンは、エクスポートの取得元のソース・データベースのタイム・ゾーン・バージョンと同じバージョンか、それより新しい(高い)バージョンである必要があります。ジョブが正常に完了するかどうかは、次の要因に依存します。

- ソース・データベースとターゲット・データベースのOracle Databaseタイムゾーン・ファイルのバージョン。
- エクスポート/インポート・モードと、使用しているデータ・ポンプのバージョンで`TIMESTAMP WITH TIME ZONE`データがサポートされるかどうか。(Oracle Data Pump 11.2.0.1以上のリリースでは、`TIMESTAMP WITH TIME ZONE`データがサポートされます。)

データベースのタイムゾーン・ファイルのバージョンを識別するには、次のSQL文を実行します。

```
SQL> SELECT VERSION FROM V$TIMEZONE_FILE;
```

関連トピック

- タイム・ゾーン・ファイルの選択

親トピック: TIMESTAMP WITH TIMEZONEの制限事項

1.14.1.2 Oracle Data PumpによるTIMESTAMP WITH TIME ZONEデータのサポート

Oracle Data Pumpは、様々なエクスポート・モードおよびインポート・モードで`TIMESTAMP WITH TIME ZONE`データをサポートしています。

Oracle Data Pumpでは、Oracle Databaseタイム・ゾーン・ファイルのバージョンがソース・データベースとターゲット・データベースで異なる場合、様々なエクスポート・モードおよびインポート・モードで`TIMESTAMP WITH TIME ZONE`データがサポートされます。サポートされているモードには、非トランスポータブル・モード、トランスポータブル表領域、トランスポータブル表モードおよび全体トランスポータブル・モードがあります。

非トランスポータブル・モード

- `TIMESTAMP WITH TIME ZONE`データをサポートするデータ・ポンプのバージョン(11.2.0.1以上)でダンプ・ファイルを作成すると、エクスポート・システムのタイムゾーン・ファイルのバージョンがダンプ・ファイルに記録されます。Oracle Data Pumpでは、この情報を使用してデータ変換が必要であるかどうかを決定します。ターゲット・データベースでソースのタイムゾーンのバージョンについて認識しているが、実際には新しいバージョンを使用している場合、データは新しいバージョンに変換されます。`TIMESTAMP WITH TIME ZONE`データはダウングレードできないため、ソースで使用しているものより古いバージョンのタイムゾーン・ファイルを使用しているターゲットにインポートを試みると、インポートは失敗します。
- ダンプ・ファイルがOracle Database 11gリリース2 (11.2.0.1)より前のOracle Data Pumpバージョンで作成された場合、`TIMESTAMP WITH TIME ZONE`データはサポートされません。変換は行われず、破損が発生する可能性があります。

トランスポータブル表領域モードおよびトランスポータブル表モード

- トランスポータブル表領域モードおよびトランスポータブル表モードでは、ソースとターゲットのタイムゾーン・ファイルのバージョンが異なる場合、TIMESTAMP WITH TIME ZONE列を含む表は作成されません。ジョブの開始時に、ソースとターゲットのデータベース・タイムゾーン・ファイルのバージョンを示す警告が表示されます。また、作成されなかった表ごとにメッセージが表示されます。このことは、ダンプ・ファイルを作成するために使用したOracle Data Pumpバージョンで TIMESTAMP WITH TIME ZONEデータがサポートされる場合でも同様です。(リリース11.2.0.1以上では TIMESTAMP WITH TIMEZONEデータがサポートされます。)
- ソースがOracle Database 11gリリース2 (11.2.0.1)よりも前の場合、トランスポータブル・セットがTIMESTAMP WITH TIME ZONE列を使用するかどうかに関係なく、タイムゾーン・ファイルのバージョンはすべてのトランスポータブル・ジョブのソースおよびターゲット・データベースで同じである必要があります。

全体トランスポータブル・モード

全体トランスポータブル・エクスポートおよびインポートがサポートされるのは、ソース・データベースがOracle Database 11gリリース2 (11.2.0.3)以降であり、ターゲットが少なくともOracle Database 12cリリース1 (12.1)以降の場合です。

Oracle Data Pump 11.2.0.1以上では、TIMESTAMP WITH TIME ZONEデータがサポートされます。したがって、全体トランスポータブル操作では、TIMESTAMP WITH TIME ZONE列を含む表が作成されます。ソース・データベースとターゲット・データベースのタイムゾーン・ファイルのバージョンが異なる場合、ソースのTIMESTAMP WITH TIME ZONE列は、ターゲットのタイムゾーン・ファイルのバージョンに変換されます。

関連トピック

- トランスポータブル表領域に関する制限事項
- 全体モード
- 全体インポート・モード

親トピック: TIMESTAMP WITH TIMEZONEの制限事項

1.14.1.3 ソースおよびターゲットのタイムゾーン・ファイルのバージョン

ジョブが正常に完了するかどうかは、ソースとターゲットのタイムゾーン・ファイルのバージョンが一致しているかどうかによります。

- Oracle Databaseのタイムゾーン・ファイルのバージョンがソース・データベースとターゲット・データベースで同じである場合、TIMESTAMP WITH TIME ZONEデータの変換は不要です。エクスポート/インポート・ジョブは正常に完了します。

  この例外は、データ・ポンプの11.2.0.1より前のリリースを使用して実行されるトランスポータブル表領域またはトランスポータブル表のエクスポートです。この場合、TIMESTAMP WITH TIME ZONE列を含むダンプ・ファイルの表は、タイムゾーン・ファイルのバージョンがソースとターゲットで同じであっても、インポート時に作成されません。
- ターゲット・データベースでソースのタイムゾーン・ファイルのバージョンを使用できない場合、ジョブは失敗します。ソースのタイムゾーン・ファイルが新しいバージョンに更新されていても、ターゲットでは更新されていないことがあるため、ソースのタイムゾーン・ファイルのバージョンはターゲットで使用できない可能性があります。たとえば、タイムゾーン・ファイルのバージョンが17のOracle Database 11gリリース2 (11.2.0.2)でエクスポートを実行し、使用できるタイムゾーン・ファイルのバージョンが16のみの11.2.0.2でインポートを実行すると、ジョブは失敗します。

親トピック: TIMESTAMP WITH TIMEZONEの制限事項

1.14.2 TIMESTAMP WITH LOCAL TIME ZONEの制限事項

トランスポータブル・モードを使用した表の移動は制限されます。

表がトランスポータブル・モード(トランスポータブル表、トランスポータブル表領域または全体トランスポータブル)を使用して移動され、次の条件が存在する場合、警告が発行されて表は作成されません。

- ソース・データベースとターゲット・データベースのデータベース・タイムゾーンが異なる場合
- 表に`TIMESTAMP WITH LOCAL TIME ZONE`データ型が含まれる場合

これらの条件が原因で作成されなかった表を正常に移動するには、トランスポータブルではないエクスポートおよびインポート・モードを使用してください。

親トピック: Oracle Data Pumpによるタイムスタンプ・データの処理方法

1.15 文字セットおよびグローバリゼーション・サポートに関する考慮点

ユーザー・データの文字セット変換とデータ定義言語(DDL)を使用したOracle Data Pumpエクスポートおよびインポートのグローバリゼーション・サポートについて学習します。

- データ定義言語(DDL)
  エクスポート・ユーティリティでは、エクスポート・システムのデータベース文字セットを使用してダンプ・ファイルが書き込まれます。
- シングルバイト文字セットとエクスポートおよびインポート
  エクスポート・データベースとインポート・データベースで同じ文字セットを使用していることを確認してください。
- マルチバイト文字セットとエクスポートおよびインポート
  Oracle Data Pumpのエクスポートおよびインポート中の文字セットの変換は、インポートするOracle Databaseの文字セットに依存します。

親トピック: Oracle Data Pumpの概要

1.15.1 データ定義言語（DDL)

エクスポート・ユーティリティは、エクスポート・システムのデータベース文字セットを使用してダンプ・ファイルを書き込みます。

インポート・システムのデータベース文字セットがエクスポート・システムのデータベース文字セットと異なる場合にのみ、ダンプ・ファイルのインポート時にDDLのために文字セット変換が必要です。

文字セット変換によるデータの損失を最小限にするには、インポートのデータベース文字セットがエクスポートのデータベース文字セットのスーパーセットであることを確認します。

親トピック: 文字セットおよびグローバリゼーション・サポートに関する考慮点

1.15.2 シングルバイト文字セットとエクスポートおよびインポート

エクスポート・データベースとインポート・データベースで同じ文字セットを使用していることを確認してください。

インポートが実行されるシステムで7ビット文字セットを使用し、8ビット文字セットのダンプ・ファイルをインポートすると、一部の8ビット・キャラクタは同等の7ビットに変換されることがあります。これが発生した場合の目印として、アクセント記号が付いている文字からアクセントが消去されます。

この不要な変換を回避するため、エクスポート・データベースとインポート・データベースで同じ文字セットを使用していることを確

認してください。

親トピック: 文字セットおよびグローバリゼーション・サポートに関する考慮点

1.15.3 マルチバイト文字セットとエクスポートおよびインポート

Oracle Data Pumpのエクスポートおよびインポート中の文字セットの変換は、インポートするOracle Databaseの文字セットに依存します。

インポート・データベースの文字セットに同等の文字がないエクスポート・ファイル中の文字は、変換時にデフォルトの文字に置換されます。インポート・データベースの文字セットは、デフォルトの文字を定義します。

DDLの変換中にインポート・システムで置換文字を使用する必要がある場合、警告メッセージが表示され、システムは変換されたDDLをロードしようと試みます。

ユーザー・データの変換中にインポート・システムで置換文字を使用する必要がある場合、デフォルトの動作として、変換されたデータがロードされます。ただし、置換文字を使用して変換されたユーザー・データの行を拒否するようにインポート・システムに指示することが可能です。詳細は、インポートのDATA OPTIONSパラメータを参照してください。

100%完全に変換されるためには、インポート・データベースの文字セットは、エクスポート・ファイルを生成するために使用される文字セットのスーパーセットであるか、それと同等である必要があります。

注意:

エクスポート・システムのデータベース文字セットがインポート・システムのものと異なる場合、ジョブの開始時に、データベース文字セットの内容を示す情報メッセージがインポート・システムに表示されます。

インポート・データベースの文字セットがエクスポート・ファイルを生成するために使用された文字セットのスーパーセットではない場合、文字セット変換によってデータ損失が発生する可能性があることを示す警告がインポート・システムに表示されます。

関連トピック

- DATA_OPTIONS

親トピック: 文字セットおよびグローバリゼーション・サポートに関する考慮点

1.16 データ・バインドされた照合に関連するOracle Data Pumpの動作

Oracle Data Pumpでは、データ・バインドされた照合(DBC)がサポートされています。

Oracle Data Pumpエクスポートでは、常に、作成されたダンプ・ファイルにすべての使用可能な照合メタデータが含められます。次のものが含まれます。

- エクスポートされたユーザーのスキーマの現在のデフォルト照合
- エクスポートされた表、ビュー、マテリアライズド・ビューおよびPL/SQL単位(ユーザー定義の型を含む)の現在のデフォルト照合
- 表およびクラスタのすべての文字データ型列の宣言された照合

Oracle Database 12cリリース2 (12.2)データベースからエクスポートされたダンプ・ファイルをインポートするときのデータ・ポンプ・インポートの動作は、インポート時のOracle Data PumpのVERSIONパラメータの有効な値と、ターゲット・データベースで

データ・バインドされた照合(DBC)機能が有効になっているかどうかの両方によって異なります。VERSIONパラメータの有効な値は、パラメータをどのように指定するかによって決まります。このパラメータは、次のように指定できます。

- VERSION=nは、有効な値が特定のバージョン番号nであることを意味します。たとえば、VERSION=19
- VERSION=LATEST: 有効な値は、現在実行中のデータベース・バージョンです。
- VERSION=COMPATIBLE: 有効な値は、データベース初期化パラメータCOMPATIBLEの値と同じです。このことは、VERSIONの値が指定されていない場合にも該当します。

データベースでDBC機能を有効にするには、初期化パラメータCOMPATIBLEを12.2以降に設定し、また初期化パラメータMAX_STRING_SIZEをEXTENDEDに設定する必要があります。

Oracle Data PumpインポートのVERSIONパラメータの有効な値が12.2であり、ターゲット・データベースでDBCが有効になっている場合、Oracle Data Pumpインポートでは、ダンプ・ファイルからの照合メタデータを参照する照合句を含むDDL文が生成されます。エクスポートされたオブジェクトは、ソース・データベースにあった元の照合メタデータを使用して作成されます。

DBCが無効になっている場合、またはOracle Data PumpインポートのVERSIONパラメータが12.2未満の値に設定されている場合、照合構文は生成されません。

親トピック: Oracle Data Pumpの概要

2 Oracle Data Pumpエクスポート

Oracle Data Pumpエクスポート・ユーティリティは、ダンプ・ファイル・セットと呼ばれる一連のオペレーティング・システム・ファイルにデータとメタデータをアンロードするために使用します。

- Oracle Data Pumpエクスポートとは
  Oracle Data Pumpエクスポートは、**ダンプ・ファイル・セット**と呼ばれる一連のオペレーティング・システム・ファイルにデータとメタデータをアンロードするためのユーティリティです。
- Oracle Data Pumpエクスポートの起動
  Oracle Data Pumpエクスポート・ユーティリティは、expdpコマンドを使用して起動します。
- エクスポート操作中のフィルタ処理
  Oracle Data Pump Exportは、データとメタデータのフィルタ処理機能を提供します。この機能により、エクスポートする情報の種類を効果的に制限できます。
- データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ
  エクスポート(expdp)用のOracle Data Pumpパラメータを使用して、データ・エクスポートを管理します。
- データ・ポンプ・エクスポートの対話方式コマンド・モードで使用可能なコマンド
  データ・ポンプ・エクスポートを対話方式コマンド・モードで使用するときに使用可能なコマンド・オプションを確認します。
- Oracle Data Pumpエクスポートの使用例
  次の一般的なシナリオ例を使用して、データの移動にOracle Data Pumpエクスポートを使用する方法について学習します。
- Oracle Data Pump Exportの構文図
  構文図を使用すると、Oracle Data Pump Exportでの有効なSQL構文を理解できます。

親トピック: Oracle Data Pump

2.1 Oracle Data Pumpエクスポートとは

Oracle Data Pumpエクスポートは、**ダンプ・ファイル・セット**と呼ばれるオペレーティング・システムのファイル・セットにデータとメタデータをアンロードするためのユーティリティです。

ダンプ・ファイル・セットは、Oracle Data Pumpインポート・ユーティリティを使用してのみインポートできます。ダンプ・ファイル・セットは、同じシステムでインポートすることも、別のシステムにインポートしてロードすることもできます。

ダンプ・ファイル・セットは、表データ、データベース・オブジェクトのメタデータ、制御情報を含む1つ以上のディスク・ファイルで構成されています。これらのファイルは独自のバイナリ形式で書き込まれています。インポート操作時に、Oracle Data Pumpインポート・ユーティリティは、このファイルを使用してダンプ・ファイル・セットの各データベース・オブジェクトの場所を特定します。

ダンプ・ファイルは、クライアントではなくサーバーによって書き込まれるため、ファイルが書き込まれるサーバーの場所を定義するディレクトリ・オブジェクトを作成する必要があります。

Oracle Data Pumpエクスポートでは、エクスポート・モードの決定に従って、ジョブがデータとメタデータのサブセットを移動するように指定できます。このサブセットの選択には、Oracle Data Pumpエクスポートのパラメータで指定するデータ・フィルタとメタデータ・フィルタを使用します。

ノート:

一部のシステム・スキーマは、ユーザー・スキーマではないため、エクスポートできません。システム・スキーマには、

Oracle が管理するデータおよびメタデータが含まれています。エクスポートされないスキーマの例としては、SYS、ORDSYS、MDSYS などがあります。また、インポート時に CREATE INDEX によって再作成されるため、セカンダリ・オブジェクトもエクスポートされません。

関連トピック

- ダンプ・ファイル、ログ・ファイルおよびSQLファイルのデフォルトの位置の理解
- エクスポート操作中のフィルタ処理
- My Oracle Supportノート1398388.1
- Oracle Data Pumpエクスポートの使用例

親トピック: Oracle Data Pumpエクスポート

2.2 Oracle Data Pumpエクスポートの起動

Oracle Data Pumpエクスポート・ユーティリティは、expdpコマンドを使用して起動します。

Oracle Data Pumpエクスポート操作の特性は、エクスポート・ユーティリティのパラメータの指定内容によって決まります。こうしたパラメータは、コマンドラインまたは構成ファイルのいずれかで指定できます。

注意:

エクスポート・ユーティリティは、Oracle サポート・サービスから要求された場合以外、SYSDBA として起動しないでください。SYSDBA は内部的に使用され、一般ユーザーとは異なる特別な機能を持ちます。

- Oracle Data Pumpエクスポートのインタフェース
  Oracle Data Pumpエクスポートは、コマンドライン、パラメータ・ファイルまたは対話方式コマンド・モードを使用して実行できます。
- Oracle Data Pump Exportモード
  エクスポートには、Oracle Databaseデータの様々な部分をアンロードするために様々なモードが用意されています。
- Oracle Data Pumpエクスポートのネットワークに関する考慮事項
  Oracle Data Pumpエクスポート・ユーティリティexpdpがOracle*Netまたはネット・サービス名を使用して接続文字列に含まれる接続識別子でインスタンスを識別する方法と、NETWORK_LINKパラメータを使用したエクスポート操作との相違点について説明します。

親トピック: Oracle Data Pumpエクスポート

2.2.1 Oracle Data Pumpエクスポートのインタフェース

Oracle Data Pumpエクスポートとは、コマンドライン、パラメータ・ファイルまたは対話方式コマンド・モードを使用して対話できます。

次の3つのオプションから選択します。

- コマンドライン・インタフェース: ほとんどのエクスポート・パラメータを直接コマンドラインで指定できます。
- パラメータ・ファイル・インタフェース: パラメータ・ファイルでコマンドラインのパラメータを指定できます。パラメータ・ファイルはネストできないため、PARFILEパラメータのみが例外となります。パラメータを使用するときに、その値に引用符が必要な場合は、パラメータ・ファイルの使用をお薦めします。

- 対話方式コマンド・インタフェース: 端末へのロギングを中止してエクスポート・ユーティリティのプロンプトを表示します。このプロンプトで、対話方式コマンド・モード固有のコマンドも含めて、様々なコマンドを入力できます。このモードは、コマンドライン・インタフェースまたはパラメータ・ファイル・インタフェースで開始されたエクスポート操作中に[Ctrl]キーを押しながら[C]キーを押すと使用可能になります。対話方式コマンド・モードは、実行中のジョブまたは停止されたジョブに接続した場合も使用可能になります。

関連トピック

- データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ
- データ・ポンプ・エクスポートの対話方式コマンド・モードで使用可能なコマンド

親トピック: Oracle Data Pumpエクスポートの起動

2.2.2 Oracle Data Pump Exportモード

エクスポート・ユーティリティには、Oracle Databaseデータの様々な部分をアンロードするための様々なモードがあります。

エクスポートのモードは、コマンドラインで適切なパラメータを使用して指定します。

> ノート:
>
> Oracle が管理するいくつかの Oracle Database のシステム・スキーマは、ユーザー・スキーマではないためにエクスポートできません。それらのスキーマには、Oracle が管理するデータおよびメタデータが含まれています。エクスポートされないシステム・スキーマの例としては、SYS、ORDSYS、MDSYS などがあります。

- 全体モード
  データ・ポンプを使用すると、FULLパラメータを使用して全データベースのエクスポートを実行できます。
- スキーマ・モード
  データ・ポンプでスキーマ・エクスポートを指定するには、SCHEMASパラメータを使用します。スキーマ・エクスポートは、デフォルトのエクスポート・モードです。
- 表モード
  データ・ポンプを使用して表モード・エクスポートを実行するには、TABLESパラメータを使用して表を指定します。
- 表領域モード
  データ・ポンプを使用して表領域エクスポートを実行するには、TABLESPACESパラメータを使用して表を指定します。
- トランスポータブル表領域モード
  Oracle Data Pumpを使用してトランスポータブル表領域エクスポートを実行するには、TRANSPORT_TABLESPACESパラメータを使用します。

関連トピック

- Oracle Data Pumpエクスポートの使用例

親トピック: Oracle Data Pumpエクスポートの起動

2.2.2.1 全体モード

データ・ポンプを使用すると、FULLパラメータを使用して全データベースのエクスポートを実行できます。

全データベース・エクスポートでは、データベース全体がアンロードされます。このモードには、DATAPUMP_EXP_FULL_DATABASEロールが必要です。

全体モード・エクスポート時のトランスポータブル・オプションの使用

FULLパラメータをTRANSPORTABLE=ALWAYSパラメータとともに指定すると、データ・ポンプによって全体トランスポータブル・エクスポートが実行されます。全体トランスポータブル・エクスポートでは、データベースの完全なコピーを作成するために必要なすべてのオブジェクトおよびデータがエクスポートされます。次のデータ移動方法の組合せが使用されます。

- トランスポータブル表領域に存在するオブジェクトは、そのメタデータのみがダンプ・ファイル・セットにアンロードされ、データ自体はデータ・ファイルをターゲット・データベースにコピーしたときに移動されます。コピーする必要のあるデータ・ファイルは、エクスポート操作のログ・ファイルの最後に表示されます。
- 非トランスポータブル表領域に存在するオブジェクト(SYSTEMやSYSAUXなど)は、ダイレクト・パス・アンロードおよび外部表を使用して、そのメタデータとデータの両方がダンプ・ファイル・セットにアンロードされます。

全体トランスポータブル・エクスポートの実行には、次の制限があります。

- 全体トランスポータブル・エクスポートを実行するユーザーには、DATAPUMP_EXP_FULL_DATABASE権限が必要です。
- エクスポートを実行するユーザーのデフォルトの表領域を、転送対象となっている表領域のいずれかに設定することはできません。
- エクスポートされるデータベースに、暗号化された表領域または暗号化された列(透過的データ暗号化(TDE)列またはSecureFiles LOB列)のある表が含まれる場合、ENCRYPTION_PASSWORDパラメータも指定する必要があります。
- ソース・データベースとターゲット・データベースは、ソース・データベースに暗号化された表領域がある場合、同じエンディアンのプラットフォーム上に存在する必要があります。
- ソース・プラットフォームとターゲット・プラットフォームのエンディアンが異なる場合、転送するデータを変換してターゲット・プラットフォームの形式にする必要があります。DBMS_FILE_TRANSFERパッケージまたはRMAN CONVERTコマンドを使用してデータを変換できます。
- 全体トランスポータブル・エクスポートは再開できません。
- エクスポートのために選択された記憶域を持つすべてのオブジェクトでは、そのすべての記憶域セグメントが、管理対象の非トランスポータブル表領域(SYSTEM/SYSAUX)内に全体として存在するか、ユーザー定義のトランスポータブル表領域内に全体として存在する必要があります。シングル・オブジェクトの記憶域は、2種類の表領域にまたがることはできません。
- 全体トランスポータブル・エクスポートを使用してネットワークを介してデータベースを転送する場合、監査証跡情報自体がユーザー定義の表領域に格納されていると、管理表領域(SYSTEMやSYSAUXなど)に格納された表に対して監査を有効にすることはできません。
- ソース・データベースとターゲット・データベースの両方でOracle Database 12cを実行している場合、全体トランスポータブル・エクスポートを実行するには、データ・ポンプのVERSIONパラメータを12.0以上に設定するか、COMPATIBLEデータベース初期化パラメータを12.0以上に設定する必要があります。

全体トランスポータブル・エクスポートは、リリース11.2.0.3を実行しているソース・データベースでサポートされます。これを行うには、次の構文例に示すように、データ・ポンプのVERSIONパラメータを12.0以上に設定します。user_nameは、全体トランスポータブル・エクスポートを実行するユーザーです。

```
> expdp user_name FULL=y DUMPFILE=expdat.dmp DIRECTORY=data_pump_dir
      TRANSPORTABLE=always VERSION=12.0 LOGFILE=export.log
```

関連トピック

- FULL
- TRANSPORTABLE

関連項目:

- RMAN CONVERTコマンドの詳細は、Oracle Databaseバックアップおよびリカバリ・リファレンスを参照してください
- 全体トランスポータブル・エクスポートの例は、Oracle Database管理者ガイドを参照してください

親トピック: Oracle Data Pump Exportモード

2.2.2.2 スキーマ・モード

データ・ポンプでスキーマ・エクスポートを指定するには、SCHEMASパラメータを使用します。スキーマ・エクスポートは、デフォルトのエクスポート・モードです。

DATAPUMP_EXP_FULL_DATABASEロールを所有している場合は、スキーマのリストを指定し、オプションでスキーマ定義およびこれらのスキーマに対するシステム権限を指定することができます。DATAPUMP_EXP_FULL_DATABASEロールを所有していない場合は、自分のスキーマのみをエクスポートできます。

SYSスキーマは、エクスポート・ジョブのソース・スキーマとして使用できません。

相互スキーマ参照は、参照されるスキーマがエクスポート対象のスキーマのリストにも指定されないかぎり、エクスポートされません。たとえば、指定されたいずれかのスキーマ内の表にトリガーが定義されていても、そのトリガーが、明示的に指定されていないスキーマ内に常駐している場合はエクスポートされません。また、指定されたスキーマ内の表が依存する外部型定義もエクスポートされません。この場合は、インポート時に、型定義がターゲット・インスタンスにすでに存在している必要があります。

関連トピック

- SCHEMAS

親トピック: Oracle Data Pump Exportモード

2.2.2.3 表モード

データ・ポンプを使用して表モード・エクスポートを実行するには、TABLESパラメータを使用して表を指定します。

表モードでは、指定した表、パーティションおよびそれらの依存オブジェクトのみがアンロードされます。所有するスキーマや列のタイプなど、表の作成に必要なすべてのオブジェクトがすでに存在している必要があります。

TRANSPORTABLE=ALWAYSパラメータとTABLESパラメータを組み合せて指定すると、オブジェクト・メタデータのみがアンロードされます。実際のデータを移動するには、データ・ファイルをターゲット・データベースにコピーします。これにより、エクスポート時間が短縮されます。異なるリリースまたはプラットフォーム間でデータ・ファイルを移動する場合は、データ・ファイルをOracle Recovery Manager (RMAN)で処理する必要があります。

自分のスキーマに存在しない表を指定するには、DATAPUMP_EXP_FULL_DATABASEロールが必要です。列の型定義は、表モードではエクスポートされません。この場合は、インポート時に、型定義がターゲット・インスタンスにすでに存在している必要があります。また、スキーマ・エクスポートの場合と同様に、相互スキーマ参照もエクスポートされません。

表および表パーティションをリカバリするには、RMANバックアップおよびRMAN RECOVER TABLEコマンドも使用できます。このプロセス中に、RMANによって、リカバリされたオブジェクトを含むデータ・ポンプ・エクスポートのダンプ・ファイルが作成(およびオプションでインポート)されます。プラットフォーム間でのデータの転送の詳細は、*Oracle Database*バックアップおよびリカバリ・ガイ

ドを参照してください。

表モード・エクスポートを実行する場合、次の制限事項があります。

- TRANSPORTABLE=ALWAYSパラメータをTABLESパラメータとともに使用する場合、エクスポート対象の表に暗号化された列(透過的データ暗号化(TDE)列またはSecureFiles LOB列のいずれか)が含まれている場合は、ENCRYPTION_PASSWORDパラメータも使用する必要があります。

関連トピック

- TABLES
- TRANSPORTABLE
- 『Oracle Databaseバックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』

親トピック: Oracle Data Pump Exportモード

2.2.2.4 表領域モード

データ・ポンプを使用して表領域エクスポートを実行するには、TABLESPACESパラメータを使用して表を指定します。

表領域モードでは、指定した表領域内に存在する表のみがアンロードされます。表がアンロードされると、その表の依存オブジェクトもアンロードされます。オブジェクトのメタデータとデータは、両方ともアンロードされます。表領域モードでは、指定した表領域内に表の一部が存在する場合、その表とその表のすべての依存オブジェクトがエクスポートされます。特権ユーザーは、すべての表を取得します。権限のないユーザーは、自分のスキーマ内の表のみを取得します。

関連トピック

- TABLESPACES

親トピック: Oracle Data Pump Exportモード

2.2.2.5 トランスポータブル表領域モード

Oracle Data Pumpを使用してトランスポータブル表領域エクスポートを実行するには、TRANSPORT_TABLESPACESパラメータを使用します。

トランスポータブル表領域モードでは、指定した表領域セット内にある表のメタデータ(およびその表の依存オブジェクト)のみがエクスポートされます。表領域データ・ファイルは別の操作でコピーされます。次に、トランスポータブル表領域インポートが実行され、メタデータを含むダンプ・ファイルのインポートと、使用するデータ・ファイルの指定が行われます。

トランスポータブル表領域モードでは、指定した表がすべてその表領域に含まれている必要があります。つまり、表領域セット内に定義されているすべての表(およびその索引)のすべての記憶域セグメントも、セット内に含まれている必要があります。表領域セット内に違反が含まれている場合は、エクスポート・ユーティリティが実際にエクスポートを実行することなくすべての問題を識別します。

指定した表領域内にある表の列の型定義がエクスポートおよびインポートされます。これらの型を所有するスキーマが、ターゲット・インスタンスに存在する必要があります。

トランスポータブル表領域エクスポートは、停止すると再開できません。また、トランスポータブル表領域エクスポートには、1を超える並列度は指定できません。

ノート:

トランスポータブル表領域をエクスポートした後、それよりも古いリリース・レベルのデータベースにインポートすることは

できません。ターゲット・データベースのリリース・レベルは、ソース・データベース以上である必要があります。

Oracle Data Pumpを使用してトランスポータブル表領域エクスポートを実行する場合、次の制限事項があります。

- エクスポート対象のいずれかの表領域に、暗号化された列(透過的データ暗号化(TDE)列またはSecureFiles LOB列のいずれか)のある表が含まれている場合は、ENCRYPTION_PASSWORDパラメータも指定する必要があります。
- エクスポート対象のいずれかの表領域が暗号化されている場合、ENCRYPTION_PASSWORDの使用はオプションですが、使用することをお薦めします。この場合、ENCRYPTION_PASSWORDを省略すると、次の警告メッセージが表示されます。

  ```
  ORA-39396: Warning: exporting encrypted data using transportable option without
  password
  ```

  この警告は、このようなトランスポータブル表領域ジョブを正常にインポートするには、エクスポートを実行するとき、ソース・データベースで使用されるものと同じデータベース・マスター・キーのコピーが、ターゲット・データベース・ウォレットに含まれている必要があることを示します。エクスポート中およびインポート中にENCRYPTION_PASSWORDパラメータを使用すると、この要件は排除されます。

関連トピック

- Oracle Data Pumpによるタイムスタンプ・データの処理方法

親トピック: Oracle Data Pump Exportモード

2.2.3 Oracle Data Pumpエクスポートのネットワークに関する考慮事項

Oracle Data Pumpエクスポート・ユーティリティexpdpがOracle* Netまたはネット・サービス名を使用して接続文字列に含まれた接続識別子でインスタンスを識別する方法と、NETWORK_LINKパラメータを使用したエクスポート操作との相違点について学習します。

expdpの起動時には、接続文字列で接続識別子を指定できます。この接続識別子は、現在のOracleシステムID (SID)で識別される現在のインスタンスとは別のものにすることができます。

接続識別子の指定には、Oracle*Net接続記述子を使用することも、接続記述子にマップされるネット・サービス名(通常はtnsnames.oraファイルで定義されているもの)を使用することもできます。接続識別子を使用する場合は、Oracle Net Listenerを実行している必要があります(デフォルト・リスナーを起動するには、lsnrctl startと入力します)。次の例では、このタイプの接続を示します(inst1が接続識別子です)。

```
expdp hr@inst1 DIRECTORY=dpump_dir1 DUMPFILE=hr.dmp TABLES=employees
```

エクスポートによってパスワードの入力を要求されます。

```
Password: password
```

ローカル・エクスポート・クライアントは接続識別子inst1 (Netサービス名)で定義されるデータベース・インスタンスに接続して、inst1からデータを取得し、そのデータをinst1のダンプ・ファイルhr.dmpに書き込みます。

エクスポート・ユーティリティの起動時に接続識別子を指定することと、NETWORK_LINKパラメータを使用してエクスポート操作を実行することは異なります。エクスポート操作を開始して接続識別子を指定すると、接続識別子によって識別されたデータベース・インスタンスにローカルのエクスポート・クライアントが接続され、データベース・インスタンスのデータが取得され、そのデータベース・インスタンスのダンプ・ファイル・セットにデータが書き込まれます。その一方、NETWORK_LINKパラメータを使用してエクスポートを実行すると、エクスポートはデータベース・リンクを使用して実行されるようになります。(データベース・リンクとは2つの物理

データベース・サーバー間の接続で、データベース・リンクによってクライアントは2つの物理データベース・サーバーに1つの論理データベースとしてアクセスできます。)

関連トピック

- NETWORK_LINK

関連項目:

- データベース・リンクの詳細は、Oracle Database管理者ガイドを参照
- 接続識別子およびOracle Net Listenerの詳細は、Oracle Database Net Services管理者ガイドを参照してください

親トピック: Oracle Data Pumpエクスポートの起動

2.3 エクスポート操作中のフィルタ処理

Oracle Data Pump Exportは、データとメタデータのフィルタ処理機能を提供します。この機能により、エクスポートする情報の種類を効果的に制限できます。

- Oracle Data Pumpエクスポートのデータ・フィルタ
  エクスポートする表の行に制限を指定するには、QUERYパラメータおよびSAMPLEパラメータを通じてOracle Data Pumpのデータ固有のフィルタ処理を使用します。
- メタデータ・フィルタ
  メタデータのフィルタ処理は、EXCLUDEおよびINCLUDEパラメータによって実装されます。EXCLUDEおよびINCLUDEは、相互に排他的なパラメータです。

親トピック: Oracle Data Pumpエクスポート

2.3.1 Oracle Data Pumpエクスポートのデータ・フィルタ

エクスポートする表の行に制限を指定するには、QUERYパラメータおよびSAMPLEパラメータを通じてOracle Data Pumpのデータ固有のフィルタ処理を使用します。

Oracle Data Pumpは、メタデータのフィルタ処理のためにデータのフィルタ処理を間接的に実装することもできます。この処理では、表オブジェクトおよび関連付けられた行データを含めたり、除外したりできます。

各データ・フィルタは、表ごとにジョブ内で1回指定できます。同じ名前を使用する異なるフィルタが特定の表とジョブ全体の両方に適用された場合は、特定の表に対して提供されたフィルタ・パラメータが優先されます。

親トピック: エクスポート操作中のフィルタ処理

2.3.2 メタデータ・フィルタ

メタデータのフィルタ処理は、EXCLUDEおよびINCLUDEパラメータによって実装されます。EXCLUDEおよびINCLUDEは、相互に排他的なパラメータです。

メタデータ・フィルタは、オブジェクトを、エクスポートまたはインポート操作に含めるか除外するかを識別します。たとえば、パッケージ仕様またはパッケージ本体を含まない全体エクスポートを要求できます。

フィルタを正しく使用して必要な結果を得た場合は、識別されたオブジェクトの依存オブジェクトも、識別されたオブジェクトとも

に処理されます。たとえば、索引を操作に含めるようにフィルタで指定すると、その索引の統計も含まれます。同様に、フィルタで表を除外すると、その表に対する索引、制約、権限およびトリガーも除外されます。

1つのオブジェクト型に対して複数のフィルタが指定されている場合は、それらのフィルタに対して暗黙的なAND処理が適用されます。つまり、ジョブに関連するオブジェクトは、オブジェクト型に適用されるすべてのフィルタで処理される必要があります。

同じメタデータ・フィルタ名を、1つのジョブ内で複数回指定できます。

DATABASE_EXPORT_OBJECTS(全体モードの場合)、SCHEMA_EXPORT_OBJECTS(スキーマ・モードの場合)、TABLE_EXPORT_OBJECTS(表および表領域モードの場合)ビューを問い合せて、有効なオブジェクト・タイプ値の一覧を表示できます。OBJECT_PATH列にリストされる値が有効なオブジェクト・タイプです。たとえば、次の問合せを実行できます。

```
SQL> SELECT OBJECT_PATH, COMMENTS FROM SCHEMA_EXPORT_OBJECTS
  2  WHERE OBJECT_PATH LIKE '%GRANT' AND OBJECT_PATH NOT LIKE '%/%';
```

次に、この問合せの出力結果の例を示します。

```
OBJECT_PATH
--------------------------------------------------------------------------------
COMMENTS
--------------------------------------------------------------------------------
GRANT
Object grants on the selected tables

OBJECT_GRANT
Object grants on the selected tables

PROCDEPOBJ_GRANT
Grants on instance procedural objects

PROCOBJ_GRANT
Schema procedural object grants in the selected schemas

ROLE_GRANT
Role grants to users associated with the selected schemas

SYSTEM_GRANT
System privileges granted to users associated with the selected schemas
```

関連トピック

- EXCLUDE
- INCLUDE
- EXCLUDE
- INCLUDE

親トピック: エクスポート操作中のフィルタ処理

2.4 データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

データ・エクスポートを管理するには、エクスポート(expdp)用のOracle Data Pumpパラメータを使用します。

- データ・ポンプ・エクスポート・パラメータについて
  大/小文字の区別、引用符、エスケープ文字、使用方法の例に関する情報など、コマンドライン・モードでOracle Data Pump Exportパラメータを使用する方法について学習します。
- ABORT_STEP

Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのABORT_STEPパラメータは、ジョブが初期化された後にジョブを停止します。

- ACCESS_METHOD
  Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのACCESS_METHODパラメータでは、エクスポートで特定のメソッドを使用してデータをアンロードするように指定します。
- ATTACH
  Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのATTACHパラメータでは、ワーカーまたはクライアント・セッションを既存のエクスポート・ジョブに接続し、自動的に対話方式コマンド・インタフェースに移動します。
- CLUSTER
  Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのCLUSTERパラメータでは、データ・ポンプでOracle RACリソースを使用できるかどうかと、ワーカーを他のOracle RACインスタンス上で開始できるかどうかを指定します。
- COMPRESSION
  Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのCOMPRESSIONパラメータでは、ダンプ・ファイル・セットに書き込む前に圧縮するデータを指定します。
- COMPRESSION_ALGORITHM
  Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのCOMPRESSION_ALGORITHMパラメータでは、ダンプ・ファイル・データの圧縮時に使用する圧縮アルゴリズムを指定します。
- CONTENT
  Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのCONTENTパラメータを使用すると、エクスポートでアンロードする内容を、データのみ、メタデータのみ(あるいはその両方)でフィルタ処理できます。
- DATA_OPTIONS
  Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのDATA_OPTIONSパラメータでは、特定のデータ型に対するエクスポート操作中の処理方法を指定します。
- DIRECTORY
  Oracle Data Pump Exportコマンドライン・ユーティリティのDIRECTORYパラメータでは、エクスポートによるダンプ・ファイル・セットおよびログ・ファイルのデフォルトの書込み場所を指定します。
- DUMPFILE
  Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのDUMPFILEパラメータでは、エクスポート・ジョブのダンプ・ファイルの名前を指定し、オプションで、これらのディレクトリ・オブジェクトを指定します。
- ENABLE_SECURE_ROLES
  Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのENABLE_SECURE_ROLESパラメータは、エクスポート時に保護されたロールを誤って使用しないようにします。
- ENCRYPTION
  Oracle Data Pump Exportのコマンドライン・ユーティリティのENCRYPTIONパラメータでは、ダンプ・ファイル・セットに書き込む前にデータを暗号化するかどうかを指定します。
- ENCRYPTION_ALGORITHM
  Oracle Data Pump Exportのコマンドライン・ユーティリティのENCRYPTION_ALGORITHMパラメータでは、暗号化の実行に使用する必要のある暗号化アルゴリズムを指定します。
- ENCRYPTION_MODE
  Oracle Data Pump Exportのコマンドライン・ユーティリティのENCRYPTION_MODEパラメータでは、暗号化および復号化を実行する際に使用するセキュリティ・タイプを指定します。
- ENCRYPTION_PASSWORD
  Oracle Data Pump Exportのコマンドライン・ユーティリティのENCRYPTION_PASSWORDパラメータでは、エクス

ポート・ダンプ・ファイル内の暗号化列のデータ、メタデータまたは表データを暗号化するためのパスワードを指定します。このパラメータにより、暗号化されたダンプ・ファイル・セットへの不正なアクセスを防ぎます。

- ENCRYPTION_PWD_PROMPT
  Oracle Data Pump Exportのコマンドライン・ユーティリティのENCRYPTION_PWD_PROMPTでは、Oracle Data Pumpによって暗号化パスワードの入力を要求するかどうかを指定します。
- ESTIMATE
  Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのESTIMATEパラメータでは、エクスポート・ジョブ内の各表で使用されるディスク領域(バイト単位)を見積もる場合にエクスポートで使用される方法を指定します。
- ESTIMATE_ONLY
  Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのESTIMATE_ONLYパラメータでは、エクスポートによって、実際のエクスポート操作を実行することなく、ジョブが消費する領域が見積もられるように指定します。
- EXCLUDE
  データ・ポンプ・エクスポートのコマンドライン・ユーティリティのEXCLUDEパラメータを使用すると、エクスポート操作から除外するオブジェクトおよびオブジェクト型を指定して、エクスポート対象のメタデータをフィルタ処理できます。
- FILESIZE
  Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのFILESIZEパラメータでは、各ダンプ・ファイルの最大サイズを指定します。
- FLASHBACK_SCN
  Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのFLASHBACK_SCNパラメータでは、エクスポートで使用されるシステム変更番号(SCN)を指定して、フラッシュバック問合せユーティリティを使用可能にします。
- FLASHBACK_TIME
  データ・ポンプ・エクスポートのコマンドライン・ユーティリティのFLASHBACK_TIMEパラメータでは、指定した時間に最も近いSCNを検索します。このSCNを使用して、フラッシュバック・ユーティリティを使用可能にします。エクスポート操作は、このSCNまでの一貫性のあるデータを使用して実行されます。
- FULL
  エクスポート・コマンドラインFULLパラメータでは、全体データベース・モード・エクスポートを実行することを指定します
- HELP
  データ・ポンプ・エクスポートのコマンドライン・ユーティリティのHELPパラメータでは、エクスポート・ユーティリティのオンライン・ヘルプを表示します。
- INCLUDE
  データ・ポンプ・エクスポートのコマンドライン・ユーティリティのINCLUDEパラメータを使用すると、現行のエクスポート・モードにオブジェクトおよびオブジェクト型を指定して、エクスポート対象のメタデータをフィルタ処理できます。指定したオブジェクトおよびこれらのオブジェクトのすべての依存オブジェクトがエクスポートされます。これらのオブジェクトに対する権限もエクスポートされます。
- JOB_NAME
  Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのJOB_NAMEパラメータでは、ATTACHを使用してジョブに接続したり、DBA_DATAPUMP_JOBSまたはUSER_DATAPUMP_JOBSビューを使用してジョブを指定する場合など、後続処理でエクスポート・ジョブを指定します。
- KEEP_MASTER
  Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのKEEP_MASTERパラメータは、正常に完了したOracle Data Pumpジョブの最後に、データ・ポンプ制御ジョブ表を削除するか、保持するかを指定します。
- LOGFILE
  データ・ポンプ・エクスポートのコマンドライン・ユーティリティのLOGFILEパラメータでは、エクスポート・ジョブのログ・ファイ

ルの名前を指定し、オプションで、そのログ・ファイルのディレクトリを指定します。

- LOGTIME
  Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのLOGTIMEパラメータでは、エクスポート操作中に表示されるメッセージにタイムスタンプを付けることを指定します。
- METRICS
  データ・ポンプ・エクスポートのコマンドライン・ユーティリティのMETRICSパラメータでは、ジョブに関する詳細情報をデータ・ポンプ・ログ・ファイルに記録するかどうかを指定します。
- NETWORK_LINK
  データ・ポンプ・エクスポートのコマンドライン・ユーティリティのNETWORK_LINKパラメータを使用すると、有効なデータベース・リンクで識別される(ソース)データベースからエクスポートできます。ソース・データベース・インスタンスのデータは、接続されたデータベース・インスタンスのダンプ・ファイル・セットに書き込まれます。
- NOLOGFILE
  データ・ポンプ・エクスポートのコマンドライン・ユーティリティのNOLOGFILEパラメータでは、ログ・ファイルを作成するかどうかを指定します。
- PARALLEL
  Oracle Data Pumpのエクスポート・コマンドライン・ユーティリティのPARALLELパラメータでは、エクスポート・ジョブのために動作するアクティブな実行プロセスの最大数を指定します。
- PARALLEL_THRESHOLD
  Oracle Data Pumpのエクスポート・コマンドライン・ユーティリティのPARALLEL_THRESHOLDパラメータでは、表サイズに基づいてデータ・ポンプが潜在的なパラレルDMLを計算するために使用する除数のサイズを指定します。
- PARFILE
  データ・ポンプ・エクスポートのコマンドライン・ユーティリティのPARFILEパラメータは、エクスポート・パラメータ・ファイルの名前を指定します。
- QUERY
  データ・ポンプ・エクスポートのコマンドライン・ユーティリティのQUERYパラメータを使用すると、エクスポート対象となるデータをフィルタ処理するために使用する問合せ句を指定できます。
- REMAP_DATA
  Oracle Data Pump Exportのコマンドライン・ユーティリティのREMAP_DATAパラメータを使用すると再マップ・ファンクションを指定できます。これにより、指定した列の元の値をソースとして再マップした値を返し、ダンプ・ファイル内の元の値をこの値に置き換えます。
- REUSE_DUMPFILES
  データ・ポンプ・エクスポートのコマンドライン・ユーティリティのxxxパラメータでは、既存のダンプ・ファイルを上書きするかどうかを指定します。
- SAMPLE
  データ・ポンプ・エクスポートのコマンドライン・ユーティリティのSAMPLEパラメータでは、ソース・データベースからサンプリングおよびアンロードするデータ行の割合を指定します。
- SCHEMAS
  データ・ポンプ・エクスポートのコマンドライン・ユーティリティのSCHEMASパラメータでは、スキーマ・モード・エクスポートを実行することを指定します。これは、エクスポートのデフォルトのモードです。
- SERVICE_NAME
  Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのSERVICE_NAMEパラメータは、CLUSTERパラメータと組み合せることで使用するサービス名を指定します。
- SOURCE_EDITION

データ・ポンプ・エクスポートのコマンドライン・ユーティリティのSOURCE_EDITIONパラメータでは、オブジェクトのエクスポート元のデータベース・エディションを指定します。

- STATUS
  データ・ポンプ・エクスポートのコマンドライン・ユーティリティのSTATUSパラメータでは、ジョブの状態の表示が更新される頻度を指定します
- TABLES
  データ・ポンプ・エクスポートのコマンドライン・ユーティリティのTABLESパラメータでは、表モード・エクスポートを実行することを指定します。
- TABLESPACES
  データ・ポンプ・エクスポートのコマンドライン・ユーティリティのTABLESPACESパラメータでは、表領域モードでエクスポートされる表領域名のリストを指定します。
- TRANSPORT_FULL_CHECK
  データ・ポンプ・エクスポートのコマンドライン・ユーティリティのTRANSPORT_FULL_CHECKパラメータでは、トランスポータブル・セット内部のオブジェクトと外部のオブジェクトの間の依存性をチェックするかどうかを指定します。このパラメータは、トランスポータブル表領域モード・エクスポートでのみ使用可能です。
- TRANSPORT_TABLESPACES
  データ・ポンプ・エクスポートのコマンドライン・ユーティリティのTRANSPORT_TABLESPACESパラメータでは、トランスポータブル表領域モードでのエクスポートを実行することを指定します
- TRANSPORTABLE
  データ・ポンプ・エクスポートのコマンドライン・ユーティリティのTRANSPORTABLEパラメータでは、表モード・エクスポート(TABLESパラメータで指定)または全体モード・エクスポート(FULLパラメータで指定)中にトランスポータブル・オプションを使用する必要があるかどうかを指定します。
- TTS_CLOSURE_CHECK
  データ・ポンプ・エクスポート・コマンドライン・モードのTTS_CLOSURE_CHECKパラメータは、データ・ポンプのトランスポータブル表領域操作の一部として実行されるクローズ・チェックの程度を示すために使用されます。
- VERSION
  データ・ポンプ・エクスポートのコマンドライン・ユーティリティのVERSIONパラメータでは、エクスポートするデータベース・オブジェクトのバージョンを指定します。
- VIEWS_AS_TABLES
  Oracle Data Pump Exportのコマンドライン・ユーティリティのVIEWS_AS_TABLESパラメータでは、1つ以上のビューを表としてエクスポートすることを指定します。

関連トピック

- PARFILE
- ダンプ・ファイル、ログ・ファイルおよびSQLファイルのデフォルトの位置の理解
- Oracleデータ・ポンプ・エクスポートの使用例
- Oracleデータ・ポンプ・エクスポートの構文図

親トピック: Oracle Data Pumpエクスポート

2.4.1 データ・ポンプ・エクスポート・パラメータについて

大/小文字の区別、引用符、エスケープ文字、使用方法の例に関する情報など、コマンドライン・モードでOracle Data Pump Exportパラメータを使用する方法について学習します。

エクスポート・パラメータの指定

値を複数個指定できるパラメータでは、カンマまたは空白で値を区切ることができます。たとえば、TABLES=employees,jobsまたはTABLES=employees jobsのように指定できます。

入力するパラメータごとに、等号(=)と値を入力する必要があります。それ以外の方法では、1つ前のパラメータ指定がどこで完了し、新しいパラメータ指定がどこから始まるかをデータ・ポンプに伝えることができません。たとえば、次のコマンドラインの場合、NOLOGFILEは有効なパラメータですが、エクスポートでは、この文字列がDUMPFILEパラメータに対する別のダンプ・ファイル名として解釈されます。

```
expdp DIRECTORY=dpumpdir DUMPFILE=test.dmp NOLOGFILE TABLES=employees
```

結果として、このコマンドによりtest.dmpおよびnologfile.dmpという2つのダンプ・ファイルが作成されることになります。

この結果を回避するには、NOLOGFILE=YESまたはNOLOGFILE=NOのいずれかを指定します。

パラメータ値を指定する場合の大/小文字の区別

パラメータ値として入力される表領域名、スキーマ名、表名などについて、Oracle Data Pumpは、小文字または大/小文字混在で入力された値をデフォルトで大文字に変更します。たとえば、TABLE=hr.employeesと入力すると、それはTABLE=HR.EMPLOYEESに変更されます。大/小文字を維持するには、引用符で値を囲む必要があります。たとえば、TABLE="hr.employees"とすると、表名がすべて小文字のまま維持されます。入力する名前は、データベースに格納されている名前と完全に一致する必要があります。

データ・ポンプ・コマンドラインでの引用符の使用

一部のオペレーティング・システムでは、特殊文字として引用符が処理されます。したがって、これらのオペレーティング・システムでは、引用符の前にバックスラッシュ(¥)などのエスケープ文字を付けないかぎり、引用符はアプリケーションに渡されません。この要件は、コマンドラインとパラメータ・ファイル内の両方に当てはまります。また、オペレーティング・システムによっては、コマンドラインで、特殊文字を含むパラメータ値全体を一重引用符または二重引用符で囲む必要があります。

次の例では、これらの概念を説明します。特定のオペレーティング・システムでは、要件が異なる場合があります。ドキュメント例では、各ユーザーに固有のオペレーティング環境を完全に予測することはできません。

この例では、パラメータ・ファイルにTABLESパラメータが指定されています。

```
TABLES = ¥"MixedCaseTableName¥"
```

コマンドラインでその値を指定する場合、一部のオペレーティング・システムでは、次のように一重引用符を使用してパラメータ・ファイル名を囲む必要があります。

```
TABLES = '¥"MixedCaseTableName¥"'
```

コマンドラインで追加の引用符を指定せずに手順を続行するには、パラメータ・ファイルを使用することをお薦めします。また、パラメータ・ファイルを使用し、指定するパラメータ値の文字列の最初の文字に引用符を使用しない場合(たとえば、TABLES=scott."EmP")は、エスケープ文字の使用が不要になるシステムがあることに注意してください。

エクスポート・パラメータの使用例

各項に示す例を試行する場合は、次の内容に注意してください。

- 例に示すようにユーザー名およびパラメータを入力した後、エクスポートが起動され、パスワードの入力を求められます。データベース接続が行われる前に、パスワードを入力する必要があります。
- ここに示す例の多くは、Oracle Databaseのインストール時にデフォルトでインストールされるシード・データベースのサンプル・スキーマを使用しています。特に、人事管理(hr)スキーマを頻繁に使用します。

- この例では、ディレクトリ・オブジェクトdpump_dir1およびdpump_dir2がすでに存在し、これらのディレクトリ・オブジェクトについてのREAD権限およびWRITE権限が、hrユーザーに付与されているものとします。
- 一部の例では、DATAPUMP_EXP_FULL_DATABASEロールおよびDATAPUMP_IMP_FULL_DATABASEロールが必要です。その例では、hrユーザーにこれらのロールが付与されているものとします。

必要に応じて、これらのディレクトリ・オブジェクトの作成と、必要な権限やロールの割当てをDBAに依頼します。

特に指定がないかぎり、これらのパラメータはパラメータ・ファイルでも指定できます。

関連項目:

- Oracle Databaseサンプル・スキーマ
- システムでの特殊文字および予約文字の処理方法の詳細は、ご使用のオペレーティング・システム固有のOracleマニュアルを参照してください。

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.2 ABORT_STEP

Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのABORT_STEPパラメータでは、ジョブの初期化後に、そのジョブを停止します。

デフォルト

Null

用途

ジョブの初期化後に、そのジョブを停止する場合に使用します。ジョブを初期化後に停止すると、データがエクスポートされる前に問い合せるデータ・ポンプ制御ジョブ表を問い合せることができます。

構文および説明

```
ABORT_STEP=[n | -1]
```

可能な値は、データ・ポンプ制御ジョブ表のプロセス順序番号に対応します。各番号の使用結果は次のとおりです。

- n: 値が0以上の場合は、エクスポート操作が開始され、対応するプロセス・オーダー番号を持つ、データ・ポンプ制御ジョブ表に格納されているオブジェクトでジョブが停止されます。
- -1: 値が負の1 (-1)の場合は、ジョブの設定後(ただし、オブジェクトまたはデータのエクスポート前)に、ジョブが中断されます。

制限事項

- なし

例

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=expdat.dmp SCHEMAS=hr ABORT_STEP=-1
```

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.3 ACCESS_METHOD

Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのACCESS_METHODパラメータでは、エクスポートで特定のメソッドを使用してデータをアンロードするように指定します。

デフォルト

AUTOMATIC

用途

エクスポート・ユーティリティで特定のメソッドを使用してデータをアンロードするように指定します。

構文および説明

```
ACCESS_METHOD=[AUTOMATIC | DIRECT_PATH | EXTERNAL_TABLE | INSERT_AS_SELECT]
```

デフォルトのメソッドがなんらかの理由で動作しない場合に別のネイティブ・メソッドを試行できるように、ACCESS_METHODパラメータが提供されます。ネットワーク・エクスポートでは、すべての方法を指定できます。指定したアクセス方法で表のデータをアンロードできない場合は、データに表のエラーが表示され、次の作業項目に進みます。

使用可能なオプションは次のとおりです。

- AUTOMATIC—それぞれの表について、データをアップロードする最適な方法がOracle Data Pumpによって判断されます。できるだけAUTOMATICを使用することをお薦めします。これは、データ・ポンプが最も効果的な方法を自動的に選択できるようになるためです。
- DIRECT_PATH—Oracle Data Pumpによって、すべての表に対して、ダイレクト・パス・アンロードが使用されます。
- EXTERNAL_TABLE—Oracle Data Pumpによって、SQL文CREATE TABLE AS SELECTを使用して、ダンプ・ファイルに格納されているデータを使用する外部表が作成されます。SELECT句では、アンロードする表から読み取ります。
- INSERT_AS_SELECT—Oracle Data Pumpによって、SQL文INSERT AS SELECTが実行され、リモート・データベースからデータがアンロードされます。このオプションは、ネットワーク・モード・エクスポートでのみ使用できます。

制限事項

- ネットワーク・エクスポートでACCESS_METHODパラメータを使用するには、Oracle Database 12c リリース2 (12.2.0.1)以上を使用している必要があります。
- Oracle Data PumpエクスポートのACCESS_METHODパラメータは、トランスポータブル表領域のジョブに対して有効ではありません。

例

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=expdat.dmp SCHEMAS=hr
ACCESS_METHOD=EXTERNAL_TABLE
```

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.4 ATTACH

Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのATTACHパラメータでは、ワーカーまたはクライアント・セッションを既存のエクスポート・ジョブに接続し、自動的に対話方式コマンド・インタフェースに移動します。

デフォルト

デフォルトは、ユーザー・スキーマに現在存在するジョブです(ジョブが存在する場合)。

用途

ワーカー・セッションを既存のデータ・ポンプ制御エクスポート・ジョブにアタッチし、対話方式コマンド・インタフェースに自動的に配置します。接続先のジョブの説明およびエクスポート・プロンプトが表示されます。

構文および説明

```
ATTACH [=[schema_name.]job_name]
```

schema_nameは、オプションです。自分のスキーマ以外のスキーマを指定するには、DATAPUMP_EXP_FULL_DATABASEロールが必要です。

job_nameは、スキーマに対応するエクスポート・ジョブが1つのみで、そのジョブがアクティブの場合はオプションです。停止しているジョブに接続する場合は、このジョブ名を指定する必要があります。DBA_DATAPUMP_JOBSビューまたはUSER_DATAPUMP_JOBSビューを問い合せて、データ・ポンプ・ジョブ名の一覧を表示できます。

ジョブに接続している場合、エクスポート・ユーティリティでは、ジョブの説明が表示され、次にエクスポート・プロンプトが表示されます。

制限事項

- ATTACHパラメータを指定する場合、コマンドラインで他に指定できるデータ・ポンプ・パラメータは、ENCRYPTION_PASSWORDのみです。
- 接続するジョブが最初に暗号化パスワードを使用して起動している場合、そのジョブへの接続時に、コマンドライン上のENCRYPTION_PASSWORDパラメータを再入力してそのパスワードを再指定する必要があります。この要件の唯一の例外は、ジョブが最初にENCRYPTION=ENCRYPTED_COLUMNS_ONLYパラメータを使用して開始されている場合です。この場合、ジョブへの接続時に暗号化パスワードは必要ありません。
- そのジョブが実行中でなければ、別のスキーマのジョブに接続することはできません。
- ジョブのダンプ・ファイル・セットまたはデータ・ポンプ制御表が削除されている場合、接続操作は失敗します。
- データ・ポンプ制御表をなんらかの方法で変更すると、予測できない結果になります。

例

次に、ATTACHパラメータの使用例を示します。ジョブhr.export_jobがすでに存在するものとします。

```
> expdp hr ATTACH=hr.export_job
```

関連トピック

- データ・ポンプ・エクスポートの対話方式コマンド・モードで使用可能なコマンド

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.5 CLUSTER

Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのCLUSTERパラメータでは、データ・ポンプでOracle RACリソースを使用できるかどうかと、ワーカーを他のOracle RACインスタンス上で開始できるかどうかを指定します。

デフォルト

YES

用途

Oracle Data PumpでOracle Real Application Clusters(Oracle RAC)リソースを使用できるかどうかと、ワーカーを他のOracle RACインスタンス上で開始できるかどうかを決定します。

構文および説明

```
CLUSTER=[YES | NO]
```

Oracle Data Pumpエクスポートで、ジョブが開始されたインスタンスのみを使用し、Oracle Database 11gリリース2(11.2)より前の動作と同じにするには、CLUSTER=NOを指定します。

特定の既存サービスを指定し、そのサービスに定義されたインスタンスでのみ実行されるようにワーカー・プロセスを制限するには、SERVICE_NAMEパラメータをCLUSTER=YESパラメータとともに使用します。

CLUSTERパラメータを使用すると、エクスポート・ジョブをOracle RACインスタンス間で分散するための追加のオーバーヘッドが発生するため、パフォーマンスに影響する場合があります。ジョブが小さいときは、CLUSTER=NOを指定して、ジョブが開始されたインスタンス上のみでジョブを実行した方がよい場合もあります。CLUSTERパラメータを使用してジョブのパフォーマンスが最もよくなるのは、ジョブのデータ量が多い場合です。

例

次に、CLUSTERパラメータの使用例を示します。

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=hr_clus%U.dmp CLUSTER=NO PARALLEL=3
```

この例では、hrスキーマのスキーマ・モード・エクスポート(デフォルト)を実行します。CLUSTER=NOが指定されているため、ジョブでは起動時のインスタンスのみが使用されます。CLUSTERパラメータを指定しない場合は、デフォルト値のYが使用されます。この値を使用すると、必要に応じて、ワーカーはOracle RACクラスタ内の他のインスタンスで開始されます。ダンプ・ファイルは、dpump_dir1ディレクトリ・オブジェクトに指定された場所に書き込まれます。ジョブでは、パラレル処理を最大3つまで実行できます。

関連トピック

- SERVICE_NAME
- Oracle RACでのOracle Data Pumpの使用方法の理解

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.6 COMPRESSION

Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのCOMPRESSIONパラメータでは、ダンプ・ファイル・セットに書き込む前に圧縮するデータを指定します。

デフォルト

METADATA_ONLY

用途

ダンプ・ファイル・セットに書き込む前に圧縮するデータを指定します。

構文および説明

```
COMPRESSION=[ALL | DATA_ONLY | METADATA_ONLY | NONE]
```

- ALLを指定すると、エクスポート操作全体について圧縮が有効になります。ALLオプションでは、Oracle Advanced

Compressionオプションを有効にする必要があります。

- DATA_ONLYを指定すると、すべてのデータが圧縮形式でダンプ・ファイルに書き込まれます。DATA_ONLYオプションでは、Oracle Advanced Compressionオプションを有効にする必要があります。
- METADATA_ONLYを指定すると、すべてのメタデータが圧縮形式でダンプ・ファイルに書き込まれます。これはデフォルトです。
- NONEを指定すると、エクスポート操作全体について圧縮が無効になります。

制限事項

- これらすべての圧縮オプションを使用するには、COMPATIBLE初期化パラメータを11.0.0以上に設定する必要があります。
- METADATA_ONLYオプションは、たとえCOMPATIBLE初期化パラメータが10.2に設定されていても使用できます。
- ALLまたはDATA_ONLYを使用したデータ圧縮は、Oracle Database 11g以降のEnterprise Editionでのみ有効で、Oracle Advanced Compressionオプションを有効にする必要があります。

例

次に、COMPRESSIONパラメータの使用例を示します。

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=hr_comp.dmp COMPRESSION=METADATA_ONLY
```

このコマンドは、スキーマ・モードのエクスポートを実行し、すべてのメタデータを圧縮してからダンプ・ファイルhr_comp.dmpに書き出します。エクスポート・モードが指定されていないため、デフォルトでスキーマ・モードのエクスポートになります。

Oracle Advanced Compressionオプションのライセンス要件の詳細は、*Oracle Database*ライセンス情報を参照してください。

関連トピック

- Oracle Databaseのオプションおよびそれらに許可されている機能

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.7 COMPRESSION_ALGORITHM

Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのCOMPRESSION_ALGORITHMパラメータでは、ダンプ・ファイル・データの圧縮時に使用する圧縮アルゴリズムを指定します。

デフォルト

BASIC

用途

ダンプ・ファイル・データを圧縮する場合に使用する圧縮アルゴリズムを指定します。

構文および説明

```
COMPRESSION_ALGORITHM = [BASIC | LOW | MEDIUM | HIGH]
```

パラメータ・オプションの定義は次のとおりです。

- BASIC: 圧縮率と速度の適度な組合せが実現し、使用されるアルゴリズムはOracle Data Pumpの以前のバージョンと同じです。

- LOW: エクスポート・スループットへの影響が最小です。このオプションは、CPUリソースが制限要因となる環境に適しています。
- MEDIUM: ほとんどの環境に適しています。このオプションはBASICオプションと同様、圧縮率と処理速度を加味したよい組合せですが、BASICとは異なるアルゴリズムが使用されています。
- HIGH: ネットワーク速度が限定要因となっている、速度が遅いネットワークでダンプ・ファイルがコピーされる状況に最適です。

圧縮アルゴリズムのパフォーマンスを、そのCPU使用量および圧縮率(非圧縮の入力に対する割合としての圧縮済の出力のサイズ)によって特徴付けます。これらの測定値は、使用される圧縮アルゴリズムの速度に加え、入力のサイズおよびタイプに基づいて変化します。基本的に圧縮率はLOWからHIGHへと高くなります。ただし、圧縮率の上昇に応じて、消費されるCPUリソースが多くなります。

ご使用の環境で、データに対して様々な圧縮レベルをテストすることをお薦めします。エクスポートされるダンプ・ファイル・セットの圧縮レベルがパフォーマンスおよび記憶域要件を満たすようにするには、環境、ワークロードの特性、およびデータのサイズとタイプに基づいて圧縮レベルを選択します。

制限事項

- この機能を使用するには、データベース互換性を12.0.0以上に設定する必要があります。
- この機能では、Oracle Advanced Compressionオプションを有効にする必要があります。

例1

この例では、HRスキーマのスキーマ・モード・アンロードを実行し、低い圧縮レベルの圧縮アルゴリズムを使用して表データのみを圧縮します。このコマンド・オプションを使用すると、最適な圧縮率を得ることはできませんが、使用されるCPUリソースの数を抑えることができます。

```
        > expdp hr DIRECTORY=dpump_dir1 DUMPFILE=hr.dmp COMPRESSION=DATA_ONLY
COMPRESSION_ALGORITHM=LOW
```

例2

この例では、HRスキーマのスキーマ・モード・アンロードを実行し、基本の圧縮レベルを使用してメタデータと表データの両方を圧縮します。COMPRESSION_ALGORITHMパラメータを完全に省略することは、値としてBASICを指定することと同じです。

```
        > expdp hr DIRECTORY=dpump_dir1 DUMPFILE=hr.dmp COMPRESSION=ALL
COMPRESSION_ALGORITHM=BASIC
```

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.8 CONTENT

Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのCONTENTパラメータを使用すると、エクスポートでアンロードする内容を、データのみ、メタデータのみ(あるいはその両方)でフィルタ処理できます。

デフォルト

ALL

用途

エクスポート・ユーティリティでアンロードする内容を、データのみ、メタデータのみ(あるいはその両方)でフィルタ処理できます。

構文および説明

```
CONTENT=[ALL | DATA_ONLY | METADATA_ONLY]
```

- ALLを指定すると、データとメタデータの両方がアンロードされます。このオプションはデフォルトです。
- DATA_ONLYを指定すると、表の行データのみがアンロードされます。データベース・オブジェクト定義はアンロードされません。
- METADATA_ONLYを指定すると、データベース・オブジェクト定義のみがアンロードされ、表の行データはアンロードされません。CONTENT=METADATA_ONLYを指定した場合、ダンプ・ファイルが後でインポートされるときに、ダンプ・ファイルからインポートされる索引または表の統計はインポート後にロックされることに注意してください。

制限事項

- CONTENT=METADATA_ONLYパラメータは、TRANSPORT_TABLESPACES(トランスポータブル表領域モード)パラメータまたはQUERYパラメータと組み合せて使用することはできません。

例

次に、CONTENTパラメータの使用例を示します。

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=hr.dmp CONTENT=METADATA_ONLY
```

このコマンドでは、hrのスキーマに関連付けられたメタデータのみをアンロードするスキーマ・モード・エクスポートが実行されます。エクスポート・モードが指定されていないため、デフォルトでhrスキーマのスキーマ・モードのエクスポートになります。

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.9 DATA_OPTIONS

Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのDATA_OPTIONSパラメータでは、特定のデータ型に対するエクスポート操作中の処理方法を指定します。

デフォルト: デフォルト値は設定されていません。このパラメータが使用されていない場合、このパラメータが提供する特別なデータ処理オプションは無効になります。

用途

DATA_OPTIONSパラメータは、エクスポート操作時に特定のタイプのデータをどのように処理するかを指定します。

構文および説明

- GROUP_PARTITION_TABLE_DATA: 各表パーティションを別々の操作としてアンロードするのではなく、1つの操作ですべての表データをアンロードするように、データ・ポンプに指示します。その結果、インポート時に表の定義は考慮されません(インポートでは表全体にロードされる1つのデータ・パーティションが参照されるため)。
- VERIFY_STREAM_FORMAT: データ・ストリームがデータ・ポンプ・ダンプ・ファイルに書き込まれる前に、データ・ストリームの形式を検査します。この検証では、ストリームが生成されてからディスクに書き込まれる前に、有効な形式かどうかがチェックされます。これにより、ダンプ・ファイル作成時のエラーがなくなるため、インポート時にストリームが読み取られる際のエラーも回避できます。

制限事項

```
DATA_OPTIONS= [GROUP_PARTITION_TABLE_DATA | VERIFY_STREAM_FORMAT]
```

- エクスポート・ユーティリティのDATA_OPTIONSパラメータを使用する場合は、ジョブ・バージョンを11.0.0以上に設定する必要があります。VERSIONを参照してください。

例

この例では、パーティションごとにデータをアンロードする個別のデフォルトの動作ではなく、表のすべてのパーティションのデータが一緒にアンロードされるエクスポート操作を示しています。

```
> expdp hr TABLES=hr.tab1 DIRECTORY=dpump_dir1 DUMPFILE=hr.dmp VERSION=11.2
GROUP_PARTITION_TABLE_DATA
```

XMLType表のエクスポートおよびインポートに固有の情報は、*Oracle XML DB*開発者ガイドを参照してください。

関連トピック

- VERSION

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.10 DIRECTORY

Oracle Data Pump Exportコマンドライン・ユーティリティのDIRECTORYパラメータでは、エクスポートがダンプ・ファイル・セットおよびログ・ファイルを書き込むことができるデフォルトの場所を指定します。

デフォルト

DATA_PUMP_DIR

用途

エクスポート・ユーティリティによるダンプ・ファイル・セットおよびログ・ファイルのデフォルトの書込み先を指定します。

構文および説明

```
DIRECTORY=directory_object
```

directory_objectは、データベース・ディレクトリ・オブジェクトの名前です。実際のディレクトリのファイル・パスではありません。特権ユーザーにDATA_PUMP_DIRという名前のデフォルトのディレクトリ・オブジェクトへのアクセス権が付与されます。DATA_PUMP_DIRディレクトリの定義は、アップグレード時またはパッチの適用時にOracleによって変更される可能性があります。

デフォルトのDATA_PUMP_DIRディレクトリ・オブジェクトにアクセス可能なユーザーは、DIRECTORYパラメータを使用する必要はありません。

DUMPFILEパラメータやLOGFILEパラメータで指定したディレクトリ・オブジェクトは、DIRECTORYパラメータに指定したディレクトリ・オブジェクトよりも優先されます。

例

次に、DIRECTORYパラメータの使用例を示します。

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=employees.dmp CONTENT=METADATA_ONLY
```

この例では、employees.dumpが、ディレクトリ・オブジェクトdpump_dir1に関連付けられているパスに書き込まれます。

関連トピック

- ダンプ・ファイル、ログ・ファイルおよびSQLファイルのデフォルトの位置の理解
- Oracle RACでのOracle Data Pumpの使用方法の理解
- Oracle Database SQL言語リファレンス

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.11 DUMPFILE

Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのDUMPFILEパラメータでは、エクスポート・ジョブのダンプ・ファイルの名前を指定し、オプションで、これらのディレクトリ・オブジェクトを指定します。

デフォルト

expdat.dmp

用途

名前を指定します。指定する場合は、エクスポート・ジョブに対するダンプ・ファイルのディレクトリ・オブジェクトを指定します。

構文および説明

```
DUMPFILE=[directory_object:]file_name [, ...]
```

DIRECTORYパラメータを使用してディレクトリ・オブジェクトをすでに指定している場合、directory_objectの指定はオプションです。ここで値を指定する場合は、アクセス権がある既存のディレクトリ・オブジェクトを指定します。DUMPFILEパラメータの一部に指定されるデータベース・ディレクトリ・オブジェクトは、DIRECTORYパラメータによって指定された値またはデフォルトのディレクトリ・オブジェクトよりも優先されます。

カンマで区切ったリストまたは個別のDUMPFILEパラメータ指定で、複数のfile_name指定を定義できます。ファイル拡張子を指定しない場合は、デフォルトのファイル拡張子.dmpが使用されます。ファイル名には置換変数を含めることができます。次の表に、使用可能な置換変数を示します。

置換変数	意味
%U	生成されるファイル名では、置換変数が、01 から 99 まで増加する固定幅の 2 桁の整数に変換されます。ファイル指定に 2 つの置換変数が含まれている場合は、両方の変数が同時に増加します。たとえば、exp%Uaa%U.dmp は、exp01aa01.dmp、exp02aa02.dmp というように変換されます。
%d、%D	グレゴリオ暦による月の現在の日付を DD 形式で指定します。 ノート: この置換変数はインポート・ファイル名には使用できません。
%m、%M	グレゴリオ暦による月を MM 形式で指定します。 ノート: この置換変数はインポート・ファイル名には使用できません。
%t、%T	グレゴリオ暦による年、月および日を YYYYMMDD 形式で指定します。 ノート: この置換変数はインポート・ファイル名には使用できません。
%l、%L	システムによって生成される一意のファイル名を指定します。 ファイル名には、複数のファイルを生成できることを示す置換変数

置換変数	意味
	(%L)を含めることができます。生成されるファイル名では、置換変数が、01 から 99 まで増加する固定幅の 2 桁の整数(%U と同じ)に変換されます。また、生成されるファイル名では、置換変数が、100 から 2147483646 まで増加する 3 桁から 10 桁までの可変幅の整数に変換されます。幅フィールドは、整数の桁数によって決まります。 たとえば、現在の整数が 1 の場合、exp%Laa%L.dmp は次のように解決されます。 exp01aa01.dmp exp02aa02.dmp 以降、99 までは同様に続きます。その次のファイル名では 3 桁に置換されます。 exp100aa100.dmp exp101aa101.dmp 以降、999 まで同様に続き、その次のファイルでは 4 桁に置換されます。最大許容置換数(2147483646)まで、置換が続きます。
%y、%Y	年を YYYY 形式で指定します。 ノート: この置換変数はインポート・ファイル名には使用できません。

FILESIZEパラメータを指定すると、各ダンプ・ファイルがそのサイズを上限とするため、拡張できなくなります。置換変数を持つテンプレートを指定していて、ダンプ・ファイル・セットに必要な領域が不足し、デバイスに余裕がある場合は、FILESIZEパラメータで指定したサイズの新しいダンプ・ファイルが自動的に作成されます。

置換変数が含まれているファイル指定またはファイル・テンプレートは、定義されているとおり、完全修飾されたファイル名としてインスタンス化され、エクスポートによってファイルが作成されます。ファイル指定は、指定した順序で処理されます。ファイル・サイズが上限に達したか、またはパラレル処理をアクティブなままにするためにジョブにファイルを追加する必要がある場合、置換変数を持つファイル・テンプレートが指定されていれば、追加のファイルが作成されます。

DUMPFILEパラメータで複数のファイルを指定することもできますが、エクスポート・ジョブでエクスポート・データを保持するために必要となるのは、それらのファイルのサブセットのみの場合もあります。エクスポート・ジョブの最後に表示されるダンプ・ファイル・セットには、実際に使用されたファイルが示されます。このダンプ・ファイル・セットを使用するインポート操作には、このファイル・リストが必要になります。使用されなかったファイルは無視してかまいません。

DUMPFILEパラメータを指定する場合、置換変数を使用するかどうかに関係なく、ファイル名の競合が発生する可能性があります。ファイル名の競合が発生するexpdpコマンドのいくつかの例を次に示します。これらのすべての例で、「ORA-27308 作成したファイルはすでに存在します」というエラーが返されます。

```
expdp system/manager directory=dpump_dir schemas=hr DUMPFILE=foo%U.dmp,foo%U.dmp

expdp system/manager directory=dpump_dir schemas=hr DUMPFILE=foo%U.dmp,foo%L.dmp
```

```
expdp system/manager directory=dpump_dir schemas=hr DUMPFILE=foo%U.dmp,foo%D.dmp

expdp system/manager directory =dpump_dir schemas=hr
DUMPFILE=foo%tK_%t_%u_%y_P,foo%TK_%T_%U_%Y_P
```

制限事項

- 作成されるダンプ・ファイルの名前と既存のダンプ・ファイルの名前が一致すると、エラーが生成されます。既存のダンプ・ファイルが上書きされることはありません。この動作を上書きするには、エクスポート・ユーティリティのパラメータREUSE_DUMPFILES=YESを指定します。
- Oracle Data PumpのパラメータVERSION=12を使用してOracle Database 11gリリースで作成したダンプ・ファイルは、Oracle Database 12cリリース1 (12.1)以降でのみインポートできます。

例

次に、DUMPFILEパラメータの使用例を示します。

```
> expdp hr SCHEMAS=hr DIRECTORY=dpump_dir1 DUMPFILE=dpump_dir2:exp1.dmp,
 exp2%U.dmp PARALLEL=3
```

ダンプ・ファイルexp1.dmpは、ディレクトリ・オブジェクトdpump_dir2に対応するパスに書き込まれます。これは、dpump_dir2が、ダンプ・ファイル名の一部として指定されたことによって、DIRECTORYパラメータで指定されたディレクトリ・オブジェクトよりも優先されるためです。3つのパラレル処理のすべてに、このジョブの間に実行する作業が与えられるため、指定した置換変数exp2%U.dmpに基づいて、exp201.dmpおよびexp202.dmpという名前のダンプ・ファイルが作成されます。ディレクトリが指定されていないため、作成されたファイルは、DIRECTORYパラメータで指定されたディレクトリ・オブジェクトdpump_dir1に関連付けられたパスに書き込まれます。

関連トピック

- Oracle Data Pump Exportでの代替変数の使用

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.12 ENABLE_SECURE_ROLES

Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのENABLE_SECURE_ROLESパラメータでは、エクスポート時の保護されているロールの誤用を防止します。

デフォルト

Oracle Database 19c以降のリリースでは、デフォルト値はNOです。

用途

Oracleロールには、認証が必要なものがあります。そのようなロールをOracle Data Pumpエクスポートで使用する必要がある場合は、ENABLE_SECURE_ROLESパラメータをYESに設定することで明示的に使用可能にする必要があります。

構文

ENABLE_SECURE_ROLES=[NO|YES]

- NO: 認証が必要なOracleロールを使用できないようにします。
- YES: 認証が必要なOracleロールを使用可能にします。

例

```
expdp hr SCHEMAS=hr DIRECTORY=dpump_dir1 DUMPFILE=dpump_dir2:exp1.dmp,
```

```
exp2%U.dmp ENABLE_SECURE_ROLES=YES
```

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.13 ENCRYPTION

Oracle Data Pump Exportのコマンドライン・ユーティリティのENCRYPTIONパラメータでは、ダンプ・ファイル・セットに書き込む前にデータを暗号化するかどうかを指定します。

デフォルト

デフォルト値は、使用される暗号化関連パラメータの組合せによって異なります。暗号化を有効にするには、ENCRYPTIONまたはENCRYPTION_PASSWORDパラメータ、あるいは両方を指定する必要があります。

ENCRYPTION_PASSWORDパラメータのみが指定されている場合は、デフォルトでENCRYPTIONパラメータがALLに設定されます。

ENCRYPTIONパラメータのみが指定され、Oracle暗号化ウォレットがオープン状態の場合、デフォルト・モードはTRANSPARENTになります。ENCRYPTIONパラメータのみが指定され、ウォレットが閉じられている場合は、エラーが返されます。

ENCRYPTIONとENCRYPTION_PASSWORDのいずれも指定しない場合、ENCRYPTIONのデフォルト値はNONEとなります。

用途

ダンプ・ファイル・セットに書き込む前にデータを暗号化するかどうかを指定します。

構文および説明

```
ENCRYPTION = [ALL | DATA_ONLY | ENCRYPTED_COLUMNS_ONLY | METADATA_ONLY | NONE]
```

- ALLを指定すると、エクスポート操作ですべてのデータおよびメタデータについて暗号化が有効になります。
- DATA_ONLYを指定すると、データのみが暗号化形式でダンプ・ファイル・セットに書き込まれます。
- ENCRYPTED_COLUMNS_ONLYを指定すると、暗号化された列のみが暗号化形式でダンプ・ファイル・セットに書き込まれます。列にはすでに割り当てられた暗号化形式があり、定義によって列が保持できる暗号化形式は1つのみであるため、このオプションをENCRYPTION_ALGORITHMパラメータとともに使用することはできません。

  ENCRYPTED_COLUMNS_ONLYオプションを使用するには、ENCRYPTION_PASSWORDパラメータも使用する必要があります。

  ENCRYPTED_COLUMNS_ONLYオプションを使用するには、Oracle Advanced Securityのデータの透過的な暗号化(TDE)を有効にする必要があります。TDEの詳細は、『*Oracle Database Advanced Security*ガイド』を参照してください。
- METADATA_ONLYを指定すると、メタデータのみが暗号化形式でダンプ・ファイル・セットに書き込まれます。
- NONEを指定すると、データは暗号化形式でダンプ・ファイル・セットに書き込まれません。

暗号化に関するSecureFilesの考慮点

エクスポート対象のデータに暗号化の必要なSecureFilesが含まれている場合は、ENCRYPTION=ALLを指定してダンプ・ファイル・セット全体を暗号化する必要があります。ダンプ・ファイル・セット全体の暗号化は、データ・ポンプ・エクスポート操作中にSecureFilesの暗号化セキュリティを実現する唯一の方法です。SecureFilesの詳細は、『*Oracle Database SecureFiles*およびラージ・オブジェクト開発者ガイド』を参照してください。

暗号化に関するOracle Database Vaultの考慮点

エクスポート操作が起動されると、データ・ポンプによって、Oracle Database Vaultが有効であるかどうかが確認されます。それが有効であり、ダンプ・ファイルの暗号化がジョブに指定されていない場合、セキュアなデータがセキュアではない方法(クリア・テキスト)でダンプ・ファイル・セットに書き込まれていることを示す警告メッセージが返されます。

```
ORA-39327: Oracle Database Vault data is being stored unencrypted in dump file set
```

現在のエクスポート操作を停止し、出力ダンプ・ファイル・セットを暗号化するように指定して新しい操作を開始できます。

制限事項

- ALL、DATA_ONLYまたはMETADATA_ONLYの各オプションを指定するには、COMPATIBLE初期化パラメータを11.0.0以上に設定する必要があります。
- このパラメータは、Oracle Database 11g以上のEnterprise Editionでのみ有効です。
- 暗号化パスワードを使用せずにALL、DATA_ONLYまたはMETADATA_ONLYオプションを使用する場合、Oracle Advanced Securityオプションを有効にする必要があります。Oracle Advanced Securityオプションのライセンス要件の詳細は、『*Oracle Database*ライセンス情報』を参照してください。

例

次の例では、ダンプ・ファイルでデータのみが暗号化されるエクスポート操作を実行します。

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=hr_enc.dmp JOB_NAME=enc1
ENCRYPTION=data_only ENCRYPTION_PASSWORD=foobar
```

関連トピック

- 『Oracle Databaseセキュリティ・ガイド』
- SecureFiles LOB記憶域
- Oracle Databaseのオプションおよびそれらに許可されている機能

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.14 ENCRYPTION_ALGORITHM

Oracle Data Pump Exportのコマンドライン・ユーティリティのENCRYPTION_ALGORITHMパラメータでは、暗号化の実行に使用する必要のある暗号化アルゴリズムを指定します。

デフォルト

AES128

用途

暗号化の実行に使用する必要のある暗号化アルゴリズムを指定します。

構文および説明

```
ENCRYPTION_ALGORITHM = [AES128 | AES192 | AES256]
```

制限事項

- この暗号化機能を使用するには、COMPATIBLE初期化パラメータを11.0.0以上に設定する必要があります。
- ENCRYPTION_ALGORITHMパラメータを使用する場合は、ENCRYPTIONまたはENCRYPTION_PASSWORDパラメータも指定する必要があります。指定しないと、エラーが返されます。
- すでに暗号化されている列に追加の暗号化形式を割り当てることはできないため、ENCRYPTION_ALGORITHMパラ

メータをENCRYPTION=ENCRYPTED_COLUMNS_ONLYと組み合せて使用することはできません。

- このパラメータは、Oracle Database 11g以上のEnterprise Editionでのみ有効です。
- ENCRYPTION _ALGORITHMパラメータでは、Oracle Advanced Securityを有効にする必要はありませんが、そのオプションを必要とするその他の暗号化関連のパラメータとともに使用できます。Oracle Advanced Securityオプションのライセンス要件の詳細は、『*Oracle Database*ライセンス情報』を参照してください。

例

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=hr_enc3.dmp
ENCRYPTION_PASSWORD=foobar ENCRYPTION_ALGORITHM=AES128
```

関連トピック

- 『Oracle Databaseセキュリティ・ガイド』
- Oracle Databaseライセンス情報ユーザー・マニュアル

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.15 ENCRYPTION_MODE

Oracle Data Pump Exportのコマンドライン・ユーティリティのENCRYPTION_MODEパラメータでは、暗号化および復号化を実行する際に使用するセキュリティ・タイプを指定します。

デフォルト

デフォルト・モードは、使用されるその他の暗号化関連パラメータによって異なります。ENCRYPTIONパラメータのみが指定され、Oracle暗号化ウォレットがオープン状態の場合、デフォルト・モードはTRANSPARENTになります。ENCRYPTIONパラメータのみが指定され、ウォレットが閉じられている場合は、エラーが返されます。

ENCRYPTION_PASSWORDパラメータが指定され、ウォレットがオープン状態の場合、デフォルト設定はDUALになります。ENCRYPTION_PASSWORDパラメータが指定され、ウォレットが閉じられている場合、デフォルト設定はPASSWORDになります。

用途

暗号化および復号化を実行する際に使用するセキュリティ・タイプを指定します。

構文および説明

```
ENCRYPTION_MODE = [DUAL | PASSWORD | TRANSPARENT]
```

DUALモードでは、後から透過的にインポートしたり、デュアルモードで暗号化されたダンプ・ファイル・セットの作成時に使用したパスワードを指定してインポートすることができるダンプ・ファイル・セットを作成します。DUALモードで作成されたダンプ・ファイル・セットを後からインポートするときは、ウォレットまたはENCRYPTION_PASSWORDパラメータで指定されたパスワードのいずれかを使用できます。DUALモードは、ウォレットを使用する場合にはダンプ・ファイル・セットのインポートをオンサイトで行い、ウォレットが使用できない場合にはオフサイトでインポートを行う必要もある場合に最適です。

PASSWORDモードでは、暗号化されたダンプ・ファイル・セットの作成時にパスワードを指定する必要があります。ダンプ・ファイル・セットをインポートするときは同じパスワードを指定する必要があります。PASSWORDモードでは、ENCRYPTION_PASSWORDパラメータを指定する必要もあります。PASSWORDモードは、ダンプ・ファイル・セットが別のデータベースやリモート・データベースにインポートされるときに、転送中もセキュリティで保護する必要がある場合に最適です。

TRANSPARENTモードでは、必要なウォレットを使用できる場合、暗号化されるダンプ・ファイル・セットをデータベース管理者(DBA)の介入なしで作成できます。したがって、ENCRYPTION_PASSWORDパラメータは必要ありません。実際、このパラメー

タをTRANSPARENTモードで使用すると、エラーが発生します。この暗号化モードは、ダンプ・ファイル・セットをエクスポート元と同じデータベースにインポートする場合に最適です。

制限事項

- DUALまたはTRANSPARENTモードを使用するには、COMPATIBLE初期化パラメータを11.0.0以上に設定する必要があります。
- ENCRYPTION_MODEパラメータを使用する場合は、ENCRYPTIONまたはENCRYPTION_PASSWORDパラメータのいずれかを使用する必要もあります。そうでない場合は、エラーが戻されます。
- ENCRYPTION=ENCRYPTED_COLUMNS_ONLYを使用する場合は、ENCRYPTION_MODEパラメータを使用できません。そうでない場合は、エラーが戻されます。
- このパラメータは、Oracle Database 11g以上のEnterprise Editionでのみ有効です。
- DUALモードまたはTRANSPARENTモードを使用するには、Oracle Advanced Securityオプションが有効である必要があります。Oracle Advanced Securityオプションのライセンス要件の詳細は、『*Oracle Database*ライセンス情報』を参照してください。

例

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=hr_enc4.dmp
ENCRYPTION=all ENCRYPTION_PASSWORD=secretwords
ENCRYPTION_ALGORITHM=AES256 ENCRYPTION_MODE=DUAL
```

関連トピック

- Oracle Databaseライセンス情報ユーザー・マニュアル

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.16 ENCRYPTION_PASSWORD

Oracle Data Pump Exportのコマンドライン・ユーティリティのENCRYPTION_PASSWORDパラメータでは、エクスポート・ダンプ・ファイル内の暗号化列のデータ、メタデータまたは表データを暗号化するためのパスワードを指定します。このパラメータにより、暗号化されたダンプ・ファイル・セットへの不正なアクセスを防ぎます。

デフォルト: デフォルト値は設定されていません。ユーザーが値を指定します。

用途

エクスポート・ダンプ・ファイル内の暗号化列のデータ、メタデータまたは表データを暗号化するためのパスワードを指定します。このパラメータを使用することにより、暗号化されたダンプ・ファイル・セットへの不正なアクセスを防ぎます。

ノート:

Oracle Data Pump の暗号化機能は、Oracle Database 11g リリース 1 (11.1)で変更されています。リリース 11.1 より前は、ENCRYPTION_PASSWORD パラメータが適用されるのは暗号化列のみでした。しかし、リリース 11.1 では、新しい ENCRYPTION パラメータが他のタイプのデータを暗号化するためのオプションを提供するようになりました。この変更により、ENCRYPTION と固有のオプションを指定せずに ENCRYPTION_PASSWORD を指定すると、ダンプ・ファイルに書き込まれるすべてのデータが暗号化されるようになりました(ENCRYPTION=ALL を指定した場合と同じ)。暗号化列のみを再暗号化するには、ENCRYPTION_PASSWORD に加えて

ENCRYPTION=ENCRYPTED_COLUMNS_ONLY を指定する必要があります。

構文および説明

```
ENCRYPTION_PASSWORD = password
```

指定するpassword値は、暗号化された表の列、メタデータまたは表データをダンプ・ファイル・セットにクリア・テキストとして書き込まないように、再度暗号化するキーを指定します。エクスポート操作に暗号化された表の列が含まれていて、暗号化パスワードが指定されていない場合、暗号化された列はダンプ・ファイル・セットにクリア・テキストで書き込まれ、警告が発行されます。

入力したパスワードは画面に表示されます。パスワードを入力時に画面に表示しない場合、ENCRYPTION_PWD_PROMPTパラメータを使用します。

暗号化パスワードに許容される最大長は、通常は128バイトです。ただし、ENCRYPTION=ENCRYPTED_COLUMNS_ONLYであり、VERSIONパラメータまたはデータベース互換性のいずれかが12.2未満に設定されている場合、制限は30バイトになります。

エクスポート操作については、ENCRYPTION_MODEパラメータがPASSWORDまたはDUALに設定されている場合に、このパラメータが必要になります。

ノート:

データ・ポンプの ENCRYPTION_PASSWORD パラメータで指定されるキーと、暗号化された列が含まれる表が最初に作成されるときに ENCRYPT キーワードで指定されるキーとの間には、関連性も依存性もありません。たとえば、次のように、キーが xyz の暗号化列を持つ表を作成するとします。

```
CREATE TABLE emp (col1 VARCHAR2(256) ENCRYPT IDENTIFIED BY "xyz");
```

emp 表をエクスポートするときは、ENCRYPTION_PASSWORD に任意の値を指定できます。xyz である必要はありません。

制限事項

- このパラメータは、Oracle Database 11g以上のEnterprise Editionでのみ有効です。
- 全体トランスポータブル・エクスポートで、暗号化された表領域および暗号化された列が存在する表を含む表領域を転送する場合、ENCRYPTION_PASSWORDパラメータが必要です。
- Oracle Data Pumpの暗号化機能を使用するには、Oracle Advanced Securityオプションが有効である必要があります。Oracle Advanced Securityオプションのライセンス要件の詳細は、『*Oracle Database*ライセンス情報』を参照してください。
- 全体トランスポータブル・エクスポートで、暗号化された表領域および暗号化された列が存在する表を含む表領域を転送する場合、ENCRYPTION_PASSWORDパラメータが必要です。
- ENCRYPTION_PASSWORDを指定して、ENCRYPTION_MODEを指定しない場合は、ENCRYPTION_MODEのデフォルトがPASSWORDになるため、Oracle Advanced Securityのデータの透過的データ暗号化を有効にする必要はありません。
- 要求された暗号化モードがTRANSPARENTの場合、ENCRYPTION_PASSWORDパラメータは無効です。
- ENCRYPTION_MODEがDUALに設定されている場合、ENCRYPTION_PASSWORDパラメータを使用するには、

Oracle Advanced Securityの透過的データ暗号化(TDE)を有効にする必要があります。TDEの詳細は、『*Oracle Database Advanced Security*ガイド』を参照してください。

- ネットワーク・エクスポートの場合、ENCRYPTION_PASSWORDパラメータとENCRYPTION=ENCRYPTED_COLUMNS_ONLYの併用は、暗号化された列を含むユーザー定義の外部表ではサポートされていません。表はスキップされ、エラー・メッセージが表示されますが、ジョブは続行します。
- すべての列に対する暗号化属性は、エクスポートされた表の定義とターゲット表で一致している必要があります。たとえば、EMP表にEMPNOという列があるとします。次のいずれのシナリオも、ソース表のEMP列の暗号化属性が、ターゲット表のEMP列の暗号化属性と一致していないためエラーになります。
    - EMPNO列を暗号化してEMP表をエクスポートし、表をインポートする前にEMPNO列から暗号化属性を削除する。
    - EMPNO列を暗号化せずにEMP表をエクスポートし、表をインポートする前にEMPNO列の暗号化を有効にする。

例

次の例では、暗号化パスワード123456がダンプ・ファイルdpcd2be1.dmpに割り当てられています。

```
> expdp hr TABLES=employee_s_encrypt DIRECTORY=dpump_dir1
DUMPFILE=dpcd2be1.dmp ENCRYPTION=ENCRYPTED_COLUMNS_ONLY
ENCRYPTION_PASSWORD=123456
```

employee_s_encrypt表の暗号化列は、dpcd2be1.dmpダンプ・ファイルにクリア・テキストとして書き込まれません。その後、この例で作成したdpcd2be1.dmpファイルをインポートする場合は、同じ暗号化パスワードを指定する必要があります。

関連トピック

- Oracle Databaseライセンス情報ユーザー・マニュアル
- Oracle Database Advanced Securityガイド

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.17 ENCRYPTION_PWD_PROMPT

Oracle Data Pump Exportのコマンドライン・ユーティリティのENCRYPTION_PWD_PROMPTでは、Oracle Data Pumpによって暗号化パスワードの入力を要求するかどうかを指定します。

デフォルト

NO

用途

データ・ポンプによって暗号化パスワードの入力を要求するかどうかを指定します。

構文および説明

```
ENCRYPTION_PWD_PROMPT=[YES | NO]
```

コマンドラインでENCRYPTION_PWD_PROMPT=YESと指定すると、ENCRYPTION_PASSWORDパラメータを使用してコマンドラインでパスワードを入力するかわりに、データ・ポンプによって暗号化パスワードの入力を要求されます。この方法のメリットは、プロンプトで入力すると、暗号化パスワードが画面に表示されないことです。これに対し、ENCRYPTION_PASSWORDパラメータを使用してコマンドラインで入力すると、それがプレーン・テキストで表示されます。

プロンプトで入力する暗号化パスワードは、ENCRYPTION_PASSWORDパラメータで説明されているものと同じ基準に従います。

エクスポート操作で暗号化パスワードを指定した場合、インポート操作でもそれを指定する必要があります。

制限事項

- ENCRYPTION_PWD_PROMPTパラメータとENCRYPTION_PASSWORDパラメータの同時使用は禁止されています。

例

次の構文例では、データ・ポンプによって最初にユーザー・パスワードが要求され、次に暗号化パスワードが要求されます。

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=hr.dmp ENCRYPTION_PWD_PROMPT=YES
.
.
.
Copyright (c) 1982, 2017, Oracle and/or its affiliates.  All rights reserved.
Password:

Connected to: Oracle Database 18c Enterprise Edition Release 18.0.0.0.0 - Production
Version 18.1.0.0.0

Encryption Password:
Starting "HR"."SYS_EXPORT_SCHEMA_01":  hr/******** directory=dpump_dir1
dumpfile=hr.dmp encryption_pwd_prompt=Y
.
.
.
```

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.18 ESTIMATE

Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのESTIMATEパラメータでは、エクスポート・ジョブ内の各表で使用されるディスク領域(バイト単位)を見積もる場合にエクスポートで使用される方法を指定します。

デフォルト

STATISTICS

用途

エクスポート・ジョブ内の各表で使用されるディスク領域(バイト単位)を見積もる場合にエクスポートで使用される方法を指定します。見積りはログ・ファイルに出力され、クライアントの標準出力デバイスに表示されます。見積りの対象は、表の行データのみです。メタデータは含まれません。

構文および説明

```
ESTIMATE=[BLOCKS | STATISTICS]
```

- BLOCKS - 見積りは、ソース・オブジェクトで使用されるデータベース・ブロックの数に、適切なブロック・サイズを掛けて計算されます。
- STATISTICS - 見積りは、表別の統計を使用して計算されます。この方法による見積りをできるかぎり正確にするには、すべての表を新しく分析しておく必要があります。(表の分析は、SQL ANALYZE文またはDBMS_STATS PL/SQLパッケージのいずれかで実行可能です。)

制限事項

- 圧縮された表がデータ・ポンプのエクスポート・ジョブに含まれている場合、ESTIMATE=BLOCKSを使用すると、その圧縮表に対するデフォルト・サイズの見積りは不正確になります。この不正確性は、データが圧縮形式で格納されていることが、サイズの見積もりに反映されていないために発生します。圧縮された表に対するサイズ見積りをより正確に行う

には、ESTIMATE=STATISTICSを使用してください。

- QUERYまたはREMAP_DATAのいずれかのパラメータが使用されている場合も、見積りが不正確になる可能性があります。

例

次の例に、ESTIMATEパラメータの使用例を示します。ここでは、employees表に対する統計を使用して見積りが計算されます。

```
> expdp hr TABLES=employees ESTIMATE=STATISTICS DIRECTORY=dpump_dir1
 DUMPFILE=estimate_stat.dmp
```

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.19 ESTIMATE_ONLY

Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのESTIMATE_ONLYパラメータでは、エクスポートによって、実際のエクスポート操作を実行することなく、ジョブが消費する領域が見積もられるように指定します。

デフォルト

NO

用途

エクスポートによって、実際のエクスポート操作を実行することなく、ジョブが消費する領域が見積もられるように指定します。

構文および説明

```
ESTIMATE_ONLY=[YES | NO]
```

ESTIMATE_ONLY=YESの場合、エクスポート・ユーティリティは、使用される領域の見積りを行いますが、実際にエクスポート操作は実行せずに終了します。

制限事項

- ESTIMATE_ONLYパラメータは、QUERYパラメータと組み合せて使用することはできません。

例

次に、ESTIMATE_ONLYパラメータを使用して、HRスキーマのエクスポートに必要な領域の大きさを指定する例を示します。

```
> expdp hr ESTIMATE_ONLY=YES NOLOGFILE=YES SCHEMAS=HR
```

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.20 EXCLUDE

データ・ポンプ・エクスポートのコマンドライン・ユーティリティのEXCLUDEパラメータを使用すると、エクスポート操作から除外するオブジェクトおよびオブジェクト型を指定して、エクスポート対象のメタデータをフィルタ処理できます。

デフォルト: デフォルト値は設定されていません。

用途

エクスポート操作から除外するオブジェクトおよびオブジェクト型を指定して、エクスポート対象のメタデータをフィルタ処理できます。

構文および説明

```
EXCLUDE=object_type[:name_clause] [, ...]
```

object_typeでは、除外するオブジェクトの型を指定します。DATABASE_EXPORT_OBJECTS(全体モードの場合)、SCHEMA_EXPORT_OBJECTS(スキーマ・モードの場合)、TABLE_EXPORT_OBJECTS(表および表領域モードの場合)ビューを問い合せて、object_typeの有効な値の一覧を表示できます。OBJECT_PATH列にリストされる値が有効なオブジェクト・タイプです。

EXCLUDE文に指定されたオブジェクト型を除き、指定されたエクスポート・モードに対するすべてのオブジェクト型が、エクスポートに含まれます。オブジェクトが除外される場合、すべての依存オブジェクトも除外されます。たとえば、表を除外すると、その表のすべての索引およびトリガーも除外されます。

name_clauseは、オプションです。このパラメータを使用すると、あるオブジェクト型のうち、特定のオブジェクトを選択できます。その型のオブジェクト名に対するフィルタとして使用されるSQL式です。SQL演算子および指定した型のオブジェクト名の比較対象となる値で構成されています。このname_clauseは、名前付きのインスタンスを持つオブジェクト型にのみ適用されます(たとえば、TABLEには適用されますが、GRANTには適用されません)。それは、コロンでオブジェクト型と区切り、二重引用符(一重引用符は名前文字列を区切るために必要なため)で囲む必要があります。たとえば、EXCLUDE=INDEX:"LIKE 'EMP%'"と設定した場合、EMPで始まる名前を持つすべての索引を除外できます。

name_clauseに指定する名前は、データベース内の既存のオブジェクトに、大文字と小文字も含めて完全に一致させる必要があります。たとえば、指定するname_clauseがEMPLOYEESという表に対するものだった場合は、すべてが大文字のEMPLOYEESという表が存在する必要があります。name_clauseに、Employees、employeesまたは既存の表と一致しないその他のバリエーションを指定した場合、表は検出されません。

name_clauseを指定しない場合、指定した型のすべてのオブジェクトが除外されます。

複数のEXCLUDE文を指定できます。

オペレーティング・システムによっては、このパラメータの値を引用符を使用して指定する場合、エスケープ文字を使用することも必要になる可能性があります。このパラメータはパラメータ・ファイルで指定することをお薦めします。パラメータ・ファイルを使用すると、コマンドラインでは必要なエスケープ文字の数を減らすことができます。

object_typeにCONSTRAINT、GRANTまたはUSERを指定する場合、次に説明する影響があることに注意してください。

制約の除外

次の制約は明示的に除外できません。

- 表の作成とロードを正常に行うために必要な制約。たとえば、索引構成表の主キー制約、REF列を持つ表のREF SCOPEおよびWITH ROWID制約など

たとえば、次のEXCLUDE文は、次のように解釈されます。

- EXCLUDE=CONSTRAINTでは、表の正常な作成およびロードに必要な制約を除く、すべての制約を除外します。
- EXCLUDE=REF_CONSTRAINTは、参照整合性(外部キー)制約を除外します。

権限とユーザーの除外

EXCLUDE=GRANTを指定すると、すべてのオブジェクト型に対するオブジェクト権限およびシステム権限が除外されます。

EXCLUDE=USERを指定すると、ユーザーの定義のみが除外され、そのユーザーのスキーマ内のオブジェクトは除外されません。

特定のユーザーとそのユーザーのすべてのオブジェクトを除外するには、次のコマンドを指定します(hrは除外するユーザーのスキーマ名です)。

```
expdp FULL=YES DUMPFILE=expfull.dmp EXCLUDE=SCHEMA:"='HR'"
```

この例では、エクスポート・モードのFULLが指定されています。モードを指定しない場合は、デフォルトのモードが使用されます。デフォルト・モードは、SCHEMASです。ただし、デフォルト・モードを使用する場合、この例では、SCHEMASを指定することで、コマンドによりスキーマのエクスポートと除外を同時に指定することになるため、エラーが発生します。

EXCLUDE=USER:"='HR'"のような文を使用してユーザーを除外しようとすると、CREATE USER hr DDL文内で使用される情報のみが除外され、予期しない結果になる場合があります。

制限事項

- EXCLUDEおよびINCLUDEは、相互に排他的なパラメータです。

例

次に、EXCLUDE文の使用例を示します。

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=hr_exclude.dmp EXCLUDE=VIEW,
PACKAGE, FUNCTION
```

これにより、この例では、hrスキーマ全体がエクスポートされるスキーマ・モード・エクスポート(デフォルトのエクスポート・モード)が実行されます。ただし、このスキーマのビュー、パッケージおよびファンクションは除外されます。

関連トピック

- メタデータ・フィルタ
- エクスポート操作中のフィルタ処理
- INCLUDE
- データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.21 FILESIZE

Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのFILESIZEパラメータでは、各ダンプ・ファイルの最大サイズを指定します。

デフォルト

0 (最大サイズの16TBと同じ)

用途

各ダンプ・ファイルの最大サイズを指定します。ダンプ・ファイル・セット内にあるダンプ・ファイルが最大サイズになると、そのファイルは閉じられ、ファイル指定に置換変数が含まれている場合または追加ダンプ・ファイルがジョブに追加されている場合は、新しいファイルが作成されます。

構文および説明

```
FILESIZE=integer[B | KB | MB | GB | TB]
```

integerの後に(スペースを挿入しない)、B、KB、MB、GBまたはTB(それぞれバイト、キロバイト、メガバイト、ギガバイト、テラバイトを示す)を指定できます。デフォルトは、B(バイト)です。作成されるファイルの実際のサイズは、ダンプ・ファイル内で使用されている内部ブロックのサイズと一致するように切り捨てられる場合があります。

制限事項

- ファイルの最小サイズは、デフォルトのデータ・ポンプ・ブロック・サイズの10倍、つまり4KBです。

- ファイルの最大サイズは16TBです。

例

次の例は、ダンプ・ファイルのサイズを3MBに設定する方法を示しています。

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=hr_3m.dmp FILESIZE=3MB
```

このシナリオでは、割り当てられた3MBではすべてのエクスポート・データを保持するのに十分でなかった場合、次のエラーが発生して表示されジョブが停止します。

```
ORA-39095: Dump file space has been exhausted: Unable to allocate 217088 bytes
```

割り当てることができなかった実際のバイト数は、場合によって異なります。また、この数字は、エクスポート操作全体を完了するために必要な容量を表しているわけではありません。これは、ジョブがダンプ・ファイルの領域を使い果たした時点でエクスポートされていたオブジェクトのサイズのみを示します。この問題を修正するには、最初に停止したジョブに接続し、ADD_FILEコマンドを使用して1つ以上のファイルを追加した後、操作を再開します。

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.22 FLASHBACK_SCN

Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのFLASHBACK_SCNパラメータでは、エクスポートで使用されるシステム変更番号(SCN)を指定して、フラッシュバック問合せユーティリティを使用可能にします。

デフォルト: デフォルト値は設定されていません。

用途

エクスポートで使用されるシステム変更番号(SCN)を指定して、フラッシュバック問合せユーティリティを使用可能にします。

構文および説明

```
FLASHBACK_SCN=scn_value
```

エクスポート操作は、指定したSCNまでの一貫性のあるデータを使用して実行されます。NETWORK_LINKパラメータが指定されている場合、SCNはソース・データベースのSCNを示します。

Oracle Database 12cリリース2 (12.2)からは、SCN値をビッグSCN (8バイト)にすることができます。また、ビッグSCNをサポートしていない以前のバージョンでダンプ・ファイルを作成するときも、実際のSCN値は移動されないため、ビッグSCNを指定できます。

制限事項

- FLASHBACK_SCNおよびFLASHBACK_TIMEは、相互に排他的なパラメータです。
- FLASHBACK_SCNパラメータは、Oracle Databaseのフラッシュバック問合せ機能にのみ関係します。フラッシュバック・データベース、フラッシュバック削除およびフラッシュバック・データ・アーカイブには適用できません。
- ビッグSCNをサポートしていないバージョンからのネットワーク・エクスポートまたはネットワーク・インポートに対して、ビッグSCNを指定することはできません。

例

次の例では、384632 というSCN値が存在するとします。この例では、hrスキーマをSCN 384632までエクスポートします。

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=hr_scn.dmp FLASHBACK_SCN=384632
```

ノート:

ロジカル・スタンバイ・システムにおいてロジカル・スタンバイのプライマリへのアクセスにネットワーク・リンクを使用する場合は、ロジカル・スタンバイによって SCN が選択されるため、FLASHBACK_SCN パラメータは無視されます。ロジカル・スタンバイ・データベースの詳細は、『*Oracle Data Guard* 概要および管理』を参照してください。

関連トピック

- ロジカル・スタンバイ・データベース

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.23 FLASHBACK_TIME

データ・ポンプ・エクスポートのコマンドライン・ユーティリティのFLASHBACK_TIMEパラメータでは、指定した時間に最も近いSCNを検索します。このSCNを使用して、フラッシュバック・ユーティリティを使用可能にします。エクスポート操作は、このSCNまでの一貫性のあるデータを使用して実行されます。

デフォルト: デフォルト値は設定されていません。

用途

指定した時間に最も近いSCNを検索します。このSCNを使用して、フラッシュバック・ユーティリティを使用可能にします。エクスポート操作は、このSCNまでの一貫性のあるデータを使用して実行されます。

構文および説明

```
FLASHBACK_TIME="TO_TIMESTAMP(time-value)"
```

TO_TIMESTAMPの値は引用符で囲まれるため、パラメータ・ファイルに記述することをお薦めします。

または、次のパラメータ設定を入力することもできます。この設定により、現在のシステム時間に基づいた一貫性エクスポートが開始されます。

```
FLASHBACK_TIME=systimestamp
```

制限事項

- FLASHBACK_TIMEおよびFLASHBACK_SCNは、相互に排他的なパラメータです。
- FLASHBACK_TIMEパラメータは、Oracle Databaseのフラッシュバック問合せ機能にのみ関係します。フラッシュバック・データベース、フラッシュバック削除およびフラッシュバック・データ・アーカイブには適用できません。

例

DBMS_FLASHBACK.ENABLE_AT_TIMEプロシージャで使用可能な形式で時刻を指定できます。たとえば、次の内容のパラメータ・ファイルflashback.parを作成したとします。

```
DIRECTORY=dpump_dir1
DUMPFILE=hr_time.dmp
FLASHBACK_TIME="TO_TIMESTAMP('27-10-2012 13:16:00', 'DD-MM-YYYY HH24:MI:SS')"
```

次のコマンドを発行します。

```
> expdp hr PARFILE=flashback.par
```

エクスポート操作は、指定した時間に最も近いSCNと整合性のあるデータで実行されます。

ノート:

ロジカル・スタンバイ・システムにおいてロジカル・スタンバイのプライマリへのアクセスにネットワーク・リンクを使用する場合は、ロジカル・スタンバイによって SCN が選択されるため、FLASHBACK_SCN パラメータは無視されます。ロジカル・スタンバイ・データベースの詳細は、『*Oracle Data Guard* 概要および管理』を参照してください。

フラッシュバック問合せの使用方法の詳細は、*Oracle Database* 開発ガイドを参照してください。

関連トピック

- データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ
- 『Oracle Data Guard概要および管理』
- Oracle Database開発ガイド

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.24 FULL

エクスポート・コマンドラインFULLパラメータでは、全体データベース・モード・エクスポートを実行することを指定します

デフォルト: NO

用途

全体データベース・モード・エクスポートの実行を指定します。

構文および説明

```
FULL=[YES | NO]
```

FULL=YESと指定すると、すべてのデータおよびメタデータがエクスポートされます。全体エクスポートを実行するには、DATAPUMP_EXP_FULL_DATABASEロールが必要です。

このエクスポート・モードを使用したエクスポート対象を、フィルタ処理によって制限できます。

トランスポータブル・オプション(TRANSPORTABLE=ALWAYS)を使用して全体モード・エクスポートを実行できます。これは、全体トランスポータブル・エクスポートと呼ばれ、データベースの完全なコピーを作成するために必要なすべてのオブジェクトおよびデータがエクスポートされます。全体モードを参照してください。

ノート:

全体モード・エクスポートによって作成されたダンプ・ファイルを後でインポートする場合、インポート操作は SYS アカウントのパスワードをソース・データベースからコピーしようとすることに注意してください。これは、失敗する場合があります(パスワードが共有パスワード・ファイル内にある場合など)。失敗した場合は、インポートの完了後に、ターゲット・データベース上の SYS アカウントのパスワードを任意のパスワードに設定する必要があります。

制限事項

- FULLパラメータをTRANSPORTABLE (全体トランスポータブル・エクスポート)と組み合せて使用するには、データ・ポンプのVERSIONパラメータを12.0以上に設定するか、COMPATIBLEデータベース初期化パラメータを12.0以上に設

定する必要があります。

- 全体エクスポートでは、Oracleが管理するデータおよびメタデータを含むシステム・スキーマがデフォルトでエクスポートされません。デフォルトでエクスポートされないシステム・スキーマの例としては、SYS、ORDSYS、MDSYSなどがあります。
- SYSスキーマが所有しているオブジェクトに対する権限はエクスポートされません。
- 全体エクスポート操作では、1つのデータベース・エディションのみからオブジェクトがエクスポートされ、デフォルトでは現在のエディションがエクスポートされますが、エクスポートのSOURCE_EDITIONパラメータを使用して別のエディションを指定することもできます。
- レルムにより保護されているデータをエクスポートする場合は、そのレルムに対する権限が必要です。
- 自動ワークロード・リポジトリ(AWR)は、全データベースのエクスポートおよびインポート操作では移動されません。(データ・ポンプを使用してAWRスナップショットを移動する方法の詳細は、『*Oracle Database*パフォーマンス・チューニング・ガイド』を参照してください。)
- XDBリポジトリは、全データベースのエクスポートおよびインポート操作では移動されません。ユーザーが作成したXMLスキーマは移動されます。

例

次に、FULLパラメータの使用例を示します。ダンプ・ファイルexpfull.dmpは、dpump_dir2ディレクトリに書き込まれます。

```
> expdp hr DIRECTORY=dpump_dir2 DUMPFILE=expfull.dmp FULL=YES NOLOGFILE=YES
```

全体トランスポータブル・エクスポートの実行方法の詳細な例は、*Oracle Database*管理者ガイドを参照してください。レルムの構成方法の詳細は、*Oracle Database Vault*管理者ガイドを参照してください。

関連トピック

- Oracle Databaseパフォーマンス・チューニング・ガイド
- Oracle Database管理者ガイド
- Oracle Database Vaultの管理者ガイド

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.25 HELP

データ・ポンプ・エクスポートのコマンドライン・ユーティリティのHELPパラメータでは、エクスポート・ユーティリティのオンライン・ヘルプを表示します。

デフォルト: NO

用途

エクスポート・ユーティリティのオンライン・ヘルプを表示します。

構文および説明

```
HELP = [YES | NO]
```

HELP=YESが指定されている場合は、エクスポート・ユーティリティによって、そのすべてのコマンドライン・パラメータと対話方式コマンドの要約が表示されます。

例

```
> expdp HELP = YES
```

この例では、すべてのエクスポート・パラメータおよびコマンドの簡単な説明が表示されます。

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.26 INCLUDE

データ・ポンプ・エクスポートのコマンドライン・ユーティリティのINCLUDEパラメータを使用すると、現行のエクスポート・モードにオブジェクトおよびオブジェクト型を指定して、エクスポート対象のメタデータをフィルタ処理できます。指定したオブジェクトおよびこれらのオブジェクトのすべての依存オブジェクトがエクスポートされます。これらのオブジェクトに対する権限もエクスポートされます。

デフォルト: デフォルト値は設定されていません。

用途

現行のエクスポート・モードにオブジェクトとオブジェクト型を指定して、エクスポート対象のメタデータをフィルタ処理できます。指定したオブジェクトおよびこれらのオブジェクトのすべての依存オブジェクトがエクスポートされます。これらのオブジェクトに対する権限もエクスポートされます。

構文および説明

```
INCLUDE = object_type[:name_clause] [, ...]
```

object_typeは含めるオブジェクトのタイプを指定します。DATABASE_EXPORT_OBJECTS(全体モードの場合)、SCHEMA_EXPORT_OBJECTS(スキーマ・モードの場合)、TABLE_EXPORT_OBJECTS(表および表領域モードの場合)ビューを問い合せて、object_typeの有効な値の一覧を表示できます。OBJECT_PATH列にリストされる値が有効なオブジェクト・タイプです。

INCLUDE文で明示的に指定されたオブジェクト型とその依存オブジェクトのみがエクスポートされます。他のオブジェクト型(通常、DATAPUMP_EXP_FULL_DATABASEロールを所有している場合にスキーマ・モード・エクスポートの一部となるスキーマ定義情報など)はエクスポートされません。

name_clauseは、オプションです。これを使用すると、あるオブジェクト型のうち、特定のオブジェクトをファイングレイン選択できます。その型のオブジェクト名に対するフィルタとして使用されるSQL式です。SQL演算子および指定した型のオブジェクト名の比較対象となる値で構成されています。このname_clauseは、名前付きのインスタンスを持つオブジェクト型にのみ適用されます(たとえば、TABLEには適用されますが、GRANTには適用されません)。それは、コロンでオブジェクト型と区切り、二重引用符(一重引用符は名前文字列を区切るために必要なため)で囲む必要があります。

name_clauseに指定する名前は、データベース内の既存のオブジェクトに、大文字と小文字も含めて完全に一致させる必要があります。たとえば、指定するname_clauseがEMPLOYEESという表に対するものだった場合は、すべてが大文字のEMPLOYEESという表が存在する必要があります。name_clauseに、Employees、employeesまたはその他のバリエーションを指定した場合、表は検出されません。

オペレーティング・システムによっては、このパラメータの値を引用符を使用して指定する場合、エスケープ文字を使用することも必要になる可能性があります。このパラメータはパラメータ・ファイルで指定することをお薦めします。パラメータ・ファイルを使用すると、コマンドラインでは入力する必要があるエスケープ文字の数を減らすことができます。

たとえば、次の内容のパラメータ・ファイルhr.parを作成したとします。

```
SCHEMAS=HR
DUMPFILE=expinclude.dmp
DIRECTORY=dpump_dir1
LOGFILE=expinclude.log
INCLUDE=TABLE:"IN ('EMPLOYEES', 'DEPARTMENTS')"
INCLUDE=PROCEDURE
INCLUDE=INDEX:"LIKE 'EMP%'"
```

この場合、コマンドラインで他のパラメータを入力しなくても、hr.parファイルを使用してエクスポート操作を開始できます。EMPLOYEESとDEPARTMENTS表、すべてのプロシージャおよびEMP接頭辞を持つすべての索引名が、エクスポートに含まれます。

```
> expdp hr PARFILE=hr.par
```

制約の追加

指定したobject_typeがCONSTRAINTの場合は、制約の使用による影響に注意してください。

次の制約を明示的に含めることはできません。

- NOT NULL制約
- 表を正常に作成およびロードするために必要な制約。たとえば、索引構成表に対して主キー制約を含めたり、REF列を含む表に対してREF SCOPEおよびWITH ROWID制約を含めることはできません。

たとえば、次のINCLUDE文は、次のように解釈されます。

- INCLUDE=CONSTRAINTでは、NOT NULL制約および表の正常な作成およびロードに必要な制約を除き、すべての制約(非参照)が含まれます。
- INCLUDE=REF_CONSTRAINTでは、参照整合性(外部キー)制約が含まれます。

制限事項

- INCLUDEおよびEXCLUDEは、相互に排他的なパラメータです。
- SYSスキーマが所有しているオブジェクトに対する権限はエクスポートされません。

例

次の例では、hrスキーマのすべての表(およびその依存オブジェクト)をエクスポートします。

```
> expdp hr INCLUDE=TABLE DUMPFILE=dpump_dir1:exp_inc.dmp NOLOGFILE=YES
```

関連トピック

- メタデータ・フィルタ
- データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.27 JOB_NAME

Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのJOB_NAMEパラメータでは、ATTACHを使用してジョブに接続したり、DBA_DATAPUMP_JOBSまたはUSER_DATAPUMP_JOBSビューを使用してジョブを指定する場合など、後続処理でエクスポート・ジョブを指定します。

デフォルト

SYS_EXPORT_<mode>_NNという書式のシステム生成による名前

用途

ジョブへの接続にATTACHパラメータを使用したり、DBA_DATAPUMP_JOBSまたはUSER_DATAPUMP_JOBSビューを使用してジョブを指定する場合など、後続処理でエクスポート・ジョブを指定するために使用されます。

構文および説明

```
JOB_NAME=jobname_string
```

jobname_stringには、このエクスポート・ジョブの名前を、128バイト以内で指定します。これらのバイトは印字可能文字と空白を表します。空白を含む場合は、一重引用符で囲みます(たとえば、'Thursday Export'とします)。ジョブ名は、エクスポート操作を実行しているユーザーのスキーマによって暗黙的に修飾されます。ジョブ名は、エクスポート・ジョブを制御するデータ・ポンプ制御インポート・ジョブ表の名前として使用されます。

デフォルトのジョブ名はSYS_EXPORT_mode_NNという形式で、システムによって生成されます。NNは、01から増加する2桁の整数に変換されます。デフォルト名は、'SYS_EXPORT_TABLESPACE_02'などです。

例

次に、ジョブ名exp_jobを割り当てるエクスポート操作の例を示します。

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=exp_job.dmp JOB_NAME=exp_job
NOLOGFILE=YES
```

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.28 KEEP_MASTER

Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのKEEP_MASTERパラメータは、正常に完了したOracle Data Pumpジョブの最後に、データ・ポンプ制御ジョブ表を削除するか、保持するかを指定します。

デフォルト

NO

用途

正常に完了したOracle Data Pumpジョブの最後に、データ・ポンプ制御ジョブ表を削除するか保持するかを示します。データ・ポンプ制御ジョブ表は、正常に完了しなかったジョブに対して自動的に保持されます。

構文および説明

```
KEEP_MASTER=[YES | NO]
```

制限事項

- なし

例

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=expdat.dmp SCHEMAS=hr KEEP_MASTER=YES
```

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.29 LOGFILE

データ・ポンプ・エクスポートのコマンドライン・ユーティリティのLOGFILEパラメータでは、エクスポート・ジョブのログ・ファイルの名前を指定し、オプションで、そのログ・ファイルのディレクトリを指定します。

デフォルト: export.log

用途

エクスポート・ジョブのログ・ファイルの名前を指定します。また、オプションで、そのログ・ファイルを格納するディレクトリを指定します。

構文および説明

```
LOGFILE=[directory_object:]file_name
```

directory_objectには、DBAによって作成済であるデータベースのディレクトリ・オブジェクトを指定できます(そのオブジェクトへのアクセス権がある場合)。この設定は、DIRECTORYパラメータに指定されたディレクトリ・オブジェクトよりも優先されます。

file_nameには、ログ・ファイル名を指定します。デフォルトでは、DIRECTORYパラメータで指定されたディレクトリ・オブジェクトが示すディレクトリに、export.logというファイルが作成されます。

処理中の作業、完了した作業および発生したエラーに関するすべてのメッセージがログ・ファイルに書き込まれます。(ジョブのリアルタイムの状態を把握するには、対話方式モードでSTATUSコマンドを使用します。)

NOLOGFILEパラメータを指定しないかぎり、エクスポート・ジョブには、常にログ・ファイルが作成されます。ダンプ・ファイル・セットと同様に、ログ・ファイルの基準となるのは、クライアントではなく、サーバーです。

このファイル名と一致する既存のファイルは上書きされます。

制限事項

- Oracle Automatic Storage Management (Oracle ASM)を使用してデータ・ポンプ・エクスポートを実行する場合、LOGFILEパラメータは、Oracle ASMの+表記法を使用せずにディレクトリ・オブジェクトを指定する必要があります。つまり、ログ・ファイルはディスク・ファイルに書き込まれ、Oracle ASMの記憶域には書き込まれません。かわりに、NOLOGFILE=YESを指定することもできます。ただし、NOLOGFILE=YESを指定すると、この設定によりログ・ファイルの書込みは行われません。

例

次の例に、デフォルトの名前を使用しない場合に、ログ・ファイル名を指定する方法を示します。

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=hr.dmp LOGFILE=hr_export.log
```

ノート:

データ・ポンプ・エクスポート・ユーティリティは、データベースの文字セットを使用してログ・ファイルに書込みを行います。クライアントの NLS_LANG 環境設定がデータベースの文字セットと異なる文字セットの場合は、ログ・ファイル内の表の名前が、クライアントの出力画面に表示される名前と異なることがあります。

関連トピック

- STATUS
- Oracle Automatic Storage Managementを使用可能にした場合のディレクトリ・オブジェクトの使用方法

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.30 LOGTIME

Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのLOGTIMEパラメータでは、エクスポート操作中に表示されるメッセージにタイムスタンプを付けることを指定します。

デフォルト

タイムスタンプは記録されません

用途

エクスポート操作中に表示されるメッセージにタイムスタンプを付けることを指定します。タイムスタンプを使用して、データ・ポンプ操作の異なるフェーズ間の経過時間を計算できます。この情報は、パフォーマンス問題を診断する場合や、将来の同じような操作の時間を見積もる場合に役立ちます。

構文および説明

```
LOGTIME=[NONE | STATUS | LOGFILE | ALL]
```

使用可能なオプションの定義は次のとおりです。

- NONE: ステータス・メッセージまたはログ・ファイル・メッセージにタイムスタンプを付けません(デフォルトと同じ)。
- STATUS: ステータス・メッセージにのみタイムスタンプを付けます。
- LOGFILE: ログ・ファイル・メッセージにのみタイムスタンプを付けます。
- ALL: ステータス・メッセージとログ・ファイル・メッセージの両方にタイムスタンプを付けます。

制限事項

なし

例

次の例では、エクスポート操作中に表示されるすべてのステータス・メッセージとログ・ファイル・メッセージにタイムスタンプを記録します。

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=expdat.dmp SCHEMAS=hr LOGTIME=ALL
```

出力は、次のようになります。

```
10-JUL-12 10:12:22.300: Starting "HR"."SYS_EXPORT_SCHEMA_01":  hr/********
directory=dpump_dir1 dumpfile=expdat.dmp schemas=hr logtime=all
10-JUL-12 10:12:22.915: Estimate in progress using BLOCKS method...
10-JUL-12 10:12:24.422: Processing object type SCHEMA_EXPORT/TABLE/TABLE_DATA
10-JUL-12 10:12:24.498: Total estimation using BLOCKS method: 128 KB
10-JUL-12 10:12:24.822: Processing object type SCHEMA_EXPORT/USER
10-JUL-12 10:12:24.902: Processing object type SCHEMA_EXPORT/SYSTEM_GRANT
10-JUL-12 10:12:24.926: Processing object type SCHEMA_EXPORT/ROLE_GRANT
10-JUL-12 10:12:24.948: Processing object type SCHEMA_EXPORT/DEFAULT_ROLE
10-JUL-12 10:12:24.967: Processing object type SCHEMA_EXPORT/TABLESPACE_QUOTA
10-JUL-12 10:12:25.747: Processing object type
SCHEMA_EXPORT/PRE_SCHEMA/PROCACT_SCHEMA
10-JUL-12 10:12:32.762: Processing object type SCHEMA_EXPORT/SEQUENCE/SEQUENCE
10-JUL-12 10:12:46.631: Processing object type SCHEMA_EXPORT/TABLE/TABLE
10-JUL-12 10:12:58.007: Processing object type
SCHEMA_EXPORT/TABLE/GRANT/OWNER_GRANT/OBJECT_GRANT
10-JUL-12 10:12:58.106: Processing object type SCHEMA_EXPORT/TABLE/COMMENT
10-JUL-12 10:12:58.516: Processing object type SCHEMA_EXPORT/PROCEDURE/PROCEDURE
10-JUL-12 10:12:58.630: Processing object type
SCHEMA_EXPORT/PROCEDURE/ALTER_PROCEDURE
10-JUL-12 10:12:59.365: Processing object type SCHEMA_EXPORT/TABLE/INDEX/INDEX
10-JUL-12 10:13:01.066: Processing object type
SCHEMA_EXPORT/TABLE/CONSTRAINT/CONSTRAINT
10-JUL-12 10:13:01.143: Processing object type
SCHEMA_EXPORT/TABLE/INDEX/STATISTICS/INDEX_STATISTICS
10-JUL-12 10:13:02.503: Processing object type SCHEMA_EXPORT/VIEW/VIEW
10-JUL-12 10:13:03.288: Processing object type
SCHEMA_EXPORT/TABLE/CONSTRAINT/REF_CONSTRAINT
10-JUL-12 10:13:04.067: Processing object type SCHEMA_EXPORT/TABLE/TRIGGER
10-JUL-12 10:13:05.251: Processing object type
SCHEMA_EXPORT/TABLE/STATISTICS/TABLE_STATISTICS
10-JUL-12 10:13:06.172: . . exported "HR"."EMPLOYEES"
17.05 KB     107 rows
```

```
10-JUL-12 10:13:06.658: . . exported "HR"."COUNTRIES"
6.429 KB      25 rows
10-JUL-12 10:13:06.691: . . exported "HR"."DEPARTMENTS"
7.093 KB      27 rows
10-JUL-12 10:13:06.723: . . exported "HR"."JOBS"
7.078 KB      19 rows
10-JUL-12 10:13:06.758: . . exported "HR"."JOB_HISTORY"
7.164 KB      10 rows
10-JUL-12 10:13:06.794: . . exported "HR"."LOCATIONS"
8.398 KB      23 rows
10-JUL-12 10:13:06.824: . . exported "HR"."REGIONS"
5.515 KB       4 rows
10-JUL-12 10:13:07.500: Master table "HR"."SYS_EXPORT_SCHEMA_01" successfully
loaded/unloaded
10-JUL-12 10:13:07.503:
******************************************************************************
```

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.31 METRICS

データ・ポンプ・エクスポートのコマンドライン・ユーティリティのMETRICSパラメータでは、ジョブに関する詳細情報をデータ・ポンプ・ログ・ファイルに記録するかどうかを指定します。

デフォルト: NO

用途

ジョブに関する詳細情報をデータ・ポンプ・ログ・ファイルに記録するかどうかを示します。

構文および説明

```
METRICS=[YES | NO]
```

METRICS=YESを使用すると、オブジェクト数および経過時間がデータ・ポンプ・ログ・ファイルに記録されます。

制限事項

- なし

例

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=expdat.dmp SCHEMAS=hr METRICS=YES
```

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.32 NETWORK_LINK

データ・ポンプ・エクスポートのコマンドライン・ユーティリティのNETWORK_LINKパラメータを使用すると、有効なデータベース・リンクで識別される(ソース)データベースからエクスポートできます。ソース・データベース・インスタンスのデータは、接続されたデータベース・インスタンスのダンプ・ファイル・セットに書き込まれます。

デフォルト: デフォルト値は設定されていません。

用途

有効なデータベース・リンクによって指定される(ソース)データベースからのエクスポートを使用可能にします。ソース・データベース・インスタンスのデータは、接続されたデータベース・インスタンスのダンプ・ファイル・セットに書き込まれます。

構文および説明

```
NETWORK_LINK=source_database_link
```

NETWORK_LINKパラメータは、データベース・リンクを使用してエクスポートを開始します。このエクスポート設定では、expdpクライアントの接続先となるシステムから、source_database_linkで指定されたソース・データベースに接続し、そこからデータを取り出して、接続されたシステムのダンプ・ファイル・セットに書き込みます。

source_database_linkには、使用可能なデータベースへのデータベース・リンク名を指定する必要があります。対象インスタンスのデータベースにデータベース・リンクが指定されていない場合、ユーザーまたはDBAが、SQL CREATE DATABASE LINK文を使用してデータベース・リンクを作成する必要があります。

ソース・データベースが読取り専用の場合、ソース・データベースのユーザーは、デフォルト一時表領域として割り当てられたローカル管理一時表領域を持つ必要があります。それ以外の場合、ジョブは失敗します。

データ・ポンプ・エクスポートでは、次のタイプのデータベース・リンクの使用がサポートされています。

- パブリック固定ユーザー
- パブリック接続ユーザー
- パブリック共有ユーザー(リンク所有者により使用される場合のみ)
- プライベート共有ユーザー(リンク所有者により使用される場合のみ)
- プライベート固定ユーザー(リンク所有者により使用される場合のみ)

注意:

暗号化されていないネットワーク・リンクを介してエクスポート操作が行われる場合、すべてのデータはクリア・テキストとしてエクスポートされます。これは、データがデータベースで暗号化されている場合でも同様です。ネットワーク・セキュリティの詳細は、Oracle Database セキュリティ・ガイドを参照してください。

制限事項

- データ・ポンプ・エクスポートでは、次のタイプのデータベース・リンクの使用はサポートされていません。
  - プライベート接続ユーザー
  - 現在のユーザー
- データ・ポンプ操作がネットワーク・リンクを介して行われる場合、ソース・データベースとターゲット・データベースのバージョンの差違が2バージョン以下である必要があります。たとえば、一方のデータベースがOracle Database 12cの場合、他方のデータベースは12c、11gまたは10gである必要があります。データ・ポンプがチェックするのはメジャー・バージョン番号のみ(10g、11g、12cなど)で、具体的なリリース番号(12.1、12.2、11.1、11.2、10.1、10.2など)ではありません。
- 全体トランスポータブル・エクスポートを使用してネットワークを介してデータベースを転送する場合、監査証跡情報自体がユーザー定義の表領域に格納されていると、管理表領域(SYSTEMやSYSAUXなど)に格納された表に対して監査を有効にすることはできません。
- NETWORK_LINKパラメータも使用されていると、メタデータはパラレルでインポートできなくなります

例

次に、NETWORK_LINKパラメータを使用した構文例を示します。変数source_database_linkを、すでに存在する必

要がある有効なデータベース・リンクの名前に置き換えます。

```
> expdp hr DIRECTORY=dpump_dir1 NETWORK_LINK=source_database_link
  DUMPFILE=network_export.dmp LOGFILE=network_export.log
```

関連項目:

- データベース・リンクの詳細は、Oracle Database管理者ガイドを参照
- CREATE DATABASE LINK文の詳細は、Oracle Database SQL言語リファレンスを参照してください
- ローカル管理表領域の詳細は、Oracle Database管理者ガイドを参照してください

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.33 NOLOGFILE

データ・ポンプ・エクスポートのコマンドライン・ユーティリティのNOLOGFILEパラメータでは、ログ・ファイルを作成するかどうかを指定します。

デフォルト: NO

用途

ログ・ファイルを作成するかどうかを指定します。

構文および説明

```
NOLOGFILE=[YES | NO]
```

デフォルトでログ・ファイルを作成しないようにするには、NOLOGFILE=YESを指定します。ただし、進捗とエラーに関する情報は、接続されているいずれかのクライアント(元のエクスポート操作を開始したクライアントを含む)の標準出力デバイスに書き込まれます。実行中のジョブに接続されているクライアントが存在しないときにNOLOGFILE=YESを指定すると、重要な進捗情報およびエラー情報が失われる危険性があります。

例

次に、NOLOGFILEパラメータの使用例を示します。

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=hr.dmp NOLOGFILE=YES
```

このコマンドによって、ログ・ファイルの書込みを行わないスキーマ・モード・エクスポート(デフォルト),が実行されます。

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.34 PARALLEL

Oracle Data Pumpのエクスポート・コマンドライン・ユーティリティのPARALLELパラメータでは、エクスポート・ジョブのために動作するアクティブな実行プロセスの最大数を指定します。

デフォルト

1

用途

エクスポート・ジョブのために動作するアクティブな実行プロセスの最大数を指定します。この実行セットは、ワーカー・プロセスとパ

ラレル入出力(I/O)サーバー・プロセスの組合せで構成されます。パラレル問合せ操作で問合せコーディネータとして機能するデータ・ポンプ制御プロセスおよびワーカー・プロセスは、この合計にカウントされません。

このパラメータを使用して、リソース消費と経過時間のバランスをとることができます。

構文および説明

```
PARALLEL=integer
```

integerに指定する値は、ダンプ・ファイル・セット内のファイル数以下にする必要があります(または、ダンプ・ファイル指定に%Uまたは%L置換変数を指定する必要があります)。アクティブなワーカー・プロセッサI/Oサーバー・プロセスは、1つのファイルに対してそれぞれが同時に排他的に書込みを行うため、ファイル数が不足していると、逆効果になります。たとえば、ファイルの待機中に、一部のワーカー・プロセスがアイドル状態になる可能性があるため、そのジョブの全体的なパフォーマンスが低下します。また、パラレルI/Oサーバー・プロセスを共有で実行しているメンバーが出力用ファイルを取得できない場合は、ORA-39095エラーを返してエクスポート操作が停止します。いずれの場合も、データ・ポンプ・エクスポート・ユーティリティを使用してジョブに接続することによって、問題を解決できます。接続後、対話方式モードで、ADD_FILEコマンドを使用してファイルを追加し、ジョブが停止した場合は、ジョブを再開します。

ジョブの実行中にPARALLELの値を増減するには、対話方式コマンド・モードを使用します。並列度を下げても、ジョブに関連付けられたワーカー・プロセスは減少しません。任意の時点で実行されるワーカー・プロセスの数が減少します。また、プロセス数が減少する前に、継続中の処理が適正な完了ポイントに到達する必要があります。そのため、値を小さくした効果の確認に時間がかかる場合があります。アイドル状態のワーカー・プロセスは、ジョブが終了するまで削除されません。

パラレル実行できる処理が存在する場合、並列度の増加はすぐに反映されます。

Oracle RAC環境でのエクスポートにおけるPARALLELの使用

Oracle Real Application Clusters(Oracle RAC)環境内でPARALLEL=1としてエクスポート操作を実行すると、すべてのOracle Data Pumpプロセスは、ジョブが開始されたインスタンス上に配置されます。そのため、ディレクトリ・オブジェクトは、そのインスタンスのローカル記憶域を示すことができます。

PARALLELを1より大きな値に設定してエクスポート操作を実行する場合は、ジョブが開始されたインスタンス以外のインスタンスにもOracle Data Pumpプロセスを置くことができます。したがって、ディレクトリ・オブジェクトは、すべてのOracle RACクラスタ・メンバーからアクセス可能な共有記憶域を指している必要があります。

制限事項

- このパラメータは、Oracle Database 11g以上のEnterprise Editionでのみ有効です。
- パラレルで(パラレル問合せまたはPQワーカー・プロセスを使用して)表または表パーティションをエクスポートするには、DATAPUMP_EXP_FULL_DATABASEロールが必要です。
- トランスポータブル表領域のメタデータは、パラレルでエクスポートできません。
- NETWORK_LINKパラメータも使用されていると、メタデータはパラレルでエクスポートできなくなります。
- 次のオブジェクトはパラレルでエクスポートできません。
  - TRIGGER
  - VIEW
  - OBJECT_GRANT
  - SEQUENCE
  - CONSTRAINT
  - REF_CONSTRAINT

例

次に、PARALLELパラメータの使用例を示します。

```
> expdp hr DIRECTORY=dpump_dir1 LOGFILE=parallel_export.log
JOB_NAME=par4_job DUMPFILE=par_exp%u.dmp PARALLEL=4
```

この例では、hrスキーマのスキーマ・モード・エクスポート(デフォルトのモード)が実行され、ディレクトリ・オブジェクトdpump_dir1に指定されたパスに、最大で4つのファイルが作成されます。

関連トピック

- DUMPFILE
- データ・ポンプ・エクスポートの対話方式コマンド・モードで使用可能なコマンド
- パラレル全データベース・エクスポートの実行

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.35 PARALLEL_THRESHOLD

Oracle Data PumpのエクスポートコマンドラインユーティリティのPARALLEL_THRESHOLDパラメータでは、表サイズに基づいてデータ・ポンプが潜在的なパラレルDMLを計算するために使用する除数のサイズを指定します。

デフォルト

250MB

用途

PARALLEL_THRESHOLDは、単一の非パーティション表のエクスポート・ジョブまたはインポート・ジョブ、あるいはパーティション表の1つのパーティションでのみ使用する必要があります。ジョブでPARALLELを指定する場合、PARALLEL_THRESHOLDを指定して、インポートおよびエクスポート時にパラレル・データ操作文(PDML)を使用して表をエクスポートまたはインポートする必要があるかどうかをOracle Data Pumpが判断するために使用する除数のサイズを変更できます。デフォルトより小さい値を指定すると、より小さい表サイズでOracle Data Pumpのパラレル・アルゴリズムを使用できます。たとえば、100MBの表があり、PDML 5を使用して5単位に分割する場合は、PARALLEL_THRESHOLD=20Mを指定します。オプティマイザによってSQLに対して生成されるデータベース、オプティマイザおよび実行計画は、ジョブで指定されたオブジェクトのロードまたはアンロードに使用される実際の並列度を決定することに注意してください。

構文および説明

パラメータ値は、しきい値サイズをバイト単位で指定します。

```
PARALLEL_THRESHOLD=size-in-bytes
```

単一の表のエクスポートまたはインポートで、より高い並列度が必要な場合は、小さい表または表パーティションの並列度を利用するために、PARALLEL_THRESHOLDを小さい値に設定できます。ただし、このリソース割当ての利点は、ロードまたはアンロード先となるファイル・システムのI/Oのパフォーマンスによって制限される可能性があります。また、表とメタデータ・オブジェクトの両方について、ジョブに複数のオブジェクトが含まれている場合、PARALLELおよびPARALLEL_THRESHOLDで指定されたPQ割当て要求の値は制限されます。表に割り当てられるPQプロセスの実際の量は、Oracle Data Pumpが同時に実行している操作の数の影響を受けており、並列度が共有される必要があります。オプティマイザによってSQLに対して生成されるデータベース、オプティマイザおよび実行計画は、ジョブで指定されたオブジェクトのロードまたはアンロードに使用される実際の並列度を決定します。

このパラメータを使用して、特定のデータ移動の問題を支援できます。例:

- Oracle Data Pumpを使用して、あるデータベースの大きな表を別のデータベースのより大きな表にロードする場合。

考えられるユースケースの1つ: OLTPデータベースからレポートまたはビジネス分析のデータ・ウェアハウス・データベースに週次売上データをアップロードします。

- 単一の大きな表をエクスポートするが、最近RDBMS統計を収集していない場合。デフォルト・サイズは表の統計から決定されます。ただし、統計が古い(または一度も実行されていない)と仮定します。その場合、Oracle Data Pumpが使用する値は、表の実際のサイズより小さい可能性があります。このようなケースを補正するには、parallel_threshold値を小さく指定して、並列度(しきい値量で割った表サイズ)のアルゴリズムによって、より適切な並列度の値が得られるようにします。

制限事項

PARALLEL_THRESHOLDは、PARALLELパラメータに1より大きい値が指定されている場合にのみ、組み合せて使用されます。

例

次に、PARALLEL_THRESHOLDパラメータを使用して表table_to_use_PDMLをエクスポートする例を示します。ここでは、PQプロセスの除数のサイズは1 KBに設定され、変数userおよびuser-passwordはエクスポートを実行しているユーザーのユーザーおよびパスワード(expdp)であり、ジョブ名はparathresh_exampleです。

```
expdp user/user-password ¥
    directory=dpump_dir ¥
    dumpfile=parathresh_example.dmp
    tables=table_to_use_PDML ¥
    parallel=8 ¥
    parallel_threshold=1K ¥
    job_name=parathresh_example
```

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.36 PARFILE

データ・ポンプ・エクスポートのコマンドライン・ユーティリティのPARFILEパラメータは、エクスポート・パラメータ・ファイルの名前を指定します。

デフォルト: デフォルト値は設定されていません。

用途

エクスポート・パラメータ・ファイルの名前を指定します。

構文および説明

```
PARFILE=[directory_path]file_name
```

パラメータ・ファイルを使用すると、ファイル内のデータ・ポンプ・パラメータを指定できます。その後、個々のコマンドをすべて入力するかわりに、コマンドラインでそのファイルを指定できます。このパラメータ・ファイルを使用すると、同じパラメータの組合せを何度も使用する場合に便利です。値の指定に引用符が必要なパラメータを使用する場合も、パラメータ・ファイルを使用することをお薦めします。

パラメータ・ファイルには、ディレクトリ・オブジェクトを指定しません。サーバーによって作成されて書き込まれるダンプ・ファイル、ログ・ファイルおよびSQLファイルとは異なり、パラメータ・ファイルはexpdpクライアントによって開かれて読み取られるため、ディレクトリ・オブジェクトを指定しません。パラメータ・ファイルのデフォルトの場所は、ユーザーのカレント・ディレクトリです。

パラメータ・ファイル内では、カンマはすべての改行文字に暗黙的に含まれるため、各行の最後にカンマを入力する必要はありま

せん。長い表名など、折返しを含む長い行がある場合、現在の行の最後にバックスラッシュ継続文字(¥)を入力して次の行に継続します。

パラメータ・ファイルの内容がデータ・ポンプ・ログ・ファイルに書き込まれます。

制限事項

- PARFILEパラメータは、パラメータ・ファイル内には指定できません。

例

パラメータ・ファイルhr.parの内容は、次のとおりです。

```
SCHEMAS=HR
DUMPFILE=exp.dmp
DIRECTORY=dpump_dir1
LOGFILE=exp.log
```

このパラメータ・ファイルを指定するには、次のExportコマンドを実行します。

```
> expdp hr PARFILE=hr.par
```

関連トピック

- データ・ポンプ・エクスポート・パラメータについて

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.37 QUERY

データ・ポンプ・エクスポートのコマンドライン・ユーティリティのQUERYパラメータを使用すると、エクスポート対象となるデータをフィルタ処理するために使用する問合せ句を指定できます。

デフォルト: デフォルト値は設定されていません。

用途

エクスポート対象となるデータをフィルタ処理するために使用する問合せ句を指定できます。

構文および説明

```
QUERY = [schema.][table_name:] query_clause
```

通常、query_clauseでは、ファイングレイン行選択のためのSQL WHERE句を使用しますが、任意のSQL句を使用できます。たとえば、ORDER BY句を使用すると、ヒープ構成表から索引構成表への移行を高速化できます。スキーマおよび表名を指定しなかった場合は、エクスポート・ジョブ内のすべての表に問合せが適用されます(この場合、問合せは、これらのすべての表に対して有効である必要があります)。表固有の問合せは、すべての表に適用される問合せより優先されます。

特定の表に問合せを適用する場合は、表名と問合せ句をコロンで区切る必要があります。表固有の問合せは複数指定できますが、1つの表に指定できるのは1つの問合せのみです。

QUERYパラメータと一緒にNETWORK_LINKパラメータを指定する場合、リモート(ソース)ノード上にあるquery_clauseで指定されるすべてのオブジェクトは、NETWORK_LINK値で明示的に修飾する必要があります。それ以外の場合、データ・ポンプではオブジェクトがローカル(ターゲット)ノード上に存在することを前提とし、存在しない場合はエラーが返され、リモート(ソース)システムからの表のインポートは失敗します。

たとえば、NETWORK_LINK=dblink1と指定した場合、QUERYパラメータのquery_clauseは、次の例に示すようにそのリンクを指定する必要があります。

```
QUERY=(hr.employees:"WHERE last_name IN(SELECT last_name
FROM hr.employees@dblink1)")
```

使用しているオペレーティング・システムによっては、このパラメータの値を引用符とともに指定する際に、エスケープ文字を使用することが必要になる場合もあります。このパラメータはパラメータ・ファイルで指定することをお薦めします。パラメータ・ファイルを使用すると、コマンドラインでは必要なエスケープ文字の数を減らすことができます。

表固有の問合せで自分のスキーマ以外のスキーマを指定するには、その特定の表に対するアクセス権限が付与されている必要があります。

制限事項

- QUERYパラメータは次のパラメータとは併用できません。
  - CONTENT=METADATA_ONLY
  - ESTIMATE_ONLY
  - TRANSPORT_TABLESPACES
- 表にQUERYパラメータが指定されている場合、データ・ポンプは外部表を使用してターゲット表をアンロードします。外部表は、SQL CREATE TABLE AS SELECT文を使用します。QUERYパラメータの値は、CREATE TABLE文のSELECT部分にあるWHERE句です。QUERYパラメータにアンロードする表と一致する名前の列がある他の表への参照が含まれていて、これらの列が問合せで使用される場合は、表別名を使用して、アンロードする表内の列と、SELECT文内の同じ名前を持つ列を区別する必要があります。アンロードする表に対してデータ・ポンプで使用される表別名は、KU$です。

  たとえば、sh.customers表にある顧客のクレジットの上限に基づいてsh.sales表のサブセットをエクスポートするとします。次の例では、KU$を使用して、sh.salesをアンロードするためにQUERYパラメータ内のcust_idフィールドを修飾します。この結果、データ・ポンプによって、クレジットの上限が$10,000を超える顧客の行のみがエクスポートされます。

  ```
  QUERY='sales:"WHERE EXISTS (SELECT cust_id FROM customers c
     WHERE cust_credit_limit > 10000 AND ku$.cust_id = c.cust_id)"'
  ```

  次の問合せでは、表別名にKU$は使用されません。その結果、すべての行がアンロードされます。

  ```
  QUERY='sales:"WHERE EXISTS (SELECT cust_id FROM customers c
     WHERE cust_credit_limit > 10000 AND cust_id = c.cust_id)"'
  ```

- QUERY文字列で使用できる最大長は4000バイトで、引用符が含まれます。この制限事項は、許容される実際の最大長は3998バイトであることを意味します。

例

次に、QUERYパラメータの使用例を示します。

```
> expdp hr PARFILE=emp_query.par
```

emp_query.parファイルの内容は次のとおりです。

```
QUERY=employees:"WHERE department_id > 10 AND salary > 10000"
NOLOGFILE=YES
DIRECTORY=dpump_dir1
DUMPFILE=exp1.dmp
```

この例では、hrスキーマのすべての表がアンロードされます。ただし、アンロードされるのは、問合せ式に適合する行のみです。この場合、hrスキーマ内のすべての表(employeesを除く)のすべての行がアンロードされます。employees表に対しては、問

合せ基準を満たす行のみがアンロードされます。

関連トピック

- データ・ポンプ・エクスポート・パラメータについて

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.38 REMAP_DATA

Oracle Data Pump Exportのコマンドライン・ユーティリティのREMAP_DATAパラメータを使用すると再マップ・ファンクションを指定できます。これにより、指定した列の元の値をソースとして再マップした値を返し、ダンプ・ファイル内の元の値をこの値に置き換えます。

デフォルト

デフォルトはありません

用途

REMAP_DATAパラメータを使用すると再マップ・ファンクションを指定できます。これにより、指定した列の元の値をソースとして再マップした値を返し、ダンプ・ファイル内の元の値をこの値に置き換えます。このオプションは、一般的に、本番システムからテスト・システムへ移動するときにデータをマスクするために使用されます。たとえば、クレジット・カード番号などの顧客の機密データの列を、REMAP_DATAファンクションで生成された番号に置き換えることができます。機密データを数値に置換することで、重要な書式設定および処理特性を、権限のない担当者に個人データを公開することなく保持できます。

同じファンクションを、ダンプされる複数の列に適用できます。このファンクションは、参照制約で子と親両方の列を再マップするときに整合性を保つ必要がある場合に役立ちます。

構文および説明

```
REMAP_DATA=[schema.]tablename.column_name:[schema.]pkg.function
```

次に、各構文要素の説明を構文で出現する順に示します。

schema: 再マップする表を含むスキーマ。デフォルトでは、これはエクスポートを実行するユーザーのスキーマです。

tablename: 列が再マップされる表。

column_name: データが再マップされる列。

schema: 再マップ・ファンクションを含むユーザー作成のPL/SQLパッケージを含むスキーマ。デフォルトでは、これはエクスポートを実行するユーザーのスキーマです。

pkg: 再マップ・ファンクションを含むユーザー作成のPL/SQLパッケージの名前。

function: 指定した表の各行で、列表を再マップする場合にコールされるPL/SQL内のファンクションの名前。

制限事項

- ソース引数および戻り値のデータ型とサイズはともに、表内の指定した列のデータ型とサイズ一致している必要があります。
- 再マップ・ファンクションでは、自律型トランザクション以外のコミットまたはロール・バックを実行できません。
- REMAP_DATAパラメータの値としてのシノニムの使用はサポートされていません。たとえば、hrスキーマ内のregions表にregnのシノニムがある場合は、regnをREMAP_DATA仕様の一部として指定した場合にエラーが返されます。

- リモート表のLOB列データの再マップはサポートされていません。
- REMAP_DATAでは、ユーザー定義タイプ、ユーザー定義タイプの属性、LONG、REF、VARRAY、ネストした表、BFILEおよびXMLtypeのタイプの列はサポートされていません。

例

次の例では、minus10およびplusxという名前のファンクションが含まれたremapという名前のパッケージが作成されていることを前提としています。これらのファンクションは、employees表のemployee_idおよびfirst_nameの値を変更します。

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=remap1.dmp TABLES=employees
REMAP_DATA=hr.employees.employee_id:hr.remap.minus10
REMAP_DATA=hr.employees.first_name:hr.remap.plusx
```

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.39 REUSE_DUMPFILES

データ・ポンプ・エクスポートのコマンドライン・ユーティリティのxxxパラメータでは、既存のダンプ・ファイルを上書きするかどうかを指定します。

デフォルト: NO

用途

すでに存在しているダンプ・ファイルを上書きするかどうかを指定します。

構文および説明

```
REUSE_DUMPFILES=[YES | NO]
```

通常、データ・ポンプ・エクスポートは、すでに存在するダンプ・ファイル名を指定するとエラーを返します。REUSE_DUMPFILESパラメータを使用すると、動作を変更してダンプ・ファイル名を再利用できます。たとえば、DUMPFILE=hr.dmpおよびREUSE_DUMPFILES=YESを指定してエクスポートを実行すると、hr.dmpは、すでに存在する場合には上書きされます。前の内容は消去され、かわりに現行のエクスポートのデータが格納されます。

例

次のエクスポート操作では、enc1.dmpという名前のダンプ・ファイルが作成されます。これは、この名前の付いたダンプ・ファイルがすでに存在する場合でも同様です。

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=enc1.dmp
TABLES=employees REUSE_DUMPFILES=YES
```

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.40 SAMPLE

データ・ポンプ・エクスポートのコマンドライン・ユーティリティのSAMPLEパラメータでは、ソース・データベースからサンプリングおよびアンロードするデータ行の割合を指定します。

デフォルト: デフォルト値は設定されていません。

用途

ソース・データベースからサンプリングおよびアンロードするデータ行の割合を指定します。

構文および説明

```
SAMPLE=[[schema_name.]table_name:]sample_percent
```

このパラメータを使用すると、サンプリングしてエクスポートするデータの割合を指定し、データのサブセットをエクスポートできます。sample_percentは、行がサンプルの一部として選択される可能性を示します。ただし、指定した正確な行数が表から取り出されるわけではありません。sample_percentには、0.000001から100未満の任意の数を指定できます。

特定の表にsample_percentを適用できます。次の例では、HR.EMPLOYEES表の50%がエクスポートされます。

```
SAMPLE="HR"."EMPLOYEES":50
```

スキーマを指定する場合は、表も指定する必要があります。ただし、スキーマを指定せずに表を指定できます。このシナリオでは、現在のユーザーが想定されます。表の指定がない場合は、エクスポート・ジョブ全体にsample_percent値が適用されます。

このパラメータとデータ・ポンプ・インポートのPCTSPACE変換を組み合せて使用すると、記憶域の割当てサイズをサンプリングされたデータ・サブセットに合わせることができます。(詳細は、インポートのTRANSFORMパラメータを参照してください)。

制限事項

- SAMPLEパラメータは、ネットワークのエクスポートに対して無効です。

例

次の例では、表の名前が指定されていないため、全体のエクスポート・ジョブにSAMPLE値70が適用されます。

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=sample.dmp SAMPLE=70
```

関連トピック

- TRANSFORM

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.41 SCHEMAS

データ・ポンプ・エクスポートのコマンドライン・ユーティリティのSCHEMASパラメータでは、スキーマ・モード・エクスポートを実行することを指定します。これは、エクスポートのデフォルトのモードです。

デフォルト: 現在のユーザーのスキーマ

用途

スキーマ・モード・エクスポートの実行を指定します。これは、エクスポートのデフォルトのモードです。

構文および説明

```
SCHEMAS=schema_name [, ...]
```

DATAPUMP_EXP_FULL_DATABASEロールがある場合、自分のスキーマ以外の単一のスキーマまたはスキーマ名のリストを指定できます。DATAPUMP_EXP_FULL_DATABASEロールでは、インポート時にスキーマを再作成できるように、指定した各スキーマのスキーマ・オブジェクト以外の補足情報をエクスポートすることもできます。この補足情報には、ユーザー定義、関連するすべてのシステムおよびロールの権限、ユーザー・パスワードの履歴などが含まれます。スキーマ・モードを使用したエクスポート対象を、フィルタ処理によってさらに制限できます。

制限事項

- DATAPUMP_EXP_FULL_DATABASEロールを持っていない場合は、自分のスキーマのみ指定できます。
- SYSスキーマは、エクスポート・ジョブのソース・スキーマとして使用できません。

例

次に、SCHEMASパラメータの使用例を示します。前述の例でDATAPUMP_EXP_FULL_DATABASEロールがすでに割り当てられているため、ユーザーhrが複数のスキーマを指定できることに注意してください。

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=expdat.dmp SCHEMAS=hr,sh,oe
```

このコマンドで、スキーマ・モード・エクスポートが実行され、dpump_dir1ディレクトリにあるexpdat.dmpダンプ・ファイルに、スキーマhr、shおよびoeが書き込まれます。

関連トピック

- エクスポート操作中のフィルタ処理

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.42 SERVICE_NAME

Oracle Data Pumpエクスポートのコマンドライン・ユーティリティのSERVICE_NAMEパラメータは、CLUSTERパラメータと組み合せることで使用するサービス名を指定します。

デフォルト: デフォルト値は設定されていません。

用途

CLUSTERパラメータと組み合せることで使用するサービス名を指定します。

構文および説明

```
SERVICE_NAME=name
```

SERVICE_NAMEパラメータをCLUSTER=YESパラメータとともに使用して、リソース・グループに属しているOracle Real Application Clusters (Oracle RAC)インスタンスのセットを定義するリソース・グループに関連付けられている、既存のサービスを指定できます。通常、リソース・グループはすべてのOracle RACインスタンスのサブセットです。

サービス名は、リソース・グループおよびそのリソース・グループに対して定義されたインスタンスを決定する目的でのみ使用されます。ジョブが開始されたインスタンスは、そのインスタンスがリソース・グループに含まれているかどうかとは関係なく、常に使用されます。

CLUSTER=NOも指定した場合、SERVICE_NAMEパラメータは無視されます

インスタンスA、B、CおよびDを含むOracle RAC構成があります。インスタンスA、BおよびCのみで構成されるリソース・グループを持つmy_serviceというサービスが存在するとします。そのような場合は、次の内容が成立します。

- Oracle Data Pumpジョブをインスタンス Aで開始したときに、CLUSTER=YESを指定し(またはデフォルトのYをそのまま使用し)、SERVICE_NAMEパラメータを指定しなかった場合、Oracle Data Pumpは、指定されている並列度に応じて、すべてのインスタンス(A、B、CおよびD)にワーカーを作成します。
- データ・ポンプ・ジョブをインスタンスAで開始し、CLUSTER=YES,およびSERVICE_NAME=my_serviceを指定した場合、ワーカーは、インスタンスA、BおよびCのみで開始可能です。
- データ・ポンプ・ジョブをインスタンスDで開始し、CLUSTER=YES,およびSERVICE_NAME=my_serviceを指定した場合、ワーカーは、インスタンスA、B、CおよびDのみで開始可能です。インスタンスDはmy_serviceに存在しませんが、インスタンスDはジョブが開始されたインスタンスであるため含まれています。
- データ・ポンプ・ジョブをインスタンスAで開始し、CLUSTER=NOを指定した場合、SERVICE_NAMEパラメータを指定し

ても無視されます。すべてのプロセスは、インスタンスAで開始されます。

例

次に、SERVICE_NAMEパラメータの使用例を示します。

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=hr_svname2.dmp SERVICE_NAME=sales
```

この例では、hrスキーマのスキーマ・モード・エクスポート(デフォルト・モード)を実行します。コマンドラインではCLUSTER=YESは指定されていませんが、それはデフォルトの動作なので、ジョブではサービス名salesに関連付けられたリソース・グループのすべてのインスタンスが使用されます。hr_svname2.dmpという名前のダンプ・ファイルが、ディレクトリ・オブジェクトdpump_dir1で指定された場所に書き込まれます。

関連トピック

- CLUSTER

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.43 SOURCE_EDITION

データ・ポンプ・エクスポートのコマンドライン・ユーティリティのSOURCE_EDITIONパラメータでは、オブジェクトのエクスポート元のデータベース・エディションを指定します。

デフォルト: システム上のデフォルトのデータベース・エディション

用途

オブジェクトのエクスポート元のデータベース・エディションを指定します。

構文および説明

```
SOURCE_EDITION=edition_name
```

SOURCE_EDITION=edition_nameを指定すると、そのエディションのオブジェクトがエクスポートされます。データ・ポンプは、変更されずに継承されているすべてのオブジェクトと、変更されているすべての実際のオブジェクトを選択します。

このパラメータが指定されていない場合は、デフォルトのエディションが使用されます。指定されたエディションが存在しない場合や使用できない場合は、エラー・メッセージが返されます。

制限事項

- このパラメータは、バージョニング可能な同じオブジェクトのバージョンがデータベース内に2つ以上ある場合にのみ有効です。
- ジョブ・バージョンは、11.2以上である必要があります。

例

次に、SOURCE_EDITIONパラメータの使用例を示します。

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=exp_dat.dmp SOURCE_EDITION=exp_edition
EXCLUDE=USER
```

この例は、オブジェクトのエクスポート元となるexp_editionというエディションがシステム上に存在することを前提としています。エクスポート・モードは指定されていないため、デフォルトのスキーマ・モードが使用されます。EXCLUDE=userパラメータは、ユーザーの定義のみを除外します。ユーザーのスキーマに含まれるオブジェクトは除外されません。

関連トピック

- VERSION
- Oracle Database SQL言語リファレンス
- Oracle Database開発ガイド

関連項目:

- エディションの作成方法の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。
- エディション機能(継承されたオブジェクトと実際のオブジェクトも含む)の詳細は、『*Oracle Database開発ガイド*』を参照してください。

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.44 STATUS

データ・ポンプ・エクスポートのコマンドライン・ユーティリティのSTATUSパラメータでは、ジョブの状態の表示が更新される頻度を指定します

デフォルト: `0`

用途

ジョブ状態の表示が更新される頻度を指定します。

構文および説明

```
STATUS=[integer]
```

`integer`に値を入力すると、ロギング・モードでジョブの状態を表示する頻度を秒単位で指定できます。値を入力しなかった場合またはデフォルト値の0を使用した場合、各オブジェクト型、表またはパーティションの完了に関する情報のみ表示されます。

この状態情報は、標準出力デバイスのみに書き込まれ、ログ・ファイルには(使用可能な場合でも)書き込まれません。

例

次に、STATUSパラメータの使用例を示します。

```
> expdp hr DIRECTORY=dpump_dir1 SCHEMAS=hr,sh STATUS=300
```

この例では、hrおよびshスキーマをエクスポートし、エクスポート状態を5分ごと(60秒×5 = 300秒)に表示します。

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.45 TABLES

データ・ポンプ・エクスポートのコマンドライン・ユーティリティのTABLESパラメータでは、表モード・エクスポートを実行することを指定します。

デフォルト: デフォルト値は設定されていません。

用途

表モード・エクスポートの実行を指定します。

構文および説明

```
TABLES=[schema_name.]table_name[:partition_name] [, ...]
```

このモードを使用したエクスポート対象を、フィルタ処理によって制限できます。表およびパーティションまたはサブパーティションをカンマで区切ったリストを指定して、エクスポート対象のデータおよびメタデータをフィルタ処理できます。パーティションの名前を指定する場合は、関連表にあるパーティションまたはサブパーティションの名前にする必要があります。指定した表、パーティションおよびそれらの依存オブジェクトのみがアンロードされます。

パーティション表全体をエクスポートすると、パーティション表として全体的にインポートされます。これが当てはまらないのは、インポート時にPARTITION_OPTIONS=DEPARTITIONが指定されている場合のみです。

指定する表名の先頭にスキーマ名を修飾できます。デフォルトのスキーマは、現在のユーザーのスキーマです。自分のスキーマ以外のスキーマを指定するには、DATAPUMP_EXP_FULL_DATABASEロールが必要です。

ワイルドカード文字(%)を使用して、表名およびパーティション名を指定できます。

表名には次の制限があります。

- デフォルトでは、表名は大文字でデータベースに格納されます。表名が大文字と小文字または小文字のみで表記され、大/小文字を区別する場合、名前を引用符で囲む必要があります。したがって、表名は、データベースに格納されている表名と完全に一致するように指定する必要があります。

  ただし、オペレーティング・システムによっては、コマンドラインの引用符自体をエスケープする必要がある場合があります。次の例は、異なるエクスポート・モードで大/小文字の区別を保持する方法を示します。

  - コマンドライン・モード

    ```
    TABLES='¥"Emp¥"'
    ```

  - パラメータ・ファイル・モード

    ```
    TABLES='"Emp"'
    ```

- 表名を引用符で囲まないと、コマンドラインで指定する表名にポンド記号(#)は使用できません。同様に、パラメータ・ファイルでは、表名がポンド記号(#)を含んでいると、エクスポート・ユーティリティは、表名が引用符で囲まれていない場合コメントとして解釈します。

  たとえば、パラメータ・ファイルに次のコマンドラインが記述されている場合、エクスポートではemp#の右側がすべてコメントとして解釈されるため、表deptおよびmydataはエクスポートされません。

  ```
  TABLES=(emp#, dept, mydata)
  ```

  ただし、パラメータ・ファイルに次の行が含まれる場合は、emp#が引用符で囲まれているため、エクスポート・ユーティリティは3つの表をすべてエクスポートします。

  ```
  TABLES=('"emp#"', dept, mydata)
  ```

> ノート:
>
> オペレーティング・システムによっては、エスケープ文字として一重引用符を使用する場合と、二重引用符を使用する場合があります。ご使用のオペレーティング・システム固有のドキュメントで確認してください。表のネーミング方法に制限があるオペレーティング・システムもあります。
>
> たとえば、UNIX の C シェルではドル記号($)やポンド記号(#)またはその他の特定の特殊文字には特別

な意味があります。このような文字を名前で使用し、それがシェルで無視され、エクスポートによって使用されるようにするには、エスケープ文字を使用する必要があります。

表モード・エクスポート中のトランスポータブル・オプションの使用

表モード・エクスポート中にトランスポータブル・オプションを使用するには、TRANSPORTABLE=ALWAYSパラメータとTABLESパラメータを組み合せて指定します。指定した表、パーティションまたはサブパーティションのメタデータはダンプ・ファイルにエクスポートされます。実際のデータを移動するには、データ・ファイルをターゲット・データベースにコピーします。

表のパーティションの一部のみをエクスポートし、TRANSPORTABLE=ALWAYSパラメータを使用すると、インポート時にそれぞれのパーティションが非パーティション表になります。

制限事項

- 相互スキーマ参照はエクスポートされません。たとえば、指定されたいずれかのスキーマ内の表にトリガーが定義されていても、そのトリガーが、明示的に指定されていないスキーマ内に常駐している場合はエクスポートされません。
- 表で使用される型は、表モードではエクスポートされません。この制限事項により、後でダンプ・ファイルをインポートするときに宛先データベースに型が存在しない場合、表の作成は失敗します。
- TABLESパラメータの値としてのシノニムの使用はサポートされていません。たとえば、hrスキーマのregions表にregnのシノニムが存在する場合、TABLES=regnを使用すると無効になります。シノニムを使用しようとすると、エラーが返されます。
- 表名にワイルドカード文字(%)を含む表のエクスポートは、表がパーティションの場合はサポートされません。
- TABLESパラメータに指定する表名のリストの長さは、最大4MBに制限されます。ただし、NETWORK_LINKパラメータでリリース10.2.0.3以前のOracle Databaseまたは読取り専用のデータベースが設定されている場合は異なります。この場合の上限は4KBです。
- エクスポートに対してTRANSPORTABLE=ALWAYSも設定されている場合、1つの表からのパーティションのみを指定できます。

例

次の例に、hrスキーマにある3つ表employees、jobsおよびdepartmentsをエクスポートするためにTABLESパラメータを使用する簡単な例を示します。ユーザーhrは、hrスキーマ内の表をエクスポートしているため、表名の前にスキーマ名を指定する必要はありません。

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=tables.dmp
TABLES=employees,jobs,departments
```

次の例では、ユーザーhrにDATAPUMP_EXP_FULL_DATABASEロールが付与されていることを前提としています。ここでは、TABLESパラメータを使用したパーティションのエクスポートを示します。

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=tables_part.dmp
TABLES=sh.sales:sales_Q1_2012,sh.sales:sales_Q2_2012
```

この例では、shスキーマのsales表から、パーティションsales_Q1_2012およびsales_Q2_2012をエクスポートします。

関連トピック

- エクスポート操作中のフィルタ処理
- TRANSPORTABLE
- REMAP_TABLE

- データ・ファイル・コピーを使用したデータ移動

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.46 TABLESPACES

データ・ポンプ・エクスポートのコマンドライン・ユーティリティのTABLESPACESパラメータでは、表領域モードでエクスポートされる表領域名のリストを指定します。

デフォルト: デフォルト値は設定されていません。

用途

表領域モードでエクスポートされる表領域名のリストを指定します。

構文および説明

```
TABLESPACES=tablespace_name [, ...]
```

表領域モードでは、指定した表領域内に存在する表のみがアンロードされます。表がアンロードされると、その表の依存オブジェクトもアンロードされます。オブジェクトのメタデータとデータは、両方ともアンロードされます。指定した表領域内に表の一部が存在する場合、その表とその表のすべての依存オブジェクトがエクスポートされます。特権ユーザーは、すべての表を取得します。権限のないユーザーは、自分のスキーマ内の表のみを取得します

このモードを使用したエクスポート対象を、フィルタ処理によって制限できます。

制限事項

- TABLESPACESパラメータに指定する表領域名のリストの長さは、最大4MBに制限されます。ただし、NETWORK_LINKでリリース10.2.0.3以前のOracle Databaseまたは読取り専用のデータベースが設定されている場合は異なります。この場合の上限は4KBです。

例

次に、TABLESPACESパラメータの使用例を示します。この例では、表領域tbs_4、tbs_5およびtbs_6がすでに存在するとします。

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=tbs.dmp
TABLESPACES=tbs_4, tbs_5, tbs_6
```

このコマンドによって、表領域エクスポートが実行され、指定した表領域(tbs_4、tbs_5およびtbs_6)から表(および表の依存オブジェクト)がアンロードされます。

関連トピック

- エクスポート操作中のフィルタ処理

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.47 TRANSPORT_FULL_CHECK

データ・ポンプ・エクスポートのコマンドライン・ユーティリティのTRANSPORT_FULL_CHECKパラメータでは、トランスポータブル・セット内部のオブジェクトと外部のオブジェクトの間の依存性をチェックするかどうかを指定します。このパラメータは、トランスポータブル表領域モード・エクスポートでのみ使用可能です。

デフォルト: NO

用途

トランスポータブル・セット内部のオブジェクトと外部のオブジェクト間の依存性をチェックするかどうかを指定します。このパラメータは、トランスポータブル表領域モード・エクスポートでのみ使用可能です。

構文および説明

```
TRANSPORT_FULL_CHECK=[YES | NO]
```

TRANSPORT_FULL_CHECK=YESを指定すると、エクスポート・ユーティリティによって、トランスポータブル・セット内のオブジェクトとトランスポータブル・セット外のオブジェクトの間に依存性が存在しないことが確認されます。ここでは、双方向の依存性がチェックされます。たとえば、トランスポータブル・セット内に表は存在するが、その表の索引は存在しない場合は、エラーが返され、エクスポート操作が終了します。同様に、トランスポータブル・セット内に索引は存在するが表は存在しない場合も、エラーが返されます。

TRANSPORT_FULL_CHECK=NOを指定すると、エクスポートによって、トランスポータブル・セット外のオブジェクトに依存するオブジェクトが、トランスポータブル・セット内に存在しないことのみ確認されます。ここでは、一方向の依存性がチェックされます。たとえば、表は索引に依存しませんが、索引は表に依存します。これは、索引は表なしでは意味を持たないためです。そのため、トランスポータブル・セット内に表は存在するが、表の索引は存在しない場合、このチェックは正常に終了します。ただし、トランスポータブル・セット内に索引は存在するが表は存在しない場合は、エクスポート操作が終了します。

他のチェックも実行されます。たとえば、エクスポートでは、常に、TRANSPORT_TABLESPACESで指定された表領域セット内に定義されているすべての表(およびその索引)のすべての記憶域セグメントが、表領域セット内に実際に含まれていることが確認されます。

クローズ・チェック全体を制御する現在のコマンドライン・パラメータは、次の2つです。

```
TTS_FULL_CHECK=[YES|NO]
TRANSPORT_FULL_CHECK=[YES|NO]
```

[TTS|TRANSPORT]_FULL_CHECK=YESはTTS_CLOSURE_CHECK=FULLと解釈されます。

[TTS|TRANSPORT]_FULL_CHECK=NOはTTS_CLOSURE_CHECK=ONと解釈されます。

例

次に、TRANSPORT_FULL_CHECKパラメータの使用例を示します。ここでは、表領域tbs_1が存在するとします。

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=tts.dmp
TRANSPORT_TABLESPACES=tbs_1 TRANSPORT_FULL_CHECK=YES LOGFILE=tts.log
```

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.48 TRANSPORT_TABLESPACES

データ・ポンプ・エクスポートのコマンドライン・ユーティリティのTRANSPORT_TABLESPACESパラメータでは、トランスポータブル表領域モードでのエクスポートを実行することを指定します

デフォルト: デフォルト値は設定されていません。

用途

トランスポータブル表領域モードでのエクスポート実行を指定します。

構文および説明

```
TRANSPORT_TABLESPACES=tablespace_name [, ...]
```

TRANSPORT_TABLESPACESパラメータは、ソース・データベースからターゲット・データベースにオブジェクト・メタデータがエクスポートされる表領域名のリストを指定するために使用します。

エクスポートのログ・ファイルには、トランスポータブル・セットで使用されているデータ・ファイル、ダンプ・ファイルおよび制約違反が一覧表示されます。

TRANSPORT_TABLESPACESパラメータを使用すると、指定した表領域内のすべてのオブジェクトのメタデータがエクスポートされます。特定の表、パーティションまたはサブパーティションのみのトランスポータブル・エクスポートを実行する場合は、TABLESパラメータとTRANSPORTABLE=ALWAYSパラメータを併用する必要があります。

ノート:

トランスポータブル表領域をエクスポートした後、それよりも古いリリース・レベルのデータベースにインポートすることはできません。ターゲット・データベースのリリース・レベルは、ソース・データベース以上である必要があります。

制限事項

- トランスポータブル表領域ジョブは、再開できません。
- トランスポータブル表領域ジョブは、並列度1に制限されます。
- トランスポータブル表領域モードでは、DATAPUMP_EXP_FULL_DATABASEロールが必要です。
- エクスポートを実行するユーザーのデフォルトの表領域を、転送対象となっている表領域のいずれかに設定することはできません。
- SYSTEMおよびSYSAUX表領域は、トランスポータブル表領域モードで転送できません。
- トランスポータブル・セット内のすべての表領域は、読取り専用に設定する必要があります。
- データ・ポンプ・エクスポートのVERSIONパラメータとTRANSPORT_TABLESPACESパラメータと一緒に指定する場合、バージョンはOracle Database COMPATIBLE初期化パラメータ以上である必要があります。
- TRANSPORT_TABLESPACESパラメータは、QUERYパラメータと組み合せて使用することはできません。
- トランスポータブル表領域のジョブでは、データ・ポンプ・エクスポートのACCESS_METHODパラメータはサポートされません。

例

次に、TRANSPORT_TABLESPACESパラメータを(ネットワークベースではなく)ファイル・ベースのジョブに使用した例を示します。表領域tbs_1は、移動する表領域です。この例では、表領域tbs_1がすでに存在し、読取り専用に設定されていると仮定しています。また、この例では、このエクスポート・コマンドの実行前にデフォルトの表領域が変更されていると仮定しています。

```
> expdp hr DIRECTORY=dpump_dir1 DUMPFILE=tts.dmp
TRANSPORT_TABLESPACES=tbs_1 TRANSPORT_FULL_CHECK=YES LOGFILE=tts.log
```

データベース間の表領域の転送の詳細は、*Oracle Database*管理者ガイドを参照してください

関連トピック

- トランスポータブル表領域モード
- データ・ファイル・コピーを使用したデータ移動
- Oracle Data Pumpによるタイムスタンプ・データの処理方法
- Oracle Database管理者ガイド

2.4.49 TRANSPORTABLE

データ・ポンプ・エクスポートのコマンドライン・ユーティリティのTRANSPORTABLEパラメータでは、表モード・エクスポート(TABLESパラメータで指定)または全体モード・エクスポート(FULLパラメータで指定)中にトランスポータブル・オプションを使用する必要があるかどうかを指定します。

デフォルト: NEVER

用途

表モード・エクスポート(TABLESパラメータで指定)または全体モード・エクスポート(FULLパラメータで指定)中にトランスポータブル・オプションを使用する必要があるかどうかを指定します。

構文および説明

```
TRANSPORTABLE = [ALWAYS | NEVER]
```

使用可能な値の定義は、次のとおりです。

ALWAYS - エクスポート・ジョブでトランスポータブル・オプションを使用するように指定します。トランスポータブルが使用できない場合、ジョブは失敗します。

表モード・エクスポートでトランスポータブル・オプションを使用すると、指定した表、パーティションまたはサブパーティションのメタデータのみがエクスポートされるトランスポータブル表領域エクスポートになります。

全体モード・エクスポートでトランスポータブル・オプションを使用すると、データベースの完全なコピーを作成するために必要なすべてのオブジェクトおよびデータがエクスポートされる全体トランスポータブル・エクスポートになります。

NEVER - エクスポート・ジョブでトランスポータブル・オプションではなくダイレクト・パスまたは外部表による方法を使用してデータをアンロードするように指定します。これはデフォルトです。

> ノート:
>
> トランスポータブル・モードで表領域全体をエクスポートするには、TRANSPORT_TABLESPACES パラメータを使用します。

- 表のパーティションの一部のみをエクスポートし、TRANSPORTABLE=ALWAYSパラメータを使用すると、インポート時にそれぞれのパーティションが非パーティション表になります。
- 表のパーティションのサブセットのみをエクスポートし、TRANSPORTABLEパラメータを使用しない場合、またはそのパラメータがNEVERに設定されている場合(デフォルト)は、インポート時に次のようになります。
  - PARTITION_OPTIONS=DEPARTITIONを使用している場合は、ダンプ・ファイル・セットに含まれるそれぞれのパーティションが、非パーティション表として作成されます。
  - PARTITION_OPTIONSを使用していない場合は、完全な表が作成されます。つまり、完全な表内のすべてのメタデータが、ソース上での表定義と同じになるようにターゲット・システム上に存在します。ただし、指定されたパーティション用にエクスポートされたデータのみが表に挿入されます。

制限事項

- TRANSPORTABLEパラメータは、表モード・エクスポートおよび全体モード・エクスポートでのみ有効です。

- TRANSPORTABLEパラメータを使用するには、COMPATIBLE初期化パラメータを11.0.0以上に設定する必要があります。
- FULLパラメータをTRANSPORTABLE (全体トランスポータブル・エクスポートの実行)と組み合せて使用するには、データ・ポンプのVERSIONパラメータを12.0以上に設定する必要があります。VERSIONパラメータを指定しない場合、COMPATIBLEデータベース初期化パラメータを12.0以上に設定する必要があります。
- トランスポータブル・エクスポートを実行するユーザーには、DATAPUMP_EXP_FULL_DATABASE権限が必要です。
- 表、パーティションおよびサブパーティションに関連付けられている表領域は読取り専用である必要があります。
- 全体トランスポータブル・エクスポートでは、データ移動方法の組合せが使用されます。トランスポータブル表領域に存在するオブジェクトは、そのメタデータのみがアンロードされ、データはデータ・ファイルがソース・システムからターゲット・システムにコピーされるときにコピーされます。コピーする必要のあるデータ・ファイルは、エクスポート操作のログ・ファイルの最後に表示されます。非トランスポータブル表領域に存在するオブジェクト(SYSTEMやSYSAUXなど)は、そのメタデータとデータの両方がダンプ・ファイル・セットにアンロードされます。(全体トランスポータブル・エクスポートの実行の詳細は、『*Oracle Database*管理者ガイド』を参照してください。)
- エクスポートを実行するユーザーのデフォルトの表領域を、転送対象となっている表領域のいずれかに設定することはできません。

例

次の例では、shユーザーがDATAPUMP_EXP_FULL_DATABASEロールを持ち、表sales2がパーティション化されて表領域tbs2内に格納されているとします。(tbs2表領域はソース・データベースで読取り専用に設定する必要があります。)

```
> expdp sh DIRECTORY=dpump_dir1 DUMPFILE=tto1.dmp
TABLES=sh.sales2 TRANSPORTABLE=ALWAYS
```

エクスポートが正常に完了した後は、データ・ファイルをターゲット・データベース領域にコピーする必要があります。次に、PARTITION_OPTIONSとREMAP_SCHEMAの各パラメータを使用してインポート操作を実行し、sales2の各パーティションを固有の表にします。

```
> impdp system PARTITION_OPTIONS=DEPARTITION
TRANSPORT_DATAFILES=oracle/dbs/tbs2 DIRECTORY=dpump_dir1
DUMPFILE=tto1.dmp REMAP_SCHEMA=sh:dp
```

関連トピック

- Oracle Database管理者ガイド
- 全体モード
- データ・ファイル・コピーを使用したデータ移動

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.50 TTS_CLOSURE_CHECK

データ・ポンプ・エクスポート・コマンドライン・モードのTTS_CLOSURE_CHECKパラメータは、データ・ポンプのトランスポータブル表領域操作の一部として実行されるクローズ・チェックの程度を示すために使用されます。

用途

トランスポータブル・エクスポート操作の一環として実行するクローズ・チェックのレベルを指定します。TTS_CLOSURE_CHECKパラメータを使用して、テスト・モードのトランスポータブル表領域操作中に、読取り/書込み可能な状態に表領域を保持することを示すこともできます。このオプションは、エクスポート操作のタイミング要件を取得するために使用されます。これはテスト目的

の場合のみです。インポートにはダンプ・ファイルを使用できません。

構文および説明

TTS_CLOSURE_CHECK = [ON | OFF | FULL | TEST_MODE]

TTS_CLOSURE_CHECKパラメータでは、次のオプションがサポートされます。

- ON - 自己完結型クローズ・チェックが実行されることを示します
- OFF - クローズ・チェックが実行されないことを示します
- FULL - 完全な双方向クローズ・チェックが実行されることを示します
- TEST_MODE - 表領域を読取り専用モードにする必要がないことを示します

ON、OFFおよびFULLオプションは相互に排他的です。TEST_MODEは、データ・ポンプ・エクスポート・オプションのみです。

例2-1 例

TTS_CLOSURE_CHECK=FULL

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.51 VERSION

データ・ポンプ・エクスポートのコマンドライン・ユーティリティのVERSIONパラメータでは、エクスポートするデータベース・オブジェクトのバージョンを指定します。

デフォルト: COMPATIBLE

用途

エクスポートするデータベース・オブジェクトのバージョンを指定します。指定したリリースと互換性のあるデータベース・オブジェクトおよび属性のみがエクスポートされます。VERSIONパラメータを使用して、以前のリリースのOracle Databaseと互換性のあるダンプ・ファイル・セットを作成できます。データ・ポンプ・エクスポートは、Oracle Database10gリリース1 (10.1)より前のOracle Databaseのリリースでは使用できません。データ・ポンプ・エクスポートは、Oracle Database 10gリリース1(10.1)以降でのみ動作します。VERSIONパラメータを使用して可能になるのは、エクスポートするオブジェクトのバージョンの識別のみです。

Oracle Database 11gリリース2 (11.2.0.3)以降では、VERSIONパラメータをVERSION=12とし、FULL=Yとともに指定することで、Oracle Database 12cにインポートできる全体エクスポートのダンプ・ファイルを生成できます。以降のリリース・ターゲットのVERSION値を含むエクスポートには、登録済データベースのオプションおよびコンポーネントからの情報が含まれます。以降のリリース・バージョンを指定するダンプ・ファイル・セットは、Oracle Database 12cリリース1 (12.1.0.1)以降にのみインポートできます。たとえば、VERSION=12をFULL=Yとともに使用して、TRANSPORTABLE=ALWAYSも指定すると、Oracle Database 12cにインポートできる全体トランスポータブル・エクスポート・ダンプ・ファイルが生成されます。詳細は、FULLエクスポート・パラメータのオプションを参照してください。

構文および説明

```
VERSION=[COMPATIBLE | LATEST | version_string]
```

VERSIONパラメータで使用可能な値は、次のとおりです。

- COMPATIBLE - この値がデフォルト値です。メタデータのバージョンは、COMPATIBLE初期化パラメータで指定したデータベース互換性レベルに対応します。

  ノート: データベースの互換性は、9.2以降に設定する必要があります。

- LATEST - メタデータおよび生成されるSQL DDLのバージョンは、互換性レベルに関係なく、データベース・リリースに対応します。
- version_string - 特定のデータベース・リリース(11.2.0など)。Oracle Database 11gの場合、9.2未満の値は指定できません。

VERSIONで指定したリリースと互換性のないデータベース・オブジェクトまたは属性は、エクスポートされません。たとえば、指定したリリースではサポートされていない新しいデータ型を含む表はエクスポートされません。

制限事項

- アーカイブされたLOBがある表を、リリース11.2よりも前のデータベースにエクスポートすることはできません。
- データ・ポンプ・エクスポートのVERSIONパラメータをTRANSPORT_TABLESPACESパラメータとともに指定する場合、VERSIONの値はOracle Database COMPATIBLE初期化パラメータ以上である必要があります。
- データ・ポンプのVERSIONパラメータを12.1より前の任意の値に指定すると、4000バイトを超える長さのVARCHAR2またはNVARCHAR2列と、2000バイトを超える長さのRAW列を含む表は、データ・ポンプのダンプ・ファイルから除外されます。
- データ・ポンプのパラメータVERSION=12を使用してOracle Database 11gリリースで作成したダンプ・ファイルは、Oracle Database 12cリリース1 (12.1)以上でのみインポートできます。

例

次の例は、メタデータのバージョンがデータベースのリリースに対応している場合のエクスポートの例を示します。

```
> expdp hr TABLES=hr.employees VERSION=LATEST DIRECTORY=dpump_dir1
DUMPFILE=emp.dmp NOLOGFILE=YES
```

関連トピック

- 全体モード
- 異なるOracle Databaseリリース間のエクスポートとインポート

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.4.52 VIEWS_AS_TABLES

Oracle Data Pump Exportのコマンドライン・ユーティリティのVIEWS_AS_TABLESパラメータでは、1つ以上のビューを表としてエクスポートすることを指定します。

デフォルト

デフォルトはありません

注意:

VIEWS_AS_TABLES パラメータによって、暗号化されていない形式のビュー・データはアンロードされ、暗号化されていない表が作成されます。機密データをアンロードする場合、エクスポート操作で暗号化を有効にし、必ず暗号化された表領域に表を作成することをお薦めします。REMAP_TABLESPACE パラメータを使用して、このような表領域に表を移動できます。

用途

1つ以上のビューを表としてエクスポートすることを指定します。

構文および説明

```
VIEWS_AS_TABLES=[schema_name.]view_name[:table_name], ...
```

Oracle Data Pumpによって、ビューと同じ列を含み、ビューから取得された行データを含む表がエクスポートされます。Oracle Data Pumpでは、権限や制約などのビューに依存するオブジェクトもエクスポートされます。表に適用されない依存オブジェクト(UNDERオブジェクト権限の付与など)は、エクスポートされません。VIEWS_AS_TABLESパラメータは、単独で使用することも、TABLESパラメータと組み合せて使用することもできます。どちらを使用しても、Oracle Data Pumpでは表モード・エクスポートが実行されます。

構文要素の定義は次のとおりです。

schema_name: ビューが存在するスキーマの名前。スキーマ名を指定しない場合、デフォルトで、エクスポートを実行するユーザーになります。

view_name: 表としてエクスポートするビューの名前。ビューは存在し、LOB以外のスカラー列のみを含むリレーショナル・ビューである必要があります。無効なビューや存在しないビューを指定すると、そのビューはスキップされてエラー・メッセージが返されます。

table_name: エクスポートされるビューのメタデータのソースとなる表の名前。デフォルトでは、Oracle Data Pumpによって、ビューと同じ列およびデータ型を持つ、行の含まれない一時的なテンプレート表が自動的に作成されます。データベースが読取り専用の場合、このテンプレート表のデフォルトの作成には失敗します。この場合、表名を指定できます。表は、ビューと同じスキーマに属している必要があります。これは、ヒープ構成のパーティション化されていないリレーショナル表である必要があります。ネストした表にすることはできません。

テンプレート表が明示的に指定された複数のビューがエクスポート・ジョブに含まれる場合、それらのテンプレート表はすべて異なっている必要があります。たとえば、2つのビューで同じテンプレート表を使用している次のジョブでは、ビューの1つがスキップされます。

```
expdp scott/password directory=dpump_dir dumpfile=a.dmp views_as_tables=v1:emp,v2:emp
```

オブジェクトの省略をレポートするエラー・メッセージが返されます。

テンプレート表は、エクスポート操作の完了後に自動的に削除されます。それらが存在している間は、次の問合せを実行してその名前を表示できます(名前はすべてKU$VATで始まります)。

```
SQL> SELECT * FROM user_tab_comments WHERE table_name LIKE 'KU$VAT%';
TABLE_NAME                     TABLE_TYPE
------------------------------ -----------
COMMENTS
-------------------------------------------------------
KU$VAT_63629                   TABLE
Data Pump metadata template table for view SCOTT.EMPV
```

制限事項

- VIEWS_AS_TABLESパラメータは、TRANSPORTABLE=ALWAYSパラメータと組み合せて使用することはできません。
- VIEWS_AS_TABLESパラメータを使用して作成された表には、指定したビューの一部であった非表示列は含まれません。
- VIEWS_AS_TABLESパラメータでは、LONGのデータ型の列のある表をサポートしていません。

例

次の例では、ビューscott.view1の内容をscott1.dmpというダンプ・ファイルにエクスポートします。

```
> expdp scott/password views_as_tables=view1 directory=data_pump_dir
dumpfile=scott1.dmp
```

ダンプ・ファイルには、ビューから取得された行を持つview1という表が含まれます。

親トピック: データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータ

2.5 データ・ポンプ・エクスポートの対話方式コマンド・モードで使用可能なコマンド

データ・ポンプ・エクスポートを対話方式コマンド・モードで使用するときに使用可能なコマンド・オプションを確認します。

- Oracle Data Pump Exportの対話方式コマンド・モードについて
  現行のジョブの実行中にOracle Data Pump Exportの対話方式コマンド・モードで使用可能なコマンドについて学習します。
- ADD_FILE
  Oracle Data Pump Exportの対話方式コマンド・モードのADD_FILEパラメータにより、エクスポート・ダンプ・ファイル・セットに、さらにファイルまたは置換変数を追加します。
- CONTINUE_CLIENT
  Oracle Data Pump Exportの対話方式コマンド・モードのCONTINUE_CLIENTパラメータにより、対話方式コマンド・モードからロギング・モードに変更します。
- EXIT_CLIENT
  Oracle Data Pump Exportの対話方式コマンド・モードのEXIT_CLIENTパラメータにより、エクスポート・クライアント・セッションを停止し、エクスポート・ユーティリティを終了して、端末へのロギングを中断します。ただし、現行のジョブの実行は続行します。
- FILESIZE
  Oracle Data Pump Exportの対話方式コマンド・モードのFILESIZEパラメータでは、後のダンプ・ファイルの最大サイズを再定義します。
- HELP
  Oracle Data Pump Exportの対話方式コマンド・モードのHELPパラメータにより、対話方式コマンド・モードで使用可能なデータ・ポンプ・エクスポート・コマンドの情報を表示します。
- KILL_JOB
  Oracle Data Pumpエクスポートの対話方式コマンド・モードのKILL_JOBパラメータでは、現在接続中のすべてのワーカー・クライアント・セッションを切断してから、現行のジョブを終了します。エクスポート・ユーティリティを終了し、端末プロンプトに戻します。
- PARALLEL
  エクスポート・インタラクティブ・コマンド・モードのPARALLELパラメータを使用すると、現在のジョブのアクティブなプロセス(子プロセスおよびパラレル子プロセス)の数を増減できます。
- START_JOB
  Oracle Data Pumpエクスポートの対話方式コマンド・モードのSTART_JOBパラメータでは、接続している現行のジョブを開始します。
- STATUS
  Oracle Data Pump Exportの対話方式コマンドのSTATUSパラメータにより、エクスポートに関する状態情報を表示し、ロギング・モードの状態を表示する間隔を再設定できます。
- STOP_JOB

Oracle Data Pumpエクスポートの対話方式コマンド・モードのSTOP_JOBパラメータでは、現行のジョブを停止します。ジョブを即時にまたは手順に従って停止し、エクスポートを終了します。

親トピック: Oracle Data Pumpエクスポート

2.5.1 Oracle Data Pump Exportの対話方式コマンド・モードについて

現行のジョブの実行中にOracle Data Pump Exportの対話方式コマンド・モードで使用可能なコマンドについて学習します。

対話方式コマンド・モードでは、現行のジョブは継続して続行されますが、端末へのロギングは一時停止され、エクスポート・プロンプト(Export>)が表示されます。

対話方式コマンド・モードを開始するには、次のいずれかの方法を使用します。

- 接続されたクライアントから、[Ctrl]を押しながら[C]を押します。
- ジョブを実行している端末以外の端末から、expdpコマンドでATTACHパラメータを指定してジョブに接続します。ATTACHは、ある場所で開始したジョブを、後で別の場所から確認する必要がある場合に有効な機能です。

次の表に、現行のジョブに対して対話方式コマンド・モードでデータ・ポンプ・エクスポート・プロンプトから実行できる操作を示します。

表2-1 データ・ポンプ・エクスポートの対話方式コマンド・モードでサポートされているコマンド

操作	使用するコマンド
追加ダンプ・ファイルを追加する。	ADD_FILE
対話方式モードを終了し、ロギング・モードに切り替える。	CONTINUE_CLIENT
ジョブは続行したままエクスポート・クライアント・セッションを停止する。	EXIT_CLIENT
ダンプ・ファイルのデフォルト・サイズを再定義して、その後のすべてのダンプ・ファイルに適用する。	FILESIZE
使用可能なコマンドの概要を表示する。	HELP
現在接続中のすべてのクライアント・セッションを切断し、現行のジョブを停止する。	KILL_JOB
現行のジョブに対するアクティブなワーカー・プロセスの数を増減する。このコマンドは、Oracle Database 11g 以上の Enterprise Edition でのみ有効。	PARALLEL
接続している停止ジョブを再開する。	START_JOB
現行のジョブの詳細な状態を表示したり、状態の表示周期を設定する。	STATUS
後で再開するために、現行のジョブを停止する。	STOP_JOB

親トピック: データ・ポンプ・エクスポートの対話方式コマンド・モードで使用可能なコマンド

2.5.2 ADD_FILE

Oracle Data Pump Exportの対話方式コマンド・モードのADD_FILEパラメータにより、エクスポート・ダンプ・ファイル・セットに、さらにファイルまたは置換変数を追加します。

用途

エクスポート・ダンプ・ファイル・セットに、追加ファイルまたは置換変数を追加します。

構文および説明

```
ADD_FILE=[directory_object:]file_name [,...]
```

各ファイル名に、異なるディレクトリ・オブジェクトを指定できます。ディレクトリ・オブジェクトを指定しない場合は、デフォルトが使用されます。

file_nameに、ディレクトリ・パスの情報を含めないでください。ただし、置換変数(%U)を含めることはできます。この置換変数は、指定されたファイル名をテンプレートとして使用し、複数のファイルが生成できることを示します。

追加されるファイルのサイズは、FILESIZEパラメータの設定によって決定されます。

例

次の例では、2つのダンプ・ファイルをダンプ・ファイル・セットに追加します。ダンプ・ファイルhr2.dmpに、ディレクトリ・オブジェクトが指定されていないため、ジョブに対するデフォルトのディレクトリ・オブジェクトを使用するとします。別のディレクトリ・オブジェクトdpump_dir2が、ダンプ・ファイルhr3.dmpに指定されています。

```
Export> ADD_FILE=hr2.dmp, dpump_dir2:hr3.dmp
```

関連トピック

- Oracle Data Pumpによるファイルの割当て

親トピック: データ・ポンプ・エクスポートの対話方式コマンド・モードで使用可能なコマンド

2.5.3 CONTINUE_CLIENT

Oracle Data Pump Exportの対話方式コマンド・モードのCONTINUE_CLIENTパラメータにより、対話方式コマンド・モードからロギング・モードに変更します。

用途

エクスポート・モードを、対話方式コマンド・モードからロギング・モードに変更します。

構文および説明

```
CONTINUE_CLIENT
```

ロギング・モードでは、状態が端末に継続的に出力されます。ジョブが現在停止している場合、CONTINUE_CLIENTを指定すると、クライアントがジョブの開始を試みます。

例

```
Export> CONTINUE_CLIENT
```

親トピック: データ・ポンプ・エクスポートの対話方式コマンド・モードで使用可能なコマンド

2.5.4 EXIT_CLIENT

Oracle Data Pump Exportの対話方式コマンド・モードのEXIT_CLIENTパラメータにより、エクスポート・クライアント・セッションを停止し、エクスポート・ユーティリティを終了して、端末へのロギングを中断します。ただし、現行のジョブの実行は続行します。

用途

エクスポート・クライアント・セッションを停止し、エクスポート・ユーティリティを終了して、端末へのロギングを中断します。ただし、現行のジョブの実行は続行します。

構文および説明

```
EXIT_CLIENT
```

EXIT_CLIENTでは、ジョブが実行されたままになるため、後でこのジョブに接続できます。ジョブの状態を確認するには、ジョブのログ・ファイルを監視するか、USER_DATAPUMP_JOBSビューまたはV$SESSION_LONGOPSビューを問い合せることができます。

例

```
Export> EXIT_CLIENT
```

親トピック: データ・ポンプ・エクスポートの対話方式コマンド・モードで使用可能なコマンド

2.5.5 FILESIZE

Oracle Data Pump Exportの対話方式コマンド・モードのFILESIZEパラメータでは、後のダンプ・ファイルの最大サイズを再定義します。

用途

後のダンプ・ファイルの最大サイズを再定義します。ダンプ・ファイル・セット内にあるダンプ・ファイルが最大サイズになると、そのファイルはクローズされ、ファイル指定に置換変数が含まれている場合または追加ダンプ・ファイルがジョブに追加されている場合は、新しいファイルが作成されます。

構文および説明

```
FILESIZE=integer[B | KB | MB | GB | TB]
```

integerの後に(スペースを挿入しない)、B、KB、MB、GBまたはTB(それぞれバイト、キロバイト、メガバイト、ギガバイト、テラバイトを示す)を指定できます。デフォルトは、B(バイト)です。作成されるファイルの実際のサイズは、ダンプ・ファイル内で使用されている内部ブロックのサイズと一致するように切り捨てられる場合があります。

ファイル・サイズ0は最大ファイル・サイズの16TBと同じです。

制限事項

- ファイルの最小サイズは、デフォルトのOracle Data Pumpブロック・サイズの10倍、つまり4KBです。
- ファイルの最大サイズは16TBです。

例

```
Export> FILESIZE=100MB
```

親トピック: データ・ポンプ・エクスポートの対話方式コマンド・モードで使用可能なコマンド

2.5.6 HELP

Oracle Data Pump Exportの対話方式コマンド・モードのHELPパラメータにより、対話方式コマンド・モードで使用可能なデータ・ポンプ・エクスポート・コマンドの情報を表示します。

用途

対話方式コマンド・モードで使用可能なOracle Data Pump Exportコマンドの情報を表示します。

構文および説明

```
HELP
```

対話方式コマンド・モードで使用可能なコマンドの情報を表示します。

例

```
Export> HELP
```

親トピック: データ・ポンプ・エクスポートの対話方式コマンド・モードで使用可能なコマンド

2.5.7 KILL_JOB

Oracle Data Pumpエクスポートの対話方式コマンド・モードのKILL_JOBパラメータでは、現在接続中のすべてのワーカー・クライアント・セッションを切断してから、現行のジョブを終了します。エクスポート・ユーティリティを終了し、端末プロンプトに戻します。

用途

現在接続中のすべての子クライアント・セッションを切断してから、現行のジョブを停止します。エクスポート・ユーティリティを終了し、端末プロンプトに戻します。

構文および説明

```
KILL_JOB
```

KILL_JOBを使用して中断されたジョブは、再開できません。接続中のすべてのクライアント(KILL_JOBコマンドを発行しているクライアントを含む)は、現在のユーザーがジョブを停止しているという警告を受け取った後、切断されます。すべての子クライアントが切断されると、ジョブのプロセス構造がただちに停止し、データ・ポンプ制御ジョブ表およびダンプ・ファイルが削除されます。ログ・ファイルは、削除されません。

例

```
Export> KILL_JOB
```

親トピック: データ・ポンプ・エクスポートの対話方式コマンド・モードで使用可能なコマンド

2.5.8 PARALLEL

エクスポート・インタラクティブ・コマンド・モードのPARALLELパラメータを使用すると、現在のジョブのアクティブなプロセス(子プロセスおよびパラレル子プロセス)の数を増減できます。

用途

現在のジョブのアクティブなプロセス(子プロセスおよびパラレル子プロセス)の数を増減できます。

構文および説明

```
PARALLEL=integer
```

PARALLELは、コマンドライン・パラメータおよび対話方式コマンド・モードのパラメータとして使用可能です。必要な数のパラレル・プロセス(子プロセスおよびパラレル子プロセス)に設定します。増加処理は、ファイルとリソースが十分にある場合は即時に実行されます。減少処理は、既存のプロセスが現行のタスクを終了してから実行されます。値が減少すると、子プロセスはアイドル状態になりますが、ジョブが終了するまで削除されません。

制限事項

- このパラメータは、Oracle Database 11g以降のリリースのEnterprise Editionでのみ有効です。
- トランスポータブル表領域のメタデータは、パラレルでインポートできません。
- NETWORK_LINKパラメータが使用されていると、メタデータはパラレルでインポートできなくなります。

また、次のオブジェクトもパラレルでインポートできません。

- TRIGGER
- VIEW
- OBJECT_GRANT
- SEQUENCE
- CONSTRAINT
- REF_CONSTRAINT

例

```
Export> PARALLEL=10
```

関連トピック

- PARALLEL

親トピック: データ・ポンプ・エクスポートの対話方式コマンド・モードで使用可能なコマンド

2.5.9 START_JOB

Oracle Data Pumpエクスポートの対話方式コマンド・モードのSTART_JOBパラメータでは、接続している現行のジョブを開始します。

用途

接続している現行のジョブを開始します。

構文および説明

```
START_JOB
```

START_JOBコマンドは、接続先の現在のジョブを再起動します。このコマンドを発行したときは、ジョブを実行できません。ダンプ・ファイル・セットおよびデータ・ポンプ制御ジョブ表が元のまま変更されていない場合は、予期しない障害またはSTOP_JOBコマンドの発行後に、データの損失や破損なしにジョブが再開されます。

トランスポータブル表領域モードで実行されたエクスポートは、再開できません。

例

```
Export> START_JOB
```

親トピック: データ・ポンプ・エクスポートの対話方式コマンド・モードで使用可能なコマンド

2.5.10 STATUS

Oracle Data Pump Exportの対話方式コマンドのSTATUSパラメータにより、エクスポートに関する状態情報を表示し、ロギング・モードの状態を表示する間隔を再設定できます。

用途

ジョブの累積的な状態、現行の操作の説明および推定完了率を表示します。ロギング・モードの状態を表示する間隔を再設定することもできます。

構文および説明

```
STATUS[=integer]
```

ロギング・モードでのこの状態の表示頻度を秒単位で指定できるオプションがあります。値を入力しなかった場合またはデフォルト値の0を使用した場合は、状態の定期表示はオフになり、状態は1回のみ表示されます。

この状態情報は、標準出力デバイスのみに書き込まれ、ログ・ファイルには(使用可能な場合でも)書き込まれません。

例

次に、現行のジョブの状態を表示し、ロギング・モードの表示間隔を5分(300秒)に変更する例を示します。

```
Export> STATUS=300
```

親トピック: データ・ポンプ・エクスポートの対話方式コマンド・モードで使用可能なコマンド

2.5.11 STOP_JOB

Oracle Data Pumpエクスポートの対話型コマンド・モードのSTOP_JOBパラメータは、現在のジョブを停止します。ジョブを即時にまたは手順に従って停止し、エクスポートを終了します。

用途

現行のジョブを即時にまたは手順に従って停止し、エクスポートを終了します。

構文および説明

```
STOP_JOB[=IMMEDIATE]
```

STOP_JOBコマンド発行時または発行後にデータ・ポンプ制御ジョブ表およびダンプ・ファイル・セットに障害が発生していない場合は、そのジョブに接続し、START_JOBコマンドを使用して再開できます。

手順に従って停止する場合は、関連する値を指定しないでSTOP_JOBを使用します。確認を要求する警告が発行されます。手順に従った停止では、ワーカー・プロセスで現行のタスクが終了した後、ジョブが停止されます。

即時に停止するには、STOP_JOB=IMMEDIATEを指定します。確認を要求する警告が発行されます。接続中のすべてのクライアント(STOP_JOBコマンドを発行しているクライアントを含む)は、現在のユーザーがジョブを停止および切断中であるという警告を受け取ります。すべてのクライアントが切断されると、ジョブのプロセス構造が即時に停止されます。つまり、データ・ポンプ制御ジョブ・プロセスは、子プロセスが現在のタスクを終了するまで待機しません。STOP_JOB=IMMEDIATEを指定した場合、データ破損やデータ損失の危険性はありません。ただし、停止時に完了しなかった一部のタスクは、再開時に再実行する必要があります。

例

```
Export> STOP_JOB=IMMEDIATE
```

親トピック: データ・ポンプ・エクスポートの対話方式コマンド・モードで使用可能なコマンド

2.6 Oracle Data Pumpエクスポートの使用例

次の一般的なシナリオ例を使用して、データの移動にOracle Data Pumpエクスポートを使用する方法について学習します。

- 表モード・エクスポートの実行
  この例では、TABLESパラメータを使用して指定される表モード・エクスポートを示します。
- 選択した表および行のデータのみのアンロード
  この例では、選択した表および行のデータのみのアンロードを示します。
- 表モード・エクスポートに必要なディスク領域の見積り
  この例では、表モード・エクスポートで使用される領域を見積もる方法を示します。
- スキーマ・モード・エクスポートの実行
  この例では、スキーマ・モード・エクスポートの実行方法を示します。
- パラレル全体データベース・エクスポートの実行
  パラレル全体データベース・エクスポートを実行する方法を学習するには、この例を使用して構文を理解します。
- 対話方式モードを使用したジョブの停止および再接続
  この例では、対話方式モードを使用してジョブを停止する方法、およびジョブに再接続する方法を示します。

親トピック: Oracle Data Pumpエクスポート

2.6.1 表モード・エクスポートの実行

この例は、TABLESパラメータを使用して指定される表モード・エクスポートを示しています。

この例では、データ・ポンプ・エクスポート・コマンドにより、人事管理(hr)スキーマから表employeesおよびjobsの表エクスポートを実行します。

ユーザーhrは、自分のスキーマ内の表をエクスポートしているため、表名の前にスキーマ名を指定する必要はありません。NOLOGFILE=YESパラメータは、その操作のエクスポート・ログ・ファイルが生成されないことを示します。

例2-2 表モード・エクスポートの実行

```
expdp hr TABLES=employees,jobs DUMPFILE=dpump_dir1:table.dmp NOLOGFILE=YES
```

親トピック: Oracle Data Pumpエクスポートの使用例

2.6.2 選択した表および行のデータのみのアンロード

この例は、選択した表および行のデータのみのアンロードを示しています。

この例では、表countriesおよびregionsを除く人事管理(hr)スキーマ内のすべての表について、データのみのアンロードを実行するために使用可能なパラメータ・ファイル(exp.par)の内容を示します。employees表から、department_idが50以外の行がアンロードされます。行は、employee_id順にソートされます。

次のコマンドを使用して、exp.parパラメータ・ファイルを実行できます。

```
> expdp hr PARFILE=exp.par
```

このエクスポートにより、スキーマ・モード・エクスポート(デフォルト・モード)が実行されますが、CONTENTパラメータによって、エクスポートが表のデータのみのアンロードに効果的に制限されます。ディレクトリ・オブジェクトdpump_dir1,は、DBAによってすでに作成されています。このオブジェクトは、エクスポート・ダンプ・ファイルに対する読取りおよび書込み権限がユーザーhrに付与され

ているサーバー上のディレクトリを示しています。ダンプ・ファイルdataonly.dmpは、dpump_dir1に作成されます。

例2-3 選択した表および行のデータのみのアンロード

```
DIRECTORY=dpump_dir1
DUMPFILE=dataonly.dmp
CONTENT=DATA_ONLY
EXCLUDE=TABLE:"IN ('COUNTRIES', 'REGIONS')"
QUERY=employees:"WHERE department_id !=50 ORDER BY employee_id"
```

親トピック: Oracle Data Pumpエクスポートの使用例

2.6.3 表モード・エクスポートに必要なディスク領域の見積り

この例では、表モード・エクスポートで使用される領域を見積もる方法を示します。

この例では、ESTIMATE_ONLYパラメータを使用して、実際のエクスポート操作を実行することなく、表モード・エクスポートで消費される領域を見積もります。次のコマンドを実行し、BLOCKSメソッドを使用して、人事管理(hr)スキーマの3つの表employees、departmentsおよびlocationsのデータのエクスポートに必要なバイト数を見積もります。

見積りはログ・ファイルに出力され、クライアントの標準出力デバイスに表示されます。見積りの対象は、表の行データのみです。メタデータは含まれません。

例2-4 表モード・エクスポートに必要なディスク領域の見積り

```
> expdp hr DIRECTORY=dpump_dir1 ESTIMATE_ONLY=YES TABLES=employees,
departments, locations LOGFILE=estimate.log
```

親トピック: Oracle Data Pumpエクスポートの使用例

2.6.4 スキーマ・モード・エクスポートの実行

この例では、スキーマ・モード・エクスポートの実行方法を示します。

この例では、hrスキーマのスキーマ・モード・エクスポートを示します。スキーマ・モード・エクスポートでは、対応するスキーマに属するオブジェクトのみがアンロードされます。スキーマ・モードは、デフォルトのモードであるため、複数のスキーマや自分のスキーマ以外を指定する場合以外、コマンドラインでSCHEMASパラメータを指定する必要はありません。

例2-5 スキーマ・モード・エクスポートの実行

```
> expdp hr DUMPFILE=dpump_dir1:expschema.dmp LOGFILE=dpump_dir1:expschema.log
```

親トピック: Oracle Data Pumpエクスポートの使用例

2.6.5 パラレル全データベース・エクスポートの実行

パラレル全データベース・エクスポートの実行方法を学習するには、この例を使用して構文を理解します。

この例は、3つのパラレル・プロセス(ワーカーまたはパラレル問合せワーカー・プロセス)を使用できる全データベース・エクスポートを示しています。

例2-6 パラレル全体エクスポート

```
> expdp hr FULL=YES DUMPFILE=dpump_dir1:full1%U.dmp, dpump_dir2:full2%U.dmp
FILESIZE=2G PARALLEL=3 LOGFILE=dpump_dir1:expfull.log JOB_NAME=expfull
```

このエクスポートは、全データベース・エクスポートであるため、データベースのすべてのデータおよびメタデータがエクスポートされま

す。full101.dmp、full201.dmp、full102.dmpなどのダンプ・ファイルが、dpump_dir1およびdpump_dir2ディレクトリ・オブジェクトで示されたディレクトリに、ラウンドロビン法で作成されます。最高のパフォーマンスを得るために、Oracleではダンプ・ファイルを個別の入出力(I/O)チャネルに配置することをお薦めします。各ファイルのサイズは、必要に応じて2GBまで拡張されます。最初に、最大3つのファイルが作成されます。必要に応じて、さらにファイルが作成されます。ジョブおよびデータ・ポンプ制御プロセス表の名前はexpfullです。ログ・ファイルは、dpump_dir1ディレクトリのexpfull.logに書き込まれます。

親トピック: Oracle Data Pumpエクスポートの使用例

2.6.6 対話方式モードを使用したジョブの停止および再接続

この例では、対話方式モードを使用してジョブを停止したり、ジョブに再接続する方法を示します。

この例を開始する前に、次に示すパラレル全体エクスポートを再実行します。

パラレル全データベース・エクスポートの実行

エクスポートの実行中、[Ctrl]を押しながら[C]を押します。このキーボード・コマンドでは、データ・ポンプ・エクスポートの対話方式コマンド・インタフェースを起動します。対話方式インタフェースでは、端末へのロギングは停止され、データ・ポンプ・エクスポートのプロンプトが表示されます。

ジョブの状態が表示された後、CONTINUE_CLIENT コマンドを実行して、ロギング・モードを再開し、expfullジョブを再起動できます。

```
Export> CONTINUE_CLIENT
```

ジョブが再オープンされたことを示すメッセージが表示され、処理の状態がクライアントに出力されます。

例2-7 ジョブの停止と再接続

エクスポート・プロンプトで、次のコマンドを実行してジョブを停止します。

```
Export> STOP_JOB=IMMEDIATE
Are you sure you wish to stop this job ([y]/n): y
```

このジョブは停止状態でクライアントを終了します。

停止したジョブに再接続するには、次のコマンドを入力します。

```
> expdp hr ATTACH=EXPFULL
```

親トピック: Oracle Data Pumpエクスポートの使用例

2.7 Oracle Data Pumpエクスポートの構文図

構文図を使用すると、Oracle Data Pump Exportでの有効なSQL構文を理解できます。

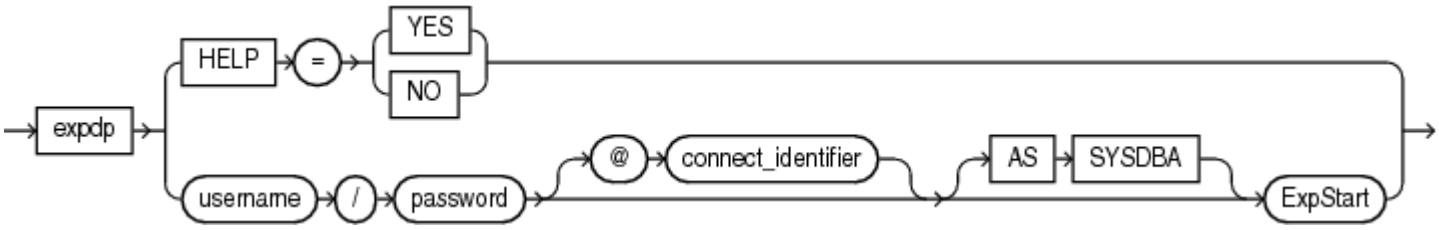
構文図の読み方

構文図とは、SQLの有効な構文を図で示したものです。構文図は、矢印が示す方向に左から右へ読んでください。

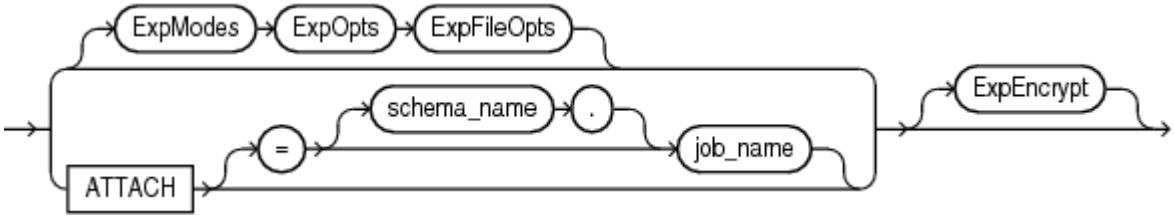
標準のSQL構文表記法の詳細は、次を参照してください。

『Oracle Database SQL言語リファレンス』の「構文図の読み方」

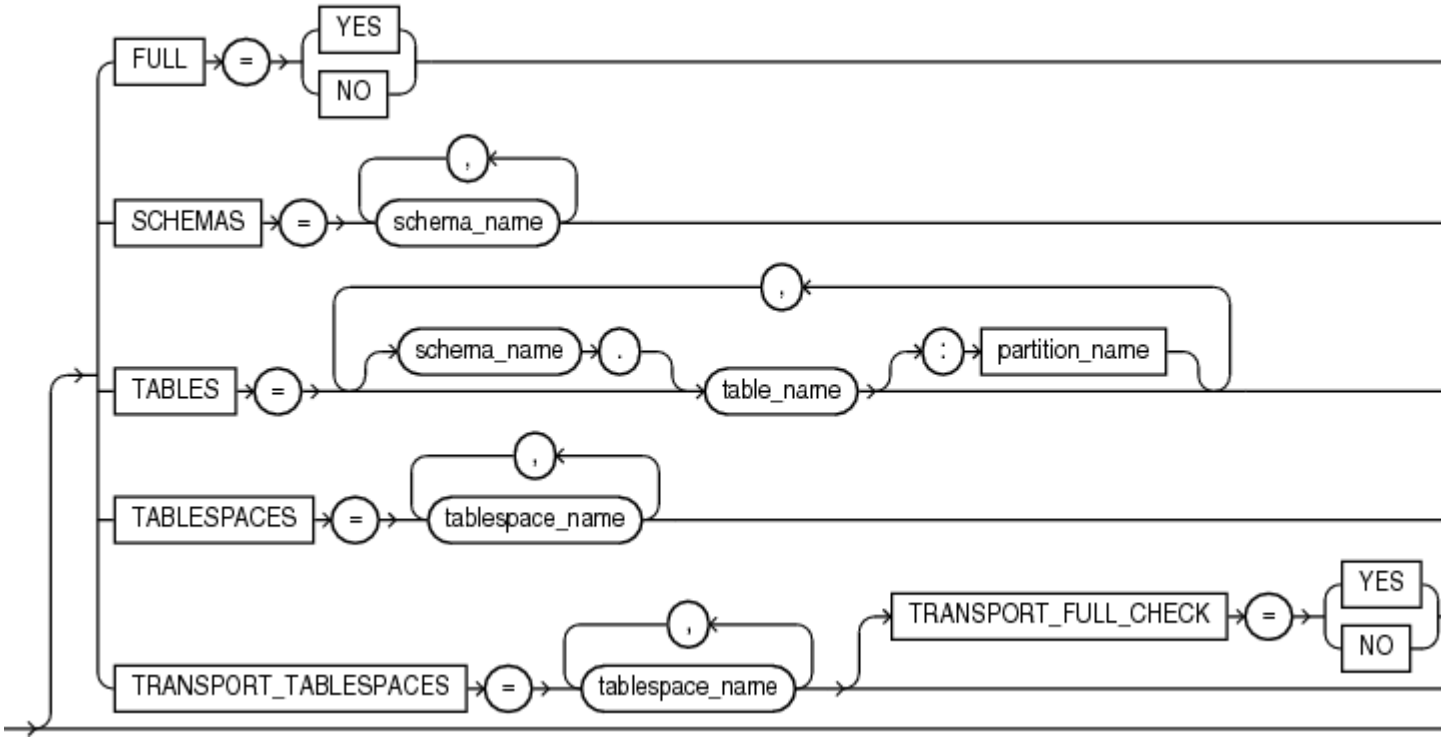
ExpInit



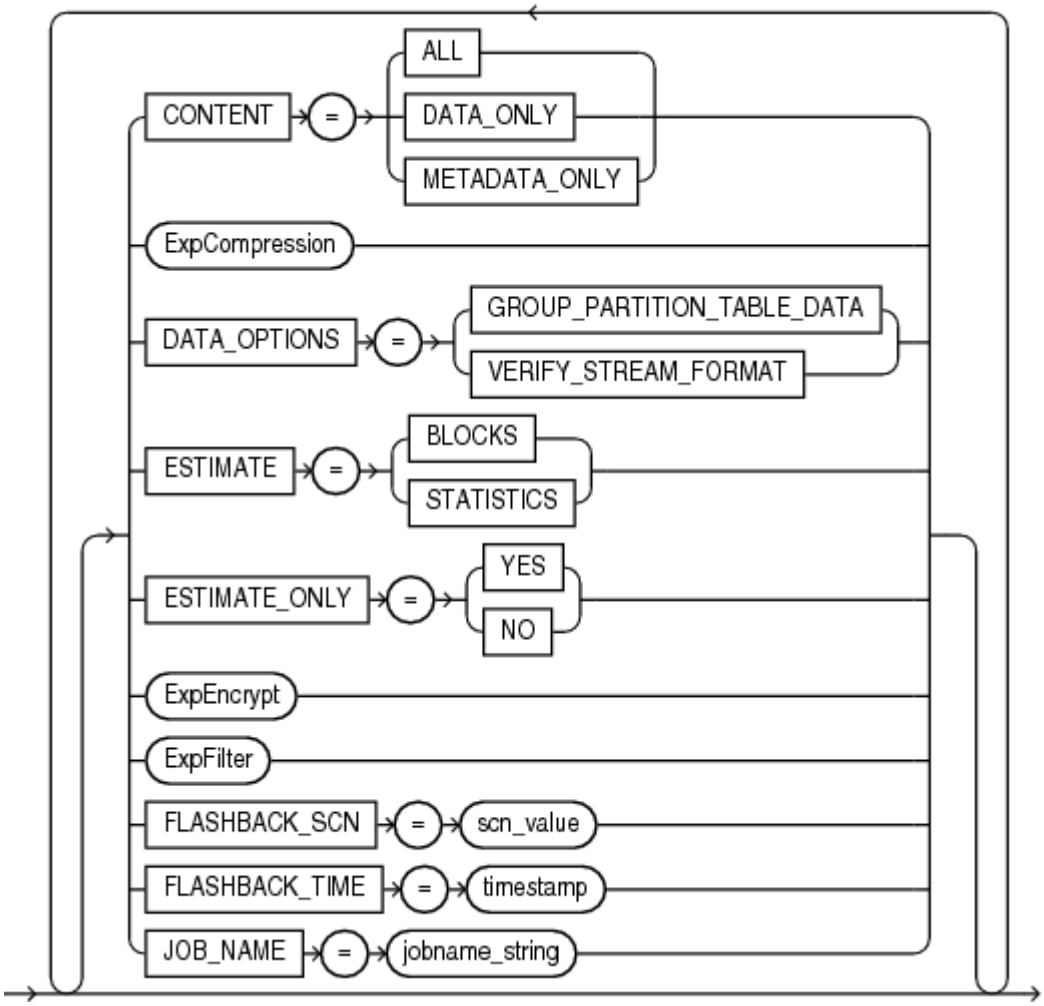
ExpStart



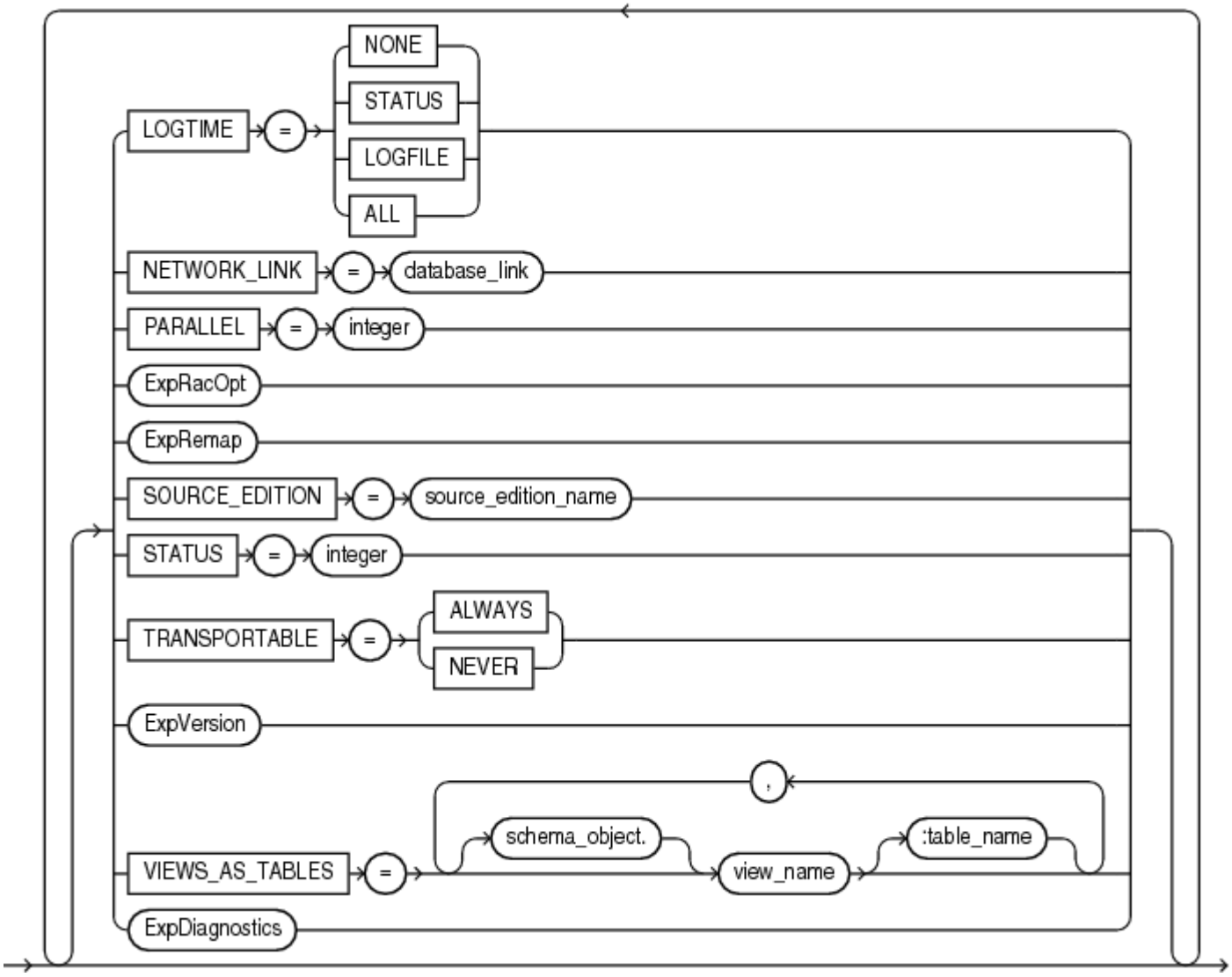
ExpModes



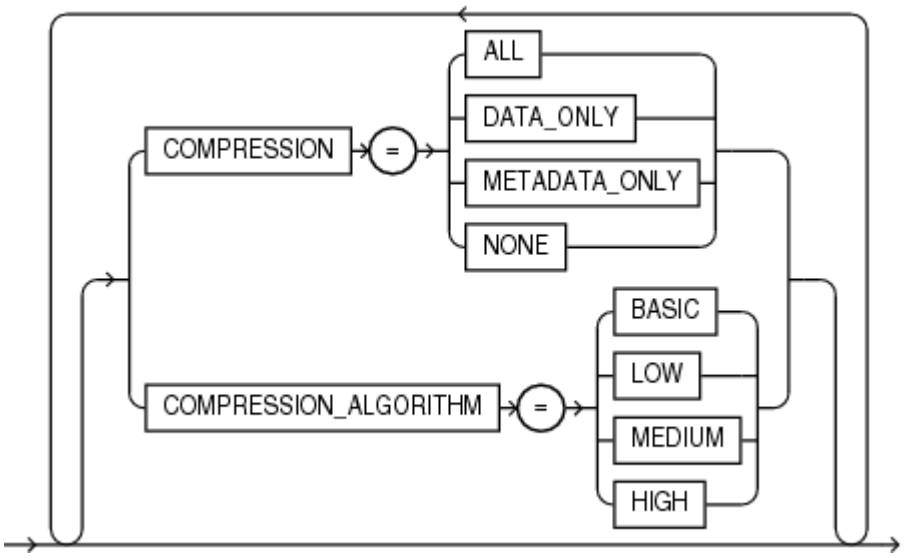
ExpOpts



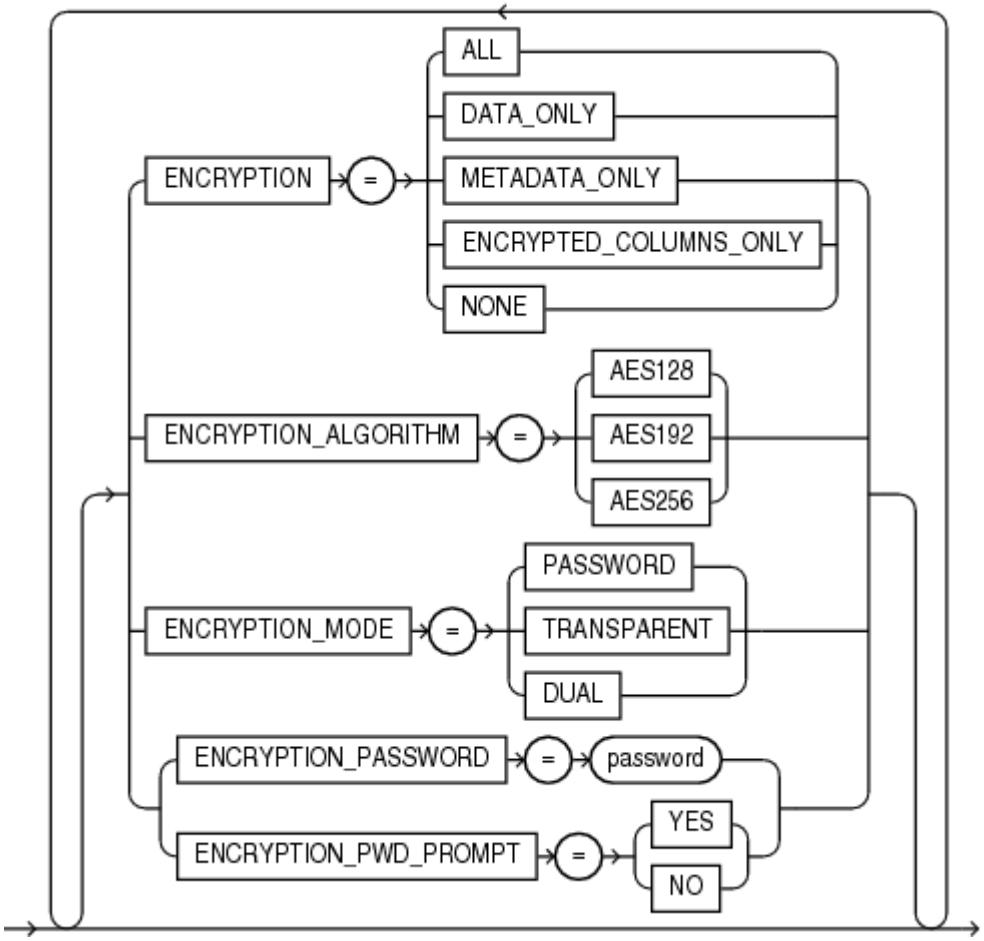
ExpOpts_Cont



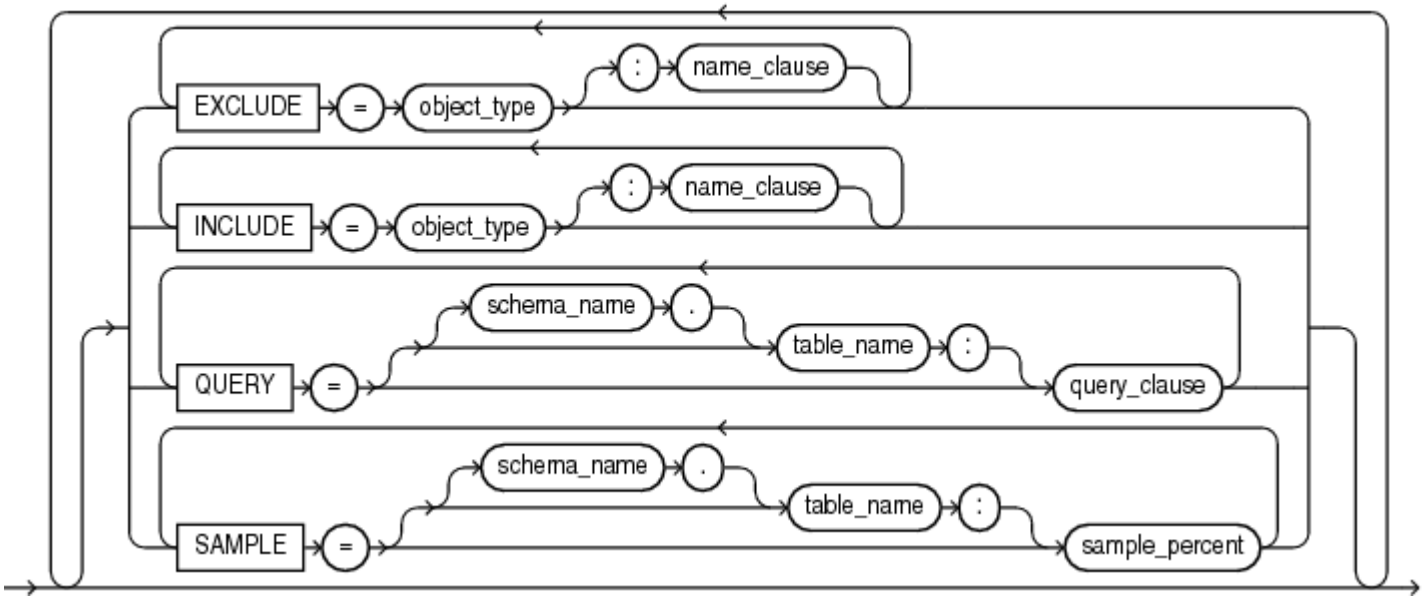
ExpCompression



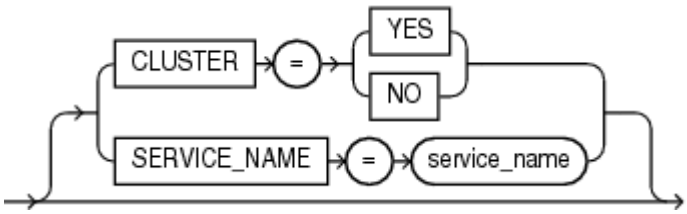
ExpEncrypt



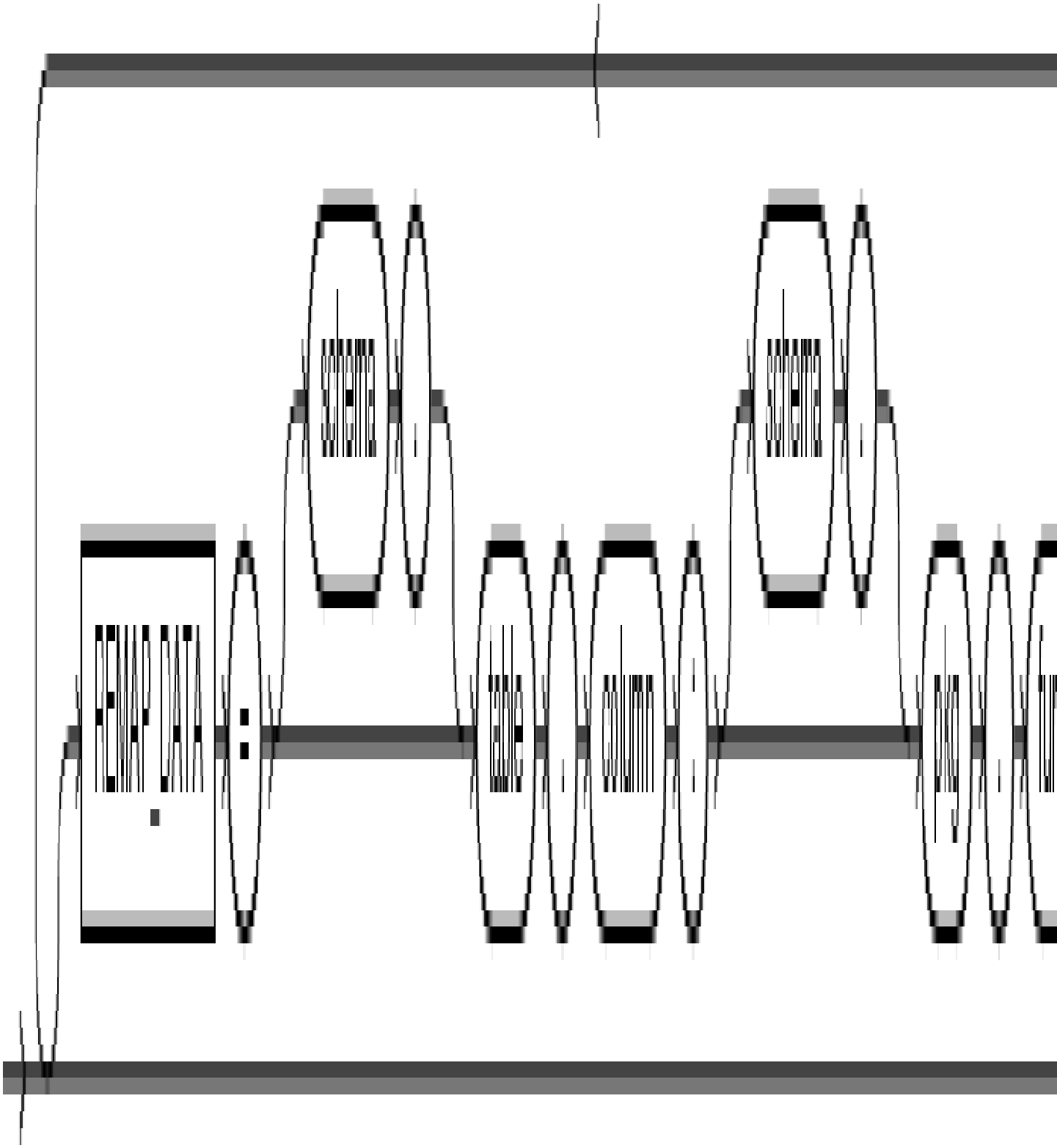
ExpFilter



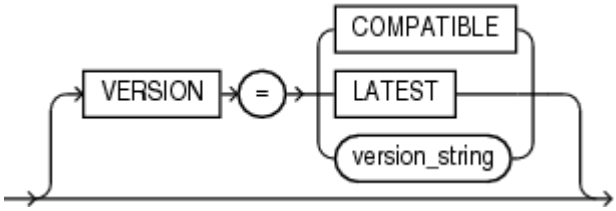
ExpRacOpt



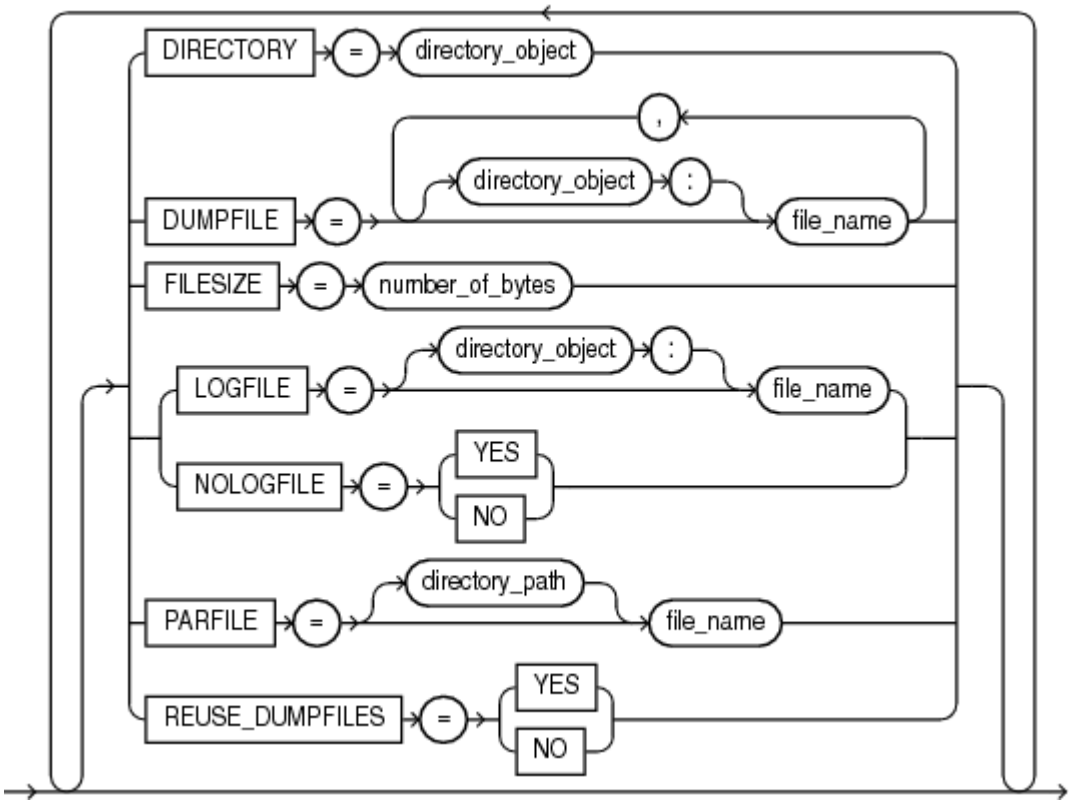
ExpRemap



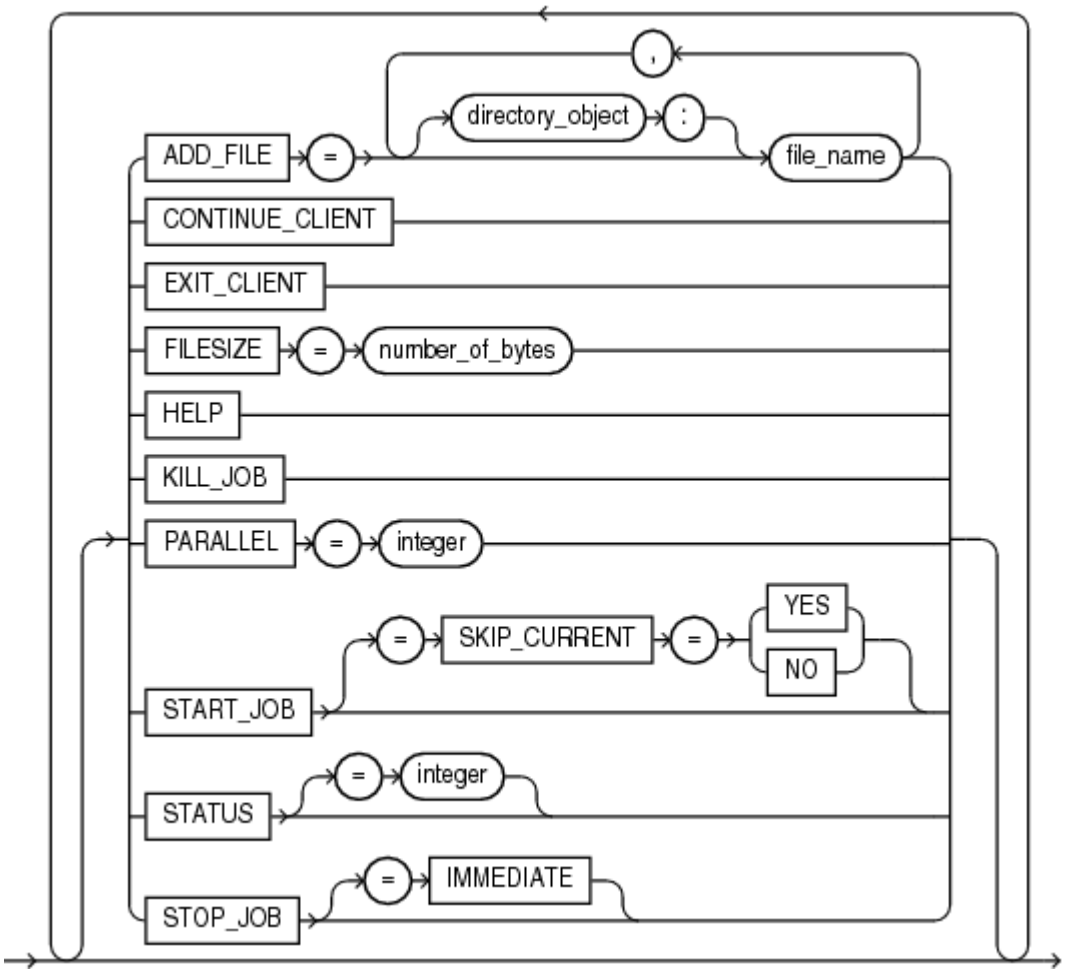
ExpVersion



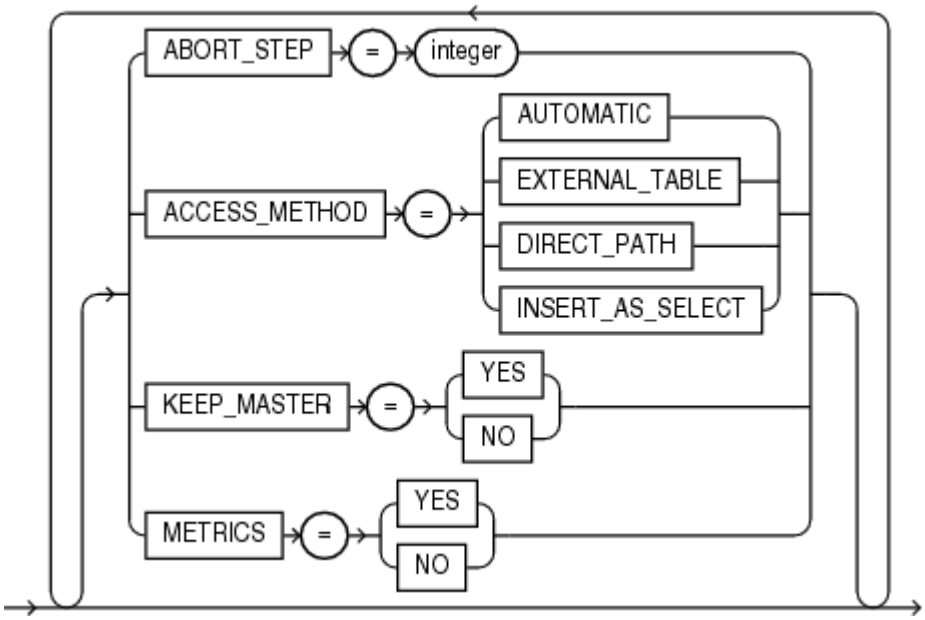
ExpFileOpts



ExpDynOpts



ExpDiagnostics



親トピック: Oracle Data Pumpエクスポート

3 Oracleデータ・ポンプ・インポート

Oracleデータ・ポンプ・インポートを使用すると、エクスポート・ダンプ・ファイル・セットをターゲット・データベースにロードできます。また、中間ファイルを使用することなく、ソース・データベースからターゲット・データベースへの直接的なロードも可能です。

- Oracleデータ・ポンプ・インポートとは
  Oracleデータ・ポンプ・インポートとは、ターゲット・システムにOracleエクスポート・ダンプ・ファイルをロードするためのユーティリティです。
- Oracle Data Pump Importの起動
  Oracle Data Pump Importの起動には、`impdp`コマンドを使用します。
- インポート操作中のフィルタ処理
  Oracleデータ・ポンプ・インポートでは、データとメタデータのフィルタ処理機能を使用して、インポートする情報のタイプを効果的に制限できます。
- Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ
  インポート(`impdp`)用のOracle Data Pumpパラメータを使用して、データ・インポートを管理します。
- Oracle Data Pump Importの対話方式コマンド・モードで使用可能なコマンド
  対話方式コマンド・モードでは、現行のジョブは継続して続行されますが、端末へのロギングは一時停止され、インポート・プロンプト(`Import>`)が表示されます。
- Oracleデータ・ポンプ・インポートの使用例
  次の一般的なシナリオ例を使用して、データの移動にOracleデータ・ポンプ・インポートを使用する方法について学習します。
- Oracle Data Pump Importの構文図
  構文図を使用すると、Oracle Data Pump Importでの有効なSQL構文を理解できます。

親トピック: Oracle Data Pump

3.1 Oracleデータ・ポンプ・インポートとは

Oracleデータ・ポンプ・インポートとは、ターゲット・システムにOracleエクスポート・ダンプ・ファイルをロードするためのユーティリティです。

エクスポート・ダンプ・ファイル・セットは、表データ、データベース・オブジェクトのメタデータ、制御情報を含む1つ以上のディスク・ファイルで構成されています。これらのファイルは独自のバイナリ形式で書き込まれています。Oracleデータ・ポンプ・インポート・ユーティリティは、インポート操作中、これらのファイルを使用してダンプ・ファイル・セット内の各データベース・オブジェクトの位置を特定します。

また、ダンプ・ファイルを介さずにソース・データベースから直接ターゲット・データベースをロードするために、インポートを使用することもできます。このタイプのインポートを**ネットワーク・インポート**と呼びます。

インポートでは、インポート・モードで設定されているとおりにジョブによって、データおよびメタデータのサブセットが、ダンプ・ファイル・セットまたはソース・データベース(ネットワーク・インポートの場合)から移動されるように指定できます。この指定は、インポート・ユーティリティのコマンドによって実装されるデータ・フィルタおよびメタデータ・フィルタを使用することで行います。

親トピック: Oracleデータ・ポンプ・インポート

3.2 Oracleデータ・ポンプ・インポートの開始

Oracle Data Pump Importの起動には、impdpコマンドを使用します。

インポート操作の特性は、指定するインポート・パラメータによって決定されます。これらのパラメータは、コマンドラインまたはパラメータ・ファイルのいずれかで指定できます。

ノート:

- インポート・ユーティリティは、Oracle サポート・サービスから要求された場合以外、SYSDBA として起動しないでください。SYSDBA は内部的に使用され、一般ユーザーとは異なる特別な機能を持ちます。
- NOLOGGING 句を有効にして作成された表または表領域に対してデータ・ポンプ・インポートを実行する場合でも、REDO ログ・ファイルが生成される場合があることに注意してください。このような場合に生成される REDO は、通常、データ・ポンプ制御表のメンテナンスを目的としているか、または基礎となる再帰的領域トランザクション、データ・ディクショナリの変更、およびロギングを必要とする表の索引メンテナンスに関係しています。
- エクスポート・データベースで使用されているタイムゾーン・バージョンが、インポート・データベースで使用されているバージョンより古い場合は、データ型が TIMESTAMP WITH TIMEZONE の列をロードすると、それ以外の場合よりも時間がかかります。この追加時間は、新しいタイムゾーン・ルールによってロードされる値が変更されるかどうかを確認するために必要です。

- Oracle Data Pump Importのインタフェース
  Oracle Data Pump Importは、コマンドライン、パラメータ・ファイルまたは対話方式コマンド・モードを使用して操作できます。
- Oracle Data Pump Importのモード
  Oracle Data Pumpで使用するインポート・モードにより、何をインポートするかを決定します。
- Oracleデータ・ポンプ・インポートのネットワークに関する考慮事項
  OracleデータポンプインポートユーティリティimpdpがOracle*Netまたはネット・サービス名を使用して接続文字列に含まれた接続識別子でインスタンスを識別する方法と、NETWORK_LINKパラメータを使用したインポート操作との相違点について説明します。

親トピック: Oracleデータ・ポンプ・インポート

3.2.1 Oracle Data Pump Importのインタフェース

Oracle Data Pump Importは、コマンドライン、パラメータ・ファイルまたは対話方式コマンド・モードを使用して操作できます。

- コマンドライン・インタフェース: 直接コマンドラインでインポートのパラメータを指定できます。コマンドライン・インタフェースで使用可能なパラメータの詳細。
- パラメータ・ファイル・インタフェース: パラメータ・ファイルでコマンドラインのパラメータを指定できます。パラメータ・ファイルはネストできないため、PARFILEパラメータのみが例外となります。値の指定に引用符が必要なパラメータを指定する場合は、パラメータ・ファイルを使用することをお薦めします。
- 対話方式コマンド・インタフェース: 端末へのロギングを中止してインポート・ユーティリティのプロンプトを表示します。対話方式コマンド・モード固有のコマンドも含めて、様々なコマンドが入力できます。このモードは、コマンドライン・インタフェースまたはパラメータ・ファイル・インタフェースで開始されたインポート操作中に[Ctrl]キーを押しながら[C]キーを押

すと使用可能になります。対話方式コマンド・モードは、実行中のジョブまたは停止されたジョブに接続した場合も使用可能になります。

関連トピック

- Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ
- Oracle Data Pump Import対話方式コマンド・モードで使用可能なコマンド

親トピック: Oracleデータ・ポンプ・インポートの開始

3.2.2 Oracle Data Pump Importのモード

Oracle Data Pumpで使用するインポート・モードにより、何をインポートするかを決定します。

- Oracle Data Pump Importのモードについて
  Oracle Data Pump Importの各モードの、インポート中の動作について学習します。
- 全体インポート・モード
  Oracle Data Pumpで全体インポートを指定するには、FULLパラメータを使用します。
- スキーマ・モード
  Oracle Data Pumpでスキーマ・インポートを指定するには、SCHEMASパラメータを使用します。
- 表モード
  Oracle Data Pumpで表モードでのインポートを指定するには、TABLESパラメータを使用します。
- 表領域モード
  Oracle Data Pumpで表領域モードでのインポートを指定するには、TABLESPACESパラメータを使用します。
- トランスポータブル表領域モード
  Oracle Data Pumpでトランスポータブル表領域モードでのインポートを指定するには、TRANSPORT_TABLESPACESパラメータを使用します。

親トピック: Oracleデータ・ポンプ・インポートの開始

3.2.2.1 Oracle Data Pump Importのモードについて

Oracle Data Pump Importの各モードの、インポート中の動作について学習します。

インポートについて指定するOracle Data Pumpインポート・モードは、操作のソースに適用されます。NETWORK_LINKパラメータを指定すると、そのソースはダンプ・ファイル・セットまたは別のデータベースのいずれかになります。

インポート操作のソースがダンプ・ファイル・セットの場合、モードの指定はオプションです。モードを指定しない場合、インポート・ユーティリティは、エクスポート操作実行時のモードでダンプ・ファイル・セット全体をロードしようとします。

モードは、適切なパラメータを使用してコマンドラインで指定します。

ノート:

全体モード・エクスポートによって作成されたダンプ・ファイルをインポートする場合、インポート操作は SYS アカウントのパスワードをソース・データベースからコピーしようとします。このコピーは、失敗する場合があります(パスワードが共有パスワード・ファイル内にある場合など)。失敗した場合は、インポートの完了後に、ターゲット・データベース上の SYS アカウントのパスワードを任意のパスワードに設定する必要があります。

親トピック: Oracle Data Pump Importのモード

3.2.2.2 全体インポート・モード

Oracle Data Pumpで全体インポートを指定するには、FULLパラメータを使用します。

全体インポート・モードでは、ソース(ダンプ・ファイル・セットまたは別のデータベース)の全内容がターゲット・データベースにロードされます。このモードは、ファイル・ベース・インポートのデフォルトです。ソースが自分のスキーマ以外のスキーマを含む別のデータベースである場合、DATAPUMP_IMP_FULL_DATABASEロールが必要です。

権限のないユーザーの場合、相互スキーマ参照はインポートされません。たとえば、インポートを実行するユーザーのスキーマ内の表にトリガーが定義されていても、そのトリガーが別のユーザーのスキーマに存在している場合はインポートされません。

ターゲット・データベースでは、DATAPUMP_IMP_FULL_DATABASEロールが必要です。全体インポートにNETWORK_LINKパラメータを使用する場合、ソース・データベースでは、DATAPUMP_EXP_FULL_DATABASEロールが必要です

全体モード・インポート時のトランスポータブル・オプションの使用

全体モード・インポート時にトランスポータブル・オプションを使用して、全体トランスポータブル・インポートを実行できます。

ネットワーク・ベースの全体トランスポータブル・インポートでは、FULL=YES、TRANSPORTABLE=ALWAYSおよびTRANSPORT_DATAFILES=datafile_nameパラメータを使用する必要があります。

ファイル・ベースの全体トランスポータブル・インポートでは、TRANSPORT_DATAFILES=datafile_nameパラメータのみを使用する必要があります。データ・ポンプ・インポートでは、TRANSPORTABLE=ALWAYSおよびFULL=Yパラメータの存在が推測されます。

全体トランスポータブル・インポートを実行する場合、次のようないくつかの要件があります。

- 同時にNETWORK_LINKパラメータを指定するか、エクスポート中にインポート対象のダンプ・ファイル・セットがトランスポータブル・オプションを使用して作成されている必要があります。
- ネットワーク・リンクを使用する場合、NETWORK_LINKパラメータで指定されているデータベースがOracle Database 11gリリース2 (11.2.0.3)以上であり、Oracle Data PumpのVERSIONパラメータが12以上に設定されている必要があります。(ネットワーク・インポート以外では、ダンプ・ファイルから暗黙的にVERSION=12が決定されます。)
- ソース・プラットフォームとターゲット・プラットフォームのエンディアンが異なる場合、転送するデータを変換してターゲット・プラットフォームの形式にする必要があります。データを変換するには、DBMS_FILE_TRANSFERパッケージまたはRMAN CONVERTコマンドのどちらかを使用します。
- ソース・プラットフォームとターゲット・プラットフォームでエンディアンが同じでない場合、暗号化された表領域の全体トランスポータブル・インポートは、ネットワーク・モードまたはダンプ・ファイル・モードではサポートされません

全体トランスポータブル・インポートの詳細な例は、*Oracle Database*管理者ガイドを参照してください。

関連トピック

- FULL
- TRANSPORTABLE
- 『Oracle Database管理者ガイド』の「データベース間の表領域の転送」

親トピック: Oracle Data Pump Importのモード

3.2.2.3 スキーマ・モード

Oracle Data Pumpでスキーマ・インポートを指定するには、SCHEMASパラメータを使用します。

スキーマ・インポートでは、指定されたスキーマが所有しているオブジェクトのみがロードされます。ソースは、全体インポート・モー

ド、表モード、表領域モードまたはスキーマ・モードのエクスポート・ダンプ・ファイル・セット、または別のデータベースです。ユーザーにDATAPUMP_IMP_FULL_DATABASEロールが付与されている場合は、スキーマ・リストを指定できます。これにより、スキーマ内のオブジェクトに加えてスキーマ自体(システム権限を含む)もデータベース内に作成されます。

相互スキーマ参照は、残りのスキーマが現行のスキーマに再マップされないかぎり、権限のないユーザーに対してインポートされません。たとえば、インポートを実行するユーザーのスキーマ内の表にトリガーが定義されていても、そのトリガーが別のユーザーのスキーマに存在している場合はインポートされません。

関連トピック

- SCHEMAS

親トピック: Oracle Data Pump Importのモード

3.2.2.4 表モード

Oracle Data Pumpで表モードでのインポートを指定するには、TABLESパラメータを使用します。

表モード・インポートは、TABLESパラメータを使用して指定します。表モードでは、指定した表、パーティションおよびそれらの依存オブジェクトのみがロードされます。ソースは、全体インポート・モード、スキーマ・モード、表領域モードまたは表モードのエクスポート・ダンプ・ファイル・セット、または別のデータベースです。自分のスキーマに存在しない表を指定するには、DATAPUMP_IMP_FULL_DATABASEロールが必要です。

TRANPORTABLE=ALWAYSパラメータをTABLESパラメータと組み合せて指定することで、表モードのインポート中にトランスポータブル・オプションを使用できます。このオプションを使用する場合、NETWORK_LINKパラメータも使用する必要があります。

表および表パーティションのリカバリには、RMANバックアップおよびRMAN RECOVER TABLEコマンドも使用できます。このプロセスの間に、RMANによって、リカバリされたオブジェクトを含むOracle Data Pumpエクスポート・ダンプ・ファイルが作成(およびオプションでインポート)されます。

関連トピック

- TABLES
- TRANSPORTABLE
- データ・ファイル・コピーを使用したデータ移動
- 『Oracle Databaseバックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』

親トピック: Oracle Data Pump Importのモード

3.2.2.5 表領域モード

Oracle Data Pumpで表領域モードでのインポートを指定するには、TABLESPACESパラメータを使用します。

表領域モードのインポートは、TABLESPACESパラメータを使用して指定します。表領域モードでは、指定した表領域内のすべてのオブジェクトが、依存オブジェクトとともにロードされます。ソースは、全体インポート・モード、スキーマ・モード、表領域モードまたは表モードのエクスポート・ダンプ・ファイル・セット、または別のデータベースです。権限のないユーザーの場合、現行のスキーマに再マッピングされていないオブジェクトは処理されません。

関連トピック

- TABLESPACES

親トピック: Oracle Data Pump Importのモード

3.2.2.6 トランスポータブル表領域モード

Oracle Data Pumpでトランスポータブル表領域モードでのインポートを指定するには、TRANSPORT_TABLESPACESパラメータを使用します。

トランスポータブル表領域モードでは、(NETWORK_LINKパラメータで指定された)データベース・リンクを使用するか、メタデータを含むダンプ・ファイルを指定することで、別のデータベースからメタデータがロードされます。TRANSPORT_DATAFILESパラメータで指定した実際のデータ・ファイルは、ターゲット・データベースで使用するために、通常は、データ・ファイルをターゲット・システムにコピーすることによって、ソース・システムで使用可能にする必要があります。

トランスポータブル・ジョブを実行する場合は、インポート・ジョブがターゲット・システムで正常に完了するまで、ソース・システムにデータ・ファイルのコピーを保持することをお薦めします。データ・ファイルのコピーがあると、インポート・ジョブがなんらかの理由で失敗しても、破損していないデータ・ファイルのコピーが残ることになります。

このモードの使用には、DATAPUMP_IMP_FULL_DATABASEロールが必要です。

ノート:

トランスポータブル表領域をエクスポートした後、それよりも古いリリース・レベルのデータベースにインポートすることはできません。ターゲット・データベースのリリース・レベルは、ソース・データベース以上である必要があります。

関連トピック

- Oracle Data Pumpによるタイムスタンプ・データの処理方法
- データ・ファイル・コピーを使用したデータ移動

親トピック: Oracle Data Pump Importのモード

3.2.3 Oracleデータ・ポンプ・インポートのネットワークに関する考慮事項

Oracleデータ・ポンプ・インポート・ユーティリティimpdpがOracle*Netまたはネット・サービス名を使用して接続文字列に含まれた接続識別子でインスタンスを識別する方法と、NETWORK_LINKパラメータを使用したインポート操作との相違点について学習します。

impdpの起動時には、接続文字列で接続識別子を指定できます。この接続識別子は、現在のOracleシステムID (SID)で識別される現在のインスタンスとは別のものにすることができます。

接続識別子の指定には、Oracle*Net接続記述子を使用することも、接続記述子にマップされるネット・サービス名(通常はtnsnames.oraファイルで定義されているもの)を使用することもできます。接続識別子を使用するには、Oracle Net Listenerが実行中であることが必要です(デフォルト・リスナーを起動するには、lsnrctl startと入力します)。次の例では、このタイプの接続を示します(inst1が接続識別子です)。

```
impdp hr@inst1 DIRECTORY=dpump_dir1 DUMPFILE=hr.dmp TABLES=employees
```

インポートによってパスワードの入力を要求されます。

```
Password: password
```

ローカル・インポート・クライアントは接続識別子inst1 (ネット・サービス名)によって識別されるデータベース・インスタンスに接続し、ダンプ・ファイルhr.dmpからinst1にデータをインポートします。

インポート・ユーティリティの起動時に接続識別子を指定することと、NETWORK_LINKパラメータを使用してインポート操作を実

行することは異なります。インポート操作を開始して接続識別子を指定すると、接続識別子によって識別されるデータベース・インスタンスにローカルのインポート・クライアントが接続され、コマンドラインで指定されるダンプ・ファイルからそのデータベース・インスタンスにデータがインポートされます。

その一方、NETWORK_LINKパラメータを使用してインポートを実行すると、インポートはデータベース・リンクを使用して実行されます。ダンプ・ファイルは介在しません。(データベース・リンクとは2つの物理データベース・サーバー間の接続で、データベース・リンクによってクライアントは2つの物理データベース・サーバーに1つの論理データベースとしてアクセスできます。)

関連トピック

- NETWORK_LINK

関連項目:

- データベース・リンクの詳細は、Oracle Database管理者ガイドを参照してください。
- 接続識別子およびOracle Net Listenerの詳細は、『Oracle Database Net Services管理者ガイド』を参照してください。

親トピック: Oracleデータ・ポンプ・インポートの開始

3.3 インポート操作中のフィルタ処理

Oracleデータ・ポンプ・インポートでは、データとメタデータのフィルタ処理機能を使用して、インポートする情報のタイプを効果的に制限できます。

- データ・フィルタ
  データ固有のフィルタ処理は、QUERYおよびSAMPLEパラメータによって実装されます。このパラメータは、表のインポートされる行に対する制限を指定します。
- メタデータ・フィルタ
  メタデータのフィルタ処理は、EXCLUDEおよびINCLUDEパラメータによって実装されます。

親トピック: Oracleデータ・ポンプ・インポート

3.3.1 データ・フィルタ

データ固有のフィルタ処理は、QUERYおよびSAMPLEパラメータによって実装されます。このパラメータは、表のインポートされる行に対する制限を指定します。

メタデータのフィルタ処理のため、間接的にデータのフィルタ処理が実行される場合もあります。この処理では、表オブジェクトおよび関連付けられた行データを含めたり、除外することができます。

各データ・フィルタは、表およびジョブごとにそれぞれ1回指定できます。同じ名前を使用する異なるフィルタが特定の表とジョブ全体の両方に適用された場合は、特定の表に対して提供されたフィルタ・パラメータが優先されます。

親トピック: インポート操作中のフィルタ処理

3.3.2 メタデータ・フィルタ

メタデータのフィルタ処理は、EXCLUDEおよびINCLUDEパラメータによって実装されます。

データ・ポンプ・インポート・ユーティリティで提供されるメタデータのフィルタ処理機能は、オリジナルのインポート・ユーティリティと比

較すると大幅に拡張されています。EXCLUDEおよびINCLUDEは、相互に排他的なパラメータです。

メタデータ・フィルタは、データ・ポンプ操作に含めるか、データ・ポンプ操作から除外するオブジェクトの集合を特定します。たとえば、パッケージ仕様またはパッケージ本体を含まない全体インポートを要求できます。

フィルタを正しく使用して必要な結果を得た場合は、識別されたオブジェクトの依存オブジェクトも、識別されたオブジェクトとともに処理されます。たとえば、パッケージを操作に含めるようにフィルタで指定すると、そのパッケージに対する権限も含まれます。同様に、フィルタで表を除外すると、その表に対する索引、制約、権限およびトリガーも除外されます。

1つのオブジェクト型に対して複数のフィルタが指定されている場合は、それらのフィルタに対して暗黙的なAND処理が適用されます。つまり、ジョブに関連するオブジェクトは、オブジェクト型に適用されるすべてのフィルタで処理される必要があります。

1つのジョブ内には、同じフィルタ名を複数回指定できます。

DATABASE_EXPORT_OBJECTS(全体モードの場合)、SCHEMA_EXPORT_OBJECTS(スキーマ・モードの場合)、TABLE_EXPORT_OBJECTS(表および表領域モードの場合)ビューを問い合せて、有効なオブジェクト・タイプ値の一覧を表示できます。OBJECT_PATH列にリストされる値が有効なオブジェクト・タイプです。完全なオブジェクト・パス名は、インポート・モードでなくエクスポート・モードによって決定されます。

関連トピック

- メタデータ・フィルタ
- EXCLUDE
- INCLUDE

親トピック: インポート操作中のフィルタ処理

3.4 Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

データ・インポートを管理するには、インポート(impdp)用のOracle Data Pumpパラメータを使用します。

- インポート・コマンドライン・モードについて
  大/小文字の区別、引用符、エスケープ文字、例の使用方法に関する情報など、コマンドライン・モードでOracle Data Pump Importパラメータを使用する方法について学習します。
- ABORT_STEP
  Oracle Data Pumpインポートのコマンドライン・モードのABORT_STEPパラメータでは、ジョブの初期化後に、そのジョブを停止します。ジョブを停止すると、データがインポートされる前にデータ・ポンプ制御ジョブ表を問い合せることができます。
- ACCESS_METHOD
  Orale Data Pump Importのコマンドライン・モードのACCESS_METHODパラメータでは、インポートでデータのロードに特定の方法を使用することを指定します
- ATTACH
  Oracle Data Pumpインポートのコマンドライン・モードのATTACHパラメータでは、クライアント・セッションを既存のインポート・ジョブに接続し、自動的に対話方式コマンド・モードにします。
- CLUSTER
  Oracle Data Pump Importのコマンドライン・モードのCLUSTERパラメータでは、データ・ポンプでOracle Real Application Clusters (Oracle RAC)リソースを使用できるかどうかと、ワーカーを他のOracle RACインスタンス上で開始できるかどうかを指定します。

- CONTENT
  Oracle Data Pump Importのコマンドライン・モードのCONTENTパラメータを使用すると、インポート操作中にロード対象をフィルタ処理できます。
- CREDENTIAL
  Oracle Data Pumpインポートのコマンドライン・モードのCREDENTIALパラメータでは、データベース・ユーザーが所有する資格証明オブジェクトの名前を指定します。インポート・ユーティリティは、このオブジェクト名を使用してクラウド・ストレージにインポートされるダンプ・ファイルセットのファイルを処理します。
- DATA_OPTIONS
  Oracle Data Pump Importのコマンドライン・モードのDATA_OPTIONSパラメータでは、インポート操作中に特定のタイプのデータをどのように処理するかを指定します。
- DIRECTORY
  Oracle Data Pump Importのコマンドライン・モードのDIRECTORYパラメータでは、インポート・ジョブでのダンプ・ファイル・セットの検索先およびログ・ファイルとSQLファイルの作成先となるデフォルトの場所を指定します。
- DUMPFILE
  Oracle Data Pump Importのコマンドライン・モードのDUMPFILEパラメータでは、エクスポート・ユーティリティによって作成されたダンプ・ファイル・セットの名前を指定し、オプションで、これらのディレクトリ・オブジェクトを指定します。
- ENABLE_SECURE_ROLES
  Oracle Data Pumpインポートのコマンドライン・ユーティリティのENABLE_SECURE_ROLESパラメータは、エクスポート時に保護されたロールを誤って使用しないようにします。
- ENCRYPTION_PASSWORD
  Oracle Data Pump Importのコマンドライン・モードのENCRYPTION_PASSWORDパラメータでは、ダンプ・ファイル・セット内の暗号化列のデータにアクセスするためのパスワードを指定します。これにより、暗号化されたダンプ・ファイル・セットへの不正なアクセスを防ぎます。
- ENCRYPTION_PWD_PROMPT
  Oracle Data Pumpインポートのコマンドライン・モードのENCRYPTION_PWD_PROMPTパラメータでは、データ・ポンプによって暗号化パスワードの入力を要求するかどうかを指定します。
- ESTIMATE
  Oracle Data Pumpインポートのコマンドライン・モードのESTIMATEパラメータでは、ネットワーク・インポート操作のソース・システムに対して、インポート中に生成されるデータ量を見積もるように指定します。
- EXCLUDE
  Oracle Data Pumpインポートのコマンドライン・モードのEXCLUDEパラメータを使用すると、インポート・ジョブから除外するオブジェクトとオブジェクト型を指定して、インポート対象のメタデータをフィルタ処理できます。
- FLASHBACK_SCN
  Oracle Data Pump Importのコマンドライン・モードのFLASHBACK_SCNでは、インポート・ユーティリティで使用されるシステム変更番号(SCN)を指定して、フラッシュバック・ユーティリティを使用可能にします。
- FLASHBACK_TIME
  Oracle Data Pump Importのコマンドライン・モードのFLASHBACK_TIMEパラメータでは、インポート・ユーティリティで使用されるシステム変更番号(SCN)を指定して、フラッシュバック・ユーティリティを使用可能にします。
- FULL
  Oracle Data Pump Importのコマンドライン・モードのFULLパラメータにより、全データベース・インポートを実行することを指定します。
- HELP
  Oracle Data Pump Importのコマンドライン・モードのHELPパラメータにより、インポート・ユーティリティのオンライン・

ヘルプを表示します。

- INCLUDE
  Oracle Data Pumpインポートのコマンドライン・モードのINCLUDEパラメータを使用すると、現行のインポート・モードにオブジェクトとオブジェクト型を指定して、インポート対象のメタデータをフィルタ処理できます。
- JOB_NAME
  Oracle Data Pumpインポートのコマンドライン・モードのJOB_NAMEパラメータを使用して、後続処理でインポート・ジョブを指定します。
- KEEP_MASTER
  Oracle Data Pumpインポートのコマンドライン・モードのKEEP_MASTERパラメータは、正常に完了したOracle Data Pumpジョブの最後にデータ・ポンプ制御ジョブ表を削除するか保持するかを指定します。
- LOGFILE
  Oracle Data Pump Importのコマンドライン・モードのLOGFILEパラメータでは、インポート・ジョブのログ・ファイルの名前を指定し、オプションでディレクトリ・オブジェクトを指定します。
- LOGTIME
  Oracle Data Pump Importのコマンドライン・モードのLOGTIMEパラメータにより、インポート中にメッセージをタイムスタンプとともに表示することを指定します。
- MASTER_ONLY
  Oracle Data Pumpインポートのコマンドライン・モードのMASTER_ONLYパラメータは、データ・ポンプ制御ジョブ表の内容を確認できるように、データ・ポンプ制御ジョブ表のみをインポートしてジョブを停止するかどうかを示します。
- METRICS
  Oracle Data Pump Importのコマンドライン・モードのMETRICSパラメータにより、ジョブに関する詳細情報をログ・ファイルに記録するかどうかを指定します。
- NETWORK_LINK
  Oracleデータ・ポンプ・インポートのコマンドライン・モードでNETWORK_LINKパラメータを使用すると、有効なデータベース・リンクで識別されるソース・データベースからのインポートが可能になります。
- NOLOGFILE
  Oracle Data Pump Importのコマンドライン・モードのNOLOGFILEパラメータにより、デフォルトでログ・ファイルを作成するかどうかを指定します。
- PARALLEL
  Oracleデータ・ポンプ・インポートのコマンドライン・モードのPARALLELパラメータでは、パラレルにロードできるワーカー・プロセスの最大数を設定します。
- PARALLEL_THRESHOLD
  Oracle Data Pump Importコマンドライン・ユーティリティのPARALLEL_THRESHOLDパラメータでは、表サイズに基づいてデータ・ポンプが潜在的なパラレルDMLを計算するために使用する除数のサイズを指定します。
- PARFILE
  Oracle Data Pump Importのコマンドライン・モードのPARFILEパラメータでは、インポート・パラメータ・ファイルの名前を指定します。
- PARTITION_OPTIONS
  Oracle Data Pump Importのコマンドライン・モードのPARTITION_OPTIONSパラメータでは、インポート操作中に表パーティションを作成する方法を指定します。
- QUERY
  Oracle Data Pump Importのコマンドライン・モードのQUERYパラメータを使用すると、インポート対象のデータをフィルタ処理する問合せ句を指定できます。

- REMAP_DATA
  Oracle Data Pump Importのコマンドライン・モードのREMAP_DATAパラメータを使用すると、新規データベースへの挿入時に、データを再マップできます。
- REMAP_DATAFILE
  Oracle Data Pump Importのコマンドライン・モードのREMAP_DATAFILEパラメータにより、ソース・データ・ファイルを参照するすべてのSQL文において、ソース・データファイル名がターゲット・データファイル名に変更されます。
- REMAP_DIRECTORY
  Oracle Data Pump Importのコマンドライン・モードのREMAP_DIRECTORYパラメータを使用すると、プラットフォーム間でデータベースを移動するときにディレクトリを再マップできます。
- REMAP_SCHEMA
  Oracle Data Pump Importのコマンドライン・モードのREMAP_SCHEMAパラメータにより、すべてのオブジェクトをソース・スキーマからターゲット・スキーマにロードします。
- REMAP_TABLE
  Oracle Data Pump Importのコマンドライン・モードのREMAP_TABLEパラメータを使用すると、インポート操作中に表の名前を変更できます。
- REMAP_TABLESPACE
  Oracle Data Pump Importのコマンドライン・モードのREMAP_TABLESPACEパラメータにより、インポート用に選択されたすべてのオブジェクトを再マップし、ソース表領域内の永続データがターゲット表領域内に作成されるようにします。
- SCHEMAS
  Oracle Data Pump Importのコマンドライン・モードのSCHEMASパラメータにより、スキーマ・モード・インポートを実行することを指定します。
- SERVICE_NAME
  Oracle Data Pump Importのコマンドライン・モードのSERVICE_NAMEパラメータでは、CLUSTERパラメータとともに使用するサービス名を指定します。
- SKIP_UNUSABLE_INDEXES
  Oracle Data Pump Importのコマンドライン・モードのSKIP_UNUSABLE_INDEXESパラメータにより、(システムまたはユーザーのどちらかによって)索引使用禁止状態に設定されている索引を持つ表のロードをインポート・ユーティリティでスキップするかどうかを指定します。
- SOURCE_EDITION
  Oracle Data Pump Importのコマンドライン・モードのSOURCE_EDITIONパラメータでは、オブジェクトのフェッチ元となるリモート・ノード上のデータベース・エディションを指定します。
- SQLFILE
  Oracle Data Pump Importのコマンドライン・モードのSQLFILEパラメータでは、選択した他のインポート・パラメータに基づいて、インポート・ユーティリティによって実行準備が整えられたすべてのSQL DDLの書込み先となるファイルを指定します。
- STATUS
  Oracle Data Pump Importのコマンドライン・モードのSTATUSパラメータでは、ジョブの状態を表示する頻度を指定します。
- STREAMS_CONFIGURATION
  Oracle Data Pump Importのコマンドライン・モードのSTREAMS_CONFIGURATIONパラメータでは、エクスポート・ダンプ・ファイル内に存在する可能性のあるすべてのGoldenGateレプリケーション・メタデータをインポートするかどうかを指定します。

- TABLE_EXISTS_ACTION
  Oracle Data Pump Importのコマンドライン・モードのTABLE_EXISTS_ACTIONパラメータでは、インポート・ユーティリティで作成しようとしている表がすでに存在する場合の動作を指定します。
- REUSE_DATAFILES
  Oracle Data Pump Importのコマンドライン・モードのREUSE_DATAFILESパラメータでは、インポート・ジョブで表領域の作成に既存のデータ・ファイルを再利用するかどうかを指定します。
- TABLES
  Oracle Data Pump Importのコマンドライン・モードのTABLESパラメータにより、表モード・インポートを実行することを指定します。
- TABLESPACES
  Oracle Data Pump Importのコマンドライン・モードのTABLESPACESパラメータにより、表領域モード・インポートを実行することを指定します。
- TARGET_EDITION
  Oracle Data Pump Importのコマンドライン・モードのTARGET_EDITIONパラメータにより、オブジェクトをインポートするデータベース・エディションを指定します。
- TRANSFORM
  Oracle Data Pump Importのコマンドライン・モードのTRANSFORMパラメータを使用すると、インポート中のオブジェクトに対するオブジェクト作成DDLを変更できます。
- TRANSPORT_DATAFILES
  Oracle Data Pump Importのコマンドライン・モードのTRANSPORT_DATAFILESパラメータでは、エクスポート中にTRANSPORTABLE=ALWAYSが設定された場合にターゲット・データベースにインポートするデータ・ファイルのリストを指定します。
- TRANSPORT_FULL_CHECK
  Oracle Data Pump Importのコマンドライン・モードのTRANSPORT_FULL_CHECKパラメータでは、指定したトランスポータブル表領域セットが他の表領域内のオブジェクトによって参照されていることを確認するかどうかを指定します。
- TRANSPORT_TABLESPACES
  Oracle Data Pump Importのコマンドライン・モードのTRANSPORT_TABLESPACESパラメータにより、データベース・リンク経由でトランスポータブル表領域モードでのインポートを実行することを指定します。
- TRANSPORTABLE
  Oracle Data Pump Importのコマンドライン・モードのTRANSPORTABLEパラメータにより、表モード・インポートまたは全体モード・インポートの際にトランスポータブル・オプションを使用するかどうかを指定します。
- VERSION
  Oracle Data Pump Importのコマンドライン・モードのVERSIONパラメータでは、インポートするデータベース・オブジェクトのバージョンを指定します。
- VIEWS_AS_TABLES (ネットワーク・インポート)
  Oracle Data Pump Importのコマンドライン・モードのVIEWS_AS_TABLES (ネットワーク・インポート)パラメータでは、1つ以上のビューを表としてインポートすることを指定します。

関連トピック

- PARFILE
- ダンプ・ファイル、ログ・ファイルおよびSQLファイルのデフォルトの位置の理解
- Oracleデータ・ポンプ・インポートの使用例
- Oracle Data Pump Importの構文図

3.4.1 インポート・コマンドライン・モードについて

大/小文字の区別、引用符、エスケープ文字、例の使用方法に関する情報など、コマンドライン・モードでOracle Data Pump Importパラメータを使用する方法について学習します。

Oracle Data Pumpインポート・パラメータを使用する前に、次の項を参照してください。

- インポート・パラメータの指定
- データ・ポンプ・コマンドラインでの引用符の使用

ここで説明する内容の多くは、パラメータの使用例を含みます。使用例を実行するために必要な環境の設定の詳細は、次の項を参照してください。

- インポート・パラメータの使用例

インポート・パラメータの指定

値を複数個指定できるパラメータでは、カンマまたは空白で値を区切ることができます。たとえば、`TABLES=employees,jobs`または`TABLES=employees jobs`のように指定できます。

入力するパラメータごとに、等号(=)と値を入力する必要があります。それ以外の方法では、1つ前のパラメータ指定がどこで完了し、新しいパラメータ指定がどこから始まるかをデータ・ポンプに伝えることができません。たとえば、次のコマンドラインの場合、`NOLOGFILE`は有効なパラメータですが、`DUMPFILE`パラメータに対するダンプ・ファイル名がもう1つあると解釈されます。

```
impdp DIRECTORY=dpumpdir DUMPFILE=test.dmp NOLOGFILE TABLES=employees
```

その結果、`test.dmp`および`nologfile.dmp`という2つのダンプ・ファイルが作成されることになります。

これを回避するには、`NOLOGFILE=YES`または`NOLOGFILE=NO`のいずれかを指定します。

パラメータ値を指定する場合の大/小文字の区別

パラメータ値として入力される表領域名、スキーマ名、表名などについて、Oracle Data Pumpは、小文字または大/小文字混在で入力された値をデフォルトで大文字に変更します。たとえば、`TABLE=hr.employees`と入力すると、それは`TABLE=HR.EMPLOYEES`に変更されます。大/小文字を維持するには、引用符で値を囲む必要があります。たとえば、`TABLE="hr.employees"`とすると、表名がすべて小文字のまま維持されます。入力する名前は、データベースに格納されている名前と完全に一致する必要があります。

データ・ポンプ・コマンドラインでの引用符の使用

一部のオペレーティング・システムでは引用符が特殊文字として処理されるため、バックスラッシュ(¥)などのエスケープ文字を前に付けないと、アプリケーションに正しく渡されません。これは、コマンドラインとパラメータ・ファイル内の両方に当てはまります。また、オペレーティング・システムによっては、コマンドラインで、特殊文字を含むパラメータ値全体を一重引用符または二重引用符で囲む必要があります。

次の例では、これらの概念を説明します。特定のオペレーティング・システムに該当しない場合があること、および各ユーザーに固有の運用環境は想定していないことに注意してください。

次のように、パラメータ・ファイルでTABLESパラメータを指定するとします。

```
TABLES = ¥"MixedCaseTableName¥"
```

これをコマンドラインで指定するとき、一部のオペレーティング・システムでは、次のように一重引用符で囲む必要があります。

```
TABLES = '¥"MixedCaseTableName¥"'
```

コマンドラインで追加の引用符を指定せずに手順を続行するには、パラメータ・ファイルを使用することをお薦めします。また、パラメータ・ファイルを使用し、指定するパラメータ値の文字列の最初の文字に引用符を使用しない場合(たとえば、TABLES=scott."EmP")は、エスケープ文字の使用が不要になるシステムがあることに注意してください。

インポート・パラメータの使用例

各項に示す例を試行する場合は、次の内容に注意してください。

- 例に示すようにユーザー名およびパラメータを入力した後、インポート・ユーティリティが起動され、パスワードの入力が要求されます。データベース接続が行われる前に、パスワードを入力する必要があります。
- ここに示す例の多くは、Oracle Databaseのインストール時にデフォルトでインストールされるシード・データベースのサンプル・スキーマを使用しています。特に、人事管理(hr)スキーマを頻繁に使用します。
- インポートするダンプ・ファイルを指定する例では、そのダンプ・ファイルが存在するものとします。可能なかぎり、Exportの例を実行して生成されるダンプ・ファイルを使用します。
- この例では、ディレクトリ・オブジェクトdpump_dir1およびdpump_dir2がすでに存在し、これらのディレクトリ・オブジェクトについてのREAD権限およびWRITE権限が、hrユーザーに付与されているものとします。
- 一部の例では、DATAPUMP_EXP_FULL_DATABASEロールおよびDATAPUMP_IMP_FULL_DATABASEロールが必要です。その例では、hrユーザーにこれらのロールが付与されているものとします。

必要に応じて、これらのディレクトリ・オブジェクトの作成と、必要な権限やロールの割当てをDBAに依頼します。

特に指定がないかぎり、これらのパラメータはパラメータ・ファイルでも指定できます。

関連項目:

Oracle Databaseサンプル・スキーマ

システムでの特殊文字および予約文字の処理方法の詳細は、ご使用のオペレーティング・システム固有のOracleマニュアルを参照してください。

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.2 ABORT_STEP

Oracle Data Pumpインポートのコマンドライン・モードのABORT_STEPパラメータでは、ジョブの初期化後に、そのジョブを停止します。ジョブを停止すると、データがインポートされる前にデータ・ポンプ制御ジョブ表を問い合せることができます。

デフォルト

Null

用途

ジョブの初期化後に、そのジョブを停止します。ジョブを停止すると、データがインポートされる前にデータ・ポンプ制御ジョブ表を問い合せることができます。

構文および説明

```
ABORT_STEP=[n | -1]
```

可能な値は、データ・ポンプ制御ジョブ表のプロセス順序番号に対応します。各番号の使用結果は次のとおりです。

- n: 値がゼロ以上の場合、インポート操作が開始されます。ジョブは、対応するプロセス順序番号とともにデータ・ポンプ制御ジョブ表に格納されているオブジェクトで停止されます。
- -1インポート・ジョブでNETWORK_LINKを使用する場合: ジョブは、その設定後、いずれかのオブジェクトがインポートされる前に中断されます。
- -1 インポート・ジョブがNETWORK_LINKを使用しないインポートの場合: ジョブは、マスター表をロードしてフィルタを適用した後に中断されます。

制限事項

- なし

例

```
> impdp hr SCHEMAS=hr DIRECTORY=dpump_dir1 LOGFILE=schemas.log
DUMPFILE=expdat.dmp ABORT_STEP=-1
```

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.3 ACCESS_METHOD

Orale Data Pump Importのコマンドライン・モードのACCESS_METHODパラメータでは、インポートでデータのロードに特定の方法を使用することを指定します

デフォルト

AUTOMATIC

用途

インポート・ユーティリティで特定のメソッドを使用してデータをロードするように指定します。

構文および説明

```
ACCESS_METHOD=[AUTOMATIC | DIRECT_PATH | EXTERNAL_TABLE | CONVENTIONAL_PATH |
INSERT_AS_SELECT]
```

デフォルトのメソッドがなんらかの理由で動作しない場合に別のネイティブ・メソッドを試行できるように、ACCESS_METHODパラメータが提供されます。指定したアクセス方法で表のデータをロードできない場合は、データに表のエラーが表示され、次の作業項目に進みます。

使用可能なオプションは次のとおりです。

- AUTOMATIC: このアクセス方法はデフォルトです。それぞれの表について、データをロードする最適な方法がデータ・ポンプによって判断されます。できるだけAUTOMATICを使用することをお薦めします。これは、データ・ポンプが最も効果的な方法を自動的に選択できるようになるためです。
- DIRECT_PATH: データ・ポンプによって、すべての表に対して、ダイレクト・パス・ロードが使用されます。
- EXTERNAL_TABLE: データ・ポンプによって、ダンプ・ファイルに格納されているデータに対する外部表が作成され、SQL文INSERT AS SELECTを使用してデータが表にロードされます。データ・ポンプによって、APPENDヒントがINSERT文に適用されます。
- CONVENTIONAL_PATH: データ・ポンプによって、ダンプ・ファイルに格納されているデータに対する外部表が作成され、外部表から1つずつ行が読み取られます。データ・ポンプでは、行を読み取るたびにinsert文を実行して、その行を

ターゲット表にロードします。この方法は、データのロードに時間がかかるものの、ダイレクト・パスおよび外部表によってロードできないデータをロードする唯一の方法となります。

- INSERT_AS_SELECT: データ・ポンプによって、リモート・データベースからデータを選択するSQL文INSERT AS SELECTを実行することによって表がロードされ、ターゲット表に挿入されます。このオプションは、ネットワーク・モード・インポートでのみ使用できます。ネットワーク経由でデータを移動するときにDIRECT_PATHの使用を無効にする場合は、これを使用します。

制限事項

- ネットワーク・モード・インポートで有効なオプションは、AUTOMATIC、DIRECT_PATHおよびINSERT_AS_SELECTです。
- ダンプ・ファイルからインポートするときに有効なオプションは、AUTOMATIC、DIRECT_PATH、EXTERNAL_TABLEおよびCONVENTIONAL_PATHのみです。
- ネットワーク・インポートでACCESS_METHODパラメータを使用するには、Oracle Database 12c リリース2 (12.2.0.1)以上を使用している必要があります。
- Oracle Data Pump ImportのACCESS_METHODパラメータは、トランスポータブル表領域のジョブに対して有効ではありません。

例

次の例では、Oracle Data Pumpで既存の表SALESの複数のパーティションのデータを同時にロードできます。

```
impdp hr SCHEMAS=hr DIRECTORY=dpump_dir1 LOGFILE=schemas.log
DUMPFILE=expdat.dmp ACCESS_METHOD=CONVENTIONAL
```

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.4 ATTACH

Oracle Data Pump Importインポートのコマンドライン・モードのATTACHパラメータでは、クライアント・セッションを既存のインポート・ジョブに接続し、自動的に対話方式コマンド・モードにします。

デフォルト

実行中のジョブが1つのみの場合は、ユーザーのスキーマの現在のジョブ。

用途

このコマンドは、クライアント・セッションを既存のインポート・ジョブに接続し、自動的に対話方式コマンド・モードにします。

構文および説明

```
ATTACH [=[schema_name.]job_name]
```

schema_nameは、接続しているスキーマが、自分のスキーマにない場合に指定します。これを行うには、DATAPUMP_IMP_FULL_DATABASEロールが必要となります。

job_nameは、スキーマに対応する実行中ジョブが1つのみで、そのジョブがアクティブな場合、指定する必要はありません。停止しているジョブに接続する場合は、このジョブ名を指定する必要があります。DBA_DATAPUMP_JOBSビューまたはUSER_DATAPUMP_JOBSビューを問い合せて、Oracle Data Pumpジョブ名の一覧を表示できます。

ジョブに接続している場合、インポート・ユーティリティでは、ジョブの説明が表示され、次にインポート・プロンプトが表示されます。

制限事項

- ATTACHパラメータを指定する場合、コマンドラインで他に指定できるデータ・ポンプ・パラメータは、ENCRYPTION_PASSWORDのみです。
- 接続するジョブが最初に暗号化パスワードを使用して起動している場合、そのジョブへの接続時に、コマンドライン上のENCRYPTION_PASSWORDパラメータを再入力してそのパスワードを再指定する必要があります。
- そのジョブが実行中でなければ、別のスキーマのジョブに接続することはできません。
- ジョブのダンプ・ファイル・セットまたはマスター表が削除されている場合、接続操作は失敗します。
- マスター表を変更すると、それがどのような変更であっても、予期しない結果になる場合があります。

例

次に、ATTACHパラメータの使用例を示します。

```
> impdp hr ATTACH=import_job
```

この例では、import_jobというジョブが、hrスキーマに存在するとします。

関連トピック

- Oracle Data Pump Import対話方式コマンド・モードで使用可能なコマンド

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.5 CLUSTER

Oracle Data Pump Importのコマンドライン・モードのCLUSTERパラメータでは、データ・ポンプでOracle Real Application Clusters (Oracle RAC)リソースを使用できるかどうかと、ワーカーを他のOracle RACインスタンス上で開始できるかどうかを指定します。

デフォルト

YES

用途

Oracle Data PumpでOracle Real Application Clusters (Oracle RAC)リソースを使用できるかどうかと、ワーカーを他のOracle RACインスタンス上で開始できるかどうかを指定します。

構文および説明

```
CLUSTER=[YES | NO]
```

データ・ポンプ・インポートで、ジョブが開始されたインスタンスのみを使用し、Oracle Database 11g リリース2 (11.2)より前の動作と同じにするには、CLUSTER=NOを指定します。

特定の既存サービスを指定し、そのサービスに定義されたインスタンスでのみ実行されるようにワーカー・プロセスを制限するには、SERVICE_NAMEパラメータをCLUSTER=YESパラメータとともに使用します。

CLUSTERパラメータを使用すると、インポート・ジョブをOracle RACインスタンス間で分散するための追加のオーバーヘッドが発生するため、パフォーマンスに影響する場合があります。ジョブが小さいときは、CLUSTER=NOを指定して、ジョブが開始されたインスタンス上のみでジョブを実行した方がよい場合もあります。CLUSTERパラメータを使用してジョブのパフォーマンスが最もよくなるのは、ジョブのデータ量が多い場合です。

例

```
> impdp hr DIRECTORY=dpump_dir1 SCHEMAS=hr CLUSTER=NO PARALLEL=3 NETWORK_LINK=dbs1
```

この例では、hrスキーマのスキーマ・モード・インポートを実行します。CLUSTER=NOが使用されているため、ジョブでは起動時のインスタンスのみが使用されます。最大3つまでのパラレル処理を使用できます。dbs1のNETWORK_LINK値は、データのインポート元であるソース・データベース名に置き換えます。(これはネットワーク・インポートであるため、ダンプ・ファイルは生成されません。)

この例では、NETWORK_LINKパラメータは例の一部としてのみ使用されます。CLUSTERパラメータの使用時は必須ではありません。

関連トピック

- SERVICE_NAME
- Oracle RACでのOracle Data Pumpの使用方法の理解

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.6 CONTENT

Oracle Data Pump Importのコマンドライン・モードのCONTENTパラメータを使用すると、インポート操作中にロード対象をフィルタ処理できます。

デフォルト

ALL

用途

インポート操作でロードする内容をフィルタ処理できます。

構文および説明

```
CONTENT=[ALL | DATA_ONLY | METADATA_ONLY]
```

- ALL:を指定すると、ソースに含まれているすべてのデータおよびメタデータがロードされます。これはデフォルトです。
- DATA_ONLY: 表の行データのみが既存の表にロードされます。データベース・オブジェクトは作成されません。
- METADATA_ONLY: データベース・オブジェクト定義のみがロードされます。表の行データはロードされません。CONTENT=METADATA_ONLYを指定した場合、ダンプ・ファイルからインポートされる任意の索引または表の統計は、インポート操作の完了後にロックされることに注意してください。

制限事項

- CONTENT=METADATA_ONLYパラメータおよび値は、TRANSPORT_TABLESPACES(トランスポータブル表領域モード)パラメータまたはQUERYパラメータと組み合せて使用することはできません。
- CONTENT=ALLおよびCONTENT=DATA_ONLYパラメータおよび値は、SQLFILEパラメータと組み合せて使用することはできません。

例

次に、CONTENTパラメータの使用例を示します。この例では、ExportのFULLパラメータで示した例を実行して、expfull.dmpダンプ・ファイルを作成できます。

```
> impdp hr DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp CONTENT=METADATA_ONLY
```

このコマンドにより、expfull.dmpダンプ・ファイル内のメタデータのみをロードする全体インポートが実行されます。全体インポートが実行されるのは、インポート・モードを指定しないファイル・ベースのインポートでは、全体インポートがデフォルトであるため

です。

関連トピック

- FULL

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.7 CREDENTIAL

Oracle Data Pumpインポートのコマンドライン・モードのCREDENTIALパラメータでは、データベース・ユーザーが所有する資格証明オブジェクトの名前を指定します。インポート・ユーティリティは、このオブジェクト名を使用してクラウド・ストレージにインポートされるダンプ・ファイルセットのファイルを処理します。

デフォルト

none

用途

Oracle Cloud Infrastructureのクラウド・ストレージにインポートされたダンプ・ファイル・セット内のファイルを処理するためにインポート・ユーティリティによって使用される、データベース・ユーザーが所有する資格証明オブジェクト名を指定します。

構文および説明

```
CREDENTIAL=credential_object_name
```

インポート操作では、クラウドに格納されているダンプ・ファイル・セット内のファイルを、ローカル・ファイル・システムに格納されているファイルと同じように読み取り、処理します。

CREDENTIALパラメータを指定した場合は、DUMPFILEパラメータの値をカンマ区切りの文字列のリストにします。インポート・ユーティリティでは、このリストがURIの値として処理されます。Oracle Database 19c以降では、ダンプ・ファイル・セット内のURIファイルにデータ・ポンプ置換変数(%Uや%Lなど)が含まれているテンプレートを含めることができます。たとえば、urlpathexp%U.dmpです。

> ノート:
>
> 置換変数は、ファイル名の URI 部分にのみ使用できます。

DUMPFILEパラメータを使用すると、ユーザーはオプションのディレクトリ・オブジェクトを指定できます。このオブジェクトの指定には、directory_object _name:file_nameの形式を使用します。ただし、CREDENTIALパラメータを指定すると、インポート・ユーティリティはDUMPFILEで渡される文字列内のディレクトリ・オブジェクト名を検索しなくなります。そのかわりに、文字列はURI文字列として処理されます。

DIRECTORYパラメータは、ログ・ファイルとSQLファイルの場所として引き続き使用されます。また、LOGFILEとSQLFILEのファイル名の一部として引き続きディレクトリ・オブジェクト名を指定することもできます。

Oracleデータ・ポンプ・インポートでは、Oracle Autonomous Databaseのdefault_credential値の使用に対する制約がなくなりました。インポート・ユーティリティのCREDENTIALパラメータには、Oracle Autonomous Databaseで作成されたOracle Cloud Infrastructure (OCI)オブジェクト・ストレージの資格証明を指定できるようになりました。この資格証明は、DBMS_CLOUD.CREATE_CREDENTIAL()プロシージャを使用してデータベースに追加します。Oracleデータ・ポンプは、資格証明が存在しているかどうかと、ユーザーに資格証明の読取りアクセス権が付与されているかどうかを検証します。エラーはimpdpクライアントに返されます。

制限事項

資格証明パラメータは、OCIリソース・プリンシパル、Azureサービス・プリンシパル、Amazonリソース名(ARN)またはGoogleサービス・アカウントにはできません。

例: インポート・ユーティリティのCREDENTIALパラメータの使用

次に、インポート・ユーティリティのCREDENTIALパラメータの使用例を示します。この例で使用されているダンプ・ファイルを作成するには、エクスポート・ユーティリティのDUMPFILEパラメータに示した例を実行します。そのダンプ・ファイルをクラウド・ストレージにアップロードしておきます。

```
> impdp hr/your_password DIRECTORY=dpump_dir1 CREDENTIAL=user_accessible_credential
      DUMPFILE='https://objectstorage.example.com/exp1.dmp',
  'https://objectstorage.example.com/exp201.dmp',
     'https://objectstorage.example.com/exp202.dmp'
```

インポート・ジョブでは、そのダンプ・ファイルに指定されたクラウド・ストレージが検索されます。ログ・ファイルは、DIRECTORYパラメータで指定されたディレクトリ・オブジェクトdpump_dir1,に関連付けられたパスに書き込まれます。

例: ユーザー定義資格証明の指定

次の例では、Oracle Autonomous Database内に新しいユーザー定義資格証明を作成して、それと同じ資格証明をimpdpコマンドで使用しています。

```
   BEGIN
        DBMS_CLOUD.CREATE_CREDENTIAL(
        credential_name => 'MY_CRED_NAME',
        username => 'adwc_user@example.com',
        password => 'Auth token' ); END;
   > impdp admin/password@ADWC1_high
        directory=data_pump_dir
        credential=MY_cred_name …
```

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.8 DATA_OPTIONS

Oracle Data Pump Importのコマンドライン・モードのDATA_OPTIONSパラメータでは、インポート操作中に特定のタイプのデータをどのように処理するかを指定します。

デフォルト

デフォルトはありませんこのパラメータが使用されていない場合、このパラメータが提供する特別なデータ処理オプションは無効になります。

用途

DATA_OPTIONSパラメータでは、インポート操作中に特定のタイプのデータをどのように処理するかを指定します。

構文および説明

```
DATA_OPTIONS = [DISABLE_APPEND_HINT | SKIP_CONSTRAINT_ERRORS |
REJECT_ROWS_WITH_REPL_CHAR | GROUP_PARTITION_TABLE_DATA |
TRUST_EXISTING_TABLE_PARTITIONS |
VALIDATE_TABLE_DATA | ENABLE_NETWORK_COMPRESSION |
CONTINUE_LOAD_ON_FORMAT_ERROR]
```

- DISABLE_APPEND_HINT: インポート操作で、データ・オブジェクトのロード時にAPPENDヒントを使用しないことを指定します。APPENDヒントを無効化すると、重複データへの対処に役立ちます。たとえば、ロード対象のデータ・オブ

ジェクトの小規模なセットがデータベースにすでに存在し、他の一部のアプリケーションから1つ以上のデータ・オブジェクトに同時にアクセス可能な場合は、DISABLE_APPEND_HINTを使用できます。

DISABLE_APPEND_HINT: デフォルトの動作を変更して、データ・オブジェクトのロードにAPPENDヒントが使用されないようにします。設定しない場合、デフォルトでは、データ・オブジェクトのロードにAPPENDヒントが使用されます。

- GROUP_PARTITION_TABLE_DATA: 表のすべてのパーティションに1回の操作として表データをインポートするようにOracleデータポンプに指示します。デフォルトの動作では、各表パーティションが個別の操作としてインポートされます。インポート・ユーティリティでは、このデフォルトが選択されます。たとえば、このパーティションが設定されているときに、インポートの一環として表をロードする際に表が別のパーティションに移動される可能性があると、Oracleデータ・ポンプは表データを1つのパーティションにグループ化します。さらに、Oracleデータ・ポンプはインポート操作で作成された表に対する1つの操作として表のすべてのパーティションをグループ化します。

- REJECT_ROWS_WITH_REPL_CHAR: 文字セットの変換中にデフォルトの置換文字が使用されていたためにデータ損失が発生したすべての行をインポート操作で拒否するように指定します。

  REJECT_ROWS_WITH_REPL_CHARが設定されていない場合、デフォルトの動作では、置換文字で変換された行がロードされます。

- SKIP_CONSTRAINT_ERRORS: データ・オブジェクト(表、パーティションまたはサブパーティション)をロードする際の、非遅延制約違反の処理方法に適用されます。

  遅延制約違反が発生しても、SKIP_CONSTRAINT_ERRORSによるロードへの影響はありません。遅延制約違反は常に、ロード全体のロール・バックの原因となります。

  SKIP_CONSTRAINT_ERRORSオプションでは、非遅延の制約違反が発生した場合もインポート操作を続行することを指定します。非遅延の制約違反の原因となっているすべての行はログに記録されますが、違反が発生しているデータ・オブジェクトのロードは停止されません。

  SKIP_CONSTRAINT_ERRORS: 非遅延の制約違反が発生した場合に、データ・オブジェクト全体のロールバックを防ぎます。

  SKIP_CONSTRAINT_ERRORSが設定されていない場合のデフォルトの動作では、非遅延の制約違反が発生しているデータ・オブジェクトのロード全体がロール・バックされます。

- TRUST_EXISTING_TABLE_PARTITIONS: パーティション・データをパラレルで既存の表にロードするようにOracle Data Pumpに指示します。

  Oracle Data Pumpを使用してエクスポート・データベース内の定義から表を作成した後で表データ・インポートを開始する場合は、このオプションを使用します。通常、このパラメータは、メタデータが静的で、データを移行するためにデータベースをオフラインにする前に移動可能な場合に、移行の一環として使用します。メタデータを別々に移動すると、停止時間が最小限になります。このオプションを使用した場合、データベースの他の属性(文字セットなど)が同じであれば、エクスポート・データベースからのデータはインポート・データベース内の同じパーティションに移動します。

  表はデータ・ポンプの外部で作成できます。ただし、Oracle Data Pumpを使用する以外の方法で表を作成する場合は、パーティション属性およびパーティション名をエクスポート・データベースと同じにする必要があります。

- VALIDATE_TABLE_DATA: 表のデータ列内の数値および日付データ型を検査するようにOracle Data Pumpに指示します。

  インポートで無効なデータが検出された場合、ORA-39376エラーが.logファイルに書き込まれます。エラー・テキストには列名が含まれます。デフォルトでは妥当性チェックは行われません。Oracle Data Pumpのダンプ・ファイルのソースを信頼できない場合は、このオプションを使用します。

- ENABLE_NETWORK_COMPRESSION: リモート表データをロードするためのOracle Data PumpのACCESS_METHODパラメータがDIRECT_PATHに設定されている場合に、ネットワーク・インポートで使用されます。

  ENABLE_NETWORK_COMPRESSIONを指定すると、Oracle Data Pumpによってリモート・ノード上のデータが圧縮され、ネットワーク経由でターゲット・データベースに送信された後、そこで解凍されます。リモート・データベースとローカル・データベースの間のネットワーク接続の速度が低い場合は、ネットワークを経由して送信されるデータの量が減少するため、このオプションが役立ちます。

  ACCESS_METHOD=AUTOMATICに設定すると、Oracle Data Pumpで、ネットワーク・インポートにDIRECT_PATHが使用されている場合に、インポート中にENABLE_NETWORK_COMPRESSIONを自動的に設定できます。

  データ・ポンプがダンプ・ファイルからデータをインポートする場合、リモート・データベースがOracle Database 12c リリース2 (12.2)より前の場合、またはINSERT_AS_SELECT文を使用してリモート・データベースからデータをロードする場合、ENABLE_NETWORK_COMPRESSIONオプションは無視されます。

- CONTINUE_LOAD_ON_FORMAT_ERROR: 表のデータをロード中にストリーム書式エラーが発生した場合に次のグラニュルの開始点にスキップするように、Oracle Data Pumpに指示します。

  通常、ストリーム書式エラーは、ダンプ・ファイルの破損によって発生します。Oracle Data Pumpでストリーム書式エラーが発生したときに、元のエクスポート・データベースを使用して表データを再度エクスポートできない場合は、CONTINUE_LOAD_ON_FORMAT_ERRORを使用します。Oracle Data Pumpでデータがスキップされた場合、ソース・データベースからデータの一部がインポートされず、数百または数千の行がスキップされる可能性があります。

制限事項

- DISABLE_APPEND_HINTを使用すると、データ・オブジェクトのロードにかかる時間が長くなる場合があります。
- SKIP_CONSTRAINT_ERRORSを使用し、データ・オブジェクトにロード時に定義された一意の索引または制約が含まれる場合、そのデータ・オブジェクトをロードする際にAPPENDヒントは使用されません。したがって、SKIP_CONSTRAINT_ERRORSオプションを使用した場合、このようなデータ・オブジェクトのロードにはさらに時間がかかる可能性があります。
- データ・オブジェクトが外部表によるアクセス方法を使用せずにロードされている場合、SKIP_CONSTRAINT_ERRORSは、たとえ指定されていても使用されません。

例

この例では、SKIP_CONSTRAINT_ERRORSが有効化されているデータのみの表モード・インポートを示します。

```
> impdp hr TABLES=employees CONTENT=DATA_ONLY
DUMPFILE=dpump_dir1:table.dmp DATA_OPTIONS=skip_constraint_errors
```

このインポート操作中に非遅延の制約違反が発生した場合、これらの違反はログに記録されます。インポートは完了するまで続行します。

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.9 DIRECTORY

Oracle Data Pump Importのコマンドライン・モードのDIRECTORYパラメータでは、インポート・ジョブでのダンプ・ファイル・セットの検索先およびログ・ファイルとSQLファイルの作成先となるデフォルトの場所を指定します。

デフォルト

DATA_PUMP_DIR

用途

インポート・ジョブがダンプ・ファイル・セットを検出し、ログ・ファイルおよびSQLファイルが作成されるデフォルトの位置を指定します。

構文および説明

DIRECTORY=directory_object

directory_objectは、データベース・ディレクトリ・オブジェクトの名前です。実際のディレクトリのファイル・パスではありません。特権ユーザーにDATA_PUMP_DIRという名前のデフォルトのディレクトリ・オブジェクトへのアクセス権が付与されます。DATA_PUMP_DIRディレクトリの定義は、アップグレード時またはパッチの適用時にOracleによって変更される可能性があります。

デフォルトのDATA_PUMP_DIRディレクトリ・オブジェクトにアクセス可能なユーザーは、DIRECTORYパラメータを使用する必要はありません。

DUMPFILEパラメータ、LOGFILEパラメータまたはSQLFILEパラメータで指定したディレクトリ・オブジェクトは、DIRECTORYパラメータに指定したディレクトリ・オブジェクトよりも優先されます。ダンプ・ファイル・セットに使用されるディレクトリへの読取りアクセス権が必要です。ログ・ファイルおよびSQLファイルの作成に使用するディレクトリへの書込みアクセス権が必要です。

例

次に、DIRECTORYパラメータの使用例を示します。この例では、ExportのFULLパラメータで示した例を実行して、expfull.dmpダンプ・ファイルを作成できます。

```
> impdp hr DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp
LOGFILE=dpump_dir2:expfull.log
```

このコマンドによって、インポート・ジョブが、dpump_dir1ディレクトリ・オブジェクトに示されたディレクトリのexpfull.dmpダンプ・ファイルを検索します。LOGFILEパラメータに指定したdpump_dir2ディレクトリ・オブジェクトは、DIRECTORYパラメータよりも優先されるため、ログ・ファイルは、dpump_dir2に書き込まれます。CREATE DIRECTORYコマンドの詳細は、*Oracle Database SQL言語リファレンス*を参照してください。

関連トピック

- ダンプ・ファイル、ログ・ファイルおよびSQLファイルのデフォルトの位置の理解
- Oracle RACでのOracle Data Pumpの使用方法の理解
- 『*Oracle Database SQL言語リファレンス*』のCREATE DIRECTORYに関する項

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.10 DUMPFILE

Oracle Data Pump Importのコマンドライン・モードのDUMPFILEパラメータでは、エクスポート・ユーティリティによって作成されたダンプ・ファイル・セットの名前を指定し、オプションで、これらのディレクトリ・オブジェクトを指定します。

デフォルト

expdat.dmp

用途

エクスポート・ユーティリティによって作成されたダンプ・ファイル・セットの名前を指定します。そのセットのディレクトリ・オブジェクトまたはデフォルトの資格証明も指定できます。

構文および説明

```
DUMPFILE=[directory_object:]file_name [, ...]
```

または

```
DUMPFILE=[DEFAULT_CREDENTIAL:]URI_file [, ...]
```

DIRECTORYパラメータですでに指定されている場合、directory_objectはオプションです。値を指定する場合は、すでに存在し、かつ自分にアクセス権があるディレクトリ・オブジェクトにする必要があります。DUMPFILEパラメータの一部に指定されるデータベース・ディレクトリ・オブジェクトは、DIRECTORYパラメータで指定された値よりも優先されます。

file_nameには、ダンプ・ファイル・セット内のファイルの名前を指定します。ファイル名には、置換変数%Uを含むテンプレートを指定することもできます。インポート・プロセスでは、各ファイルがテンプレートと一致するかどうかがチェックされます。ダンプ・ファイル・セットに含まれるすべてのファイルは、一致するファイルがなくなるまで検索されます。DUMPFILEパラメータで定義されたファイルの指定にセット全体が含まれている場合は、インポート・ユーティリティでセット全体を見つけるために十分な情報がファイルに含まれます。ファイルの名前、位置または順序は、エクスポート時に使用されたものと同じである必要はありません。

次の表に、使用可能な置換変数を示します。

置換変数	説明
%U	%U を使用すると、%U が 01 から増加する 2 桁の整数に変換されます。
%l、%L	システムによって生成される一意のファイル名を指定します。 ファイル名には、複数のファイルを生成できることを示す置換変数(%L)を含めることができます。生成されるファイル名では、置換変数が、01 から 99 まで増加する固定幅の 2 桁の整数(%U と同じ)に変換されます。また、生成されるファイル名では、置換変数が、100 から 2147483646 まで増加する 3 桁から 10 桁までの可変幅の整数に変換されます。幅フィールドは、整数の桁数によって決まります。 たとえば、現在の整数が 1 の場合、exp%Laa%L.dmp は次の順序に解決されます exp01aa01.dmp exp02aa02.dmp 2 桁の増加が 99 まで続きます。その後、次のファイル名は 3 桁の増加に置換されます。 exp100aa100.dmp exp101aa101.dmp 3 桁の増加が 999 まで続きます。その後のファイル名は 4 桁の増加に置換されます。最大許容置換数(2147483646)まで、置換が続きます。

制限事項

- Oracle Data PumpのパラメータVERSION=12を使用してOracle Database 11gリリースで作成したダンプ・ファイルは、Oracle Database 12cリリース1 (12.1)以降でのみインポートできます。

インポート・ユーティリティのDUMPFILEパラメータの使用例

この例では、ExportのDUMPFILEパラメータで示した例を実行して、ダンプ・ファイルを作成できます。

```
> impdp hr DIRECTORY=dpump_dir1 DUMPFILE=dpump_dir2:exp1.dmp, exp2%U.dmp
```

exp1.dmpダンプ・ファイルにディレクトリ・オブジェクト(dpump_dir2)が指定されているため、インポート・ジョブは、そのファイルを検索します。また、dpump_dir1にある、exp2nn.dmpの形式のダンプ・ファイルも検索します。ログ・ファイルはdpump_dir1に書き込まれます。

インポート・ユーティリティのDUMPFILEパラメータに代替のDEFAULT_CREDENTIALキーワード構文を使用する場合は、ユーザー・アクセス権を持つデフォルトの資格証明が事前に存在している必要があります。インポート操作では、デフォルトの資格証明を使用して、指定したURI_fileの場所のクラウドに格納されたダンプ・ファイル・セット内のファイルを読み取って処理します。

変数URI_fileは、ダンプ・ファイルセット内のURIファイルの名前を表します。このファイル名は、データ・ポンプ置換変数(%Uや%Lなど)が含まれているテンプレートと同じ名前にはできません。

DUMPFILEパラメータのDEFAULT_CREDENTIALキーワード構文は、directory_object構文と相互に排他的です。同じコマンドラインで使用できるフォームは1つのみです。

ユーザー定義資格証明を指定したインポート・ユーティリティのDUMPFILEの使用例

この例では、インポート・ジョブがダンプ・ファイル・セットを検索してログ・ファイルとSQLファイルを作成できるデフォルトの場所を指定しています。また、以前にクラウド・ストレージにインポートしたダンプ・ファイル・セット内のファイルを処理するために、インポート・ユーティリティで使用するデータベース・ユーザーが所有する資格証明オブジェクトの名前も指定しています。

```
> impdp admin/password@ADWC1_high
      directory=data_pump_dir
      credential=MY_cred_name …
```

DEFAULT_CREDENTIALキーワードを指定したインポート・ユーティリティのDUMPFILEパラメータの使用例。

この例では、ExportのDUMPFILEパラメータで示した例を実行して、ダンプ・ファイルを作成できます。

```
> impdp hr/your_password DIRECTORY=dpump_dir1
      DUMPFILE='DEFAULT_CREDENTIAL:https://objectstorage.example.com/exp1.dmp',
  'DEFAULT_CREDENTIAL:https://objectstorage.example.com/exp201.dmp',
     'DEFAULT_CREDENTIAL:https://objectstorage.example.com/exp202.dmp'
```

インポート・ジョブでは、ユーザーにすでに設定されているデフォルトの資格証明を使用して、指定されたURI_fileの場所でダンプ・ファイルが検索されます。ログ・ファイルは、DIRECTORYパラメータで指定されたディレクトリ・オブジェクトdpump_dir1に関連付けられたパスに書き込まれます。

ユーザー定義資格証明を指定したインポート・ユーティリティのDUMPFILEパラメータの使用例

この例では、インポート・ジョブがダンプ・ファイル・セットを検索してログ・ファイルとSQLファイルを作成できるデフォルトの場所を指定しています。また、以前にクラウド・ストレージにインポートしたダンプ・ファイル・セット内のファイルを処理するために、インポート・ユーティリティで使用するデータベース・ユーザーが所有する資格証明オブジェクトの名前も指定しています。

```
> impdp impdp admin/password@ADWC1_high  DIRECTORY=data_pump_dir
      DUMPFILE='MY_cred_name:https://objectstorage.example.com/exp1.dmp',
  'MY_cred_name:https://objectstorage.example.com/exp201.dmp',
     'MY_cred_name:https://objectstorage.example.com/exp202.dmp'
```

関連トピック

- DUMPFILE
- Oracleデータ・ポンプによるファイルの割当て
- データのみ表モード・インポートの実行

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.11 ENABLE_SECURE_ROLES

Oracle Data Pumpインポートのコマンドライン・ユーティリティのENABLE_SECURE_ROLESパラメータでは、エクスポート時の保護されているロールの誤用を防止します。

デフォルト

Oracle Database 19c以降のリリースでは、デフォルト値はNOです。

用途

Oracleロールには、認証が必要なものがあります。そのようなロールをOracleデータ・ポンプ・インポートで使用する必要がある場合は、ENABLE_SECURE_ROLESパラメータをYESに設定することで明示的に使用可能にする必要があります。

構文

ENABLE_SECURE_ROLES=[NO|YES]

- NO: 認証が必要なOracleロールを使用できないようにします。
- YES: 認証が必要なOracleロールを使用可能にします。

例

```
impdp hr SCHEMAS=hr DIRECTORY=dpump_dir1 DUMPFILE=dpump_dir2:imp1.dmp,
 imp2%U.dmp ENABLE_SECURE_ROLES=YES
```

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.12 ENCRYPTION_PASSWORD

Oracle Data Pump Importのコマンドライン・モードのENCRYPTION_PASSWORDパラメータでは、ダンプ・ファイル・セット内の暗号化列のデータにアクセスするためのパスワードを指定します。これにより、暗号化されたダンプ・ファイル・セットへの不正なアクセスを防ぎます。

デフォルト

デフォルト値は設定されていません。ユーザーが値を指定します。

用途

ダンプ・ファイル・セット内の暗号化列のデータにアクセスするためのパスワードを指定します。これにより、暗号化されたダンプ・ファイル・セットへの不正なアクセスを防ぎます。

このパラメータは、全体トランスポータブル・エクスポートまたはインポート操作中に、暗号化された表領域および暗号化された列を含む表に関連するキーを転送する場合にも必要です。

入力したパスワードは画面に表示されます。パスワードを入力時に画面に表示しない場合、ENCRYPTION_PWD_PROMPTパラメータを使用します。

構文および説明

```
ENCRYPTION_PASSWORD = password
```

このパラメータは、エクスポート操作で暗号化パスワードが指定された場合に、インポート操作で必要になります。このパスワードは、エクスポート操作で指定されたものと同じものを指定する必要があります。

制限事項

- このパラメータを使用したエクスポート操作には、Oracle Database 11g以降のEnterprise Editionリリースが必要です。Standard EditionからのエクスポートにENCRYPTION_PASSWORDを使用できないため、このパラメータをStandard EditionからEnterprise Editionへの移行に使用することはできません。このパラメータは、Enterprise EditionからStandard Editionへの移行に使用できます。
- データ・ポンプの暗号化機能を使用するには、Oracle Advanced Securityオプションが有効である必要があります。Oracle Advanced Securityオプションのライセンス要件の詳細は、*Oracle Database*ライセンス情報を参照してください。
- ダンプ・ファイル・セットが暗号化の透過モードを使用して作成されている場合、ENCRYPTION_PASSWORDパラメータは無効です。
- ENCRYPTION_PASSWORDパラメータは、暗号化された表領域または暗号化された列を含む表がソース・データベースに存在するネットワーク・ベースの全体トランスポータブル・インポートでは必須です。
- すべての列に対する暗号化属性は、エクスポートされた表の定義とターゲット表で一致している必要があります。たとえば、EMP表にEMPNOという列があるとします。次のいずれの場合も、ソース表のEMP列の暗号化属性が、ターゲット表のEMP列の暗号化属性と一致していないためエラーになります。
  - EMPNO列を暗号化してEMP表をエクスポートし、表をインポートする前にEMPNO列から暗号化属性を削除する。
  - EMPNO列を暗号化せずにEMP表をエクスポートし、表をインポートする前にEMPNO列の暗号化を有効にする。

例

次の例では、暗号化パスワード123456を指定する必要があります。これは、ダンプ・ファイルdpcd2be1.dmpの作成時に、そのパスワードが指定されたためです。

```
> impdp hr TABLES=employee_s_encrypt DIRECTORY=dpump_dir
  DUMPFILE=dpcd2be1.dmp ENCRYPTION_PASSWORD=123456
```

インポート操作中、エクスポート操作時に暗号化されたemployee_s_encrypt表のすべての列は、復号化されてからインポートされます。

関連トピック

- Oracle Databaseライセンス情報ユーザー・マニュアル

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.13 ENCRYPTION_PWD_PROMPT

Oracle Data Pumpインポートのコマンドライン・モードのENCRYPTION_PWD_PROMPTパラメータでは、データ・ポンプによって暗号化パスワードの入力を要求するかどうかを指定します。

デフォルト

NO

用途

Oracle Data Pumpによって暗号化パスワードの入力を要求するかどうかを指定します。

構文および説明

```
ENCRYPTION_PWD_PROMPT=[YES | NO]
```

コマンドラインでENCRYPTION_PWD_PROMPT=YESを指定すると、暗号化パスワードの入力を求めるプロンプトがOracle Data Pumpに表示されます。値をYESに指定しない場合は、コマンドラインでENCRYPTION_PASSWORDパラメータを使用して暗号化パスワードを入力する必要があります。パラメータをYESに設定する利点は、プロンプトが表示されたときに暗号化パスワードが画面に表示されないことです。対照的に、コマンドラインでENCRYPTION_PASSWORDパラメータを使用してパスワードを入力すると、パスワードはプレーン・テキストで表示されます。

プロンプトで入力する暗号化パスワードは、ENCRYPTION_PASSWORDパラメータで説明されているものと同じ基準に従います。

エクスポート操作で暗号化パスワードを指定した場合、インポート操作でもそれを指定する必要があります。

制限事項

ENCRYPTION_PWD_PROMPTパラメータとENCRYPTION_PASSWORDパラメータの同時使用は禁止されています。

例

次の例では、Oracle Data Pumpによって最初にユーザー・パスワードが要求され、次に暗号化パスワードが要求されます。

```
> impdp hr DIRECTORY=dpump_dir1 DUMPFILE=hr.dmp ENCRYPTION_PWD_PROMPT=YES
.
.
.
Copyright (c) 1982, 2017, Oracle and/or its affiliates.  All rights reserved.
Password:

Connected to: Oracle Database 18c Enterprise Edition Release 18.0.0.0.0 - Development
Version 18.1.0.0.0

Encryption Password:
Master table "HR"."SYS_IMPORT_FULL_01" successfully loaded/unloaded
Starting "HR"."SYS_IMPORT_FULL_01":  hr/******** directory=dpump_dir1 dumpfile=hr.dmp
encryption_pwd_prompt=Y
.
.
.
```

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.14 ESTIMATE

Oracle Data Pumpインポートのコマンドライン・モードのESTIMATEパラメータでは、ネットワーク・インポート操作のソース・システムに対して、インポート中に生成されるデータ量を見積もるように指定します。

デフォルト

STATISTICS

用途

ネットワーク・インポート操作のソース・システムに対して、インポート中に生成されるデータ量を見積もるように指定します。

構文および説明

```
ESTIMATE=[BLOCKS | STATISTICS]
```

ESTIMATEパラメータでは、次の値を選択できます。

- BLOCKS: 見積りは、ソース・オブジェクトで使用されるデータベース・ブロックの数に、適切なブロック・サイズを掛けて計算されます。

- STATISTICS: 見積りは、表別の統計を使用して計算されます。この方法による見積りをできるかぎり正確にするには、すべての表を新しく分析しておく必要があります。(表の分析は、SQL ANALYZE文またはDBMS_STATS PL/SQLパッケージのいずれかで実行可能です。)

生成された見積りを使用して、インポート全体で完了したインポート・ジョブの割合を判断できます。

制限事項

- インポートのESTIMATEパラメータは、NETWORK_LINKパラメータも指定されている場合のみ有効です。
- インポート・ソースがダンプ・ファイル・セットの場合、ロードされるデータの量がすでにわかっているため、完了率は自動的に計算されます。
- QUERYまたはREMAP_DATAパラメータのいずれかを使用している場合は、見積りが不正確になることがあります。

例

次の構文例では、変数source_database_linkをソース・データベースへの有効なリンク名で置き換えます。

```
> impdp hr TABLES=job_history NETWORK_LINK=source_database_link
  DIRECTORY=dpump_dir1 ESTIMATE=STATISTICS
```

hrスキーマのjob_history表が、ソース・データベースからインポートされます。デフォルトでログ・ファイルが作成され、dpump_dir1ディレクトリ・オブジェクトで示されたディレクトリに書き込まれます。ジョブが開始すると、表の統計に基づいて、そのジョブの見積りが計算されます。

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.15 EXCLUDE

Oracle Data Pump Importのコマンドライン・モードのEXCLUDEパラメータを使用すると、インポート・ジョブから除外するオブジェクトとオブジェクト型を指定して、インポート対象のメタデータをフィルタ処理できます。

デフォルト

デフォルトはありません

用途

インポート・ジョブから除外するオブジェクトおよびオブジェクト型を指定して、インポートの対象となるメタデータをフィルタ処理できます。

構文および説明

```
EXCLUDE=object_type[:name_clause] [, ...]
```

object_typeは除外するオブジェクトのタイプを指定します。DATABASE_EXPORT_OBJECTS(全体モードの場合)、SCHEMA_EXPORT_OBJECTS(スキーマ・モードの場合)、TABLE_EXPORT_OBJECTS(表および表領域モードの場合)ビューを問い合せて、object_typeの有効な値の一覧を表示できます。OBJECT_PATH列にリストされる値が有効なオブジェクト・タイプです。

指定したインポート・モードでは、EXCLUDE文に指定されたオブジェクト型を除き、ソースに含まれるすべてのオブジェクト型およびその依存オブジェクトが含まれます。オブジェクトが除外されると、そのオブジェクトのすべての依存オブジェクトも除外されます。たとえば、表を除外すると、その表のすべての索引およびトリガーも除外されます。

name_clauseは、オプションです。これを使用すると、あるオブジェクト型のうち、特定のオブジェクトをファイングレイン選択できます。その型のオブジェクト名に対するフィルタとして使用されるSQL式です。SQL演算子および指定した型のオブジェクト名の

比較対象となる値で構成されています。name_clauseは、名前付きのインスタンスを持つオブジェクト型にのみ適用されます(たとえば、TABLEおよびVIEWには適用されますが、GRANTには適用されません)。それは、コロンでオブジェクト型と区切り、二重引用符(一重引用符は名前文字列を区切るために必要なため)で囲む必要があります。たとえば、EXCLUDE=INDEX:"LIKE 'DEPT%'"と設定した場合、deptで始まる名前を持つすべての索引を除外できます。

name_clauseに指定する名前は、データベース内の既存のオブジェクトに、大文字と小文字も含めて完全に一致させる必要があります。たとえば、指定するname_clauseがEMPLOYEESという表に対するものだった場合は、すべてが大文字のEMPLOYEESという表が存在する必要があります。name_clauseに対してEmployees、employeesなどを指定した場合、表は検出されません。

2つ以上のEXCLUDE文を指定できます。

オペレーティング・システムによっては、このパラメータの値を指定するときに引用符とともにエスケープ文字を使用する必要があります。このパラメータはパラメータ・ファイルで指定することをお薦めします。パラメータ・ファイルを使用すると、コマンドラインでは必要なエスケープ文字の数を減らすことができます。

次の項で説明するとおり、特定のオブジェクト(特にCONSTRAINT、GRANTおよびUSER)を除外対象として指定した場合の効果を認識しておく必要があります。

制約の除外

次の制約は除外できません。

- 表の作成とロードを正常に行うために必要な制約。たとえば、索引構成表の主キー制約、REF列を持つ表のREF SCOPEおよびWITH ROWID制約など。

次に、EXCLUDE文の例およびその解釈を示します。

- EXCLUDE=CONSTRAINTでは、表の正常な作成およびロードに必要な制約を除く、すべての制約を除外します。
- EXCLUDE=REF_CONSTRAINTは、参照整合性(外部キー)制約を除外します。

権限とユーザーの除外

EXCLUDE=GRANTを指定すると、すべてのオブジェクト型に対するオブジェクト権限およびシステム権限が除外されます。

EXCLUDE=USERを指定すると、ユーザーの定義のみが除外され、そのユーザーのスキーマ内のオブジェクトは除外されません。

特定のユーザーとそのユーザーのすべてのオブジェクトを除外するには、次のコマンドを指定します(hrは除外するユーザーのスキーマ名です)。

```
impdp FULL=YES DUMPFILE=expfull.dmp EXCLUDE=SCHEMA:"='HR'"
```

この例では、FULLインポート・モードが指定されていることに注意してください。モードが指定されていない場合は、デフォルト・モードであるSCHEMASが使用されます。ただし、この例で、FULLを指定しないで、かわりにSCHEMASを使用し、続いてEXCLUDE=SCHEMA引数を指定した場合、スキーマのインポートと除外を同時に指定することになるため、エラーが発生します。

EXCLUDE=USER:"= 'HR'"などの文を使用してユーザーを除外しようとすると、CREATE USER hr DDL文のみが除外され、予期しない結果が返される可能性があります。

制限事項

- EXCLUDEおよびINCLUDEは、相互に排他的なパラメータです。

例

DBAまたはDATAPUMP_IMP_FULL_DATABASEロールを持つ他のユーザーが、パラメータ・ファイルexclude.parで次の

ように実行するとします。(例を実行するには、まずこのファイルを作成する必要があります。)

```
EXCLUDE=FUNCTION
EXCLUDE=PROCEDURE
EXCLUDE=PACKAGE
EXCLUDE=INDEX:"LIKE 'EMP%' "
```

次に、次のコマンドを発行します。

```
> impdp system DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp PARFILE=exclude.par
```

このコマンドでは、FULLリファレンス・トピックでエクスポートのFULLパラメータについて示した例を実行し、expfull.dmpダンプ・ファイルを作成できます。expfull.dmpダンプ・ファイルから、empで始まる名前を持つファンクション、プロシージャ、パッケージおよび索引を除くすべてのデータがロードされます。

関連トピック

- FULL
- メタデータ・フィルタ
- インポート操作中のフィルタ処理
- インポート・コマンドライン・モードについて

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.16 FLASHBACK_SCN

Oracle Data Pump Importのコマンドライン・モードのFLASHBACK_SCNでは、インポート・ユーティリティで使用されるシステム変更番号(SCN)を指定して、フラッシュバック・ユーティリティを使用可能にします。

デフォルト

デフォルトはありません

用途

インポートで使用されるシステム変更番号(SCN)を指定して、フラッシュバック・ユーティリティを使用可能にします。

構文および説明

```
FLASHBACK_SCN=scn_number
```

インポート操作は、指定したscn_numberまでの一貫性のあるデータを使用して実行されます。

Oracle Database 12c リリース2 (12.2)以降では、SCN値をビッグSCN (8バイト)にすることができます。ビッグSCNの使用の詳細は、次の制限事項を参照してください。

制限事項

- FLASHBACK_SCNパラメータは、NETWORK_LINKパラメータも指定されている場合のみ有効です。
- FLASHBACK_SCNパラメータは、Oracle Databaseのフラッシュバック問合せ機能にのみ関係します。フラッシュバック・データベース、フラッシュバック削除およびフラッシュバック・データ・アーカイブには適用できません。
- FLASHBACK_SCNおよびFLASHBACK_TIMEは、相互に排他的なパラメータです。
- ビッグSCNをサポートしていないバージョンからのネットワーク・エクスポートまたはネットワーク・インポートに対して、ビッグSCNを指定することはできません。

例

次に、FLASHBACK_SCNパラメータを使用した構文例を示します。

```
> impdp hr DIRECTORY=dpump_dir1 FLASHBACK_SCN=123456
NETWORK_LINK=source_database_link
```

このコマンドを使用する場合、変数123456およびsource_database_linkを、データのインポート元となるソース・データベースのSCNと名前に置き換えます。

ノート:

ロジカル・スタンバイ・システムでは、ロジカル・スタンバイによって SCN が選択されるため、FLASHBACK_SCN パラメータは無視されます。ロジカル・スタンバイ・データベースの詳細は、『*Oracle Data Guard* 概要および管理』を参照してください。

関連トピック

- 『Oracle Data Guard概要および管理』の「ロジカル・スタンバイ・データベース」

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.17 FLASHBACK_TIME

Oracle Data Pump Importのコマンドライン・モードのFLASHBACK_TIMEパラメータでは、インポート・ユーティリティで使用されるシステム変更番号(SCN)を指定して、フラッシュバック・ユーティリティを使用可能にします。

デフォルト

デフォルトはありません

用途

インポートで使用されるシステム変更番号(SCN)を指定して、フラッシュバック・ユーティリティを使用可能にします。

構文および説明

```
FLASHBACK_TIME="TO_TIMESTAMP()"
```

指定された時刻に最も近いSCNを検出し、このSCNを使用してフラッシュバック・ユーティリティを使用可能にします。インポート操作は、このSCNまでの一貫性のあるデータを使用して実行されます。TO_TIMESTAMPの値は引用符で囲まれるため、パラメータ・ファイルに記述することをお薦めします。

ノート:

ロジカル・スタンバイ・システムでは、ロジカル・スタンバイによって SCN が選択されるため、FLASHBACK_TIME パラメータは無視されます。ロジカル・スタンバイ・データベースの詳細は、『*Oracle Data Guard* 概要および管理』を参照してください。

制限事項

- このパラメータは、NETWORK_LINKパラメータも指定されている場合にのみ有効です。

- FLASHBACK_TIMEパラメータは、Oracle Databaseのフラッシュバック問合せ機能にのみ関係します。フラッシュバック・データベース、フラッシュバック削除およびフラッシュバック・データ・アーカイブには適用できません。
- FLASHBACK_TIMEおよびFLASHBACK_SCNは、相互に排他的なパラメータです。

例

DBMS_FLASHBACK.ENABLE_AT_TIMEプロシージャで使用可能な形式で時刻を指定できます。たとえば、次の内容のパラメータ・ファイルflashback_imp.parを作成したとします。

```
FLASHBACK_TIME="TO_TIMESTAMP('27-10-2012 13:40:00', 'DD-MM-YYYY HH24:MI:SS')"
```

次のコマンドを発行します。

```
> impdp hr DIRECTORY=dpump_dir1 PARFILE=flashback_imp.par
NETWORK_LINK=source_database_link
```

インポート操作は、指定した時間に最も近いSCNと整合性のあるデータで実行されます。

ノート:

フラッシュバックの使用方法の詳細は、*Oracle Database* 開発ガイドを参照してください

関連トピック

- インポート・コマンドライン・モードについて
- 『Oracle Data Guard概要および管理』の「ロジカル・スタンバイ・データベース」
- 『Oracle Database開発ガイド』の「Oracleフラッシュバック問合せ(SELECT AS OF)の使用」

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.18 FULL

Oracle Data Pump Importのコマンドライン・モードのFULLパラメータにより、全データベース・インポートを実行することを指定します。

デフォルト

YES

用途

全データベース・インポートの実行を指定します。

構文および説明

```
FULL=YES
```

FULL=YESの値は、ソースからのすべてのデータおよびメタデータがインポートされることを示します。ソースは、ダンプ・ファイル・セットにするか(ファイル・ベース・インポートの場合)、NETWORK_LINKパラメータで指定された別のデータベースにすることができます(ネットワーク・インポートの場合)。

ファイルからインポートする際に、DATAPUMP_IMP_FULL_DATABASEロールを持っていない場合、自分のスキーマにマップするスキーマのみインポートされます。

NETWORK_LINKパラメータが使用され、インポート・ジョブを実行するユーザーがターゲット・データベースの

DATAPUMP_IMP_FULL_DATABASEロールを持っている場合、そのユーザーは、ソース・データベースのDATAPUMP_EXP_FULL_DATABASEロールも持っている必要があります。

このインポート・モードを使用したインポート対象を、フィルタ処理によって制限できます。

FULLはデフォルトのモードで、ファイル・ベースのインポートを実行している場合、コマンドラインで指定する必要はありませんが、ネットワーク・ベースの全体インポートを実行している場合は、コマンドラインでFULL=Yを指定する必要があります。

全体モード・インポート時にトランスポータブル・オプションを使用して、全体トランスポータブル・インポートを実行できます。

制限事項

- 自動ワークロード・リポジトリ(AWR)は、全データベースのエクスポートおよびインポート操作では移動されません。(データ・ポンプを使用してAWRスナップショットを移動する方法の詳細は、『*Oracle Database*パフォーマンス・チューニング・ガイド』を参照してください。)
- XDBリポジトリは、全データベースのエクスポートおよびインポート操作では移動されません。ユーザーが作成したXMLスキーマは移動されます。
- ターゲットがOracle Database12cリリース1 (12.1.0.1)以降で、ソースがOracle Database11gリリース2 (11.2.0.3)以降である場合、ネットワーク・リンクで実行される全体インポートでは、VERSION=12を設定する必要があります

例

次に、FULLパラメータの使用例を示します。この例では、ExportのFULLパラメータで示した例を実行して、expfull.dmpダンプ・ファイルを作成できます。

```
> impdp hr DUMPFILE=dpump_dir1:expfull.dmp FULL=YES
LOGFILE=dpump_dir2:full_imp.log
```

この例では、expfull.dmpダンプ・ファイルのすべての内容をインポートします。ここでは、DIRECTORYパラメータは指定されていません。そのため、DUMPFILEパラメータおよびLOGFILEパラメータの両方にディレクトリ・オブジェクトを指定する必要があります。例に示すとおり、ディレクトリ・オブジェクトは、別のものを指定することができます。

関連トピック

- 『Oracle Databaseパフォーマンス・チューニング・ガイド』の「自動ワークロード・リポジトリ・データの転送」
- 『Oracle Database管理者ガイド』の「データベースの転送」
- FULL

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.19 HELP

Oracle Data Pump Importのコマンドライン・モードのHELPパラメータにより、インポート・ユーティリティのオンライン・ヘルプを表示します。

デフォルト

NO

用途

インポート・ユーティリティのオンライン・ヘルプを表示します。

構文および説明

```
HELP=YES
```

HELP=YESが指定されている場合は、インポート・ユーティリティのすべてのコマンドライン・パラメータと対話方式コマンドの要約が表示されます。

例

この例では、すべてのインポート・パラメータおよびコマンドの簡単な説明が表示されます。

```
> impdp HELP = YES
```

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.20 INCLUDE

Oracle Data Pump Importのコマンドライン・モードのINCLUDEパラメータを使用すると、現行のインポート・モードにオブジェクトおよびオブジェクト型を指定して、インポート対象のメタデータをフィルタ処理できます。

デフォルト

デフォルトはありません

用途

現行のインポート・モードにオブジェクトとオブジェクト型を指定して、インポート対象のメタデータをフィルタ処理できます。

構文および説明

```
INCLUDE = object_type[:name_clause] [, ...]
```

構文の変数object_typeでは、含めるオブジェクトの型を指定します。object_typeの有効な値のリストを表示するには、次のビューを問い合せます。

- 全体モード: DATABASE_EXPORT_OBJECTS
- スキーマ・モード: SCHEMA_EXPORT_OBJECTS
- 表および表領域モード: TABLE_EXPORT_OBJECTS

問合せ結果で、OBJECT_PATH列にリストされる値が有効なオブジェクト型です。(このような問合せの実行方法の例は、「メタデータのフィルタ」を参照してください。)

INCLUDE文に明示的に指定した、ソース内のオブジェクト型とその依存オブジェクトのみがインポートされます。

構文の変数name_clauseはオプションです。これを使用すると、あるオブジェクト型のうち、特定のオブジェクトのファイングレイン選択を実行できます。その型のオブジェクト名に対するフィルタとして使用されるSQL式です。SQL演算子および指定した型のオブジェクト名の比較対象となる値で構成されています。このname_clauseは、名前付きのインスタンスを持つオブジェクト型にのみ適用されます(たとえば、TABLEには適用されますが、GRANTには適用されません)。それは、コロンでオブジェクト型と区切り、二重引用符で囲む必要があります。一重引用符は名前文字列を区切るために必要となるため、二重引用符を使用する必要があります。

name_clauseに指定する名前文字列は、データベース内の既存のオブジェクトに、大文字と小文字も含めて完全に一致する必要があります。たとえば、指定するname_clauseがEMPLOYEESという表に対するものだった場合は、すべてが大文字のEMPLOYEESという表が存在する必要があります。name_clauseにEmployeesまたはemployeesを指定したり、既存の表の名前文字列からのその他のバリエーションを使用する場合、表は検出されません。

複数のINCLUDE文を指定できます。

オペレーティング・システムによっては、このパラメータに引用符を使用する値を指定する場合、エスケープ文字を使用することも必要になる可能性があります。このパラメータはパラメータ・ファイルで指定することをお薦めします。パラメータ・ファイルを使用すると、コマンドラインでは使用する必要があるエスケープ文字の数を減らすことができます。

INCLUDEパラメータで使用する有効なパスのリストを表示するには、次のビューを問い合せます。

- 全体モード: DATABASE_EXPORT_OBJECTS
- スキーマ・モード: SCHEMA_EXPORT_OBJECTS
- 表および表領域モード: TABLE_EXPORT_OBJECTS

制限事項

- INCLUDEおよびEXCLUDEは、相互に排他的なパラメータです。

例

imp_include.parという名前のパラメータ・ファイルに、次が含まれているとします。このパラメータ・ファイルは、DBAまたはロールDATAPUMP_IMP_FULL_DATABASEが付与されているその他のユーザーによって使用されています。

```
INCLUDE=FUNCTION
INCLUDE=PROCEDURE
INCLUDE=PACKAGE
INCLUDE=INDEX:"LIKE 'EMP%' "
```

このパラメータ・ファイルを使用すると、次のコマンドを発行できます。

```
> impdp system SCHEMAS=hr DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp
PARFILE=imp_include.par
```

この例では、ExportのFULLパラメータで示した例を実行して、expfull.dmpダンプ・ファイルを作成できます。

このインポートでは、hrスキーマのファンクション、プロシージャ、パッケージ、および名前がEMPで始まる索引のみロードされます。これは特権モードのインポート(ユーザーにDATAPUMP_IMP_FULL_DATABASEロールがある)ですが、USERオブジェクト型がINCLUDE文に指定されていないため、スキーマ定義はインポートされません。

関連トピック

- メタデータ・フィルタ
- インポート・コマンドライン・モードについて
- FULL

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.21 JOB_NAME

Oracle Data Pumpインポートのコマンドライン・モードのJOB_NAMEパラメータを使用して、後続処理でインポート・ジョブを指定します。

デフォルト

SYS_IMPORTまたはSQLFILE_mode_NNという書式のシステム生成による名前

用途

後続処理でインポート・ジョブを指定する場合は、JOB_NAMEパラメータを使用します。たとえば、ATTACHパラメータを使用してジョブに接続する場合、JOB_NAMEパラメータを使用して、接続するジョブを指定します。ビューDBA_DATAPUMP_JOBSま

たはUSER_DATAPUMP_JOBSを使用することにより、JOB_NAMEを使用してジョブを指定することもできます。

構文および説明

```
JOB_NAME=jobname_string
```

変数jobname_stringでは、インポート・ジョブに128バイト以内の名前を指定します。これらのバイトは印字可能文字と空白を表します。文字列に空白が含まれる場合は、名前を一重引用符で囲む必要があります('Thursday Import'など)。ジョブ名は、インポート操作を実行しているユーザーのスキーマによって暗黙的に修飾されます。ジョブ名は、エクスポート・ジョブを制御するデータ・ポンプ制御インポート・ジョブ表の名前として使用されます。

デフォルトのジョブ名はSYS_IMPORT_mode_NNまたはSYS_SQLFILE_mode_NNという形式で、システムによって生成されます。NNは、01から開始して増加する2桁の整数です。たとえば、SYS_IMPORT_TABLESPACE_02'はデフォルトのジョブ名です。

例

次に、JOB_NAMEパラメータの使用例を示します。この例では、エクスポートのFULLパラメータで示した例を実行して、expfull.dmpダンプ・ファイルを作成できます。

```
> impdp hr DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp JOB_NAME=impjob01
```

関連トピック

- FULL

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.22 KEEP_MASTER

Oracle Data Pumpインポートのコマンドライン・モードのKEEP_MASTERパラメータは、正常に完了したOracle Data Pumpジョブの最後に、データ・ポンプ制御ジョブ表を削除するか保持するかを指定します。

デフォルト

NO

用途

正常に完了したOracle Data Pumpジョブの最後に、データ・ポンプ制御ジョブ表を削除するか保持するかを示します。データ・ポンプ制御ジョブ表は、正常に完了しなかったジョブに対して自動的に保持されます。

構文および説明

```
KEEP_MASTER=[YES | NO]
```

制限事項

- なし

例

```
> impdp hr SCHEMAS=hr DIRECTORY=dpump_dir1 LOGFILE=schemas.log
DUMPFILE=expdat.dmp KEEP_MASTER=YES
```

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.23 LOGFILE

Oracle Data Pump Importのコマンドライン・モードのLOGFILEパラメータでは、インポート・ジョブのログ・ファイルの名前を指定し、オプションでディレクトリ・オブジェクトを指定します。

デフォルト

import.log

用途

インポート・ジョブのログ・ファイルの名前を指定します。オプションで、そのログ・ファイルのディレクトリ・オブジェクトを指定します。

構文および説明

```
LOGFILE=[directory_object:]file_name
```

directory_objectを指定する場合、これはDBAによって作成済で、自分にアクセス権があるものにする必要があります。このパラメータは、DIRECTORYパラメータに指定されたディレクトリ・オブジェクトよりも優先されます。デフォルトでは、DIRECTORYパラメータに指定されているディレクトリ・オブジェクトによって参照されるディレクトリ内に、import.logが作成されます。

指定したfile_nameがすでに存在する場合は、上書きされます。

処理中の作業、完了した作業および発生したエラーに関するすべてのメッセージがログ・ファイルに書き込まれます。(ジョブのリアルタイムの状態を把握するには、対話方式モードでSTATUSコマンドを使用します。)

NOLOGFILEパラメータを指定しないかぎり、ログ・ファイルは常に作成されます。ダンプ・ファイル・セットと同様に、ログ・ファイルの基準となるのは、クライアントではなく、サーバーです。

> ノート:
>
> Oracle Data Pump Import は、データベース文字セットを使用してログ・ファイルに書き込みます。クライアントの NLS_LANG 環境にデータベースの文字セットと異なる文字セットを設定した場合は、ログ・ファイル内の表の名前が、クライアントの出力画面に表示される名前と異なる可能性があります。

制限事項

- Oracle Automatic Storage Management (Oracle ASM)を使用してOracle Data Pump Importを実行する場合は、LOGFILEパラメータで、Oracle ASMの+表記法を使用せずにディレクトリ・オブジェクトを指定する必要があります。つまり、ログ・ファイルはディスク・ファイルに書き込まれ、Oracle ASMの記憶域には書き込まれません。かわりに、NOLOGFILE=YESを指定することもできます。ただし、この場合はログ・ファイルの書込みは行われません。

例

次に、LOGFILEパラメータの使用例を示します。この例では、ExportのFULLパラメータで示した例を実行して、expfull.dmpダンプ・ファイルを作成できます。

```
> impdp hr SCHEMAS=HR DIRECTORY=dpump_dir2 LOGFILE=imp.log
 DUMPFILE=dpump_dir1:expfull.dmp
```

LOGFILEパラメータにはディレクトリ・オブジェクトが指定されていないため、ログ・ファイルは、DIRECTORYパラメータに指定したディレクトリ・オブジェクトに書き込まれます。

関連トピック

- STATUS
- Oracle Automatic Storage Managementを使用可能にした場合のディレクトリ・オブジェクトの使用方法
- FULL

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.24 LOGTIME

Oracle Data Pump Importのコマンドライン・モードのLOGTIMEパラメータにより、インポート中にメッセージをタイムスタンプとともに表示することを指定します。

デフォルト

タイムスタンプは記録されません

用途

インポート中にメッセージをタイムスタンプとともに表示することを指定します。タイムスタンプを使用して、データ・ポンプ操作の異なるフェーズ間の経過時間を計算できます。この情報は、パフォーマンス問題を診断する場合や、将来の同じような操作の時間を見積もる場合に役立ちます。

構文および説明

```
LOGTIME=[NONE | STATUS | LOGFILE | ALL]
```

使用可能なオプションの定義は次のとおりです。

- NONE: ステータス・メッセージまたはログ・ファイル・メッセージにタイムスタンプを付けません(デフォルトと同じ)。
- STATUS: ステータス・メッセージにのみタイムスタンプを付けます。
- LOGFILE: ログ・ファイル・メッセージにのみタイムスタンプを付けます。
- ALL: ステータス・メッセージとログ・ファイル・メッセージの両方にタイムスタンプを付けます。

制限事項

- なし

例

次の例では、インポート操作中に表示されるすべてのステータス・メッセージとログ・ファイル・メッセージにタイムスタンプを記録します。

```
> impdp hr DIRECTORY=dpump_dir1 DUMPFILE=expdat.dmp SCHEMAS=hr LOGTIME=ALL
TABLE_EXISTS_ACTION=REPLACE
```

LOGTIMEの出力例は、エクスポートの「LOGTIME」パラメータを参照してください。

関連トピック

- LOGTIME

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.25 MASTER_ONLY

Oracle Data Pumpインポートのコマンドライン・モードのMASTER_ONLYパラメータは、データ・ポンプ制御ジョブ表の内容を確認できるように、データ・ポンプ制御ジョブ表のみをインポートしてジョブを停止するかどうかを示します。

デフォルト

NO

用途

データ・ポンプ制御ジョブ表の内容を確認できるように、データ・ポンプ制御ジョブ表のみをインポートしてジョブを停止するかどうかを示します。

構文および説明

```
MASTER_ONLY=[YES | NO]
```

制限事項

- NETWORK_LINKパラメータも指定する場合、MASTER_ONLY=YESはサポートされません。

例

```
> impdp hr SCHEMAS=hr DIRECTORY=dpump_dir1 LOGFILE=schemas.log
DUMPFILE=expdat.dmp MASTER_ONLY=YES
```

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.26 METRICS

Oracle Data Pump Importのコマンドライン・モードのMETRICSパラメータにより、ジョブに関する詳細情報をログ・ファイルに記録するかどうかを指定します。

デフォルト

NO

用途

ジョブに関する詳細情報をOracle Data Pumpログ・ファイルに記録するかどうかを示します。

構文および説明

```
METRICS=[YES | NO]
```

METRICS=YESを使用すると、オブジェクト数および経過時間がOracle Data Pumpログ・ファイルに記録されます。

制限事項

- なし

例

```
> impdp hr SCHEMAS=hr DIRECTORY=dpump_dir1 LOGFILE=schemas.log
DUMPFILE=expdat.dmp METRICS=YES
```

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.27 NETWORK_LINK

Oracleデータ・ポンプ・インポートのコマンドライン・モードでNETWORK_LINKパラメータを使用すると、有効なデータベース・リンクで識別されるソース・データベースからのインポートが可能になります。

デフォルト:

デフォルトはありません

用途

有効なデータベース・リンクによって指定されるソース・データベースからのインポートを可能にします。ソース・データベース・インスタンスのデータは、接続されたデータベース・インスタンスに直接書き込まれます。

構文および説明

```
NETWORK_LINK=source_database_link
```

NETWORK_LINKパラメータは、データベース・リンクを使用してインポートを開始します。つまり、impdpクライアントの接続先となるシステムから、source_database_linkで指定されたソース・データベースに接続し、そこからデータを取り出して、接続されたインスタンスのデータベースに直接書き込みます。ダンプ・ファイルは含まれません。

source_database_linkには、使用可能なデータベースへのデータベース・リンク名を指定する必要があります。対象インスタンスのデータベースにデータベース・リンクが指定されていない場合、ユーザーまたはDBAが、SQL CREATE DATABASE LINK文を使用してデータベース・リンクを作成する必要があります。

トランスポータブル・メソッドを使用してネットワーク・インポートを実行する場合は、インポートを開始する前に、ソース・データ・ファイルをターゲット・データベースにコピーする必要があります。

ソース・データベースが読取り専用の場合、接続されたユーザーは、デフォルト一時表領域として割り当てられたローカル管理一時表領域をソース・データベース上に持つ必要があります。それ以外の場合、ジョブは失敗します。

このパラメータは、FLASHBACK_SCN、FLASHBACK_TIME、ESTIMATE、TRANSPORT_TABLESPACESまたはTRANSPORTABLEのいずれかのパラメータを指定する場合に必要です。

Oracleデータ・ポンプ・インポートでは、次のタイプのデータベース・リンクの使用がサポートされています。

- パブリック固定ユーザー
- パブリック接続ユーザー
- パブリック共有ユーザー(リンク所有者により使用される場合のみ)
- プライベート共有ユーザー(リンク所有者により使用される場合のみ)
- プライベート固定ユーザー(リンク所有者により使用される場合のみ)

注意:

暗号化されていないネットワーク・リンクを介してインポート操作が行われる場合、すべてのデータはクリア・テキストとしてインポートされます。これは、データがデータベースで暗号化されている場合でも同様です。ネットワーク・セキュリティの詳細は、Oracle Database セキュリティ・ガイドを参照してください。

制限事項

- Oracleデータ・ポンプ・インポートでは、次のタイプのデータベース・リンクの使用はサポートされていません。
  - プライベート接続ユーザー
  - 現在のユーザー
- ContentTypeセットがあるSecureFiles、または現在Oracle Database File System Linksを介してSecureFilesセグメントの外に格納されているSecureFilesが表に含まれている場合、インポートの`NETWORK_LINK`パラメータはサポートされません。
- ネットワーク・インポートは進化した型の使用をサポートしません。
- データ・ポンプ操作がネットワーク・リンクを介して行われる場合、ソース・データベースとターゲット・データベースのバージョンの差違が2バージョン以下である必要があります。たとえば、一方のデータベースがOracle Database 12cの場合、他方のデータベースは12c、11gまたは10gである必要があります。Oracle Data Pumpはメジャー・バージョン番号(10g、11g、12cなど)のみをチェックします。具体的なリリース番号(12.1、12.2、11.1、11.2、10.1、10.2など)はチェックしません。
- インポート・ジョブを実行する`USERID`がターゲット・データベースの`DATAPUMP_IMP_FULL_DATABASE`ロールを持っている場合、そのユーザーは、ソース・データベースの`DATAPUMP_EXP_FULL_DATABASE`ロールも持っている必要があります。
- ネットワーク・モードのインポートでは、パラレル問合せ(PQ)子プロセスは使用されません。
- `NETWORK_LINK`パラメータも使用されていると、メタデータはパラレルでインポートできなくなります
- 全体トランスポータブル・インポートを使用してネットワークを介してデータベースを転送する場合、監査証跡情報自体がユーザー定義の表領域に格納されていると、管理表領域(`SYSTEM`や`SYSAUX`など)に格納された表に対して監査を有効にすることはできません。

例

次の構文例では、`source_database_link`を有効なデータベース・リンクの名前に置き換えます。

```
> impdp hr TABLES=employees DIRECTORY=dpump_dir1
NETWORK_LINK=source_database_link EXCLUDE=CONSTRAINT
```

この例では、ソース・データベースから`employees` 表(制約を除く)がインポートされます。ログ・ファイルは、`DIRECTORY`パラメータに指定した`dpump_dir1`に書き込まれます。

関連トピック

- PARALLEL

関連項目:

- データベース・リンクの詳細は、Oracle Database管理者ガイドを参照
- `CREATE DATABASE LINK`文の詳細は、Oracle Database SQL言語リファレンスを参照してください
- ローカル管理表領域の詳細は、Oracle Database管理者ガイドを参照してください

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.28 NOLOGFILE

Oracle Data Pump Importのコマンドライン・モードのNOLOGFILEパラメータにより、デフォルトでログ・ファイルを作成するかどうかを指定します。

デフォルト

NO

用途

デフォルトでログ・ファイルを作成するかどうかを指定します。

構文および説明

```
NOLOGFILE=[YES | NO]
```

NOLOGFILE=YESを指定すると、ログ・ファイルは作成されません。ただし、進捗とエラーに関する情報が、接続されているいずれかのクライアント(オリジナルのエクスポート操作を開始したクライアントを含む)の標準出力デバイスに書き込まれます。実行中のジョブに接続されているクライアントが存在しないときにNOLOGFILE=YESを指定すると、重要な進捗情報およびエラー情報が失われる危険性があります。

例

次に、NOLOGFILEパラメータの使用例を示します。

```
> impdp hr DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp NOLOGFILE=YES
```

このコマンドを実行すると、expfull.dmpダンプ・ファイルの全体インポート・モード(ファイル・ベース・インポートのデフォルト)が実行されます。NOLOGFILEにYESが設定されているため、ログ・ファイルは書き込まれません。

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.29 PARALLEL

Oracleデータ・ポンプ・インポートのコマンドライン・モードのPARALLELパラメータでは、パラレルにロードできるワーカー・プロセスの最大数を設定します。

デフォルト

1

用途

データ・ポンプ制御インポート・ジョブのかわりに動作するアクティブな実行のワーカー・プロセスの最大数を指定します。

構文および説明

```
PARALLEL=integer
```

integerに指定する値によって、インポート・ジョブのために動作するアクティブな実行プロセスの最大数が指定されます。この実行セットは、ワーカー・プロセスとパラレル入出力(I/O)サーバー・プロセスの組合せで構成されます。パラレルI/O操作でパラレル実行コーディネータとして機能するデータ・ポンプ制御プロセス、アイドル・ワーカー・プロセスおよびワーカー・プロセスは、この合計にカウントされません。このパラメータを使用して、リソース消費と経過時間のバランスをとることができます。

インポートのソースがファイルで構成されるダンプ・ファイル・セットの場合、同じファイルから複数のプロセスが読取り可能ですが、パフォーマンスは、I/O競合によって制限される可能性があります。

ジョブの実行中にPARALLELの値を増減するには、対話方式コマンド・モードを使用します。

ネットワーク・モード・インポートにおけるPARALLELの使用

ネットワーク・モード・インポート中、PARALLELパラメータによって、ジョブに割り当てることができるワーカー・プロセスの最大数が定義されます。ネットワーク・インポート・モード中のPARALLELパラメータの影響を理解するには、Oracle Data Pumpによって定義されるtable_dataオブジェクトの概念を理解することが重要です。Oracle Data Pumpでは、データを移動するとき、次の項目を個々のtable_dataオブジェクトとみなします。

- 完全な表(パーティションまたはサブパーティションされていないもの)
- パーティション(表がパーティション化されているがサブパーティション化されていない場合)
- サブパーティション(表がサブパーティション化されている場合)

たとえば:

- 非パーティション表scott.non_part_tableには、次の1つのtable_dataオブジェクトがあります。

  scott.non_part_table
- パーティション表scott.part_table (パーティションp1とパーティションp2を持つ)には、次の2つのtable_dataオブジェクトがあります。

  scott.part_table:p1

  scott.part_table:p2
- サブパーティション表scott.sub_part_table (パーティションp1とp2、およびサブパーティションp1s1、p1s2、p2s1およびp2s2を持つ)には、次の4つのtable_dataオブジェクトがあります。

  scott.sub_part_table:p1s1

  scott.sub_part_table:p1s2

  scott.sub_part_table:p2s1

  scott.sub_part_table:p2s2

ネットワーク・モード・インポート中、各table_dataオブジェクトには、PARALLELパラメータに指定された値までの独自のワーカー・プロセスが割り当てられます。ネットワーク・モードのインポートではパラレル問合せ(PQ)ワーカー・プロセスが使用されないため、パラレル問合せ(PQ)ワーカー・プロセスは割り当てられません。複数のtable_dataオブジェクトを同時にアンロードできます。ただし、各table_dataオブジェクトは、1つのプロセスを使用してアンロードされます。

Oracle RAC環境でのインポートにおけるPARALLELの使用

Oracle Real Application Clusters(Oracle RAC)環境でPARALLEL=1としてインポート操作を実行すると、すべてのOracle Data Pumpプロセスは、ジョブが開始されたインスタンス上に配置されます。そのため、ディレクトリ・オブジェクトは、そのインスタンスのローカル記憶域を示すことができます。

PARALLELを1より大きな値に設定してインポート操作を実行する場合は、ジョブが開始されたインスタンス以外のインスタンスにもOracle Data Pumpプロセスを置くことができます。したがって、ディレクトリ・オブジェクトは、すべてのOracle RACクラスタ・メンバー・ノードからアクセス可能な共有記憶域を指す必要があります。

制限事項

- このパラメータは、Oracle Database 11g以上のEnterprise Editionでのみ有効です。
- トランスポータブル表領域のメタデータは、パラレルでインポートできません。
- パラレルで(パラレル問合せワーカー・プロセスを使用して)表または表パーティションをインポートするには、

DATAPUMP_IMP_FULL_DATABASEロールが必要です。

- また、次のオブジェクトもパラレルでインポートできません。
  - TRIGGER
  - VIEW
  - OBJECT_GRANT
  - SEQUENCE
  - CONSTRAINT
  - REF_CONSTRAINT

例

次に、PARALLELパラメータの使用例を示します。

```
> impdp hr DIRECTORY=dpump_dir1 LOGFILE=parallel_import.log
JOB_NAME=imp_par3 DUMPFILE=par_exp%U.dmp PARALLEL=3
```

このコマンドにより、エクスポートのPARALLELパラメータの例を実行した場合に作成されるダンプ・ファイル・セットがインポートされます。ダンプ・ファイルの名前は、par_exp01.dmp、par_exp02.dmpおよびpar_exp03.dmpです。

関連トピック

- PARALLEL

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.30 PARALLEL_THRESHOLD

Oracle Data Pump Importコマンドライン・ユーティリティのPARALLEL_THRESHOLDパラメータでは、表サイズに基づいてデータ・ポンプが潜在的なパラレルDMLを計算するために使用する除数のサイズを指定します。

デフォルト

250MB

用途

PARALLEL_THRESHOLDは、単一の非パーティション表のエクスポート・ジョブまたはインポート・ジョブ、あるいはパーティション表の1つのパーティションでのみ使用する必要があります。ジョブでPARALLELを指定する場合、PARALLEL_THRESHOLDを指定して、インポートおよびエクスポート時にパラレル・データ操作文(PDML)を使用して表をエクスポートまたはインポートする必要があるかどうかをOracle Data Pumpが判断するために使用する除数のサイズを変更できます。デフォルトより小さい値を指定すると、より小さい表サイズでOracle Data Pumpのパラレル・アルゴリズムを使用できます。たとえば、100MBの表があり、PDML 5を使用して5単位に分割する場合は、PARALLEL_THRESHOLD=20Mを指定します。

構文および説明

パラメータ値は、しきい値サイズをバイト単位で指定します。

```
PARALLEL_THRESHOLD=size-in-bytes
```

単一の表のエクスポートまたはインポートで、より高い並列度が必要な場合は、小さい表または表パーティションの並列度を利用するために、PARALLEL_THRESHOLDを小さい値に設定できます。ただし、このリソース割当ての利点は、ロードまたはアンロード先となるファイル・システムのI/Oのパフォーマンスによって制限される可能性があります。また、表とメタデータ・オブジェクトの両方について、ジョブに複数のオブジェクトが含まれている場合、PARALLELおよびPARALLEL_THRESHOLDで指定されたPQ割当て要求の値は制限されます。表に割り当てられるPQプロセスの実際の量は、Oracle Data Pumpが同時に実行している操作の数の影響を受けており、並列度が共有される必要があります。オプティマイザによってSQLに対して生成されるデー

タベース、オプティマイザおよび実行計画は、ジョブで指定されたオブジェクトのロードまたはアンロードに使用される実際の並列度を決定します。

このパラメータのユースケースの1つ: Oracle Data Pumpを使用して、あるデータベースの大きな表を別のデータベースのより大きな表にロードします。例: OLTPデータベースからレポートまたはビジネス分析のデータ・ウェアハウス・データベースに週次売上データをアップロードします。

制限事項

PARALLEL_THRESHOLDは、PARALLELパラメータに1より大きい値が指定されている場合にのみ、組み合せて使用されます。

例

次に、PARALLEL_THRESHOLDパラメータを使用して表table_to_use_PDMLをエクスポートする例を示します。ここでは、PQプロセスの除数のサイズは1 KBに設定され、変数userおよびuser-passwordはインポートを実行しているユーザーのユーザーおよびパスワード(impdp)であり、ジョブ名はparathresh_exampleです。

```
impdp user/user-password ¥
    directory=dpump_dir ¥
    dumpfile=parathresh_example.dmp
    tables=table_to_use_PDML ¥
    parallel=8 ¥
    parallel_threshold=1K ¥
    job_name=parathresh_example
```

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.31 PARFILE

Oracle Data Pump Importのコマンドライン・モードのPARFILEパラメータでは、インポート・パラメータ・ファイルの名前を指定します。

デフォルト

デフォルトはありません

用途

インポート・パラメータ・ファイルの名前を指定します。

構文および説明

```
PARFILE=[directory_path]file_name
```

パラメータ・ファイルでは、ファイル内にデータ・ポンプ・パラメータを指定して、すべてのコマンドを個別に入力するかわりにそのファイルをコマンドラインに指定できます。これは、同じパラメータの組合せを何度も使用する場合に便利です。値の指定に引用符が必要なパラメータを使用する場合も、パラメータ・ファイルを使用することをお薦めします。

サーバーによって作成されて書き込まれるダンプ・ファイル、ログ・ファイルおよびSQLファイルとは異なり、パラメータ・ファイルはimpdpクライアントによって開かれて読み取られるため、パラメータ・ファイルにディレクトリ・オブジェクトは指定されません。パラメータ・ファイルのデフォルトの場所は、ユーザーのカレント・ディレクトリです。

パラメータ・ファイル内では、カンマはすべての改行文字に暗黙的に含まれるため、各行の最後にカンマを入力する必要はありません。長い表名など、折返しを含む長い行がある場合、現在の行の最後にバックスラッシュ継続文字(¥)を入力して次の行に継続します。

パラメータ・ファイルの内容がOracle Data Pumpログ・ファイルに書き込まれます。

制限事項

- PARFILEパラメータは、パラメータ・ファイル内には指定できません。

例

サンプルのパラメータ・ファイルhr_imp.parの内容が次のとおりであるとします。

```
TABLES= countries, locations, regions
DUMPFILE=dpump_dir2:exp1.dmp,exp2%U.dmp
DIRECTORY=dpump_dir1
PARALLEL=3
```

このパラメータ・ファイルを指定するには、次のコマンドを発行できます。

```
> impdp hr PARFILE=hr_imp.par
```

コマンドの結果として、表countries、locationsおよびregionsは、エクスポートのDUMPFILEパラメータの例を実行した場合に作成されるダンプ・ファイル・セットからインポートされます。(エクスポートのDUMPFILEパラメータを参照してください。)インポート・ジョブは、dpump_dir2で示される位置にあるexp1.dmpファイルを検索します。また、dpump_dir1によって示される位置にあるexp2nn.dmpの形式のすべてのダンプ・ファイルも検索します。そのジョブのログ・ファイルも、dpump_dir1に書き込まれます。

関連トピック

- DUMPFILE
- インポート・コマンドライン・モードについて

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.32 PARTITION_OPTIONS

Oracle Data Pump Importのコマンドライン・モードのPARTITION_OPTIONSパラメータでは、インポート操作中に表パーティションを作成する方法を指定します。

デフォルト

パーティション名がTABLESパラメータで指定され、TRANPORTABLE=ALWAYSが(インポート操作時またはエクスポート中に)設定されている場合、デフォルトはdepartitionです。それ以外の場合、デフォルトはnoneとなります。

用途

インポート操作中に表パーティションを作成する方法を指定します。

構文および説明

```
PARTITION_OPTIONS=[NONE | DEPARTITION | MERGE]
```

NONEの値を指定すると、エクスポート操作が実行されたときにシステム上に存在していたものと同じように表が作成されます。エクスポートがトランスポータブル・メソッドで実行されていて、パーティションまたはサブパーティションのフィルタが指定されていた場合は、NONEオプションやMERGEオプションは使用できません。そのような場合は、DEPARTITIONオプションを使用する必要があります。

DEPARTITIONの値を指定すると、パーティションまたはサブパーティションは、それぞれが新しい個別の表に昇格します。新規表のデフォルト名は、表とパーティションの名前、または表とサブパーティションの名前を適切に組み合せたものとなります。

MERGEの値を設定すると、すべてのパーティションとサブパーティションが1つの表に統合されます。

パーティション表のインポート中の並列処理では次のような影響があります。

- パーティション表が既存のパーティション表にインポートされた場合、データ・ポンプはPARALLELパラメータで指定される値にかかわらず、一度に1つのパーティションまたはサブパーティションのみを処理します。
- インポート先の表がまだ存在しておらず、データ・ポンプでそれを作成する必要がある場合、インポートは、インポートが開始されたときにPARALLELパラメータで指定された並列度までパラレルに実行されます。

制限事項

- ソース表のパーティションに基づいた表を作成して移入する場合は、非パーティション化を使用します。

  名前の競合を回避するために、PARTITION_OPTIONSの値がDEPARTITIONに設定されているときには、依存オブジェクト(制約や索引など)が該当する表と同時には作成されなくなります。このエラーメッセージ「ORA-39427: パーティション表の依存オブジェクトはインポートされません。」は、この制限による影響を受ける表がある場合に、ログ・ファイルに含まれます。このメッセージを抑止する場合は、EXCLUDEパラメータを使用して、インポートから依存オブジェクトを除外してください。
- PARTITION_OPTIONSの値をMERGEに設定していると、これらの表にドメイン索引は作成されません。このイベントが発生すると、エラー「ORA-39426: パーティション表のドメイン索引はインポートされません。」がログ・ファイルに報告されます。このメッセージを抑止する場合は、EXCLUDEパラメータを使用して索引を除外してください(EXCLUDE=DOMAIN_INDEX)。
- ダンプ・ファイルを作成したエクスポート操作がトランスポータブル・メソッドによって実行されていたときに、パーティションまたはサブパーティションを指定していた場合は、インポート操作でDEPARTITIONオプションを使用する必要があります。
- ダンプ・ファイルを作成したエクスポート操作がトランスポータブル・メソッドにより実行されている場合、インポート操作でPARTITION_OPTIONS=MERGEは使用できません。
- 非パーティション化されるオブジェクトに対する権限付与がある場合は、エラー・メッセージが生成され、オブジェクトはロードされません。

例

次の例は、sh.sales表がsales.dmpという名前のダンプ・ファイルにエクスポートされていることを前提としています。ここでは、mergeオプションを使用して、sh.sales内のすべてのパーティションをscottスキーマ内の非パーティション表にマージします。

```
> impdp system TABLES=sh.sales PARTITION_OPTIONS=MERGE
DIRECTORY=dpump_dir1 DUMPFILE=sales.dmp REMAP_SCHEMA=sh:scott
```

関連トピック

- TRANSPORTABLE

関連項目:

PARTITION_OPTIONS=DEPARTITIONを使用してインポート操作を実行する例は、エクスポートのTRANSPORTABLEパラメータを参照してください

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.33 QUERY

Oracle Data Pump Importコマンドライン・モードのQUERYパラメータを使用すると、インポート対象のデータをフィルタ処理する問合せ句を指定できます。

デフォルト

デフォルトはありません

用途

インポート対象のデータをフィルタ処理する問合せ句を指定できます。

構文および説明

```
QUERY=[[schema_name.]table_name:]query_clause
```

通常、query_clauseでは、ファイングレイン行選択のためのSQL WHERE句を使用します。ただし、任意のSQL句を使用できます。たとえば、ORDER BY句を使用すると、ヒープ構成表から索引構成表への移行を高速化できます。スキーマおよび表名を指定しなかった場合は、ソース・ダンプ・ファイル・セットまたはデータベース内のすべての表に問合せが適用されます(この場合、問合せは、これらのすべての表に対して有効である必要があります)。表固有の問合せは、すべての表に適用される問合せより優先されます。

特定の表に問合せを適用する場合は、表名と問合せ句をコロン(:)で区切る必要があります。表固有の問合せは複数指定できますが、1つの表に指定できるのは1つの問合せのみです。

QUERYパラメータと一緒にNETWORK_LINKパラメータを指定する場合、リモート(ソース)ノード上にあるquery_clauseで指定されるすべてのオブジェクトは、NETWORK_LINK値で明示的に修飾する必要があります。それ以外の場合、データ・ポンプではオブジェクトがローカル(ターゲット)ノード上に存在することを前提とし、存在しない場合はエラーが返され、リモート(ソース)システムからの表のインポートは失敗します。

たとえば、NETWORK_LINK=dblink1と指定した場合、QUERYパラメータのquery_clauseは、次の例に示すようにそのリンクを指定する必要があります。

```
QUERY=(hr.employees:"WHERE last_name IN(SELECT last_name
FROM hr.employees@dblink1)")
```

オペレーティング・システムによっては、このパラメータの値を指定するときに引用符とともにエスケープ文字を使用する必要があります。このパラメータはパラメータ・ファイルで指定することをお薦めします。パラメータ・ファイルを使用すると、コマンドラインでは必要なエスケープ文字の数を減らすことができます。「インポート・コマンドライン・モードについて」を参照してください。

QUERYパラメータを使用すると、外部表による方法(ダイレクト・パスによる方法ではなく)でデータベース・アクセスが実行されます。

表固有の問合せで自分のスキーマ以外のスキーマを指定するには、その特定の表に対するアクセス権限が付与されている必要があります。

制限事項

- QUERYパラメータを次のパラメータとともに使用することはできません。
  - CONTENT=METADATA_ONLY
  - SQLFILE
  - TRANSPORT_DATAFILES
- 表にQUERYパラメータが指定されている場合、データ・ポンプは外部表を使用してターゲット表をロードします。外部表

は、SQLのINSERT文をSELECT句とともに使用します。QUERYパラメータの値は、INSERT文のSELECT部分にあるWHERE句に含まれています。QUERYパラメータにロードする表と一致する名前の列がある他の表への参照が含まれていて、これらの列が問合せで使用される場合は、表別名を使用して、ロードする表内の列と、SELECT文内の同じ名前を持つ列を区別する必要があります。

たとえば、sh.customers表にある顧客のクレジットの上限に基づいてsh.sales表のサブセットをインポートするとします。次の例で、ロードされる表のデータ・ポンプで使用される表別名はKU$です。KU$を使用して、sh.salesをロードするためにQUERYパラメータ内のcust_idフィールドを修飾します。この結果、データ・ポンプによって、クレジットの上限が$10,000を超える顧客の行のみがインポートされます。

```
QUERY='sales:"WHERE EXISTS (SELECT cust_id FROM customers c
WHERE cust_credit_limit > 10000 AND ku$.cust_id = c.cust_id)"'
```

表別名としてKU$を使用しないと、すべての行がロードされます。

```
QUERY='sales:"WHERE EXISTS (SELECT cust_id FROM customers c
WHERE cust_credit_limit > 10000 AND cust_id = c.cust_id)"'
```

- QUERYに使用できる文字列の最大長は、引用符を含めて4000バイトです。つまり、許容される実際の最大長は3998バイトです。

例

次に、QUERYパラメータの使用例を示します。この例では、ExportのFULLパラメータで示した例を実行して、expfull.dmpダンプ・ファイルを作成できます。詳細は、エクスポートのFULLパラメータを参照してください。QUERY値には引用符を使用するため、パラメータ・ファイルを使用することをお薦めします。

次の内容のパラメータ・ファイルquery_imp.parを作成したとします。

```
QUERY=departments:"WHERE department_id < 120"
```

次のコマンドを入力します。

```
> impdp hr DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp
  PARFILE=query_imp.par NOLOGFILE=YES
```

expfull.dmp内のすべての表はインポートされますが、departments表については、QUERYパラメータに指定した基準を満たすデータのみがインポートされます。

関連トピック

- インポート・コマンドライン・モードについて
- FULL

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.34 REMAP_DATA

Oracle Data Pump Importのコマンドライン・モードのREMAP_DATAパラメータを使用すると、新規データベースへの挿入時に、データを再マップできます。

デフォルト

デフォルトはありません

用途

REMAP_DATAパラメータを使用すると、新規データベースへの挿入時に、データを再マップできます。一般的には、プライマリ・キーを再生成して、ターゲット・データベース上の既存の表に表をインポートする場合の競合を回避するために使用されます。

ダンプ・ファイルまたはリモート・データベースのいずれかから、指定した列の値をソースとして取得するには、再マップ・ファンクションを指定します。再マップ・ファンクションを指定すると、ターゲット・データベースの元の値を置き換える再マップした値が返されます。

同じファンクションを、ダンプされる複数の列に適用できます。このファンクションは、参照制約で子と親両方の列を再マップするときに整合性を保つ必要がある場合に役立ちます。

構文および説明

```
REMAP_DATA=[schema.]tablename.column_name:[schema.]pkg.function
```

次のリストは、各構文要素を構文で出現する順に示したものです。

schema: 再マップする表を含むスキーマ。デフォルトでは、このスキーマはインポートを実行するユーザーのスキーマです。

tablename: 列の再マップが行われる表。

column_name: データの再マップが行われる列。

schema: 再マップ・ファンクションを含むユーザー作成のPL/SQLパッケージを含むスキーマ。デフォルトでは、これはインポートを実行するユーザーのスキーマです。

pkg: 再マップ・ファンクションを含むユーザー作成のPL/SQLパッケージの名前。

function: 指定した表の各行で、列表を再マップする場合にコールされるPL/SQL内のファンクションの名前。

制限事項

- ソース引数および戻り値のデータ型とサイズはともに、表内の指定した列のデータ型とサイズ一致している必要があります。
- 再マップ・ファンクションでは、自律型トランザクション以外のコミットまたはロール・バックを実行できません。
- REMAP_DATAパラメータの値としてのシノニムの使用はサポートされていません。たとえば、hrスキーマ内のregions表にregnのシノニムがある場合は、regnをREMPA_DATA仕様の一部として指定した場合にエラーが返されます。
- リモート表のLOB列データの再マップはサポートされていません。
- REMAP_DATAでは、ユーザー定義の型、ユーザー定義の型の属性、LONG、REF、VARRAY、ネストされた表、BFILEおよびXMLtypeの型の列はサポートされません。

例

次の例では、plusxという名前のファンクションを格納するremapという名前のパッケージが作成されており、このファンクションはemployees表内のfirst_nameの値を変更すると想定しています。

```
> impdp hr DIRECTORY=dpump_dir1 DUMPFILE=expschema.dmp
TABLES=hr.employees REMAP_DATA=hr.employees.first_name:hr.remap.plusx
```

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.35 REMAP_DATAFILE

Oracle Data Pump Importのコマンドライン・モードのREMAP_DATAFILEパラメータにより、ソース・データ・ファイルを参照するすべてのSQL文において、ソース・データファイル名がターゲット・データファイル名に変更されます。

デフォルト

デフォルトはありません

用途

ソース・データ・ファイルが参照されるすべてのSQL文(CREATE TABLESPACE、CREATE LIBRARYおよびCREATE DIRECTORY)で、ソース・データ・ファイルの名前をターゲット・データ・ファイル名に変更します。

構文および説明

```
REMAP_DATAFILE=source_datafile:target_datafile
```

データ・ファイルの再マップは、ファイル名のネーミング規則が異なるプラットフォーム間でデータベースを移動する場合に有効です。source_datafileとtarget_datafileの名前は、SQL文で指定するとおりのものである必要があります。コロンが有効なファイル指定文字として使用されるプラットフォームでの曖昧さを排除するために、データ・ファイル名は引用符で囲むことをお薦めします。

オペレーティング・システムによっては、このパラメータの値を指定するときに引用符を使用する場合、エスケープ文字も必要になる可能性があります。このパラメータはパラメータ・ファイルで指定することをお薦めします。パラメータ・ファイルを使用すると、コマンドラインでは必要なエスケープ文字の数を減らすことができます。

このパラメータを指定するには、DATAPUMP_IMP_FULL_DATABASEロールが必要です。

例

次の内容のパラメータ・ファイルpayroll.parを作成したとします。

```
DIRECTORY=dpump_dir1
FULL=YES
DUMPFILE=db_full.dmp
REMAP_DATAFILE="'DB1$:[HRDATA.PAYROLL]tbs6.dbf':'/db1/hrdata/payroll/tbs6.dbf'"
```

次のコマンドを発行します。

```
> impdp hr PARFILE=payroll.par
```

この例では、インポート中に、すべてのSQL DDL文に対するVMSファイル指定(DR1$:[HRDATA.PAYROLL]tbs6.dbf)をUnixファイル指定(/db1/hrdata/payroll/tbs6.dbf)に再マップします。ダンプ・ファイルdb_full.dmpは、ディレクトリ・オブジェクトdpump_dir1によって位置が示されます。

関連トピック

- インポート・コマンドライン・モードについて

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.36 REMAP_DIRECTORY

Oracle Data Pump Importのコマンドライン・モードのREMAP_DIRECTORYパラメータを使用すると、プラットフォーム間でデータベースを移動するときにディレクトリを再マップできます。

デフォルト

デフォルトはありません

用途

REMAP_DIRECTORYパラメータは、ソース・ディレクトリがファイルまたはディレクトリ指定全体の左端部分にあるすべてのSQL文(CREATE TABLESPACE、CREATE LIBRARYおよびCREATE DIRECTORY)において、ソース・ディレクトリ文字列を

ターゲット・ディレクトリ文字列に変更します。

構文および説明

```
REMAP_DIRECTORY=source_directory_string:target_directory_string
```

ディレクトリの再マップは、ファイルのネーミング規則が異なるプラットフォーム間でデータベースを移動する場合に便利です。元のデータ・ファイル名を維持してディレクトリ・ファイル指定のみを変更する必要がある場合は、これにより、ディレクトリ内の複数のデータ・ファイルを簡単に再マップできます。

source_directory_stringおよびtarget_directory_stringは、SQL文に出現する内容に正確に一致する必要があります。また、個々のソースおよびターゲット・プラットフォームごとに、ディレクトリ・ファイル終了記号でディレクトリを適切に終了することをお薦めします。コロンが有効なディレクトリ・ファイル指定文字として使用されるプラットフォームでの曖昧さを排除するために、ディレクトリ名は引用符で囲むことをお薦めします。

オペレーティング・システムによっては、このパラメータの値を指定するときに引用符を使用する場合、エスケープ文字も必要になる可能性があります。このパラメータはパラメータ・ファイルで指定することをお薦めします。パラメータ・ファイルを使用すると、コマンドラインでは必要なエスケープ文字の数を減らすことができます。

このパラメータを指定するには、DATAPUMP_IMP_FULL_DATABASEロールが必要です。

制限事項

- REMAP_DIRECTORYおよびREMAP_DATAFILEは、相互に排他的なパラメータです。

例

たとえば、次のデータ・ファイルを再マップするとします。

```
DB1$:[HRDATA.PAYROLL]tbs5.dbf
DB1$:[HRDATA.PAYROLL]tbs6.dbf
```

また、次の内容のパラメータ・ファイルpayroll.parがあるとします。

```
DIRECTORY=dpump_dir1
FULL=YES
DUMPFILE=db_full.dmp
REMAP_DIRECTORY="'DB1$:[HRDATA.PAYROLL]':'/db1/hrdata/payroll/'"
```

この場合、次のコマンドを発行できます。

```
> impdp hr PARFILE=payroll.par
```

この例では、インポート中に、すべてのSQL DDL文について、VMSファイル指定(DB1$:[HRDATA.PAYROLL]tbs5.dbfおよびDB1$:[HRDATA.PAYROLL]tbs6.dbf)がUNIXファイル指定(/db1/hrdata/payroll/tbs5.dbfおよび/db1/hrdata/payroll/tbs6.dbf)に再マップされます。ダンプ・ファイルdb_full.dmpは、ディレクトリ・オブジェクトdpump_dir1によって位置が示されます。

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.37 REMAP_SCHEMA

Oracle Data Pump Importのコマンドライン・モードのREMAP_SCHEMAパラメータにより、すべてのオブジェクトをソース・スキーマからターゲット・スキーマにロードします。

デフォルト

デフォルトはありません

用途

ソース・スキーマにあるすべてのオブジェクトをターゲット・スキーマにロードします。

構文および説明

```
REMAP_SCHEMA=source_schema:target_schema
```

複数のREMAP_SCHEMA行を指定できますが、ソース・スキーマは行ごとに異なっている必要があります。ただし、異なるソース・スキーマを同じターゲット・スキーマにマップすることはできます。マッピングが不完全である可能性があります。このトピックの「制限事項」の項を参照してください。

再マッピングするスキーマがインポート前に存在しない場合、インポート操作によってそのスキーマを作成できます。ただし、SYSTEMユーザーにREMAP_SCHEMAがある場合を除きます。REMAP_SCHEMAのターゲット・スキーマは、インポート前に存在している必要があります。スキーマを作成するには、ダンプ・ファイル・セットにソース・スキーマに必要なCREATE USERメタデータが含まれている必要があり、十分な権限でインポートを実行する必要があります。たとえば、次のExportコマンドの場合、ユーザーSYSTEMには必要な権限があるため、スキーマの作成に必要なメタデータを含むダンプ・ファイル・セットが作成されます。

```
> expdp system SCHEMAS=hr
Password: password
> expdp system FULL=YES
Password: password
```

スキーマの作成に必要なメタデータがダンプ・ファイル・セットに含まれていない場合や、ユーザーに必要な権限がない場合は、インポート操作を実行する前にターゲット・スキーマを作成しておく必要があります。権限が付与されていないダンプ・ファイルには、インポートでスキーマを自動作成するために必要な情報が含まれていないため、インポート前にターゲット・スキーマ作成しておく必要があります。

Oracle Database 11gより前のOracle Databaseリリースでは、インポート操作でスキーマが作成された場合、インポートの完了後に、有効なパスワードをそのスキーマに割り当てて接続する必要があります。この場合、次のSQL文を使用してパスワードを割り当てることができます。権限が必要であることに注意してください。

```
SQL> ALTER USER schema_name IDENTIFIED BY new_password
```

Oracle Database 11gリリース1 (11.1.0.1)より後のOracle Databaseリリースでは、スキーマ・パスワードをリセットする必要がなくなりました。元のパスワードは引き続き有効です。

制限事項

- 権限のないユーザーは、自分のスキーマが再マップのターゲット・スキーマの場合にのみ、スキーマの再マップを実行できます。(特権ユーザーがスキーマの再マップを実行する場合は、制限がありません。)たとえば、SCOTTは、自分のBLAKEのオブジェクトをSCOTTに再マップできますが、SCOTTは、SCOTTのオブジェクトをBLAKEに再マップすることはできません。
- インポートで検出できない一部のスキーマ参照があるため、マッピングは不完全となる可能性があります。たとえば、インポートでは、型定義、ビュー、プロシージャおよびパッケージの本体に埋め込まれたスキーマ参照は検出されません。
- トリガーの場合、REMAP_SCHEMAはトリガー所有者にのみ影響を及ぼします。
- 再マップするスキーマ内の表にユーザー定義のオブジェクト型が含まれており、その表が、エクスポートされる時刻とインポートしようとする時刻の間に変更される場合、その表のインポートは失敗します。ただし、インポート操作自体は続行されます。

- デフォルトでは、ソース・データベースのスキーマ・オブジェクトにオブジェクト識別子(OID)がある場合、それらと同じOIDを持つターゲット・データベースにインポートされます。オブジェクトがエクスポート元と同じデータベースであって異なるスキーマにインポートされた場合、新しい(インポートされた)オブジェクトのOIDは、既存のオブジェクトと同じものになり、インポートは失敗します。インポートを成功させるには、インポート時にTRANSFORM=OID:Nパラメータも指定する必要があります。変換OID:Nにより新しいオブジェクトに対して新しいOIDが作成され、インポートが成功します。

例

ユーザーSYSTEMとして、次のエクスポートおよびインポート・コマンドを実行して、hrスキーマをscottスキーマに再マップするとします。

```
> expdp system SCHEMAS=hr DIRECTORY=dpump_dir1 DUMPFILE=hr.dmp
> impdp system DIRECTORY=dpump_dir1 DUMPFILE=hr.dmp REMAP_SCHEMA=hr:scott
```

この例では、インポート前にユーザーscottが存在する場合、インポートのREMAP_SCHEMAコマンドによって、hrスキーマにあるオブジェクトが既存のscottスキーマに追加されます。インポート後、scottスキーマに既存のパスワードで(パスワードの再設定なしで)接続できます。

インポート操作の実行前にユーザーscottが存在しない場合は、インポートによって、このユーザーがパスワードなしで自動作成されます。この処理が可能なのは、ダンプ・ファイルhr.dmpが、スキーマの作成に必要なメタデータを含むダンプ・ファイルを作成する権限を所有するSYSTEMによって作成されたためです。ただし、インポート完了後に、ターゲット・データベース上のscottのパスワードを再設定しないかぎり、インポートの完了時にscottには接続できません。

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.38 REMAP_TABLE

Oracle Data Pump Importのコマンドライン・モードのREMAP_TABLEパラメータを使用すると、インポート操作中に表の名前を変更できます。

デフォルト

デフォルトはありません

用途

インポート操作中に、表の名前を変更できます。

構文および説明

次のいずれかの構文を使用できます(「使用上のノート」を参照)。

```
REMAP_TABLE=[schema.]old_tablename[.partition]:new_tablename
```

または

```
REMAP_TABLE=[schema.]old_tablename[:partition]:new_tablename
```

表が非パーティション化されている場合は、REMAP_TABLEパラメータを使用して表全体の名前を変更することや、表パーティションの名前を変更することができます(「PARTITION_OPTIONS」を参照)。

REMAP_TABLEを使用して、エクスポートされた表パーティションの自動ネーミングを上書きすることもできます。

使用上のノート

最初の構文では、REMAP_TABLE=A.B:Cを指定すると、インポートによってAはスキーマ名、Bは古い表名、Cは新しい表名

とみなされます。最初の構文を使用して、非パーティション表にするパーティションの名前を変更するには、スキーマ名を指定する必要があります。

2番目の構文を使用して、非パーティション表にするパーティションの名前を変更するには、その表を古い表名で修飾します。スキーマ名は必要ありません。

制限事項

- REMAP_TABLEパラメータでは、ユーザーが作成した表のみが処理されます。データ・ポンプには、内部的に作成された依存表に関する十分な情報がありません。そのため、内部的に作成された表をREMAP_TABLEパラメータで再マップできません。
- インポートによって作成されたオブジェクトのみ、再マップされます。既存の表は再マップされないことに注意してください。
- 再マッピングする表に、同じスキーマ内の名前付きの制約が含まれ、表の作成時に制約を作成する必要がある場合、REMAP_TABLEパラメータは機能しません

例

次の例では、REMAP_TABLEパラメータを使用して、employees表をempsという新しい名前に変更します。

```
> impdp hr DIRECTORY=dpump_dir1 DUMPFILE=expschema.dmp
TABLES=hr.employees REMAP_TABLE=hr.employees:emps
```

関連トピック

- PARTITION_OPTIONS

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.39 REMAP_TABLESPACE

Oracle Data Pump Importのコマンドライン・モードのREMAP_TABLESPACEパラメータにより、インポート用に選択されたすべてのオブジェクトを再マップし、ソース表領域内の永続データがターゲット表領域内に作成されるようにします。

デフォルト

デフォルトはありません

用途

ターゲット表領域に作成するソース表領域内の永続データを使用して、インポート用に選択されたすべてのオブジェクトを再マップします。

構文および説明

```
REMAP_TABLESPACE=source_tablespace:target_tablespace
```

複数のREMAP_TABLESPACEパラメータを指定できますが、ソース表領域はパラメータごとに1つのみです。ターゲット・スキーマのターゲット表領域には、十分な割当て制限が必要です。

データ・ポンプ・インポートで表領域を再マップする唯一の方法は、REMAP_TABLESPACEです。REMAP_TABLESPACEを使用する方法は、オリジナルのインポート・ユーティリティの機能よりも簡単で正確な方法です。その方法には、表領域の副次句の数など多くの制限事項があり、一部のDDLコマンドを正常に実行できない場合がありました。

これに対し、REMAP_TABLESPACEパラメータを使用するデータ・ポンプ・インポートの方法は、ユーザーを含むすべてのオブジェクトに対して、DDL文に含まれる表領域副次句の数にかかわりなく有効に使用できます。

制限事項

- データ・ポンプ・インポートで再マップ可能なトランスポータブル・インポートの表領域は、互換性レベルが10.1以降に設定されているデータベースにあるもののみです。
- インポートによって作成されたオブジェクトのみ、再マップされます。特に、TABLE_EXISTS_ACTIONがSKIP、TRUNCATEまたはAPPENDに設定されている場合、既存の表の表領域は再マップされません。

例

次に、REMAP_TABLESPACEパラメータの使用例を示します。

```
> impdp hr REMAP_TABLESPACE=tbs_1:tbs_6 DIRECTORY=dpump_dir1
  DUMPFILE=employees.dmp
```

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.40 SCHEMAS

Oracle Data Pump Importのコマンドライン・モードのSCHEMASパラメータにより、スキーマ・モード・インポートを実行することを指定します。

デフォルト

デフォルトはありません

用途

スキーマ・モード・インポートの実行を指定します。

構文および説明

```
SCHEMAS=schema_name [,...]
```

DATAPUMP_IMP_FULL_DATABASEロールがある場合は、インポートするスキーマのリストをこのパラメータで指定して、スキーマ・モードのインポートを実行できます。まず、システムおよびロールの権限、パスワード履歴などを含むユーザー定義がインポートされます(存在しない場合)。次に、スキーマ内のすべてのオブジェクトがインポートされます。権限のないユーザーは、自分のスキーマか、自分のスキーマに再マップされるスキーマのみを指定できます。この場合、スキーマ定義についての情報はインポートされず、その定義内に含まれているオブジェクトのみがインポートされます。

このインポート・モードを使用してインポートされる内容を制限するには、フィルタ処理を使用します。

スキーマ・モードは、ネットワーク・ベース・インポートを実行する際のデフォルト・モードです。

例

次に、SCHEMASパラメータの使用例を示します。この例では、ExportのSCHEMASパラメータで示した例を実行して、expdat.dmpファイルを作成できます。

```
> impdp hr SCHEMAS=hr DIRECTORY=dpump_dir1 LOGFILE=schemas.log
DUMPFILE=expdat.dmp
```

hrスキーマは、expdat.dmpファイルからインポートされます。ログ・ファイルschemas.logは、dpump_dir1に書き込まれます。

関連トピック

- インポート操作中のフィルタ処理

- SCHEMAS

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.41 SERVICE_NAME

Oracle Data Pump Importのコマンドライン・モードのSERVICE_NAMEパラメータでは、CLUSTERパラメータとともに使用するサービス名を指定します。

デフォルト

デフォルトはありません

用途

CLUSTERパラメータとともに使用するサービス名を指定するために使用します。

構文および説明

```
SERVICE_NAME=name
```

SERVICE_NAMEパラメータをCLUSTER=YESパラメータとともに使用して、リソース・グループに属しているOracle Real Application Clusters(Oracle RAC)インスタンス群(通常は全Oracle RACインスタンスのサブセット)を定義するリソース・グループに関連付けられている、既存のサービスを指定できます。

サービス名は、リソース・グループおよびそのリソース・グループに対して定義されたインスタンスを決定する目的でのみ使用されます。ジョブが開始されたインスタンスは、そのインスタンスがリソース・グループに含まれているかどうかとは関係なく、常に使用されます。

SERVICE_NAMEパラメータは、CLUSTER=NOと同時に指定されると、無視されます。

インスタンスA、B、CおよびDを含むOracle RAC構成があります。インスタンスA、BおよびCのみで構成されるリソース・グループを持つmy_serviceというサービスが存在するとします。そのような場合は、次の内容が成立します。

- Oracle Data Pumpジョブをインスタンス Aで開始したときに、CLUSTER=YESを指定し(またはデフォルトのYESをそのまま使用し)、SERVICE_NAMEパラメータを指定しなかった場合、Oracle Data Pumpは、指定されている並列度に応じて、すべてのインスタンス(A、B、CおよびD)にワーカーを作成します。
- Oracle Data Pumpジョブをインスタンス Aで開始し、CLUSTER=YESおよびSERVICE_NAME=my_serviceを指定した場合、ワーカーは、インスタンスA、BおよびCのみで開始可能です。
- Oracle Data Pumpジョブをインスタンス Dで開始し、CLUSTER=YESおよびSERVICE_NAME=my_serviceを指定した場合、ワーカーは、インスタンスA、B、CおよびDのみで開始可能です。インスタンスDはmy_serviceに存在しませんが、インスタンスDはジョブが開始されたインスタンスであるため含まれています。
- Oracle Data Pumpジョブをインスタンス Aで開始し、CLUSTER=NOを指定した場合、SERVICE_NAMEパラメータを指定しても無視され、すべてのプロセスがインスタンスAで開始されます。

例

```
> impdp system DIRECTORY=dpump_dir1 SCHEMAS=hr
  SERVICE_NAME=sales NETWORK_LINK=dbs1
```

この例では、hrスキーマのスキーマ・モード・ネットワーク・インポートを開始します。コマンドラインではCLUSTER=YESは指定されていませんが、それはデフォルトの動作なので、ジョブではサービス名salesに関連付けられたリソース・グループのすべてのインスタンスが使用されます。dbs1のNETWORK_LINK値は、データのインポート元であるソース・データベースの名前に置き換えま

す。(ネットワーク・インポートではダンプ・ファイルは生成されないことに注意してください。)

NETWORK_LINKパラメータは、単にこの例の一部として使用されています。SERVICE_NAMEパラメータの使用時は必須ではありません。

関連トピック

- CLUSTER

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.42 SKIP_UNUSABLE_INDEXES

Oracle Data Pump Importのコマンドライン・モードのSKIP_UNUSABLE_INDEXESパラメータにより、(システムまたはユーザーのどちらかによって)索引使用禁止状態に設定されている索引を持つ表のロードをインポート・ユーティリティでスキップするかどうかを指定します。

デフォルト

Oracle Databaseの構成パラメータSKIP_UNUSABLE_INDEXESの値。

用途

インポートで、(システムまたはユーザーのいずれかによって)索引使用禁止に設定されている索引を持つ表をロードするかどうかを指定します。

構文および説明

```
SKIP_UNUSABLE_INDEXES=[YES | NO]
```

SKIP_UNUSABLE_INDEXESがYESに設定されているときに、索引が使用禁止の状態になっている表またはパーティションが検出された場合、その表やパーティションは、使用禁止の索引が存在しない場合と同様にロードされます。

SKIP_UNUSABLE_INDEXESがNOに設定されているときに、索引が使用禁止の表またはパーティションが検出された場合、その表やパーティションはロードされません。索引が使用禁止に設定されていない他の表に対しては、行の挿入時に更新が行われます。

SKIP_UNUSABLE_INDEXESパラメータが指定されていない場合は、Oracle Databaseの構成パラメータSKIP_UNUSABLE_INDEXESの設定値が参照され、使用禁止の索引の処理が決定されます。このパラメータのデフォルト値はyです。

制約の施行に使用される索引に使用禁止のマークが付けられている場合、その表にデータはインポートされません。

> ノート:
>
> SKIP_UNUSABLE_INDEXES は、既存の表にデータをインポートする場合にのみ有効です。インポートの一部として表が作成される場合、実質的な効果はありません。この場合、表および索引は新規に作成され、使用禁止のマークは付けられません。

例

次に、SKIP_UNUSABLE_INDEXESパラメータの使用例を示します。この例では、ExportのFULLパラメータで示した例を実行して、expfull.dmpダンプ・ファイルを作成できます。

```
> impdp hr DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp LOGFILE=skip.log
```

```
SKIP_UNUSABLE_INDEXES=YES
```

関連トピック

- FULL

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.43 SOURCE_EDITION

Oracle Data Pump Importのコマンドライン・モードのSOURCE_EDITIONパラメータでは、オブジェクトのフェッチ元となるリモート・ノード上のデータベース・エディションを指定します。

デフォルト

オブジェクトのフェッチ元となるリモート・ノード上のデフォルトのデータベース・エディション。

用途

オブジェクトのフェッチ元となるリモート・ノード上のデータベース・エディションを指定します。

構文および説明

```
SOURCE_EDITION=edition_name
```

SOURCE_EDITION=edition_nameを指定すると、そのエディションのオブジェクトがインポートされます。Oracle Data Pumpは、変更されずに継承されているすべてのオブジェクトと、変更されているすべての実際のオブジェクトを選択します。

このパラメータが指定されていない場合は、デフォルトのエディションが使用されます。指定されたエディションが存在しない場合や使用できない場合は、エラー・メッセージが返されます。

制限事項

- SOURCE_EDITIONパラメータがインポート操作で有効なのは、NETWORK_LINKパラメータも指定されている場合のみです。
- このパラメータは、バージョニング可能な同じオブジェクトのバージョンがデータベース内に2つ以上ある場合にのみ有効です。
- ジョブ・バージョンは、11.2以上に設定する必要があります。

例

次に、インポートのSOURCE_EDITIONパラメータの使用例を示します。

```
> impdp hr DIRECTORY=dpump_dir1 SOURCE_EDITION=exp_edition
NETWORK_LINK=source_database_link EXCLUDE=USER
```

この例では、オブジェクトのインポート元となるexp_editionというエディションがシステム上に存在することを前提としています。インポート・モードは指定されていないため、デフォルト(スキーマ・モード)が使用されます。source_database_linkは、データのインポート元であるソース・データベースの名前に置き換えます。EXCLUDE=USERパラメータは、ユーザーの定義のみを除外し、ユーザーのスキーマに含まれるオブジェクトは除外しません。これはネットワーク・インポートであるため、ダンプ・ファイルは生成されません。

関連トピック

- NETWORK_LINK
- VERSION

関連項目:

- エディションの作成方法の詳細は、『Oracle Database SQL言語リファレンス』の「CREATE EDITION」を参照してください
- エディション機能(継承されたオブジェクトと実際のオブジェクトなど)の詳細は、『Oracle Database開発ガイド』の「エディション」を参照してください

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.44 SQLFILE

Oracle Data Pump Importのコマンドライン・モードのSQLFILEパラメータでは、選択した他のインポート・パラメータに基づいて、インポート・ユーティリティによって実行準備が整えられたすべてのSQL DDLの書込み先となるファイルを指定します。

デフォルト

デフォルトはありません

用途

選択した他のインポート・パラメータに基づいて、インポートによって実行準備が整えられたすべてのSQL DDLが書き込まれるファイルを指定します。

構文および説明

```
SQLFILE=[directory_object:]file_name
```

file_nameには、ジョブで実行する準備が整ったDDLをインポート・ジョブが書き込む場所を指定します。そのSQLは、実際には実行されず、ターゲット・システムも変更されません。ファイルは、別のディレクトリ・オブジェクトを明示的に指定しないかぎり、DIRECTORYパラメータに指定されたディレクトリ・オブジェクトに書き込まれます。このパラメータで指定した名前と一致する名前を持つ既存のファイルはすべて上書きされます。

パスワードは、SQLファイルに含まれないことに注意してください。たとえば、実行したDDLにCONNECT文が含まれている場合、この文はコメントで置き換えられ、スキーマ名のみが示されます。次の例で、ダッシュ(--)はコメントが後に続くことを示しています。hrスキーマ名が表示されますが、パスワードは表示されません。

```
-- CONNECT hr
```

したがって、SQLファイルは、実行する前に、コメントを示すダッシュを削除しhrスキーマのパスワードを追加することで編集する必要があります。

Oracle Data Pumpは、ALTER SESSION文を、Oracle Data Pumpインポートにより作成されたSQLファイルの上に配置します。インポート操作に異なる接続文がある場合は、各ALTER SESSION文を手動でコピーして、適切なCONNECT文の後に貼り付ける必要があります。

一部のOracle Databaseオプションでは、無名PL/SQLブロックがSQLFILE出力に含まれていることがあります。これらのPL/SQLブロックは直接実行しないでください。

制限事項

- SQLFILEが指定されている場合、CONTENTパラメータは、ALLまたはDATA_ONLYのいずれかに設定されていると無視されます。

- Oracle Automatic Storage Management (Oracle ASM)を使用してSQLファイルへのOracle Data Pump Importを実行する場合、SQLFILEパラメータでは、Oracle ASM +表記法を使用せずにディレクトリ・オブジェクトを指定する必要があります。つまり、SQLファイルはディスク・ファイルに書き込まれ、Oracle ASMの記憶域には書き込まれません。
- SQLFILEパラメータをQUERYパラメータと組み合せて使用することはできません。

例

次に、SQLFILEパラメータの使用例を示します。この例では、ExportのFULLパラメータで示した例を実行して、expfull.dmpダンプ・ファイルを作成できます。

```
> impdp hr DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp
SQLFILE=dpump_dir2:expfull.sql
```

SQLファイルexpfull.sqlは、dpump_dir2に書き込まれます。

関連トピック

- FULL

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.45 STATUS

Oracle Data Pump Importのコマンドライン・モードのSTATUSパラメータでは、ジョブの状態を表示する頻度を指定します。

デフォルト

0

用途

ジョブの状態を表示する頻度を指定します。

構文および説明

```
STATUS[=integer]
```

integerに値を入力すると、ロギング・モードでジョブの状態を表示する頻度を秒単位で指定できます。値を入力しなかった場合またはデフォルト値の0を使用した場合、各オブジェクト型、表またはパーティションの完了に関する情報のみ表示されます。

この状態情報は、標準出力デバイスのみに書き込まれ、ログ・ファイルには(使用可能な場合でも)書き込まれません。

例

次に、STATUSパラメータの使用例を示します。この例では、エクスポートのFULLパラメータで示した例を実行して、expfull.dmpダンプ・ファイルを作成できます。

```
> impdp hr NOLOGFILE=YES STATUS=120 DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp
```

この例では、状態が2分(120秒)ごとに表示されます。

関連トピック

- FULL

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.46 STREAMS_CONFIGURATION

Oracle Data Pump Importのコマンドライン・モードのSTREAMS_CONFIGURATIONパラメータでは、エクスポート・ダンプ・ファイル内に存在する可能性のあるすべてのGoldenGateレプリケーション・メタデータをインポートするかどうかを指定します。

デフォルト

YES

用途

エクスポート・ダンプ・ファイル内に存在する可能性のあるすべてのGoldenGateレプリケーション・メタデータをインポートするかどうかを指定します。

構文および説明

```
STREAMS_CONFIGURATION=[YES | NO]
```

例

次に、STREAMS_CONFIGURATIONパラメータの使用例を示します。この例では、ExportのFULLパラメータで示した例を実行して、expfull.dmpダンプ・ファイルを作成できます。

```
> impdp hr DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp STREAMS_CONFIGURATION=NO
```

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.47 TABLE_EXISTS_ACTION

Oracle Data Pump Importのコマンドライン・モードのTABLE_EXISTS_ACTIONパラメータでは、インポート・ユーティリティで作成しようとしている表がすでに存在する場合の動作を指定します。

デフォルト

SKIP

ノート:

CONTENT=DATA_ONLY が指定されている場合、デフォルトは SKIP ではなく APPEND です。

用途

インポートで作成しようとしている表がすでに存在する場合の動作を指定します。

構文および説明

```
TABLE_EXISTS_ACTION=[SKIP | APPEND | TRUNCATE | REPLACE]
```

次の値を指定できます。

- SKIP: 表はそのままにして、次のオブジェクトに移動します。CONTENTパラメータがDATA_ONLYに設定されている場合、このオプションは無効です。
- APPEND: ソースから行をロードし、既存の行は変更しません。
- TRUNCATE: 既存の行を削除した後、ソースから行をロードします。
- REPLACE: 既存の表を削除した後、ソースから表を作成およびロードします。CONTENTパラメータがDATA_ONLYに

設定されている場合、このオプションは無効です。

これらのオプションを使用する場合は、次の点に注意してください。

- TRUNCATEまたはREPLACEを使用する場合は、影響を受ける表の行が参照制約のターゲットではないことを確認してください。
- SKIP、APPENDまたはTRUNCATEを使用する場合、索引、権限、トリガー、制約など、ソースの既存の表依存オブジェクトは変更されません。REPLACEを使用すると、依存オブジェクトが明示的または暗黙的に(EXCLUDEを使用して)除外され、それらがソースのダンプ・ファイルまたはシステムに存在する場合、依存オブジェクトは削除され、ソースから再作成されます。
- APPENDまたはTRUNCATEを使用する場合は、操作を実行する前に、ソースの行が既存の表と互換性があるかどうかがインポートによりチェックされます。

  既存の表にアクティブな制約およびトリガーがある場合は、外部表によるアクセス方法を使用してロードされます。アクティブな制約に違反する行がある場合、ロードは失敗し、データはロードされません。この動作を変更するには、インポート・ユーティリティのコマンドラインでDATA_OPTIONS=SKIP_CONSTRAINT_ERRORSを指定します。

  制約違反の可能性があるデータをロードする必要がある場合は、制約を無効にし、データをロードした後、制約を再度有効にする前に問題のある行を削除する方法を検討してください。

- APPENDを使用すると、常に、データは新しい領域にロードされます。既存の領域は、使用可能な場合でも再利用されません。そのため、ロード後にデータを圧縮することもできます。
- パラレル処理を使用する場合は、既存のターゲット表の有無によりパーティション表のパラレル処理がどのように影響を受けるかについて、インポート・ユーティリティのPARTITION_OPTIONSパラメータの説明を参照してください。
- 既存の表(TABLE_EXISTS_ACTION=REPLACEまたはTRUNCATE)にインポートする場合は、表のパーティション化スキームに応じて次のガイドラインに従います。
  - パーティション化スキームがソースとターゲット間で一致する場合は、インポート時にDATA_OPTIONS=TRUST_EXISTING_TABLE_PARTITIONSを使用します。
  - パーティション化スキームがソースとターゲットで異なる場合は、エクスポート時にDATA_OPTIONS=GROUP_PARTITION_TABLE_DATAを使用します。

ノート:

Oracle Data Pump は、ソース表とターゲット表の不一致(2 つの表の列数が同じでないか、ソース表にない列名がターゲット表にある)を検出すると、2 つの表間で列名を比較します。これらの表に共通の列が少なくとも 1 つあると、(データ型に互換性があるとみなされる)共通する列のデータが表にインポートされます。次の制限があります。

- この動作はネットワーク・インポートではサポートされません。
- 次の型の列は削除できません。オブジェクト列、オブジェクト属性、ネストされた表の列、主キーに基づく REF 列。

制限事項

- TRUNCATEは、クラスタ化された表では使用できません。

例

次に、TABLE_EXISTS_ACTIONパラメータの使用例を示します。この例では、ExportのFULLパラメータで示した例を実行

して、expfull.dmpダンプ・ファイルを作成できます。

```
> impdp hr TABLES=employees DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp
TABLE_EXISTS_ACTION=REPLACE
```

関連トピック

- PARTITION_OPTIONS
- FULL

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.48 REUSE_DATAFILES

Oracle Data Pump Importのコマンドライン・モードのREUSE_DATAFILESパラメータでは、インポート・ジョブで表領域の作成に既存のデータ・ファイルを再利用するかどうかを指定します。

デフォルト

NO

用途

インポート・ジョブで、表領域の作成に既存のデータ・ファイルを再利用するかどうかを指定します。

構文および説明

```
REUSE_DATAFILES=[YES | NO]
```

デフォルト(n)を使用して、CREATE TABLESPACE文で指定されているデータ・ファイルがすでに存在する場合は、CREATE TABLESPACE文の失敗によるエラー・メッセージが発行されますが、インポート・ジョブは続行されます。

このパラメータにYを指定すると、既存のデータ・ファイルが再度初期化されます。

> 注意:
>
> REUSE_DATAFILES=YES を指定すると、データが失われる可能性があります。

例

次に、REUSE_DATAFILESパラメータの使用例を示します。この例では、ExportのFULLパラメータで示した例を実行して、expfull.dmpダンプ・ファイルを作成できます。

```
> impdp hr DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp LOGFILE=reuse.log
REUSE_DATAFILES=YES
```

この例では、expfull.dmpファイルのCREATE TABLESPACE 文で指定されたデータ・ファイルを再度初期化します。

関連トピック

- FULL

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.49 TABLES

Oracle Data Pump Importのコマンドライン・モードのTABLESパラメータにより、表モード・インポートを実行することを指定

します。

デフォルト

デフォルトはありません

用途

表モード・インポートの実行を指定します。

構文および説明

```
TABLES=[schema_name.]table_name[:partition_name]
```

表モード・インポートでは、表およびパーティションまたはサブパーティションをカンマで区切ったリストを指定して、ソースからインポートするデータをフィルタ処理できます。

schema_nameを指定しなかった場合は、デフォルトで現在のユーザーのスキーマ名になります。自分のスキーマ以外のスキーマを指定するには、DATAPUMP_IMP_FULL_DATABASEロールを持っているか、またはスキーマを現在のユーザーに再マップする必要があります。

インポートする内容を制限する場合は、このインポート・モードでフィルタ処理を使用できます。

partition_nameを指定する場合は、関連表にあるパーティションまたはサブパーティションの名前にする必要があります。

ワイルドカード文字%を使用して表名とパーティション名を指定できます。

表名には次の制限があります。

- デフォルトでは、表名は大文字でデータベースに格納されます。表名が大文字と小文字または小文字のみで表記され、大/小文字を区別する場合、名前を引用符で囲む必要があります。したがって、表名は、データベースに格納されている表名と完全に一致するように指定する必要があります。

  ただし、オペレーティング・システムによっては、コマンドラインの引用符自体をエスケープする必要がある場合があります。次に、異なるインポート・モードで大/小文字の区別を保持する方法を示します。

  - コマンドライン・モード

    ```
    TABLES='¥"Emp¥"'
    ```

  - パラメータ・ファイル・モード

    ```
    TABLES='"Emp"'
    ```

- 表名を引用符で囲まないと、コマンドラインで指定する表名にポンド記号(#)は使用できません。同様に、パラメータ・ファイルでは、表名がポンド記号(#)を含んでいると、インポート・ユーティリティは、表名が引用符で囲まれていない場合コメントとして解釈します。

  たとえば、パラメータ・ファイルに次のコマンドラインが記述されている場合、インポート・ユーティリティではemp#の右側がすべてコメントとして解釈されるため、表deptおよびmydataはインポートされません。

  ```
  TABLES=(emp#, dept, mydata)
  ```

  ただし、パラメータ・ファイルに次の行が含まれる場合は、emp#が引用符で囲まれているため、インポート・ユーティリティは3つの表をすべてインポートします。

  ```
  TABLES=('"emp#"', dept, mydata)
  ```

ノート:

オペレーティング・システムによっては、一重引用符を使用する必要がある場合と、二重引用符を使用する必要がある場合があります。ご使用のオペレーティング・システムのドキュメントで確認してください。表のネーミング方法に制限があるオペレーティング・システムもあります。

たとえば、Unix の C シェルではドル記号($)やポンド記号(#)またはその他の特定の特殊文字には特別な意味があります。名前に特殊文字を使用する場合は、オペレーティング・システム・シェルによって無視され、インポートで使用できるように、エスケープ文字を使用する必要があります。

制限事項

- TABLESパラメータの値としてのシノニムの使用はサポートされていません。たとえば、hrスキーマのregions表にregnのシノニムが存在する場合、TABLES=regnを使用すると無効になります。この場合、エラーが返されます。
- インポートに対してPARTITION_OPTIONS=DEPARTITION も指定されている場合、1つの表からのパーティションのみを指定できます。
- TRANSPORTABLE=ALWAYSを指定する場合は、TABLESパラメータで指定されるすべてのパーティションが同じ表内に存在する必要があります。
- TABLESパラメータに指定する表名のリストの長さは、最大4MBに制限されます。ただし、NETWORK_LINKパラメータで10.2.0.3以前のOracle Databaseまたは読取り専用のデータベースが設定されている場合は異なります。この場合の上限は4KBです。

例

次に、TABLESパラメータを使用して、expfull.dmpファイルからemployeesおよびjobs表のみをインポートする簡単な例を示します。この例では、ExportのFULLパラメータで示した例を実行して、expfull.dmpダンプ・ファイルを作成できます。

```
> impdp hr DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp TABLES=employees,jobs
```

次に、TABLESを使用してパーティションをインポートするコマンドの例を示します。

```
> impdp hr DIRECTORY=dpump_dir1 DUMPFILE=expdat.dmp
TABLES=sh.sales:sales_Q1_2012,sh.sales:sales_Q2_2012
```

この例では、shスキーマのsales表のパーティションsales_Q1_2012およびsales_Q2_2012をインポートします。

関連トピック

- インポート操作中のフィルタ処理
- FULL

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.50 TABLESPACES

Oracle Data Pump Importのコマンドライン・モードのTABLESPACESパラメータにより、表領域モード・インポートを実行することを指定します。

デフォルト

デフォルトはありません

用途

表領域モード・インポートの実行を指定します。

構文および説明

```
TABLESPACES=tablespace_name [, ...]
```

TABLESPACESを使用して、表と依存オブジェクトがソース(全体インポート・モード、スキーマ・モード、表領域モードまたは表モードのエクスポート・ダンプ・ファイル・セット、あるいは別のデータベース)からインポートされる表領域名のリストを指定します。

インポートの次の状況では、データ・ポンプによりデータのインポート先に自動的に表領域が作成されます。

- インポートがFULLモードまたはTRANSPORT_TABLESPACESモードで実行されている場合
- インポートがTRANSPORTABLE=ALWAYSを含む表モードで実行されている場合

その他のすべての場合では、選択したオブジェクトの表領域がインポート先のデータベースにすでに存在している必要があります。REMAP_TABLESPACEインポート・パラメータを使用して、インポート先のデータベースにある表領域に表領域名をマッピングすることもできます。

インポートする内容を制限する場合は、このインポート・モードでフィルタ処理を使用できます。

制限事項

- TABLESPACESパラメータに指定する表領域名のリストの長さは、最大4MBに制限されます。ただし、NETWORK_LINKパラメータで10.2.0.3以前のデータベースまたは読取り専用のデータベースが設定されている場合は異なります。この場合の上限は4KBです。

例

次に、TABLESPACESパラメータの使用例を示します。表領域はすでに存在するものとします。この例では、ExportのFULLパラメータで示した例を実行して、expfull.dmpダンプ・ファイルを作成できます。

```
> impdp hr DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp
TABLESPACES=tbs_1,tbs_2,tbs_3,tbs_4
```

この例では、表領域tbs_1、tbs_2、tbs_3およびtbs_4にデータがある表がすべてインポートされます。

関連トピック

- インポート操作中のフィルタ処理
- FULL

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.51 TARGET_EDITION

Oracle Data Pump Importのコマンドライン・モードのTARGET_EDITIONパラメータにより、オブジェクトをインポートするデータベース・エディションを指定します。

デフォルト

システム上のデフォルトのデータベース・エディション。

用途

オブジェクトのインポート先のデータベース・エディションを指定します。

構文および説明

```
TARGET_EDITION=name
```

TARGET_EDITION=nameを指定すると、データ・ポンプ・インポートによって、ダンプ・ファイル内で検出されたすべてのオブジェクトが作成されます。エディション化可能でないオブジェクトは、すべてのエディションで作成されます。

たとえば、表はエディション化可能でないため、ダンプ・ファイル内に表があると、その表が作成されてすべてのエディションから参照されます。ダンプ・ファイル内のオブジェクトでエディション化可能なもの(プロシージャなど)は、指定されたターゲット・エディションのみに作成されます。

このパラメータが指定されていない場合は、エクスポート・ジョブでエディションが指定されていた場合でも、インポートでターゲット・データベースのデフォルトのエディションが使用されます。指定されたエディションが存在しない場合や使用できない場合は、エラー・メッセージが返されます。

制限事項

- このパラメータは、バージョニング可能な同じオブジェクトのバージョンがデータベース内に2つ以上ある場合にのみ有効です。
- ジョブ・バージョンは、11.2以上である必要があります。

例

次に、TARGET_EDITIONパラメータの使用例を示します。

```
> impdp hr DIRECTORY=dpump_dir1 DUMPFILE=exp_dat.dmp TARGET_EDITION=exp_edition
```

この例は、オブジェクトのインポート先となるexp_editionというエディションがシステム上に存在することを前提としています。インポート・モードは指定されていないため、デフォルトのスキーマ・モードが使用されます。

エディションの作成方法の詳細は、*Oracle Database SQL*言語リファレンスを参照してください。エディションの機能の詳細は、*Oracle Database*開発ガイドを参照してください。

関連トピック

- VERSION
- 『Oracle Database SQL言語リファレンス』の「CREATE EDITION」
- 『Oracle Database開発者ガイド』の「エディション」

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.52 TRANSFORM

Oracle Data Pump Importのコマンドライン・モードのTRANSFORMパラメータを使用すると、インポート中のオブジェクトに対するオブジェクト作成DDLを変更できます。

デフォルト

デフォルトはありません

用途

インポート中のオブジェクトに対するオブジェクト作成DDLを変更できます。

構文および説明

```
TRANSFORM = transform_name:value[:object_type]
```

transform_nameには、変換の名前を指定します。

object_typeの指定はオプションです。指定した場合、このパラメータでは、変換が適用されるオブジェクト型を指定します。オブジェクト型を指定しなかった場合、変換はすべての有効なオブジェクト型に適用されます。

使用可能な変換は次のとおりです(アルファベット順に記載)。

- CONSTRAINT_NAME_FROM_INDEX: [Y | N]

  この変換は、TABLEおよびCONSTRAINTの各オブジェクト型に対して有効です。

  この変換パラメータは、ユーザー作成索引を参照するpkまたはfkの制約の生成に影響します。Yに設定すると、制約の名前が索引の名前と一致するように強制されます。

  N (デフォルト)に設定すると、ソース・データベースで指定されたとおりに制約が作成されます。

- CONSTRAINT_USE_DEFAULT_INDEX: [Y | N]

  この変換は、TABLEおよびCONSTRAINTの各オブジェクト型に対して有効です。

  この変換パラメータは、pkまたはfkの制約に関連する索引の生成に影響します。変換パラメータがYに設定されている場合、変換により、制約を強制するために自動的に作成される索引の名前が、制約名と同一になります。また、索引は、ターゲット・データベースのデフォルトの制約定義を使用して作成および定義され、ソース・データベースで定義されている可能性がある特別な特性は使用しません。

  デフォルトの索引は、UNIQUE、PRIMARY KEY、FOREIGN KEYなどの標準スキーマ整合性制約を使用しないかぎり許可されません。したがって、制限のないシステムからOracle Data Pumpインポートを実行し、ソース索引に追加の制約(たとえば、ハッシュ・パーティション索引などのユーザー生成制約)がある場合、これらの追加の制約はインポート中に削除されます。

  N (デフォルト)に設定すると、ソース・データベースで指定され定義されているとおりに索引が作成されます。

- DISABLE_ARCHIVE_LOGGING:[Y | N]

  この変換は、INDEXおよびTABLEの各オブジェクト型に対して有効です。

  Yに設定すると、データがインポートされる前に、指定したオブジェクト型(TABLEまたはINDEXあるいはその両方)のロギング属性が無効になります。Nに設定すると(デフォルト)、アーカイブ・ロギングはインポート中に無効になりません。データがロードされた後に、オブジェクトのロギング属性は元の設定に戻されます。オブジェクト型を指定しない場合、DISABLE_ARCHIVE_LOGGINGの動作は、TABLEとINDEXの両方のオブジェクト型に適用されます。この変換は、ファイル・モード・インポートとネットワーク・モード・インポートの両方で動作します。これは、トランスポータブル表領域インポートには適用されません。

  ノート:

  データベースが FORCE LOGGING モードである場合、索引および表の作成時に DISABLE_ARCHIVE_LOGGING オプションではロギングは無効になりません。

- DWCS_CVT_IOTS: [Y | N]

  この変換は、TABLEオブジェクト型に対して有効です。

  Yに設定すると、表の作成時にORGANIZATION INDEX句を抑止して、索引構成表をヒープ構成表に変換するようOracle Data Pumpに指示します。

N (デフォルト)に設定すると、生成されるDDLでは、ソース・オブジェクトの表特性が保持されます。

- DWCS_CVT_CONSTRAINTS: [Y | N]

  この変換は、TABLEおよびCONSTRAINTの各オブジェクト型に対して有効です。

  Yに設定すると、pk、fkまたはukの制約を無効として作成するようOracle Data Pumpに指示します。

  N (デフォルト)に設定すると、pk、fkまたはukの制約をソース・データベースのステータスに基づいて作成するようOracle Data Pumpに指示します。

- INMEMORY:[Y | N]

  この変換は、TABLEおよびTABLESPACEの各オブジェクト型に対して有効です。

  INMEMORY変換は、インメモリー列ストア(IM列ストア)に関連があります。IM列ストアは、表、表パーティションおよびその他のデータベース・オブジェクトのコピーを格納するSystem Global Area(SGA)のオプションの部分です。IM列ストアはSGAの一部であるため、データは行ではなく、列によって移入されます。データは高速スキャンで最適化されます。IM列ストアでは、バッファ・キャッシュは置換しませんが、両方のメモリー領域において同じデータを異なる形式で格納するための補足としての役割を果たします。IM列ストアは、Oracle Databaseインメモリー・オプションに含まれています。

  インポートでY (デフォルト値)が指定されている場合、データ・ポンプは、IM列ストア句を含むすべてのオブジェクトに対してIM列ストア句を保持します。これらのオブジェクトがインポート時に再作成されると、データ・ポンプは、エクスポート時にこれらのオブジェクトの設定と一致するIM列ストア句を生成します。

  インポートでNが指定されている場合、データ・ポンプは、IM列ストア句を含むすべてのオブジェクトからIM列ストア句を削除します。表領域に格納されているオブジェクトにIM列ストア句がない場合、オブジェクトは表領域からIM列ストア句を継承します。したがって、データベースを移行していて、IM列ストア機能を使用する新しいデータベースが必要な場合、適切なIM列ストア句を持つ表領域を事前に作成できます。その後、インポート・コマンドでTRANSFORM=INMEMORY:Nを使用します。オブジェクトは、事前に作成済の新しい表領域からIM列ストア句を継承します。

  INMEMORY変換を使用しない場合、すべてのオブジェクトを個別に変更して、適切なIM列ストア句を追加する必要があります。

  ノート:

  INMEMORY 変換は、Oracle Database 12c リリース 1 (12.1.0.2)以上でのみ使用できます。

  インメモリー列ストア(IM 列ストア)の使用方法の詳細は、*Oracle Database* 管理者ガイドを参照してください。

- INMEMORY_CLAUSE:"string with a valid in-memory parameter"

  この変換は、TABLEおよびTABLESPACEの各オブジェクト型に対して有効です。

  INMEMORY_CLAUSE変換は、インメモリー列ストア(IM列ストア)に関連があります。IM列ストアは、表、表パーティションおよびその他のデータベース・オブジェクトのコピーを格納するSystem Global Area(SGA)のオプションの部分です。IM列ストアはSGAの一部であるため、データは行ではなく、列によって移入されます。データは高速スキャンで最適化されます。IM列ストアでは、バッファ・キャッシュは置換しませんが、両方のメモリー領域において同じデータを異なる形式で格納するための補足としての役割を果たします。IM列ストアは、Oracle Databaseインメモリー・オプションに

含まれています。

この変換を指定すると、データ・ポンプは、DDL内にIM列ストア句が含まれる、インポート対象のすべてのオブジェクトに対して、文字列の内容をINMEMORY_CLAUSEとして使用します。この変換は、ダンプ・ファイルのオブジェクトに対してIM列ストア句を優先させる必要がある場合に有効です。

指定する文字列を二重引用符で囲む必要があります。コマンドラインでコマンドを入力する場合、オペレーティング・システムによってはコマンドの解析中に疑問符が削除され、エラーが発生する可能性があることに注意してください。バックスラッシュ・エスケープ文字(¥)を使用すると、このエラーを回避できます。たとえば:

```
transform=inmemory_clause:¥"INMEMORY MEMCOMPRESS FOR DML PRIORITY CRITICAL¥"
```

または、パラメータ・ファイルにパラメータを配置することもできます。処理中に、パラメータ・ファイル内の引用符は維持されます。

> ノート:
>
> INMEMORY_CLAUSE 変換は、Oracle Database 12c リリース 1 (12.1.0.2)以上でのみ使用できます。
>
> インメモリー列ストア(IM 列ストア)の使用方法の詳細は、*Oracle Database* 管理者ガイドを参照してください。IM 列ストア句で指定できるパラメータのリストおよび詳細は、*Oracle Database* リファレンスを参照してください

- LOB_STORAGE:[SECUREFILE | BASICFILE | DEFAULT | NO_CHANGE]

  この変換は、オブジェクト型TABLEに対して有効です。

  指定した記憶域(SECUREFILEまたはBASICFILE)で、LOBセグメントが作成されます。値がNO_CHANGE (デフォルト)の場合、LOBセグメントは、ソース・データベースに存在するものと同じ記憶域に作成されます。値がDEFAULTの場合、キーワード(SECUREFILEまたはBASICFILE)は省略され、LOBセグメントはデフォルト記憶域に作成されます。

  この変換を指定すると、マテリアライズド・ビューに記憶域を提供する表を含め、ジョブのすべての表のLOB記憶域が変更されます。

  LOB_STORAGEの変換は、トランスポータブル・インポート・ジョブでは無効です。

- OID:[Y | N]

  この変換は、INC_TYPE、TABLEおよびTYPEの各オブジェクト型に対して有効です。

  インポート時にY(デフォルト値)を指定した場合、エクスポートされたOIDが、新しいオブジェクト表およびオブジェクト型に割り当てられます。Oracle Data Pumpでは、ターゲット・データベース上で、一致する既存の型を検出する際にOIDの確認も実行されます。

  インポート時にNを指定した場合、次が実行されます。

  - エクスポートされたOIDは、新しいオブジェクト表とオブジェクト型の作成で割り当てられません。かわりに、新しいOIDが割り当てられます。エクスポートされたOIDを割り当てないようにすると、スキーマのクローニングに有効ですが、参照オブジェクトには影響しません。
  - Oracle Data Pumpでは、型に関連付けられた表にデータをロードする前の、ターゲット・データベース上の

既存の一致する型を検出する際に、標準の型のOID確認がスキップされます。型のハッシュ・コード、バージョン番号、型名によるその他のチェックは実行されます。

- OMIT_ENCRYPTION_CLAUSE: [Y | N]

  この変換は、TABLEオブジェクト型に対して有効です。

  Yに設定することで、列暗号化句を抑止するようOracle Data Pumpに指示できます。ソース・データベースで暗号化された列は、インポートされた表では暗号化されません。

  N (デフォルト)に設定すると、列暗号化句をソース・データベースでのように作成するようOracle Data Pumpに指示します。

- PCTSPACE:some_number_greater_than_zero

  この変換は、CLUSTER、CONSTRAINT、INDEX、ROLLBACK_SEGMENT、TABLEおよびTABLESPACEの各オブジェクト型に対して有効です。

  この変換の値には、0より大きい数字を指定する必要があります。この値は、エクステントの割当てとデータ・ファイル・サイズの変更に使用する、割合の乗数を表します。

  なお、このPCTSPACE変換とデータ・ポンプ・エクスポートのSAMPLEパラメータを組み合せて使用すると、記憶域の割当てサイズを、サンプリングされたデータ・サブセットに合わせることができます。(SAMPLEエクスポート・パラメータを参照してください。)

- SEGMENT_ATTRIBUTES:[Y | N]

  この変換は、CLUSTER、CONSTRAINT、INDEX、ROLLBACK_SEGMENT、TABLEおよびTABLESPACEの各オブジェクト型に対して有効です。

  値をYに指定すると、適切なDDLにセグメント属性(物理属性、記憶域属性、表領域およびロギング)が指定されます。デフォルトはYです。

- SEGMENT_CREATION:[Y | N]

  この変換は、オブジェクト型TABLEに対して有効です。

  Y (デフォルト)に設定した場合、この変換によってSQL SEGMENT CREATION句がCREATE TABLE文に追加されます。つまり、CREATE TABLE文でSEGMENT CREATION DEFERREDかSEGMENT CREATION IMMEDIATEのいずれかが明示的に表現されます。値がNの場合、SEGMENT CREATION句はCREATE TABLE文から省略されます。ロードされる表に対してデフォルトのセグメント作成属性を使用するには、このパラメータをNに設定します。この機能は、Oracle Database 11gリリース2 (11.2.0.2)以降で使用できます。

- STORAGE:[Y | N]

  この変換は、CLUSTER、CONSTRAINT、INDEX、ROLLBACK_SEGMENTおよびTABLEの各オブジェクト型に対して有効です。

  値をYに指定すると、適切なDDLにSTORAGE句が指定されます。デフォルトはYです。SEGMENT_ATTRIBUTES=Nの場合、このパラメータは無視されます。

- TABLE_COMPRESSION_CLAUSE:[NONE | compression_clause]

  この変換は、オブジェクト型TABLEに対して有効です。

  NONEを指定すると、表圧縮句は省略されます(その表の表領域にはデフォルトの圧縮が使用されます)。それ以外の場合、その値が有効な表圧縮句です(NOCOMPRESS、COMPRESS BASICなど)。指定した圧縮を使用して表が作成されます。有効な表圧縮構文の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

表圧縮句が複数の語の場合、一重引用符または二重引用符で囲む必要があります。また、オペレーティング・システムでは、句をエスケープ文字(バックスラッシュ文字など)で囲むことが必要な場合があります。たとえば:

```
TRANSFORM=TABLE_COMPRESSION_CLAUSE:¥"COLUMN STORE COMPRESS FOR QUERY HIGH¥"
```

この変換を指定すると、マテリアライズド・ビューに記憶域を提供する表を含め、ジョブのすべての表の圧縮タイプが変更されます。

例

次の例では、hrスキーマのemployees表をエクスポートしたとします。表をインポートした結果返されるSQL CREATE TABLE文は、次のようになります。

```
CREATE TABLE "HR"."EMPLOYEES"
   ( "EMPLOYEE_ID" NUMBER(6,0),
     "FIRST_NAME" VARCHAR2(20),
     "LAST_NAME" VARCHAR2(25) CONSTRAINT "EMP_LAST_NAME_NN" NOT NULL ENABLE,
     "EMAIL" VARCHAR2(25) CONSTRAINT "EMP_EMAIL_NN" NOT NULL ENABLE,
     "PHONE_NUMBER" VARCHAR2(20),
     "HIRE_DATE" DATE CONSTRAINT "EMP_HIRE_DATE_NN" NOT NULL ENABLE,
     "JOB_ID" VARCHAR2(10) CONSTRAINT "EMP_JOB_NN" NOT NULL ENABLE,
     "SALARY" NUMBER(8,2),
     "COMMISSION_PCT" NUMBER(2,2),
     "MANAGER_ID" NUMBER(6,0),
     "DEPARTMENT_ID" NUMBER(4,0)
   ) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS LOGGING
  STORAGE(INITIAL 10240 NEXT 16384 MINEXTENTS 1 MAXEXTENTS 121
  PCTINCREASE 50 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
  TABLESPACE "SYSTEM" ;
```

STORAGE句またはTABLESPACE句は、保持しない場合、インポート・ユーティリティのTRANSFORMパラメータを使用してCREATE STATEMENTから削除できます。SEGMENT_ATTRIBUTESの値をNに指定します。これによって、セグメント属性(記憶域と表領域の両方)が表から除外されます。

```
> impdp hr TABLES=hr.employees DIRECTORY=dpump_dir1 DUMPFILE=hr_emp.dmp
  TRANSFORM=SEGMENT_ATTRIBUTES:N:table
```

この結果返される、employees表のCREATE TABLE文は次のようになります。STORAGEまたはTABLESPACE句は含まれていません。かわりに、HRスキーマのデフォルト表領域が使用されます。

```
CREATE TABLE "HR"."EMPLOYEES"
   ( "EMPLOYEE_ID" NUMBER(6,0),
     "FIRST_NAME" VARCHAR2(20),
     "LAST_NAME" VARCHAR2(25) CONSTRAINT "EMP_LAST_NAME_NN" NOT NULL ENABLE,
     "EMAIL" VARCHAR2(25) CONSTRAINT "EMP_EMAIL_NN" NOT NULL ENABLE,
     "PHONE_NUMBER" VARCHAR2(20),
     "HIRE_DATE" DATE CONSTRAINT "EMP_HIRE_DATE_NN" NOT NULL ENABLE,
     "JOB_ID" VARCHAR2(10) CONSTRAINT "EMP_JOB_NN" NOT NULL ENABLE,
     "SALARY" NUMBER(8,2),
     "COMMISSION_PCT" NUMBER(2,2),
     "MANAGER_ID" NUMBER(6,0),
     "DEPARTMENT_ID" NUMBER(4,0)
   );
```

前述の例で示したとおり、SEGMENT_ATTRIBUTES変換は、記憶域と表領域の両方の属性に適用されます。STORAGE句のみを省略して、TABLESPACE句を保持する場合は、STORAGE変換を次のように使用できます。

```
> impdp hr TABLES=hr.employees DIRECTORY=dpump_dir1 DUMPFILE=hr_emp.dmp
  TRANSFORM=STORAGE:N:table
```

SEGMENT_ATTRIBUTESおよびSTORAGE変換は、次のコマンドに示すとおり、TRANSFORMパラメータにオブジェクト型を指

定しないことによって、すべての適用可能な表オブジェクトおよび索引オブジェクトに適用できます。

```
> impdp hr DIRECTORY=dpump_dir1 DUMPFILE=hr.dmp SCHEMAS=hr
TRANSFORM=SEGMENT_ATTRIBUTES:N
```

関連トピック

- 『Oracle Database管理者ガイド』の「CREATE INDEX」
- 『Oracle Databaseデータ・ウェアハウス・ガイド』の「インメモリー列ストアを使用した分析の改善」
- SAMPLE
- 『Oracle Database SQL言語リファレンス』の「CREATE TABLE」

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.53 TRANSPORT_DATAFILES

Oracle Data Pump Importのコマンドライン・モードのTRANSPORT_DATAFILESパラメータでは、エクスポート中にTRANSPORTABLE=ALWAYSが設定された場合にターゲット・データベースにインポートするデータ・ファイルのリストを指定します。

デフォルト

デフォルトはありません

用途

トランスポータブル表領域モード・インポートか、エクスポート中にTRANSPORTABLE=ALWAYS が設定された場合には表モード・インポートまたは全体モード・インポートによって、ターゲット・データベースにインポートするデータ・ファイルのリストを指定します。データ・ファイルはターゲット・データベース・システム上にすでに存在する必要があります。

構文および説明

```
TRANSPORT_DATAFILES=datafile_name
```

datafile_nameには、ディレクトリ・オブジェクト名ではなく、ターゲット・データベースが存在するシステムで有効な絶対ディレクトリ・パスを指定する必要があります。

また、datafile_nameの絶対パス指定のファイル名部分にワイルドカードを使用することもできます。アスタリスク(*)は、0からN個の文字と一致します。疑問符(?)は、正確に1文字と一致します。絶対パス指定のディレクトリ部分にワイルドカードを使用することはできません。ワイルドカードを使用する場合、一致するすべてのファイルがトランスポート・セットの一部である必要があります。トランスポート・セットの一部ではないファイルが見つかった場合、エラーが表示され、インポート・ジョブは終了します。

インポート操作前の任意の時点で、ソース・システムからターゲット・システムにデータ・ファイルをコピーしておく必要があります。データ・ファイルは、オペレーティング・システムでサポートされている任意のコピー方法でコピーできます。必要に応じて、ターゲット・システムにコピーするときに、ファイルの名前を変更できます。例2を参照してください。

トランスポータブル・モード・エクスポートによって生成されたダンプ・ファイル・セットがすでにある場合、(メタデータを含んでいる)ダンプ・ファイルおよびTRANSPORT_DATAFILESパラメータを指定することで、そのダンプ・ファイルのトランスポータブル・モード・インポートを実行できます。TRANSPORT_DATAFILESパラメータが存在することで、これがトランスポータブル・モード・インポートであることと、実際のデータを取得する場所がインポートに通知されます。

オペレーティング・システムによっては、このパラメータの値を引用符を使用して指定する場合、エスケープ文字を使用することも必要になる可能性があります。このパラメータはパラメータ・ファイルで指定することをお薦めします。パラメータ・ファイルを使用すると、コマンドラインでは使用する必要があるエスケープ文字の数を減らすことができます。

制限事項

- TRANSPORT_DATAFILESパラメータをQUERYパラメータと組み合せて使用することはできません。
- トランスポータブル・インポート・ジョブは再起動できません。
- 絶対ファイル・パスのTRANSPORT_DATAFILESディレクトリ部分にワイルドカードを含めることはできません。ただし、絶対ファイル・パスのファイル名部分にはワイルドカードを含めることができます。

例1

次に、TRANSPORT_DATAFILESパラメータの使用例を示します。次の内容のパラメータ・ファイルtrans_datafiles.parを作成したとします。

```
DIRECTORY=dpump_dir1
DUMPFILE=tts.dmp
TRANSPORT_DATAFILES='/user01/data/tbs1.dbf'
```

次のコマンドを発行します。

```
> impdp hr PARFILE=trans_datafiles.par
```

例2

この例は、トランスポータブル表領域のエクスポートおよびインポート操作の一環として、データ・ファイルの名前を変更する操作を示しています。ソース・システムにemployees.datというデータ・ファイルがあることを前提としています。

1. オペレーティング・システムでサポートされている方法で、ソース・システムからターゲット・データベースがあるシステムに、employees.datというデータ・ファイルを手動でコピーします。コピー操作の一環として、ファイル名をworkers.datに変更します。
2. 表領域tbs_1について、トランスポータブル表領域のエクスポートを実行します。

   ```
   > expdp hr DIRECTORY=dpump_dir1 DUMPFILE=tts.dmp TRANSPORT_TABLESPACES=tbs_1
   ```

   tbs_1のメタデータのみ(データなし)がtts.dmpというダンプ・ファイルにエクスポートされます。実際のデータは、ステップ1でターゲット・データベースにコピーされています。
3. workers.datというデータ・ファイルの絶対ディレクトリ・パスを指定して、トランスポータブル表領域のインポートを実行します。

   ```
   > impdp hr DIRECTORY=dpump_dir1 DUMPFILE=tts.dmp
   TRANSPORT_DATAFILES='/user01/data/workers.dat'
   ```

   tts.dmpに含まれているメタデータがインポートされ、workers.datファイル内の情報がOracle Data Pumpによってデータベース内の正しい場所に割り当てられます。

例3

この例では、TRANSPORT_DATAFILESパラメータで使用するファイル名に、アスタリスク(*)のワイルドカード文字を使用しています。

```
TRANSPORT_DATAFILES='/db1/hrdata/payroll/emp*.dbf'
```

このパラメータを使用すると、ディレクトリ/db1/hrdata/payroll/内にある、タイプ.dbfのファイルのうち、名前の先頭がempであるものがすべてトランスポート・セットの一部になっていることが、Oracle Data Pumpによって検査されます。

例4

この例では、TRANSPORT_DATAFILESパラメータで使用するファイル名に、疑問符(?)のワイルドカード文字を使用していま

す。

```
TRANSPORT_DATAFILES='/db1/hrdata/payroll/m?emp.dbf'
```

このパラメータを使用すると、ディレクトリ/db1/hrdata/payroll/内にある、タイプ.dbfのファイルのうち、名前の先頭がm、その次が任意の他の単一文字、末尾がempであるものがすべてトランスポート・セットの一部になっていることが、Oracle Data Pumpによって検査されます。たとえば、myemp.dbfという名前のファイルは含められますが、memp.dbfという名前のファイルは含められません。

関連トピック

- インポート・コマンドライン・モードについて

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.54 TRANSPORT_FULL_CHECK

Oracle Data Pump Importのコマンドライン・モードのTRANSPORT_FULL_CHECKパラメータでは、指定したトランスポータブル表領域セットが他の表領域内のオブジェクトによって参照されていることを確認するかどうかを指定します。

デフォルト

NO

用途

指定したトランスポータブル表領域セットが他の表領域内のオブジェクトによって参照されていることを確認するかどうかを指定します。

構文および説明

```
TRANSPORT_FULL_CHECK=[YES | NO]
```

TRANSPORT_FULL_CHECK=YESを指定すると、インポート・ユーティリティによって、トランスポータブル・セット内のオブジェクトとトランスポータブル・セット外のオブジェクトの間に依存性が存在しないことが確認されます。ここでは、双方向の依存性がチェックされます。たとえば、トランスポータブル・セット内に表は存在するが、その表の索引は存在しない場合は、エラーが返され、インポート操作が終了します。同様に、トランスポータブル・セット内に索引は存在するが表は存在しない場合も、エラーが返されます。

TRANSPORT_FULL_CHECK=NOを指定すると、インポート・ユーティリティによって、トランスポータブル・セット外のオブジェクトに依存するオブジェクトが、トランスポータブル・セット内に存在しないことのみ確認されます。ここでは、一方向の依存性がチェックされます。たとえば、表は索引に依存しませんが、索引は表に依存します。これは、索引は表なしでは意味を持たないためです。そのため、トランスポータブル・セット内に表は存在するが、表の索引は存在しない場合、このチェックは正常に終了します。ただし、トランスポータブル・セット内に索引は存在するが表は存在しない場合は、インポート操作が終了します。

このチェックに加えて、インポートでは、常に、TRANSPORT_TABLESPACESで指定された表領域セット内に定義されているすべての表(およびその索引)のすべての記憶域セグメントが、表領域セット内に実際に含まれていることが確認されます。

制限事項

- このパラメータは、NETWORK_LINKパラメータを指定した場合にのみトランスポータブル・モード(またはTRANSPORTABLE=ALWAYSがエクスポート時に指定されていた場合は表モードまたは全体モード)に対して有効です。

例

次の例では、source_database_linkを有効なデータベース・リンクの名前に置き換えます。また、この例では、tbs6.dbfというデータ・ファイルがすでに存在するものとします。

次の内容のパラメータ・ファイルfull_check.parを作成したとします。

```
DIRECTORY=dpump_dir1
TRANSPORT_TABLESPACES=tbs_6
NETWORK_LINK=source_database_link
TRANSPORT_FULL_CHECK=YES
TRANSPORT_DATAFILES='/wkdir/data/tbs6.dbf'
```

次のコマンドを発行します。

```
> impdp hr PARFILE=full_check.par
```

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.55 TRANSPORT_TABLESPACES

Oracle Data Pump Importのコマンドライン・モードのTRANSPORT_TABLESPACESパラメータにより、データベース・リンク経由でトランスポータブル表領域モードでのインポートを実行することを指定します。

デフォルト

デフォルトはありません

用途

データベース・リンク(NETWORK_LINKパラメータで指定)経由でトランスポータブル表領域モードのインポートの実行を指定します。

構文および説明

```
TRANSPORT_TABLESPACES=tablespace_name [, ...]
```

TRANSPORT_TABLESPACESパラメータは、ソース・データベースからターゲット・データベースにオブジェクト・メタデータがインポートされる表領域の名前のリストを指定するために使用します。

このインポートはトランスポータブル・モード・インポートであるため、データのインポート先の表領域は、データ・ポンプによって自動的に作成されます。事前に表領域を作成しておく必要はありません。ただし、インポートを開始する前に、データ・ファイルをターゲット・データベースにコピーします。

インポート・コマンドラインでTRANSPORT_TABLESPACESを指定する場合は、データベース・リンクを指定するためにNETWORK_LINKパラメータも使用する必要があります。データベース・リンクは2つの物理データベース・サーバー間の接続であり、データベース・リンクを使用すると、クライアントから1つの論理データベースとしてこれらのサーバーにアクセスできます。オブジェクト・メタデータはデータベース・リンクを使用してソース(NETWORK_LINKによって指定されるデータベース)からエクスポートされ、ターゲット(impdpコマンドの発行元のデータベース)に直接インポートされるため、NETWORK_LINKパラメータが必要となります。この処理にはダンプ・ファイルは関連しません。他の方法を使用して個別の操作で実際のデータをターゲットにコピーする場合は、TRANSPORT_DATAFILESパラメータを指定し、データの場所を指定します。

ノート:

トランスポータブル表領域モード・エクスポートによって生成されたダンプ・ファイル・セットがすでにある場合、そのダンプ・ファイルのトランスポータブル・モード・インポートを実行できますが、この場合 TRANSPORT_TABLESPACES または

NETWORK_LINK を指定しません。指定するとエラーが発生します。かわりに、(メタデータを含んでいる)ダンプ・ファイルと TRANSPORT_DATAFILES パラメータを指定します。TRANSPORT_DATAFILES パラメータが存在することで、これがトランスポータブル・モード・インポートであることと、実際のデータを取得する場所がインポートに通知されます。

トランスポータブル・ジョブを実行する場合、ベスト・プラクティスは、インポート・ジョブがターゲット・システムで正常に完了するまで、ソース・システムにデータ・ファイルのコピーを保持することです。インポート・ジョブが失敗しても、破損していないデータ・ファイルのコピーが残ります。

制限事項

- トランスポータブル表領域をエクスポートした後、それよりも古いリリース・レベルのデータベースにインポートすることはできません。インポート先のターゲット・データベースのリリース・レベルは、ソース・データベース以上である必要があります。
- TRANSPORT_TABLESPACESパラメータは、NETWORK_LINKパラメータも指定されている場合のみ有効です。
- TRANSPORT_TABLESPACESパラメータを使用してトランスポータブル表領域インポートを実行するには、COMPATIBLE初期化パラメータを11.0.0以上に設定する必要があります。
- オペレーティング・システムによっては、このパラメータの値を引用符を使用して指定する場合、エスケープ文字を使用することも必要になる可能性があります。このパラメータはパラメータ・ファイルに配置することをお薦めします。パラメータ・ファイルを使用すると、コマンドラインの場合には使用する必要があるエスケープ文字の数を減らすことができます。
- トランスポータブル表領域のジョブでは、データ・ポンプ・インポートのACCESS_METHODパラメータはサポートされません。

例

次の例では、source_database_linkを有効なデータベース・リンクの名前に置き換えます。また、この例では、tbs6.dbfというデータ・ファイルが、ソース・データベースからローカル・システムにすでにコピーされているものとします。次の内容のパラメータ・ファイルtablespaces.parを作成したとします。

```
DIRECTORY=dpump_dir1
NETWORK_LINK=source_database_link
TRANSPORT_TABLESPACES=tbs_6
TRANSPORT_FULL_CHECK=NO
TRANSPORT_DATAFILES='user01/data/tbs6.dbf'
```

次のコマンドを発行します。

```
> impdp hr PARFILE=tablespaces.par
```

関連トピック

- 『Oracle Database管理者ガイド』の「データベース・リンク」
- データ・ファイル・コピーを使用したデータ移動
- Oracle Data Pumpによるタイムスタンプ・データの処理方法
- インポート・コマンドライン・モードについて

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.56 TRANSPORTABLE

Oracle Data Pump Importのコマンドライン・モードのTRANSPORTABLEパラメータにより、表モード・インポートまたは全体モード・インポートの際にトランスポータブル・オプションを使用するかどうかを指定します。

デフォルト

NEVER

用途

トランスポータブル・オプションを使用する必要があるかどうかは、表モード・インポート(TABLESパラメータで指定)または全体モード・インポート(FULLパラメータで指定)の際に指定します。KEEP_READ_ONLYおよびNO_BITMAP_REBUILDは、TRANSPORTABLEパラメータ値として追加されます。KEEP_READ_ONLYおよびNO_BITMAP_REBUILDは、トランスポータブル・モードのインポート操作について有効となります。

構文および説明

```
TRANSPORTABLE = [NEVER|ALWAYS|KEEP_READ_ONLY|NO_BITMAP_REBUILD]
```

使用可能な値の定義は、次のとおりです。

- ALWAYS: トランスポータブル・オプションを使用するようにインポート・ジョブに指示します。トランスポータブルが使用できない場合、ジョブは失敗します。

  表モード・インポートでトランスポータブル・オプションを使用すると、指定した表、パーティションまたはサブパーティションのメタデータのみがインポートされるトランスポータブル表領域インポートになります。

  全体モード・インポートでトランスポータブル・オプションを使用すると、指定したデータベースのすべてのオブジェクトのメタデータがインポートされる全体トランスポータブル・インポートになります。

  どちらの場合も、別々の操作でターゲット・データベースに実際のデータ・ファイルをコピー(および通常は変換)する必要があります。

  トランスポータブル・ジョブを実行する場合、ベスト・プラクティスは、インポート・ジョブがターゲット・システムで正常に完了するまで、ソース・システムにデータ・ファイルのコピーを保持することです。インポート・ジョブがなんらかの理由で失敗しても、破損していないデータ・ファイルのコピーが残ります。

- NEVER: トランスポータブル・オプションではなく、ダイレクト・パス方式または外部表方式を使用してデータをロードするようにインポート・ジョブに指示します。これはデフォルトです。

  表のパーティションのサブセットのみをインポートし、TRANSPORTABLE=ALWAYSパラメータを使用すると、それぞれのパーティションが非パーティション表になります。

  表のパーティションのサブセットのみをインポートし、TRANSPORTABLEパラメータを使用しない場合や、そのパラメータがNEVERに設定されている場合(デフォルト)は、次のオプションが適用されます。

  - PARTITION_OPTIONS=DEPARTITION: 各パーティションを非パーティション表として作成することを指定します。
  - PARTITION_OPTIONS: 値が指定されていない場合は、この設定により完全な表の作成が可能になります。つまり、完全な表のすべてのメタデータが存在するようになり、ターゲット・システムでの表定義はソースに存在していたときと同じになります。ただし、その表には指定したパーティションのデータのみが挿入されます。

- KEEP_READ_ONLY: トランスポータブル・データ・ファイルがread-onlyモードを維持する必要があることを指示します。KEEP_READ_ONLYを使用すると、読取り専用アクセスのためにデータ・ファイルを1つ以上の追加の互換データベースにプラグインできるようになります。

- NO_BITMAP_REBUILD: トランスポータブル・データ・ファイル・ヘッダーのビットマップを再構築する必要がないことを指示します。未使用のデータ・セグメントを再利用しないと、インポート操作の時間が短縮されます。ビットマップは、

dbms_space_admin.tablespace_rebuild_bitmapsを使用することで再構築できます。

- TRANSPORTABLE=KEEP_READ_ONLY: 指定すると、データ・ファイルはread-onlyモードを維持します。この設定の結果として、TSTZ列のデータを格納している表は更新できなくなり、インポートから削除されます。また、データ・ファイル・ビットマップを再構築することはできません。
- TRANSPORTABLE=NO_BITMAP_REBUILD: 指定すると、インポート時にTSTZ列のデータを格納している表の更新が必要になったときに、データ・ファイルはread/writeモードに移行します。

APIまたはクラス

TRANSPORTABLEパラメータの値は、既存のプロシージャDBMS_DATAPUMP.SET_PARAMETERを使用して設定できます。

制限事項

- インポートのTRANSPORTABLEパラメータは、NETWORK_LINKパラメータも指定されている場合のみ有効です。
- TRANSPORTABLEパラメータは、表モード・インポートおよび全体モード・インポートでのみ有効です。
- トランスポータブル・インポートを実行するユーザーには、ソース・データベースでのDATAPUMP_EXP_FULL_DATABASEロールと、ターゲット・データベースでのDATAPUMP_IMP_FULL_DATABASEロールの両方が必要になります。
- ネットワーク・インポートのために選択された記憶域を持つすべてのオブジェクトでは、ソース・システム上のすべての記憶域セグメントが、管理対象の非トランスポータブル表領域(SYSTEM / SYSAUX),内に全体として存在するか、ユーザー定義のトランスポータブル表領域内に全体として存在する必要があります。シングル・オブジェクトの記憶域は、2種類の表領域にまたがることはできません。
- ソース・データベースがリリース11.2.0.3の場合にTRANSPORTABLEパラメータを使用してネットワーク・ベースの全体トランスポータブル・インポートを実行するには、データ・ポンプのVERSIONパラメータを12.0以上に設定する必要があります。ソース・データベースがリリース12.1以上の場合、VERSIONパラメータは不要ですが、COMPATIBLEデータベース初期化パラメータを12.0.0以上に設定する必要があります。

ネットワーク・リンク・インポートの例

次に、ネットワーク・リンク・インポート中にTRANSPORTABLEパラメータを使用する例を示します。datafile_nameは、インポート対象のデータ・ファイルです。

```
> impdp system TABLES=hr.sales TRANSPORTABLE=ALWAYS
  DIRECTORY=dpump_dir1 NETWORK_LINK=dbs1 PARTITION_OPTIONS=DEPARTITION
  TRANSPORT_DATAFILES=datafile_name
```

全体トランスポータブル・インポートの例

次の例は、データベース・リンクdbs1を通じて全体トランスポータブル・インポートを実行する際のTRANSPORTABLEパラメータの使用方法を示しています。この例では、暗号化された列を含む表のパスワードを指定しています。

```
> impdp import_admin FULL=Y TRANSPORTABLE=ALWAYS VERSION=12 NETWORK_LINK=dbs1
  ENCRYPTION_PASSWORD=password TRANSPORT_DATAFILES=datafile_name
  LOGFILE=dpump_dir1:fullnet.log
```

NEVERまたはALWAYSの設定例

文字列値を使用したTRANSPORTABLEパラメータの設定は、NEVER値またはALWAYS値に制限されます。

```
SYS.DBMS_DATAPUMP.SET_PARAMETER(jobhdl, 'TRANSPORTABLE','ALWAYS');
SYS.DBMS_DATAPUMP.SET_PARAMETER(jobhdl, 'TRANSPORTABLE','NEVER');
```

新しいTRANSPORTABLEパラメータ・オプションは、新しい数値ビットマスク値を使用して設定します。

DBMS_DATAPUMP.KU$_TTS_NEVERは値1

DBMS_DATAPUMP.KU$_TTS_ALWAYSは値2

DBMS_DATAPUMP.KU$_TTS_KEEP_READ_ONLYは値4

DBMS_DATAPUMP.KU$_TTS_NO_BITMAP_REBUILDは値8

```
SYS.DBMS_DATAPUMP.SET_PARAMETER(jobhdl, 'TRANSPORTABLE',
DBMS_DATAPUMP.KU$_TTS_ALWAYS+DBMS_DATAPUMP.KU$_TTS_KEEP_READ_ONLY);
```

ファイル・ベースのトランスポータブル表領域インポートの例

次の例は、ファイル・ベースのトランスポータブル表領域インポートでのTRANSPORTABLEパラメータの使用を示しています。指定されたKEEP_READ_ONLYオプションは、インポート操作全体にわたってデータ・ファイルが読取り専用アクセスを保持することを示します。必要なデータ・ファイルは、トランスポータブル表領域エクスポートによって報告されます。

```
impdp system DIRECTORY=dpump_dir DUMPFILE=dumpfile_name
TRANSPORT_DATAFILES=datafile_name TRANSPORTABLE=KEEP_READ_ONLY
```

関連トピック

- インポート・コマンドライン・モードについて
- データ・ファイル・コピーを使用したデータ移動

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.57 VERSION

Oracle Data Pump Importのコマンドライン・モードのVERSIONパラメータでは、インポートするデータベース・オブジェクトのバージョンを指定します。

デフォルト

通常はインポート操作でVERSIONパラメータを指定する必要はありません。Oracle Data Pumpでは、次のうちの古い方が使用されます。

- ネットワーク・インポートの場合、ダンプ・ファイルまたはソース・データベースに関連付けられているバージョン
- ターゲット・データベースのCOMPATIBLE初期化パラメータで指定されたバージョン

用途

インポートするデータベース・オブジェクトのバージョンを指定します(つまり、指定されたリリースと互換性のあるデータベース・オブジェクトおよび属性のみがインポートされます)。なお、これは、10.1より前のリリースのOracle DatabaseでOracle Data Pump Importを使用可能ということではありません。Oracle Data Pump Importは、Oracle Database 10gリリース1(10.1)以降でのみ動作します。VERSIONパラメータを使用して可能になるのは、インポートするオブジェクトのバージョンの識別のみです。

構文および説明

```
VERSION=[COMPATIBLE | LATEST | version_string]
```

このパラメータは、ソース・システムのリリースより古い互換リリースのOracle Databaseが稼働しているターゲット・システムのロードに使用できます。VERSIONパラメータが設定されている場合、指定したリリースと互換性のないソース・システム上のデー

タベース・オブジェクトまたは属性はターゲットに移動されません。たとえば、指定したリリースではサポートされていない新しいデータ型を含む表はインポートされません。このパラメータの有効な値は次のとおりです。

- COMPATIBLE - デフォルト値。メタデータのバージョンは、データベースの互換性レベルに対応します。データベースの互換性は、9.2.0以上に設定する必要があります。
- LATEST - メタデータのバージョンは、データベースのリリースに対応します。ターゲット・データベースの実際のバージョンがCOMPATIBLE初期化パラメータで指定されたバージョンより新しいである場合、インポート・ジョブでVERSION=LATESTを指定しても影響がありません。
- version_string - 特定のデータベース・リリース(12.2.0など)。

制限事項

- Oracle Data PumpのVERSIONパラメータを12.1より前の任意の値に指定すると、4000バイトを超える長さのVARCHAR2列またはNVARCHAR2列と、2000バイトを超える長さのRAW列を含む表は、Oracle Data Pumpのダンプ・ファイルから除外されます。
- ターゲットがOracle Database 12cリリース1 (12.1.0.1)以降で、ソースがOracle Database 11gリリース2 (11.2.0.3)以降である場合、ネットワーク・リンクで実行される全体インポートでは、VERSION=12を設定する必要があります。
- Oracle Data PumpのパラメータVERSION=12を使用してOracle Database 11gリリースで作成したダンプ・ファイルは、Oracle Database 12cリリース1 (12.1)以降でのみインポートできます。
- VERSIONパラメータの値がインポートに及ぼす影響は、データ・バインドされた照合(DBC)が有効になっているかどうかで異なります。

例

次の例では、ターゲット・データベースをOracle Database 12cリリース1 (12.1.0.1)、ソース・データベースをOracle Database 11gリリース2 (11.2.0.3)と想定しています。ここでは、ネットワーク・ベース・インポートにVERSION=12を設定する必要があります。また、NETWORK_LINKパラメータを使用している場合、デフォルト・インポート・モードが全体モードであっても、コマンドラインでこれを指定する必要があります。

```
> impdp hr FULL=Y DIRECTORY=dpump_dir1
  NETWORK_LINK=source_database_link VERSION=12
```

関連トピック

- データ・バインドされた照合に関連するOracle Data Pumpの動作
- 異なるOracle Databaseリリース間のエクスポートとインポート

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.4.58 VIEWS_AS_TABLES (ネットワーク・インポート)

Oracle Data Pump Importのコマンドライン・モードのVIEWS_AS_TABLES (ネットワーク・インポート)パラメータでは、1つ以上のビューを表としてインポートすることを指定します。

デフォルト

デフォルトはありません

ノート:

この VIEWS_AS_TABLES の説明は、ネットワーク・インポート時に適用されます(データ・ポンプ・インポートの NETWORK_LINK パラメータの値を指定する場合)。

用途

1つ以上のビューを表としてインポートするように指定します。

構文および説明

```
VIEWS_AS_TABLES=[schema_name.]view_name[:table_name], ...
```

Oracle Data Pumpによって、ビューと同じ列を含み、ビューからフェッチされた行データを含む表がインポートされます。Oracle Data Pumpでは、権限や制約などのビューに依存するオブジェクトもインポートされます。表に適用されない依存オブジェクト(UNDERオブジェクト権限の付与など)はインポートされません。VIEWS_AS_TABLESパラメータは、単独で使用することも、TABLESパラメータと組み合せて使用することもできます。どちらを使用しても、Oracle Data Pumpでは表モード・インポートが実行されます。

構文要素の定義は次のとおりです。

schema_name: ビューが存在するスキーマの名前。スキーマ名を指定しない場合、デフォルトで、インポートを実行するユーザーになります。

view_name: 表としてインポートするビューの名前。ビューは存在し、LOB以外のスカラー列のみを含むリレーショナル・ビューである必要があります。無効なビューや存在しないビューを指定すると、そのビューはスキップされてエラー・メッセージが返されます。

table_name: インポートされるビューのメタデータのソースとなる表の名前。デフォルトでは、Oracle Data Pumpによって、ビューと同じ列およびデータ型を持つ、行の含まれない一時的なテンプレート表が自動的に作成されます。データベースが読取り専用の場合、このテンプレート表のデフォルトの作成には失敗します。この場合、表名を指定できます。表は、ビューと同じスキーマに属している必要があります。これは、ヒープ構成のパーティション化されていないリレーショナル表である必要があります。ネストした表にすることはできません。

テンプレート表が明示的に指定された複数のビューがインポート・ジョブに含まれる場合、それらのテンプレート表はすべて異なっている必要があります。たとえば、2つのビューで同じテンプレート表を使用している次のジョブでは、ビューの1つがスキップされます。

```
impdp hr DIRECTORY=dpump_dir NETWORK_LINK=dblink1
VIEWS_AS_TABLES=v1:employees,v2:employees
```

オブジェクトの省略をレポートするエラー・メッセージが返されます。

テンプレート表は、インポート操作の完了後に自動的に削除されます。それらが存在している間は、次の問合せを実行してその名前を表示できます(名前はすべてKU$VATで始まります)。

```
SQL> SELECT * FROM user_tab_comments WHERE table_name LIKE 'KU$VAT%';
TABLE_NAME                     TABLE_TYPE
------------------------------ -----------
COMMENTS
-------------------------------------------------------
KU$VAT_63629                   TABLE
Data Pump metadata template table for view HR.EMPLOYEESV
```

制限事項

- VIEWS_AS_TABLESパラメータは、TRANSPORTABLE=ALWAYSパラメータと組み合せて使用することはできません。

- VIEWS_AS_TABLESパラメータを使用して作成された表には、指定したビューの一部であった非表示列は含まれません。
- VIEWS_AS_TABLESパラメータでは、LONGのデータ型の列のある表をサポートしていません。

例

次の例では、ネットワーク・インポートを実行して、読取り専用データベースからビューhr.v1のコンテンツをインポートします。ソース・データベースのhrスキーマには、ビューview1と同じジオメトリを持つテンプレート表が含まれる必要があります(この表はview1_tabと呼ばれます)。VIEWS_AS_TABLESパラメータには、ビュー名と表名をコロンで区切ってリストします。

```
> impdp hr VIEWS_AS_TABLES=view1:view1_tab NETWORK_LINK=dblink1
```

ビューは、ビューからフェッチされた行を持つview1という表としてインポートされます。表のメタデータは、テンプレート表のview1_tabからコピーされます。

親トピック: Oracle Data Pump Importコマンドライン・モードで使用可能なパラメータ

3.5 Oracle Data Pump Import対話方式コマンド・モードで使用可能なコマンド

対話方式コマンド・モードでは、現行のジョブは継続して続行されますが、端末へのロギングは一時停止され、インポート・プロンプト(Import>)が表示されます。

- Oracle Data Pump Importの対話型コマンド・モードについて
  接続されているクライアント、またはジョブが実行されている端末以外からOracle Data Pumpコマンドを実行する方法について説明します。
- CONTINUE_CLIENT
  Oracle Data Pumpインポートの対話方式コマンド・モードのCONTINUE_CLIENTパラメータでは、対話方式コマンド・モードからロギング・モードに変更します。
- EXIT_CLIENT
  Oracle Data Pumpインポートの対話方式コマンド・モードのEXIT_CLIENTパラメータでは、インポート・クライアント・セッションを停止し、インポートを終了して、端末へのロギングを中断します。ただし、現行のジョブの実行は続行します。
- HELP
  Oracle Data Pumpインポートの対話型コマンド・モードのHELPパラメータは、対話型コマンド・モードで使用可能なインポート・コマンドに関する情報を提供します。
- KILL_JOB
  Oracle Data Pumpインポートの対話方式コマンド・モードのKILL_JOBパラメータでは、現在接続中のすべてのクライアント・セッションを切断してから、現行のジョブを停止します。インポート・ユーティリティを終了し、端末プロンプトに戻します。
- PARALLEL
  Oracle Data Pumpインポートの対話型コマンド・モードのPARALLELパラメータを使用すると、現在のジョブのアクティブな子プロセスまたはPQ子プロセス(あるいはその両方)の数を増減できます。
- START_JOB
  Oracle Data Pumpインポートの対話方式コマンド・モードのSTART_JOBパラメータでは、接続している現行のジョブを開始します。
- STATUS

Oracle Data Pump Importの対話型コマンドのSTATUSパラメータでは、ジョブ・ステータスが表示され、ロギング・モード・ステータスの表示間隔を更新できます。

- STOP_JOB
  Oracle Data Pumpインポートの対話方式コマンド・モードのSTOP_JOBパラメータでは、現行のジョブを即時にまたは手順に従って停止し、インポートを終了します。

親トピック: Oracleデータ・ポンプ・インポート

3.5.1 Oracle Data Pump Importの対話方式コマンド・モードについて

接続されているクライアント、またはジョブが実行されている端末以外からOracle Data Pumpコマンドを実行する方法を学習します。

対話方式コマンド・モードを開始するには、次のいずれかの方法を使用します。

- 接続されたクライアントから、[Ctrl]を押しながら[C]を押します。
- ジョブを実行している端末以外の端末から、ATTACHパラメータを使用してジョブに接続します。この機能は、ある場所で開始したジョブを、後で別の場所から確認する必要がある場合に有効です。

Oracle Data Pumpの対話方式モード用のコマンド

次の表に、現行のジョブに対して対話方式コマンド・モードでOracle Data Pump Importプロンプトから実行できる操作を示します。

表3-1 Oracle Data Pump Importの対話方式コマンド・モードでサポートされているコマンド

操作	使用するコマンド
対話方式コマンド・モードを終了する。	CONTINUE_CLIENT
現行のジョブは続行したままインポート・クライアント・セッションを停止する。	EXIT_CLIENT
使用可能なコマンドの概要を表示する。	HELP
現在接続中のすべてのクライアント・セッションを切断し、現行のジョブを停止する。	KILL_JOB
現行のジョブに対するアクティブなワーカー・プロセスの数を増減する。このコマンドは、Oracle Database Enterprise Edition でのみ使用可能です。	PARALLEL
接続している停止ジョブを再開する。	START_JOB
現行のジョブの詳細な状態を表示する。	STATUS
現行のジョブを停止する。	STOP_JOB

親トピック: Oracle Data Pump Importの対話方式コマンド・モードで使用可能なコマンド

3.5.2 CONTINUE_CLIENT

Oracle Data Pumpインポートの対話方式コマンド・モードのCONTINUE_CLIENTパラメータでは、対話方式コマンド・モードからロギング・モードに変更します。

用途

モードを、対話方式コマンド・モードからロギング・モードに変更します。

構文および説明

```
CONTINUE_CLIENT
```

ロギング・モードでは、ジョブの状態が端末に継続的に出力されます。ジョブが現在停止している場合、CONTINUE_CLIENTを指定すると、クライアントがジョブの開始を試みます。

例

```
Import> CONTINUE_CLIENT
```

親トピック: Oracle Data Pump Importの対話方式コマンド・モードで使用可能なコマンド

3.5.3 EXIT_CLIENT

Oracle Data Pumpインポートの対話方式コマンド・モードのEXIT_CLIENTパラメータでは、インポート・クライアント・セッションを停止し、インポートを終了して、端末へのロギングを中断します。ただし、現行のジョブの実行は続行します。

用途

インポート・クライアント・セッションを停止し、インポート・ユーティリティを終了して、端末へのロギングを中断します。ただし、現行のジョブの実行は続行します。

構文および説明

```
EXIT_CLIENT
```

EXIT_CLIENTはジョブを実行したままにするため、ジョブがまだ実行中の場合、またはジョブが停止状態の場合は、後でジョブに接続できます。ジョブの状態を確認するには、ジョブのログ・ファイルを監視するか、USER_DATAPUMP_JOBSビューまたはV$SESSION_LONGOPSビューを問い合せることができます。

例

```
Import> EXIT_CLIENT
```

親トピック: Oracle Data Pump Importの対話方式コマンド・モードで使用可能なコマンド

3.5.4 HELP

Oracle Data Pumpインポートの対話型コマンド・モードのHELPパラメータは、対話型コマンド・モードで使用可能なインポート・コマンドに関する情報を提供します。

用途

対話方式コマンド・モードで使用可能なOracle Data Pumpインポート・コマンドの情報を表示します。

構文および説明

```
HELP
```

対話方式コマンド・モードで使用可能なコマンドの情報を表示します。

例

```
Import> HELP
```

親トピック: Oracle Data Pump Importの対話方式コマンド・モードで使用可能なコマンド

3.5.5 KILL_JOB

Oracle Data Pumpインポートの対話方式コマンド・モードのKILL_JOBパラメータでは、現在接続中のすべてのクライアント・セッションを切断してから、現行のジョブを停止します。インポート・ユーティリティを終了し、端末プロンプトに戻します。

用途

現在接続中のすべてのクライアント・セッションを切断してから、現行のジョブを停止します。インポート・ユーティリティを終了し、端末プロンプトに戻します。

構文および説明

```
KILL_JOB
```

KILL_JOBを使用して中断されたジョブは、再開できません。接続中のすべてのクライアント(KILL_JOBコマンドを発行しているクライアントを含む)は、現在のユーザーがジョブを停止しているという警告を受け取った後、切断されます。すべてのクライアントが切断されると、ジョブ・プロセス構造がただちに停止し、データ・ポンプ制御ジョブ表が削除されます。ログ・ファイルは、削除されません。

例

```
Import> KILL_JOB
```

親トピック: Oracle Data Pump Importの対話方式コマンド・モードで使用可能なコマンド

3.5.6 PARALLEL

Oracle Data Pumpインポートの対話型コマンド・モードのPARALLELパラメータを使用すると、現在のジョブのアクティブな子プロセスまたはPQ子プロセス(あるいはその両方)の数を増減できます。

用途

現在のジョブのアクティブな子プロセス、パラレル問合せ(PQ)子プロセス、またはその両方の数を増減できます。

構文および説明

```
PARALLEL=integer
```

PARALLELは、コマンドライン・パラメータおよび対話方式モードのパラメータとして使用可能です。必要な数のパラレル処理を設定できます。増加処理は、リソースが十分にありパラレル化を必要とする作業量が十分にある場合は、即時に実行されます。減少処理は、既存のプロセスが現行のタスクを終了してから実行されます。整数値が減少すると、子プロセスはアイドル状態になりますが、ジョブが終了するまで削除されません。

制限事項

- このパラメータは、Oracle Database 11g以降のリリースのEnterprise Editionでのみ有効です。
- トランスポータブル表領域のメタデータは、パラレルでインポートできません。
- NETWORK_LINKパラメータも使用されていると、メタデータはパラレルでインポートできなくなります

- 次のオブジェクトはパラレルでインポートできません。
  - TRIGGER
  - VIEW
  - OBJECT_GRANT
  - SEQUENCE
  - CONSTRAINT
  - REF_CONSTRAINT

例

```
Import> PARALLEL=10
```

親トピック: Oracle Data Pump Importの対話方式コマンド・モードで使用可能なコマンド

3.5.7 START_JOB

Oracle Data Pumpインポートの対話方式コマンド・モードのSTART_JOBパラメータでは、接続している現行のジョブを開始します。

用途

接続している現行のジョブを開始します。

構文および説明

```
START_JOB[=SKIP_CURRENT=YES]
```

START_JOBコマンドは、(現在実行できない)接続中のジョブを再開します。ダンプ・ファイル・セットおよび親ジョブ表が元のまま保持されている場合は、予期しない障害またはSTOP_JOBコマンドの発行後にデータの損失や破損なしにジョブが再開されます。

SKIP_CURRENTオプションでは、以前失敗したジョブ、または特定のオブジェクトでハングしたか、実行が遅いジョブを再開できます。失敗する文または現在処理されているオブジェクトはスキップされ、ジョブは次の項目から再開されます。パラレル・ジョブの場合は、このオプションが原因で、各ワーカーでの現在の処理がスキップされ、再開時に次の項目に移動されます。

トランスポータブル表領域モードで実行されたSQLFILEジョブまたはインポートを再開することはできません。

例

```
Import> START_JOB
```

親トピック: Oracle Data Pump Importの対話方式コマンド・モードで使用可能なコマンド

3.5.8 STATUS

Oracle Data Pump Importの対話型コマンドのSTATUSパラメータでは、ジョブ・ステータスが表示され、ロギング・モード・ステータスの表示間隔を更新できます。

用途

ジョブの累積的な状態、現行の操作の説明および推定完了率を表示します。ロギング・モードの状態を表示する間隔を再設定することもできます。

構文および説明

```
STATUS[=integer]
```

ロギング・モードでのこの状態の表示頻度を秒単位で指定できるオプションがあります。値を入力しなかった場合またはデフォルト

値の0を使用した場合は、状態の定期表示はオフになり、状態は1回のみ表示されます。

この状態情報は、標準出力デバイスのみに書き込まれ、ログ・ファイルには(使用可能な場合でも)書き込まれません。

例

次に、現行のジョブの状態を表示し、ロギング・モードの表示間隔を2分(120秒)に変更する例を示します。

```
Import> STATUS=120
```

親トピック: Oracle Data Pump Importの対話方式コマンド・モードで使用可能なコマンド

3.5.9 STOP_JOB

Oracle Data Pumpインポートの対話方式コマンド・モードのSTOP_JOBパラメータでは、現行のジョブを即時にまたは手順に従って停止し、インポートを終了します。

用途

現行のジョブを即時にまたは手順に従って停止し、インポートを終了します。

構文および説明

```
STOP_JOB[=IMMEDIATE]
```

STOP_JOBを実行してから、START_JOBを使用して、後でジョブを接続および再開できます。ジョブを接続および再開するには、コマンドの発行時またはコマンドの発行後に、マスター表とダンプ・ファイル・セットに障害が発生していない必要があります。

手順に従って停止する場合は、関連する値を指定しないでSTOP_JOBを使用します。確認を要求する警告が発行されます。手順に従った停止では、ワーカー・プロセスで現行のタスクが終了した後、ジョブが停止されます。

即時に停止するには、STOP_JOB=IMMEDIATEを指定します。確認を要求する警告が発行されます。接続中のすべてのクライアント(STOP_JOBコマンドを発行しているクライアントを含む)は、現在のユーザーがジョブを停止しているという警告を受け取ります。その後、これらは切断されます。すべてのクライアントが切断されると、ジョブのプロセス構造が即時に停止されます。つまり、データ・ポンプ制御ジョブ・プロセスは、ワーカー・プロセスが現在のタスクを終了するまで待機しません。STOP_JOB=IMMEDIATEを指定した場合、破損やデータ損失の危険性はありません。ただし、停止時に完了していなかった一部のタスクは、再開時に再実行が必要になる可能性があります。

例

```
Import> STOP_JOB=IMMEDIATE
```

親トピック: Oracle Data Pump Importの対話方式コマンド・モードで使用可能なコマンド

3.6 Oracleデータ・ポンプ・インポートの使用例

次の一般的なシナリオ例を使用して、データの移動にOracleデータ・ポンプ・インポートを使用する方法について学習します。

- データのみ表モード・インポートの実行
  Oracle Data Pumpを使用してデータのみ表モード・インポートを実行する方法を参照してください。
- スキーマ・モード・インポートの実行
  Oracle Data Pumpを使用してスキーマ・モード・インポートを実行する方法を参照してください。
- ネットワーク・モード・インポートの実行
  Oracle Data Pumpを使用してネットワーク・モード・インポートを実行する方法を参照してください。
- URLベースのダンプ・ファイル名でのワイルドカードの使用

Oracleデータ・ポンプでは、URLベースのダンプ・ファイル名にワイルドカードを使用できるため、Oracleオブジェクト・ストア・サービスからOracle Autonomous Databaseへの複数のダンプ・ファイルのインポートが簡単になります。

親トピック: Oracleデータ・ポンプ・インポート

3.6.1 データのみ表モード・インポートの実行

Oracle Data Pumpを使用してデータのみ表モード・インポートを実行する方法を参照してください。

この例では、表はemployeesという名前です。ここでは、表モード・エクスポートの実行で作成したダンプ・ファイルを使用します。

CONTENT=DATA_ONLYパラメータは、すべてのデータベース・オブジェクト定義(メタデータ)をフィルタから除外します。表の行データのみロードされます。

例3-1 データのみ表モード・インポートの実行

```
> impdp hr TABLES=employees CONTENT=DATA_ONLY DUMPFILE=dpump_dir1:table.dmp
NOLOGFILE=YES
```

関連トピック

- 表モード・エクスポートの実行

親トピック: Oracleデータ・ポンプ・インポートの使用例

3.6.2 スキーマ・モード・インポートの実行

Oracle Data Pumpを使用してスキーマ・モード・インポートを実行する方法を参照してください。

これは、スキーマ・モード・エクスポートの実行で作成したダンプ・ファイル・セットのスキーマ・モード・インポートの例です。

例3-2 スキーマ・モード・インポートの実行

```
> impdp hr SCHEMAS=hr DIRECTORY=dpump_dir1 DUMPFILE=expschema.dmp
 EXCLUDE=CONSTRAINT,REF_CONSTRAINT,INDEX TABLE_EXISTS_ACTION=REPLACE
```

EXCLUDEパラメータは、インポートしたメタデータをフィルタします。指定したインポート・モードでは、EXCLUDE文に指定されたオブジェクトを除き、ソースに含まれるすべてのオブジェクトおよびその依存オブジェクトが含まれます。オブジェクトが除外されると、そのオブジェクトのすべての依存オブジェクトも除外されます。TABLE_EXISTS_ACTION=REPLACEパラメータでは、表がすでに存在する場合は削除し、ダンプ・ファイルの内容でその表を再作成してロードするように指定します。

関連トピック

- スキーマ・モード・エクスポートの実行

親トピック: Oracleデータ・ポンプ・インポートの使用例

3.6.3 ネットワーク・モード・インポートの実行

Oracle Data Pumpを使用してネットワーク・モード・インポートを実行する方法を参照してください。

ネットワーク・モード・インポートでは、NETWORK_LINKパラメータで指定されたデータベースをソースとして使用します。

例3-3 スキーマのネットワーク・モード・インポート

```
> impdp hr TABLES=employees REMAP_SCHEMA=hr:scott DIRECTORY=dpump_dir1
NETWORK_LINK=dblink
```

この例では、hrスキーマからscottスキーマへemployees表をインポートします。dblinkは、ターゲット・データベースとは異なるソース・データベースを示します。

スキーマを再マップするには、ユーザーhrに、ローカル・データベースのDATAPUMP_IMP_FULL_DATABASEロールおよびソース・データベースのDATAPUMP_EXP_FULL_DATABASEロールが必要です。

REMAP_SCHEMAは、ソース・スキーマにあるすべてのオブジェクトをターゲット・スキーマにロードします。

関連トピック

- NETWORK_LINK

親トピック: Oracleデータ・ポンプ・インポートの使用例

3.6.4 URLベースのダンプ・ファイル名でのワイルドカードの使用

Oracleデータ・ポンプでは、URLベースのダンプ・ファイル名にワイルドカードを使用できるため、Oracleオブジェクト・ストア・サービスからOracle Autonomous Databaseへの複数のダンプ・ファイルのインポートが簡単になります。

例3-4 URLベースのファイル名に使用したワイルドカード

この例は、Oracleオブジェクト・ストア・サービスからOracle Autonomous Databaseに複数のダンプ・ファイルをインポートするために、ファイル名にワイルドカードを使用する方法を示しています。

```
> impdp admin/password@ATPC1_high
        directory=data_pump_dir credential=my_cred_name
        dumpfile= https://objectstorage.example.com/v1/atpc/atpc_user/exp%u.dmp"
```

ノート:

ワイルドカード文字は、URL のバケット名コンポーネントには使用できません。

親トピック: Oracleデータ・ポンプ・インポートの使用例

3.7 Oracle Data Pump Importの構文図

構文図を使用すると、Oracle Data Pump Importでの有効なSQL構文を理解できます。

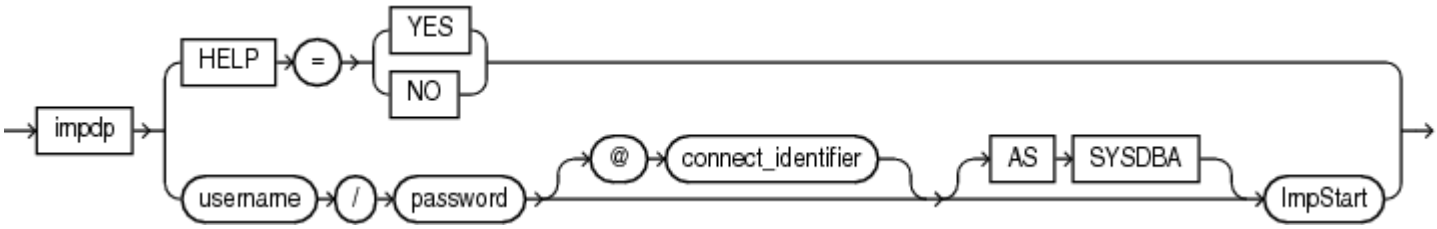
構文図の読み方

構文図とは、SQLの有効な構文を図で示したものです。構文図は、矢印が示す方向に左から右へ読んでください。

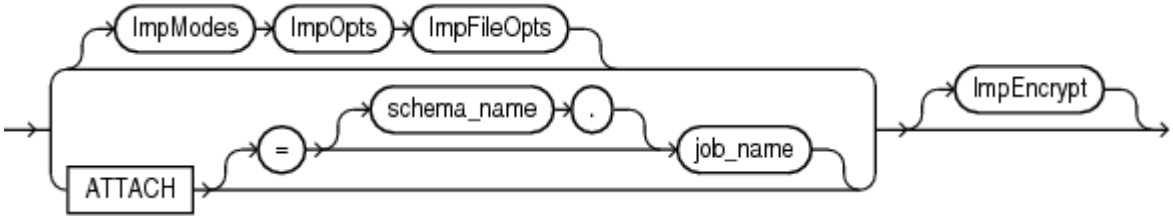
標準のSQL構文表記法の詳細は、次を参照してください。

『Oracle Database SQL言語リファレンス』の「構文図の読み方」

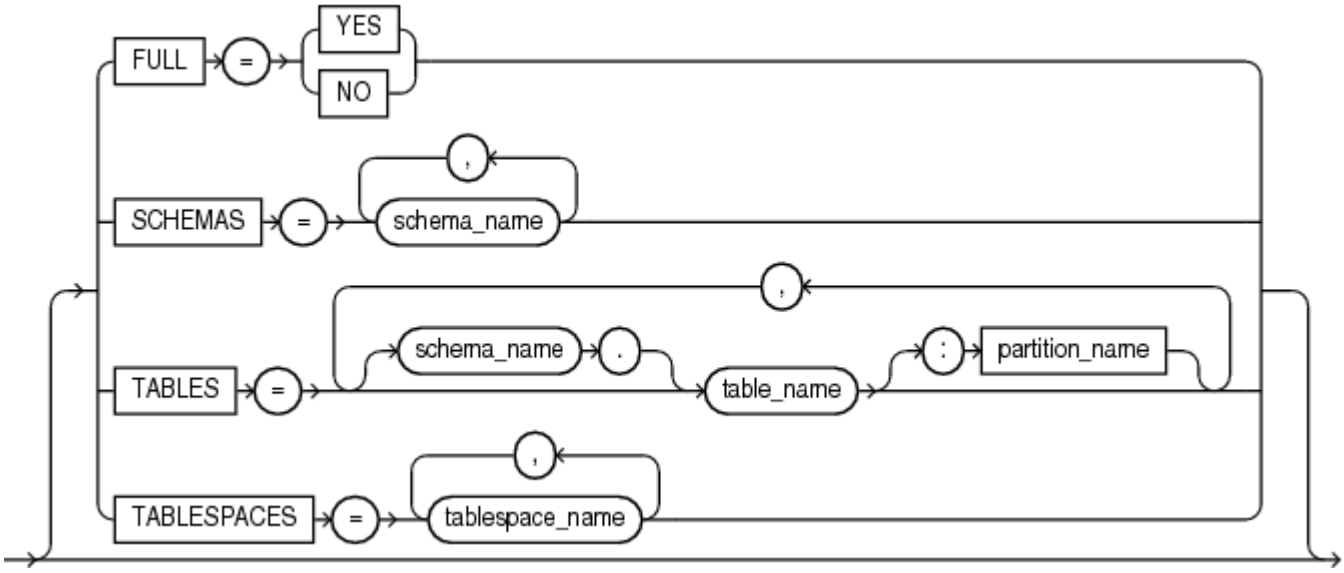
ImpInit



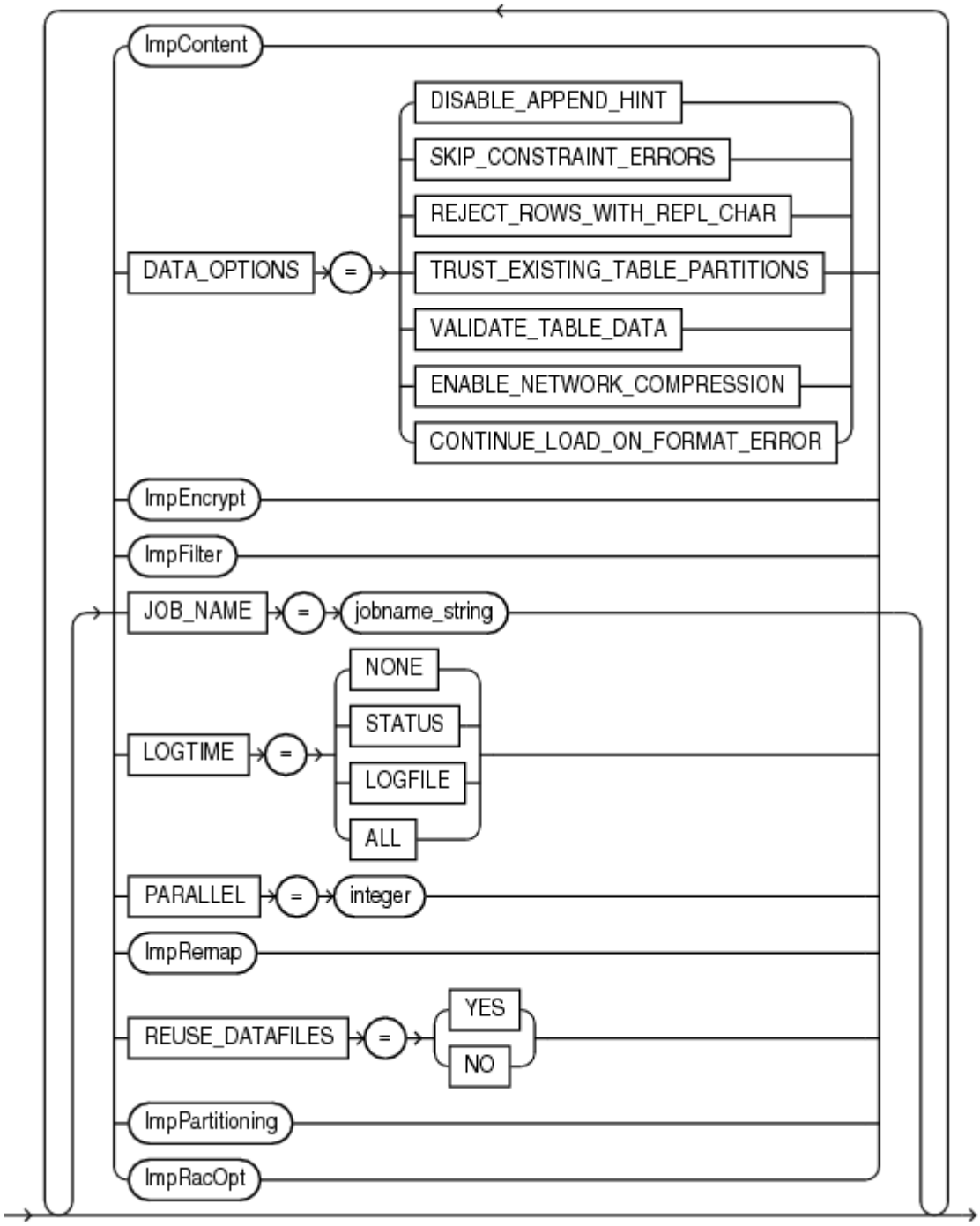
ImpStart



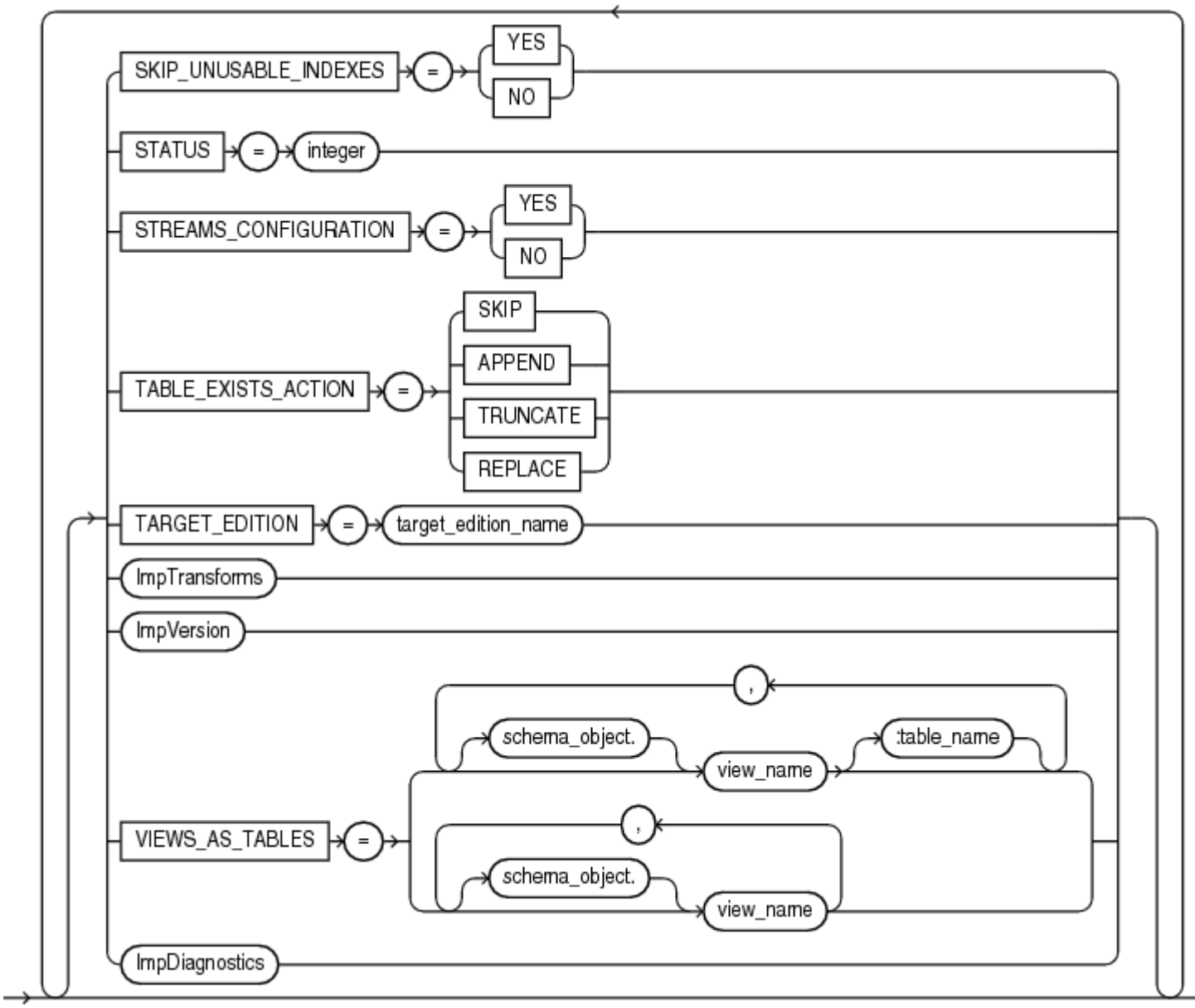
ImpModes



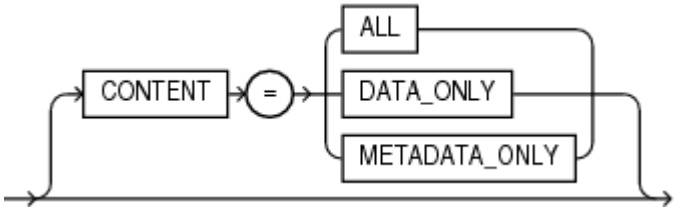
ImpOpts


ImpContent
DISABLE_APPEND_HINT
SKIP_CONSTRAINT_ERRORS
REJECT_ROWS_WITH_REPL_CHAR
DATA_OPTIONS
=
TRUST_EXISTING_TABLE_PARTITIONS
VALIDATE_TABLE_DATA
ENABLE_NETWORK_COMPRESSION
CONTINUE_LOAD_ON_FORMAT_ERROR
ImpEncrypt
ImpFilter
JOB_NAME
=
jobname_string
NONE
STATUS
LOGTIME
=
LOGFILE
ALL
PARALLEL
=
integer
ImpRemap
YES
REUSE_DATAFILES
=
NO
ImpPartitioning
ImpRacOpt

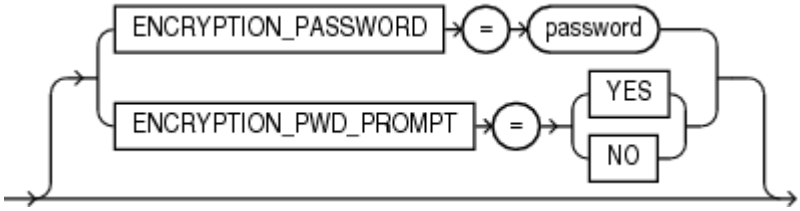
ImpOpts_続き



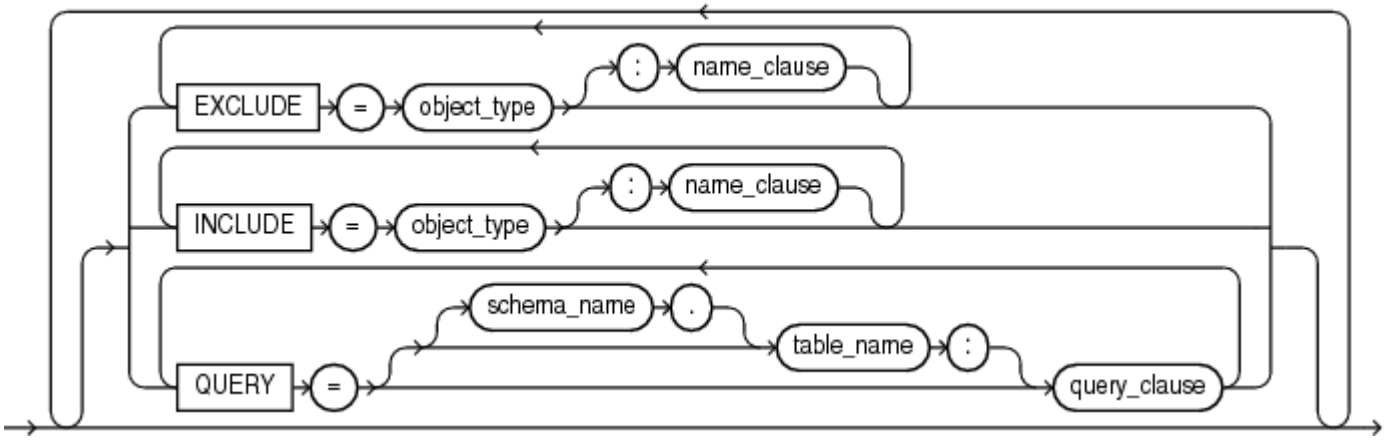
ImpContent



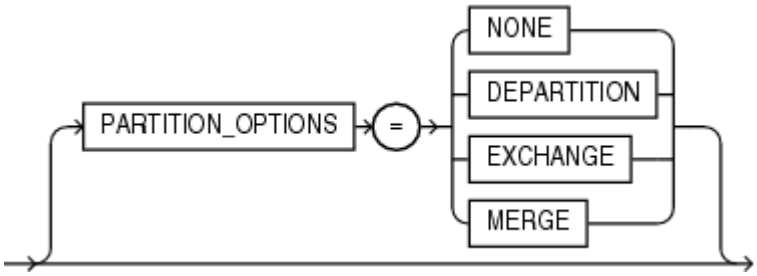
ImpEncrypt



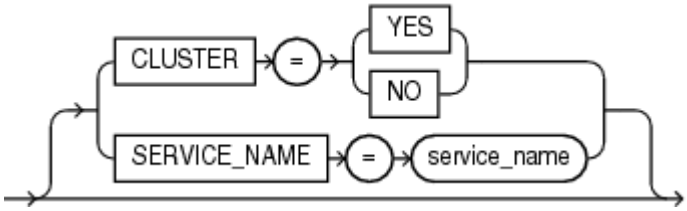
ImpFilter



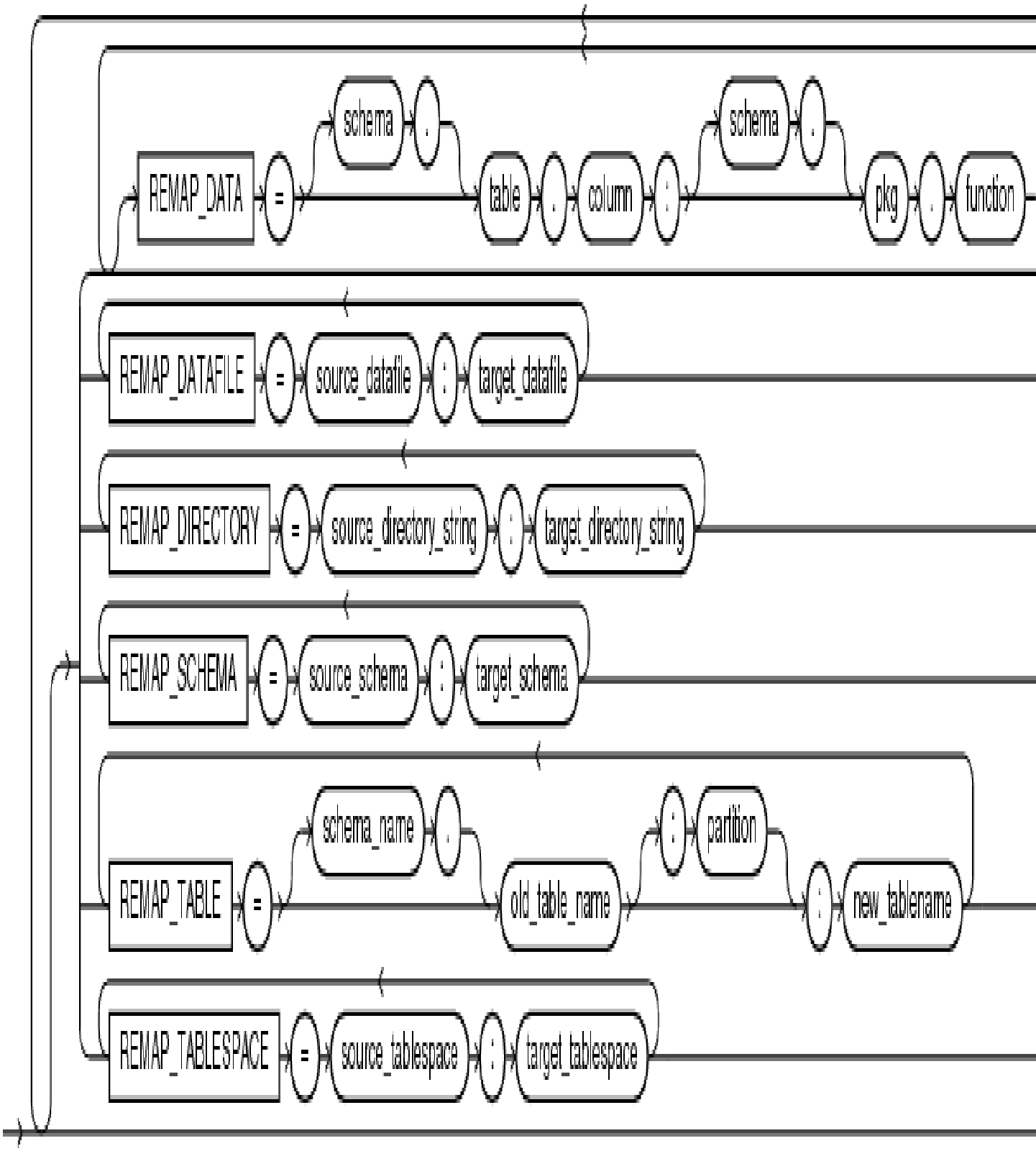
ImpPartitioning



ImpRacOpt

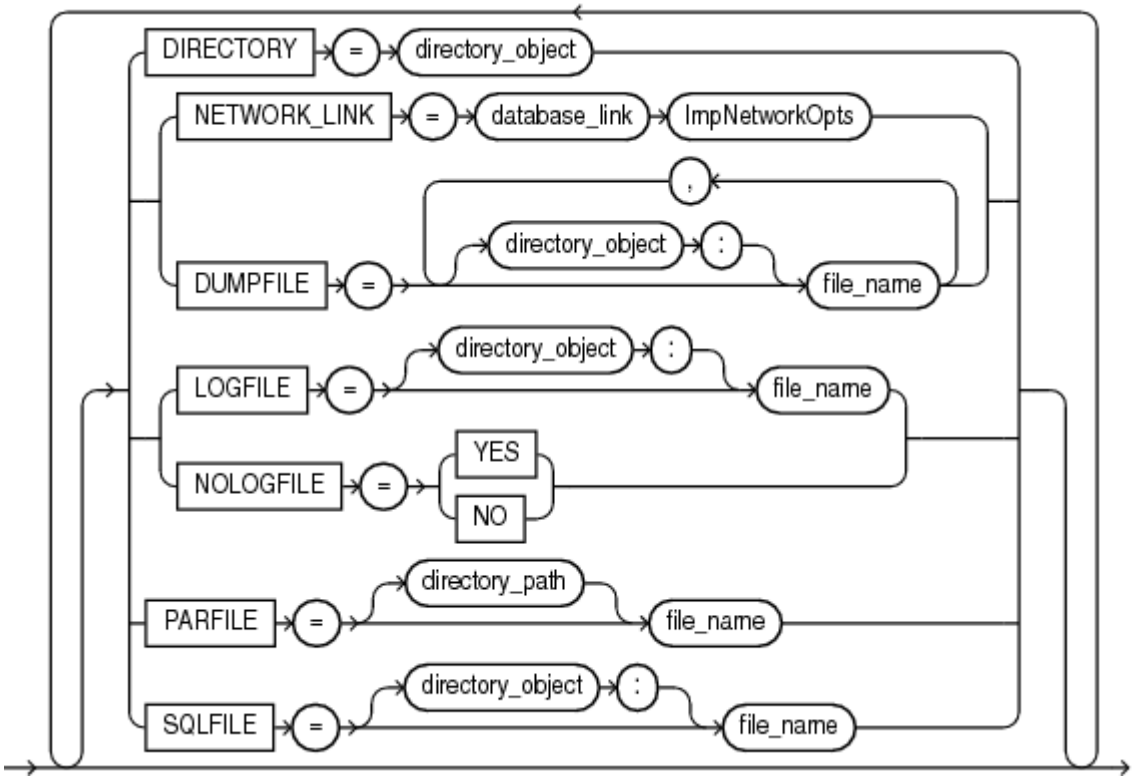


ImpRemap

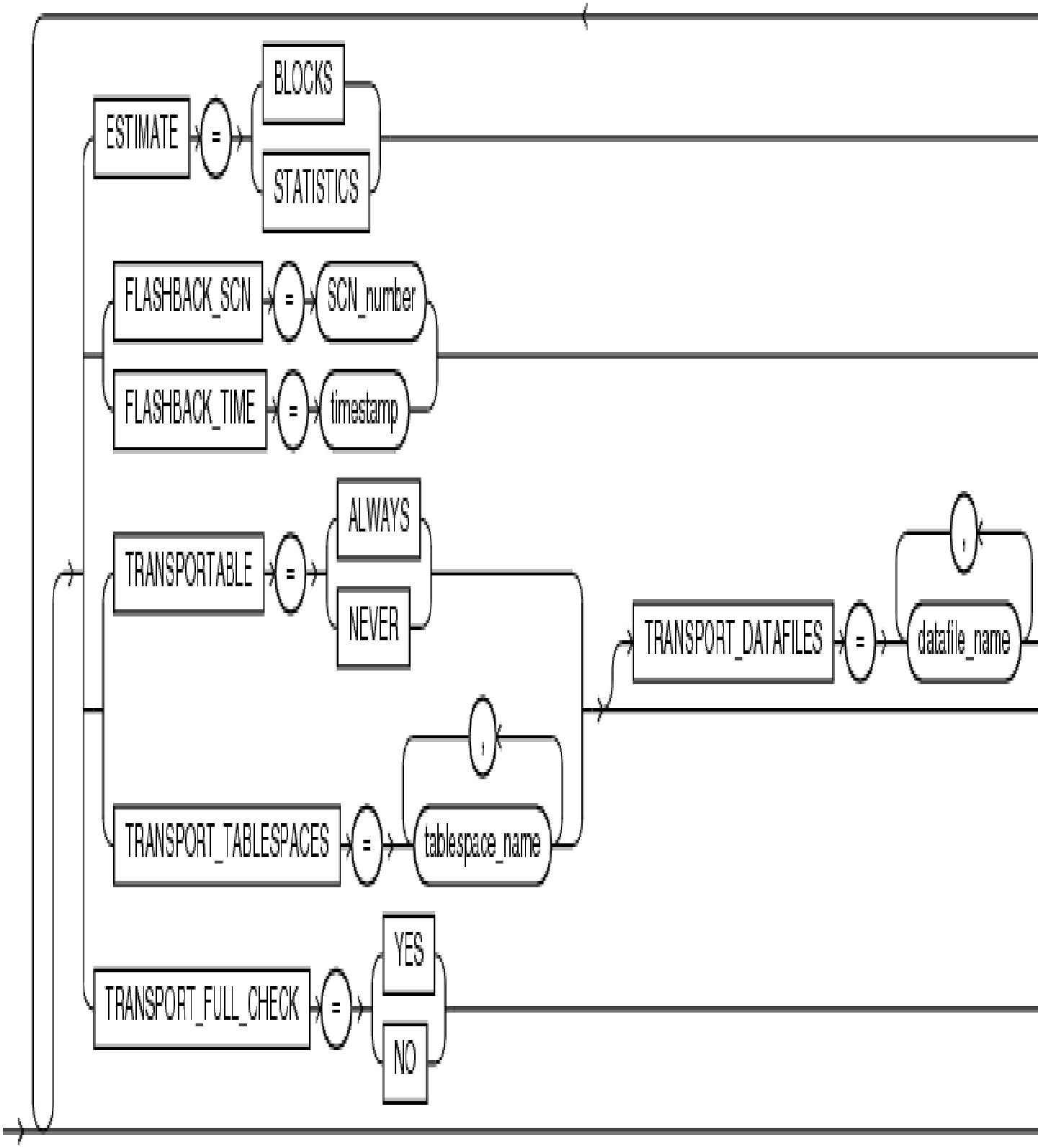


ノート: REMAP_DIRECTORYおよびREMAP_DATAFILEは、相互に排他的なパラメータです。

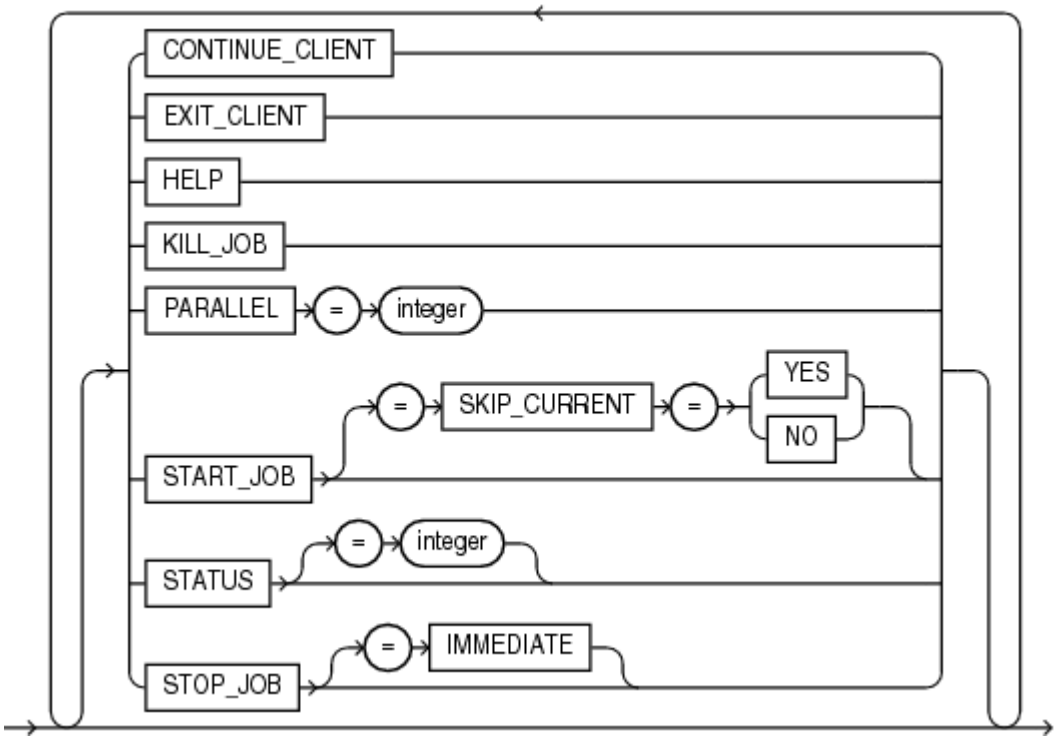
ImpFileOpts



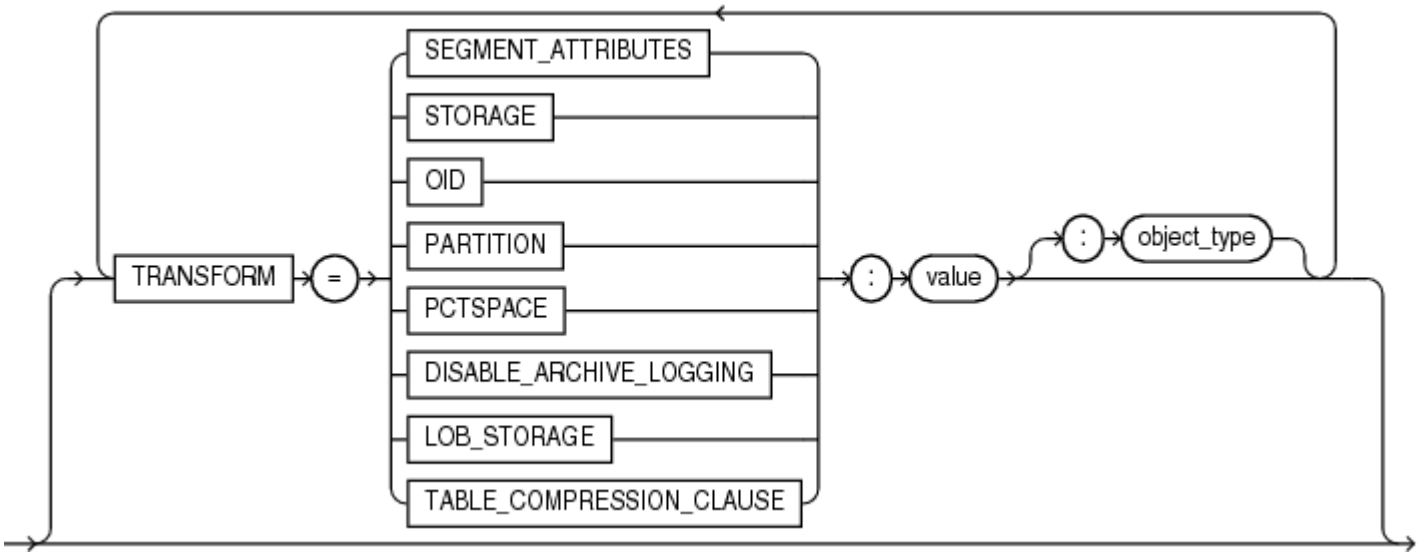
ImpNetworkOpts


ESTIMATE
=
BLOCKS
STATISTICS
FLASHBACK_SCN
=
SCN_number
FLASHBACK_TIME
=
timestamp
TRANSPORTABLE
=
ALWAYS
NEVER
TRANSPORT_DATAFILES
=
,
datafile_name
TRANSPORT_TABLESPACES
=
,
tablespace_name
TRANSPORT_FULL_CHECK
=
YES
NO

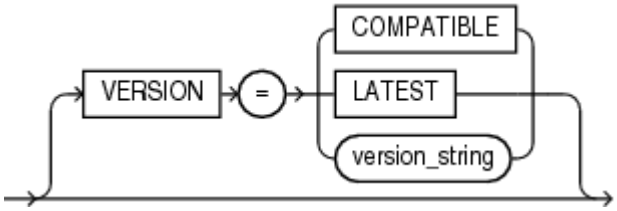
ImpDynOpts



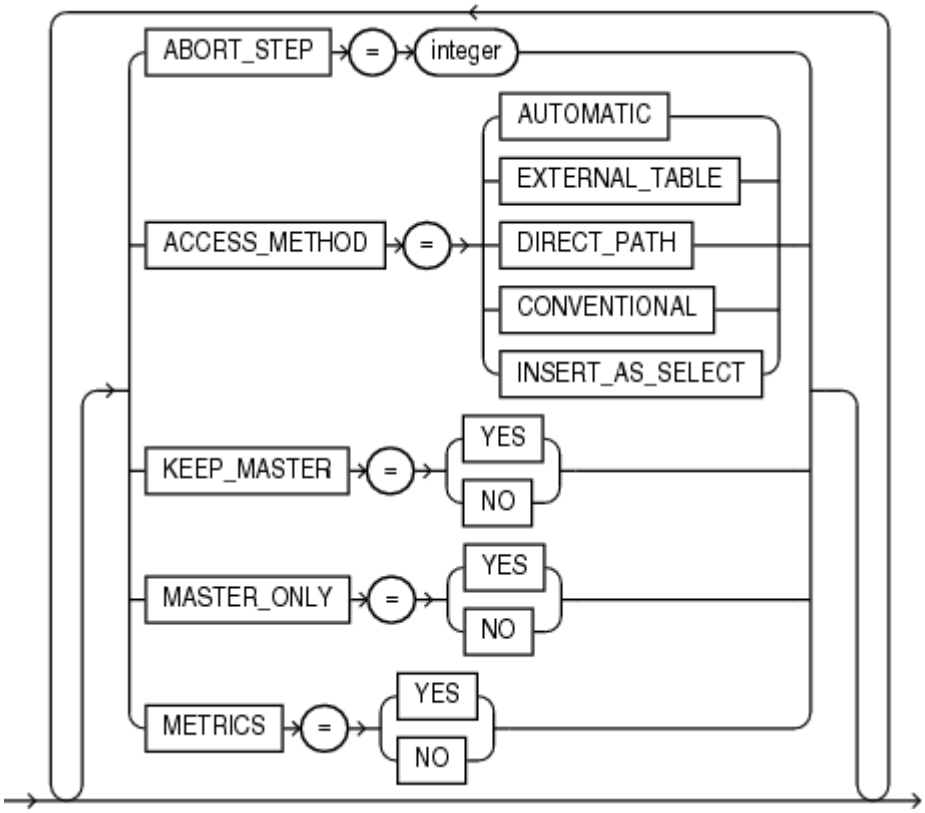
ImpTransforms



ImpVersion



ImpDiagnostics



親トピック: Oracleデータ・ポンプ・インポート

4 Oracle Data Pumpのレガシー・モード

Oracle Data Pumpのレガシー・モードでは、Oracle Data Pumpエクスポートとデータ・ポンプ・インポートのコマンドラインで、オリジナルのエクスポート・パラメータとインポート・パラメータを使用できます。

- Oracle Data Pumpのレガシー・モードのユースケース
  Oracle Data Pumpは、レガシーのエクスポートまたはインポート・パラメータが検出されるとレガシー・モードになります。そのため、既存のスクリプトを引き続き使用できます。
- パラメータのマッピング
  オリジナルのエクスポートおよびインポート・パラメータと、Oracleデータ・ポンプ・エクスポートおよびデータ・ポンプ・インポートの同様の機能を持つパラメータとのマッピング関係について説明します。
- データ・ポンプ・レガシー・モードでのファイルの場所の管理
  オリジナルのバージョンはクライアントベース、データ・ポンプはサーバーベースであるため、オリジナルのエクスポートとインポート、およびデータ・ポンプ・エクスポートとデータ・ポンプ・インポートは、ダンプ・ファイルとログ・ファイルの書込み/読取りを行える場所によって異なります。
- Oracle Data Pumpのログ・ファイルとエラーに対する既存のスクリプトの調整
  Oracle Data Pumpをレガシー・モードで使用する場合、オリジナルのエクスポートおよびインポート用に作成された既存のスクリプトを確認し、更新する必要があります

親トピック: Oracle Data Pump

4.1 Oracle Data Pumpのレガシー・モードのユースケース

Oracle Data Pumpは、レガシーのエクスポートまたはインポート・パラメータが検出されるとレガシー・モードになります。そのため、既存のスクリプトを引き続き使用できます。

オリジナルのエクスポート・ユーティリティ(exp)とインポート・ユーティリティ(imp)を使用している場合は、長年使用してきたスクリプトがあるでしょう。データ・ポンプには、既存のスクリプトをOracle Data Pumpで引き続き使用できるレガシー・モードが用意されています。

コマンドラインまたはスクリプトのどちらかにオリジナルのエクスポートまたはインポートに特有のパラメータが見つかると、Oracle Data Pumpはレガシー・モードに移行します。そのパラメータがデータ・ポンプで処理されるときに、Oracle Data Pumpエクスポートまたは Oracle Data Pumpインポートの同等のパラメータが表示されます。新しい構文を確認し、時間があるときにスクリプトを変更することをお薦めします。

> ノート:
>
> Oracle Data Pump エクスポートおよびデータ・ポンプ・インポート・ユーティリティでは、Oracle Data Pump 形式のダンプ・ファイルとログ・ファイルのみの作成と読取りが実行されます。オリジナルのエクスポートやインポートと互換性のあるダンプ・ファイルが作成されたり、読み込まれることはありません。オリジナルのエクスポートで作成されたダンプ・ファイルがある場合は、オリジナルのインポートを使用してデータベースにデータをインポートする必要があります。

親トピック: Oracle Data Pumpのレガシー・モード

4.2 パラメータのマッピング

オリジナルのエクスポートおよびインポート・パラメータと、Oracleデータ・ポンプ・エクスポートおよびデータ・ポンプ・インポートの同様の機能を持つパラメータとのマッピング関係について説明します。

- データ・ポンプでのオリジナルのエクスポート・パラメータの使用
  対応するデータ・ポンプ・パラメータにマップするオリジナルのエクスポート・パラメータがある場合は、そのエクスポート・パラメータをデータ・ポンプ・エクスポートで使用できます。
- データ・ポンプでのオリジナルのインポート・パラメータの使用
  データ・ポンプ・パラメータにマップするオリジナルのインポート・パラメータがある場合は、そのインポート・パラメータをデータ・ポンプ・インポートで使用できます。

関連トピック

- Oracleデータ・ポンプ・エクスポート
- Oracleデータ・ポンプ・インポート
- オリジナルのエクスポート
- オリジナルのインポート

親トピック: Oracle Data Pumpのレガシー・モード

4.2.1 データ・ポンプでのオリジナルのエクスポート・パラメータの使用

対応するデータ・ポンプ・パラメータにマップするオリジナルのエクスポート・パラメータがある場合は、そのエクスポート・パラメータをデータ・ポンプ・エクスポートで使用できます。

この表では、オリジナルのエクスポート・パラメータがデータ・ポンプ・エクスポートでどのように解釈されるかを示します。オリジナルのエクスポートとデータ・ポンプのエクスポートで名前と機能が同じパラメータは、この表に含まれていません。

表4-1 データ・ポンプ・エクスポートでのオリジナルのエクスポート・パラメータの扱い

オリジナルのエクスポート・パラメータ	データ・ポンプ・エクスポート・パラメータでの処理
BUFFER	このパラメータは無視されます。
COMPRESS	このパラメータは無視されます。オリジナルのエクスポートの場合、COMPRESS パラメータは初期エクステントの管理方法に影響していました。オリジナルのエクスポートでは、COMPRESS=n に設定すると、初期エクステントと第 2 エクステントに対して現行の記憶域パラメータが使用されていました。 データ・ポンプ・エクスポートの COMPRESSION パラメータは、ダンプ・ファイルのデータ圧縮方法を指定するために使用されます。オリジナルのエクスポートの COMPRESS パラメータとは関連がありません。
CONSISTENT	データ・ポンプ・エクスポートは現在の時刻を確認して、FLASHBACK_TIME を使用します。
CONSTRAINTS	オリジナルのエクスポートで CONSTRAINTS=n を使用していた場合、データ・ポンプ・エクスポー

オリジナルのエクスポート・パラメータ	データ・ポンプ・エクスポート・パラメータでの処理
	トは EXCLUDE=CONSTRAINTS を使用します。 デフォルトの動作では、制約がエクスポートに含まれます。
DIRECT	このパラメータは無視されます。データ・ポンプ・エクスポートでは、最適なエクスポート方法が自動的に選択されます。
FEEDBACK	データ・ポンプ・エクスポートの STATUS=30 コマンドが使用されます。STATUS コマンドでは、処理されている行に加えてエクスポート・ジョブの状態も返されるため、これは直接のマッピングではありません。 オリジナルのエクスポートでは、一定の行数(FEEDBACK コマンドで指定)の後にフィードバックが表示されていました。データ・ポンプ・エクスポートでは、STATUS で指定された秒数ごとに状態が表示されます。
FILE	データ・ポンプ・エクスポートは、FILE パラメータに対する指定パスまたはデフォルト・パスを確認します。また、スキーマに書込みアクセス権があるディレクトリ・オブジェクトが存在するかどうかも確認します。 データ・ポンプでのオリジナルのエクスポートの FILE パラメータの処理方法の詳細は、「データ・ポンプ・レガシー・モードでのファイルの場所の管理」を参照してください。
GRANTS	オリジナルのエクスポートで GRANTS=n を使用していた場合、データ・ポンプ・エクスポートは EXCLUDE=GRANT を使用します。 オリジナルのエクスポートで GRANTS=y を使用していた場合、そのパラメータは無視されます。その指定はデータ・ポンプ・エクスポートのデフォルトの動作であるため、再マップする必要はありません。
INDEXES	オリジナルのエクスポートで INDEXES=n を使用していた場合、データ・ポンプ・エクスポートは EXCLUDE=INDEX パラメータを使用します。 オリジナルのエクスポートで INDEXES=y を使用していた場合、そのパラメータは無視されます。その指定はデータ・ポンプ・エクスポートのデフォルトの動作であるため、再マップする必要はありません。
LOG	データ・ポンプ・エクスポートは、LOG パラメータに対する指定パスまたはデフォルト・パスを確認します。また、スキーマに書込みアクセス権があるディレクトリ・オブジェクトが存在するかどうかも確認します。

オリジナルのエクスポート・パラメータ	データ・ポンプ・エクスポート・パラメータでの処理
	データ・ポンプでのオリジナルのエクスポートの LOG パラメータの処理方法の詳細は、「データ・ポンプ・レガシー・モードでのファイルの場所の管理」を参照してください。 ログ・ファイルの内容は、データ・ポンプ・エクスポート操作の内容になります。ログ・ファイルの場所と内容については、「ログ・ファイル」を参照してください。
OBJECT_CONSISTENT	このパラメータは無視されます。データ・ポンプ・エクスポートの処理では、エクスポートされる各オブジェクトの状態に一貫性があることが保証されています。
OWNER	データ・ポンプの SCHEMAS パラメータが使用されます。
RECORDLENGTH	このパラメータは無視されます。データ・ポンプ・エクスポートでは、バッファ・サイズは自動的に調整されます。
RESUMABLE	このパラメータは無視されます。データ・ポンプ・エクスポートの場合、この機能は EXP_FULL_DATABASE ロールを付与されているユーザーに自動的に提供されます。
RESUMABLE_NAME	このパラメータは無視されます。データ・ポンプ・エクスポートの場合、この機能は EXP_FULL_DATABASE ロールを付与されているユーザーに自動的に提供されます。
RESUMABLE_TIMEOUT	このパラメータは無視されます。データ・ポンプ・エクスポートの場合、この機能は EXP_FULL_DATABASE ロールを付与されているユーザーに自動的に提供されます。
ROWS	オリジナルのエクスポートで ROWS=y を使用していた場合、データ・ポンプ・エクスポートは CONTENT=ALL パラメータを使用します。 オリジナルのエクスポートで ROWS=n を使用していた場合、データ・ポンプ・エクスポートは CONTENT=METADATA_ONLY パラメータを使用します。
STATISTICS	このパラメータは無視されます。統計は、データ・ポンプ・エクスポート操作の一環として、表に対して常に保存されます。
TABLESPACES	オリジナルのエクスポートで TRANSPORT_TABLESPACE=n も指定していた場合、データ・ポンプ・エクスポートは TABLESPACES パラメータを無視します。 オリジナルのエクスポートで TRANSPORT_TABLESPACE=y も指定していた場合、データ・ポンプ・エクスポートは、TABLESPACES パラメータに対して指定されていた名前を、データ・ポンプ・エクスポートの TRANSPORT_TABLESPACES パラメータに対して使用します。

オリジナルのエクスポート・パラメータ	データ・ポンプ・エクスポート・パラメータでの処理
TRANSPORT_TABLESPACE	オリジナルのエクスポートで TRANSPORT_TABLESPACE=n(デフォルト)を使用していた場合、データ・ポンプ・エクスポートは TABLESPACES パラメータを使用します。 オリジナルのエクスポートで TRANSPORT_TABLESPACE=y を使用していた場合、データ・ポンプ・エクスポートは TRANSPORT_TABLESPACES パラメータを使用し、メタデータのみがエクスポートされます。
TRIGGERS	オリジナルのエクスポートで TRIGGERS=n を使用していた場合、データ・ポンプ・エクスポートは EXCLUDE=TRIGGER パラメータを使用します。 オリジナルのエクスポートで TRIGGERS=y を使用していた場合、そのパラメータは無視されます。その指定はデータ・ポンプ・エクスポートのデフォルトの動作であるため、再マップする必要はありません。
TTS_FULL_CHECK	オリジナルのエクスポートで TTS_FULL_CHECK=y を使用していた場合、データ・ポンプ・エクスポートは TRANSPORT_FULL_CHECK パラメータを使用します。 オリジナルのエクスポートで TTS_FULL_CHECK=y を使用していた場合、そのパラメータは無視されます。その指定はデータ・ポンプ・エクスポートのデフォルトの動作であるため、再マップする必要はありません。
VOLSIZE	オリジナルのエクスポートで VOLSIZE パラメータが使用されている場合、それはダンプ・ファイルの指定場所がテープ・デバイスであることを意味します。テープ・デバイスは、データ・ポンプ・エクスポートのダンプ・ファイル形式でサポートされていません。したがって、この操作はエラーが発生して終了します。

親トピック: パラメータのマッピング

4.2.2 データ・ポンプでのオリジナルのインポート・パラメータの使用

データ・ポンプ・パラメータにマップするオリジナルのインポート・パラメータがある場合は、そのインポート・パラメータをデータ・ポンプ・インポートで使用できます。

この表では、オリジナルのインポート・パラメータがデータ・ポンプ・インポートでどのように解釈されるかを示します。オリジナルのインポートとデータ・ポンプのインポートで名前と機能が同じパラメータは、この表に含まれていません。

表4-2 データ・ポンプ・インポートでのオリジナルのインポート・パラメータの扱い

オリジナルのインポート・パラメータ	データ・ポンプ・インポート・パラメータでの処理
BUFFER	このパラメータは無視されます。

オリジナルのインポート・パラメータ	データ・ポンプ・インポート・パラメータでの処理
CHARSET	このパラメータは、数リリース前からサポートされなくなりました。今後は使用しないでください。このパラメータは、データ・ポンプ・インポート操作が中断される原因になります。
COMMIT	このパラメータは無視されます。データ・ポンプ・インポートでは、表の処理が終了するたびにコミットが自動的に実行されます。
COMPILE	このパラメータは無視されます。データ・ポンプ・インポートでは、プロシージャの作成後にコンパイルが実行されます。依存性の理由で必要な場合は、再コンパイルを実行できます。
CONSTRAINTS	オリジナルのインポートで CONSTRAINTS=n を使用していた場合、データ・ポンプ・インポートは EXCLUDE=CONSTRAINT パラメータを使用します。 オリジナルのインポートで CONSTRAINTS=y を使用していた場合、そのパラメータは無視されます。その指定はデータ・ポンプ・インポートのデフォルトの動作であるため、再マップする必要はありません。
DATAFILES	データ・ポンプ・インポートの TRANSPORT_DATAFILES パラメータが使用されます。
DESTROY	オリジナルのインポートで DESTROY=y を使用していた場合、データ・ポンプ・インポートは REUSE_DATAFILES=y パラメータを使用します。 オリジナルのインポートで DESTROY=n を使用していた場合、そのパラメータは無視されます。その指定はデータ・ポンプ・インポートのデフォルトの動作であるため、再マップする必要はありません。
FEEDBACK	データ・ポンプ・インポートの STATUS=30 コマンドが使用されます。STATUS コマンドでは、処理されている行に加えてインポート・ジョブの状態も返されるため、これは直接のマッピングではありません。 オリジナルのインポートでは、一定の行数(FEEDBACK コマンドで指定)の後にフィードバックが表示されていました。データ・ポンプ・インポートでは、STATUS で指定された秒数ごとに状態が表示されます。
FILE	データ・ポンプ・インポートは、FILE パラメータに対する指定パスまたはデフォルト・パスを確認します。また、スキーマに書込みアクセス権があるディレクトリ・オブジェクトが存在するかどうかも確認します。 データ・ポンプでのオリジナルのインポートの FILE パラメータの処理方法の詳細は、「データ・ポンプ・レガシー・モードでのファイルの場所の管理」を参照してください。

オリジナルのインポート・パラメータ	データ・ポンプ・インポート・パラメータでの処理
FILESIZE	このパラメータは無視されます。この情報はデータ・ポンプのダンプ・ファイル・セットにすでに含まれています。
FROMUSER	データ・ポンプ・インポートの SCHEMAS パラメータが使用されます。TOUSER を使用せずに FROMUSER が使用されていた場合は、IMP_FULL_DATABASE ロールを持つインポート・スキーマによって、データ・ポンプ・インポートでスキーマが作成され、そのスキーマのオブジェクトがインポートされます。IMP_FULL_DATABASE ロールを持っていないインポート・スキーマは、ダンプ・ファイル・セットから自分のスキーマのみをインポートできます。
GRANTS	オリジナルのインポートで GRANTS=n を使用していた場合、データ・ポンプ・インポートは EXCLUDE=OBJECT_GRANT パラメータを使用します。 オリジナルのインポートで GRANTS=y を使用していた場合、そのパラメータは無視されます。その指定はデータ・ポンプ・インポートのデフォルトの動作であるため、再マップする必要はありません。
IGNORE	オリジナルのインポートで IGNORE=y を使用していた場合、データ・ポンプ・インポートは TABLE_EXISTS_ACTION=APPEND パラメータを使用します。これにより、表データの処理が続行されます。 オリジナルのインポートで IGNORE=n を使用していた場合、そのパラメータは無視されます。その指定はデータ・ポンプ・インポートのデフォルトの動作であるため、再マップする必要はありません。
INDEXES	オリジナルのインポートで INDEXES=n を使用していた場合、データ・ポンプ・インポートは EXCLUDE=INDEX パラメータを使用します。 オリジナルのインポートで INDEXES=y を使用していた場合、そのパラメータは無視されます。その指定はデータ・ポンプ・インポートのデフォルトの動作であるため、再マップする必要はありません。
INDEXFILE	データ・ポンプ・インポートの SQLFILE={directory-object:}filename および INCLUDE=INDEX パラメータが使用されます。 FILE パラメータに対してディレクトリ・オブジェクトを検索する場合の方法と手順が、INDEXFILE パラメータに対しても同様に適用されます。 オリジナルのインポートでディレクトリ・オブジェクトが指定されていなかった場合は、DIRECTORY パラメータで指定されたディレクトリ・オブジェクトがデータ・ポンプ・インポートで使用されます。
LOG	データ・ポンプ・インポートは、LOG パラメータに対する指定パスまたはデフォルト・パスを確認します。また、スキーマに書込みアクセス権があるディレクトリ・オブジェクトが存在するかどうかも確認し

オリジナルのインポート・パラメータ	データ・ポンプ・インポート・パラメータでの処理
	ます。 データ・ポンプでのオリジナルのインポートの LOG パラメータの処理方法の詳細は、「データ・ポンプ・レガシー・モードでのファイルの場所の管理」を参照してください。 ログ・ファイルの内容は、データ・ポンプ・インポート操作の内容になります。ログ・ファイルの場所と内容については、「ログ・ファイル」を参照してください。
RECORDLENGTH	このパラメータは無視されます。データ・ポンプの場合、レコード長に関する問題は内部で処理されます。
RESUMABLE	このパラメータは無視されます。この機能は IMP_FULL_DATABASE ロールを付与されているユーザーに対して自動的に提供されます。
RESUMABLE_NAME	このパラメータは無視されます。この機能は IMP_FULL_DATABASE ロールを付与されているユーザーに対して自動的に提供されます。
RESUMABLE_TIMEOUT	このパラメータは無視されます。この機能は IMP_FULL_DATABASE ロールを付与されているユーザーに対して自動的に提供されます。
ROWS=N	オリジナルのインポートで ROWS=n を使用していた場合、データ・ポンプ・インポートは CONTENT=METADATA_ONLY パラメータを使用します。 オリジナルのインポートで ROWS=y を使用していた場合、データ・ポンプ・インポートは CONTENT=ALL パラメータを使用します。
SHOW	SHOW=y が指定されている場合、データ・ポンプ・インポートの SQLFILE=[directory_object:]file_name パラメータを使用して、インポート操作の DDL がファイルに書き込まれます。指定ファイルには、(ダンプ・ファイルの全体の内容ではなく)DDL のみが書き込まれます。(これは、オリジナルのインポートなので、画面に出力は表示されません。) DUMPFILE パラメータ(または DUMPFILE に再マップされる、オリジナルのインポートの FILE パラメータ)で指定された名前がファイルの名前になります。指定されているダンプ・ファイル名が複数ある場合は、最初に指定されているファイル名が使用されます。ファイルは、DIRECTORY パラメータで指定されたディレクトリ・オブジェクトの場所か、DUMPFILE パラメータに含まれるディレクトリ・オブジェクトに配置されます。(DUMPFILE パラメータで指定されたディレクトリ・オブジェクトが優先されます。)

オリジナルのインポート・パラメータ	データ・ポンプ・インポート・パラメータでの処理
STATISTICS	このパラメータは無視されます。統計は、データ・ポンプ・インポート操作の一環として、表に対して常に保存されます。
STREAMS_CONFIGURATION	このパラメータは無視されます。この値はデータ・ポンプ・インポートによって自動的に決定されるため、指定する必要はありません。
STREAMS_INSTANTIATION	このパラメータは無視されます。この値はデータ・ポンプ・インポートによって自動的に決定されるため、指定する必要はありません。
TABLESPACES	オリジナルのインポートで TRANSPORT_TABLESPACE=n(デフォルト)も指定されていた場合、データ・ポンプ・インポートは TABLESPACES パラメータを無視します。 オリジナルのインポートで TRANSPORT_TABLESPACE=y も指定していた場合、データ・ポンプ・インポートは、この TABLESPACES パラメータに対して指定されていた名前を、データ・ポンプ・インポートの TRANSPORT_TABLESPACES パラメータに適用します。
TOID_NOVALIDATE	このパラメータは無視されます。型比較に、OID は使用されなくなりました。
TOUSER	データ・ポンプ・インポートの REMAP_SCHEMA パラメータが使用されます。オリジナルのインポートより多くのオブジェクトをインポートできます。また、ターゲットのスキーマがまだ存在していない場合は、データ・ポンプ・インポートで作成することもできます。 オリジナルのインポートで FROMUSER パラメータも指定されている必要があります。指定されていない場合は、操作が失敗します。
TRANSPORT_TABLESPACE	TRANSPORT_TABLESPACE パラメータは無視されますが、DATAFILES パラメータも指定した場合、インポート・ジョブはメタデータのロードを続行します。DATAFILES パラメータを指定しない場合、ORA-39002:invalid operation エラー・メッセージが返されます。
TTS_OWNERS	このパラメータは無視されます。この情報はデータ・ポンプのダンプ・ファイル・セットに自動的に保存されます。
VOLSIZE	オリジナルのインポートで VOLSIZE パラメータが使用されている場合、それはダンプ・ファイルの指定場所がテープ・デバイスであることを意味します。テープ・デバイスは、データ・ポンプ・インポートのダンプ・ファイル形式でサポートされていません。したがって、この操作はエラーが発生して終了します。

親トピック: パラメータのマッピング

4.3 データ・ポンプ・レガシー・モードでのファイルの場所の管理

オリジナルのバージョンはクライアントベース、データ・ポンプはサーバーベースであるため、オリジナルのエクスポートとインポート、およびデータ・ポンプ・エクスポートとデータ・ポンプ・インポートは、ダンプ・ファイルとログ・ファイルの書込み/読取りを行える場所によって異なります。

オリジナルのエクスポートおよびインポートでは、FILEパラメータとLOGパラメータを使用して、それぞれダンプ・ファイル名とログ・ファイル名を指定します。これらのファイル名は常にクライアント・システムに対してローカルなファイルを参照し、パスの指定を含めることもできます。

データ・ポンプ・エクスポートおよびデータ・ポンプ・インポートでは、DUMPFILEパラメータとLOGFILEパラメータを使用して、それぞれダンプ・ファイル名とログ・ファイル名を指定します。これらのファイル名は常にサーバー・システムに対してローカルなファイルを参照し、パス情報を含めることはできません。かわりに、ディレクトリ・オブジェクトを使用して、パス情報を間接的に指定します。ディレクトリ・オブジェクトによって定義されるパス値は、サーバーからアクセスできる必要があります。ディレクトリ・オブジェクトは、データ・ポンプ・ジョブ用にDIRECTORYパラメータを使用して指定されます。DUMPFILEパラメータおよびLOGFILEパラメータに渡されるファイル名にディレクトリ・オブジェクトを付加することもできます。データ・ポンプを使用する特権ユーザーは、ディレクトリ・オブジェクトがコマンドラインで指定されていない場合に、デフォルト・ディレクトリ・オブジェクトを使用できます。このデフォルトのディレクトリ・オブジェクトDATA_PUMP_DIRは、インストール時に設定されます。

データ・ポンプのレガシー・モードを有効にし、オリジナルのエクスポートのFILE=filespecパラメータまたはLOG=filespecパラメータ(あるいはその両方)をコマンドラインに指定すると、次の優先順位ルールで、ファイルの場所が決まります。

> ノート:
>
> コマンドラインに FILE パラメータと LOG パラメータの両方を指定した場合は、優先順位ルールがそれぞれのパラメータに個別に適用されます。
>
> また、オリジナルのエクスポートおよびインポートとデータ・ポンプ・エクスポート、データ・ポンプ・インポートのパラメータが混在する場合は、個別のルールがそれぞれ適用されます。たとえば、次のコマンドを指定したとします。
>
> ```
> expdp system FILE=/user/disk/foo.dmp LOGFILE=foo.log DIRECTORY=dpump_dir
> ```
>
> この項で説明したとおり、データ・ポンプのレガシー・モードによるファイル管理ルールが、FILE パラメータに適用されます。通常の(レガシー・モードではない)データ・ポンプのファイル管理ルールは、「ダンプ・ファイル、ログ・ファイルおよびSQL ファイルのデフォルトの位置」で説明されているとおりに、LOGFILE パラメータに適用されます。

1. パスの位置がファイル指定の一部として指定されると、データ・ポンプはエクスポート・ジョブを実行するスキーマからアクセスでき、そのパスの位置がファイル指定でのパスの位置に一致するディレクトリ・オブジェクトの検索を試行します。そのようなディレクトリ・オブジェクトが見つからないと、エラーが返されます。たとえば、USER_DUMP_FILESという名前のサーバーベースのディレクトリ・オブジェクトが'/disk1/user1/dumpfiles/'のパス値とともに定義されていて、このディレクトリ・オブジェクトへの読取り/書込みアクセス権限がhrスキーマに付与されているとします。次のコマンドを実行すると、データ・ポンプはそのパス値に'/disk1/user1/dumpfiles/'が含まれ、そのhrスキーマに読取り/書込みアクセス権限が付与されているサーバーベースのディレクトリ・オブジェクトを探します。

   ```
   expdp hr FILE=/disk1/user1/dumpfiles/hrdata.dmp
   ```

   その場合、データ・ポンプはディレクトリ・オブジェクトUSER_DUMP_FILESを使用します。この例で使用されているパス

値'/disk1/user1/dumpfiles/'は、Oracle Databaseからアクセスできるサーバー・システム上のパスを参照している必要があります。

パスの位置がファイル指定の一部として指定されている場合は、DIRECTORYパラメータで指定されたディレクトリ・オブジェクトが無視されます。たとえば、次のコマンドを実行すると、データ・ポンプはファイル・パラメータのDPUMP_DIRディレクトリ・オブジェクトを使用せず、かわりにそのパス値に'/disk1/user1/dumpfiles/'が含まれ、そのhrスキーマに読取り/書込みアクセス権限が付与されているサーバーベースのディレクトリ・オブジェクトを探します。

```
expdp hr FILE=/disk1/user1/dumpfiles/hrdata.dmp DIRECTORY=dpump_dir
```

2. パスの位置がファイル指定の一部として指定されない場合は、DIRECTORYパラメータに指定されたディレクトリ・オブジェクトが使用されます。たとえば、次のコマンドを実行すると、データ・ポンプはDPUMP_DIRディレクトリ・オブジェクト用に定義されたパスの位置をhrdata.dmpファイルに適用します。

```
expdp hr FILE=hrdata.dmp DIRECTORY=dpump_dir
```

3. パスの位置がファイル指定の一部として指定されず、DIRECTORYパラメータにディレクトリ・オブジェクトが指定されていない場合は、データ・ポンプによって次のように動作が順次実行されます。

   a. データ・ポンプはDATA_PUMP_DIR_schema_nameという形式のディレクトリ・オブジェクトを探します。ここでのschema_nameは、データ・ポンプ・ジョブを実行するスキーマです。たとえば、次のコマンドを実行すると、データ・ポンプはDATA_PUMP_DIR_HRという名前のサーバーベースのディレクトリ・オブジェクトを探します。

   ```
   expdp hr FILE=hrdata.dmp
   ```

   また、hrスキーマには、このディレクトリ・オブジェクトへの読取り/書込みアクセス権限が付与されている必要があります。そのようなディレクトリ・オブジェクトが存在しないとき、処理はステップbに進みます。

   b. データ・ポンプは、クライアントベースの環境変数DATA_PUMP_DIRを探します。たとえば、DUMP_FILES1という名前のサーバーベースのディレクトリ・オブジェクトが定義されていて、hrスキーマにそのオブジェクトへの読取り/書込みアクセス権限が付与されているとします。その場合は、クライアント・システムで、次のようにDUMP_FILES1を指すように環境変数DATA_PUMP_DIRを設定します。

   ```
   setenv DATA_PUMP_DIR DUMP_FILES1
   expdp hr FILE=hrdata.dmp
   ```

   データ・ポンプは、次にサーバーベースのディレクトリ・オブジェクトDUMP_FILES1をhrdata.dmpファイルに対して使用します。

   クライアントベースの環境変数DATA_PUMP_DIRが存在しないとき、処理はステップcに進みます。

   c. データ・ポンプ・ジョブを実行するスキーマにDBA権限があると、デフォルトのデータ・ポンプ・ディレクトリ・オブジェクトDATA_PUMP_DIRが使用されます。このデフォルトのディレクトリ・オブジェクトは、インストール時に設定されます。たとえば、次のコマンドを実行すると、データ・ポンプはシステムにDBA権限があるとみなし、デフォルトのDATA_PUMP_DIRディレクトリ・オブジェクトを使用しようとします。

   ```
   expdp system FILE=hrdata.dmp
   ```

関連項目:

通常モードのデータ・ポンプ(レガシー・モードではない)でのデータ・ポンプのファイル管理優先順位ルールの詳細は、「ダンプ・ファイル、ログ・ファイルおよびSQLファイルのデフォルトの位置」を参照してください

親トピック: Oracle Data Pumpのレガシー・モード

4.4 Oracle Data Pumpのログ・ファイルとエラーに対する既存のスクリプトの調整

Oracle Data Pumpをレガシー・モードで使用する場合は、オリジナルのエクスポートおよびインポート用に作成された既存のスクリプトを確認し、更新する必要があります

Oracle Data Pumpのレガシー・モードでは、ファイル形式とエラー報告が異なるため、既存のスクリプトを調整する必要があります。

- ログ・ファイル
  Oracle Data Pumpのエクスポートおよびインポートで作成されるログ・ファイルは、オリジナルのエクスポートおよびインポートとは形式が異なります。
- エラー・ケース
  Oracle Data Pumpエクスポートおよびインポートで生成されるエラーは、オリジナルのエクスポートおよびインポートによって生成されるエラーとは異なる場合があります。
- 終了ステータス
  Oracle Data Pumpエクスポートおよびインポートには、エクスポート・ジョブとインポート・ジョブの成功または失敗をスクリプトでより適切に判定できるようにするための、拡張された終了状態の値があります。

親トピック: Oracle Data Pumpのレガシー・モード

4.4.1 ログ・ファイル

Oracle Data Pumpのエクスポートおよびインポートで作成されるログ・ファイルは、オリジナルのエクスポートおよびインポートとは形式が異なります。

オリジナルのエクスポートおよびインポートの出力を解析するスクリプトを更新して、Oracle Data Pumpエクスポートおよびインポートで使用されるログ・ファイル形式を処理する必要があります。たとえば、「`Successfully Terminated`」というメッセージは、Oracle Data Pumpのログ・ファイルには表示されません。

親トピック: Oracle Data Pumpのログ・ファイルとエラーに対する既存のスクリプトの調整

4.4.2 エラー・ケース

Oracle Data Pumpエクスポートおよびインポートで生成されるエラーは、オリジナルのエクスポートおよびインポートによって生成されるエラーとは異なる場合があります。

たとえば、Oracle Data Pumpエクスポートによって無視されるパラメータが、オリジナルのエクスポートで範囲外の値を生成するとします。その場合、パラメータが無視されていることを示す情報メッセージがログ・ファイルに書き込まれます。ただし、値のチェックは実行されないため、エラー・メッセージは生成されません。

親トピック: Oracle Data Pumpのログ・ファイルとエラーに対する既存のスクリプトの調整

4.4.3 終了状態

Oracle Data Pumpエクスポートおよびインポートには、エクスポート・ジョブとインポート・ジョブの成功または失敗をスクリプトでより適切に判定できるようにするための、拡張された終了状態の値があります。

Oracle Data Pumpのエクスポートとインポートの終了ステータス値が異なる可能性があるため、終了ステータスを確認するスクリプトを調べて、必要に応じて更新することをお薦めします。

親トピック: Oracle Data Pumpのログ・ファイルとエラーに対する既存のスクリプトの調整

5 Oracle Data Pumpのパフォーマンス

Oracle Data Pumpエクスポートおよびインポートがオリジナルのエクスポートおよびインポート・ユーティリティよりも優れている点と、エクスポート操作およびインポート操作のパフォーマンスを向上させる方法を学習します。

Oracle Data Pumpエクスポートおよびインポート・ユーティリティは、大規模データベースを対象に設計されています。大量のデータおよびメタデータがある場合は、オリジナルのエクスポート・ユーティリティおよびインポート・ユーティリティと比較して、データのパフォーマンスが向上します。(データ・ポンプ・エクスポート・ユーティリティおよびデータ・ポンプ・インポート・ユーティリティでのメタデータ抽出およびデータベース・オブジェクト作成のパフォーマンスは、基本的にはオリジナルのエクスポート・ユーティリティおよびインポート・ユーティリティのパフォーマンスと同様です。)

- Oracle Data Pumpエクスポートおよびインポートのデータ・パフォーマンスの改善点
  Oracle Data Pumpエクスポート(expdp)およびインポート(impdp)には、従来のエクスポート(exp)およびインポート(imp)と比べてパフォーマンスが向上した多数の機能が含まれています。
- パフォーマンスのチューニング
  Oracle Data Pumpは、すべての使用可能なリソースを最大限に活用し、スループットを最大化してジョブ経過時間を最小化するように設計されています。
- データ・ポンプのパフォーマンスに影響する初期化パラメータ
  特定のOracle Database初期化パラメータの設定が、データ・ポンプ・エクスポートおよびデータ・ポンプ・インポートのパフォーマンスに影響する場合があります。

親トピック: Oracle Data Pump

5.1 Oracle Data Pumpエクスポートおよびインポートのデータ・パフォーマンスの改善点

Oracle Data Pumpエクスポート(expdp)およびインポート(impdp)には、従来のエクスポート(exp)およびインポート(imp)と比べてパフォーマンスが向上した多数の機能が含まれています。

次に、データ・ポンプ・エクスポートおよびインポート・ユーティリティでのパフォーマンス向上の要因を示します。

- 複数のワーカー・プロセスで、表間およびパーティション間のパラレル処理を実行して、複数のパラレル・ダイレクト・パス・ストリームで表をロードおよびアンロードできます。
- 非常に大きい表およびパーティションの場合は、単一のワーカー・プロセスで、外部表による方法を使用してデータにアクセスすると、複数のパラレル問合せおよびパラレルDML I/Oサーバー・プロセスを介してパーティション間のパラレル処理を選択できます。
- Oracle Data Pumpでは、索引の作成およびパッケージ本体のロードに並列処理を使用します。
- ダンプ・ファイルの読取りと書込みはサーバーで直接実行されるため、クライアントにデータを移動する必要はありません。
- ダンプ・ファイルの格納形式は、ダイレクト・パスAPIの内部ストリーム形式です。この形式は、表領域内部のOracle Databaseデータ・ファイルに格納されている形式とほとんど同じです。したがって、クライアント側ではINSERT文バインド変数への変換は実行されません。
- サポートされているデータ・アクセス方法(ダイレクト・パスおよび外部表による方法)は従来のSQLより高速です。ダイレクト・パスAPIを使用すると、単一ストリームでの最大のパフォーマンスが実現します。外部表機能を使用すると、Oracle Databaseのパラレル問合せ機能とパラレルDML機能を有効活用できます。

- エクスポート時に、メタデータとデータの抽出を同時に実行できます。

親トピック: Oracle Data Pumpのパフォーマンス

5.2 パフォーマンスのチューニング

Oracle Data Pumpは、すべての使用可能なリソースを最大限に活用し、スループットを最大化してジョブ経過時間を最小化するように設計されています。

使用可能なリソースを最大化するために、システムではCPU、メモリーおよびI/Oの全体的なバランスが取られている必要があります。また、パフォーマンス・チューニングの標準原理が適用されます。たとえば、ダンプ・ファイルの書込みおよび読取りは並列処理されるため、パフォーマンスを最大化するには、ダンプ・ファイル・セットに含まれるファイルを個別のディスクに配置するようにします。また、それらのダンプ・ファイルは、ソースまたはターゲットの表領域が常駐するディスクとは別のディスクに常駐させる必要があります。

パフォーマンス・チューニングを行う場合は、パフォーマンスとリソース消費のバランスをとることが必要です。

- リソース消費の制御
  データ・ポンプ・エクスポートおよびインポート・ユーティリティを使用すると、ジョブごとのリソース消費量を動的に増減できます。
- 圧縮および暗号化によるパフォーマンスへの影響
  Oracle Data Pumpの圧縮と暗号化に関連するパラメータを使用すると、特にネットワーク・モードで実行されるジョブのパフォーマンスを向上できます。
- 統計のエクスポートおよびインポートにおけるメモリーの考慮事項
  リリース12.1より前のリリースで作成したOracle Data Pumpエクスポートのダンプ・ファイルを使用するときに、大量の統計データが含まれていると、インポート操作中に大量のメモリー要求が発生することがあります。

親トピック: Oracle Data Pumpのパフォーマンス

5.2.1 リソース消費の制御

データ・ポンプ・エクスポートおよびインポート・ユーティリティを使用すると、ジョブごとのリソース消費量を動的に増減できます。

これを実行するには、データ・ポンプのPARALLELパラメータを使用してそのジョブの並列度を指定します。最大のスループットを得るには、PARALLELをCPU数の2倍(CPUごとに2つのワーカー)以下に設定してください。

並列度を増加すると、CPU使用量、メモリー使用量およびI/O帯域幅使用も増大します。これらのリソースの使用可能な量が適切であることを確認してください。必要に応じて、異なるディスク・デバイスまたはチャネル間にファイルを分散して、必要なI/O帯域幅を確保できます。

並列度を最大にするには、並列度ごとに少なくとも1つのファイルを提供する必要があります。これを簡単に行うには、たとえば`file%u.dmp`のように、ファイル名に置換変数を使用します。ただし、ディスク設定によっては(たとえば、単純な非ストライプ・ディスクなどの場合)、すべてのダンプ・ファイルを1つのデバイスに配置しない場合があります。その場合は、置換変数を使用して複数のファイル名を指定し、それぞれのファイルを別々のディレクトリ(別々のディスク)に配置します。高速CPUと高速ディスクを使用する場合でも、CPUとディスク間のパスは、持続可能な並列度の量を制約する要因になることがあります。

データ・ポンプのPARALLELパラメータは、Oracle Database 11g以上のEnterprise Editionでのみ有効です。

親トピック: パフォーマンスのチューニング

5.2.2 圧縮および暗号化によるパフォーマンスへの影響

Oracle Data Pumpの圧縮と暗号化に関連するパラメータを使用すると、特にネットワーク・モードで実行されるジョブのパフォーマンスを向上できます。

パフォーマンスのチューニングを試行しているときには、リソースの可用性に注意してください。圧縮と暗号化がパフォーマンスに悪影響を与えることがあります。これは、RAWデータの変換を実行するために余分なCPUリソースが必要になるためです。両者はトレードオフの関係にあります。

親トピック: パフォーマンスのチューニング

5.2.3 統計のエクスポートおよびインポートにおけるメモリーの考慮事項

リリース12.1より前のリリースで作成したOracle Data Pumpエクスポートのダンプ・ファイルを使用するときに、大量の統計データが含まれていると、インポート操作中に大量のメモリー要求が発生することがあります。

インポート操作時のメモリーの不足を回避するには、インポートの開始前に十分にメモリーを割り当てる必要があります。必要とされるメモリーの正確な量は、インポートするデータの量、使用しているプラットフォーム、または使用する構成に固有の要素によって異なります。

この問題にまとめて対処するには、エクスポートまたはインポート操作のいずれかの際にデータ・ポンプの`EXCLUDE=STATISTICS`パラメータを設定します。ターゲット・データベースで統計を再生成する場合は、インポートの完了後に`DBMS_STATS` PL/SQLパッケージを使用できます。

関連トピック

- EXCLUDE
- EXCLUDE
- Oracle Database SQLチューニング・ガイド

親トピック: パフォーマンスのチューニング

5.3 データ・ポンプ・ユーティリティのパフォーマンスに影響する初期化パラメータ

特定のOracle Database初期化パラメータの設定が、データ・ポンプ・エクスポートおよびデータ・ポンプ・インポートのパフォーマンスに影響する場合があります。

特に、次の設定を使用してパフォーマンスを改善できます。ただし、プラットフォームによっては同様の効果を得られない場合もあります。

- `DISK_ASYNCH_IO=TRUE`
- `DB_BLOCK_CHECKING=FALSE`
- `DB_BLOCK_CHECKSUM=FALSE`

次の初期化パラメータには、並列度が最大になる値を指定する必要があります。

- `PROCESSES`
- `SESSIONS`
- `PARALLEL_MAX_SERVERS`

さらに、初期化パラメータ`SHARED_POOL_SIZE`と`UNDO_TABLESPACE`は、余裕のある大きさにする必要があります。具体的な値は、データベースのサイズによって異なります。

- GoldenGateレプリケーション環境におけるバッファ・キャッシュのサイズの設定
  Oracle Data Pumpでは、プロセス間の通信にGoldenGateレプリケーション機能を使用します。
- 複数ユーザーによるデータ・ポンプ・ジョブのリソース使用率の管理
  同一のデータベース環境でデータ・ポンプ・ジョブを実行するユーザーが複数いる場合は、初期化パラメータのMAX_DATAPUMP_JOBS_PER_PDBとMAX_DATAPUMP_PARALLEL_PER_JOBを使用すると、リソースの使用率を詳細に制御できます。

親トピック: Oracle Data Pumpのパフォーマンス

5.3.1 GoldenGateレプリケーション環境におけるバッファ・キャッシュのサイズの設定

Oracle Data Pumpでは、プロセス間の通信にGoldenGateレプリケーション機能を使用します。

SGA_TARGET初期化パラメータが設定されていると、STREAMS_POOL_SIZE初期化パラメータは自動的に合理的な値に設定されます。

SGA_TARGET初期化パラメータが設定されていない状態で、STREAMS_POOL_SIZE初期化パラメータも定義されていない場合は、ストリーム・プールのサイズは自動的に共有プール・サイズの10%(デフォルト)になります。

ストリーム・プールが作成されると、バッファ・キャッシュに割り当てられたメモリーから必要なSGAメモリーが確保されるため、キャッシュのサイズは、DB_CACHE_SIZE初期化パラメータで指定したサイズよりも少なくなります。つまり、バッファ・キャッシュが必要最小限のSGAで構成されていた場合、データ・ポンプ操作は正しく動作しません。データ・ポンプ操作を正常に実行するために、STREAMS_POOL_SIZEの値は、最小サイズの10MBにすることをお薦めします。

親トピック: データ・ポンプ・ユーティリティのパフォーマンスに影響する初期化パラメータ

5.3.2 複数ユーザーによるデータ・ポンプ・ジョブのリソース使用率の管理

同一のデータベース環境でデータ・ポンプ・ジョブを実行するユーザーが複数いる場合は、初期化パラメータのMAX_DATAPUMP_JOBS_PER_PDBとMAX_DATAPUMP_PARALLEL_PER_JOBを使用すると、リソースの使用率を詳細に制御できます。

初期化パラメータMAX_DATAPUMP_JOBS_PER_PDBでは、プラガブル・データベース(PDB)ごとの同時Oracle Data Pumpジョブの最大数を決定します。Oracle Database 19c以降のリリースでは、このパラメータをAUTOに設定できます。この設定は、SESSIONS初期化パラメータ値の50パーセント(50%)になるように、Oracle Data PumpがMAX_DATAPUMP_JOBS_PER_PDBの実際の値を導出することを意味します。この値をAUTOに設定しない場合、デフォルト値は100です。この値は0から250にまで設定できます。

Oracle Databaseリリース19c以降には、初期化パラメータMAX_DATAPUMP_PARALLEL_PER_JOBがあります。特定のデータベース環境で複数のユーザーが同時にデータ・ポンプ・ジョブを実行する場合は、このパラメータを使用するとリソースの使用率を詳細に制御できます。パラメータMAX_DATAPUMP_PARALLEL_PER_JOBでは、Oracle Data Pumpジョブごとに使用可能にするパラレル処理の最大数を指定します。具体的なプロセスの最大数を指定することも、AUTOを選択することもできます。設定値を指定する場合、この最大数は1から1024に設定できます(デフォルトは、1024 です)。AUTOを指定すると、Oracle Data PumpはSESSIONS初期化パラメータの値の25パーセント(25%)になるようにMAX_DATAPUMP_PARALLEL_PER_JOBの実際の値を導出します。

関連トピック

- MAX_DATAPUMP_JOBS_PER_PDB、『Oracle Databaseリファレンス』
- MAX_DATAPUMP_PARALLEL_PER_JOB、『Oracle Databaseリファレンス』

親トピック: データ・ポンプ・ユーティリティのパフォーマンスに影響する初期化パラメータ

6 Oracle Data Pump APIの使用

データ移動の操作は、Oracleデータ・ポンプPL/SQL APIのDBMS_DATAPUMPを使用することで自動化できます。

Oracleデータ・ポンプAPIのDBMS_DATAPUMPは、あるサイトのデータとメタデータのすべてまたは一部をOracle Database間で移動するために使用できる高速なメカニズムを提供します。Oracleデータ・ポンプ・エクスポート・ユーティリティとOracleデータ・ポンプ・インポート・ユーティリティは、Oracleデータ・ポンプAPIに基づいています。

『*Oracle Database PL/SQLパッケージおよびタイプ・リファレンス*』

- Oracleデータ・ポンプのクライアント・インタフェースAPIの動作
  クライアント・インタフェースで使用されている主な構造体はジョブ・ハンドルで、コール元に対しては整数として表示されます。
- Oracleデータ・ポンプAPIを使用する場合の基本ステップ
  Oracleデータ・ポンプAPIを使用するには、DBMS_DATAPUMPパッケージで提供されるプロシージャを使用します。
- データ・ポンプAPIの使用例
  データ・ポンプAPIを使用します。

関連トピック

- Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス

親トピック: Oracle Data Pump

6.1 Oracleデータ・ポンプのクライアント・インタフェースAPIの動作

クライアント・インタフェースで使用されている主な構造体はジョブ・ハンドルで、コール元に対しては整数として表示されます。

ハンドルは、DBMS_DATAPUMP.OPENまたはDBMS_DATAPUMP.ATTACHファンクションを使用して作成します。他のセッションをジョブに接続してその進捗状況を監視および制御できます。ハンドルはセッション固有です。同じジョブによって、セッションごとに異なるハンドルを作成できます。DBAとして、この機能には仕事を離れる前にジョブを開始し、自宅からジョブの進行状況を見ることができる利点があります。

- DBMS_DATAPUMPジョブの状態
  Oracleデータ・ポンプのDBMS_DATAPUMPジョブの状態を使用して、データ移動ジョブが実行しているステージと、各ステージで使用可能なオプションを確認します。

親トピック: Oracle Data Pump APIの使用

6.1.1 DBMS_DATAPUMPジョブの状態

Oracleデータ・ポンプのDBMS_DATAPUMPジョブの状態を使用して、データ移動ジョブが実行しているステージと、各ステージで使用可能なオプションを確認します。

ジョブ状態の定義

ジョブの各フェーズは、次に示すように1つの状態に関連付けられています。

- 未定義 — ハンドル作成前
- 定義中 — ハンドルの最初の作成時
- 実行中 — DBMS_DATAPUMP.START_JOBプロシージャの実行時

- 完了中 ― ジョブがその作業を完了し、Oracle Data Pumpプロセスを終了中
- 完了 ― ジョブの完了時
- 停止保留中 ― 手順に従ったジョブの停止が要求された場合
- 停止中 ― ジョブの停止処理を実行中
- アイドル ― 停止しているジョブに接続するためにDBMS_DATAPUMP.ATTACHが実行されてから、そのジョブを再開するためのDBMS_DATAPUMP.START_JOBが実行されるまでの期間
- 未実行 ― 実行されていない(関連付けられたOracle Data Pumpプロセスが存在しない)ジョブに対してData Pump制御ジョブ表が存在する状態

使用上のノート

アイドル状態のジョブに対してDBMS_DATAPUMP.START_JOBを実行すると、そのジョブは実行中状態に戻ります。

すべてのユーザーがDBMS_DATAPUMP.DETACHを実行して定義中状態のジョブとの接続を解除すると、そのジョブはデータベースから完全に削除されます。

ジョブが予期せず終了した場合やジョブを実行しているインスタンスが停止された場合、実行中状態またはアイドル状態であったジョブは未実行状態になります。その後、ジョブを再起動できます。

Oracle Data Pump制御ジョブ・プロセスは、「定義」、「アイドル」、「実行中」、「停止中」、「停止保留中」および「完了」状態でアクティブです。また、一時的に停止および完了の状態でもアクティブになります。ジョブのData Pump制御表は、未定義状態を除いてすべての状態で存在します。子プロセスは、実行中状態と停止保留中状態でのみアクティブになります。また、インポート・ジョブのワーカー・プロセスは、一時的に定義中状態でもアクティブになります。

ジョブが実行中状態のときに接続解除しても、そのジョブは停止しません。いつでも実行中のジョブに再接続して、そのジョブに関する状態情報の取得を再開できます。

DBMS_DATAPUMP.DETACHプロシージャが実行されると、明示的な接続解除が発生します。また、Oracle Data Pump APIセッションが停止した場合、Oracle Data Pump APIがOracle Data Pumpジョブと通信できない場合、またはDBMS_DATAPUMP.STOP_JOBプロシージャが実行された場合は、暗黙的な接続解除が発生します。

「未実行」状態は、Data Pump制御ジョブ表が実行中のジョブのコンテキスト外に存在することを示します。この状態は、ジョブを停止した(通常は後で再開する)場合や、ジョブが異常な方法で終了した場合に発生します。また、この状態は、ジョブの開始時とOracle Data Pump制御ジョブ表を削除するまでのジョブの終了時に発生するジョブ状態の移行中に一時的に確認されることがあります。未実行状態は、DBA_DATAPUMP_JOBSビューとUSER_DATAPUMP_JOBSビューにのみ表示されます。GET_STATUSプロシージャから返されることはありません。

次の表に、DBMS_DATAPUMPプロシージャを実行できる有効なジョブ状態を示します。この表に示す状態は、特に指定がないかぎり、エクスポートとインポートの両方で有効です。

表6-1 DBMS_DATAPUMPプロシージャを実行できる有効なジョブの状態

プロシージャ名	有効な状態	説明
ADD_FILE	定義中(エクスポート・ジョブとインポート・ジョブの両方で有効) 実行中およびアイドル(エクスポート・ジョブでダンプ・ファイルを指定する場合	ダンプ・ファイル・セット、ログ・ファイルまたは SQLFILE 出力用のファイルを指定します。

プロシージャ名	有効な状態	説明
	にのみ有効)	
ATTACH	定義中、実行中、アイドル、停止、完了、完了中、未実行	ユーザー・セッションで、ジョブの監視または停止したジョブの再開を可能にします。ジョブのダンプ・ファイル・セットまたは Data Pump 制御ジョブ表が削除されている場合や、なんらかの方法で変更されている場合、接続操作は失敗します。
DATA_FILTER	定義中	ジョブが処理するデータを制限する。
DETACH	すべて	ユーザー・セッションをジョブから切断する。
GET_DUMPFILE_INFO	すべて	ダンプ・ファイルのヘッダー情報を取得する。
GET_STATUS	完了、未実行、停止および未定義を除くすべての状態	ジョブの状態を取得する。
LOG_ENTRY	定義中、実行中、アイドル、停止保留中、完了処理中	ログ・ファイルにエントリを追加する。
METADATA_FILTER	定義中	ジョブが処理するメタデータを制限する。
METADATA_REMAP	定義中	ジョブが処理するメタデータを再マップする。
METADATA_TRANSFORM	定義中	ジョブが処理するメタデータを変更する。
OPEN	未定義	新しいジョブを作成する。
SET_PARALLEL	定義中、実行中、アイドル	ジョブの並列度を指定する。
SET_PARAMETER	定義中 ノート: ENCRYPTION_PASSWORD パラメータは、状態が定義中およびアイドルのときに入力できます。	ジョブのデフォルトの処理を変更する。
START_JOB	定義中、アイドル	ジョブを開始または再開する。

プロシージャ名	有効な状態	説明
STOP_JOB	定義中、実行中、アイドル、停止保留中	ジョブの停止を開始する。
WAIT_FOR_JOB	完了、未実行、停止および未定義を除くすべての状態	ジョブの終了を待機する。

親トピック: Oracle Data Pumpのクライアント・インタフェースAPIの動作

6.2 Oracleデータ・ポンプAPIを使用する場合の基本ステップ

Oracleデータ・ポンプAPIを使用するには、DBMS_DATAPUMPパッケージで提供されるプロシージャを使用します。

次のステップでは、オプションのステップを実行できるOracleデータ・ポンプ・ジョブの実行時点を含む、データ・ポンプAPIの使用に関連する基本的なアクティビティを示します。これらのステップは、通常アクティビティを実行する順序で表示されます。

1. Oracleデータ・ポンプ・ジョブとそのインフラストラクチャを作成するには、DBMS_DATAPUMP.OPENプロシージャを実行します。
   プロシージャを実行すると、Oracleデータ・ポンプ・ジョブが開始されます。
2. ジョブで使用するパラメータを定義します。
3. ジョブを開始します。
4. (オプション)ジョブを完了まで監視できます。
5. (オプション)ジョブとの接続を切断し、後で再接続します。
6. (オプション)ジョブを停止します。
7. (オプション)ジョブを再開します。

関連トピック

- Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス

親トピック: Oracle Data Pump APIの使用

6.3 データ・ポンプAPIの使用例

データ・ポンプAPIを使用します。

この項では、データ・ポンプAPIの使用方法を理解するために有効な例を示します。

- 簡単なスキーマ・エクスポートの実行(例6-1)
- ダンプ・ファイルのインポートおよびすべてのスキーマ・オブジェクトの再マップ(例6-2)
- 簡単なスキーマ・エクスポート実行中の例外処理機能の使用方法(例6-3)
- ダンプ・ファイルの情報の表示(例6-4)

これらの例は、PL/SQLスクリプトの形式で記述します。これらのスクリプトをコピーして実行するには、まず、SQL*Plusを使用して次の操作を実行する必要があります。

- ディレクトリ・オブジェクトを作成し、そのオブジェクトへのREADおよびWRITE権限を付与します。たとえば、権限を持つdmpdirというディレクトリ・オブジェクトを作成するには、次のように行います。userは、ユーザー名に置き換えます。

```
SQL> CREATE DIRECTORY dmpdir AS '/rdbms/work';
SQL> GRANT READ, WRITE ON DIRECTORY dmpdir TO user;
```

- EXP_FULL_DATABASEロールおよびIMP_FULL_DATABASEロールを所有していることを確認します。セキュリティ・ドメイン内でユーザー自身に割り当てられているすべてのロールのリストを表示するには、次のように行います。

  ```
  SQL> SELECT * FROM SESSION_ROLES;
  ```

  必要なロールが割り当てられていない場合は、システム管理者に連絡してください。

- サーバーの出力がオンになっていない場合はオンにします。次のように行います。

  ```
  SQL> SET SERVEROUTPUT ON
  ```

  これを行わなかった場合は、画面に出力が表示されません。この操作は、例を実行するセッションと同じセッションで行う必要があります。この設定は、SQL*Plusを終了すると失われるため、新しいセッションの開始時に再設定する必要があります。(別のユーザー名に接続する場合も、再設定する必要があります。)

例6-1 簡単なスキーマ・エクスポートの実行

次の例のPL/SQLスクリプトは、データ・ポンプAPIを使用してHRスキーマのスキーマ・エクスポートを簡単に実行する方法を示しています。ジョブの作成、開始、監視の方法を示しています。この例の詳細は、スクリプト内のコメントを参照してください。例を簡単にしておくために、APIのコールでの例外は検出されません。ただし、本番環境でエラーが発生した場合は、例外ハンドラを定義してGET_STATUSをコールし、エラー情報の詳細を取得することをお薦めします。

このスクリプトを使用するには、ユーザーSYSTEMとして接続します。

```
DECLARE
  ind NUMBER;               -- Loop index
  h1 NUMBER;                -- Data Pump job handle
  percent_done NUMBER;      -- Percentage of job complete
  job_state VARCHAR2(30);   -- To keep track of job state
  le ku$_LogEntry;          -- For WIP and error messages
  js ku$_JobStatus;         -- The job status from get_status
  jd ku$_JobDesc;           -- The job description from get_status
  sts ku$_Status;           -- The status object returned by get_status
BEGIN
-- Create a (user-named) Data Pump job to do a schema export.
  h1 := DBMS_DATAPUMP.OPEN('EXPORT','SCHEMA',NULL,'EXAMPLE1','LATEST');
-- Specify a single dump file for the job (using the handle just returned)
-- and a directory object, which must already be defined and accessible
-- to the user running this procedure.
  DBMS_DATAPUMP.ADD_FILE(h1,'example1.dmp','DMPDIR');
-- A metadata filter is used to specify the schema that will be exported.
  DBMS_DATAPUMP.METADATA_FILTER(h1,'SCHEMA_EXPR','IN (''HR'')');
-- Start the job. An exception will be generated if something is not set up
-- properly.
  DBMS_DATAPUMP.START_JOB(h1);
-- The export job should now be running. In the following loop, the job
-- is monitored until it completes. In the meantime, progress information is
-- displayed.

  percent_done := 0;
  job_state := 'UNDEFINED';
  while (job_state != 'COMPLETED') and (job_state != 'STOPPED') loop
    dbms_datapump.get_status(h1,
           dbms_datapump.ku$_status_job_error +
           dbms_datapump.ku$_status_job_status +
           dbms_datapump.ku$_status_wip,-1,job_state,sts);
    js := sts.job_status;
-- If the percentage done changed, display the new value.
    if js.percent_done != percent_done
```

```
    then
      dbms_output.put_line('*** Job percent done = ' ||
                           to_char(js.percent_done));
      percent_done := js.percent_done;
    end if;
-- If any work-in-progress (WIP) or error messages were received for the job,
-- display them.
   if (bitand(sts.mask,dbms_datapump.ku$_status_wip) != 0)
    then
      le := sts.wip;
    else
      if (bitand(sts.mask,dbms_datapump.ku$_status_job_error) != 0)
      then
        le := sts.error;
      else
        le := null;
      end if;
    end if;
    if le is not null
    then
      ind := le.FIRST;
      while ind is not null loop
        dbms_output.put_line(le(ind).LogText);
        ind := le.NEXT(ind);
      end loop;
    end if;
  end loop;
-- Indicate that the job finished and detach from it.
  dbms_output.put_line('Job has completed');
  dbms_output.put_line('Final job state = ' || job_state);
  dbms_datapump.detach(h1);
END;
/
```

例6-2 ダンプ・ファイルのインポートおよびすべてのスキーマ・オブジェクトの再マップ

この例のスクリプトでは、例6-1(hrスキーマのエクスポート)で作成したダンプ・ファイルをインポートします。すべてのスキーマ・オブジェクトがhrスキーマからblakeスキーマに再マップされます。例を簡単にしておくために、APIのコールでの例外は検出されません。ただし、本番環境でエラーが発生した場合は、例外ハンドラを定義してGET_STATUSをコールし、エラー情報の詳細を取得することをお薦めします。

このスクリプトを使用するには、ユーザーSYSTEMとして接続します。

```
DECLARE
  ind NUMBER;               -- Loop index
  h1 NUMBER;                -- Data Pump job handle
  percent_done NUMBER;      -- Percentage of job complete
  job_state VARCHAR2(30);   -- To keep track of job state
  le ku$_LogEntry;          -- For WIP and error messages
  js ku$_JobStatus;         -- The job status from get_status
  jd ku$_JobDesc;           -- The job description from get_status
  sts ku$_Status;           -- The status object returned by get_status
BEGIN
-- Create a (user-named) Data Pump job to do a "full" import (everything
-- in the dump file without filtering).
  h1 := DBMS_DATAPUMP.OPEN('IMPORT','FULL',NULL,'EXAMPLE2');
-- Specify the single dump file for the job (using the handle just returned)
-- and directory object, which must already be defined and accessible
-- to the user running this procedure. This is the dump file created by
-- the export operation in the first example.
  DBMS_DATAPUMP.ADD_FILE(h1,'example1.dmp','DMPDIR');
-- A metadata remap will map all schema objects from HR to BLAKE.
  DBMS_DATAPUMP.METADATA_REMAP(h1,'REMAP_SCHEMA','HR','BLAKE');
-- If a table already exists in the destination schema, skip it (leave
-- the preexisting table alone). This is the default, but it does not hurt
```

```
-- to specify it explicitly.
  DBMS_DATAPUMP.SET_PARAMETER(h1,'TABLE_EXISTS_ACTION','SKIP');
-- Start the job. An exception is returned if something is not set up properly.
  DBMS_DATAPUMP.START_JOB(h1);
-- The import job should now be running. In the following loop, the job is
-- monitored until it completes. In the meantime, progress information is
-- displayed. Note: this is identical to the export example.

 percent_done := 0;
  job_state := 'UNDEFINED';
  while (job_state != 'COMPLETED') and (job_state != 'STOPPED') loop
    dbms_datapump.get_status(h1,
           dbms_datapump.ku$_status_job_error +
           dbms_datapump.ku$_status_job_status +
           dbms_datapump.ku$_status_wip,-1,job_state,sts);
    js := sts.job_status;
-- If the percentage done changed, display the new value.
     if js.percent_done != percent_done
    then
      dbms_output.put_line('*** Job percent done = ' ||
                           to_char(js.percent_done));
      percent_done := js.percent_done;
    end if;
-- If any work-in-progress (WIP) or Error messages were received for the job,
-- display them.
       if (bitand(sts.mask,dbms_datapump.ku$_status_wip) != 0)
    then
      le := sts.wip;
    else
      if (bitand(sts.mask,dbms_datapump.ku$_status_job_error) != 0)
      then
        le := sts.error;
      else
        le := null;
      end if;
    end if;
    if le is not null
    then
      ind := le.FIRST;
      while ind is not null loop
        dbms_output.put_line(le(ind).LogText);
        ind := le.NEXT(ind);
      end loop;
    end if;
  end loop;
-- Indicate that the job finished and gracefully detach from it.
  dbms_output.put_line('Job has completed');
  dbms_output.put_line('Final job state = ' || job_state);
  dbms_datapump.detach(h1);
END;
/
```

例6-3 簡単なスキーマ・エクスポート実行中の例外処理機能の使用方法

この例のスクリプトでは、データ・ポンプAPIを使用した簡単なスキーマ・エクスポートを示します。例6-1の延長で、例外処理機能を使用してSUCCESS_WITH_INFOケースを捕捉する方法およびGET_STATUSプロシージャを使用したエラーの詳細を取得する方法を示します。DBMS_DATAPUMP.OPENまたはDBMS_DATAPUMP.ATTACHエラーの例外情報を取得する場合は、DBMS_DATAPUMP.KU$_STATUS_JOB_ERROR情報マスクとNULLジョブ・ハンドルを使用してDBMS_DATAPUMP.GET_STATUSをコールし、エラー情報の詳細を取得します。

この例を使用するには、ユーザーSYSTEMとして接続します。

```
DECLARE
  ind NUMBER;               -- Loop index
```

```
  spos NUMBER;             -- String starting position
  slen NUMBER;             -- String length for output
  h1 NUMBER;               -- Data Pump job handle
  percent_done NUMBER;     -- Percentage of job complete
  job_state VARCHAR2(30);  -- To keep track of job state
  le ku$_LogEntry;         -- For WIP and error messages
  js ku$_JobStatus;        -- The job status from get_status
  jd ku$_JobDesc;          -- The job description from get_status
  sts ku$_Status;          -- The status object returned by get_status
BEGIN
-- Create a (user-named) Data Pump job to do a schema export.
  h1 := dbms_datapump.open('EXPORT','SCHEMA',NULL,'EXAMPLE3','LATEST');
-- Specify a single dump file for the job (using the handle just returned)
-- and a directory object, which must already be defined and accessible
-- to the user running this procedure.
  dbms_datapump.add_file(h1,'example3.dmp','DMPDIR');
-- A metadata filter is used to specify the schema that will be exported.
  dbms_datapump.metadata_filter(h1,'SCHEMA_EXPR','IN (''HR'')');
-- Start the job. An exception will be returned if something is not set up
-- properly.One possible exception that will be handled differently is the
-- success_with_info exception. success_with_info means the job started
-- successfully, but more information is available through get_status about
-- conditions around the start_job that the user might want to be aware of.
    begin
    dbms_datapump.start_job(h1);
    dbms_output.put_line('Data Pump job started successfully');
    exception
      when others then
        if sqlcode = dbms_datapump.success_with_info_num
        then
          dbms_output.put_line('Data Pump job started with info available:');
          dbms_datapump.get_status(h1,
                                   dbms_datapump.ku$_status_job_error,0,
                                   job_state,sts);
          if (bitand(sts.mask,dbms_datapump.ku$_status_job_error) != 0)
          then
            le := sts.error;
            if le is not null
            then
              ind := le.FIRST;
              while ind is not null loop
                dbms_output.put_line(le(ind).LogText);
                ind := le.NEXT(ind);
              end loop;
            end if;
          end if;
        else
          raise;
        end if;
  end;
-- The export job should now be running. In the following loop,
-- the job is monitored until it completes. In the meantime, progress information --
is displayed.

 percent_done := 0;
  job_state := 'UNDEFINED';
  while (job_state != 'COMPLETED') and (job_state != 'STOPPED') loop
    dbms_datapump.get_status(h1,
           dbms_datapump.ku$_status_job_error +
           dbms_datapump.ku$_status_job_status +
           dbms_datapump.ku$_status_wip,-1,job_state,sts);
    js := sts.job_status;
-- If the percentage done changed, display the new value.
     if js.percent_done != percent_done
    then
      dbms_output.put_line('*** Job percent done = ' ||
                           to_char(js.percent_done));
```

```
      percent_done := js.percent_done;
    end if;
-- Display any work-in-progress (WIP) or error messages that were received for
-- the job.
      if (bitand(sts.mask,dbms_datapump.ku$_status_wip) != 0)
    then
      le := sts.wip;
    else
      if (bitand(sts.mask,dbms_datapump.ku$_status_job_error) != 0)
      then
        le := sts.error;
      else
        le := null;
      end if;
    end if;
    if le is not null
    then
      ind := le.FIRST;
      while ind is not null loop
        dbms_output.put_line(le(ind).LogText);
        ind := le.NEXT(ind);
      end loop;
    end if;
  end loop;
-- Indicate that the job finished and detach from it.
  dbms_output.put_line('Job has completed');
  dbms_output.put_line('Final job state = ' || job_state);
  dbms_datapump.detach(h1);
-- Any exceptions that propagated to this point will be captured. The
-- details will be retrieved from get_status and displayed.
  exception
    when others then
      dbms_output.put_line('Exception in Data Pump job');
      dbms_datapump.get_status(h1,dbms_datapump.ku$_status_job_error,0,
                               job_state,sts);
      if (bitand(sts.mask,dbms_datapump.ku$_status_job_error) != 0)
      then
        le := sts.error;
        if le is not null
        then
          ind := le.FIRST;
          while ind is not null loop
            spos := 1;
            slen := length(le(ind).LogText);
            if slen > 255
            then
              slen := 255;
            end if;
            while slen > 0 loop
              dbms_output.put_line(substr(le(ind).LogText,spos,slen));
              spos := spos + 255;
              slen := length(le(ind).LogText) + 1 - spos;
            end loop;
            ind := le.NEXT(ind);
          end loop;
        end if;
      end if;
END;
/
```

例6-4 ダンプ・ファイルの情報の表示

この例のPL/SQLスクリプトは、データ・ポンプAPIプロシージャDBMS_DATAPUMP.GET_DUMPFILE_INFOを使用して、データ・ポンプ・ジョブのコンテキストの外部でデータ・ポンプ・ダンプ・ファイルに関する情報を表示する方法を示しています。この例では、例6-1のサンプルPL/SQLスクリプトで作成されたexample1.dmpダンプ・ファイルに含まれる情報を表示しています。

このPL/SQLスクリプトを使用して、オリジナルのエクスポート(expユーティリティ)およびORACLE_DATAPUMP外部表アクセス・ドライバで作成されたダンプ・ファイルの情報を表示することもできます。

このスクリプトを使用するには、ユーザーSYSTEMとして接続します。

```
SET VERIFY OFF
SET FEEDBACK OFF

DECLARE
  ind        NUMBER;
  fileType   NUMBER;
  value      VARCHAR2(2048);
  infoTab    KU$_DUMPFILE_INFO := KU$_DUMPFILE_INFO();

BEGIN
  --
  -- Get the information about the dump file into the infoTab.
  --
  BEGIN
    DBMS_DATAPUMP.GET_DUMPFILE_INFO('example1.dmp','DMPDIR',infoTab,fileType);
    DBMS_OUTPUT.PUT_LINE('----------------------------------------------');
    DBMS_OUTPUT.PUT_LINE('Information for file: example1.dmp');

    --
    -- Determine what type of file is being looked at.
    --
    CASE fileType
      WHEN 1 THEN
        DBMS_OUTPUT.PUT_LINE('example1.dmp is a Data Pump dump file');
      WHEN 2 THEN
        DBMS_OUTPUT.PUT_LINE('example1.dmp is an Original Export dump file');
      WHEN 3 THEN
        DBMS_OUTPUT.PUT_LINE('example1.dmp is an External Table dump file');
      ELSE
        DBMS_OUTPUT.PUT_LINE('example1.dmp is not a dump file');
        DBMS_OUTPUT.PUT_LINE('----------------------------------------------');
    END CASE;

  EXCEPTION
    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('----------------------------------------------');
      DBMS_OUTPUT.PUT_LINE('Error retrieving information for file: ' ||
                           'example1.dmp');
      DBMS_OUTPUT.PUT_LINE(SQLERRM);
      DBMS_OUTPUT.PUT_LINE('----------------------------------------------');
      fileType := 0;
  END;

  --
  -- If a valid file type was returned, then loop through the infoTab and
  -- display each item code and value returned.
  --
  IF fileType > 0
  THEN
    DBMS_OUTPUT.PUT_LINE('The information table has ' ||
                         TO_CHAR(infoTab.COUNT) || ' entries');
    DBMS_OUTPUT.PUT_LINE('----------------------------------------------');

    ind := infoTab.FIRST;
    WHILE ind IS NOT NULL
    LOOP
      --
      -- The following item codes return boolean values in the form
      -- of a '1' or a '0'. Display them as 'Yes' or 'No'.
      --
      value := NVL(infoTab(ind).value, 'NULL');
```

```
      IF infoTab(ind).item_code IN
         (DBMS_DATAPUMP.KU$_DFHDR_MASTER_PRESENT,
          DBMS_DATAPUMP.KU$_DFHDR_DIRPATH,
          DBMS_DATAPUMP.KU$_DFHDR_METADATA_COMPRESSED,
          DBMS_DATAPUMP.KU$_DFHDR_DATA_COMPRESSED,
          DBMS_DATAPUMP.KU$_DFHDR_METADATA_ENCRYPTED,
          DBMS_DATAPUMP.KU$_DFHDR_DATA_ENCRYPTED,
          DBMS_DATAPUMP.KU$_DFHDR_COLUMNS_ENCRYPTED)
      THEN
        CASE value
          WHEN '1' THEN value := 'Yes';
          WHEN '0' THEN value := 'No';
        END CASE;
      END IF;

      --
      -- Display each item code with an appropriate name followed by
      -- its value.
      --
      CASE infoTab(ind).item_code
        --
        -- The following item codes have been available since Oracle
        -- Database 10g, Release 10.2.
        --
        WHEN DBMS_DATAPUMP.KU$_DFHDR_FILE_VERSION   THEN
          DBMS_OUTPUT.PUT_LINE('Dump File Version:         ' || value);
        WHEN DBMS_DATAPUMP.KU$_DFHDR_MASTER_PRESENT THEN
          DBMS_OUTPUT.PUT_LINE('Master Table Present:      ' || value);
        WHEN DBMS_DATAPUMP.KU$_DFHDR_GUID THEN
          DBMS_OUTPUT.PUT_LINE('Job Guid:                  ' || value);
        WHEN DBMS_DATAPUMP.KU$_DFHDR_FILE_NUMBER THEN
          DBMS_OUTPUT.PUT_LINE('Dump File Number:          ' || value);
        WHEN DBMS_DATAPUMP.KU$_DFHDR_CHARSET_ID  THEN
          DBMS_OUTPUT.PUT_LINE('Character Set ID:          ' || value);
        WHEN DBMS_DATAPUMP.KU$_DFHDR_CREATION_DATE THEN
          DBMS_OUTPUT.PUT_LINE('Creation Date:             ' || value);
        WHEN DBMS_DATAPUMP.KU$_DFHDR_FLAGS THEN
          DBMS_OUTPUT.PUT_LINE('Internal Dump Flags:       ' || value);
        WHEN DBMS_DATAPUMP.KU$_DFHDR_JOB_NAME THEN
          DBMS_OUTPUT.PUT_LINE('Job Name:                  ' || value);
        WHEN DBMS_DATAPUMP.KU$_DFHDR_PLATFORM THEN
          DBMS_OUTPUT.PUT_LINE('Platform Name:             ' || value);
        WHEN DBMS_DATAPUMP.KU$_DFHDR_INSTANCE THEN
          DBMS_OUTPUT.PUT_LINE('Instance Name:             ' || value);
        WHEN DBMS_DATAPUMP.KU$_DFHDR_LANGUAGE THEN
          DBMS_OUTPUT.PUT_LINE('Language Name:             ' || value);
        WHEN DBMS_DATAPUMP.KU$_DFHDR_BLOCKSIZE THEN
          DBMS_OUTPUT.PUT_LINE('Dump File Block Size:      ' || value);
        WHEN DBMS_DATAPUMP.KU$_DFHDR_DIRPATH THEN
          DBMS_OUTPUT.PUT_LINE('Direct Path Mode:          ' || value);
        WHEN DBMS_DATAPUMP.KU$_DFHDR_METADATA_COMPRESSED THEN
          DBMS_OUTPUT.PUT_LINE('Metadata Compressed:       ' || value);
        WHEN DBMS_DATAPUMP.KU$_DFHDR_DB_VERSION THEN
          DBMS_OUTPUT.PUT_LINE('Database Version:          ' || value);

        --
        -- The following item codes were introduced in Oracle Database 11g
        -- Release 11.1
        --
        WHEN DBMS_DATAPUMP.KU$_DFHDR_MASTER_PIECE_COUNT THEN
          DBMS_OUTPUT.PUT_LINE('Master Table Piece Count:  ' || value);
        WHEN DBMS_DATAPUMP.KU$_DFHDR_MASTER_PIECE_NUMBER THEN
          DBMS_OUTPUT.PUT_LINE('Master Table Piece Number: ' || value);
        WHEN DBMS_DATAPUMP.KU$_DFHDR_DATA_COMPRESSED THEN
          DBMS_OUTPUT.PUT_LINE('Table Data Compressed:     ' || value);
        WHEN DBMS_DATAPUMP.KU$_DFHDR_METADATA_ENCRYPTED THEN
          DBMS_OUTPUT.PUT_LINE('Metadata Encrypted:        ' || value);
```

```
        WHEN DBMS_DATAPUMP.KU$_DFHDR_DATA_ENCRYPTED THEN
          DBMS_OUTPUT.PUT_LINE('Table Data Encrypted:      ' || value);
        WHEN DBMS_DATAPUMP.KU$_DFHDR_COLUMNS_ENCRYPTED THEN
          DBMS_OUTPUT.PUT_LINE('TDE Columns Encrypted:     ' || value);

        --
        -- For the DBMS_DATAPUMP.KU$_DFHDR_ENCRYPTION_MODE item code a
        -- numeric value is returned. So examine that numeric value
        -- and display an appropriate name value for it.
        --
        WHEN DBMS_DATAPUMP.KU$_DFHDR_ENCRYPTION_MODE THEN
          CASE TO_NUMBER(value)
            WHEN DBMS_DATAPUMP.KU$_DFHDR_ENCMODE_NONE THEN
              DBMS_OUTPUT.PUT_LINE('Encryption Mode:           None');
            WHEN DBMS_DATAPUMP.KU$_DFHDR_ENCMODE_PASSWORD THEN
              DBMS_OUTPUT.PUT_LINE('Encryption Mode:           Password');
            WHEN DBMS_DATAPUMP.KU$_DFHDR_ENCMODE_DUAL THEN
              DBMS_OUTPUT.PUT_LINE('Encryption Mode:           Dual');
            WHEN DBMS_DATAPUMP.KU$_DFHDR_ENCMODE_TRANS THEN
              DBMS_OUTPUT.PUT_LINE('Encryption Mode:           Transparent');
          END CASE;

        --
        -- The following item codes were introduced in Oracle Database 12c
        -- Release 12.1
        --

        --
        -- For the DBMS_DATAPUMP.KU$_DFHDR_COMPRESSION_ALG item code a
        -- numeric value is returned. So examine that numeric value and
        -- display an appropriate name value for it.
        --
        WHEN DBMS_DATAPUMP.KU$_DFHDR_COMPRESSION_ALG THEN
          CASE TO_NUMBER(value)
            WHEN DBMS_DATAPUMP.KU$_DFHDR_CMPALG_NONE THEN
              DBMS_OUTPUT.PUT_LINE('Compression Algorithm:     None');
            WHEN DBMS_DATAPUMP.KU$_DFHDR_CMPALG_BASIC THEN
              DBMS_OUTPUT.PUT_LINE('Compression Algorithm:     Basic');
            WHEN DBMS_DATAPUMP.KU$_DFHDR_CMPALG_LOW THEN
              DBMS_OUTPUT.PUT_LINE('Compression Algorithm:     Low');
            WHEN DBMS_DATAPUMP.KU$_DFHDR_CMPALG_MEDIUM THEN
              DBMS_OUTPUT.PUT_LINE('Compression Algorithm:     Medium');
            WHEN DBMS_DATAPUMP.KU$_DFHDR_CMPALG_HIGH THEN
              DBMS_OUTPUT.PUT_LINE('Compression Algorithm:     High');
          END CASE;
        ELSE NULL;  -- Ignore other, unrecognized dump file attributes.
      END CASE;
      ind := infoTab.NEXT(ind);
    END LOOP;
  END IF;
END;
/
```

親トピック: Oracle Data Pump APIの使用

第II部 SQL*Loader

SQL*Loaderおよびその機能の説明に加え、データ・ロードの概念についても説明します(オブジェクト・サポートも含む)。

- SQL*Loaderの使用方法の理解
  SQL*Loaderを使用してOracle Databaseにデータをロードする前に知っておく必要がある基本概念について説明します。
- SQL*Loaderコマンドライン・リファレンス
  通常のSQL*Loaderを起動するには、コマンドライン・パラメータを使用します。
- SQL*Loader制御ファイル・リファレンス
  SQL*Loader制御ファイルは、SQL*Loaderジョブ用のデータ定義言語(DDL)命令を含むテキスト・ファイルです。
- SQL*Loaderフィールド・リスト・リファレンス
  SQL*Loader制御ファイルのフィールド・リストには、位置、データ型、条件、デリミタなど、ロードするフィールドの情報が提供されます。
- SQL*Loaderを使用したオブジェクト、LOBおよびコレクションのロード
  SQL*Loaderを使用して、様々な形式で列オブジェクトがロードでき、オブジェクト表、REF列、LOBおよびコレクションのロードができます。
- 従来型パス・ロードおよびダイレクト・パス・ロード
  SQL*Loaderには、従来のパス・ロード方法とダイレクト・パス・ロード方法を使用してデータをロードするオプションがあります。
- SQL*Loaderエクスプレス
  SQL*Loaderエクスプレス・モードでは、SQL*Loaderを使用して単純なデータ型を迅速かつ簡単にロードできます。

7 SQL*Loaderの使用方法の理解

SQL*Loaderを使用してOracle Databaseにデータをロードする前に知っておく必要がある基本概念について学習します。

- SQL*Loaderの機能
  SQL*Loaderは、外部ファイルからOracle Databaseの表にデータをロードします。
- SQL*Loaderのパラメータ
  SQL*Loaderは、`sqlldr`コマンドを指定するか、ロード操作の様々な特性を確立するパラメータを指定すると起動します。
- SQL*Loader制御ファイル
  制御ファイルは、SQL*Loaderが解釈できる言語で記述されたテキスト・ファイルです。
- SQL*Loaderによる入力データおよびデータ・ファイルの読取り方法
  制御ファイルに指定された1つ以上のデータ・ファイル(またはファイルのオペレーティング・システムに相当するもの)などから、SQL*Loaderにデータが読み込まれます。
- LOBFILEおよびセカンダリ・データ・ファイル(SDF)
  ラージ・オブジェクト(LOB)データは、非常に長いデータであるため、LOBFILEからロードすると有効です。
- データ変換およびデータ型の指定
  従来型パス・ロード中に、データ・ファイルのデータ・フィールドが、データベースの列に変換されます(概念的にはダイレクト・パス・ロードと同様ですが、実装は異なります)。
- SQL*Loaderの廃棄レコードと拒否レコード
  SQL*Loaderは、入力ファイルから読み取ったレコードの一部を拒否または廃棄することがあります。これは、ファイルに問題がある場合またはレコードのロードからの除外を選択した場合に発生します。
- ログ・ファイルおよびログ情報
  SQL*Loaderで処理が開始されると、**ログ・ファイル**が作成されます。
- 従来型パス・ロード、ダイレクト・パス・ロードおよび外部表ロード
  SQL*Loaderでデータをロードするには、いくつかの方法があります。
- SQL*Loaderによるオブジェクト、コレクションおよびLOBのロード
  実社会のエンティティ(顧客と発注など)のモデル化に必要な列、行、LOBおよびJSONデータベース・オブジェクトを一括ロードできます。
- SQL*Loaderでのパーティション・オブジェクトのサポート
  パーティション・データベース・オブジェクトを使用すると、データのセクションをまとめて管理することも個別に管理することもできます。SQL*Loaderでは、パーティション・オブジェクトのロードをサポートしています。
- アプリケーション開発: ダイレクト・パス・ロードAPI
  ダイレクト・パス・ロードを使用すると、外部表からのデータを表とパーティションにロードできます。Oracleでは、アプリケーション開発者のためにダイレクト・パス・ロードAPIを提供しています。
- SQL*Loaderの事例
  SQL*Loaderの機能の使用方法を学習するために、Oracleが提供する様々な事例を実行できます。

親トピック: SQL*Loader

7.1 SQL*Loaderの機能

SQL*Loaderは、外部ファイルからOracle Databaseの表にデータをロードします。

強力なデータ解析エンジンによって、あらゆるデータ形式のデータファイルに対応できます。SQL*Loaderを使用して、次のこと

が可能です。

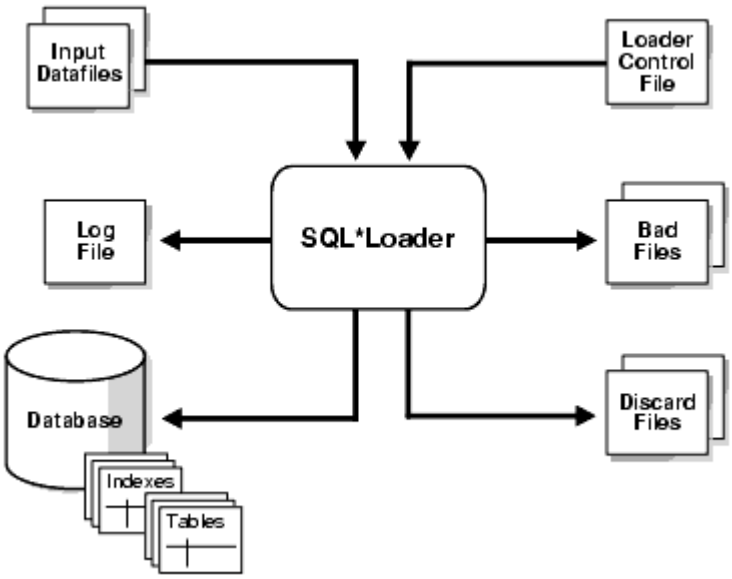
- データ・ファイルがデータベースと異なるシステム上にある場合のネットワークを介したデータのロード。
- 同一のロード・セッションでの複数のデータ・ファイルからのデータのロード。
- 同一のロード・セッションでの複数の表へのデータのロード。
- データの文字セットの指定。
- ロード・データの選択(レコード値に基づいたロード)。
- ロード前の、SQL関数を使用したデータ処理。
- 指定した列に対する、一意の順序キーの生成。
- オペレーティング・システムのファイル・システムを使用したデータ・ファイルへのアクセス。
- ディスク、テープまたはNamed Pipeからのデータのロード。
- 高度なエラー・レポートの生成による、トラブルシューティングの支援。
- 複合オブジェクト・リレーショナル・データの任意のロード。
- セカンダリ・データ・ファイルを使用した、LOBおよびコレクションのロード。
- 従来型ロード、ダイレクト・パス・ロードまたは外部表ロードの使用。「従来型パス・ロード、ダイレクト・パス・ロードおよび外部表ロード」を参照してください。

SQL*Loaderは、制御ファイルを指定する場合と指定しない場合の2つの方法で使用できます。制御ファイルは、SQL*Loaderの動作と、ロードに使用される1つ以上のデータ・ファイルを制御します。制御ファイルを使用すると、ロード操作を詳細に制御できるため、より複雑なロード環境に対応できます。ただし、単純なロードでは、制御ファイルを指定せずにSQL*Loaderを使用できます(これはSQL*Loaderエクスプレス・モードと呼ばれます)。詳細は、「SQL*Loaderエクスプレス」を参照してください。

SQL*Loaderの出力先は、データがロードされるOracle Database、ログ・ファイル、不良ファイル(拒否されたレコードが存在する場合)で、廃棄ファイルに出力される場合もあります。

次の図に、制御ファイルを使用する標準的なSQL*Loaderセッションのフローの例を示します。

図7-1 SQL*Loaderの概要



親トピック: SQL*Loaderの使用方法の理解

7.2 SQL*Loaderのパラメータ

SQL*Loaderは、sqlldrコマンドを指定するか、ロード操作の様々な特性を確立するパラメータを指定すると起動します。

常に、値がほとんど変わらない同じパラメータを使用する場合は、コマンドラインではなく、次の方法を使用してパラメータを指定すると効率的です。

- パラメータをパラメータ・ファイルとしてグループ化できます。その後、PARFILEパラメータを使用することで、そのパラメータ・ファイルの名前をコマンドラインで指定できます。
- OPTIONS句を使用すると、SQL*Loader制御ファイル内のいくつかのパラメータを指定できます。

コマンドラインで指定したパラメータは、パラメータ・ファイルまたはOPTIONS句で指定したパラメータ値を上書きします。

関連トピック

- SQL*Loaderコマンドライン・リファレンス
- PARFILE
- OPTIONS Clause

親トピック: SQL*Loaderの使用方法の理解

7.3 SQL*Loader制御ファイル

制御ファイルは、SQL*Loaderが解釈できる言語で記述されたテキスト・ファイルです。

制御ファイルは、データの場所、データの分析と解釈方法、データの挿入先などをSQL*Loaderに通知します。

通常、制御ファイルには、次の3つの項目が次の順序で含まれます。

- セッション全体の情報
- 表およびフィールド・リストの情報
- 入力データ(オプションの項目)

制御ファイルの構文には、次の注意事項があります。

- 構文は、自由形式で記述できます(文は複数行になってもかまいません)。
- 構文の大文字と小文字は、一重引用符または二重引用符で囲まれた文字列の場合のみ区別され、それ以外では区別されません。
- 制御ファイルの構文では、コメントは、コメントの始まりから行の終わりまでマークする2つのハイフン(- -)から拡張されます。オプションである第3セクションでは、二重ハイフンがコメントとしてではなくデータとして解釈されるため、このセクションでのコメントはサポートされません。
- CONSTANTおよびZONEキーワードは、SQL*Loaderでは特別な意味があり、予約されています。競合を回避するために、表または列の名前にCONSTANTまたはZONEという語を使用しないことをお薦めします。

関連トピック

- SQL*Loader制御ファイル・リファレンス

親トピック: SQL*Loaderの使用方法の理解

7.4 SQL*Loaderによる入力データおよびデータ・ファイルの読取り方法

制御ファイルに指定された1つ以上のデータ・ファイル(またはファイルのオペレーティング・システムに相当するもの)などから、SQL*Loaderにデータが読み込まれます。

SQL*Loaderから見ると、データ・ファイルのデータはレコードとして構成されています。データ・ファイルには、固定レコード形式、可変レコード形式またはストリーム・レコード形式があります。レコード形式は、INFILEパラメータを使用して制御ファイルに指定することができます。レコード形式が指定されない場合、デフォルトはストリーム・レコード形式になります。

ノート:

制御ファイル内部でデータが指定されている場合(INFILE *が制御ファイルに指定されている場合)、そのデータはデフォルトでレコード終了記号を使用したストリーム・レコード形式として解釈されます。

- 固定レコード形式
  固定レコード形式のファイルでは、データ・ファイルにあるすべてのレコードが同じバイト長です。
- 可変レコード形式およびSQL*Loader
  可変レコード形式のファイルでは、文字フィールドの各レコード長がデータ・ファイルの各レコードの開始位置に含まれています。
- ストリーム・レコード形式およびSQL*Loader
  ストリーム・レコード形式では、レコードのサイズ指定ではなく、SQL*Loaderで**レコード終了記号**を読み込むことによって、レコードが確認されます。
- 論理レコードおよびSQL*Loader
  SQL*Loaderでは、入力データは、指定されたレコード形式で物理レコードに編成されます。デフォルトでは、1つの物理レコードが1つの論理レコードになります。
- データ・フィールド設定およびSQL*Loader
  論理レコードを形成した後にSQL*Loaderで論理レコードのフィールド設定がどのように決定されるかについて説明します。

親トピック: SQL*Loaderの使用方法の理解

7.4.1 固定レコード形式

固定レコード形式のファイルでは、データ・ファイルにあるすべてのレコードが同じバイト長です。

固定レコード形式は最も柔軟性の低い形式ですが、これを使用すると、変数形式またはストリーム形式よりもパフォーマンスが向上します。また、固定形式は簡単に指定できます。たとえば:

```
INFILE datafile_name "fix n"
```

ここでは、特殊なデータ・ファイルが、SQL*Loaderによって全レコードがnバイト長の固定レコード形式で解釈されるように指定しています。

次の例は、固定レコード形式で解釈されるデータ・ファイル(example1.dat)を指定する制御ファイルを示します。この例のデータ・ファイルには5つの物理レコードが含まれ、各レコードには従業員の番号と名前を含むフィールドがあります。5つの各フィールドは空白を含めて11バイト長です。この例を説明する目的で、レコードの空白を示すためにピリオドが使用されていますが、実際のレコードにはピリオドはありません。この点に注意すると、1つ目のレコード396,...ty,.は、ちょうど11バイトです(シングルバイト文字セットだとします)。2つ目のレコードは4922,beth,で、改行文字(¥n)が後に続き、11バイトとなります。(改行文

字は固定レコード形式では必要ありません。ここでは単に、使用された場合はレコード長のバイトとしてカウントされることを説明するために使用されています。)

例7-1 固定レコード形式でのデータのロード

データのロード:

```
load data
infile 'example1.dat'  "fix 11"
into table example
fields terminated by ',' optionally enclosed by '"'
(col1, col2)
```

example1.datの内容:

```
396,...ty,.4922,beth,¥n
68773,ben,.
1,.."dave",
5455,mike,.
```

文字長セマンティクスが使用されているファイルでも、長さは常にバイト単位で解析されます。これは、ファイル内にフィールドが混在しているために必要なことです。文字長セマンティクスで処理されるものもあれば、バイト長セマンティクスで処理されるものもあります。

関連トピック

- 文字長セマンティクス

親トピック: SQL*Loaderによる入力データおよびデータ・ファイルの読取り方法

7.4.2 可変レコード形式およびSQL*Loader

可変レコード形式のファイルでは、文字フィールドの各レコード長がデータ・ファイルの各レコードの開始位置に含まれています。

この形式は、固定レコード形式より柔軟性があり、ストリーム・レコード形式よりパフォーマンスに優れています。可変レコード形式の場合、たとえば、次のように指定できます。

```
INFILE "datafile_name" "var n"
```

nには、レコード長フィールドのバイト数を指定します。nを指定しない場合、SQL*Loaderは長さを5バイトとみなします。nに、40より大きい値を指定すると、エラーになります。

次の例に、データ・ファイルexample2.datのデータを検索し、レコードの最初の3バイトがフィールドの長さを示す可変レコード形式を使用するようにSQL*Loaderに指示する制御ファイルの指定を示します。example2.datデータ・ファイルは、3つの物理レコードで構成されています。1つ目のレコードは009 (9)バイト長、2つ目のレコードは010 (10)バイト長(1バイトの改行を含む)、3つ目は012 (12)バイト長(1バイトの改行を含む)で指定されています。改行文字は、可変レコード形式では必要ありません。この例でも、データ・ファイルはシングルバイト文字セットであるとします。この例の説明目的として、example2.datのピリオドは空白を示していますが、フィールドには実際のピリオドは含まれません。

例7-2 可変レコード形式でのデータのロード

データのロード:

```
load data
infile 'example2.dat'  "var 3"
into table example
fields terminated by ',' optionally enclosed by '"'
(col1 char(5),
 col2 char(7))
```

example2.datの内容:

```
009.396,.ty,0104922,beth,01268773,benji,
```

文字長セマンティクスが使用されているファイルでも、長さは常にバイト単位で解析されます。これは、文字長セマンティクスで処理されるフィールドとバイト長セマンティクスで処理されるフィールドがファイル内に混在する可能性があるため必要です。

関連トピック

- 文字長セマンティクス

親トピック: SQL*Loaderによる入力データおよびデータ・ファイルの読取り方法

7.4.3 ストリームレコード形式およびSQL*Loader

ストリーム・レコード形式では、レコードをサイズで指定してではなく、SQL*Loaderで**レコード終了記号**を読み込むことによって、レコードが確認されます。

ストリーム・レコード形式は最も柔軟性のある形式ですが、パフォーマンスに悪影響を与える場合があります。ストリーム・レコード形式として解釈されるようにデータ・ファイルを指定するには、次のように指定します。

```
INFILE datafile_name ["str terminator_string"]
```

前述の例では、strによってファイルがストリーム・レコード形式であることを示しています。terminator_stringには、次の'char_string'またはX'hex_string'を指定します。

- 'char_string'は、一重引用符または二重引用符で囲まれた文字列です。
- X'hex_string'は、16進形式のバイト列です。

terminator_stringに特別な(印字不可能な)文字が含まれる場合は、X'hex_string'バイト文字列として指定する必要があります。ただし、一部の印字不可能な文字はバックスラッシュを使用することで('char_string')として指定できます。たとえば:

- ¥nは、LFを示します。
- ¥tは、水平タブを示します。
- ¥fは、改ページを示します。
- ¥vは、垂直タブを示します。
- ¥rは、改行を示します。

セッションのNLS_LANG初期化パラメータで指定される文字セットがデータ・ファイルの文字セットと異なる場合、文字列はデータ・ファイルの文字セットに変換されます。これはSQL*Loaderがデフォルトのレコード終了記号を確認する前に実行されます。

16進文字列はデータ・ファイルの文字セット内にあるとみなされ、変換は実行されません。

UNIXベースのプラットフォームでは、terminator_stringを指定しない場合、デフォルトで改行文字¥nが使用されます。

Windowsベースのプラットフォームでは、terminator_stringを指定していない場合、SQL*Loaderは、¥nまたは¥r¥nのうちデータ・ファイル内で先に現れる文字をレコード終了記号として使用します。つまり、データ・ファイルの1つ以上のレコードで、フィールドに¥nが埋め込まれていることがわかっている場合に、¥r¥nをレコード終了記号として使用するには、これを指定する必要があります。

次の例は、文字列'|¥n'を使用して終了記号文字列が指定されている場合に、ストリーム・レコード形式でデータをロードする方法を示しています。バックスラッシュを使用すると、文字列に印字不可能な改行文字を指定することができます。

関連項目:

- Language and Character Set File Scanner (LCSSCAN)ユーティリティを使用して不明なファイル・テキストの言語および文字セットを確認する方法の詳細は、『Oracle Databaseグローバリゼーション・サポート・ガイド』を参照してください。

例7-3 ストリーム・レコード形式でのデータのロード

データのロード:

```
load data
infile 'example3.dat'  "str '|¥n'"
into table example
fields terminated by ',' optionally enclosed by '"'
(col1 char(5),
 col2 char(7))
```

`example3.dat`の内容:

```
396,ty,|
4922,beth,|
```

親トピック: SQL*Loaderによる入力データおよびデータ・ファイルの読取り方法

7.4.4 論理レコードおよびSQL*Loader

SQL*Loaderでは、入力データは、指定されたレコード形式で物理レコードに編成されます。デフォルトでは、1つの物理レコードが1つの論理レコードになります。

柔軟性の向上により、SQL*Loaderでは複数の物理レコードを1つの論理レコードに結合することもできます。

論理レコードは、次のいずれかの方針に従って構成できます。

- 固定数の物理レコードを結合してそれぞれの論理レコードを構成する
- 特定の条件が真である場合に物理レコードを結合して論理レコードを構成する

関連トピック

- 物理レコードからの論理レコードの作成
- SQL*Loaderの事例

親トピック: SQL*Loaderによる入力データおよびデータ・ファイルの読取り方法

7.4.5 データ・フィールド設定およびSQL*Loader

論理レコードを形成した後にSQL*Loaderで論理レコードのフィールド設定がどのように決定されるかについて学習します。

フィールド設定では、制御ファイルのフィールド指定を使用して、論理レコードのデータのどの部分が制御ファイルのフィールドに対応しているのかをSQL*Loaderで判断します。2つ以上のフィールド指定に同じデータを使用できます。また、論理レコードに、制御ファイルのフィールド指定で使用されないデータを含めることもできます。

ほとんどの場合、制御ファイルのフィールドに、論理レコードの特定の位置や長さの指定が必要です。この部分は、次のような形式で指定します。

- データ・フィールドの開始バイト位置または終了位置(あるいはその両方)を指定できます。この指定形式に柔軟性はあ

りませんが、フィールド設定によって高パフォーマンスが得られます。

- 特定のデータ・フィールドの区切り(囲みまたは終了、あるいはその両方)文字列を指定できます。デリミタ付きデータ・フィールドは、データ・フィールドの開始バイト位置が指定されている場合を除いて、直前のデータ・フィールドの終了位置から始まるとみなされます。
- バイト・オフセットまたはデータ・フィールドの長さ(あるいはその両方)を指定できます。この方法では、各フィールドは、直前のフィールドが終了した位置から、指定されたバイト数の位置で始まり、指定された長さの位置で終了します。
- 長さと値のペアのデータ型を使用できます。この場合、データ・フィールドの最初のnバイト数に、データ・フィールドの残りの長さについての情報が含まれています。

関連トピック

- デリミタの指定

親トピック: SQL*Loaderによる入力データおよびデータ・ファイルの読取り方法

7.5 LOBFILEおよびセカンダリ・データ・ファイル(SDF)

ラージ・オブジェクト(LOB)データは、非常に長いデータであるため、LOBFILEからロードすると有効です。

LOBFILEでは、LOBデータのインスタンスがフィールド(事前にサイズが決まっているフィールド、デリミタ付きフィールド、LENGTH-VALUEフィールド)内に収まっているとみなされます。ただし、そうしたフィールドはレコードに編成されません(LOBFILEにはレコードの概念が存在しません)。そのため、レコードを扱うことによって発生する処理のオーバーヘッドを回避できます。このようなデータの編成方法は、LOBのロードにとって理想的です。

たとえば、従業員の名前、IDおよび履歴を格納する表があるとします。この表をロードする場合、従業員名および従業員IDをメイン・データ・ファイルから読み込み、非常に長い従業員の履歴をLOBFILEから読み込むことができます。

また、簡単にXMLデータをロードするためにLOBFILEを使用することもできます。XML列を使用して、構造化データおよび半構造化データのモデルを保持できます。そのようなデータは、非常に長いデータです。

セカンダリ・データ・ファイル(SDF)とプライマリ・データ・ファイルの概念は類似しています。プライマリ・データ・ファイルと同様に、SDFは、レコードおよびフィールドで構成されたレコードの集まりです。SDFは必要に応じて制御ファイルに指定します。SDFをデータ・ソースとして命名できるのは、`collection_fld_spec`のみです。

SDFの指定にはSDFパラメータを使用します。SDFパラメータの値は、ファイル指定文字列を使用して入力することも、1つ以上のファイル指定文字列が含まれているデータ・フィールドにマップされたFILLERフィールドを使用して入力することもできます。

関連トピック

- LOBFILEからのLOBデータのロード
- セカンダリ・データ・ファイル(SDF)

親トピック: SQL*Loaderの使用方法の理解

7.6 データ変換およびデータ型の指定

従来型パス・ロード中に、データ・ファイルのデータ・フィールドが、データベースの列に変換されます(概念的にはダイレクト・パス・ロードと同様ですが、実装は異なります)。

変換には、次の2つのステップがあります。

1. SQL*Loaderで、制御ファイルにあるフィールド指定を使用してデータ・ファイルの形式を解釈し、次に、入力データを

解析します。そのデータを使用してSQL INSERT文に対応するバインド配列を移入します。バインド配列は、SQL*Loaderがロード対象のデータを格納するメモリー内の領域です。バインド配列が一杯の場合、データはデータベースに転送されます。バインド配列のサイズは、SQL*LoaderのBINDSIZEおよびREADSIZEパラメータによって制御されます。

2. データベースがデータを受け取り、INSERT文を実行してデータベースにデータを格納します。

Oracle Databaseでは、列のデータ型を使用して最終的な格納形式にデータを変換します。データ・ファイル内のフィールドとデータベース内の列の違いに注意する必要があります。また、SQL*Loaderの制御ファイルで定義されているフィールドのデータ型は、列のデータ型と同じではないことにも注意してください。

関連項目:

- BINDSIZE
- READSIZE

親トピック: SQL*Loaderの使用方法の理解

7.7 SQL*Loaderの廃棄レコードと拒否レコード

SQL*Loaderは入力ファイルから読み込まれ一部のレコードを拒否または破棄できます。これは、ファイルの問題のため、またはロードからレコードをフィルタ処理することを選択したためです。

拒否レコードは不良ファイルに格納され、廃棄レコードは廃棄ファイルに格納されます。

- SQL*Loader不良ファイル
  不良ファイルには、SQL*LoaderまたはOracle Databaseによって拒否レコードが書き込まれます。
- SQL*Loader廃棄ファイル
  SQL*Loaderの実行時に、一部のレコードがロードから除外されるようにして、廃棄ファイルというファイルが作成されるようにすることもできます。

親トピック: SQL*Loaderの使用方法の理解

7.7.1 SQL*Loader不良ファイル

不良ファイルには、SQL*LoaderまたはOracle Databaseによって拒否レコードが書き込まれます。

不良ファイルを指定していない場合に拒否されたレコードがあると、SQL*Loaderによって不良ファイルが自動的に作成されます。拒否レコードはデータ・ファイルと同じ名前になりますが、拡張子.badが付きます。拒否には、いくつかの原因があります。

- SQL*Loaderによって拒否されたレコード
  入力形式が不適切なデータ・ファイル・レコードは、SQL*Loaderによって拒否されます。
- SQL*Loaderの操作中にOracle Databaseによって拒否されたレコード
  データ・ファイル・レコードは、SQL*Loaderによって受け取られた後、データベースに送られ、行として表に挿入されます。

親トピック: SQL*Loaderの廃棄レコードと拒否レコード

7.7.1.1 SQL*Loaderによって拒否されたレコード

入力形式が不適切なデータ・ファイル・レコードは、SQL*Loaderによって拒否されます。

たとえば、2番目の囲みデリミタがない場合や、デリミタ付きフィールドが最大長を超えている場合、レコードは、SQL*Loaderに

よって拒否されます。拒否レコードは、不良ファイルに書き込まれます。

親トピック: SQL*Loader不良ファイル

7.7.1.2 SQL*Loaderの操作中にOracle Databaseによって拒否されたレコード

データ・ファイル・レコードは、SQL*Loaderによって受け取られた後、データベースに送られ、行として表に挿入されます。

データベースによって有効であると判断された行は、表に挿入されます。行が無効であると判断された場合、レコードは拒否され、SQL*Loaderにより不良ファイルに書き込まれます。行が無効であると判断される例としては、キーが重複している場合、必須入力フィールドに対応するデータがNULL値の場合、またはフィールドにOracleデータ型ではないデータ型が指定された場合が考えられます。

親トピック: SQL*Loader不良ファイル

7.7.2 SQL*Loader廃棄ファイル

SQL*Loaderの実行時に、一部のレコードがロードから除外されるようにして、廃棄ファイルというファイルが作成されるようにすることもできます。

廃棄ファイルが作成されるのは、廃棄ファイルが必要な場合と、廃棄ファイルを有効にするように指定した場合のみです。廃棄ファイルには、制御ファイルに指定されているレコード選択基準に一致しなかったため、ロード対象から除外されたレコードが入ります。

廃棄ファイルにはロードから除外されたレコードが格納されているため、廃棄ファイルの内容はデータベースのどの表にも挿入されなかったレコードになります。廃棄ファイルに格納可能なレコードの最大数を指定できます。レコードのデータがいずれかの表に書き込まれる場合、このレコードは廃棄ファイルには書き込まれません。

親トピック: SQL*Loaderの廃棄レコードと拒否レコード

7.8 ログ・ファイルおよびログ情報

SQL*Loaderで処理が開始されると、**ログ・ファイル**が作成されます。

SQL*Loaderでは、ログ・ファイルを作成できないと処理は終了します。このログ・ファイルにはロード中に発生したエラーに関する記述など、ロードに関する詳細情報が記録されます。

親トピック: SQL*Loaderの使用方法の理解

7.9 従来型パス・ロード、ダイレクト・パス・ロードおよび外部表ロード

SQL*Loaderでデータをロードするには、いくつかの方法があります。

- 従来型パス・ロード
  従来型パス・ロードでは、入力レコードがフィールド指定を基に解析され、各データ・フィールドが対応するバインド配列(SQL*Loaderがロードするデータを格納するメモリー内の領域)にコピーされます。
- ダイレクト・パス・ロード
  ダイレクト・パス・ロードでは、入力レコードがフィールド指定を基に解析され、入力フィールド・データが列のデータ型に変換されて列配列が作成されます。
- パラレル・ダイレクト・パス
  パラレル・ダイレクト・パス・ロードでは、複数のダイレクト・パス・ロード・セッションで同じデータ・セグメントを同時にロードできます(セグメント内の並列化が可能です)。

- 外部表ロード
  外部表はデータベース内に存在しない表として定義されている表で、アクセス・ドライバが用意されている任意の形式が可能です。
- 外部表およびSQL*Loaderの選択
  データのロード時にロード・パフォーマンスが最も高い方法を学習します。
- SQL*Loaderと外部表との処理内容の違い
  外部表を使用したデータのロード(ORACLE_LOADERアクセス・ドライバを使用)と、SQL*Loaderの従来型パス・ロードおよびダイレクト・パス・ロードを使用したデータのロードとの違いを確認することをお薦めします。
- オブジェクト・ストレージに格納されているデータを使用した表のロード
  SQL*Loaderを使用して、オブジェクト・ストレージから標準のOracle Database表にデータをロードする方法について学習します。

親トピック: SQL*Loaderの使用方法の理解

7.9.1 従来型パス・ロード

従来型パス・ロードでは、入力レコードがフィールド指定を基に解析され、各データ・フィールドが対応するバインド配列(SQL*Loaderがロードするデータを格納するメモリー内の領域)にコピーされます。

バインド配列が一杯の場合(または読み取るデータが残っていない場合)、配列挿入操作が実行されます。

SQL*Loaderでは、バインドの配列に挿入後、LOBフィールドが格納されます。そのため、LOBフィールドの処理にエラーが発生した場合(たとえば、LOBFILEがないなど)、LOBフィールドは空のままになります。配列の挿入の実行後にLOBデータがロードされるため、BEFOREおよびAFTER行トリガーはLOB列に対して機能しない場合もあります。これは、SQL*Loaderで列にLOBの内容をロードする機会を持つ前にトリガーが起動されるためです。たとえば、LOB列C1にデータをロードして、BEFORE行トリガーを使用してそのLOB列の内容を調べ、その結果を基に、他の列C2にロードされる値を導出するとします。これは、トリガーの起動時にLOBの内容がロードされていないため不可能です。

関連項目:

- データのロード方法
- バインド配列および従来型パス・ロード

親トピック: 従来型パス・ロード、ダイレクト・パス・ロードおよび外部表ロード

7.9.2 ダイレクト・パス・ロード

ダイレクト・パス・ロードでは、入力レコードがフィールド指定を基に解析され、入力フィールド・データが列のデータ型に変換されて列配列が作成されます。

この列配列は、Oracle Databaseブロック形式でデータ・ブロックを作成するブロック・フォーマッタに渡されます。新しくフォーマットされたデータベース・ブロックはデータベースに直接書き込まれるため、通常行われるデータ処理の大部分が省略されます。ダイレクト・パス・ロードによる処理は、従来型パス・ロードと比較すると非常に高速ですが、制限事項がいくつかあります。

親トピック: 従来型パス・ロード、ダイレクト・パス・ロードおよび外部表ロード

7.9.3 パラレル・ダイレクト・パス

パラレル・ダイレクト・パス・ロードでは、複数のダイレクト・パス・ロード・セッションで 同じデータ・セグメントを同時にロードできます

(セグメント内の並列化が可能です)。

パラレル・ダイレクト・パスには、ダイレクト・パスより多くの制約事項があります。

関連項目:

パラレル・データ・ロード・モデル

ダイレクト・パス・ロード

親トピック: 従来型パス・ロード、ダイレクト・パス・ロードおよび外部表ロード

7.9.4 外部表ロード

外部表はデータベース内に存在しない表として定義されている表で、アクセス・ドライバが用意されている任意の形式が可能です。

Oracle Databaseでは、ORACLE_LOADER,およびORACLE_DATAPUMPの2つのアクセス・ドライバが提供されています。外部表を記述するメタデータを提供することで、外部表内のデータをあたかも標準的なデータベース表内に存在しているデータのように公開できます。

外部表ロードでは、外部データ・ファイルに含まれているデータの外部表が作成されます。ロード処理では、INSERT文が実行され、データ・ファイルからデータがターゲット表に挿入されます。

従来型パス・ロードおよびダイレクト・パス・ロードではなく、外部表ロードを使用した場合のメリットは、次のとおりです。

- データ・ファイルが大きい場合、外部表のロードではパラレルでファイルのロードを試みます。
- 外部表ロードでは、外部表の作成に使用されるINSERT文の一部としてSQL関数およびPL/SQLファンクションを使用することによってロードされるデータを変更できます。

ノート:

Windows オペレーティング・システムでの名前付きパイプを使用した外部表ロードは、サポートされません。

関連トピック

- ORACLE_LOADERアクセス・ドライバ
- ORACLE_DATAPUMPアクセス・ドライバ
- 『Oracle Database管理者ガイド』の「外部表の管理」

親トピック: 従来型パス・ロード、ダイレクト・パス・ロードおよび外部表ロード

7.9.5 外部表およびSQL*Loaderの選択

データのロード時にロード・パフォーマンスが最も高い方法を学習します。

外部表とSQL*Loaderのレコード解析は類似しているため、通常、同じレコード形式での大幅なパフォーマンスの違いはありません。ただし、外部表とSQL*Loaderのアーキテクチャは異なるため、一方の方法が他方より適している場合があります。

次の状況では、最適なロード・パフォーマンスを得るために外部表を使用します。

- データベースへのロード時にデータを変換する場合

- 透過的にパラレル処理を行う前に、外部データを分割する必要がない場合

次の状況では、最適なロード・パフォーマンスを得るためにSQL*Loaderを使用します。

- リモートでデータをロードする場合
- データに対して変換を行う必要がなく、そのデータをパラレルでロードする必要がない場合
- データをロードする必要があり、ステージング表に索引を追加する必要がある場合

親トピック: 従来型パス・ロード、ダイレクト・パス・ロードおよび外部表ロード

7.9.6 SQL*Loaderと外部表との処理内容の違い

外部表を使用したデータのロード(ORACLE_LOADERアクセス・ドライバを使用)と、SQL*Loaderの従来型パス・ロードおよびダイレクト・パス・ロードを使用したデータのロードとの違いを確認することをお薦めします。

この項で示す情報は、ORACLE_DATAPUMPアクセス・ドライバには当てはまりません。

- 複数のプライマリ入力データ・ファイル
  SQL*Loaderのロードを使用したプライマリ入力データ・ファイルが複数存在する場合は、入力データ・ファイルごとに不良ファイルおよび廃棄ファイルが作成されます。
- 構文およびデータ型
  この項では、外部表ロードでサポートされていない構文およびデータ型について説明します。
- バイト順序マーク
  SQL*Loaderでは、バイト順序マークが書き込まれるかどうかは、文字セットまたは表ロードによって決まります。
- デフォルトの文字セット、日付マスク、小数点区切り
  NLS文字セットの表示は、SQL*Loaderと外部表の各種設定で制御します。
- バックスラッシュ・エスケープ文字の使用
  SQL*Loaderと外部表では、一重引用符を囲み文字として識別するために様々な表記規則が使用されます。

親トピック: 従来型パス・ロード、ダイレクト・パス・ロードおよび外部表ロード

7.9.6.1 複数のプライマリ入力データ・ファイル

SQL*Loaderのロードを使用したプライマリ入力データ・ファイルが複数存在する場合は、入力データ・ファイルごとに不良ファイルおよび廃棄ファイルが作成されます。

外部表ロードでは、すべての入力データ・ファイルに対する不良ファイルおよび廃棄ファイルは、1つずつのみです。外部表ロードでパラレル・アクセス・ドライバが使用される場合は、各アクセス・ドライバに不良ファイルおよび廃棄ファイルが含まれます。

親トピック: SQL*Loaderと外部表との処理内容の違い

7.9.6.2 構文およびデータ型

この項では、外部表ロードでサポートされていない構文およびデータ型について説明します。

- CONTINUEIFまたはCONCATENATEを使用した、1つの論理レコードへの複数の物理レコードの結合
- SQL*Loaderデータ型(GRAPHIC、GRAPHIC EXTERNALおよびVARGRAPHIC)のロード
- データベースの列型(LONG、ネストした表、VARRAY、REF、主キーREFおよびSID)の使用

親トピック: SQL*Loaderと外部表との処理内容の違い

7.9.6.3 バイト順序マーク

SQL*Loaderでは、バイト順序マークが書き込まれるかどうかは、文字セットまたは表ロードによって決まります。

プライマリ・データ・ファイルにUnicode文字セット(UTF8またはUTF16)が使用され、バイト順序マーク(BOM)が含まれている場合、バイト順序マークは対応する不良ファイルおよび廃棄ファイルの先頭に書き込まれます。

外部表ロードでは、バイト順序マークは不良ファイルおよび廃棄ファイルの先頭に書き込まれません。

親トピック: SQL*Loaderと外部表との処理内容の違い

7.9.6.4 デフォルトの文字セット、日付マスク、小数点区切り

NLS文字セットの表示は、SQL*Loaderと外部表の各種設定で制御します。

SQL*Loaderでは、デフォルトの文字セット、日付マスクおよび小数点区切りが、クライアント側でのNLS環境変数の設定によって決まります。

外部表のフィールドでは、NLSパラメータのデータベース設定によって、デフォルトの文字セット、日付マスクおよび小数点区切りが決定されます。

親トピック: SQL*Loaderと外部表との処理内容の違い

7.9.6.5 バックスラッシュ・エスケープ文字の使用

SQL*Loaderと外部表では、一重引用符を囲み文字として識別するために様々な表記規則が使用されます。

SQL*Loaderでは、一重引用符が囲み文字として識別されるようにするために、バックスラッシュ(¥)エスケープ文字を使用できます。たとえば

```
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '¥''
```

外部表では、文字列内でバックスラッシュ・エスケープ文字を使用すると、エラーが発生します。解決策として、二重引用符を使用することで、一重引用符を囲み文字として使用できます。たとえば:

```
TERMINATED BY ',' ENCLOSED BY "'"
```

親トピック: SQL*Loaderと外部表との処理内容の違い

7.9.7 オブジェクト・ストレージに格納されたデータを使用した表のロード

SQL*Loaderを使用して、オブジェクト・ストレージから標準のOracle Database表にデータをロードする方法について学習します。

次の例では、データをロードする表(T)があります。

```
SQL> create table t (x int, y int);
```

この表にロードするfile1.txtという名前のデータ・ファイルがあります。内容は次のとおりです。

```
X,Y
1,2
4,5
```

この表をオブジェクト・ストアにロードするには、次の手順を実行します。

1. オブジェクト・ストアの入出力(I/O)を有効にするために必要なライブラリをインストールします。

   ```
   % cd $ORACLE_HOME/rdbms/lib
   ```

```
% make -f ins_rdbms.mk opc_on
```

2. ファイルfile1.txtをオブジェクト・ストレージのバケットにアップロードします。

   オブジェクト・ストレージにファイルをアップロードする最も簡単な方法は、Oracle Cloudコンソールからファイルをアップロードすることです。

   a. Oracle Cloudコンソールを開きます。
   b. オブジェクト・ストレージのタイルを選択します。
   c. まだ作成していない場合は、バケットを作成します。
   d. アップロードをクリックし、ファイルfile1.txtを選択してバケットにアップロードします。

3. Oracle Databaseで、ウォレットおよび資格証明を作成します。

   たとえば:

```
$ orapki wallet create -wallet /home/oracle/wallets -pwd mypassword-auto_login
$ mkstore -wrl /home/oracle/wallets -createEntry
oracle.sqlldr.credential.myfedcredential.username
oracleidentitycloudservice/myuseracct@example.com
$ mkstore -wrl /home/oracle/wallets -createEntry
oracle.sqlldr.credential.myfedcredential.password "MhAVCDfW+-ReskK4:Ho-zH"
```

   この例では、フェデレーテッド・ユーザー・アカウント(myfedcredential)の使用方法を示します。Oracle Cloud Infrastructureドキュメントで説明されているように、パスワードは自動的に生成されます。(「資格証明の管理」の認証トークンの作成に関する項を参照)。

4. ウォレットを作成した後、$ORACLE_HOME/network/adminディレクトリのsqlnet.oraファイルに場所を追加します。

   たとえば:

```
vi test.ctl
LOAD DATA
INFILE 'https://objectstorage.eu-frankfurt-
1.oraclecloud.com/n/dbcloudoci/b/myobjectstore/o/file1.txt'
truncate
INTO TABLE T
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
(X,Y)
```

5. SQL*Loaderを実行して、データをオブジェクト・ストアにロードします。

   たとえば:

```
sqlldr test/mypassword@pdb1 /home/oracle/test.ctl
credential=myfedcredentiallog=test.log external_table=not_used
```

関連トピック

- Oracle Cloud Infrastructureドキュメント、「資格証明の管理」の認証トークンの作成に関する項
- Oracle Cloud Infrastructureドキュメント、コンソールの使用に関する項

親トピック: 従来型パス・ロード、ダイレクト・パス・ロードおよび外部表ロード

7.10 SQL*Loaderによるオブジェクト、コレクションおよびLOBのロード

実社会のエンティティ(顧客と発注など)のモデル化に必要な列、行、LOBおよびJSONデータベース・オブジェクトを一括ロードできます。

- サポートされるオブジェクト型
  SQL*Loaderでは、列オブジェクト型および行オブジェクト型のロードがサポートされています。
- サポートされるコレクション型
  SQL*Loaderでは、ネストした表およびVARRAYコレクション型のロードがサポートされています。
- サポートされるLOBデータ型
  LOBは、ラージ・オブジェクト型です。

親トピック: SQL*Loaderの使用方法の理解

7.10.1 サポートされるオブジェクト型

SQL*Loaderでは、列オブジェクト型および行オブジェクト型のロードがサポートされています。

- 列オブジェクト
  表の列が、なんらかのオブジェクト型である場合、その列のオブジェクトは列オブジェクトと呼ばれます。
- 行オブジェクト
  これらのオブジェクトはオブジェクト表と呼ばれる表に格納され、オブジェクト表にはオブジェクトの属性に対応する列があります。

親トピック: SQL*Loaderによるオブジェクト、コレクションおよびLOBのロード

7.10.1.1 列オブジェクト

表の列が、なんらかのオブジェクト型である場合、その列のオブジェクトは列オブジェクトと呼ばれます。

概念的には、そのようなオブジェクトは、行の単一の列位置に全体が格納されます。これらのオブジェクトにはオブジェクト識別子がなく、参照することはできません。

列オブジェクトのオブジェクト型がNOT FINALであると宣言されると、SQL*Loaderで導出された型(またはサブタイプ)を列オブジェクトにロードできます。

親トピック: サポートされるオブジェクト型

7.10.1.2 行オブジェクト

これらのオブジェクトはオブジェクト表と呼ばれる表に格納され、オブジェクト表にはオブジェクトの属性に対応する列があります。

さらに、そのオブジェクト表にはシステムが生成するSYS_NC_OID$という列があり、その列に、表の各オブジェクトに対してシステムが生成する一意の識別子(OID)が格納されます。他の表の列は、これらのオブジェクトをOIDを使用して参照できます。

オブジェクト表のオブジェクト型がNOT FINALであると宣言されると、SQL*Loaderで導出された型(またはサブタイプ)を行オブジェクトにロードできます。

関連項目:

- 列オブジェクトのロード
- オブジェクト表のロード

親トピック: サポートされるオブジェクト型

7.10.2 サポートされるコレクション型

SQL*Loaderでは、ネストした表およびVARRAYコレクション型のロードがサポートされています。

- ネストした表
  ネストした表は、別の表に列があるように見える表です。
- VARRAY
  VARRAYは、可変サイズの配列です。

親トピック: SQL*Loaderによるオブジェクト、コレクションおよびLOBのロード

7.10.2.1 ネストした表

ネストした表は、別の表に列があるように見える表です。

別の表に対して実行できるすべての操作は、ネストした表に対しても実行できます。

親トピック: サポートされるコレクション型

7.10.2.2 VARRAY

VARRAYは、可変サイズの配列です。

配列は、要素と呼ばれる、一連の組込み型またはオブジェクトの順序付けられた集合です。各配列の要素は同一の型であり、VARRAY内の要素の位置に対応する一意の番号(index)を持ちます。

VARRAY型の作成時に、最大サイズを指定する必要があります。VARRAY型を宣言すると、リレーショナル表の列のデータ型、オブジェクト型属性、またはPL/SQL変数として使用することができます。

関連項目:

SQL*Loader制御ファイルのデータ定義言語を使用してこれらのコレクション型をロードする方法の詳細は、「コレクション(ネストした表およびVARRAY)のロード」を参照してください。

親トピック: サポートされるコレクション型

7.10.3 サポートされるLOBデータ型

LOBは、ラージ・オブジェクト型です。

今回のリリースのSQL*Loaderでは、4つのLOBデータ型のロードをサポートしています。

- BLOB: 構造化されていないバイナリ・データを含むLOB。
- CLOB: 文字データを含むLOB。
- NCLOB: データベースの各国語文字セットの文字を含むLOB。
- BFILE: サーバー側のオペレーティング・システム・ファイルのデータベース表領域外に格納されるBLOB。

LOBは、列のデータ型にすることができ、NCLOB以外はオブジェクトの属性のデータ型にすることもできます。LOBには、実際の値、NULLまたは「値なし(空)」を指定できます。

関連項目:

SQL*Loader制御ファイルのデータ定義言語を使用してこれらのLOB型をロードする方法の詳細は、「LOBのロード」を参照してください。

親トピック: SQL*LoaderによるオブジェクトーコレクションおよびLOBのロード

7.11 SQL*Loaderでのパーティション・オブジェクトのサポート

パーティション・データベース・オブジェクトを使用すると、データのセクションをまとめて管理することも個別に管理することもできます。SQL*Loaderでは、パーティション・オブジェクトのロードをサポートしています。

Oracle Databaseインスタンスでは、グループ化されたパーティション(部分)で構成される表または索引が**パーティション・オブジェクト**に相当します。一般に、パーティションは共通の論理属性によってグループ化されます。たとえば、特定の年度の売上データを、月別にパーティション化するとします。この場合、各月のデータは、売上表の中のそれぞれ別のパーティションに保存されます。このパーティションはそれぞれ、データベース内の異なるセグメントに保存されます。また、パーティションごとに異なる物理属性を指定できます。

次に、パーティション・オブジェクトがサポートされたことによって、SQL*Loaderでロードが可能になったものを示します。

- パーティション表中の個別パーティション
- パーティション表中の全パーティション
- 非パーティション表

親トピック: SQL*Loaderの使用方法の理解

7.12 アプリケーション開発: ダイレクト・パス・ロードAPI

ダイレクト・パス・ロードを使用すると、外部表からのデータを表とパーティションにロードできます。Oracleでは、アプリケーション開発者のためにダイレクト・パス・ロードAPIを提供しています。

関連トピック

- Oracle Call Interfaceプログラマーズ・ガイド

親トピック: SQL*Loaderの使用方法の理解

7.13 SQL*Loaderの事例

SQL*Loaderの機能の使用方法を学習できるように、Oracleでは実行可能な様々な事例を提供しています。

- 事例用ファイル
  SQL*Loaderの各事例ファイルには、その事例の使用に必要なファイルのセットが含まれています。
- 事例の実行
  SQL*Loaderの事例を実行するための一般的なステップは、その他の事例を実行する場合とほとんど同じです。
- 事例用ログ・ファイル
  事例用のログ・ファイルは、`$ORACLE_HOME/rdbms/demo`ディレクトリにあらかじめ用意されているわけではありません。
- 事例の結果確認
  事例の実行結果を確認するには、SQL*Plusを起動して、事例でロードされた表から選択操作を実行します。

親トピック: SQL*Loaderの使用方法の理解

7.13.1 事例用ファイル

SQL*Loaderの各事例用ファイルには、その事例の使用に必要なファイルのセットが含まれています。

使用上のノート

通常、各事例は次の種類のファイルで構成されています。

- 制御ファイル(`ulcase5.ctl`など)
- データ・ファイル(`ulcase5.dat`など)
- セットアップ・ファイル(`ulcase5.sql`など)

これらのファイルは、Oracle Database Examples (以前のCompanion)メディアをインストールするとインストールされます。ファイルは`$ORACLE_HOME/rdbms/demo`ディレクトリにインストールされます。

事例用の例データが制御ファイルに含まれている場合、その事例の`.dat`ファイルはありません。

事例2では特別な設定が不要なため、事例2の`.sql`スクリプトはありません。事例7では、開始(セットアップ)スクリプトと終了(クリーンアップ)スクリプトの両方を実行する必要があります。

次の表に、各事例に関連のあるファイルを示します。

表7-1 事例用ファイルおよび関連ファイル

事例	.ctl	.dat	.sql
1	ulcase1.ctl	適用外	ulcase1.sql
2	ulcase2.ctl	ulcase2.dat	適用外
3	ulcase3.ctl	適用外	ulcase3.sql
4	ulcase4.ctl	ulcase4.dat	ulcase4.sql
5	ulcase5.ctl	ulcase5.dat	ulcase5.sql
6	ulcase6.ctl	ulcase6.dat	ulcase6.sql
7	ulcase7.ctl	ulcase7.dat	ulcase7s.sql ulcase7e.sql
8	ulcase8.ctl	ulcase8.dat	ulcase8.sql
9	ulcase9.ctl	ulcase9.dat	ulcase9.sql
10	ulcase10.ctl	適用外	ulcase10.sql

事例	.ctl	.dat	.sql
11	ulcase11.ctl	ulcase11.dat	ulcase11.sql

親トピック: SQL*Loaderの事例

7.13.2 事例の実行

SQL*Loaderの事例を実行するための一般的なステップは、その他の事例を実行する場合とほとんど同じです。

`$ORACLE_HOME/rdbms/demo`ディレクトリに移動していることを確認してください。事例用ファイルは、このディレクトリに格納されています。

また、各事例を実行する前に、制御ファイルを確認する必要もあります。制御ファイルの先頭には、事例で実施される内容に関する情報と把握する必要があるその他の特別な情報が含まれています。たとえば、事例6では、SQL*LoaderのコマンドラインにDIRECT=TRUEを追加する必要があります。

1. システム・プロンプトで`sqlplus`と入力し、[Enter]を押してSQL*Plusを起動します。ユーザー名のプロンプトで、`scott`と入力します。パスワードのプロンプトで、`tiger`と入力します。

   SQLプロンプトが表示されます。

2. SQLプロンプトで、事例用のSQLスクリプトを実行します。 :

   たとえば、事例1のSQLスクリプトを実行するには、次のコマンドを入力します。

   ```
   SQL> @ulcase1
   ```

   このコマンドは、事例用の表を準備して移入すると、システム・プロンプトに戻ります。

3. システム・プロンプトでSQL*Loaderを起動し、事例を実行します。

   たとえば、事例1を実行する場合は次のコマンドを入力します。

   ```
   sqlldr USERID=scott CONTROL=ulcase1.ctl LOG=ulcase1.log
   ```

   `CONTROL`パラメータと`LOG`パラメータを適切な制御ファイル名とログ・ファイル名に置き換え、[Enter]を押します。パスワードを入力するように要求されたら、`tiger`と入力して[Enter]を押します。

親トピック: SQL*Loaderの事例

7.13.3 事例用ログ・ファイル

事例用のログ・ファイルは、`$ORACLE_HOME/rdbms/demo`ディレクトリにあらかじめ用意されているわけではありません。

これは、各事例のログ・ファイルは、事例を実行したときに生成されるためです(ただし、`LOG`パラメータを使用していることが条件です)。ログ・ファイルを生成しない場合は、コマンドラインで`LOG`パラメータを指定しないようにします。

親トピック: SQL*Loaderの事例

7.13.4 事例の結果確認

事例の実行結果を確認するには、SQL*Plusを起動して、事例でロードされた表から選択操作を実行します。

1. システム・プロンプトで`sqlplus`と入力し、[Enter]を押してSQL*Plusを起動します。ユーザー名のプロンプトで、`scott`と入力します。パスワードのプロンプトで、`tiger`と入力します。

SQLプロンプトが表示されます。

2. SQLプロンプトでSELECT文を使用して、事例がロードされた表からすべての行を選択します。

   たとえば、表empをロードする場合は、次の文を入力します。

```
SQL> SELECT * FROM emp;
```

   emp表の各行の内容が表示されます。

親トピック: SQL*Loaderの事例

8 SQL*Loaderコマンドライン・リファレンス

通常のSQL*Loaderを起動するには、コマンドライン・パラメータを使用します。

> ノート:
>
> 通常の SQL*Loader と SQL*Loader エクスプレス・モードでは、同じパラメータをいくつか共有していますが、これらのパラメータの動作はユーティリティごとに異なる場合があります。ここで説明するパラメータの説明は、通常の SQL*Loader を対象としています。SQL*Loader Express のオプションについては、SQL*Loader Express のパラメータを参照してください。

- SQL*Loaderの起動
  SQL*Loaderの起動方法と、ロードの実行方法を管理するパラメータの指定方法について説明します。
- SQL*Loaderのコマンドライン・パラメータ
  コマンドライン・パラメータを使用して、SQL*Loaderを管理します。
- 終了コードによる結果の検査と表示
  Oracle SQL*Loaderでは、SQL*Loaderの完了後、すぐに実行結果を確認できます。

親トピック: SQL*Loader

8.1 SQL*Loaderの起動

SQL*Loaderの起動方法と、ロードの実行方法を管理するパラメータの指定方法を学習します。

すべてのSQL*Loaderパラメータをリストするヘルプ画面を表示するには、プロンプトで`sqlldr`と入力し、[Enter]を押します。出力には、パラメータのデフォルト値を含む各パラメータと、各パラメータの簡単な説明が表示されます。

- コマンドラインでのパラメータの指定
  SQL*Loaderを起動する際に、ロード操作の様々な特性を確立するためにパラメータを指定します。
- SQL*Loaderのパラメータ指定の代替方法
  コマンドライン・パラメータを制御ファイルに移動する方法、またはよく使用されるパラメータをパラメータ・ファイルに配置する方法について学習します。
- ネットワークを介してデータをロードするためのSQL*Loaderの使用
  ネットワーク接続を介してデータをロードするためにSQL*Loaderを使用するには、SQL*Loaderユーティリティの起動時に、接続文字列に接続識別子を指定します。

親トピック: SQL*Loaderコマンドライン・リファレンス

8.1.1 コマンドラインでのパラメータの指定

SQL*Loaderを起動する際に、ロード操作の様々な特性を確立するためにパラメータを指定します。

SQL*Loaderパラメータの指定方法については、次の例を参照してください。
パラメータはカンマで区切ることができます。ただし、パラメータをカンマで区切る必要はありません。

```
> sqlldr CONTROL=ulcase1.ctl LOG=ulcase1.log
Username: scott
Password: password
```

位置で指定するとは、パラメータ名は入力せずに、値を入力するということです。次の例では、ユーザー名scottが指定され、その後にulcase1.ctlという制御ファイルの名前が指定されています。パスワードを入力するように要求されます。

```
> sqlldr scott ulcase1.ctl
Password: password
```

パラメータ名を使用した後は、すべての後続の指定でパラメータ名を指定する必要があります。位置で指定することはできなくなります。たとえば、次のコマンドではCONTROLパラメータを使用して制御ファイル名を指定していますが、ログ・ファイル名はLOGパラメータなしで指定しています(前の例ではLOGパラメータが指定されてました)。ulcase1.logの位置が正しい場合でも、このコマンドを送信するとエラーになります。

```
> sqlldr scott CONTROL=ulcase1.ctl ulcase1.log
```

コマンドを実行するには、LOGパラメータを具体的に指定してコマンドを入力する必要があります。

```
> sqlldr scott CONTROL=ulcase1.ctl LOG=ulcase1.log
```

親トピック: SQL*Loaderの起動

8.1.2 SQL*Loaderのパラメータ指定の代替方法

コマンドライン・パラメータを制御ファイルに移動する方法、またはよく使用されるパラメータをパラメータ・ファイルに配置する方法について学習します。

コマンドラインの長さが、システムの行の最大サイズを超える場合は、OPTIONS句を使用して、一部のコマンドライン・パラメータを制御ファイルに格納できます。

また、パラメータをパラメータ・ファイルとしてグループ化することもできます。このファイルの名前は、SQL*Loaderの起動時に、PARFILEパラメータを使用してコマンドラインで指定できます。

パラメータ指定のこれらの代替方法は、同じパラメータを同じ値で繰り返し使用する場合に有効です。

コマンドラインで指定したパラメータ値は、パラメータ・ファイルまたはOPTIONS句で指定したパラメータ値を上書きします。

関連トピック

- OPTIONS句
- PARFILE

親トピック: SQL*Loaderの起動

8.1.3 ネットワークを介してデータをロードするためのSQL*Loaderの使用

ネットワーク接続を介してデータをロードするためにSQL*Loaderを使用するには、SQL*Loaderユーティリティの起動時に、接続文字列に接続識別子を指定します。

この識別子では、現在のユーザーのORACLE_SID環境変数の設定によって指定した現行のインスタンスとは別のデータベース・インスタンスを指定できます。接続識別子には、Oracle Net接続記述子または接続記述子にマップするネット・サービス名(通常、tnsnames.oraファイルで定義される)を指定できます。接続識別子を使用するには、Oracle Net Listenerが実行中であることが必要です(デフォルト・リスナーを起動するには、lsnrctl startと入力します)。次に、ユーザーscottがinst1という接続識別子を使用してSQL*Loaderを起動する例を示します。

```
> sqlldr CONTROL=ulcase1.ctl
Username: scott@inst1
Password: password
```

ローカルSQL*Loaderクライアントは接続識別子`inst1`(ネット・サービス名)によって定義されるデータベース・インスタンスに接続し、`ulcase1.ctl`制御ファイルの指定に従ってデータをロードします。

> ノート:
>
> プラガブル・データベース(PDB)にデータをロードするには、SQL*Loader の起動時に、単純に接続文字列に接続識別子を指定します。

関連項目:

- 接続識別子およびOracle Net Listenerの詳細は、『Oracle Database Net Services管理者ガイド』を参照してください。
- PDBの詳細は、『Oracle Database概要』を参照してください。

親トピック: SQL*Loaderの起動

8.2 SQL*Loaderのコマンドライン・パラメータ

コマンドライン・パラメータを使用してSQL*Loaderを管理します。

ここで使用されているコマンドライン・パラメータのデフォルト値および最大値は、LinuxベースおよびUnixベースのシステムでの値です。これらの値はオペレーティング・システムによって異なります。詳細は、オペレーティング・システムのドキュメントを参照してください。

- BAD
  SQL*Loaderの`BAD`コマンドライン・パラメータでは、最初のデータ・ファイル指定に関連付けられた不良ファイルの名前と場所(またはそのどちらか)を指定します。
- BINDSIZE
  SQL*Loaderの`BINDSIZE`コマンドライン・パラメータでは、バインド配列の最大サイズ(バイト単位)を指定します。
- COLUMNARRAYROWS
  SQL*Loaderの`COLUMNARRAYROWS`コマンドライン・パラメータでは、ダイレクト・パス列配列に割り当てる行数を指定します。
- CONTROL
  SQL*Loaderの`CONTROL`コマンドライン・パラメータでは、データのロード方法が記述されているSQL*Loader制御ファイルの名前を指定します。
- DATA
  SQL*Loaderの`DATA`コマンドライン・パラメータでは、ロードするデータを含むデータ・ファイルの名前を指定します。
- DATE_CACHE
  SQL*Loaderの`DATE_CACHE`コマンドライン・パラメータでは、日付キャッシュのサイズ(エントリ数)を指定します。
- DEFAULTS
  SQL*Loaderの`DEFAULTS`コマンドライン・パラメータでは、デフォルト式の評価とロードを制御します。
- DEGREE_OF_PARALLELISM
  SQL*Loaderの`DEGREE_OF_PARALLELISM`コマンドライン・パラメータでは、ロード操作中に使用する並列度を指定します。
- DIRECT

SQL*LoaderのDIRECTコマンドライン・パラメータでは、従来型パスまたはダイレクト・パスのどちらの方法でデータをロードするかを指定します。

- DIRECT_PATH_LOCK_WAIT
  SQL*LoaderのDIRECT_PATH_LOCK_WAITコマンドライン・パラメータでは、ダイレクト・パス・ロードの表ロック待機中の動作を制御します。
- DISCARD
  SQL*LoaderのDISCARDコマンドライン・パラメータでは、表に挿入されず、拒否もされないレコードを格納するための廃棄ファイルをオプションで指定できます。
- DISCARDMAX
  SQL*LoaderのDISCARDMAXコマンドライン・パラメータでは、データのロードが終了するまでに許可される廃棄レコードの数を指定します。
- DNFS_ENABLE
  DNFS_ENABLE SQL*Loaderコマンドライン・パラメータを使用すると、SQL*Loader操作中の入力データ・ファイルに対するDirect NFSクライアントの使用を有効化および無効化できます。
- DNFS_READBUFFERS
  DNFS_READBUFFERS SQL*Loaderコマンドライン・パラメータを使用すると、Direct NFSクライアントで使用される読取りバッファの数を制御できます。
- EMPTY_LOBS_ARE_NULL
  SQL*LoaderのコマンドラインEMPTY_LOBS_ARE_NULLでは、使用可能なデータが存在しないLOB列を空のLOBではなくNULLに設定するように指定します。
- ERRORS
  ERRORS SQL*Loaderコマンドライン・パラメータでは、許可される挿入エラーの最大数を指定します。
- EXTERNAL_TABLE
  EXTERNAL_TABLEパラメータでは、データのロードに外部表オプションを使用するかどうかをSQL*Loaderに指示します。
- FILE
  FILE SQL*Loaderコマンドライン・パラメータでは、エクステントの割当て元となるデータベース・ファイルを指定します。
- HELP
  SQL*LoaderのコマンドラインHELPでは、SQL*Loaderユーティリティのオンライン・ヘルプを表示します。
- LOAD
  SQL*LoaderのコマンドラインLOADでは、ロードするレコードの最大数を指定します。
- LOG
  SQL*LoaderのコマンドラインLOGでは、SQL*Loaderがロード処理に関するログ情報を保存するログ・ファイルのディレクトリ・パスとファイル名(またはそのどちらか)を指定します。
- MULTITHREADING
  SQL*LoaderのコマンドラインMULTITHREADINGでは、サーバー・システムでのストリームのロードと同時にクライアント・システムでのストリームの構築を実行できるようにします。
- NO_INDEX_ERRORS
  SQL*LoaderのコマンドラインNO_INDEX_ERRORSでは、ダイレクト・パス・ロード中に索引付けエラーを許容するかどうかを指定します。
- PARALLEL
  SQL*LoaderのPARALLELパラメータでは、ダイレクト・パスまたは外部表を使用するロードで、同じ表に複数の同時セッションでデータをロードできるかどうかを指定します。

- PARFILE
  SQL*Loaderのコマンドライン・パラメータPARFILEでは、頻繁に使用するコマンドライン・パラメータを格納したファイルの名前を指定します。
- PARTITION_MEMORY
  SQL*Loaderのコマンドライン・パラメータPARFILEでは、多数のパーティションのロード時に使用するメモリーの量を指定します。
- READSIZE
  デフォルト値を使用しない場合は、SQL*Loaderのコマンドライン・パラメータREADSIZEで読取りバッファのサイズ(バイト単位)を指定します。
- RESUMABLE
  SQL*Loaderのコマンドライン・パラメータRESUMABLEでは、再開可能な領域割当てを有効および無効にします。
- RESUMABLE_NAME
  SQL*Loaderのコマンドライン・パラメータRESUMABLE_NAMEでは、再開可能な文を特定します。
- RESUMABLE_TIMEOUT
  SQL*Loaderのコマンドライン・パラメータRESUMABLE_TIMEOUTでは、エラー修正までに必要な時間(秒単位)を指定します。
- ROWS
  従来型パス・ロードの場合は、SQL*Loaderのコマンドライン・パラメータROWSでバインド配列の行数を指定し、ダイレクト・パス・ロードの場合は、保存前にデータ・ファイルから読み取る行数を指定します。
- SDF_PREFIX
  SQL*Loaderのコマンドライン・パラメータSDF_PREFIXでは、ロード操作の一環として開かれるLOBFILEおよびセカンダリ・データ・ファイル(SDF)のファイル名に追加するディレクトリ接頭辞を指定します。
- SILENT
  SQL*Loaderのコマンドライン・パラメータSILENTでは、SQL*Loaderの操作中に画面に出力される内容の一部を抑止します。
- SKIP
  SQL*Loaderのコマンドライン・パラメータSKIPでは、ファイルの先頭からのロードが不要な論理レコード数を指定します。
- SKIP_INDEX_MAINTENANCE
  SQL*Loaderのコマンドライン・パラメータSKIP_INDEX_MAINTENANCEでは、ダイレクト・パス・ロードの索引メンテナンスを停止するかどうかを指定します。
- SKIP_UNUSABLE_INDEXES
  SQL*Loaderのコマンドライン・パラメータSKIP_UNUSABLE_INDEXESでは、索引使用禁止状態で出現した索引をスキップし、ロード操作を続行するかどうかを指定します。
- STREAMSIZE
  SQL*Loaderのコマンドライン・パラメータSTREAMSIZEでは、クライアントからサーバーに送信するデータ・ストリームのサイズ(バイト単位)を指定します。
- TRIM
  SQL*Loaderのコマンドライン・パラメータTRIMでは、テキスト・フィールドの先頭とテキスト・フィールドの末尾の両方(またはどちらか)から空白を切り捨てるかどうかを指定します。
- USERID
  SQL*Loaderのコマンドライン・パラメータUSERIDでは、SQL*LoaderのOracleユーザー名とパスワードを指定します。

親トピック: SQL*Loaderコマンドライン・リファレンス

8.2.1 BAD

SQL*LoaderのBADコマンドライン・パラメータでは、最初のデータ・ファイル指定に関連付けられた不良ファイルの名前と場所(またはそのどちらか)を指定します。

デフォルト

.bad拡張子が付いたデータ・ファイルの名前。

用途

最初のデータ・ファイルの指定に関連する不良ファイルの名前または場所(あるいはその両方)を指定します。

構文および説明

```
BAD=[directory/][filename]
```

不良ファイルには、挿入時にエラーの原因になったレコードや、不適切に書式化されたレコードが格納されます。BADパラメータを指定する場合は、ディレクトリとファイル名(またはそのどちらか)を指定する必要があります。拒否されたレコードが存在するときに、不良ファイルの名前を指定していない場合は、データ・ファイル名に拡張子またはファイル・タイプの.badが付いたものがデフォルトのファイル名になります。

directoryに指定する値によって、書込み先にする不良ファイルのディレクトリを指定します。指定には、デバイスまたはネットワーク・ノードの名前を含めることができます。directoryの値は、次のように決定されます。

- BADパラメータが指定されていないときに、不良ファイルが必要になった場合は、SQL*Loaderの制御ファイルが存在するディレクトリがデフォルト・ディレクトリになります。
- BADパラメータでファイル名を指定して、ディレクトリを指定していない場合は、カレント・ディレクトリがデフォルト・ディレクトリになります。
- BADパラメータでディレクトリを指定して、ファイル名を指定していない場合は、指定したディレクトリが使用され、そのファイルのデフォルト名は、データ・ファイルの名前に拡張子またはファイル・タイプの.badが付いたものになります。

filenameに指定する値には、対象プラットフォームで有効と認識されるファイル名を指定します。名前(および.bad以外のものを使用する場合は拡張子)のみを指定する必要があります。ファイル名に空白や句読点文字が含まれている場合は、一重引用符で囲む必要があります。

コマンドラインで指定された不良ファイルは、制御ファイルの最初のINFILE文(存在する場合)に関連する不良ファイルとなります。不良ファイルは、BADFILE句を使用することで、SQL*Loader制御ファイルで指定することもできます。不良ファイルが制御ファイルとコマンドラインの両方で指定された場合は、コマンドラインの値が使用されます。同じ名前の不良ファイルが存在している場合は、既存のファイルが上書きされることも、新しいバージョンのファイルが作成されることもあります(オペレーティング・システムに応じて異なります)。

例

次の指定では、カレント・ディレクトリにemp1.badという不良ファイルを作成します。

```
BAD=emp1
```

関連トピック

- 不良ファイルの理解と指定

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.2 BINDSIZE

SQL*LoaderのBINDSIZEコマンドライン・パラメータでは、バインド配列の最大サイズ(バイト単位)を指定します。

デフォルト

256000

用途

バインド配列の最大サイズ(バイト単位)を指定します。

構文および説明

```
BINDSIZE=n
```

バインド配列は、SQL*Loaderがロード対象のデータを格納するメモリー内の領域です。バインド配列が一杯の場合、データはデータベースに転送されます。バインド配列のサイズは、パラメータのBINDSIZEとREADSIZEで制御します。

BINDSIZEで指定されたバインド配列サイズは、デフォルト・サイズ(システムによって異なる)およびROWSに基づいて計算されたサイズよりも優先されます。

制限事項

- BINDSIZEパラメータは、従来型パス・ロードでのみ使用します。

例

次のBINDSIZEの指定では、バインド配列の最大サイズを356,000バイトに制限します。

```
BINDSIZE=356000
```

関連トピック

- バインド配列および従来型パス・ロード
- READSIZE
- ROWS

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.3 COLUMNARRAYROWS

SQL*LoaderのCOLUMNARRAYROWSコマンドライン・パラメータでは、ダイレクト・パス列配列に割り当てる行数を指定します。

デフォルト

5000

用途

ダイレクト・パス列配列に割り当てる行数を指定します。

構文および説明

```
COLUMNARRARYROWS=n
```

このパラメータの値は、SQL*Loaderでは計算されません。値を指定するか、またはデフォルトを使用する必要があります。

例

次の例では、ダイレクト・パス列配列に1000行を割り当てるように指定しています。

```
COLUMNARRAYROWS=1000
```

関連トピック

- CONCATENATEを使用した論理レコードの作成
- 列配列の行数およびストリーム・バッファ・サイズの指定

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.4 CONTROL

SQL*LoaderのCONTROLコマンドライン・パラメータでは、データのロード方法が記述されているSQL*Loader制御ファイルの名前を指定します。

デフォルト

デフォルトはありません。

用途

データのロード方法が記述されているSQL*Loader制御ファイルの名前を指定します。

構文および説明

```
CONTROL=control_file_name
```

ファイル拡張子またはファイル・タイプを指定しないと、デフォルト設定の.ctlになります。CONTROLパラメータを指定しない場合、SQL*Loaderによってその入力を要求されます。

SQL*Loader制御ファイルの名前に特殊文字が含まれている場合、オペレーティング・システムによっては、その文字をエスケープする必要があります。また、ファイル・システムのパスにバックスラッシュが使用されているオペレーティング・システムの場合は、複数のエスケープ文字を使用するか、そのパスを引用符で囲む必要があります。特殊文字の使用方法の詳細は、対象オペレーティング・システムのマニュアルを参照してください。

例

次の例では、emp1という制御ファイルを指定します。デフォルトの拡張子.ctlが自動的に付けられます。

```
CONTROL=emp1
```

関連トピック

- SQL*Loader制御ファイル・リファレンス

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.5 DATA

SQL*LoaderのDATAコマンドライン・パラメータでは、ロードするデータを含むデータ・ファイルの名前を指定します。

デフォルト

制御ファイルと同じ名前ですが、拡張子は.datになります。

用途

DATAパラメータでは、ロードするデータを含むデータ・ファイルの名前を指定します。

構文および説明

```
DATA=data_file_name
```

ファイル拡張子を指定しない場合、デフォルトで.datになります。

ファイルの指定にはワイルド・カード(ファイル名およびファイル拡張子にのみ可能で、デバイスまたはディレクトリ名には不可)を含めることができます。アスタリスク(*)は複数文字を表し、疑問符(?)は1文字を表します。たとえば:

```
DATA='emp*.dat'
DATA='m?emp.dat'
```

複数のデータ・ファイルの指定(それぞれにワイルド・カードを含めることが可能)をリストするには、ファイル名をカンマで区切る必要があります。

ファイル名に任意の特殊文字(空白、*、?など)が含まれる場合、名前全体を一重引用符で囲む必要があります。

次に、DATAパラメータの有効な3つの使用例を示します(ファイル名に特殊文字が含まれる場合のみ、一重引用符が必要です)。

```
DATA='file1','file2','file3','file4','file5','file6'
DATA='file1','file2'
DATA='file3,'file4','file5'
DATA='file6'
```

注意:

複数のデータ・ファイルがロードされる状況で BAD パラメータも指定する場合、ファイル名ではなく、不良ファイルのディレクトリのみを指定することをお薦めします。指定した名前を持つファイルがすでに存在する場合、使用しているオペレーティング・システムに応じて、既存ファイルが上書きされるか、新しいバージョンのファイルが生成されます。

コマンドラインでDATAパラメータを使用してデータ・ファイルを指定し、同時に制御ファイルでINFILE句を使用してデータ・ファイルを指定すると、制御ファイルの最初のINFILEの指定は無視されます。コマンドラインと制御ファイル内で指定された他のすべてのデータ・ファイルは処理されます。

DATAパラメータと一緒にファイル処理オプションを指定すると、制御ファイルからのデータのロード時に、警告メッセージが発行されます。

例

次の例では、employees.datというデータ・ファイルをロードするように指定します。拡張子が指定されていないため、.dat拡張子がデフォルトとみなされます。

```
DATA=employees
```

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.6 DATE_CACHE

SQL*LoaderのDATE_CACHEコマンドライン・パラメータでは、日付キャッシュのサイズ(エントリ数)を指定します。

デフォルト

使用可能(1000要素)。日付キャッシュ機能を完全に使用禁止にするには、0 (ゼロ)に設定します。

用途

日付キャッシュのサイズ(エントリ数)を指定します。

日付キャッシュは、テキスト文字列から内部の日付書式への変換結果を格納するために使用されます。このキャッシュが役立つ理由は、テキスト形式から日付形式に変換するよりも、日付を検索するコストのほうがはるかに低いためです。日付ファイルに同じ日付が繰り返し出現する場合は、日付キャッシュを使用することで、ダイレクト・パス・ロードの速度を向上できます。

構文および説明

```
DATE_CACHE=n
```

必要に応じて、すべての表に固有の日付キャッシュが作成されます。日付キャッシュは、表に格納する前にデータ型の変換が必要になる日付値またはタイムスタンプ値が1つ以上ロードされた場合にのみ作成されます。

日付キャッシュ機能は、デフォルトで使用可能です。デフォルトの日付キャッシュ・サイズは1000要素です。デフォルトのサイズを使用しているときに、ロードされた一意の入力値の数が1000を超えると、その表に対する日付キャッシュ機能は自動的に使用禁止になります。ただし、デフォルトを上書きして0以外の日付キャッシュ・サイズを指定したときには、そのサイズを超過してもキャッシュは使用禁止になりません。

将来の同様のロードに向けてキャッシュ・サイズを調整する場合は、ログ・ファイルに含まれている日付キャッシュ統計(エントリ数、ヒット数およびミス数)を使用します。

制限事項

- 日付キャッシュ機能は、ダイレクト・パス・ロードと外部表ロードでのみ使用できます。

例

次の指定では、日付キャッシュ機能を完全に使用禁止にします。

```
DATE_CACHE=0
```

関連トピック

- DATE_CACHEの値の指定

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.7 DEFAULTS

SQL*LoaderのDEFAULTSコマンドライン・パラメータでは、デフォルト式の評価とロードを制御します。

デフォルト

EVALUATE_ONCE (順序が関係しない場合)。順序が関係する場合のデフォルトは、EVALUATE_EVERY_ROWです。

用途

デフォルト式の評価とロードを制御します。

DEFAULTSパラメータは、ダイレクト・パス・ロードにのみ適用できます。

構文および説明

```
DEFAULTS={IGNORE | IGNORE_UNSUPPORTED_EVALUATE_ONCE |
IGNORE_UNSUPPORTED_EVALUATE_EVERY_ROW |
EVALUATE_ONCE | EVALUATE_EVERY_ROW}
```

各オプションの動作は次のとおりです。

- IGNORE: 列のデフォルト句は無視されます。
- IGNORE_UNSUPPORTED_EVALUATE_ONCE: ロードの開始時にデフォルト式を1回評価します。サポートされていないデフォルト式は無視されます。DEFAULTSパラメータを使用しないと、デフォルト式が順序を参照しないかぎり、デフォルト式は1回評価されます。デフォルト式が順序を参照する場合は、すべての行が評価されます。
- IGNORE_UNSUPPORTED_EVALUATE_EVERY_ROW: サポートされていないデフォルト句を無視して、すべての行のデフォルト式を評価します。
- EVALUATE_ONCE: ロード開始時にデフォルト式を1回評価します。DEFAULTSパラメータを使用しないと、デフォルトが順序を参照しないかぎり、デフォルト式は1回評価されます。デフォルトが順序を参照する場合は、すべての行が評価されます。サポートされていないデフォルト式の句に対してはエラーが発行されます。(これがこのパラメータのデフォルト・オプションです。)
- EVALUATE_EVERY_ROW: すべての行のデフォルト式を評価し、サポートされていないデフォルトにはエラーを発行します。

例

この例では、testという名前で表を作成します(SQL*Loader制御ファイルの名前はtest.ctlです)。

```
create table test
(
   c0 varchar2(10),
   c1 number default '100'
)
;
test.ctl:
load data
infile *
truncate
into table test
fields terminated by ','
trailing nullcols
(
  c0 char
)
begindata
1,
```

その後、c1にNULLをロードするには、次の文を発行します。

```
sqlldr scott/password t.ctl direct=true defaults=ignore
```

c1にデフォルト値の100をロードするには、次の文を発行します。

```
sqlldr scott/password t.ctl direct=true
```

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.8 DEGREE_OF_PARALLELISM

SQL*LoaderのDEGREE_OF_PARALLELISMコマンドライン・パラメータでは、ロード操作中に使用する並列度を指定します。

デフォルト

```
NONE
```

用途

DEGREE_OF_PARALLELISMパラメータでは、ロード操作中に使用する並列度を指定します。

構文および説明

```
DEGREE_OF_PARALLELISM=[degree-num|DEFAULT|AUTO|NONE]
```

degree-numを指定する場合、1からnの整数値にする必要があります。

DEFAULTを指定すると、(AUTOのデフォルト・パラメータ値ではなく)データベースのデフォルトの並列度が使用されます。

AUTOを使用すると、Oracle Databaseによってロードの並列度が自動的に設定されます。

NONEを指定すると、ロードはパラレルに実行されません。

関連項目:

- パラレル実行の詳細は、『Oracle Database VLDBおよびパーティショニング・ガイド』を参照してください。

制限事項

- DEGREE_OF_PARALLELISMパラメータは、外部表ロード方法が使用される場合にのみ有効です。

例

次の例では、ロードの並列度を3に設定します。

```
DEGREE_OF_PARALLELISM=3
```

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.9 DIRECT

SQL*LoaderのDIRECTコマンドライン・パラメータでは、従来型パスまたはダイレクト・パスのどちらの方法でデータをロードするかを指定します。

デフォルト

```
FALSE
```

用途

DIRECTパラメータでは、従来型パスまたはダイレクト・パスのどちらのロード方法を使用するかを指定します。

構文および説明

```
DIRECT=[TRUE | FALSE]
```

TRUE値はダイレクト・パス・ロードを指定します。FALSE値は従来型パス・ロードを指定します。

関連項目:

従来型パス・ロードおよびダイレクト・パス・ロード

例

次の例では、従来型パス・モードを使用してロードを実行するように指定します。

```
DIRECT=FALSE
```

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.10 DIRECT_PATH_LOCK_WAIT

SQL*LoaderのDIRECT_PATH_LOCK_WAITコマンドライン・パラメータでは、ダイレクト・パス・ロードの表ロック待機中の動作を制御します。

デフォルト

用途

FALSE

表ロック待機中のダイレクト・パス・ロードの動作を制御します。ダイレクト・パス・ロードでは、ロードを続行する前に、表をロックする必要があります。DIRECT_PATH_LOCK_WAITコマンドでは、ロック待機中のダイレクト・パスAPIの動作を制御します。

構文および説明

```
DIRECT_PATH_LOCK_WAIT = {TRUE | FALSE}
```

- TRUE: ダイレクト・パスは、表のロックを取得できるまで待機してからロードを続行します。
- FALSE: (デフォルト)。FALSEに設定すると、ダイレクト・パスAPIは表のロックを複数回試行し、各試行間に1秒待機します。実行される試行の最大回数は30回です。30回試行した後、表をロックできない場合、ダイレクト・パスAPIは表のロックの試行時に生成されたエラーを返します。

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.11 DISCARD

SQL*LoaderのDISCARDコマンドライン・パラメータでは、表に挿入されず、拒否もされないレコードを格納するための廃棄ファイルをオプションで指定できます。

デフォルト

データ・ファイルと同じファイル名ですが、拡張子は.dscになります。

用途

DISCARDパラメータでは、表に挿入されず、拒否もされないレコードを格納するための廃棄ファイルをオプションで指定できます。これらは不良レコードではなく、制御ファイル内で指定されているWHEN句などのレコード選択基準に単純に一致しなかったレコードです。

構文および説明

```
DISCARD=[directory/][filename]
```

DISCARDパラメータを指定する場合、ディレクトリまたはファイル名(あるいはその両方)を指定する必要があります。

directoryパラメータには、廃棄ファイルの書込み先となるディレクトリを指定します。指定には、デバイスまたはネットワーク・ノードの名前を含めることができます。directoryの値は、次のように決定されます。

- DISCARDパラメータが指定されていないが、DISCARDMAXパラメータは指定されている場合、デフォルト・ディレクトリは、SQL*Loaderの制御ファイルが存在するディレクトリになります。
- DISCARDパラメータでファイル名が指定されているが、ディレクトリがない場合、ディレクトリはデフォルトでカレント・ディレ

クトリになります。

- DISCARDパラメータでディレクトリが指定されているが、ファイル名がない場合、指定されたディレクトリが使用され、名前と拡張子にはデフォルトが使用されます。

filenameパラメータには、プラットフォームで有効とみなされるファイル名を指定します。名前(および.dsc以外のものを使用する場合は拡張子)のみを指定する必要があります。ファイル名に空白やその他の句読点文字が含まれている場合は、一重引用符で囲む必要があります。

DISCARDパラメータもDISCARDMAXパラメータも指定しない場合、廃棄レコードがあっても廃棄ファイルは作成されません。

DISCARDパラメータが指定されていないが、DISCARDMAXパラメータは指定されている場合に廃棄レコードが存在すると、デフォルト名を使用して廃棄ファイルが作成され、ファイルはSQL*Loaderの制御ファイルが存在する場所と同じディレクトリに書き込まれます。

注意:

複数のデータ・ファイルがロードされる状況で DISCARD パラメータも指定する場合、ファイル名ではなく、廃棄ファイルのディレクトリのみを指定することをお薦めします。指定した名前を持つファイルがすでに存在する場合、使用しているオペレーティング・システムに応じて、既存ファイルが上書きされるか、新しいバージョンのファイルが生成されます。

コマンドラインで指定された廃棄ファイルは、制御ファイルの最初のINFILE文(存在する場合)に関連する廃棄ファイルとなります。つまり、制御ファイル中で指定する廃棄ファイル名よりも、コマンドラインで指定した廃棄ファイル名の方が優先されます。生成される廃棄ファイルと同じ名前のファイルがすでに存在する場合は、既存ファイルが上書きされるか、新しいバージョンのファイルが生成されます。どちらの処理が行われるかは、使用するオペレーティング・システムによって異なります。

関連項目:

廃棄ファイルの形式の詳細は、「廃棄レコードおよび拒否レコード」を参照してください。

例

employees.datというデータ・ファイルをロードすることを前提としています。次の例では、ディレクトリ名のみを指定しているため、廃棄ファイルの名前はemployees.dscになり、mydirディレクトリに作成されます。

```
DISCARD=mydir/
```

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.12 DISCARDMAX

SQL*LoaderのDISCARDMAXコマンドライン・パラメータでは、データのロードが終了するまでに許可される廃棄レコードの数を指定します。

デフォルト

ALL

用途

DISCARDMAXパラメータは、データ・ロードが終了する前に、許可する廃棄レコードの数を指定します。

構文および説明

```
DISCARDMAX=n
```

最初の廃棄レコードで停止するには、値0を指定します。

DISCARDMAXが指定されているが、DISCARDパラメータは指定されていない場合、廃棄ファイルの名前は、拡張子.dscが付けられたデータ・ファイルの名前になります。

例

次の例では、ロードが終了するまでに25個の破棄レコードが許可されます。

```
DISCARDMAX=25
```

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.13 DNFS_ENABLE

DNFS_ENABLE SQL*Loaderコマンドライン・パラメータを使用すると、SQL*Loader操作中の入力データ・ファイルに対するDirect NFSクライアントの使用を有効化および無効化できます。

デフォルト

TRUE

用途

DNFS_ENABLEパラメータでは、SQL*Loader操作中の入力データ・ファイルに対するDirect NFSクライアントの使用を有効化および無効化できます。

構文および説明

```
DNFS_ENABLE=[TRUE|FALSE]
```

Direct NFSクライアントは、Oracle Databaseがファイル・サーバー上のファイルにアクセスする際のパフォーマンスを向上するために、ファイル・サーバーで実装できるAPIです。

SQL*Loaderでは、1GBを超えるデータ・ファイルを読み取る場合にデフォルトでDirect NFSクライアント・インタフェースが使用されます。よりサイズの小さいファイルでは、オペレーティング・システムのI/Oインタフェースが使用されます。すべての入力データ・ファイルでDirect NFSクライアントを使用するには、DNFS_ENABLE=TRUEを使用します。

すべてのデータ・ファイルでDirect NFSクライアントの使用を無効にするには、DNFS_ENABLE=FALSEを指定します。

DNFS_READBUFFERSパラメータは、Direct NFSクライアントによって使用される読取りバッファの数(デフォルトは4)を指定するために使用できます。

関連項目:

- Direct NFSクライアントの有効化の詳細は、使用しているプラットフォームの『Oracle Grid Infrastructureインストレーション・ガイド』を参照してください。

例

次の例では、ロード中の入力データ・ファイルでのDirect NFSクライアントの使用を無効にします。

```
DNFS_ENABLE=FALSE
```

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.14 DNFS_READBUFFERS

DNFS_READBUFFERS SQL*Loaderコマンドライン・パラメータを使用すると、Direct NFSクライアントで使用される読取りバッファの数を制御できます。

デフォルト

4

用途

DNFS_READBUFFERSパラメータでは、Direct NFSクライアントによって使用される読取りバッファの数を制御できます。Direct NFSクライアントは、Oracle Databaseがファイル・サーバー上のファイルにアクセスする際のパフォーマンスを向上するために、ファイル・サーバーで実装できるAPIです。

構文および説明

```
DNFS_READBUFFERS=n
```

nの値は、指定する読取りバッファの数です。読取りバッファ数を増やすことで、Direct NFSクライアントのファイル・サーバーからの一貫性のない入力/出力(I/O)を補正できます。ただし、大きい値を使用すると、メモリー使用量が増加することがあります。

制限事項

- DNFS_ENABLEパラメータを同時に指定せずにこのパラメータを使用するには、入力ファイルが1GBを超えている必要があります。

例

次の例では、Direct NFSクライアントによって使用される読取りバッファの数を10に指定します。

```
DNFS_READBUFFERS=10
```

関連トピック

- ご使用のプラットフォーム用の『Oracle Grid Infrastructureインストレーション・ガイド』

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.15 EMPTY_LOBS_ARE_NULL

SQL*LoaderのコマンドラインEMPTY_LOBS_ARE_NULLでは、使用可能なデータが存在しないLOB列を空のLOBではなくNULLに設定するように指定します。

デフォルト

FALSE

用途

SQL*LoaderのEMPTY_LOBS_ARE_NULLパラメータを指定すると、使用可能なデータがないラージ・オブジェクト(LOB)列は、空のLOBではなくNULLに設定されます。使用可能なデータがないLOB列をNULLに設定することで、その変更をデータ・ロード後の後処理で実行する必要がなくなります。

構文および説明

```
EMPTY_LOBS_ARE_NULL = {TRUE | FALSE}
```

EMPTY_LOBS_ARE_NULLパラメータはSQL*Loaderのコマンドラインで指定することも、SQL*Loader制御ファイルの

OPTIONS句で指定することもできます。

制限事項

ありません。

例

次の例では、設定empty_lobs_are_null=trueの結果として、c1のLOB列は空のLOBではなくNULLに設定されます。

```
create table t
(
   c0 varchar2(10),
   c1 clob
)
;
sqlldr control file:
options (empty_lobs_are_null=true)
load data
infile *
truncate
into table t
fields terminated by ','
trailing nullcols
(
  c0 char,
  c1 char
)
begindata
1,,
```

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.16 ERRORS

ERRORS SQL*Loaderコマンドライン・パラメータでは、許可される挿入エラーの最大数を指定します。

デフォルト

50

用途

ERRORSパラメータでは、許可する挿入エラーの最大数を指定します。

構文および説明

```
ERRORS=n
```

発生したエラーの数がERRORSに指定された値を超えると、SQL*Loaderによりロードが中止されます。その前までに挿入されたデータはコミットされます。

エラーを許容しない場合は、ERRORS=0を設定します。エラーを無制限に許容する場合は、非常に大きい値を指定します。

SQL*Loaderでは、すべての表の間でレコードの整合性を保つように処理されます。つまり、複数の表をロードする場合は、エラーの数がこの上限を超えても、すぐにはロード処理が中止されません。SQL*Loaderでは、複数の表のロードに対して許容最大数のエラーが検出されても、行のロードは続行され、すでに表にロードされた有効な行は確実にすべての表にロードされます。また、拒否された行はすべての表から削除されます。

どのような場合でも、SQL*Loaderでは、エラーになったレコードが不良ファイルに書き込まれます。

例

次の例では、ロードの挿入エラーの最大数を25に指定します。それを超えると、ロードは終了します。

```
ERRORS=25
```

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.17 EXTERNAL_TABLE

EXTERNAL_TABLEパラメータは、外部表オプションを使用してデータをロードするかどうかをSQL*Loaderに指定します。

デフォルト

NOT_USED

構文および説明

```
EXTERNAL_TABLE=[NOT_USED | GENERATE_ONLY | EXECUTE]
```

指定可能な値は次のとおりです。

- NOT_USED - デフォルト値。従来型パス・モードまたはダイレクト・パス・モードのいずれかを使用して、ロードを行います。
- GENERATE_ONLY - 制御ファイルに記述されているとおり、SQL*Loaderログ・ファイル内の外部表を使用してロードを行うために必要なすべてのSQL文を書き込みます。これらのSQL文は、編集およびカスタマイズできます。実際のロードは、SQL*Loaderを使用せずに、SQL*Plusでこれらの文を実行して、後で行うことができます。
- EXECUTE - 外部表を使用してロードを行うために必要なSQL文を実行します。ただし、SQL文からエラーが返されると、ロードは停止します。SQL文は、実行されたとおりログ・ファイルに書き込まれます。つまり、SQL文からエラーが返されると、ロードに必要な残りのSQL文はログ・ファイルには書き込まれません。

  SQL*Loader制御ファイルでEXTERNAL_TABLE=EXECUTEパラメータとSEQUENCEパラメータの両方を使用すると、SQL*Loaderは、データベースの順序を作成し、その順序に従って表をロードした後、その順序を削除します。この方法でロードを実行すると、従来型またはダイレクト・パスによる方法でロードした場合とは異なる結果になります。(順序の作成方法の詳細は、『Oracle Database SQL言語リファレンス』のCREATE SEQUENCEに関する項を参照してください。)

> ノート:
>
> EXTERNAL_TABLE パラメータを指定すると、SQL*Loader 制御ファイル内の日時データ型(TIMESTAMP など)は自動的に CHAR データ型に変換されて、外部表の date_format_spec 句が使用されます。詳細は、「date_format_spec」を参照してください。

外部表オプションで、データベース内のディレクトリ・オブジェクトを使用して、すべての入力データ・ファイルがどこに格納されるか、および出力ファイル(不良ファイルや廃棄ファイルなど)がどこで作成されるかを指定します。データ・ファイルを含むディレクトリ・オブジェクトには、READアクセスが、出力ファイルが作成されるディレクトリ・オブジェクトには、WRITEアクセスが必要です。データ・ファイルまたは出力ファイル格納用の既存のディレクトリ・オブジェクトがない場合は、SQL*LoaderでSQL文を生成して作成します。したがって、EXECUTEオプションを指定する場合は、CREATE ANY DIRECTORY権限が必要です。ロード処理の終了時にディレクトリ・オブジェクトを削除する場合は、DROP ANY DIRECCTORY権限も必要です。

ノート:

SQL*Loader で EXTERNAL_TABLE=EXECUTE 修飾子を指定すると、データのロードに使用可能な外部表が作成された後、INSERT 文が実行され、データがロードされます。外部表のすべてのファイルがディレクトリ・オブジェクト内に存在すると識別される必要があります。SQL*Loader は、ユーザーがアクセス権限を所有している既存のディレクトリ・オブジェクトを使用します。ただし、SQL*Loader により一致するディレクトリ・オブジェクトが検出されない場合は、一時ディレクトリ・オブジェクトの作成が試行されます。新しいディレクトリ・オブジェクトを作成する権限を所有していない場合、この操作は正常に実行されません。

この問題を解決するには、EXTERNAL_TABLE=GENERATE_ONLY を使用して、SQL*Loader で実行が試行される SQL 文を作成します。これらの SQL 文を抽出し、参照するディレクトリ・オブジェクトを、アクセス権限を所有しているディレクトリ・オブジェクトに変更します。その後、これらの SQL 文を実行します。

複数表のロードを行う場合は、SQL*Loaderで次の手順を実行します。

1. 任意の表にロードされる入力データ・ファイルのすべてのフィールドを記述する表を、データベースに作成します。
2. INSERT文を作成して、データの外部表記述からこの表をロードします。
3. 制御ファイルのすべての表にINSERT文を実行します。

実行例については、事例5にEXTERNAL_TABLE=GENERATE_ONLYパラメータを追加して実行します。外部表に一意の名前を保証するため、SQL*Loaderではすべてのフィールド用に生成された名前を使用します。これは、フィールド名が制御ファイル内の異なる表の間で一意でないことがあるためです。

関連項目:

- 事例の使用方法の詳細は、「SQL*Loaderの事例」を参照してください
- 外部表の概要
- ORACLE_LOADERアクセス・ドライバ

制限事項

- SQL*Loaderを介して外部表からデータベース表にデータを挿入する場合、ユリウス日は使用できません。この問題を解決するには、次の例に示すとおり、TO_DATEおよびTO_CHARを使用してユリウス日書式を変換します。

```
TO_CHAR(TO_DATE(:COL1, 'MM-DD-YYYY'), 'J')
```

- 外部表からデータベース表にデータを挿入する場合、オブジェクト要素には組込み関数およびSQL文字列は使用できません。

例

```
EXTERNAL_TABLE=EXECUTE
```

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.18 FILE

FILE SQL*Loaderコマンドライン・パラメータでは、エクステントの割当て元となるデータベース・ファイルを指定します。

デフォルト

デフォルトはありません。

用途

FILEパラメータでは、エクステントの割当て元となるデータベース・ファイルを指定します。

関連項目:

パラレル・データ・ロード・モデル

構文および説明

```
FILE=tablespace_file
```

SQL*Loaderの異なるプロセスに対するFILEパラメータの値を変えることによって、データがシステムにロードされるときのディスクの競合を最小限に抑えることができます。

制限事項

- FILEパラメータは、ダイレクト・パスのパラレル・ロードでのみ使用します。

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.19 HELP

SQL*Loaderのコマンドライン・パラメータHELPでは、SQL*Loaderユーティリティのオンライン・ヘルプを表示します。

デフォルト

FALSE

構文および説明

```
HELP = [TRUE | FALSE]
```

HELP=TRUEを指定した場合は、SQL*LoaderによってすべてのSQL*Loaderのコマンドライン・パラメータの要約が表示されます。

すべてのSQL*Loaderのコマンドライン・パラメータの要約は、コマンドラインでsqlldr -helpと入力して表示することもできます。

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.20 LOAD

SQL*Loaderのコマンドライン・パラメータLOADでは、ロードするレコードの最大数を指定します。

デフォルト

すべてのレコードがロードされます。

用途

ロードするレコードの最大数を指定します。

構文および説明

```
LOAD=n
```

ロードに指定したすべてのパラメータが正しく設定されているかどうかをテストする場合、LOADパラメータを使用して、すべてのレコードをロードするかわりに、かぎられた数のレコードを指定します。実際のレコード件数が指定された最大数より少ない場合、エラーは発生しません。

例

次の例では、最大10個のレコードをロードするように指定します。

```
LOAD=10
```

外部表の方法によるロードでは、ロードに成功したレコードのみが合計に加算されます。そのため、入力データ・ファイルに15個のレコードがあり、レコード2と4が不良の場合、レコード1、3、5、6、7、8、9、10、11および12が合計10個のレコードとして表にロードされます。

従来型パス・ロードとダイレクト・パス・ロードでは、成功と失敗の両方のロード試行が合計に加算されます。そのため、入力データ・ファイルに15個のレコードがあり、レコード2と4が不良の場合、レコード1、3、5、6、7、8、9および10の8個のみが実際に表にロードされます。

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.21 LOG

SQL*Loaderのコマンドライン・パラメータLOGでは、SQL*Loaderがロード処理に関するログ情報を保存するログ・ファイルのディレクトリ・パスとファイル名(またはそのどちらか)を指定します。

デフォルト

カレント・ディレクトリ(値を指定しない場合)。

用途

SQL*Loaderがロード処理に関するログ情報を保存するために使用するログ・ファイルのディレクトリ・パスとファイル名(またはそのどちらか)を指定します。

構文および説明

```
LOG=[[directory/][log_file_name]]
```

LOGパラメータを指定する場合、ディレクトリ名またはファイル名(あるいはその両方)を指定する必要があります。

ディレクトリ名を指定しない場合、デフォルトでカレント・ディレクトリが使用されます。

ファイル名なしでディレクトリ名を指定した場合、デフォルトのログ・ファイル名が使用されます。

例

次の例では、カレント・ディレクトリにemp1.logというログ・ファイルを作成します。拡張子を指定しない場合でも、デフォルトの.logが使用されます。

```
LOG=emp1
```

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.22 MULTITHREADING

SQL*Loaderのコマンドライン・パラメータMULTITHREADINGでは、サーバー・システムでのストリームのロードと同時にクライアント・システムでのストリームの構築を実行できるようにします。

デフォルト

複数CPUシステムではTRUE、単一CPUシステムではFALSE。

構文および説明

```
MULTITHREADING=[TRUE | FALSE]
```

デフォルトでは、マルチスレッド・オプションは常にTRUEに設定され、複数CPUシステム上で使用可能です。この場合の複数CPUシステムの定義は、2つ以上のCPUを持つ単一システムです。

単一CPUシステムでは、マルチスレッドはデフォルトでFALSEに設定されています。2つの単一CPUシステム間でマルチスレッドを使用するには、マルチスレッドを使用可能にする必要があります。デフォルトでは、使用可能になっていません。

制限事項

- MULTITHREADINGパラメータは、ダイレクト・パス・ロードでのみ使用できます。
- マルチスレッド機能は、オペレーティング・システムに依存します。すべてのオペレーティング・システムがマルチスレッドをサポートしているわけではありません。

例

次の例では、単一CPUシステムでマルチスレッドを使用可能にします。複数CPUシステムでは、デフォルトで使用可能です。

```
MULTITHREADING=TRUE
```

関連トピック

- 複数CPUシステムのダイレクト・パス・ロードの最適化

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.23 NO_INDEX_ERRORS

SQL*Loaderのコマンドライン・パラメータNO_INDEX_ERRORSでは、ダイレクト・パス・ロード中に索引付けエラーを許容するかどうかを指定します。

デフォルト

FALSE

構文および説明

```
NO_INDEX_ERRORS=[TRUE | FALSE]
```

NO_INDEX_ERRORS=FALSEの設定では、ダイレクト・パス・ロードの結果として索引が使用できなくなった場合、行がロードされ、索引は使用禁止状態のまま残されます。これはデフォルトの動作です。

NO_INDEX_ERRORS=TRUEの設定では、ダイレクト・パス・ロードの結果として索引付けエラーが発生すると、ロードは中断されます。どの行もロードされず、索引はそのままの状態で残されます。

制限事項

- NO_INDEX_ERRORSパラメータは、ダイレクト・パス・ロードでのみ有効です。従来型パス・ロードで指定した場合、このパラメータは無視されます。

例

```
NO_INDEX_ERRORS=TRUE
```

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.24 PARALLEL

SQL*LoaderのPARALLELパラメータでは、ダイレクト・パスまたは外部表を使用するロードで、同じ表に複数の同時セッションでデータをロードできるかどうかを指定します。

カレント・ディレクトリ

FALSE

用途

ダイレクト・パスまたは外部表を使用するロードによって、複数の同時セッションで同じ表にデータをロードできるかどうかを指定します。

構文および説明

```
PARALLEL=[TRUE | FALSE]
```

制限事項

- PARALLELパラメータは、従来型パス・ロードでは無効です。

例

次の例では、パラレルにロードを実行するように指定します。

```
PARALLEL=TRUE
```

関連トピック

- SQL*Loaderのパラレル・データ・ロード・モデルについて

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.25 PARFILE

SQL*Loaderのコマンドライン・パラメータPARFILEでは、頻繁に使用するコマンドライン・パラメータを格納したファイルの名前を指定します。

デフォルト

デフォルトはありません。

構文および説明

```
PARFILE=file_name
```

コマンドラインで各パラメータを指定するかわりに、単にパラメータ・ファイルの名前を指定できます。たとえば、daily_report.parというパラメータ・ファイルの内容が、次のとおりだとします。

```
USERID=scott
```

```
CONTROL=daily_report.ctl
ERRORS=9999
LOG=daily_report.log
```

セキュリティ上の理由から、パラメータ・ファイルにはUSERIDのパスワードを含めないでください。コマンドラインでパラメータ・ファイルを指定すると、SQL*Loaderからパスワードの入力を求められます。たとえば:

```
sqlldr PARFILE=daily_report.par
Password: password
```

制限事項

- システムによっては、パラメータ指定で等号(=)の前後に空白を挿入できないものもあります。

例

「構文および説明」の例を参照してください。

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.26 PARTITION_MEMORY

SQL*Loaderのコマンドライン・パラメータPARFILEでは、多数のパーティションのロード時に使用するメモリーの量を指定します。

デフォルト

0 (ゼロ)この設定は、PGA_AGGREGATE_TARGET初期化パラメータの値に基づいてメモリーの使用を制限します。メモリーの使用がこの値に達すると、一部のパーティションのロードが遅延します。

用途

多数のパーティションのロード時に使用するメモリーの量を指定します。このパラメータは、ロードするパーティションの数が大量のメモリーを消費し、使用可能なメモリーを超える可能性がある場合に役立ちます。(この状況は、特にデータが圧縮されている場合に発生する可能性があります)。

指定した制限に達すると、メモリーの使用量が制限値を下回るまで、一部のパーティション行のロードが延期されます。

構文および説明

```
PARTITION_MEMORY=n
```

パラメータ値nの単位は、KBです。

nが0 (デフォルト)に設定されている場合、SQL*LoaderはPGA_AGGREGATE_TARGET初期化パラメータによって決まる値を使用します。

nが-1 (マイナス1)に設定されている場合、SQL*Loaderは多数のパーティションをロードする際のメモリーの使用量を減らしません。

制限事項

- このパラメータは、ダイレクト・パス・ロードでのみ有効です。
- このパラメータは、Oracle Database 12cリリース1 (12.1.0.2)以降でのみ使用可能です

例

次の例では、メモリーの使用を1GBに制限しています。

```
> sqlldr hr CONTROL=t.ctl DIRECT=true PARTITION_MEMORY=1000000
```

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.27 READSIZE

デフォルトを使用しない場合は、SQL*Loaderのコマンドライン・パラメータREADSIZEで読取りバッファのサイズ(バイト単位)を指定します。

デフォルト

1048576

構文および説明

```
READSIZE=n
```

従来型パス・ロードの方法では、バインド配列は、読取りバッファのサイズによって制限されます。そのため、読取りバッファを大きくすると、コミット操作を実行する前に、より多くのデータを読み取ることができるというメリットがあります。

たとえば、READSIZEを1000000に設定すると、SQL*Loaderはコミットが必要になるまでにデータ・ファイルから1,000,000バイト単位の読取りを実行できます。

ノート:

READSIZE の値に BINDSIZE より小さい値を指定した場合、READSIZE の値は増分されます。

制限事項

- READSIZEパラメータは、データ・ファイルからデータを読み込む場合にのみ使用されます。制御ファイルからレコードを読み込む場合は、常に64KBという値がREADSIZEとして使用されます。
- READSIZEパラメータは、ラージ・オブジェクト(LOB)には反映されません。LOB読取りバッファのサイズは、64KBに固定されています。
- 最大許容サイズはプラットフォームによって異なります。

例

次の例では、読取りバッファのサイズを500,000バイトに設定します(この設定では、デフォルト値またはデフォルトよりも大きな値を使用した場合よりも頻繁なコミット操作が必要になります)。

```
READSIZE=500000
```

関連トピック

- BINDSIZE

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.28 RESUMABLE

SQL*Loaderのコマンドライン・パラメータRESUMABLEでは、再開可能な領域割当てを有効および無効にします。

デフォルト

FALSE

用途

再開可能な領域割当てを有効化および無効化します。

構文および説明

```
RESUMABLE=[TRUE | FALSE]
```

関連項目:

再開可能な領域割当ての詳細は、『Oracle Database管理者ガイド』を参照してください。

制限事項

- このパラメータはデフォルトでは無効なため、関連するRESUMABLE_NAMEパラメータおよびRESUMABLE_TIMEOUTパラメータを使用するには、RESUMABLE=TRUEを設定する必要があります。

例

次の例では、再開可能な領域割当てを有効にします。

```
RESUMABLE=TRUE
```

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.29 RESUMABLE_NAME

SQL*LoaderのコマンドラインパラメータRESUMABLE_NAMEでは、再開可能な文を特定します。

デフォルト

```
'User USERNAME(USERID), Session SESSIONID, Instance INSTANCEID'
```

構文および説明

```
RESUMABLE_NAME='text_string'
```

この値はユーザー定義のテキスト文字列で、USER_RESUMABLEまたはDBA_RESUMABLEビューに挿入して、一時停止されている特定の再開可能な文を識別できます。

制限事項

- RESUMABLEパラメータをTRUEに設定して、再開可能な領域割当てを有効にしないかぎり、このパラメータは無視されます。

例

```
RESUMABLE_NAME='my resumable sql'
```

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.30 RESUMABLE_TIMEOUT

SQL*LoaderのコマンドラインパラメータRESUMABLE_TIMEOUTでは、エラー修正までに必要な時間(秒単位)を指定します。

デフォルト

7200秒(2時間)

構文および説明

```
RESUMABLE_TIMEOUT=n
```

タイムアウト時間内にエラーを修正できない場合は、文の実行が途中で終了します。

制限事項

- RESUMABLEパラメータをTRUEに設定して、再開可能な領域割当てを有効にしないかぎり、このパラメータは無視されます。

例

次の例では、10分(600秒)以内にエラーを修正する必要があることを指定します。

```
RESUMABLE_TIMEOUT=600
```

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.31 ROWS

従来型パス・ロードの場合は、SQL*Loaderのコマンドライン・パラメータROWSでバインド配列の行数を指定し、ダイレクト・パス・ロードの場合は、保存前にデータ・ファイルから読み取る行数を指定します。

デフォルト

バインド配列の行数を指定します。従来型パスのデフォルトは64です。ダイレクト・パスのデフォルトはすべての行です。

用途

従来型パス・ロードの場合は、ROWSパラメータでバインド配列の行数を指定します。ダイレクト・パス・ロードの場合は、ROWSパラメータでデータの保存前にデータ・ファイルからSQL*Loaderが読み取る行数を指定します。

構文

```
ROWS=n
```

従来型パス・ロードの説明

従来型パス・ロードの場合のみ、ROWSパラメータでバインド配列の行数を指定します。行の最大数は65534です。

ダイレクト・パス・ロードの説明

ダイレクト・パス・ロードの場合のみ、データの保存前にデータ・ファイルから読み取る行数をROWSパラメータで特定します。デフォルトでは、ロード終了時に、すべての行の読取りおよびデータ・セーブが1回実行されます。1回のセーブで実際に表にロードされる行のおよその数は、ROWSの値から、最後のセーブ以降に廃棄および拒否されたレコードの数を引いた数です。

ノート:

ROWS に小さい値を指定したときに、表圧縮を使用してデータを圧縮しようとすると、通常は圧縮比が低下します。データの圧縮時には、大きな値を指定するか、デフォルト値をそのまま使用してください。

制限事項

- ダイレクト・パス・ロードでは、索引構成表(IOT)にデータをロードする場合や、VARRAY型、XML列またはラージ・オブジェクト(LOB)を格納している表にデータをロードする場合に、ROWSパラメータが無視されます。これは、ロードはそのまま実行されますが、セーブポイントは処理されないことを意味します。

- ダイレクト・パス・ロードでは、LONG VARCHARデータ型のデータがLOBとして格納されるため、ROWSパラメータは使用できません。ダイレクト・パス・ロードでROWSパラメータをLONG VARCHARデータとともに使用しようとすると、ORA-39777エラーが表示されます(LOB列のロード時のデータ・セーブは許可されません)。

例

従来型パス・ロードでは、指定した値が許容可能な最大行数の65534を超えているため、次の例はエラーになります。

```
ROWS=65900
```

関連トピック

- データ・セーブを使用したデータ損失の防止

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.32 SDF_PREFIX

SQL*LoaderのコマンドラインSDF_PREFIXでは、ロード操作の一環として開かれるLOBFILEおよびセカンダリ・データ・ファイル(SDF)のファイル名に追加するディレクトリ接頭辞を指定します。

デフォルト

デフォルトはありません。

用途

ロード操作の一環として開かれるLOBFILEおよびセカンダリ・データ・ファイル(SDF)のファイル名に追加するディレクトリ接頭辞を指定します。

ノート:

SDF_PREFIX パラメータは、SQL Loader 制御ファイルの OPTIONS 句で指定することもできます。

構文および説明

```
SDF_PREFIX=string
```

SDF_PREFIXを指定した場合は、文字列値も指定する必要があります。文字列の妥当性チェックや検証は行われません。SDF_PREFIXの値は、ロード中に開かれたすべてのLOBFILEおよびSDFに使用されるファイル名の先頭に追加されます。結果の文字列が有効なファイル名でない場合、そのファイルを開こうとすると失敗し、エラーが報告されます。

SDF_PREFIXを指定しない場合、LOBFILEおよびSDFのファイル名は現在の作業ディレクトリに相対的であるとみなされます。SDF_PREFIXを使用すると、それらのファイル名を異なるディレクトリに対して相対的にできます。

コマンドライン・パーサーにとって紛らわしい文字(空白など)が文字列に含まれている場合にのみ、文字列を引用符で囲む必要があります。

レコードで検出されたファイル名の先頭にSDF_PREFIXを追加することによって作成されたファイル名は、ファイルを開くためにオペレーティング・システムに渡されます。接頭辞は、SQL*Loaderが実行されている現在の作業ディレクトリに相対的にするか、または絶対パスの先頭にすることができます。

制限事項

- LOBFILEまたはSDFのファイル指定に完全なファイル名が含まれている場合は、SDF_PREFIXパラメータを使用しないでください。

例

次のSQL*Loaderコマンドでは、現在のディレクトリのlobdirサブディレクトリでLOBファイルが検索されます。

```
sqlldr control=picts.ctl log=picts.log sdf_prefix=lobdir/
```

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.33 SILENT

SQL*Loaderのコマンドライン・パラメータSILENTでは、SQL*Loaderの操作中に画面に出力される内容の一部を抑止します。

デフォルト

デフォルトはありません。

構文および説明

```
SILENT=[HEADER | FEEDBACK | ERRORS | DISCARDS | PARTITIONS | ALL]
```

適切な値を使用して、次の1つ以上の内容を抑止します(複数のオプションを指定する場合、カンマで区切る必要があります)。

- HEADER: 通常は画面に表示されるSQL*Loaderのヘッダー・メッセージを非表示にします。ただし、ヘッダー・メッセージはログ・ファイルに出力されます。
- FEEDBACK: 通常はロードの際に画面に表示される「コミット・ポイントに到達」メッセージとステータス・メッセージを非表示にします。ただし、画面には、引き続き「XX Rows successfully loaded.」と出力されます。
- ERRORS: Oracleエラーがレコードで生成される際に発生する、ログ・ファイル内のデータ・エラー・メッセージを抑止します。ただし、拒否レコード件数は出力されます。
- DISCARDS: 廃棄ファイルに書き込まれる各レコードについて、ログ・ファイル内のメッセージを抑止します。
- PARTITIONS: パーティション表のダイレクト・ロード中に、パーティション毎統計のログ・ファイルへの書込みを無効にします。
- ALL: すべての抑止値(HEADER、FEEDBACK、ERRORS、DISCARDSおよびPARTITIONS パラメータ)を実装します。ただし、画面には、引き続き「XX Rows successfully loaded.」と出力されます。

例

通常は画面に表示されるヘッダーとフィードバックのメッセージが表示されないようにするには、次のコマンドライン引数を指定します。

```
SILENT=HEADER, FEEDBACK
```

ただし、画面には、引き続き「XX Rows successfully loaded.」と出力されます。

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.34 SKIP

SQL*Loaderのコマンドライン・パラメータSKIPでは、ファイルの先頭からのロードが不要な論理レコード数を指定します。

デフォルト

0 (レコードをスキップしません)。

用途

ファイルの先頭からのロードが不要な論理レコード数を指定します。この指定により、すでに処理されているレコードをロードすることなく、なんらかの理由で中断されたロードを続行できます。

構文および説明

```
SKIP=n
```

SKIPパラメータは、すべての従来型ロード、単一表のダイレクト・パス・ロード、各表に同じ数のレコードがロードされた複数表のダイレクト・パス・ロードに使用できます。各表にロードされたレコード数が異なる場合は、複数表のダイレクト・パス・ロードにSKIPを使用することはできません。

WHEN句を使用しているときに、セカンダリ・データもロードの対象になる場合、そのセカンダリ・データは、プライマリ・データ・ファイルのレコードに対してWHEN句が正常に実行された場合にのみスキップされます。

制限事項

- SKIPパラメータは、外部表のロードには使用できません。

例

次の例では、ロードを続行する前にデータ・ファイル内の最初の500件の論理レコードをスキップします。

```
SKIP=500
```

関連トピック

- 中断されたロード

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.35 SKIP_INDEX_MAINTENANCE

SQL*Loaderのコマンドライン・パラメータSKIP_INDEX_MAINTENANCEでは、ダイレクト・パス・ロードの索引メンテナンスを停止するかどうかを指定します。

デフォルト

FALSE

用途

ダイレクト・パス・ロードの索引メンテナンスを停止するかどうかを指定します。

構文および説明

```
SKIP_INDEX_MAINTENANCE=[TRUE | FALSE]
```

このパラメータをTRUEに設定すると、索引パーティションには、索引キーが追加されるかわりに索引使用禁止が設定されます(索引セグメントと、その索引が付けられているデータとの整合性がとれないため)。ロードの影響を受けない索引セグメントについては、ロード前の状態が保持されます。

SKIP_INDEX_MAINTENANCEパラメータ:

- ローカル索引とグローバル索引の両方に適用できます。
- 索引のあるオブジェクトのパラレル・ロードを実行するために使用できます(PARALLELパラメータとともに指定)。
- グローバル索引を持つ表に単一パーティションをロードできます(INTO TABLE句でPARTITIONパラメータを指定)。

- ロードで索引使用禁止状態に設定された索引や索引パーティションのリストを(SQL*Loaderログ・ファイルに)記録します。

制限事項

- SKIP_INDEX_MAINTENANCEパラメータは、従来型パス・ロードには適用されません。
- 一意で使用禁止状態の索引に対しては、索引メンテナンスをスキップできません。この原則は、DML操作の場合にも、ダイレクト・パスでDMLと整合性を持つロードを行う場合にも適用されます。

例

次の例では、ダイレクト・パス・ロードの操作中に索引メンテナンスが発生しないようにします。

```
SKIP_INDEX_MAINTENANCE=TRUE
```

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.36 SKIP_UNUSABLE_INDEXES

SQL*LoaderのコマンドラインSKIP_UNUSABLE_INDEXESでは、索引使用禁止状態で出現した索引をスキップし、ロード操作を続行するかどうかを指定します。

デフォルト

初期化パラメータ・ファイルで指定した、Oracle Database構成パラメータSKIP_UNUSABLE_INDEXESの値。デフォルトのデータベースの設定は、TRUEです。

用途

索引使用禁止状態で出現した索引をスキップし、ロード操作を続行するかどうかを指定します。

構文および説明

```
SKIP_UNUSABLE_INDEXES=[TRUE | FALSE]
```

SKIP_UNUSABLE_INDEXESの値としてTRUEを指定すると、索引使用禁止状態の索引が検出された場合、使用禁止状態の索引はスキップされ、ロード処理は続行されます。これにより、ロード開始前の状態が使用禁止になっている索引を含む表をSQL*Loaderでロードできるようになります。ロード時に使用禁止状態でない索引は、SQL*Loaderによってメンテナンスされます。ロード時に使用禁止状態になっている索引はメンテナンスされなくなりますが、使用禁止状態はロード完了時にも維持されます。

SKIP_UNUSABLE_INDEXESパラメータは、SQL*LoaderとOracle Databaseの両方によって提供されます。SQL*LoaderのSKIP_UNUSABLE_INDEXESパラメータは、SQL*Loaderコマンドラインで指定します。Oracle DatabaseのSKIP_UNUSABLE_INDEXESパラメータは、初期化パラメータ・ファイルの構成パラメータとして指定します。相互に与える影響を理解しておく必要があります。

SQL*LoaderコマンドラインでSKIP_UNUSABLE_INDEXESの値を指定すると、SQL*Loaderは、初期化パラメータ・ファイルに指定されているSKIP_UNUSABLE_INDEXES構成パラメータの値を上書きします。

SQL*LoaderコマンドラインでSKIP_UNUSABLE_INDEXESの値を指定しない場合、SQL*Loaderは、初期化パラメータ・ファイルに指定されているSKIP_UNUSABLE_INDEXES構成パラメータのOracle Database設定を使用します。初期化パラメータ・ファイルでSKIP_UNUSABLE_INDEXESの設定を指定しない場合、デフォルトの設定はTRUEになります。

SKIP_UNUSABLE_INDEXESパラメータは、従来型パス・ロードおよびダイレクト・パス・ロードの両方に適用できます。

制限事項

- 一意で使用禁止状態の索引に対しては、索引メンテナンスをスキップできません。この原則は、DML操作の場合にも、ダイレクト・パスでDMLと整合性を持つロードを行う場合にも適用されます。

例

Oracle Database初期化パラメータの値がSKIP_UNUSABLE_INDEXES=FALSEに設定されていても、SQL*Loaderコマンドラインでの SKIP_UNUSABLE_INDEXES=TRUEの設定が優先されます。そのため、このパラメータの設定後に索引使用禁止状態の索引が出現すると、その索引はスキップされてロード操作が続行されます。

```
SKIP_UNUSABLE_INDEXES=TRUE
```

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.37 STREAMSIZE

SQL*LoaderのコマンドラインパラメータSTREAMSIZEでは、クライアントからサーバーに送信されるデータ・ストリームのサイズ(バイト単位)を指定します。

デフォルト

256000

用途

クライアントからサーバーに送信されるデータのストリーム・サイズ(バイト単位)を指定します。

構文および説明

```
STREAMSIZE=n
```

STREAMSIZEパラメータでは、ダイレクト・パス・ストリーム・バッファのサイズを指定します。列配列の行数(COLUMNARRAYROWSパラメータで指定)によって、ストリーム・バッファが作成される前にロードされた行数を判断します。これらのパラメータの最適値は、使用されるシステム、入力データ型およびOracleの列データ型に応じて異なります。独自の構成用に最適な値を使用することで、SQL*Loaderのログ・ファイルでの経過時間が少なくなります。

制限事項

- STREAMSIZEパラメータは、ダイレクト・パス・ロードにのみ適用されます。
- STREAMSIZEの最小値は65536です。65536未満の値が指定された場合、かわりに65536が使用されます。

例

次の例では、ダイレクト・パス・ストリーム・バッファのサイズを300,000バイトに指定します。

```
STREAMSIZE=300000
```

関連トピック

- 列配列の行数およびストリーム・バッファ・サイズの指定

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.38 TRIM

SQL*LoaderのコマンドラインパラメータTRIMでは、テキスト・フィールドの先頭とテキスト・フィールドの末尾の両方(またはどち

らか)から空白を切り捨てるかどうかを指定します。

デフォルト

```
LDRTRIM
```

用途

テキスト・フィールドの先頭とテキスト・フィールドの末尾の両方(またはどちらか)から空白を切り捨てるかどうかを指定します。空白には、空白文字およびその他の印字されない文字(タブ、行送り、改行など)が含まれます。

構文および説明

```
TRIM=[LRTRIM | NOTRIM | LTRIM | RTRIM | LDRTRIM]
```

TRIMパラメータの有効な値は次のとおりです。

- NOTRIMは、フィールドから文字を切り捨てないように指示ます。この設定では、通常、最高のパフォーマンスを得ることができます。
- LRTRIMは、フィールドから先頭と末尾の両方の空白を切り捨てるように指示します。
- LTRIMは、フィールドから先頭の空白を切り捨てるように指示ます。
- RTRIMは、フィールドから末尾の空白を切り捨てるように指示ます。
- LDRTRIMは、次の場合を除いて、NOTRIMと同様です。
  - フィールドがデリミタ付きフィールドではない場合、空白は右から切り捨てられます。
  - フィールドがOPTIONALLY ENCLOSED BYで指定されたデリミタ付きフィールドで、オプションの囲みが特定のインスタンスで欠落している場合、空白は左から切り捨てられます。

ノート:

空白のみのフィールドに対して切捨てを指定すると、そのフィールドは NULL に設定されます。

制限事項

- TRIMパラメータは、外部表ロード方法が使用される場合にのみ有効です。

例

次の例では、どのフィールドからも文字が切り捨てられないロード操作を指定します。

```
TRIM=NOTRIM
```

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.2.39 USERID

SQL*LoaderのコマンドラインパラメータUSERIDでは、SQL*LoaderのOracleユーザー名とパスワードを指定します。

デフォルト

デフォルトはありません。

用途

SQL*LoaderのOracleユーザー名とパスワードを指定して、その入力を求められないようにします。省略すると、入力要求が表示されます。この値としてスラッシュ(/)を指定すると、USERIDはオペレーティング・システム・ログインにデフォルト設定されます。

構文および説明

```
USERID=[username | / | SYS]
```

ユーザー名を指定します。セキュリティ上の理由から、コマンドラインにはユーザー名のみを指定することをお薦めします。SQL*Loaderによってパスワードの入力を要求されます。

USERIDパラメータを指定しない場合、その入力を求められます。スラッシュ(斜線)を使用すると、USERIDはオペレーティング・システムのログインにデフォルト設定されます。

ユーザーSYSとして接続する場合は、接続文字列にAS SYSDBAも指定する必要があります。

制限事項

- 文字列AS SYSDBAには空白が含まれるため、一部のオペレーティング・システムでは、接続文字列全体をなんらかの方法でリテラルとしてマーク(一重引用符で囲むなど)する必要があります。また、オペレーティング・システムによっては、コマンドラインの引用符をバックスラッシュなどでエスケープすることが必要になる場合もあります。

  システムの特殊文字および予約文字の詳細は、対象オペレーティング・システム固有のマニュアルを参照してください。

例

次の例では、ユーザー名のhrを指定します。SQL*Loaderによってパスワードの入力を要求されます。これが指定する最初で唯一のパラメータであるため、パラメータ名のUSERIDを含める必要はありません。

```
> sqlldr hr
Password:
```

関連トピック

- コマンドラインでのパラメータの指定

親トピック: SQL*Loaderのコマンドライン・パラメータ

8.3 終了コードによる結果の検査と表示

Oracle SQL*Loaderでは、SQL*Loaderの完了後、すぐに実行結果を確認できます。

使用上のノート

SQL*Loaderでは、実行結果をログ・ファイルに記録するのみでなく、プロセス終了コードでレポートすることもできます。。このOracle SQL*Loaderの機能によって、コマンドラインやスクリプトからSQL*Loaderを起動したときにもその結果を確認できます。次の表に、様々な結果の終了コードを示します。

表8-1 SQL*Loaderの終了コード

結果	終了コード
すべての列が正常にロードされた	EX_SUCC
すべての行または一部の行が拒否された	EX_WARN

結果	終了コード
すべての行または一部の行が廃棄された	EX_WARN
ロードが中断された	EX_WARN
コマンドラインまたは構文エラー	EX_FAIL
SQL*Loader に対してリカバリ不能な Oracle エラー	EX_FAIL
OS 関連エラー(ファイルのオープン/クローズ、malloc など)	EX_FTL

例

LinuxおよびUnixオペレーティング・システムでは、終了コードは次のとおりです。

```
EX_SUCC 0
EX_FAIL 1
EX_WARN 2
EX_FTL  3
```

Windowsオペレーティング・システムでは、終了コードは次のとおりです。

```
EX_SUCC 0
EX_FAIL 1
EX_WARN 2
EX_FTL  4
```

SQL*Loaderがゼロ以外の終了コードを返した場合、システム・ログ・ファイルおよびSQL*Loaderログ・ファイルで、詳細な診断情報を確認してください。

Unixプラットフォームでは、シェルの終了コードを調べてロード結果を確認できます。

親トピック: SQL*Loaderコマンドライン・リファレンス

9 SQL*Loader制御ファイル・リファレンス

SQL*Loader制御ファイルは、SQL*Loaderジョブ用のデータ定義言語(DDL)命令を含むテキスト・ファイルです。

ノート:

SQL*Loader は制御ファイルなしで使用することもできます(これは SQL*Loader エクスプレス・モードと呼ばれます)。詳細は、「SQL*Loader エクスプレス」を参照してください。

- 制御ファイルの内容
  SQL*Loader制御ファイルは、データ定義言語(DDL)命令を含むテキスト・ファイルです。
- 制御ファイルのコメント
  コメントはファイル中のパラメータ部分のどこにでも記述できますが、データの部分には記述できません。
- 制御ファイル中でのコマンドライン・パラメータの指定
  SQL*Loader制御ファイルでは、OPTIONS句を使用してコマンドライン・パラメータを指定できます。
- ファイル名およびオブジェクト名の指定
  通常、オブジェクト名(表名や列名など)の指定に関して、SQL*LoaderはSQL標準規則に準拠しています。
- XMLTypeの表の識別
  XMLタイプの表を識別および選択してロードするには、SQL*Loader制御ファイルでXMLTYPE句を使用します。
- フィールド順序の指定
  SQL*Loader制御ファイルでFIELD NAMES句を使用して、フィールド順序を指定できます。
- データ・ファイルの指定
  SQL*Loader制御ファイルを使用して、データ・ファイルのロード方法を指定する方法について説明します。
- CSV形式のファイルの指定
  SQL*Loaderでカンマ区切り値形式のファイルであるデータ・ファイルにアクセスするには、CSV句を使用します。
- BEGINDATAによる制御ファイルのデータの識別
  最初のデータ・レコードの前に、BEGINDATA文を指定します。
- データ・ファイル形式およびバッファリングの指定
  os_file_proc_clauseを使用して、オペレーティング・システムに依存したファイル処理指定文字列のオプションを指定できます。
- 不良ファイルの指定
  SQL*Loader不良ファイルの意味と、そのファイルの指定方法について説明します。
- 廃棄ファイルの指定
  SQL*Loader廃棄ファイルの意味と、そのファイルの指定方法について説明します。
- 表レベルでのNULLIF句の指定
  表の文字フィールドに特定の文字列または16進文字列が格納されているときに、そのフィールドをNULLとしてロードするには、SQL*Loaderで表レベルのNULLIF句を使用できます。
- 表レベルでの日時書式の指定
  SQL*Loader制御ファイルにおいて表レベルで特定の日時書式を指定するか、フィールド・レベルでマスクを指定することで表レベルの書式を上書きできます。
- 異なる文字コード体系の処理
  SQL*Loaderでは、(文字セットまたはコード・ページと呼ばれる)異なる文字コード体系がサポートされます。

- 中断されたロード
  ロードは様々な理由で中断されます。
- 物理レコードからの論理レコードの作成
  この項では、物理レコードから論理レコードを作成する方法について説明します。
- 表への論理レコードのロード
  この項では、論理レコードを表へロードする方法について説明します。
- 索引オプション
  この項では、索引オプションについて説明します。
- 複数のINTO TABLE句を使用するメリット
  これらの項では、複数のINTO TABLE句を使用するメリットについて説明します。
- バインド配列および従来型パス・ロード
  一度に複数の行が読み込まれてバインド配列に格納されます。

関連トピック

- SQL*Loaderエクスプレス

親トピック: SQL*Loader

9.1 制御ファイルの内容

SQL*Loader制御ファイルは、データ定義言語(DDL)命令を含むテキスト・ファイルです。

DDLを使用して、SQL*Loaderセッションの次の項目を制御します。

- SQL*Loaderによってロードされるデータの位置
- SQL*Loaderによるデータ書式設定の方法
- データのロード時のSQL*Loaderの設定(メモリー管理、拒否レコード、ロード処理の中断など)
- SQL*Loaderによるロード中のデータの処理方法

SQL*LoaderのDDL構文図の詳細は、「SQL*Loaderの構文図」を参照してください。

SQL*Loader制御ファイルを作成するには、viやxemacsなどのテキスト・エディタを使用します。

通常、制御ファイルには、次の3つの項目が次の順序で含まれます。

- セッション全体の情報
- 表およびフィールド・リストの情報
- 入力データ(オプションの項目)

制御ファイルの例を次に示します。

例9-1 制御ファイル

```
1     -- This is an example control file
2     LOAD DATA
3     INFILE 'sample.dat'
4     BADFILE 'sample.bad'
5     DISCARDFILE 'sample.dsc'
6     APPEND
7     INTO TABLE emp
8     WHEN (57) = '.'
9     TRAILING NULLCOLS
10   (hiredate SYSDATE,
       deptno POSITION(1:2)  INTEGER EXTERNAL(2)
               NULLIF deptno=BLANKS,
```

```
      job     POSITION(7:14)  CHAR  TERMINATED BY WHITESPACE
              NULLIF job=BLANKS  "UPPER(:job)",
      mgr     POSITION(28:31) INTEGER EXTERNAL
              TERMINATED BY WHITESPACE, NULLIF mgr=BLANKS,
      ename   POSITION(34:41) CHAR
              TERMINATED BY WHITESPACE  "UPPER(:ename)",
      empno   POSITION(45) INTEGER EXTERNAL
              TERMINATED BY WHITESPACE,
      sal     POSITION(51) CHAR  TERMINATED BY WHITESPACE
              "TO_NUMBER(:sal,'$99,999.99')",
      comm    INTEGER EXTERNAL  ENCLOSED BY '(' AND '%'
              ":comm * 100"
   )
```

この制御ファイル例での左側の数字は、実際の制御ファイルでは表示されません。これらの数字は、次の説明の番号に対応しています。

1. 制御ファイル内のエントリの前にあるこのコメントは、制御ファイルにコメントを入力する方法の例です。詳細は、「制御ファイルのコメント」を参照してください。
2. LOAD DATA文によって、SQL*Loaderで新しくデータ・ロードが開始されます。構文の詳細は、「SQL*Loaderの構文図」を参照してください。
3. INFILE句には、ロードするデータが入っているデータ・ファイルの名前を指定します。詳細は、「データ・ファイルの指定」を参照してください。
4. BADFILE句には、拒否レコードが書き込まれるファイルの名前を指定します。詳細は、「不良ファイルの指定」を参照してください。
5. DISCARDFILE句には、廃棄レコードが書き込まれるファイルの名前を指定します。詳細は、「廃棄ファイルの指定」を参照してください。
6. APPEND句は、空ではない表にデータをロードする場合に使用できるオプションの1つです。詳細は、「空でない表へのデータのロード」を参照してください。

   空の表にデータをロードするには、INSERT句を使用します。詳細は、「空の表へのデータのロード」を参照してください。
7. INTO TABLE句を使用して、表、フィールドおよびデータ型を識別できます。この句で、データ・ファイル内のレコード間、およびデータベース内の表の関係を定義します。詳細は、「表名の指定」を参照してください。
8. WHEN句には、1つ以上のフィールド条件を指定します。SQL*Loaderで、この条件に基づいてデータをロードするかどうかを判断します。詳細は、「条件に基づいたレコードのロード」を参照してください。
9. TRAILING NULLCOLS句を使用すると、相対位置で指定した列がレコード中に存在しない場合、その列の値はNULLとして処理されます。詳細は、「データが欠落しているショート・レコードの処理」を参照してください。
10. 制御ファイルの残りの部分には、ロード中の表の列形式の詳細を参照できるフィールド・リストが含まれます。制御ファイルのセクションの詳細は、「SQL*Loaderフィールド・リスト・リファレンス」を参照してください。

親トピック: SQL*Loader制御ファイル・リファレンス

9.2 制御ファイルのコメント

コメントはファイル中のパラメータ部分のどこにでも記述できますが、データの部分には記述できません。

コメントの前にハイフンを2つ続けて記述します。たとえば:

```
--This is a comment
```

二重ハイフンの右側のすべてのテキストは行末まで無視されます。

親トピック: SQL*Loader制御ファイル・リファレンス

9.3 制御ファイル中でのコマンドライン・パラメータの指定

SQL*Loader制御ファイルでは、OPTIONS句を使用してコマンドライン・パラメータを指定できます。

これは、同じオプションの組合せで何度も制御ファイルを使用する場合に役立ちます。OPTIONS句は、LOAD DATA文の前に置きます。

- OPTIONS句
  OPTIONS句を使用して、次のコマンドライン・パラメータを指定できます。
- 一度に評価されるデフォルト式の数の指定
  SQL*LoaderのDEFAULT EXPRESSION CACHE n句を使用して、ダイレクト・パス・ロードで一度に評価されるデフォルト式の数を指定します。デフォルト値は100です。

親トピック: SQL*Loader制御ファイル・リファレンス

9.3.1 OPTIONS句

OPTIONS句では、次のコマンドライン・パラメータを指定できます。

これらのパラメータの説明は、「SQL*Loaderコマンドライン・リファレンス」を参照してください。

```
BINDSIZE = n
COLUMNARRAYROWS = n
DATE_CACHE = n
DEGREE_OF_PARALLELISM= {degree-num|DEFAULT|AUTO|NONE}
DIRECT = {TRUE | FALSE}
EMPTY_LOBS_ARE_NULL = {TRUE | FALSE}
ERRORS = n
EXTERNAL_TABLE = {NOT_USED | GENERATE_ONLY | EXECUTE}
FILE = tablespace file
LOAD = n
MULTITHREADING = {TRUE | FALSE}
PARALLEL = {TRUE | FALSE}
READSIZE = n
RESUMABLE = {TRUE | FALSE}
RESUMABLE_NAME = 'text string'
RESUMABLE_TIMEOUT = n
ROWS = n
SDF_PREFIX = string
SILENT = {HEADER | FEEDBACK | ERRORS | DISCARDS | PARTITIONS | ALL}
SKIP = n
SKIP_INDEX_MAINTENANCE = {TRUE | FALSE}
SKIP_UNUSABLE_INDEXES = {TRUE | FALSE}
STREAMSIZE = n
TRIM= {LRTRIM|NOTRIM|LTRIM|RTRIM|LDRTRIM}
```

次に、SQL*Loader制御ファイルで使用できるOPTIONS句の使用例を示します。

```
OPTIONS (BINDSIZE=100000, SILENT=(ERRORS, FEEDBACK) )
```

ノート:

コマンドラインで指定したパラメータ値は、制御ファイルの OPTIONS 句で指定したパラメータ値を上書きしま

す。

親トピック: 制御ファイル中でのコマンドライン・パラメータの指定

9.3.2 一度に評価されるデフォルト式の数の指定

SQL*LoaderのDEFAULT EXPRESSION CACHE n句を使用して、ダイレクト・パス・ロードで一度に評価されるデフォルト式の数を指定します。デフォルト値は100です。

DEFAULT EXPRESSION CACHE句を使用すると、順序を含むデフォルト列式が評価される場合のパフォーマンスが大幅に向上します。

ロードが終了するとき、使用されない順序番号がキャッシュに残される場合があります。このことは、ロードする行数がnの倍数でない場合に発生することがあります。順序番号の損失を防ぐ必要がある場合は、この句に対して1の値を指定します。

親トピック: 制御ファイル中でのコマンドライン・パラメータの指定

9.4 ファイル名およびオブジェクト名の指定

通常、オブジェクト名(表名や列名など)の指定に関して、SQL*LoaderはSQL標準規則に準拠しています。

- SQLおよびSQL*Loaderの予約語と競合するファイル名
  SQLおよびSQL*Loaderの予約語と、特殊文字を含むか大/小文字を区別する単語は、引用符で囲む必要があります。
- SQL*Loader制御ファイルでのSQL文字列の指定
  SQL文字列を保持するフィールド・データにSQL演算子を適用するときには、二重引用符で囲んだSQL文字列を指定する必要があります。
- オペレーティング・システムとSQL Loader制御ファイルの文字
  制御ファイルで使用する文字は、オペレーティング・システムの予約文字、エスケープ文字および特殊文字の影響を受けます。

親トピック: SQL*Loader制御ファイル・リファレンス

9.4.1 SQLおよびSQL*Loaderの予約語と競合するファイル名

SQLおよびSQL*Loaderの予約語と、特殊文字を含むか大/小文字を区別する単語は、引用符で囲む必要があります。

SQLおよびSQL*Loaderの予約語は、二重引用符で囲みます。

SQL*Loaderの予約語はCONSTANTのみです

二重引用符は、オブジェクト名にSQLが認識可能な文字($、#、_)以外の特殊文字が含まれている場合、またはオブジェクト名で大/小文字を区別する場合に使用する必要があります。

関連トピック

- 『Oracle Database SQL言語リファレンス』の「OracleでのSQLの予約語およびキーワード」

親トピック: ファイル名およびオブジェクト名の指定

9.4.2 SQL*Loader制御ファイルでのSQL文字列の指定

SQL文字列を保持するフィールド・データにSQL演算子を適用するときには、二重引用符で囲んだSQL文字列を指定する必

要があります。

関連項目:

フィールドへのSQL演算子の適用

親トピック: ファイル名およびオブジェクト名の指定

9.4.3 オペレーティング・システムとSQL Loader制御ファイルの文字

制御ファイルで使用する文字は、オペレーティング・システムの予約文字、エスケープ文字および特殊文字の影響を受けます。

使用中のオペレーティング・システムが、SQL*Loader制御ファイルで使用できる文字に与える影響について学習します。

- 完全パスの指定
  パス名を一重引用符で囲んで指定すると、このエラーは回避できます。
- バックスラッシュ・エスケープ文字の使用
  DDL構文では、(ご使用のオペレーティング・システムでエスケープ文字の使用が許可されている場合)二重引用符の前にバックスラッシュ・エスケープ文字(¥)を付けて、二重引用符で区切られた文字列の中で二重引用符を使用できます。
- 移植不能文字列
  SQL*Loader制御ファイルには、オペレーティング・システム間で移植不能な、2種類の文字列があります(filename文字列およびfile processing option文字列)。
- エスケープ文字としてのバックスラッシュの使用
  オペレーティング・システムとデータベースの両方でバックスラッシュのエスケープ文字が実装されている場合、パス名においてディレクトリを区切るには、バックスラッシュ文字を使用します。
- エスケープ文字が使用できない場合
  オペレーティング・システムによっては、Oracle Databaseの移植不能文字列にエスケープ文字を使用できないことがあります。

親トピック: ファイル名およびオブジェクト名の指定

9.4.3.1 完全パスの指定

パス名を一重引用符で囲んで指定すると、このエラーは回避できます。

完全パス名を指定すると、問題が発生する場合があります。これは、特殊文字の使用が原因でオペレーティング・システム依存の非互換が発生するためです。

親トピック: オペレーティング・システムとSQL Loader制御ファイルの文字

9.4.3.2 バックスラッシュ・エスケープ文字

DDL構文では、(ご使用のオペレーティング・システムでエスケープ文字の使用が許可されている場合)二重引用符の前にバックスラッシュ・エスケープ文字(¥)を付けて、二重引用符で区切られた文字列の中で二重引用符を使用できます。

一重引用符で区切られた文字列中で一重引用符を使用する場合も、その前にバックスラッシュを付けます。

たとえば、`homedir¥data"norm¥mydata`という文字列には、二重引用符が含まれています。二重引用符の直前にバックスラッシュを付けることによって、二重引用符を文字列として使用できます。

```
INFILE 'homedir¥data¥"norm¥mydata'
```

また、バックスラッシュを2回続けて書くことによって、バックスラッシュ自体を文字列中に表示できます。

たとえば:

```
"so'¥"far"      or  'so¥'"far'      is parsed as   so'"far
"'so¥¥far'"     or  '¥'so¥¥far¥''   is parsed as  'so¥far'
"so¥¥¥¥far"     or  'so¥¥¥¥far'     is parsed as   so¥¥far
```

ノート:

先頭位置の二重引用符はエスケープできません。そのため、先頭に引用符の付く文字列は作成しないでください。

親トピック: オペレーティング・システムとSQL Loader制御ファイルの文字

9.4.3.3 移植不能文字列

SQL*Loader制御ファイルには、オペレーティング・システム間で移植不能な、2種類の文字列があります(filename文字列およびfile processing option文字列)。

別のオペレーティング・システム用に変換した場合、これらの文字列を修正する必要があります。SQL*Loader制御ファイルのこれ以外の文字列はすべて、異なるオペレーティング・システム間で移植可能です。

親トピック: オペレーティング・システムとSQL Loader制御ファイルの文字

9.4.3.4 エスケープ文字としてのバックスラッシュの使用

オペレーティング・システムとデータベースの両方でバックスラッシュのエスケープ文字が実装されている場合、パス名においてディレクトリを区切るには、バックスラッシュ文字を使用します。

オペレーティング・システムで、パス名のディレクトリを区切る文字としてバックスラッシュが使用されている場合、およびオペレーティング・システムで実行されているOracle Databaseのリリースで、ファイル名およびその他の移植不能文字列に対してバックスラッシュをエスケープ文字として使用可能な場合は、パス名でのディレクトリ間の区切りにバックスラッシュを2つ続けて指定して、パス名全体を一重引用符で囲みます。

親トピック: オペレーティング・システムとSQL Loader制御ファイルの文字

9.4.3.5 エスケープ文字が使用できない場合

オペレーティング・システムによっては、Oracle Databaseの移植不能文字列にエスケープ文字を使用できないことがあります。

オペレーティング・システムでエスケープ文字としてバックスラッシュ文字(¥)が使用できない場合、バックスラッシュはエスケープ文字ではなく通常の文字として処理されます。それ以外のすべての文字列で、引き続きバックスラッシュ文字を使用できます。このオペレーティング・システムの制限の結果として、次のようなパス名は問題なく指定できます。

```
INFILE 'topdir¥mydir¥myfile'
```

バックスラッシュを2つ続ける必要はありません。

バックスラッシュはエスケープ文字として認識されないため、一重引用符で囲まれた文字列の中に、一重引用符で囲まれた別の文字列を埋め込むことはできません。このルールは、二重引用符の使用にも適用されます。二重引用符で囲まれた文字列は、二重引用符で区切られた他の文字列の中に埋め込むことはできません。

親トピック: オペレーティング・システムとSQL Loader制御ファイルの文字

9.5 XMLType表の識別

XMLタイプの表を識別および選択してロードするには、SQL*Loader制御ファイルでXMLTYPE句を使用します。

Oracle Database 10gでは、SQL*Loader制御ファイルでXMLTYPE句を使用できます。形式は、XMLTYPE(フィールド名)です。この句を使用するとXMLTypeの表を識別できるため、適切なSQL文を作成できるようになります。SQL*Loader制御ファイルでXMLTYPE句を使用すると、スキーマベースのXMLType表にデータをロードできます。

例9-2 SQL*Loader制御ファイルでのXMLType表の識別

XMLスキーマ定義は次のとおりです。XMLスキーマxdb_user.xsdがOracle XML DBに登録され、表xdb_tab5が作成されます。

```
begin dbms_xmlschema.registerSchema('xdb_user.xsd',
'<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns:xdb="http://xmlns.oracle.com/xdb">
 <xs:element name = "Employee"
        xdb:defaultTable="EMP31B_TAB">
   <xs:complexType>
    <xs:sequence>
      <xs:element name = "EmployeeId" type = "xs:positiveInteger"/>
      <xs:element name = "Name" type = "xs:string"/>
      <xs:element name = "Salary" type = "xs:positiveInteger"/>
      <xs:element name = "DeptId" type = "xs:positiveInteger"
             xdb:SQLName="DEPTID"/>
    </xs:sequence>
   </xs:complexType>
 </xs:element>
</xs:schema>',
TRUE, TRUE, FALSE); end;
/
```

表を定義する方法は、次のとおりです。

```
CREATE TABLE xdb_tab5 OF XMLTYPE XMLSCHEMA "xdb_user.xsd" ELEMENT "Employee";
```

次の例では、制御ファイルを使用して表xdb_tab5にデータをロードします。この制御ファイルでは、登録済のXMLスキーマxdb_user.xsdを使用してXMLTypeデータをロードします。XMLTYPE句を使用して、この表をXMLType表として識別します。表にデータをロードするために、ダイレクト・パス・モードまたは従来型モードのどちらかを使用できます。

```
LOAD DATA
INFILE *
INTO TABLE xdb_tab5 TRUNCATE
xmltype(xmldata)
(
  xmldata   char(4000)
)
BEGINDATA
<Employee>  <EmployeeId>111</EmployeeId>  <Name>Ravi</Name>  <Salary>100000</Sal
ary>  <DeptId>12</DeptId></Employee>
<Employee>  <EmployeeId>112</EmployeeId>  <Name>John</Name>  <Salary>150000</Sal
ary>  <DeptId>12</DeptId></Employee>
<Employee>  <EmployeeId>113</EmployeeId>  <Name>Michael</Name>  <Salary>75000</S
alary>  <DeptId>12</DeptId></Employee>
<Employee>  <EmployeeId>114</EmployeeId>  <Name>Mark</Name>  <Salary>125000</Sal
ary>  <DeptId>16</DeptId></Employee>
<Employee>  <EmployeeId>115</EmployeeId>  <Name>Aaron</Name>  <Salary>600000</Sa
lary>  <DeptId>16</DeptId></Employee>
```

関連トピック

- XMLTypeの表の識別

親トピック: SQL*Loader制御ファイル・リファレンス

9.6 フィールド順序の指定

SQL*Loader制御ファイルでFIELD NAMES句を使用して、フィールド順序を指定できます。

構文は次のとおりです。

```
FIELD NAMES {FIRST FILE|FIRST FILE IGNORE|ALL FILES|ALL FILES IGNORE|NONE}
```

FIELD NAMESのオプションは次のとおりです。

- FIRST FILE: 最初のデータ・ファイルに、最初のレコードのデータに対応するフィールド名のリストが含まれることを示します。このリストでは、データ・ファイルのデータと同じデリミタが使用されます。レコードは、データ・ファイルのフィールドとターゲット表の列とのマッピングを設定するために読み取られます。レコードは、データが処理されるとスキップされます。これは、データ・ファイルのフィールドの順序が表の列の順序と異なる場合や、データ・ファイルのフィールドの数がターゲット表の列の数と異なる場合に役立ちます。
- FIRST FILE IGNORE: 最初のデータ・ファイルに、最初のレコードのデータに対応するフィールド名のリストが含まれるが、その情報は無視する必要があることを示します。レコードは、データが処理されるとスキップされますが、フィールドの設定には使用されません。
- ALL FILES: すべてのデータ・ファイルに、最初のレコードのデータに対応するフィールド名のリストが含まれることを示します。各データ・ファイルの最初のレコードは、データが処理されるとスキップされます。フィールドは、データ・ファイルごとに異なる順序で配置できます。SQL*Loaderは、各データ・ファイルのフィールドの順序に基づいてロードを設定します。
- ALL FILES IGNORE: すべてのデータ・ファイルに、最初のレコードのデータに対応するフィールド名のリストが含まれるが、その情報は無視する必要があることを示します。レコードは、すべてのデータ・ファイルでデータが処理されるとスキップされますが、フィールドの設定には使用されません。
- NONE: データ・ファイルに、最初のレコードの通常のデータが含まれることを示します。これはデフォルトです。

FIELD NAMES句は、列オブジェクト、ネストした表、VARRAYなどの複合列型では使用できません。

親トピック: SQL*Loader制御ファイル・リファレンス

9.7 データ・ファイルの指定

SQL*Loader制御ファイルを使用して、データ・ファイルのロード方法を指定する方法を学習します。

- データ・ファイルの指定方法の理解
  SQL*Loaderでデータ・ファイルをロードする場合は、制御ファイル内でINFILEキーワードを使用するとデータ・ファイルを指定できます。
- INFILE構文の例
  次に、INFILE構文を指定する様々な方法を示します。
- 複数のデータ・ファイルの指定
  1回のSQL*Loaderの実行で複数のデータ・ファイルのデータをロードする場合は、各データ・ファイルに対してINFILE句を指定します。

親トピック: SQL*Loader制御ファイル・リファレンス

9.7.1 データ・ファイルの指定方法の理解

SQL*Loaderでデータ・ファイルをロードする場合は、制御ファイル内でINFILEキーワードを使用するとデータ・ファイルを指定できます。

ロードするデータを含むデータ・ファイルを指定するには、INFILEキーワードにファイル名を続け、必要な場合はファイル処理オプション文字列を続けます。

指定する単一ファイルが複数ある場合は、INFILEキーワードを複数使用できます。ファイル名にワイルドカードを使用することもできます(アスタリスク(*)は複数文字を、疑問符(?)は1文字を表します)。

> ノート:
>
> また、コマンドラインから DATA パラメータを使用してデータ・ファイルを指定することもできます。SQL*Loader で使用可能なコマンドライン・パラメータを参照してください。コマンドラインでファイル名を指定すると、制御ファイルの最初の INFILE 句が上書きされます。

ファイル名が指定されない場合は、制御ファイル名の拡張子(ファイルのタイプ)を.datにしたファイル名がデフォルトで採用されます。

ロードする必要があるデータを制御ファイル内にも記述した場合は、ファイル名にアスタリスク(*)を指定してください。この指定の詳細は、トピック「BEGINDATAによる制御ファイルのデータの識別」を参照してください。

> ノート:
>
> ここで説明する内容は、プライマリ・データ・ファイルのみに適用されます。LOBFILE または SDF には適用されません。

INFILEの構文は次のとおりです。

次の表では、INFILEキーワードのパラメータについて説明します。

表9-1 INFILEキーワードのパラメータ

パラメータ	説明
INFILE	データ・ファイル指定が続くことを示します。

input_filename	データが入っているファイルの名前。ファイル名にはワイルドカードを使用できます。アスタリスク(*)は複数文字を表し、疑問符(?)は 1 文字を表します。たとえば:
	`INFILE 'emp*.dat'`
	`INFILE 'm?emp.dat'`
	ファイル名に空白や句読点文字が含まれている場合は、一重引用符で囲む

パラメータ	説明
	必要があります。
*	データが制御ファイル中にある場合は、ファイル名のかわりにアスタリスク(*)を指定します。制御ファイルとデータ・ファイルにデータがあるときに、そのデータを読み取るには、最初にアスタリスクを指定する必要があります。
os_file_proc_clause	ファイル処理オプション文字列。データ・ファイルの形式を指定します。また、この指定によってデータ・ファイルの読取りを最適化できます。この文字列の構文は、ご使用のオペレーティング・システムに依存します。

関連トピック

- BEGINDATAによる制御ファイルのデータの識別
- ファイル名およびオブジェクト名の指定
- データ・ファイル形式およびバッファリングの指定

親トピック: データ・ファイルの指定

9.7.2 INFILE構文の例

次に、INFILE構文を指定する様々な方法を示します。

- 制御ファイルにデータがある場合

```
INFILE  *
```

- デフォルトの拡張子.datを持つファイルsampleにデータがある場合

```
INFILE  sample
```

- フルパスに指定されたファイルdatafile.datにデータがある場合

```
INFILE 'c:/topdir/subdir/datafile.dat'
```

> ノート:
>
> ファイル名に空白やその他の句読点文字が含まれている場合は、ファイル名を一重引用符で囲む必要があります。

- 名前がempで始まる.datタイプの任意のファイルにデータがある場合

```
INFILE 'emp*.dat'
```

- 名前がmで始まり、その後に他の任意の1文字が続き、empで終わる.datタイプの任意のファイルにデータがある場合。たとえば、myemp.datという名前のファイルが次の指定に含まれます。

```
INFILE 'm?emp.dat'
```

親トピック: データ・ファイルの指定

9.7.3 複数のデータ・ファイルの指定

1回のSQL*Loaderの実行で複数のデータ・ファイルのデータをロードする場合は、各データ・ファイルに対してINFILE句を指定します。

このとき、レコードのレイアウトは同じである必要がありますが、データ・ファイルに同じファイル処理オプションは必要ありません。たとえば、最初と2番目のファイルが異なるファイル処理オプション文字列で指定されていても、3番目のファイルが制御ファイル中のデータで構成されていても実行することができます。

各データ・ファイルに、個別の不良ファイルおよび廃棄ファイルを指定することもできます。このような場合、個別の不良ファイルおよび廃棄ファイルは、各データ・ファイル名の直後に宣言する必要があります。たとえば、制御ファイルからの次の抜粋は、4つのデータ・ファイルを個別の不良ファイルおよび廃棄ファイルとともに指定します:

```
INFILE  mydat1.dat  BADFILE  mydat1.bad  DISCARDFILE mydat1.dis
INFILE  mydat2.dat
INFILE  mydat3.dat  DISCARDFILE  mydat3.dis
INFILE  mydat4.dat  DISCARDMAX  10 0
```

- mydat1.datでは、不良ファイルと廃棄ファイルの両方が明示的に指定されています。したがって、必要に応じて両方のファイルが作成されます。
- mydat2.datでは、不良ファイルも廃棄ファイルも指定されていません。そのため、不良ファイルのみが必要に応じて作成されます。不良ファイルが作成された場合、デフォルトのファイル名および拡張子mydat2.badが使用されます。一方、廃棄ファイルについては、行が廃棄されても廃棄ファイルは作成されません。
- mydat3.datでは、必要に応じてデフォルトの不良ファイルが作成されます。また、必要に応じて、指定された名前の廃棄ファイル(mydat3.dis)も作成されます。
- mydat4.datでは、必要に応じてデフォルトの不良ファイルが作成されます。DISCARDMAXオプションが指定されているため、SQL*Loaderによって廃棄ファイルが使用されることを前提として、廃棄ファイルがデフォルト名(mydat4.dsc)で作成されます。

親トピック: データ・ファイルの指定

9.8 CSV形式のファイルの指定

SQL*Loaderでカンマ区切り値形式のファイルであるデータ・ファイルにアクセスするには、CSV句を使用します。

これによって、ファイルは、通常の改行文字列(たとえば、UNIXまたはLinuxオペレーティング・システムの場合は¥n、Windowsオペレーティング・システムの場合は¥nまたは¥r¥n)を含むストリーム・レコード形式のファイルであるとみなされます。データ値にはレコード終了記号含める(埋め込む)ことができます。CSV句の構文は次のとおりです。

```
FIELDS CSV [WITH EMBEDDED|WITHOUT EMBEDDED] [FIELDS TERMINATED BY ','] [OPTIONALLY
ENCLOSED BY '"']
```

FIELDS CSV句の主なポイントは次のとおりです。

- SQL*Loaderのデフォルトでは、FIELDS CSV句を使用しません。
- WITH EMBEDDEDおよびWITHOUT EMBEDDEDオプションでは、データのフィールド内にレコード終了記号が含まれている(埋め込まれている)かどうかを指定します。
- WITH EMBEDDEDを使用する場合、埋込みのレコード終了記号は囲む必要があり、データ・ファイル内の並列度は外部表ロードでは無効になります。

- TERMINATED BY ','およびOPTIONALLY ENCLOSED BY '"'オプションはデフォルトのため、指定する必要はありません。これらは別の終了記号や囲み文字で上書きできます。
- CSV句を使用する場合、区切ることが可能なデータ型のみが制御ファイルのフィールドとして許可されます。区切ることが可能なデータ型には、CHAR、日時、期間および数値型EXTERNALが含まれます。
- CSV句を指定する場合、TERMINATED BYおよびENCLOSED BY句は、フィールド・レベルでは使用できません。
- CSV句を指定すると、通常のSQL*Loaderの空白の切捨てがデフォルトで実行されます。PRESERVE BLANKSを指定すると、空白の切捨てを回避できます。または、フィールド指定でSQL関数のLTRIMおよびRTRIMを使用すると、左側または右側(あるいはその両方)の空白を削除できます。
- CSV句を指定する場合、INFILE *句は使用できません。つまり、SQL*Loader制御ファイルに含まれているデータは使用できません。

次のサンプルSQL*Loader制御ファイルでは、デフォルトのデリミタでFIELDS CSV句を使用しています。

```
LOAD DATA
INFILE "mydata.dat"
TRUNCATE
INTO TABLE mytable
FIELDS CSV WITH EMBEDDED
TRAILING NULLCOLS
(
  c0 char,
  c1 char,
  c2 char,
)
```

親トピック: SQL*Loader制御ファイル・リファレンス

9.9 BEGINDATAによる制御ファイルのデータの識別

最初のデータ・レコードの前に、BEGINDATA文を指定します。

データが制御ファイルにある場合、ファイル名ではなくアスタリスクがINFILE句の後に続きます。実際のデータは、ロード構成の指定後、制御ファイルに書き込まれます。

構文は次のとおりです。

```
BEGINDATA
first_data_record
```

BEGINDATA文を使用する場合は、次のことに注意してください。

- 制御ファイルにデータが含まれているにもかかわらず、BEGINDATA文を省略すると、SQL*Loaderでデータが制御情報として解釈されるため、エラー・メッセージが発行されます。データが個別ファイルにある場合は、BEGINDATA文は使用できません。
- BEGINDATAを含む行がデータの先頭行として解釈されてしまうため、BEGINDATA文と同じ行には、空白またはその他の文字を入力しないでください。
- コメントもデータとして解釈されてしまうため、コメントをBEGINDATAの後に続けて記述しないでください。

関連トピック

- INFILE構文の例
- SQL*Loaderの事例

親トピック: SQL*Loader制御ファイル・リファレンス

9.10 データ・ファイル形式およびバッファリングの指定

os_file_proc_clauseを使用して、オペレーティング・システムに依存したファイル処理指定文字列のオプションを指定できます。

SQL*Loaderの設定時、制御ファイルの形式およびバッファリングの指定に対して、オペレーティング・システムに依存したファイル処理オプション文字列(os_file_proc_clause)を制御ファイルに指定できます。

たとえば、ご使用のオペレーティング・システムに、次のようなオプション文字列の構文があるとします。

この構文では、RECSIZEは固定長レコードのサイズ、BUFFERSは非同期I/Oを行うためのバッファ数です。

mydata.datというファイルを80バイトのレコードを含むファイルとして宣言し、SQL*LoaderのI/O処理で使用されるバッファ数を8とする場合は、次の制御ファイルのエントリを使用します。

```
INFILE 'mydata.dat' "RECSIZE 80 BUFFERS 8"
```

ノート:

この例では、ファイル名に一重引用符を使用し、その他の指定には二重引用符を使用しています。

関連トピック

- 『Oracle Database管理者リファレンスfor Microsoft Windows』の「Windowsの処理オプション」

親トピック: SQL*Loader制御ファイル・リファレンス

9.11 不良ファイルの指定

SQL*Loader不良ファイルの意味と、そのファイルの指定方法を学習します。

- 不良ファイルの理解と指定
  SQL*Loaderを実行すると、不良ファイルまたは拒否ファイルと呼ばれるファイルが生成されることがあります。これらのファイルには、書式エラーまたはOracleエラーの発生が原因でロードを拒否されたレコードが格納されます。
- 不良ファイル名指定の例
  SQL*Loader制御ファイル内の不良ファイルをファイル名、ファイル名と拡張子、またはディレクトリ別に指定する方法を参照してください。
- LOBFILEおよびSDFを使用した不良ファイルの処理方法
  SQL*Loaderでは、LOBFILEおよびSDFのデータのエラーは別々に管理されます。
- 拒否レコードの基準
  SQL*Loaderで従来型パス・ロードおよびダイレクト・パス・ロードでのレコードの拒否に適用される基準について説明します。

親トピック: SQL*Loader制御ファイル・リファレンス

9.11.1 不良ファイルの理解と指定

SQL*Loaderを実行すると、不良ファイルまたは拒否ファイルと呼ばれるファイルが生成されることがあります。これらのファイルには、書式エラーまたはOracleエラーが発生したためにロードを拒否されたレコードが格納されます。

不良ファイルが作成されるように指定すると、次の処理が発生します。

- レコードが1つ以上拒否された場合、不良ファイルが作成され、拒否レコードがログに記録されます。
- レコードが1つも拒否されない場合、不良ファイルは作成されません。
- 不良ファイルが作成される場合、同じ名前の既存のファイルは上書きされます。そのため、残しておきたいファイルが上書きされないようにする必要があります。

ノート:

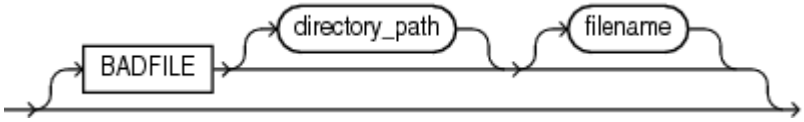
一部のシステムでは、同じ名前のファイルがすでに存在する場合でも、新しいバージョンのファイルが作成される場合があります。

不良ファイルの名前を指定する場合は、BADFILE句を使用します。また、コマンドラインからBADパラメータを使用して不良ファイルを指定することもできます。

コマンドラインで指定したファイル名は、制御ファイルの最初のINFILE句に関連付けられます。この関連付けにより、その句の一部として以前に指定された不良ファイルが存在する場合は、そのファイルがオーバーライドされます。

生成される不良ファイルのファイル形式やレコード形式は、データ・ファイルと同じです。そのため、データを修正した後そのまま再ロードできます。ストリーム・レコード形式のデータ・ファイルに関しては、データ・ファイルにあるレコード終了記号が不良ファイル内でも使用されます。

BADFILE句の構文は次のとおりです。



BADFILE句は、その後に不良ファイルのディレクトリ・パスまたはファイル名(あるいはその両方)が続くことを指定します。BADFILEを指定した場合、ディレクトリ・パス、ファイル名またはこの両方を指定する必要があります。

directoryパラメータには、不良ファイルの書込み先となるディレクトリ・パスを指定します。

filenameパラメータには、使用するプラットフォームに有効なファイル名を指定します。ファイル名に空白やその他の句読点文字が含まれている場合は、一重引用符で囲む必要があります。不良ファイルの名前を指定しなかった場合は、デフォルトで、データ・ファイル名に拡張子(またはファイル・タイプ).badを付けたものがファイル名になります

関連トピック

- SQL*Loaderのコマンドライン・パラメータ

親トピック: 不良ファイルの指定

9.11.2 不良ファイル名指定の例

SQL*Loader制御ファイル内の不良ファイルをファイル名、ファイル名と拡張子、またはディレクトリ別に指定する方法を参照してください。

ファイル名sampleおよびデフォルトのファイル拡張子(またはファイル・タイプ) .badで不良ファイルを指定するには、制御ファイルで次のように入力します。

```
BADFILE sample
```

ディレクトリ名のみを指定するには、制御ファイルで次のように入力します。

```
BADFILE '/mydisk/bad_dir/'
```

ファイル名bad0001およびファイル拡張子(またはファイル・タイプ) .rejで不良ファイルを指定するには、制御ファイルで次のいずれかの行を入力します。

```
BADFILE bad0001.rej
BADFILE '/REJECT_DIR/bad0001.rej'
```

親トピック: 不良ファイルの指定

9.11.3 LOBFILEおよびSDFを使用した不良ファイルの処理方法

SQL*Loaderでは、LOBFILEおよびSDFのデータのエラーは別々に管理されます。

拒否された行がある場合、SQL*LoaderでLOBFILEおよびSDFのデータが不良ファイルに書き込まれることはありません。

SQL*Loaderでラージ・オブジェクト(LOB)のロード中にエラーが発生した場合、行は拒否されません。この場合、LOB列は、空のまま(NULLではなく長さが0バイト)になります。ただし、LOBFILEを使用してXML列をロードし、このLOBデータのロード中にエラーが発生した場合、XML列はNULLになります。

親トピック: 不良ファイルの指定

9.11.4 拒否レコードの条件

SQL*Loaderで従来型パス・ロードおよびダイレクト・パス・ロードでのレコードの拒否に適用される基準について説明します。

SQL*Loaderでは、次の理由でレコードが拒否される可能性があります。

1. レコードの挿入時に、指定されたデータ型と実際のデータが適合しないなどのOracleエラーが発生する場合。
2. レコードが間違ってフォーマットされたため、SQL*Loaderでフィールドの境界を見つけられない場合。
3. レコードが制約に違反している場合、または一意の索引を一意でなくそうとした場合。

デリミタの指定が不適切であっても、WHEN句の指定条件に基づいてデータを評価できる場合は、条件判断が行われ、データが挿入または拒否されます。

理由リストの2番目の理由でロードが拒否された場合、従来型パス・ロードでもダイレクト・パス・ロードでも表に行は書き込まれません。

前述のリストの1番目または3番目の理由でいずれかの表に違反がある場合、従来型パス・ロードでは、表に行は書込みできません。行はその表に対して拒否され、拒否ファイルに書き込まれます。

従来型パス・ロードでは、データ・ファイルのレコードが複数の表にロードされているときに1つ以上の表から拒否されると、そのレコードはいずれの表にもロードされません。

ログ・ファイルには、拒否レコードそれぞれについてのOracleエラー情報が記録されます。拒否レコードの例の詳細は、「SQL*Loaderの事例」の事例4を参照してください。

関連トピック

- SQL*Loaderの事例

親トピック: 不良ファイルの指定

9.12 廃棄ファイルの指定

SQL*Loader廃棄ファイルの意味と、そのファイルの指定方法を学習します。

- 廃棄ファイルの指定と理解
  レコードの処理中に、ロード条件を満たさないレコードがある場合は、SQL*Loaderで廃棄ファイルが生成されることがあります。
- 制御ファイルでの廃棄ファイルの指定
  ファイルの名前を指定するには、DISCARDFILE句にディレクトリ・パスまたはファイル名(あるいはその両方)を続けます。
- 廃棄レコード数の上限付け
  データ・ファイルごとに廃棄されるレコードの数を制限するには、DISCARDSまたはDISCARDMAXパラメータで整数値を指定します。
- 廃棄ファイル名指定の例
  次に、制御ファイル内から廃棄ファイルの名前を指定する様々な方法を示します。
- 廃棄レコードの条件
  レコードに対してINTO TABLE句が指定されていない場合、そのレコードは廃棄されます。
- LOBFILEおよびSDFを使用した廃棄ファイルの処理方法
  SQL*Loaderでは、廃棄された行がある場合も、ラージ・オブジェクト(LOB)データのLOBFILEおよびセカンダリ・データ・ファイル(SDF)からのデータが廃棄ファイルに書き込まれることはありません。
- コマンドラインからの廃棄ファイルの指定
  SQL*Loaderをコマンドラインから実行するときに廃棄ファイルを指定するには、SQL*LoaderのDISCARDコマンドライン・パラメータを使用します。

親トピック: SQL*Loader制御ファイル・リファレンス

9.12.1 廃棄ファイルの理解と指定

レコードの処理中に、ロード条件を満たさないレコードがある場合は、SQL*Loaderで廃棄ファイルが生成されることがあります。

廃棄ファイルに記録されているレコードは廃棄レコードと呼ばれます。廃棄レコードは、制御ファイル中のWHEN句の指定を満たすことができなかったものです。これらのレコードは、拒否レコードとは異なります。廃棄レコードに必ず不良データがあるとはかぎりません。また、廃棄レコードに対して、挿入は行われません。

廃棄ファイルは次の規則に従って生成されます。

- 廃棄ファイル名を指定し、1つ以上のレコードが制御ファイルに指定したすべてのWHEN句の条件に満たない場合に生成されます。(廃棄ファイルが生成されると、同じ名前の既存のファイルは上書きされることに注意してください。)
- 廃棄レコードがない場合、廃棄ファイルは生成されません。

制御ファイル内で廃棄ファイルを指定するには、そのディレクトリまたは名前(またはその両方)を指定するか、廃棄の最大数を指定します。次のいずれかの句により、廃棄ファイルが必要に応じて作成されます。

- DISCARDFILE=[[directory/][filename]]
- DISCARDS
- DISCARDMAX

生成される廃棄ファイルのファイル形式やレコード形式は、データ・ファイルと同じです。ストリーム・レコード形式のデータ・ファイルに関しては、データ・ファイルにあるレコード終了記号が廃棄ファイル内でも使用されます。

DISCARDパラメータまたはDISCARDMAXパラメータを指定することで、コマンドラインから廃棄ファイルを作成することもできます。

制御ファイルまたはコマンドラインにDISCARD句がない場合、廃棄レコード(つまり、制御ファイルのWHEN句で指定の事項を満たしていないもの)がある場合でも廃棄ファイルは作成されません。

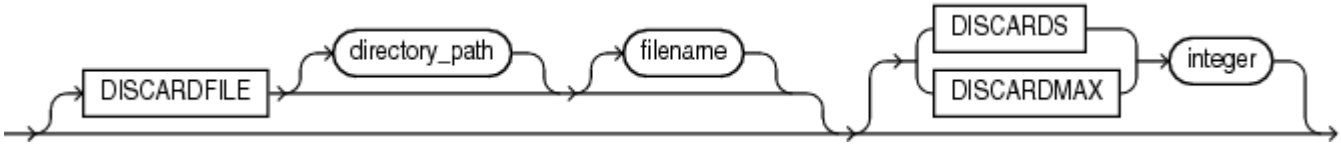
関連トピック

- SQL*Loaderコマンドライン・リファレンス

親トピック: 廃棄ファイルの指定

9.12.2 制御ファイルでの廃棄ファイルの指定

ファイルの名前を指定するには、DISCARDFILE句にディレクトリ・パスまたはファイル名(あるいはその両方)を続けます。



DISCARDFILE句は、その後に廃棄ディレクトリ・パスまたはファイル名(あるいはその両方)が続くことを指定します。directory_pathもfilenameも必須ではありません。ただし、少なくとも1つを指定する必要があります。

directoryパラメータには、廃棄ファイルの書込み先となるディレクトリを指定します。

filenameパラメータには、使用するプラットフォームに有効なファイル名を指定します。ファイル名に空白やその他の句読点文字が含まれている場合は、一重引用符で囲む必要があります。

デフォルトのファイル名は、データ・ファイル名にデフォルトのファイル拡張子(またはファイル・タイプ).dscを付けたものが採用されます。廃棄ファイル名をコマンドラインから指定した場合は、制御ファイル中で指定されたファイル名よりも、コマンドラインからの指定ファイル名が優先されます。生成される廃棄ファイルと同じ名前のファイルがすでに存在する場合は、既存ファイルが上書きされるか、新しいバージョンのファイルが生成されます。どちらの処理が行われるかは、使用するオペレーティング・システムによって異なります。

親トピック: 廃棄ファイルの指定

9.12.3 廃棄レコード数の上限付け

データ・ファイルごとに廃棄されるレコードの数を制限するには、DISCARDSまたはDISCARDMAXパラメータで整数値を指定します。

DISCARDSまたはDISCARDMAXキーワードに指定する整数は、廃棄レコード数の上限です。廃棄レコード数の上限を指定しない場合、SQL*Loaderによってレコードが廃棄され続けます。そうでない場合は、廃棄の上限に達すると、データ・ファイルの処理が中止され、引き続き、次のデータ・ファイル(存在する場合)の処理が実行されます。

データ・ファイルごとに異なる数の上限を設定できます。ただし、1回のみ廃棄数を指定した場合は、指定した廃棄上限値はすべてのファイルに適用されます。

廃棄数の上限値が指定されていて、廃棄ファイル名の指定がない場合、SQL*Loaderによってデフォルトのファイル名(そのファイルの作成プロセスの後に名前が付けられる)とファイル拡張子またはファイル・タイプ(dsc)で廃棄ファイルが作成されます。たとえば、このファイルは、それを作成するプロセスの後に名前が付けられます。たとえば、finance.dscです。

次の例では、ロードが終了するまでに25個の破棄レコードが許可されます。

```
DISCARDMAX=25
```

親トピック: 廃棄ファイルの指定

9.12.4 廃棄ファイル名指定の例

次に、制御ファイル内から廃棄ファイルの名前を指定する様々な方法を示します。

- ファイル名circularおよびデフォルトのファイル拡張子(またはファイル・タイプ).dscで廃棄ファイルを指定するには:

```
DISCARDFILE  circular
```

- ファイル名notapplおよびファイル拡張子(またはファイル・タイプ).mayで廃棄ファイルを指定するには:

```
DISCARDFILE notappl.may
```

- 廃棄ファイルforget.meにフルパスを指定するには:

```
DISCARDFILE  '/discard_dir/forget.me'
```

親トピック: 廃棄ファイルの指定

9.12.5 廃棄レコードの条件

レコードに対してINTO TABLE句が指定されていない場合、そのレコードは廃棄されます。

レコードの廃棄は、SQL*Loader制御ファイル中に記述されたすべてのINTO TABLE句がWHEN句を持っていて、レコードがそれらのどの条件にも該当しない場合、またはすべてのフィールドがNULLである場合に発生します。

INTO TABLE句がWHEN句なしで指定されている場合、レコードは廃棄されません。すべてのレコードに関して、指定の表への挿入が試みられます。このとき、レコードが拒否されることはありますが、廃棄されることはありません。

「事例7: 書式設定されたレポートからのデータの抽出」に廃棄ファイルの使用例があります。(事例の使用方法については、「SQL*Loaderの事例」を参照してください。)

親トピック: 廃棄ファイルの指定

9.12.6 LOBFILEおよびSDFを使用した廃棄ファイルの処理方法

SQL*Loaderでは、廃棄された行がある場合も、ラージ・オブジェクト(LOB)データのLOBFILEおよびセカンダリ・データ・ファイル(SDF)からのデータが廃棄ファイルに書き込まれることはありません。

親トピック: 廃棄ファイルの指定

9.12.7 コマンドラインからの廃棄ファイルの指定

SQL*Loaderをコマンドラインから実行するときに廃棄ファイルを指定するには、SQL*LoaderのDISCARDコマンドライン・パラメータを使用します。

DISCARDパラメータを使用すると、コマンドラインで指定して廃棄ファイル特定できます。このファイルには、表に挿入されず、拒否もされていないレコードを格納できます。

コマンドラインでファイル名を指定した場合は、この指定が、制御ファイル内で指定した廃棄ファイル名より優先されます。

関連トピック

- DISCARD

親トピック: 廃棄ファイルの指定

9.13 表レベルでのNULLIF句の指定

表の文字フィールドに特定の文字列または16進文字列が格納されているときに、そのフィールドをNULLとしてロードするには、SQL*Loaderで表レベルのNULLIF句を使用できます。

SQL*Loader制御ファイルでのNULLIFの構文は、次のとおりです。

```
NULLIF {=|!=}{"char_string"|x'hex_string'|BLANKS}
```

char_stringおよびhex_stringの値は、一重引用符または二重引用符で囲む必要があります。

この指定は、NULLIF句がフィールド・レベルで指定されていないかぎり、マップされた文字フィールドごとに使用されます。フィールド・レベルで指定されたNULLIF句は、表レベルで指定されたNULLIF句に優先します。

SQL*Loaderは、レコードのフィールドの値に基づいて、指定された値を確認します。等号または不等号の指定を使用して一致した場合、その行ではフィールドがNULLに設定されます。空白の切捨て後に長さが0になったフィールドも、すべてNULLに設定されます。

デフォルトのNULLIFまたは他の任意のNULLIF句をフィールドに適用しない場合、フィールド・レベルでNO NULLIFを指定できます。

関連トピック

- WHEN、NULLIFおよびDEFAULTIF句の使用

親トピック: SQL*Loader制御ファイル・リファレンス

9.14 表レベルでの日時書式の指定

SQL*Loader制御ファイルにおいて表レベルで特定の日時書式を指定するか、フィールド・レベルでマスクを指定することで表レベルの書式を上書きできます。

SQL*Loader制御ファイルにおいて、表レベルで特定の日時データ型(**datetime**)の書式を指定できます。

表レベルで指定できる各日時書式の構文は次のとおりです。

```
DATE FORMAT mask
TIMESTAMP FORMAT mask
TIMESTAMP WITH TIME ZONE mask
TIMESTAMP WITH LOCAL TIME ZONE mask
```

この日時指定は、異なるマスクがフィールド・レベルで指定されていないかぎり、日付またはタイムスタンプ・フィールドごとに使用されます。フィールド・レベルで指定されたマスクは、表レベルで指定されたマスクに優先します。

次に、SQL*Loader制御ファイルでのDATE FORMAT句の使用例を示します。DATE FORMAT句は、hiredateおよびentrydateフィールドのフィールド・レベルでのDATEによって上書きされます。

```
LOAD DATA
     INFILE myfile.dat
     APPEND
     INTO TABLE EMP
     FIELDS TERMINATED BY ","
     DATE FORMAT "DD-Month-YYYY"
     (empno,
      ename,
      job,
      mgr,
      hiredate DATE,
      sal,
```

```
      comm,
      deptno,
      entrydate DATE)
```

関連トピック

- 日時および間隔のデータ型のカテゴリ

親トピック: SQL*Loader制御ファイル・リファレンス

9.15 異なる文字コード体系の処理

SQL*Loaderでは、(文字セットまたはコード・ページと呼ばれる)異なる文字コード体系がサポートされます。

SQL*Loaderでは、Oracleグローバリゼーション・サポート・テクノロジの機能を使用して、様々な文字コード体系(シングルバイトおよびマルチバイト)を扱うことができます。

関連項目:

Oracle Databaseグローバリゼーション・サポート・ガイド

次の項では、サポートしている文字コード体系について簡単に説明します。

- マルチバイト(アジア系言語)文字セット
  マルチバイト文字セットは、アジア系言語をサポートするために使用されます。
- Unicode文字セット
  SQL*Loaderでは、Unicode文字セットのデータのロードがサポートされています。
- データベース文字セット
  Oracle Databaseでは、SQLのCHARデータ型(CHAR、VARCHAR2、CLOBおよびLONG)に格納されたデータ、表名などの識別子、SQL文およびPL/SQLソース・コードに対して、データベース文字セットが使用されます。
- データ・ファイル文字セット
  デフォルトでは、データ・ファイルには、NLS_LANGパラメータで定義されている文字セットが使用されます。
- 入力文字セット
  CHARACTERSETパラメータが指定されていない場合、すべてのデータ・ファイルに対するデフォルトの文字セットは、NLS_LANGパラメータで定義されたセッション文字セットです。
- シフト・センシティブ文字データ
  通常、シフト・センシティブ文字データのロードは、ASCIIデータまたはEBCDICデータのロードに比べて非常に遅くなります。

親トピック: SQL*Loader制御ファイル・リファレンス

9.15.1 マルチバイト(アジア系言語)文字セット

マルチバイト文字セットは、アジア系言語をサポートするために使用されます。

データをマルチバイト形式でロードしたり、データベース・オブジェクト名(フィールド、表など)をマルチバイト文字で指定することもできます。制御ファイルでは、コメントおよびオブジェクト名にもマルチバイト文字を使用できます。

親トピック: 異なる文字コード体系の処理

9.15.2 Unicode文字セット

SQL*Loaderでは、Unicode文字セットのデータのロードがサポートされています。

Unicodeは、エンコーディングされたユニバーサル文字セットであり、単一の文字セットでほとんどの言語による情報の格納をサポートします。Unicodeでは、プラットフォーム、プログラムまたは言語に関係なく、すべての文字に一意のコード値が指定されます。Unicodeには、2つの異なるエンコーディング(UTF-16およびUTF-8)が存在します。

ノート:

- このマニュアルでは、UTF-16 および UTF16 の両方の用語が使用されています。UTF-16 という用語は、一般的に Unicode の UTF-16 エンコーディングを意味します。UTF16(ハイフンなし)という用語は、文字セットの固有の名前で、UTF-16 エンコーディングの使用時に CHARACTERSET パラメータに指定する必要のある用語です。UTF-8 および UTF8 の場合も同様です。

  Unicode の UTF-16 エンコーディングは、固定幅マルチバイト・エンコーディングで、文字コード 0x0000 から 0x007f は、シングルバイト ASCII コードの 0x00 から 0x7f と同じ意味です。

  Unicode の UTF-8 エンコーディングは、可変幅マルチバイト・エンコーディングで、文字コード 0x00 から 0x7f が ASCII と同じ意味です。UTF-8 の文字は 1 バイト、2 バイトまたは 3 バイトの長さになります。

- データベースの文字セットには、AL32UTF8 を使用することをお薦めします。AL32UTF8 は、Unicode の UTF-8 エンコーディングの適切な実装です。Oracle Database12c リリース 2 以降では、Oracle Universal Installer (OUI)および Oracle Database Configuration Assistant (DBCA)を使用してデータベースを作成する際、デフォルトのデータベース文字セットとして AL32UTF8 が使用されます。

- UTF8 は Unicode の UTF-8 エンコーディングの適切な実装ではないため、データベース文字セットとして UTF8 を使用しないでください。UTF-8 処理が予期されている場合に UTF8 文字セットが使用されると、データの消失およびセキュリティの問題が発生する場合があります。このことは、XML や URL アドレスなどの Web 関連データに特に該当します。

- AL32UTF8 および UTF8 文字セットは、最大文字幅が異なる(1 文字当たり 4 バイトに対して 3 バイト)ため、相互に互換性がありません。

関連項目:

- 「事例11: Unicode文字セットのデータのロード」(事例の使用方法については、「SQL*Loaderの事例」を参照)
- Unicodeのエンコーディングの詳細は、『Oracle Databaseグローバリゼーション・サポート・ガイド』を参照してください。

親トピック: 異なる文字コード体系の処理

9.15.3 データベース文字セット

Oracle Databaseでは、SQLのCHARデータ型(CHAR、VARCHAR2、CLOBおよびLONG)に格納されたデータ、表名などの識別子、SQL文およびPL/SQLソース・コードに対して、データベース文字セットが使用されます。

シングルバイト文字セットおよびASCII文字またはEBCDIC文字のいずれかを含む可変幅文字セットのみがデータベース文字セットとしてサポートされます。マルチバイト固定幅文字セット(たとえば、AL16UTF16)は、データベース文字セットとしてサポートされません。

SQLのNCHARデータ型(NCHAR、NVARCHAR2およびNCLOB)に格納されたデータ用のデータベースでは、代替文字セットを使用できます。代替文字セットは、データベース各国語文字セットと呼ばれます。Unicode文字セットのみが、データベース各国語文字セットとしてサポートされます。

親トピック: 異なる文字コード体系の処理

9.15.4 データ・ファイル文字セット

デフォルトでは、データ・ファイルには、NLS_LANGパラメータで定義されている文字セットが使用されます。

NLS_LANGでサポートされているデータ・ファイル文字セットは、データベース文字セットとしてサポートされているものと同じです。SQL*Loaderでは、データ・ファイル内のOracleでサポートされているすべての文字セットがサポートされます(データベース文字セットとしてサポートされていないものもサポートされます)。

たとえば、SQL*Loaderでは、データ・ファイル内のマルチバイト固定幅文字セット(AL16UTF16、JA16EUCFIXEDなど)がサポートされます。また、SQL*Loaderでは、リトル・エンディアン・バイト順序によるUTF-16エンコーディングもサポートされます。ただし、Oracle Databaseでは、ビッグ・エンディアン・バイト順序によるUTF-16エンコーディング(AL16UTF16)のみが、データベース文字セットとしてではなく、データベースの各国語文字セットとしてのみサポートされます。

データ・ファイルの文字セットは、NLS_LANGパラメータ、またはSQL*LoaderのCHARACTERSETパラメータを使用して設定できます。

親トピック: 異なる文字コード体系の処理

9.15.5 入力文字の変換

CHARACTERSETパラメータが指定されていない場合、すべてのデータ・ファイルに対するデフォルトの文字セットは、NLS_LANGパラメータで定義されたセッション文字セットです。

入力データ・ファイルで使用される文字セットは、CHARACTERSETパラメータで指定できます。

SQL*Loaderでは、データ・ファイル文字セット、データベース文字セットおよびデータベース各国語文字セットが異なる場合、データ・ファイルのデータをデータベース文字セットまたはデータベース各国語文字セットに自動的に変換できます。

データの文字セット変換が必要な場合、ターゲットの文字セットは、ソースのデータ・ファイル文字セットのスーパーセットである必要があります。そうでない場合、ターゲット文字セットに同じ文字がない文字は、置換文字(通常、疑問符(?)などのデフォルトの文字)に変換されます。これによって、データが失われます。

データベース・キャラクタ・タイプCHARおよびVARCHAR2のサイズは、バイト単位(バイト長セマンティクス)または文字単位(文字長セマンティクス)で指定できます。バイト単位で指定され、データ文字セットの変換が必要な場合、変換される文字のソース文字セットより多くのバイトがターゲット文字セットで使用されると、変換後の値はソースの値より大きくなることがあります。これによって、ターゲットの大きい値がデータベースの列のサイズを超える場合、次のエラー・メッセージがレポートされます。

```
ORA-01401: inserted value too large for column
```

データベースの列サイズを文字単位で指定するか、または制御ファイルの文字サイズを使用してデータを記述することによっても、この問題を回避できます。また、最大列サイズがバイト単位で、変換後の値を保持できる大きさであることを確認する方法もあります。

- VARRAYまたは主キーベースのREFにデータをロードする場合の考慮事項
  VARRAYまたは主キーベースのREFにデータをロードすると、データでデータベースとは異なる文字セットが使用されている場合に問題が発生する可能性があります。
- CHARACTERSETパラメータ
  CHARACTERSETパラメータを指定して、SQL*Loaderに入力データ・ファイルの文字セットを示します。
- 制御ファイル文字セット
  SQL*Loader制御ファイル自体の文字セットは、NLS_LANGパラメータで指定されているセッション文字セットであることが前提です。
- 文字長セマンティクス
  バイト長セマンティクスは、UTF16文字セット(デフォルトで文字長セマンティクスを使用します)を使用するデータ・ファイル以外のすべてのデータ・ファイルのデフォルトです。

関連項目:

- 文字長セマンティクス
- Oracle Databaseグローバリゼーション・サポート・ガイド

親トピック: 異なる文字コード体系の処理

9.15.5.1 VARRAYまたは主キー・ベースREFへのデータ・ロード時の考慮事項

データをVARRAYまたは主キーベースのREFにロードすると、データでデータベースとは異なる文字セットが使用されている場合に問題が発生する可能性があります。

SQL*Loaderの従来型パスまたはOracle Call Interface(OCI)を使用して、VARRAYまたは主キー・ベースREFに、データベース文字セットとは異なる文字セットのデータをロードする場合は、次のような問題が発生する可能性があります。

- データベースの列に対してフィールドが大きすぎるという理由で、実際には大きすぎない場合でも、行が拒否される可能性があります。
- 大きすぎるフィールドのみが拒否されるロード処理が、1行もロードされないまま終了する可能性があります。
- 主キーベースのREF列をSQL*Plusで選択した場合、それらの列が空白として返されるにもかかわらず、行が正しくロードされたようにレポートされる可能性があります。

これらの問題を回避するには、データをロードする前にNLS_LANG環境変数を使用して、クライアントの文字セットをデータベース文字セットに設定します。

親トピック: 入力文字の変換

9.15.5.2 CHARACTERSETパラメータ

CHARACTERSETパラメータを指定して、SQL*Loaderに入力データ・ファイルの文字セットを示します。

CHARACTERSETパラメータが指定されていない場合、すべてのデータ・ファイルに対するデフォルトの文字セットは、NLS_LANGパラメータで定義されたセッション文字セットです。文字データ(SQL*Loaderデータ型のCHAR、VARCHAR、VARCHARC、数値型EXTERNALおよび日時データ型と期間データ型のフィールド)のみが、データ・ファイルの文字セットに影響されます。

CHARACTERSETの構文は次のとおりです。

```
CHARACTERSET char_set_name
```

char_set_name変数で、文字セット名を指定します。通常、指定された名前はOracleがサポートしている文字セットの名

前である必要があります。

UnicodeのUTF-16エンコーディングの場合は、名前はAL16UTF16でなくUTF16を使用します。AL16UTF16(UTF-16エンコードされたデータに対してサポートされるOracle文字セット名)は、ビッグ・エンディアンのバイト順序のUTF-16データにのみ使用されます。ただし、データ・ファイルを作成したシステムのバイト順序を使用してデータを設定することが許可されているため、データ・ファイル内のデータはビッグ・エンディアンまたはリトル・エンディアンのいずれかにできます。このため、異なる文字セット名(UTF16)が使用されます。また、文字セット名AL16UTF16もサポートされています。ただし、リトル・エンディアンのバイト順序のデータ・ファイルにAL16UTF16を指定すると、SQL*Loaderによって警告メッセージが発行され、データ・ファイルはリトル・エンディアンとして処理されます。

CHARACTERSETパラメータは、LOBFILEおよびSDFのみでなくプライマリ・データ・ファイルに対しても指定できます。すべてのプライマリ・データ・ファイルは同じ文字セットとみなされます。INFILEパラメータの前に指定されるCHARACTERSETパラメータは、プライマリ・データ・ファイルのリスト全体に適用されます。CHARACTERSETパラメータがプライマリ・データ・ファイルに対して指定される場合、指定された値は、LOBFILEおよびSDFのデフォルトとしても使用されます。LOBFILEまたはSDF仕様でCHARACTERSETパラメータを指定すると、デフォルト設定は上書きされます。

CHARACTERSETパラメータで指定された文字セットは、制御ファイルのINFILE句で指定されたデータには適用されません。制御ファイルは、常に、NLS_LANGパラメータで指定されているセッション文字セットを使用して処理されます。したがって、NLS_LANGパラメータで指定されているセッション文字セット以外の文字セットでデータをロードするには、そのデータを別のデータ・ファイルに置く必要があります。

関連項目:

- バイト順序
- サポートされている文字セット名の詳細は、『Oracle Databaseグローバリゼーション・サポート・ガイド』を参照してください
- 制御ファイル文字セット
- リトル・エンディアンのUTF-16でエンコーディングされたデータを含むデータ・ファイルをロードする例については、「事例11: Unicode文字セットのデータのロード」を参照してください。(事例の使用方法については、「SQL*Loaderの事例」を参照してください。)

親トピック: 入力文字の変換

9.15.5.3 制御ファイル文字セット

SQL*Loader制御ファイル自体の文字セットは、NLS_LANGパラメータで指定されているセッション文字セットであることが前提です。

制御ファイル文字セットがデータ・ファイル文字セットと異なる場合は、次のことに注意してください。SQL*Loader制御ファイルで文字列として指定されたデリミタおよび比較句の値は、比較が行われる前に、制御ファイル文字セットからデータ・ファイル文字セットに変換されます。正しく指定するには、文字列値ではなく16進文字列を指定してください。

16進文字列がUnicodeのUTF-16エンコーディングのデータ・ファイルで使用される場合、ビッグ・エンディアンのシステムとリトル・エンディアンのシステムでバイト順序が異なります。たとえば、ビッグ・エンディアンのシステムでは、UTF-16の「,」(カンマ)はX'002c'です。リトル・エンディアンのシステムではX'2c00'です。SQL*Loaderでは、常に、ビッグ・エンディアン形式で16進文字列を指定する必要があります。必要に応じて、比較が行われる前に、SQL*Loaderによってバイトが交換されます。これによって、ビッグ・エンディアンおよびリトル・エンディアンの両システム上の制御ファイルで同じ構文を使用することができます。

UnicodeのUTF-16エンコーディングのストリーム形式のデータ・ファイルで使用されるレコード終了記号のデフォルトは、UTF-16では「¥n」(ビッグ・エンディアンのシステムでは0x000Aで、リトル・エンディアンのシステムでは0x0A00)です。INFILE行上の"STR 'char_str'"指定または "STR x'hex_str'"指定を使用して、これらのデフォルト設定を上書きできます。たとえば、次のいずれかの行を使用して、'¥n'のかわりに'ab'をレコード終了記号として使用できます。

```
INFILE myfile.dat "STR 'ab'"
INFILE myfile.dat "STR x'00410042'"
```

BEGINDATA文の後に指定するデータも、NLS_LANGパラメータで指定されたセッション文字セットであることが前提です。

SQL*Loaderのデータ型(CHAR、VARCHAR、VARCHARC、DATEおよび数値型EXTERNAL)に関して、SQL*Loaderでは、バイト単位(バイト長セマンティクス)または文字単位(文字長セマンティクス)のいずれかで指定される文字フィールドの長さがサポートされます。たとえば、制御ファイルでのCHAR(10)指定は、10バイトまたは10文字を意味します。データ・ファイルでシングルバイト文字セットが使用されている場合、これらは同じです。ただし、データ・ファイルでマルチバイト文字セットが使用されている場合、異なることもあります。

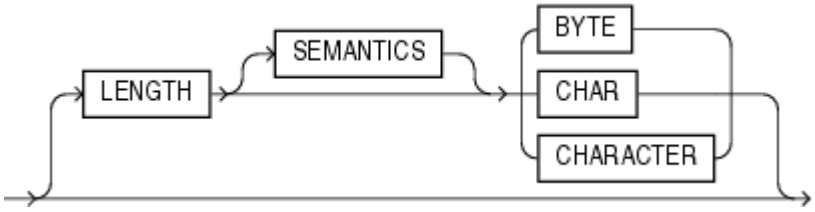
文字セット変換中に文字列の拡張によって発生する挿入エラーを回避するには、データ・ファイルおよびターゲット・データベースの列の両方で文字長セマンティクスを使用します。

親トピック: 入力文字の変換

9.15.5.4 文字長セマンティクス

バイト長セマンティクスは、UTF16文字セット(デフォルトで文字長セマンティクスを使用します)を使用するデータ・ファイル以外のすべてのデータ・ファイルのデフォルトです。

デフォルトに上書きするには、次の構文で示すとおり、CHARまたはCHARACTERを指定します。



LENGTHパラメータは、SQL*Loader制御ファイルのCHARACTERSETパラメータの後に置かれます。LENGTHパラメータは、プライマリ・データ・ファイルに関する構文の指定のみではなく、LOBFILEデータ・ファイルおよびセカンダリ・データ・ファイル(SDF)にも適用されます。INFILEパラメータの前のLENGTH指定は、プライマリ・データ・ファイルのリスト全体に適用されます。プライマリ・データ・ファイルに対するLENGTH指定は、LOBFILEおよびSDFに対するデフォルトとして使用されます。LOBFILEおよびSDF仕様でLENGTHを指定することによって、デフォルトを上書きできます。CHARACTERSETパラメータとは異なり、LENGTHパラメータは、制御ファイル内に含まれるデータ(INFILE *構文)に適用されます。

LENGTHパラメータに対しては、CHARのかわりにCHARACTERを指定できます。

文字長セマンティクスがSQL*Loaderのデータ・ファイルに対して使用されている場合は、次のSQL*Loaderデータ型で文字長セマンティクスが使用されます。

- CHAR
- VARCHAR
- VARCHARC
- DATE
- 数値型EXTERNAL(INTEGER、FLOAT、DECIMALおよびZONED)

VARCHARデータ型では、長さサブフィールドはバイナリのSMALLINT長さサブフィールドです。ただし、その値は文字単位の文字

列の長さを示します。

次のデータ型では、データ・ファイルで文字長セマンティクスが使用されていてもバイト長セマンティクスを使用します。これは、データがバイナリであるか、またはZONEDおよびDECIMALの場合は、特別にバイナリにエンコーディングされた形式であるためです。

- INTEGER
- SMALLINT
- FLOAT
- DOUBLE
- BYTEINT
- ZONED
- DECIMAL
- RAW
- VARRAW
- VARRAWC
- GRAPHIC
- GRAPHIC EXTERNAL
- VARGRAPHIC

POSITIONパラメータの開始引数および終了引数は、文字長セマンティクスがデータ・ファイルで使用されていても、バイト単位で解釈されます。これは、異なるデータ型のデータが混在するデータ・ファイル(一部では文字長セマンティクスを使用し、一部ではバイト長セマンティクスを使用する)を処理する場合に必要です。SMALLINT長さフィールドおよび文字データを含むVARCHARデータ型で位置を処理する場合にも必要です。SMALLINT長さフィールドは、システムによって一定のバイト数(通常、2バイト)を必要とします。ただし、その値は、文字列の長さを文字単位で示します。

データ・ファイルの文字長セマンティクスは、データベースの列で文字長セマンティクスが使用されているかどうかに関係なく使用できます。したがって、データ・ファイルおよびデータベースの列で、同じまたは異なる長さのセマンティクスを使用できます。

親トピック: 入力文字の変換

9.15.6 シフト・センシティブ文字データ

通常、シフト・センシティブ文字データのロードは、ASCIIデータまたはEBCDICデータのロードに比べて非常に遅くなります。

シフト・センシティブ文字データを最速でロードするには、デリミタのない固定位置フィールドを使用します。パフォーマンスを向上させるには、次の点に注意してください。

- フィールド・データに、同数のシフトアウト/シフトイン・バイトが含まれる必要があります。
- フィールドは、シングルバイト・モードで開始および終了する必要があります。
- 最初のバイトをシフトアウト、最後のバイトをシフトインにできます。
- 最初と最後の文字は、マルチバイトにできません。
- 空白が保存され、マルチバイトの空白を確認する必要がある場合、より遅いパスが使用されます。これは、シングルバイトの空白の削除が実行された後、シフトイン・バイトがフィールドの最後のバイトとなる場合に、発生する可能性があります。

親トピック: 異なる文字コード体系の処理

9.16 中断されたロード

ロードは様々な理由で中断されます。

主な理由は領域エラーで、SQL*Loaderで使用されるデータ行および索引エントリ用の領域の不足によって発生します。エラーの最大数を超えた場合、サーバーからSQL*Loaderに予期しないエラーが返された場合、レコードがデータ・ファイルに対して長すぎた場合、または[Ctrl]キーを押しながら[C]キーを押した場合にも、ロードが中断されます。

ロードが中断された場合のSQL*Loaderの動作は、ロードのモードが従来型パス・ロードかダイレクト・パス・ロードか、およびロードが中断された理由によって異なります。また、中断されたロードを継続する場合、SKIPパラメータを使用するかどうか、およびこのパラメータに指定する値は、状況によって異なります。次の項に例を示します。

- 中断された従来型パス・ロード
  従来型パス・ロードでは、バインド配列のすべてのデータがすべての表にロードされた後で、データがコミットされます。
- 中断されたダイレクト・パス・ロード
  ダイレクト・パス・ロードでは、中断されたロードの動作は、ロードが中断された理由によって異なります。
- ロードの中断後の表および索引の状態
  ロードが中断されると、すでにロードされたデータはそのまま表に残り、その表は有効な状態のままとなります。
- ログ・ファイルを使用したロード状態の確認
  SQL*Loaderのログ・ファイルには、表や索引の状態、および入力データ・ファイルから読み込まれた論理レコード数の情報が記録されます。
- 単一表へのロードの継続
  ダイレクト・パス・ロードまたは従来型パス・ロードを、終了前にSQL*Loaderで中断する必要がある場合は、すでにコミットされているか、またはセーブポイントが付けられている行が存在しています。

関連項目:

SKIP

親トピック: SQL*Loader制御ファイル・リファレンス

9.16.1 中断された従来型パス・ロード

従来型パス・ロードでは、バインド配列のすべてのデータがすべての表にロードされた後で、データがコミットされます。

ロードが中断された場合は、直前のコミット操作の時点までに処理された行のみがロードされます。データの部分コミットは行われません。

親トピック: 中断されたロード

9.16.2 中断されたダイレクト・パス・ロード

ダイレクト・パス・ロードでは、中断されたロードの動作は、ロードが中断された理由によって異なります。

次の各項では、ロードが中断された理由について説明します。

- 領域エラーによって中断されたロード
  領域エラーが原因でロードが中断されたときのSQL*Loaderの動作は、データを複数のサブパーティションにロード中だったかどうかで異なります。
- エラーが最大数を超えたことによって中断されたロード

エラーが最大数を超えると、SQL*Loaderによって、すべての表へのレコードのロードが停止され、その時点までに終了している処理がコミットされます。

- 致命的エラーによって中断されたロード
  致命的エラーが発生した場合は、ロードが停止されます。ロードの開始時にROWSが指定されていないかぎり、データは保存されません。
- [Ctrl]キーを押しながら[C]キーを押すことによって中断されたロード
  SQL*Loaderによるデータの保存中に[Ctrl]キーを押しながら[C]キーを押すと、データの保存は継続され、保存の完了後にロードが停止されます。

親トピック: 中断されたロード

9.16.2.1 領域エラーによって中断されたロード

領域エラーが原因でロードが中断されたときのSQL*Loaderの動作は、データを複数のサブパーティションにロード中だったかどうかで異なります。

- データを複数のサブパーティションにロードしているとき(パーティション表、コンポジット・パーティション表、またはコンポジット・パーティション表の1つのパーティションにロードしているとき)に領域エラーが発生した場合

  複数のサブパーティションへのロード中に領域エラーが発生した場合、ロードは中断され、ROWSが指定されていなければデータは保存されません(この場合、コミット済のデータはすべて保存されます)。このような動作は、行が順不同にロードされる可能性を考慮したためです。順不同になるのは、それぞれの行が順序を考慮せずにパーティションに割り当てられ、そのパーティションが別々にロードされることが原因です。パーティションに割り当てられているすべての行がロードされる前にロードが中断されると、レコード番号nの行はロードされていても、レコード番号n-1の行がロードされていない可能性があります。したがって、単純にSKIP=Nを使用するだけではロードを続行できません。

- 非パーティション表、パーティション表の1つのパーティション、またはコンポジット・パーティション表の1つのサブパーティションにデータをロードしているときに、領域エラーが発生した場合

  制御ファイルに記述されているINTO TABLE文が1つの場合、SQL*Loaderは、エラーが発生する前にロードされていた行と同じ数の行をコミットします。

  制御ファイルにINTO TABLE文が複数ある場合、SQL*Loaderは、データ・ファイルから他の表に読取り済のデータをロードし、そのデータをコミットします。

  いずれの場合も、この動作は、ROWSパラメータが指定されていたかどうかに関係なく実行されます。ロードを継続する場合は、SKIPパラメータを使用して、ロード済の行をスキップできます。INTO TABLE文が複数あった場合は、それぞれの表に異なる数の行がロードされている可能性があります。したがって、ロードを継続するには、表ごとに異なるSKIPパラメータの値を指定することが必要な場合があります。SKIPパラメータの値がすべての表で同じ場合にのみ、SQL*Loaderによってその値が報告されます。

親トピック: 中断されたダイレクト・パス・ロード

9.16.2.2 エラーが最大数を超えたことによって中断されたロード

エラーが最大数を超えると、SQL*Loaderによって、すべての表へのレコードのロードが停止され、その時点までに終了している処理がコミットされます。

ロードを継続する場合、SKIPパラメータに指定する値は表によって異なります。SKIPパラメータの値がすべての表で同じ場合にのみ、SQL*Loaderによってその値が報告されます。

親トピック: 中断されたダイレクト・パス・ロード

9.16.2.3 致命的エラーによって中断されたロード

致命的エラーが発生した場合は、ロードが停止されます。ロードの開始時にROWSが指定されていないかぎり、データは保存されません。

指定されている場合、コミット済のすべてのデータが保存されます。SKIPパラメータの値がすべての表で同じ場合にのみ、SQL*Loaderによってその値が報告されます。

親トピック: 中断されたダイレクト・パス・ロード

9.16.2.4 [Ctrl]キーを押しながら[C]キーを押すことによって中断されたロード

SQL*Loaderによるデータの保存中に[Ctrl]キーを押しながら[C]キーを押すと、データの保存は継続され、保存の完了後にロードが停止されます。

データを保存中でない場合は、コミットされていない処理をコミットせずにロードが停止されます。つまり、SKIPパラメータの値は、すべての表で同じになります。

親トピック: 中断されたダイレクト・パス・ロード

9.16.3 ロードの中断後の表および索引の状態

ロードが中断されると、すでにロードされたデータはそのまま表に残り、その表は有効な状態のままとなります。

従来型パスを使用した場合、すべての索引は有効な状態のままとなります。

ダイレクト・パス・ロード方法を使用した場合は、表の索引はすべて使用禁止状態のままになります。このような索引は、継続処理をする前、またはロードが再開されすべて終了した後に、再構築または再作成できます。

これ以外の索引は、他にエラーが発生していないかぎり有効です。索引が使用禁止状態のままになるその他の理由については、「使用禁止状態(Index Unusable)のままの索引」を参照してください。

親トピック: 中断されたロード

9.16.4 ログ・ファイルを使用したロード状態の確認

SQL*Loaderのログ・ファイルには、表や索引の状態、および入力データ・ファイルから読み込まれた論理レコード数の情報が記録されます。

これらの情報は、中断されたロードを再開する場合に利用できます。

親トピック: 中断されたロード

9.16.5 単一表へのロードの継続

ダイレクト・パス・ロードまたは従来型パス・ロードを、終了前にSQL*Loaderで中断する必要がある場合は、すでにコミットされているか、またはセーブポイントが付けられている行が存在しています。

中断されたロードを継続するには、SKIPパラメータを使用して、前回のロードによってすでに処理されている論理レコードの数を指定します。ロードの中断時に、次のようなメッセージ形式でログ・ファイルにSKIPの値が書き込まれます。

```
Specify SKIP=1001 when continuing the load.
```

SKIPパラメータの値を指定するこのメッセージの前には、ロードが中断された理由を示すメッセージが書き込まれています。

複数表へのロードの場合は、SKIPパラメータの値がすべての表で同じ場合にのみ、値が表示されます。

関連項目:

SKIP

親トピック: 中断されたロード

9.17 物理レコードからの論理レコードの作成

この項では、物理レコードから論理レコードを作成する方法について説明します。

複数の物理レコードを1つの論理レコードに結合するには、対象とするデータによって、次の句のうちの1つを指定します。

- CONCATENATE
- CONTINUEIF

- CONCATENATEを使用した論理レコードの作成
  常に一定の数の物理レコードを結合して1つの論理レコードを構成する場合、CONCATENATEを使用します。
- CONTINUEIFを使用した論理レコードの作成
  CONTINUEIFは、結合する物理レコードの数が異なる場合に使用します。

親トピック: SQL*Loader制御ファイル・リファレンス

9.17.1 CONCATENATEを使用した論理レコードの作成

常に一定の数の物理レコードを結合して1つの論理レコードを構成する場合、CONCATENATEを使用します。

次の例では、integerによって結合する物理レコードの数を指定します。

```
CONCATENATE  integer
```

CONCATENATEに指定されたinteger値によって、SQL*Loaderで列配列の各行に割り当てられる物理レコード構造の数が決まります。ダイレクト・パス・ロードでは、COLUMNARRAYROWSのデフォルト値が大きいため、CONCATENATEにも大きい値を指定すると、メモリーの過剰割当てが発生する可能性があります。この場合は、COLUMNARRAYROWSパラメータの値を小さくして列配列の行の数を減らすことによって、パフォーマンスを向上できます。

関連項目:

- COLUMNARRAYROWS
- 列配列の行数およびストリーム・バッファ・サイズの指定

親トピック: 物理レコードからの論理レコードの作成

9.17.2 CONTINUEIFを使用した論理レコードの作成

CONTINUEIFは、結合する物理レコードの数が異なる場合に使用します。

CONTINUEIF句を使用する場合は、後に条件を指定します。この条件は、各物理レコードが読み込まれるたびに評価されます。たとえば、2つのレコードがあって、その最初のレコードの80バイト目にポンド記号(#)がある場合、2つのレコードは結合されます。この場合、指定の位置に他の文字があると、2番目のレコードは最初のレコードに結合されません。

次に、CONTINUEIFの完全な構文を示します。この構文で、さらに柔軟な処理を実行できます。

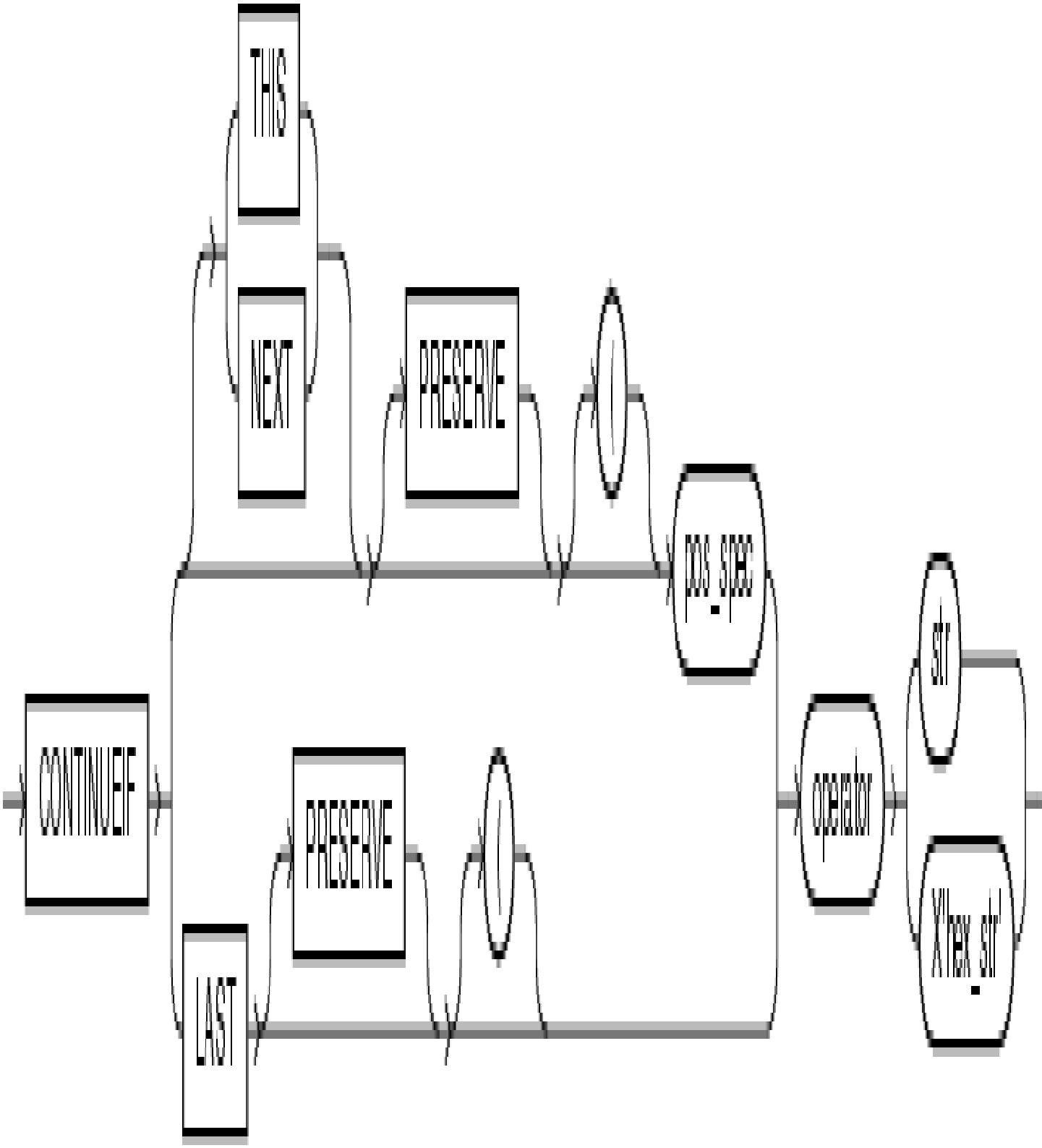


表9-2では、CONTINUEIF句のパラメータについて説明します。

表9-2 CONTINUEIF句のパラメータ

パラメータ	説明
THIS	現在のレコードについて条件が真のとき、次の物理レコードを読み込んで現在のレコードに連結します。この処理は、条件が偽になるまで繰り返されます。条件が偽になると、現在の物理レコードが現在の論理レコードを構成する最後の物理レコードになります。THIS はデ

パラメータ	説明
	フォルトです。
NEXT	次のレコードで条件が真になると、現在の物理レコードを現在の論理レコードに連結します。この処理は、条件が偽になるまで繰り返されます。
operator	サポートされているのは等号演算子(=)と不等号演算子(!= または<>)です。 等号演算子を指定すると、フィールドと比較文字列が完全に一致したときに、条件が真となります。不等号演算子を指定した場合は、どの文字が異なっていても、条件は真となります。
LAST	レコードの最後の文字(空白以外)に対して、THIS と同様の条件判断を行います。現在の物理レコードの最後の文字(空白以外)が条件を満たしていると、次の物理レコードが読み込まれ、現在の物理レコードに連結されます。この処理は、条件が偽になるまで繰り返されます。現在のレコードに関して条件が偽になると、現在の物理レコードが現在の論理レコードを構成する最後の物理レコードになります。 LAST には、単一の文字継続フィールドのみを使用できます(これに対し、THIS および NEXT には、複数の文字継続フィールドを使用できます)。
pos_spec	物理レコード中の列の開始番号と終了番号です。 列番号は 1 から始まります。ハイフンまたはコロンを使用することもできます(start-end または start:end)。 end を省略すると、継続フィールドの長さには、指定されたバイト列または文字列の長さが取られます。end が指定されていて、それによって決まる継続フィールドの長さが指定されたバイト列または文字列の長さと異なる場合は、短い方に不足分を埋めるための文字が追加されます。文字列の場合は空白が追加され、16 進数のバイト列の場合は 0(ゼロ)が追加されます。
str	start と end で定義された継続フィールドと比較する文字列です。比較演算の内容は、operator で指定された演算子によって異なります。文字列は一重引用符または二重引用符で囲みます。文字列は 1 文字ずつ比較され、必要があれば、空白埋め文字が右側に追加されます。
X'hex-str'	16 進形式で指定された文字列で、使用方法は str と同じです。たとえば、X'1FB033' と表記した場合は、1F、B0、33 の値を持つ 3 バイトの 16 進文字列を意味します。
PRESERVE	論理レコードには'char_string'または X'hex_string'が含まれます。デフォルトで

パラメータ	説明
	は、これらは排除されます。

CONTINUEIF句の中で指定する位置は、各物理レコード中の位置を示します。物理レコード中の位置が参照されるのは、この場合のみです。これ以外の場合、参照先は論理レコードとなります。

CONTINUEIF THISおよびCONTINUEIF LASTでは、PRESERVEパラメータを指定しない場合、論理レコードの作成時にすべての物理レコードから継続フィールドが削除されます。継続フィールドが読み込まれるまでは、データは同じ論理レコードに連結します。たとえば、CONTINUEIF THIS(3:5)='***'を指定すると、3から5までの位置がすべてのレコードから削除されます。つまり、レコードの3から5までの位置にある継続文字が削除されます。また、3から5までの位置にある文字は、継続文字が3から5までの位置にない場合でもレコードから削除されます。

CONTINUEIF THISおよびCONTINUEIF LASTでは、PRESERVEパラメータを使用した場合、論理レコードの作成時にすべての物理レコードに継続フィールドが保持されます。

CONTINUEIF LASTは、CONTINUEIF THISおよびCONTINUEIF NEXTとは異なります。CONTINUEIF LASTでは、継続フィールドの位置がレコードごとに異なり、PRESERVEが指定されていない場合でも、継続フィールドは削除されません。

例9-3から例9-6に、PRESERVEパラメータの有無にかかわらず、CONTINUEIF THISおよびCONTINUEIF NEXTの使用例を示します。

例9-3 PRESERVEパラメータを使用しないCONTINUEIF THIS

ここで、物理レコードは次のような14バイトのレコードとします。また、ピリオドは空白を表します。

```
%%aaaaaaaa....
%%bbbbbbbb....
..cccccccc....
%%dddddddddd..
%%eeeeeeeeee..
..ffffffffff..
```

次の例では、CONTINUEIF THIS句にPRESERVEパラメータは使用されていません。

```
CONTINUEIF THIS (1:2) = '%%'
```

したがって、論理レコードは次のように作成されます。

```
aaaaaaaa....bbbbbbbb....cccccccc....
dddddddddd..eeeeeeeeee..ffffffffff..
```

列1および列2(たとえば、物理レコード1の%%)は、論理レコード作成時に物理レコードから削除されることに注意してください。

例9-4 PRESERVEパラメータを使用するCONTINUEIF THIS

例9-3の物理レコードと同じ物理レコードがあるとします。

次の例では、CONTINUEIF THIS句にPRESERVEパラメータが使用されています。

```
CONTINUEIF THIS PRESERVE (1:2) = '%%'
```

したがって、論理レコードは次のように作成されます。

```
%%aaaaaaaa....%%bbbbbbbb......cccccccc....
%%dddddddddd..%%eeeeeeeeee....ffffffffff..
```

列1および列2は、論理レコード作成時に物理レコードから削除されないことに注意してください。

例9-5 PRESERVEパラメータを使用しないCONTINUEIF NEXT

ここで、物理レコードは次のような14バイトのレコードとします。また、ピリオドは空白を表します。

```
..aaaaaaaa....
%%bbbbbbbb....
%%cccccccc....
..dddddddddd..
%%eeeeeeeeee..
%%ffffffffff..
```

次の例では、CONTINUEIF NEXT句にPRESERVEパラメータは使用されていません。

```
CONTINUEIF NEXT (1:2) = '%%'
```

したがって、論理レコードは次のように作成されます(例9-3と同じ結果)。

```
aaaaaaaa....bbbbbbbb....cccccccc....
dddddddddd..eeeeeeeeee..ffffffffff..
```

例9-6 PRESERVEパラメータを使用するCONTINUEIF NEXT

例9-5の物理レコードと同じ物理レコードがあるとします。

次の例では、CONTINUEIF NEXT句にPRESERVEパラメータが使用されています。

```
CONTINUEIF NEXT PRESERVE (1:2) = '%%'
```

したがって、論理レコードは次のように作成されます。

```
..aaaaaaaa....%%bbbbbbbb....%%cccccccc....
..dddddddddd..%%eeeeeeeeee..%%ffffffffff..
```

関連項目:

CONTINUEIF句の例は、「事例4: 結合された物理レコードのロード」を参照してください。(事例の使用方法については、「SQL*Loaderの事例」を参照してください。)

親トピック: 物理レコードからの論理レコードの作成

9.18 表への論理レコードのロード

この項では、論理レコードを表にロードする方法について説明します。

この項では、次の内容について説明します。

- ロードする表の指定方法
- 表にロードするレコードの指定方法
- レコードに対するデフォルトのデータ・デリミタ
- データが欠落しているショート・レコードの処理方法

- 表名の指定
  LOAD DATA文のINTO TABLE句を使用して、表、フィールドおよびデータ型を識別できます。
- 表固有のロード方法
  表のロード時に、INTO TABLE句を使用して、表固有のロード方法(INSERT、APPEND、REPLACEまたは

TRUNCATE)を指定できます。指定したロード方法は、その表のみに適用されます。

- 表固有のOPTIONSパラメータ
  パラレル・ロードでは、個々の表に対してOPTIONSパラメータを指定できます。(このパラメータは、パラレル・ロードの場合のみ有効です)。
- 条件に基づいたレコードのロード
  WHEN句を使用して論理レコード中の条件をテストし、その論理レコードをロードするか廃棄するかを選択できます。
- デフォルトのデータ・デリミタ(区切り記号)の指定
  データ・ファイル中のデータ・フィールドがすべて同じ文字で終了している場合、FIELDS句を使用してデフォルトの終了デリミタおよび囲みデリミタを指定できます。
- データが欠落しているショート・レコードの処理
  制御ファイルの定義で指定したフィールドの数が、実際にレコード中に存在するフィールドより多い場合、残りの(指定された余分の)列にNULL値を設定するか、またはエラーを出力するかがSQL*Loaderによって判断されます。

親トピック: SQL*Loader制御ファイル・リファレンス

9.18.1 表名の指定

LOAD DATA文のINTO TABLE句を使用して、表、フィールドおよびデータ型を識別できます。

この句で、データ・ファイル中のレコードとデータベース中の表の関係を定義します。フィールドやデータ型の指定については後述します。

- INTO TABLE句
  INTO TABLE句が持つ様々な機能の1つに、データのロード先となる表を指定する機能があります。

親トピック: 表への論理レコードのロード

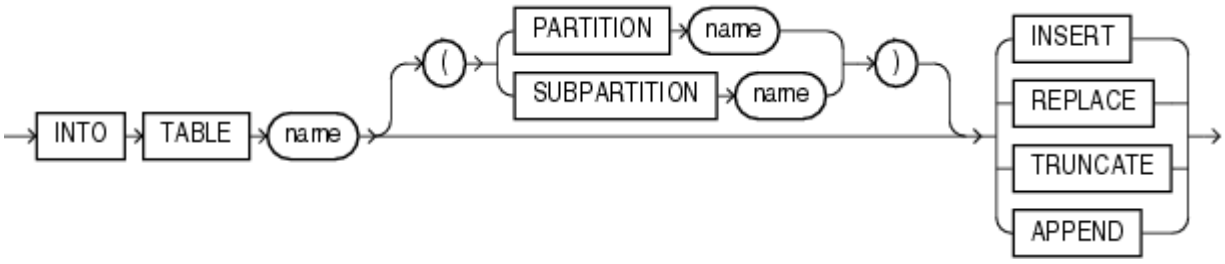
9.18.1.1 INTO TABLE句

INTO TABLE句が持つ様々な機能の1つに、データのロード先となる表を指定する機能があります。

複数の表にロードするには、それぞれの表に対してINTO TABLE句を指定する必要があります。

INTO TABLE句を記述する場合、キーワードINTO TABLEの次に、データを受け取るOracleの表名を指定します。

構文は次のとおりです。



表名には、すでに存在する表を指定してください。表名がSQLやSQL*Loaderの予約済キーワードと同じ場合、表名に特殊文字が含まれる場合、または表名の中の大/小文字を区別する必要がある場合は、表名を二重引用符で囲みます。

```
INTO TABLE scott."CONSTANT"
INTO TABLE scott."Constant"
INTO TABLE scott."-CONSTANT"
```

ユーザーが表をロードするには、INSERT権限が必要です。表がユーザーのスキーマにない場合、ユーザーはシノニムを使用して表を参照するか、またはスキーマ名を表名の一部として含める必要があります(たとえば、scott.empはscottスキーマの表empを参照します)。

ノート:

SQL*Loader では、データベースへの接続処理を完了した後の現行のスキーマが、デフォルト・スキーマであるとみなされます。したがって、データベースへの接続中に実行されるログオン・トリガーが存在する場合は、接続文字列で指定されたスキーマがデフォルト・スキーマになるとはかぎりません。

特定のデータベースに接続した時点で現行のスキーマを別のスキーマに変更するログオン・トリガーが存在する場合は、その新しいスキーマがデフォルトとして使用されます。

親トピック: 表名の指定

9.18.2 表固有のロード方法

表のロード時に、INTO TABLE句を使用して、表固有のロード方法(INSERT、APPEND、REPLACEまたはTRUNCATE)を指定できます。指定したロード方法は、その表のみに適用されます。

表固有のロード方法は、グローバルな表のロード方法よりも優先されます。INTO TABLE句の前にロード方法が特に指定されていない場合、グローバルな表のロード方法は、デフォルトでINSERTとなります。次の項では、これらのオプションを使用して空および空でない表へデータをロードする方法について説明します。

- INSERTを使用した空の表へのデータのロード
  空の表にデータをロードするには、INSERTオプションを使用します。
- 空でない表へのデータのロード
  この項では、データを空でない表にロードする方法について説明します。

親トピック: 表への論理レコードのロード

9.18.2.1 INSERTを使用した空の表へのデータのロード

空の表にデータをロードするには、INSERTオプションを使用します。

ロード先の表が空の場合は、INSERTオプションを使用します。INSERTオプションは、SQL*Loaderのデフォルトのメソッドです。INSERTを使用するには、データのロード先の表をロードする前に空にしておく必要があります。データのロード先の表に行が含まれていると、SQL*Loaderはエラーで終了します。「事例1: 可変長データのロード」に例があります。(事例の使用方法については、「SQL*Loaderの事例」を参照してください。)

SQL*Loaderは、データの挿入先となる表をチェックし、表が空であることを確認します。このため、INSERTを実行するユーザーには、SELECT権限とINSERT権限の両方を付与する必要があります。

関連トピック

- SQL*Loaderの事例

親トピック: 表固有のロード方法

9.18.2.2 空でない表へのデータのロード

この項では、空でない表へのデータのロードについて説明します。

ロード先の表にすでにデータが存在する場合は、次の3つのオプションがあります。

- APPEND

- REPLACE
- TRUNCATE

> ノート:
>
> REPLACE または TRUNCATE を指定すると、個々の行ではなく表全体が置き換えられます。行の削除が成功した後、COMMIT が実行されます。ロード前に表にあったデータは、事前にエクスポートまたはそれに相当するユーティリティで保存しておかないかぎり、リカバリできません。

- APPEND
  SQL*Loaderで表に行を追加するには、INTO TABLEのAPPEND句を使用します。
- REPLACE
- 既存の行の更新
- TRUNCATE

親トピック: 表固有のロード方法

9.18.2.2.1 APPEND

SQL*Loaderで表に行を追加するには、INTO TABLEのAPPEND句を使用します。

表にデータがすでに存在する場合、SQL*Loaderで新しい行が表に追加されます。データが存在しない場合には、単に新しい行がロードされます。APPENDオプションを使用するには、SELECT権限が必要です。「事例3: 自由区分形式ファイルのロード」に例があります。(事例の使用方法については、「SQL*Loaderの事例」を参照。)

関連トピック

- SQL*Loaderの事例

親トピック: 空でない表へのデータのロード

9.18.2.2.2 REPLACE

REPLACEオプションを使用すると、SQLのDELETE FROM TABLE文が実行されます。表の既存の行はすべて削除され、新しくデータがロードされます。この場合、その表がロード実行者のスキーマ内に存在するか、ロード実行者がその表に対してDELETE権限を持っている必要があります。「事例4: 結合された物理レコードのロード」に例があります。(事例の使用方法については、「SQL*Loaderの事例」を参照してください。)

行削除によって、その表に定義された削除トリガーが起動します。DELETE CASCADEが表に指定されている場合、カスケード化された削除が同様に実行されます。カスケード化された削除の詳細は、『Oracle Database概要』を参照してください。

親トピック: 空でない表へのデータのロード

9.18.2.2.3 既存の行の更新

REPLACEは、個々の行ではなく表を置換する方法です。既存レコードにNULL列があっても、そのレコードは更新されません。既存の行を更新するには、次の手順を実行してください。

1. データを作業表にロードします。
2. 相関副問合せを指定した、SQLのUPDATE文を使用します。
3. 作業表を削除します。

親トピック: 空でない表へのデータのロード

9.18.2.2.4 TRUNCATE

TRUNCATEオプションを使用すると、SQLのTRUNCATE TABLE table_name REUSE STORAGE文が実行され、表のエクステントが再利用されます。TRUNCATEオプションにより、表またはクラスタからすべての行が短時間で効率的に削除されるため、最大限の処理パフォーマンスを実現できます。TRUNCATE文を実行する前に、表の参照整合性制約を使用禁止にします。参照整合性制約が使用禁止でない場合、SQL*Loaderによってエラーが返されます。

整合性制約が使用禁止になると、その表に対してはDELETE CASCADEは定義されません。DELETE CASCADE機能が必要な場合は、ロードを開始する前に、表の内容を手動で削除する必要があります。

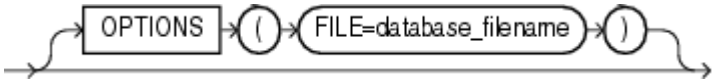
この場合、表がロード実行者のスキーマにあるか、ロード実行者がDROP ANY TABLE権限を所有している必要があります。

親トピック: 空でない表へのデータのロード

9.18.3 表固有のOPTIONSパラメータ

パラレル・ロードでは、個々の表に対してOPTIONSパラメータを指定できます(このパラメータは、パラレル・ロードの場合のみ有効です)。

OPTIONSパラメータの構文は次のとおりです。



関連項目:

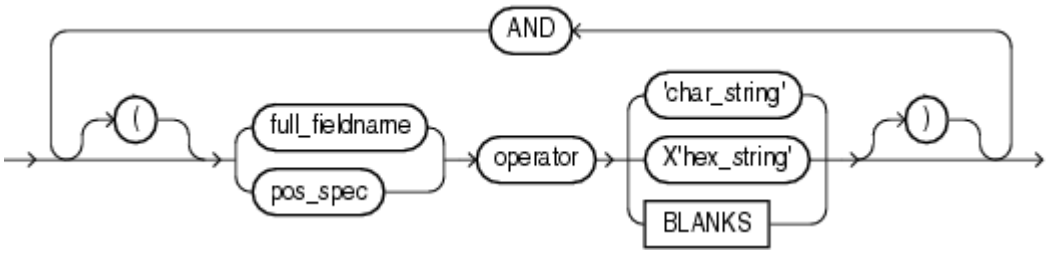
パラレル・ダイレクト・パス・ロードのパラメータ

親トピック: 表への論理レコードのロード

9.18.4 条件に基づいたレコードのロード

WHEN句を使用して論理レコード中の条件をテストし、その論理レコードをロードするか廃棄するかを選択できます。

WHEN句を表名の後に記述し、その後にフィールド条件を1つ以上指定します。field_conditionの構文は、次のとおりです。



たとえば、次のように指定すると、第5列の値がqであるレコードがすべてロードされます。

```
WHEN (5) = 'q'
```

WHEN句では各条件の前にANDを使用して、複数の条件を設定できます。小カッコの指定は任意ですが、ANDによって複数の条件を設定している場合は、あいまいさを避けるために必ず使用してください。たとえば:

```
WHEN (deptno = '10') AND (job = 'SALES')
```

- LOBFILEおよびSDFでのWHEN句の使用
  LOBFILEおよびSDFでWHEN句を使用する方法を参照してください。

関連項目:

- SQL*LoaderによるNULLIFEおよびDEFAULTIF句に対するWHEN句の評価方法の詳細は、「WHEN、NULLIFおよびDEFAULTIF句の使用」を参照してください
- WHEN句の使用例は、「事例5: 複数表へのデータのロード」(事例の使用方法については、「SQL*Loaderの事例」)を参照してください

親トピック: 表への論理レコードのロード

9.18.4.1 LOBFILEおよびSDFでのWHEN句の使用

LOBFILEおよびSDFでWHEN句を使用する方法を参照してください。

LOBFILEまたはSDFのレコードが廃棄されても、SQL*LoaderではそのLOBFILEまたはSDF内の対応するデータがスキップされません。

親トピック: 条件に基づいたレコードのロード

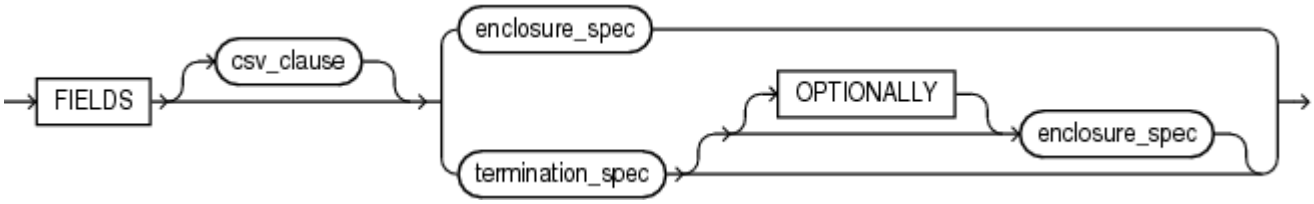
9.18.5 デフォルトのデータ・デリミタ(区切り記号)の指定

データ・ファイル中のデータ・フィールドがすべて同じ文字で終了している場合、FIELDS句を使用してデフォルトの終了デリミタおよび囲みデリミタを指定することができます。

- fields_spec
- termination_spec
  終了指定の構文について説明します。
- enclosure_spec

親トピック: 表への論理レコードのロード

9.18.5.1 fields_spec



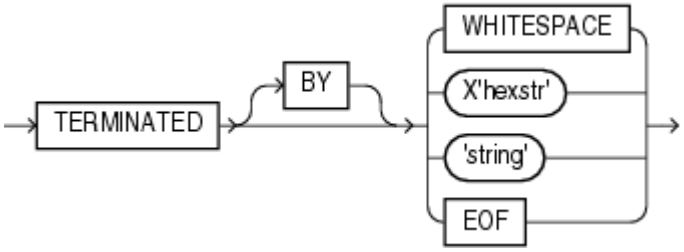
関連項目:

- csv_clauseの詳細は、「CSV形式のファイルの指定」を参照してください

親トピック: デフォルトのデータ・デリミタ(区切り記号)の指定

9.18.5.2 termination_spec

終了指定の構文について説明します。

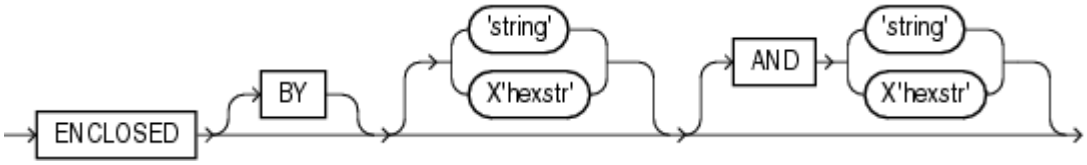


ノート:

終了記号の文字列には、1 つ以上の文字が含まれます。また、TERMINATED BY EOF は、LOBFILE からの LOB のロードにのみ適用されます。

親トピック: デフォルトのデータ・デリミタ(区切り記号)の指定

9.18.5.3 enclosure_spec



ノート:

囲み文字列には、1 つ以上の文字が含まれます。

特定の列に対してデリミタを上書きする場合は、その列名の後に適用するデリミタを指定します。「事例3: 自由区分形式ファイルのロード」に例があります。(事例の使用方法については、「SQL*Loaderの事例」を参照してください。)

関連項目:

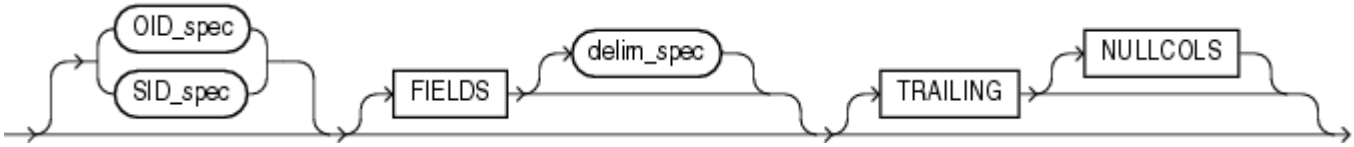
- 構文の詳細は、「デリミタの指定」を参照してください。
- LOBFILEからのLOBデータのロード

親トピック: デフォルトのデータ・デリミタ(区切り記号)の指定

9.18.6 データが欠落しているショート・レコードの処理

制御ファイルの定義で指定したフィールドの数が、実際にレコード中に存在するフィールドより多い場合、残りの(指定された余分の)列にNULL値を設定するか、またはエラーを出力するかがSQL*Loaderによって判断されます。

制御ファイルの定義でフィールドの開始位置として論理レコードの終了位置よりも後の位置が明示的に指定されている場合、SQL*Loaderによって、このフィールドにNULL値が設定されます。フィールドが(次に示す例の中のdnameおよびlocのように)相対位置に定義されていて、そのフィールドが現れる前にレコードのデータが終わった場合は、SQL*Loaderによってこのフィールドに NULL値が設定されるか、またはエラーが出力されます。SQL*Loaderによる処理は、TRAILING NULLCOLS句(次の構文図を参照)を指定しているかどうかによって決まります。



- TRAILING NULLCOLS句

親トピック: 表への論理レコードのロード

9.18.6.1 TRAILING NULLCOLS句

TRAILING NULLCOLS句を使用すると、相対位置で指定した列がレコード中に存在しない場合、その列の値はNULLとして処理されます。

たとえば、次のようなデータについて考えます。

```
10 Accounting
```

このデータが次の制御ファイルで読み込まれ、そのレコードはdnameの後で終了するとします。

```
INTO TABLE dept
    TRAILING NULLCOLS
( deptno CHAR TERMINATED BY " ",
  dname  CHAR TERMINATED BY WHITESPACE,
  loc    CHAR TERMINATED BY WHITESPACE
)
```

この場合、その後のlocフィールドにはNULL値が設定されています。この例でTRAILING NULLCOLS句を指定しなかった場合は、データ欠落のためエラーとなります。

関連項目:

TRAILING NULLCOLSの使用例は、「事例7: 書式設定されたレポートからのデータの抽出」(事例の使用方法については、「SQL*Loaderの事例」)を参照してください。

親トピック: データが欠落しているショート・レコードの処理

9.19 索引オプション

この項では、索引オプションについて説明します。

この項では、索引エントリの作成方法を制御する次のSQL*Loaderオプションについて説明します。

- SORTED INDEXES
- SINGLEROW

- SORTED INDEXES句
- SINGLEROWオプション

親トピック: SQL*Loader制御ファイル・リファレンス

9.19.1 SORTED INDEXES句

SORTED INDEXES句はダイレクト・パス・ロードに適用できます。このオプションを指定すると、入力データはロード前に指定の索引でソートされていることがSQL*Loaderで認識されるため、ロード時にはSQL*Loaderのパフォーマンスが最適化されます。

関連項目:

SORTED INDEXES句

親トピック: 索引オプション

9.19.2 SINGLEROWオプション

SINGLEROWオプションは、APPENDオプションを使用してかぎられたメモリーでダイレクト・パス・ロードを実行する場合、または少数のレコードを大規模な表にロードする場合に使用します。このオプションを使用すると、各索引エントリが、一度に1レコードずつ直接索引に挿入されます。

表に行を追加(APPEND)する場合、デフォルトではSINGLEROWは使用されません。この場合、索引エントリは個別の一時記憶域に置かれ、ロード終了時に元の索引とマージされます。この方法では、パフォーマンスが向上して最適な索引が作成されますが、記憶域も余分に必要となります。マージ処理が実行されている間は、元の索引、新しい索引、および新しいエントリのための領域が同時に記憶域を占有します。

SINGLEROWオプションの場合、新しい索引エントリや新しい索引のための記憶域は必要ありません。結果としてできる索引は、新しくソートされたものほど最適化されていない可能性がありますが、作成するための記憶域は少なくて済みます。ただし、各索引が挿入されるたびにUNDO情報が追加で作成されるため、時間がかかります。このオプションは、次の場合に使用してください。

- 使用可能な記憶域がかぎられている場合。
- ロードされる行数が表のサイズに比べて小さい場合(比率が1:20以下かどうかが目安になります)。

親トピック: 索引オプション

9.20 複数のINTO TABLE句を使用するメリット

これらの項では、複数のINTO TABLE句を使用するメリットについて説明します。

複数のINTO TABLE句を指定すると、次の処理が可能になります。

- 異なる表へのデータのロード
- 1つの入力レコードからの複数の論理レコードの抽出
- 異なる入力レコード形式の区別
- 異なる入力行オブジェクト・サブタイプの区別

ここでは、まず、複数のINTO TABLE句で同じ表を参照するという最も一般的な使用方法を示します。この項では、複数のINTO TABLE句の様々な指定方法を説明します。また、POSITIONパラメータの指定方法についても説明します。

> ノート:
>
> INTO TABLE 句を複数指定した場合、次の INTO TABLE 句の処理に移ったときのフィールド・スキャンは、前回フィールド・スキャンを停止した位置から続行されます。この項の残りの部分では、このような動作を利用する重要な方法について詳しく説明します。また、固定フィールド位置や POSITION パラメータを使用した別の手順についても説明します。

- 複数の論理レコードの抽出
  データ・レコードが短い場合、記憶域を効率的に使用するために複数の論理レコードをまとめて1つの物理レコードに記

録できます。

- 異なる入力レコード形式の区別
  1件のデータ・ファイルに様々な形式のレコードが含まれる場合があります。
- 異なる入力行オブジェクトのサブタイプの区別
  1件のデータ・ファイルには、同じ基本行オブジェクト型から継承された行オブジェクトで構成されるレコードが含まれる可能性があります。
- 複数表へのデータのロード
  複数のINTO TABLE句でPOSITIONパラメータを指定すると、1件のレコードのデータを正規化された複数の表にロードできます。
- 複数のINTO TABLE句使用のサマリー
  複数のINTO TABLE句を指定すると、1件の入力レコードから複数の論理レコードを抽出できます。また、同一ファイル中の異なる形式のレコードを区別できます。

親トピック: SQL*Loader制御ファイル・リファレンス

9.20.1 複数の論理レコードの抽出

データ・レコードが比較的短い場合、メディアを効率的に使用するために複数の論理レコードをまとめて1つの物理レコードに記録できます。

データ記憶域および転送メディアには、物理レコードが固定長のものがあります。

ここでは、入力ファイルの1件の物理レコードを2件の論理レコードとみなし、INTO TABLE句を2回指定してemp表にデータをロードする方法について説明します。たとえば、次のようなデータの例を考えます。

```
1119 Smith      1120 Yvonne
1121 Albert     1130 Thomas
```

次の制御ファイルを使用して論理レコードを抽出します。

```
INTO TABLE emp
     (empno POSITION(1:4)  INTEGER EXTERNAL,
      ename POSITION(6:15) CHAR)
INTO TABLE emp
     (empno POSITION(17:20) INTEGER EXTERNAL,
      ename POSITION(21:30) CHAR)
```

- デリミタに基づく相対的な位置指定
  同じレコードを、別の指定方法でロードできます。

親トピック: 複数のINTO TABLE句を使用するメリット

9.20.1.1 デリミタに基づく相対的な位置指定

同じレコードを、別の指定方法でロードできます。

次の制御ファイルでは、絶対的位置を指定するかわりに相対的な位置を指定しています。ここでは、各フィールドが空白(" ")1文字、またはいくつかの空白やタブ(WHITESPACE)で区切られていることを示しています。

```
INTO TABLE emp
     (empno INTEGER EXTERNAL TERMINATED BY " ",
      ename CHAR             TERMINATED BY WHITESPACE)
INTO TABLE emp
     (empno INTEGER EXTERNAL TERMINATED BY " ",
      ename CHAR)            TERMINATED BY WHITESPACE)
```

この例では、2番目のempnoフィールドは、別のINTO TABLE句に指定されていますが、1番目のenameの直後に指定されていることに注意してください。新しいINTO TABLE句に対して、レコードの先頭からのフィールド・スキャンが実行されるわけではありません。かわりに、前回スキャンが停止した位置から続行されます。

レコードのスキャンを特定の位置から強制的に開始するには、POSITIONパラメータを使用します。詳細は、「異なる入力レコード形式の区別」および「複数表へのデータのロード」を参照してください。

親トピック: 複数の論理レコードの抽出

9.20.2 異なる入力レコード形式の区別

通常、データ・ファイルには様々な形式のレコードが含まれています。

ここでは、次のようなデータについて考えます。この例では、emp表およびdept表のレコードが、データ中に混在しています。

```
1 50   Manufacturing       — DEPT record
2 1119 Smith      50       — EMP record
2 1120 Snyder     50
1 60   Shipping
2 1121 Stevens    60
```

これら2つの形式は、レコードIDフィールドで区別されます。部門レコードの最初の列は1、従業員レコードの最初の列は2になります。このデータをロードするため、次の制御ファイルではフィールドの正確な位置を指定しています。

```
INTO TABLE dept
   WHEN recid = 1
   (recid  FILLER POSITION(1:1)  INTEGER EXTERNAL,
    deptno POSITION(3:4)  INTEGER EXTERNAL,
    dname  POSITION(8:21) CHAR)
INTO TABLE emp
   WHEN recid <> 1
   (recid  FILLER POSITION(1:1)   INTEGER EXTERNAL,
    empno  POSITION(3:6)   INTEGER EXTERNAL,
    ename  POSITION(8:17)  CHAR,
    deptno POSITION(19:20) INTEGER EXTERNAL)
```

- POSITIONパラメータに基づく相対的な位置指定
  レコードをデリミタ付きデータとしてロードできます。

親トピック: 複数のINTO TABLE句を使用するメリット

9.20.2.1 POSITIONパラメータに基づく相対的な位置指定

レコードをデリミタ付きデータとしてロードできます。

前述の例のレコードは、デリミタ付きのデータとしてもロードできます。ただし、その場合はPOSITIONパラメータを使用する必要があります。次の制御ファイルを使用します。

```
INTO TABLE dept
   WHEN recid = 1
   (recid  FILLER INTEGER EXTERNAL TERMINATED BY WHITESPACE,
    deptno INTEGER EXTERNAL TERMINATED BY WHITESPACE,
    dname  CHAR TERMINATED BY WHITESPACE)
INTO TABLE emp
   WHEN recid <> 1
   (recid  FILLER POSITION(1) INTEGER EXTERNAL TERMINATED BY ' ',
    empno  INTEGER EXTERNAL TERMINATED BY ' '
    ename  CHAR TERMINATED BY WHITESPACE,
    deptno INTEGER EXTERNAL TERMINATED BY ' ')
```

2番目のINTO TABLE句のPOSITIONパラメータは、このデータを正しくロードするために必要です。このように指定すると、2つ目の書式に一致するデータを確認するときのフィールド・スキャンは、列1から開始されます。この指定がない場合、SQL*Loaderでは、recidフィールドがdnameフィールドの後にあるものとしてスキャンされます。

親トピック: 異なる入力レコード形式の区別

9.20.3 異なる入力行オブジェクトのサブタイプの区別

通常、データ・ファイルには、同じ行オブジェクト型から継承された行オブジェクトで構成される単一のレコードが含まれています。

たとえば、次のような単純なオブジェクト型およびオブジェクト表の定義について考えてみます。ここでは、NOT FINALのベース・オブジェクト型が、ベース型から行オブジェクトを継承する2つのオブジェクト・サブタイプとともに定義されています。

```
CREATE TYPE person_t AS OBJECT
 (name    VARCHAR2(30),
  age     NUMBER(3)) not final;
CREATE TYPE employee_t UNDER person_t
 (empid   NUMBER(5),
  deptno  NUMBER(4),
  dept    VARCHAR2(30)) not final;
CREATE TYPE student_t UNDER person_t
 (stdid   NUMBER(5),
  major   VARCHAR2(20)) not final;
CREATE TABLE persons OF person_t;
```

次の入力データ・ファイルには、これらの行オブジェクト・サブタイプが混在しています。型IDフィールドは、3つのサブタイプで区別されます。person_tオブジェクトでは、最初の列にPがあり、employee_tオブジェクトではE、student_tオブジェクトではSがあります。

```
P,James,31,
P,Thomas,22,
E,Pat,38,93645,1122,Engineering,
P,Bill,19,
P,Scott,55,
S,Judy,45,27316,English,
S,Karen,34,80356,History,
E,Karen,61,90056,1323,Manufacturing,
S,Pat,29,98625,Spanish,
S,Cody,22,99743,Math,
P,Ted,43,
E,Judy,44,87616,1544,Accounting,
E,Bob,50,63421,1314,Shipping,
S,Bob,32,67420,Psychology,
E,Cody,33,25143,1002,Human Resources,
```

次の制御ファイルでは、POSITIONパラメータに基づく相対的な位置指定によって、このデータをロードします。固有のオブジェクト型名を持つTREAT AS句の使用方法に注意してください。これによって、オブジェクト表のすべての入力行オブジェクトが名前付きオブジェクト型の定義に準拠することが、SQL*Loaderで認識されます。

> ノート:
>
> 複数のサブタイプを同じ INTO TABLE 文ではロードできません。複数の INTO TABLE 文を使用し、それぞれの文で異なるサブタイプをロードする必要があります。

```
INTO TABLE persons
REPLACE WHEN typid = 'P' TREAT AS person_t FIELDS TERMINATED BY "," (typid FILLER
```

POSITION(1) CHAR, name CHAR, age CHAR) INTO TABLE persons REPLACE WHEN typid = 'E' TREAT AS employee_t FIELDS TERMINATED BY "," (typid FILLER POSITION(1) CHAR, name CHAR, age CHAR, empid CHAR, deptno CHAR, dept CHAR) INTO TABLE persons REPLACE WHEN typid = 'S' TREAT AS student_t FIELDS TERMINATED BY "," (typid FILLER POSITION(1) CHAR, name CHAR, age CHAR, stdid CHAR, major CHAR)

関連項目:

オブジェクト型のロードの詳細は、「列オブジェクトのロード」を参照してください。

親トピック: 複数のINTO TABLE句を使用するメリット

9.20.4 複数表へのデータのロード

複数のINTO TABLE句でPOSITIONパラメータを指定すると、1件のレコードのデータを正規化された複数の表にロードできます。

例については、「事例5: 複数表へのデータのロード」を参照してください。(事例の使用方法については、「SQL*Loaderの事例」を参照してください。)

親トピック: 複数のINTO TABLE句を使用するメリット

9.20.5 複数のINTO TABLE句使用のサマリー

複数のINTO TABLE句を指定すると、1件の入力レコードから複数の論理レコードを抽出できます。また、同一ファイル中の異なる形式のレコードを区別できます。

デリミタ付きデータの場合、期待する結果を得るには、POSITIONパラメータを正しく指定する必要があります。

複数のINTO TABLE句でPOSITIONパラメータを指定しない場合、1件の(デリミタ付きデータ)入力レコードの異なる部分が処理されます。これによって、1件のレコードから複数の表へのデータ・ロードが可能になります。複数のINTO TABLE句でPOSITIONパラメータを指定すると、同一のレコードを異なる方法で処理できます。つまり、1つの入力ファイルで複数の形式を識別できます。

親トピック: 複数のINTO TABLE句を使用するメリット

9.21 バインド配列および従来型パス・ロード

まず、一度に複数の行が読み込まれてバインド配列に格納されます。

SQL*Loaderでは、データのデータベースへの転送時にSQL配列インタフェース・オプションが使用されます。SQL*LoaderからOracle DatabaseにINSERTコマンドが送られると、配列全体が一度に挿入されます。バインド配列内の行が挿入された後、COMMIT文が発行されます。

バインド配列サイズを決定する必要があるのは、SQL*Loaderの従来型パス・オプションを使用する場合のみです。通常、これはダイレクト・パス・ロードによる方法では、使用されません。ダイレクト・パス・ロードでは、ダイレクト・パスAPIが使用されるためです。ただし、データ変換を必要とする特殊なダイレクト・パス・ロードの場合にはバインド配列を使用することができます。

- バインド配列のサイズ要件
  バインド配列には、1行以上が入る領域を確保してください。
- バインド配列のパフォーマンスに関する考慮点

バインド配列を大きくすると、Oracle Databaseへのコール数を最小に、またパフォーマンスを最大にできます。

- 行数およびバインド配列サイズの指定
  バインド配列サイズを指定する場合に、コマンドライン・パラメータBINDSIZEまたは制御ファイル中のOPTIONS句を使用すると、バインド配列の上限値が設定されます。
- バインド配列サイズを確認するための計算
  バインド配列サイズは、配列内の行数に各行の最大長を掛け合せた値となります。
- バインド配列用のメモリー所要量の最小化
  VARCHAR型、VARGRAPHIC型フィールド、およびデリミタ付きのCHAR型、DATE型、数値型EXTERNAL型フィールドの場合、そのデータ型に割り当てられているデフォルト・サイズに特に注意してください。
- 複数のINTO TABLE句に対するバインド配列サイズの計算
  制御ファイルに複数のINTO TABLE句が指定されている場合のバインド配列サイズの計算は、複数のINTO TABLE句が指定されていない場合と同様に行います。

関連項目:

ダイレクト・パス・ロードの概要は、『Oracle Call Interfaceプログラマーズ・ガイド』を参照してください

親トピック: SQL*Loader制御ファイル・リファレンス

9.21.1 バインド配列のサイズ要件

バインド配列には、1行以上が入る領域を確保してください。

行の最大長が、BINDSIZEパラメータで指定されたバインド配列のサイズを超えると、SQL*Loaderからエラーが出力されます。通常、バインド配列内に入る範囲の行が格納されます。この場合の読取り行数の上限は、ROWSパラメータで指定された行数となります。(従来型パス・ロードでのROWSの最大値は65534です。)

バインド配列全体が連続するメモリーを占有する必要はありませんが、バインド配列内の各フィールドを格納するバッファには連続するメモリーが必要です。オペレーティング・システムで、フィールド格納用として連続する十分なメモリーを確保できないと、SQL*Loaderからエラーが出力されます。

関連項目:

- BINDSIZE
- ROWS

親トピック: バインド配列および従来型パス・ロード

9.21.2 バインド配列のパフォーマンスに関する考慮点

バインド配列を大きくすると、Oracle Databaseへのコール数を最小に、またパフォーマンスを最大にできます。

一般に、バインド配列サイズを大きくする場合、100行まではサイズの増加に比例してパフォーマンスが大幅に向上します。ただし、100行を超えるバインド配列サイズを設定しても、パフォーマンスはそれほど向上しません。したがって、一般に配列サイズ(バイト単位)は100行が目安となります。

一般に、サイズが適切であれば、SQL*Loaderで効果的に処理できます。通常は、この項で説明するような細かい計算をする必要はありません。この項は、パフォーマンスを最大にする場合、またはメモリー使用量を確認する場合に参照してください。

親トピック: バインド配列および従来型パス・ロード

9.21.3 行数およびバインド配列サイズの指定

バインド配列サイズを指定する場合に、コマンドライン・パラメータBINDSIZEまたは制御ファイル中のOPTIONS句を使用すると、バインド配列の上限値が設定されます。

バインド配列は、この上限値を超えることはありません。

初期化の段階で、SQL*Loaderでは単一行のロードに必要なサイズがバイト単位で決定されます。このサイズが指定された最大値を超える場合は、エラーが返され、ロードが終了します。

次にSQL*Loaderでは、このサイズとロードする行数が乗算されます。このとき、ロードする行数は、コマンドライン・パラメータROWSで指定されていても、制御ファイル中のOPTIONS句で指定されていてもかまいません。

このサイズがバインド配列の最大値を超えないかぎり、ロードは継続されます。SQL*Loaderでは、バインド配列の最大サイズの限界まで行数は拡張されません。行数とバインド配列の最大サイズの両方が指定された場合、SQL*Loaderでは、これらの値の小さい方がバインド配列に適用されます。

バインド配列の最大サイズが小さく、指定の行数を格納できない場合は、その最大サイズに収まる分の行数が採用されます。

親トピック: バインド配列および従来型パス・ロード

9.21.4 バインド配列サイズを確認するための計算

バインド配列サイズは、配列内の行数に各行の最大長を掛け合せた値となります。

行の最大長は、次のように、フィールドの最大長の合計にオーバーヘッドを加えた値となります。

```
bind array size =
    (number of rows) * (  SUM(fixed field lengths)
                        + SUM(maximum varying field lengths)
                        + ( (number of varying length fields)
                              * (size of length indicator) )
                        )
```

ほとんどのフィールドのサイズは、固定長です。このような固定長フィールドの場合、ロードされる各行のサイズは同じです。固定長フィールドについては、「SQL*Loaderのデータ型」で説明するとおり、フィールド・サイズがフィールドの最大長(バイト)となります。そのため、オーバーヘッドは発生しません。

行によってサイズが変化するフィールドには、次のようなものがあります。

- CHAR
- DATE
- INTERVAL DAY TO SECOND
- INTERVAL DAY TO YEAR
- LONG VARRAW
- 数値型EXTERNAL
- TIME
- TIMESTAMP
- TIME WITH TIME ZONE
- TIMESTAMP WITH TIME ZONE
- VARCHAR

- VARCHARC
- VARGRAPHIC
- VARRAW
- VARRAWC

これらのデータ型の最大長の詳細は、「SQL*Loaderのデータ型」を参照してください。ここでの最大長とは、入力データ・レコードの中でフィールドが占有できる長さを、バイト数で表したものです。この最大長は、バインド配列の中で各フィールドが占有する格納領域のサイズも表しています。バインド配列には、サイズが変化するこれらのフィールドについてのオーバーヘッドも含まれます。

文字データ型(CHAR、DATEおよび数値型EXTERNAL)がデリミタ付きで指定された場合は、これらのフィールドに対して指定されたフィールド長が最大長となります。逆に、デリミタなしでこれらのデータ型が指定された場合は、レコードのサイズは固定ですが、挿入時にフィールド中の空白文字が切り捨てられるため、フィールド長は変化します。したがって、これらのデータ型は、たとえ固定長フィールドであっても内部的には可変長フィールドとして扱われます。

長さインジケータは、バインド配列内のそれぞれのフィールドに格納されています。バインド配列におけるフィールドの領域として、そのフィールドの可能最大長のデータを格納できるだけのサイズが確保されています。一方、実際のフィールド長は、行ごとに長さインジケータで示されます。

ノート:

従来型パス・ロードでは、バインド配列のサイズの割当て時に LOBFILE は含まれません。

- 長さインジケータのサイズの決定
  制御ファイルを使用して、長さインジケータのサイズを決定します。
- フィールド・バッファ・サイズの計算
  この項では、フィールド・バッファ・サイズを計算する方法について説明します。

親トピック: バインド配列および従来型パス・ロード

9.21.4.1 長さインジケータのサイズの決定

制御ファイルを使用して、長さインジケータのサイズを決定します。

ほとんどのシステムでは、長さインジケータのサイズは2バイトです。まれに3バイトのシステムもあります。長さインジケータのサイズを調べるには、次の制御ファイルを作成して実行します。

```
OPTIONS (ROWS=1)
LOAD DATA
INFILE *
APPEND
INTO TABLE DEPT
(deptno POSITION(1:1) CHAR(1))
BEGINDATA
a
```

この制御ファイルは、1行のバインド配列を使用して、1バイトのCHARをロードします。この例では、実際にデータはロードされません。これは、aを数値型の列(deptno)にロードすると、変換エラーが発生するためです。このときのログ・ファイルに示されたバインド配列サイズから、(文字フィールドの長さである)1を引いた値が、フィールド長のインジケータのサイズとなります。

ノート:

これと同様の方法で、計算しないでバインド配列サイズを求めることもできます。制御ファイルにデータを記述せず、ROWS=1 と指定して実行すると、1 行のデータに必要なメモリーのサイズがわかります。このサイズとバインド配列に格納する行数を掛け合せれば、バインド配列サイズを判断できます。

親トピック: バインド配列サイズを確認するための計算

9.21.4.2 フィールド・バッファ・サイズの計算

この項では、フィールド・バッファ・サイズを計算する方法について説明します。

表9-3から表9-6の表に、各データ型のメモリー要件を示します。「L」は制御ファイルで指定したデータ長です。「P」は精度です。「S」はフィールド長インジケータのサイズです。これらの値の詳細は、「SQL*Loaderのデータ型」を参照してください。

表9-3 固定長フィールド

データ型	バイト単位のサイズ(オペレーティング・システムによって異なる)
INTEGER	C の INT データ型に相当するサイズ
INTEGER(N)	N バイト
SMALLINT	C の SHORT INT データ型に相当するサイズ
FLOAT	C の FLOAT データ型に相当するサイズ
DOUBLE	C の DOUBLE データ型に相当するサイズ
BYTEINT	C の UNSIGNED CHAR データ型に相当するサイズ
VARRAW	UNSIGNED SHORT に 4096 バイトまたは max_length に指定した値をプラスしたサイズ
LONG VARRAW	UNSIGNED INT に 4096 バイトまたは max_length に指定した値をプラスしたサイズ
VARCHARC	2 つの数値で構成されます。最初に長さを指定し、次に(オプションで)max_length(デフォルトは 4096 バイト)を指定します。
VARRAWC	このデータ型は RAW データ用です。2 つの数値で構成されます。最初に長さを指定し、次に(オプションで)max_length(デフォルトは 4096 バイト)を指定します。

表9-4 非グラフィック・フィールド

データ型	デフォルト・サイズ	指定するサイズ
(パック) DECIMAL	なし	(N+1)/2 切上げ
ZONED	なし	P
RAW	なし	L
CHAR 型(デリミタなし)	1	L + S
日時データ型および期間データ型(デリミタなし)	なし	L + S
数値型 EXTERNAL 型(デリミタなし)	なし	L + S

表9-5 グラフィック・フィールド

データ型	デフォルト・サイズ	POSITIONでの長さの指定	DATA TYPEでの長さの指定
GRAPHIC	なし	L	2×L
GRAPHIC EXTERNAL	なし	L - 2	2×(L-2)
VARGRAPHIC	4KB×2	L + S	(2×L)+S

表9-6 可変長フィールド

データ型	デフォルト・サイズ	最大長の指定(L)
VARCHAR	4 KB	L + S
CHAR 型(デリミタ付き)	255	L + S
日時データ型および期間データ型(デリミタ付き)	255	L + S
数値型 EXTERNAL 型(デリミタ付き)	255	L + S

親トピック: バインド配列サイズを確認するための計算

9.21.5 バインド配列用のメモリー所要量の最小化

VARCHAR型、VARGRAPHIC型フィールド、およびデリミタ付きのCHAR型、DATE型、数値型EXTERNAL型フィールドの場合、そのデータ型に割り当てられているデフォルト・サイズに特に注意してください。

これは、特にバインド配列の行数で乗算されると、大量のメモリーを消費する可能性があります。これらのフィールドに対しては、

最大長としてできるだけ小さな値を指定してください。次の例を検討してください:

```
CHAR(10) TERMINATED BY ","
```

バイト長セマンティクスを使用する場合、次の例では、バインド配列で(10+2)×64=768バイトのメモリーを使用します(ここでは、長さインジケータを2バイトで一度に64行ロードするとして計算しています)。

同じ例で文字長セマンティクスを使用する場合、バインド配列では((10 * s) + 2) * 64バイトのメモリーを使用します(ここでは、「s」はデータ・ファイル文字セットにおける1文字のバイト単位での最大サイズです)。

ここで、次の例について考えてみます。

```
CHAR TERMINATED BY ","
```

バイト長セマンティクスまたは文字長セマンティクスを使用しているかどうかにかかわらず、この例では、(255+2)×64=16,448バイトが必要になります。これは、デリミタ付きフィールドのデフォルト最大長が255バイトであるためです。この指定によって、バインド配列に入る行数が大きく違ってきます。

親トピック: バインド配列および従来型パス・ロード

9.21.6 複数のINTO TABLE句に対するバインド配列サイズの計算

制御ファイルに複数のINTO TABLE句が指定されている場合のバインド配列サイズの計算は、複数のINTO TABLE句が指定されていない場合と同様に行います。

いい換えると、制御ファイルに指定されているフィールド全体を1つの長いデータ構造体、つまりバインド配列の中の1行のデータ構造体であると考えます。

データ・レコード中の同じフィールドを複数のINTO TABLE句が参照する場合は、そのフィールドが参照されると、常に、バインド配列に追加の領域が必要となります。このようなフィールドについては、特にバッファの割当てを最小限に抑える必要があります。

> ノート:
>
> CONSTANT、EXPRESSION、RECNUM、SYSDATE および SEQUENCE の各関数を指定すると、SQL*Loader によってデータが生成されます。このようにして生成されたデータは、バインド配列の領域を必要としません。

親トピック: バインド配列および従来型パス・ロード

10 SQL*Loaderフィールド・リスト・リファレンス

SQL*Loader制御ファイルのフィールド・リストには、位置、データ型、条件、デリミタなど、ロードするフィールドの情報が提供されます。

- フィールド・リストの内容
  SQL*Loader制御ファイルのフィールド・リストには、ロードするフィールドの情報が提供されます。
- データ・フィールドの位置指定
  データ・ファイルからデータをロードする場合は、フィールドの位置と長さをSQL*Loaderに対して明示する必要があります。
- 列およびフィールドの指定
  表の列はいくつでもロードできます。
- SQL*Loaderのデータ型
  SQL*Loaderのデータ型は、移植可能なデータ型と移植不能なデータ型にグループ化できます。
- フィールド条件の指定
  フィールド条件とは、論理レコード中のフィールドに関して、それが真か偽かを評価する条件を記述したものです。
- WHEN、NULLIFおよびDEFAULTIF句の使用
  SQL*Loaderでの、スカラー・フィールドを使用したWHEN、NULLIFおよびDEFAULTIF句の処理方法について学習します。
- WHEN、NULLIFおよびDEFAULTIF句の使用例
  これらの例では、WHEN、NULLIFおよびDEFAULTIF句を使用できる様々な状況の結果について説明します。
- 異なるプラットフォーム間でのデータのロード
  データ・ファイルを作成するプラットフォームと、そのデータ・ファイルのロード先となるプラットフォームが異なる場合は、ターゲット・システムが読取り可能な形式でデータを作成する必要があります。
- バイト順序
  SQL*Loaderを実行するシステムとは異なるバイト順序スキームを持つシステムで入力データを作成する予定の場合は、バイト順序指定を確認してください。
- すべてが空白のフィールドのロード
  フィールドが完全に空白のレコードは拒否されます。これらのフィールドのいずれかをNULLとしてロードするには、BLANKSパラメータを持つNULLIF句を使用します。
- 空白の切捨て
  空白、タブおよびその他の印字されない文字(改行、LFなど)が空白を構成します。
- 空白の切捨てに対するPRESERVE BLANKSオプションの影響
  すべてのCHARフィールド、DATEフィールド、および数値型EXTERNALフィールドで空白が切り捨てられないようにするには、制御ファイル内のLOAD文でPRESERVE BLANKSを指定します。
- [NO] PRESERVE BLANKSとデリミタ句の併用
  PRESERVE BLANKSオプションは、デリミタ句の存在の影響を受けます
- フィールドへのSQL演算子の適用
  この項では、SQL演算子をフィールドに適用する方法について説明します。
- SQL*Loaderを使用した入力データの生成
  この項では、データ・ファイルからデータを読み込むのではなく、SQL*Loaderでデータを生成してデータベース・レコードに格納するパラメータについて説明します。

親トピック: SQL*Loader

10.1 フィールド・リストの内容

SQL*Loader制御ファイルのフィールド・リストには、ロードするフィールドの情報が提供されます。

フィールドには、位置、データ型、条件、デリミタなどがあります。

例10-1に、「SQL*Loader制御ファイル・リファレンス」で説明したサンプル制御ファイルのフィールド・リスト・セクションを示します。

例10-1 サンプル制御ファイルのフィールド・リスト・セクション

```
.
.
.
1  (hiredate  SYSDATE,
2     deptno  POSITION(1:2)  INTEGER EXTERNAL(2)
              NULLIF deptno=BLANKS,
3       job   POSITION(7:14)  CHAR  TERMINATED BY WHITESPACE
              NULLIF job=BLANKS  "UPPER(:job)",
       mgr    POSITION(28:31) INTEGER EXTERNAL
              TERMINATED BY WHITESPACE, NULLIF mgr=BLANKS,
       ename  POSITION(34:41) CHAR
              TERMINATED BY WHITESPACE  "UPPER(:ename)",
       empno  POSITION(45) INTEGER EXTERNAL
              TERMINATED BY WHITESPACE,
       sal    POSITION(51) CHAR  TERMINATED BY WHITESPACE
              "TO_NUMBER(:sal,'$99,999.99')",
4      comm   INTEGER EXTERNAL  ENCLOSED BY '(' AND '%'
              ":comm * 100"
    )
```

このサンプル制御ファイルの左側の数字は、実際の制御ファイルでは表示されません。これらの数字は、次の説明の番号に対応しています。

1. SYSDATEに、列を現在のシステム日付に設定します。詳細は、「列への現在の日付の設定」を参照してください。
2. POSITIONに、データ・フィールドの位置を指定します。詳細は、「データ・フィールドの位置指定」を参照してください。

   INTEGER EXTERNALは、フィールドのデータ型です。詳細は、「データ・フィールドのデータ型の指定」および「数値型EXTERNAL」を参照してください。

   NULLIF句は、フィールド条件の指定に使用する句の1つです。詳細は、「WHEN、NULLIFおよびDEFAULTIF句の使用」を参照してください。

   このサンプルでは、BLANKSパラメータを使用して、フィールドを空白と比較しています。詳細は、「フィールドとBLANKSの比較」を参照してください。
3. TERMINATED BY WHITESPACE句は、フィールドを指定できるデリミタの1つです。詳細は、「デリミタの指定」を参照してください。
4. ENCLOSED BY句は、もう1つの使用可能なフィールド・デリミタです。詳細は、「デリミタの指定」を参照してください。

親トピック: SQL*Loaderフィールド・リスト・リファレンス

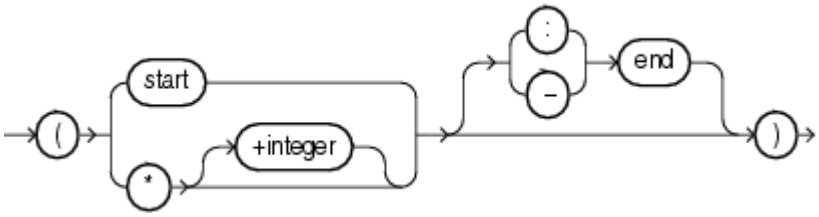
10.2 データ・フィールドの位置指定

データ・ファイルからデータをロードする場合は、フィールドの位置と長さをSQL*Loaderに対して明示する必要があります。

論理レコード中のフィールド位置は、列指定の中でPOSITION句を使用して指定します。このときフィールド位置は、明示的に指定することも、前のフィールドからの相対位置で指定することもできます。POSITIONに対する引数は、カッコで囲む必要があ

ります。文字長セマンティクスをデータ・ファイルに使用しても、start、endおよびintegerは、常にバイト単位です。

位置指定(pos_spec)句の構文は次のとおりです。



次の表では、位置指定句のパラメータについて説明します。

表10-1 位置指定句のパラメータ

パラメータ	説明
start	論理レコード中のデータ・フィールドの開始位置です。論理レコードの先頭バイト位置は1となります。
end	論理レコード中のデータ・フィールドの終了位置です。start-end と表記することも、start:end と表記することもできます。end を省略した場合、フィールド長は、データ・ファイル中のデータ型から導出されます。CHAR 型では、start または end を省略し、長さ指定(CHAR(n))をしない場合、データ長が 1 として扱われます。データ型から長さを導出できない場合は、エラー・メッセージが出力されます。
*	対象となるデータ・フィールドが前のフィールドの直後にあることを示します。制御ファイル中の最初のデータ・フィールドに対して*を指定した場合、そのフィールドは論理レコードの先頭であると判断されます。位置指定に*を使用した場合、フィールド長はデータ型から導出されます。
+integer	+integer を指定してオフセットを使用すると、前フィールドの終了位置直後の位置から現行のフィールドをオフセットできます。この場合、現行のフィールドの値は、+integer で指定されたバイト数分スキップした後で読み込まれます。

POSITIONを完全に省略することも可能です。省略した場合のデータ・フィールドの位置指定は、POSITION(*)と指定した場合と同じです。

- タブを含むデータでのPOSITIONの使用
  フィールド位置の指定で、データ・ファイル中にタブが含まれている場合は注意が必要です。
- 複数表のロードでのPOSITIONの使用
  この項では、複数表のロードでPOSITIONを使用する方法について説明します。
- POSITIONを使用した例
  この項では、POSITIONを使用する例を示します。

親トピック: SQL*Loaderフィールド・リスト・リファレンス

10.2.1 タブを含むデータでのPOSITIONの使用

フィールド位置の指定で、データ・ファイル中にタブが含まれている場合は注意が必要です。

SQL*Loaderの拡張SQL文字列機能を使用して、書式設定されたレポートのデータをロードするとします。最初に、レポートの印刷出力を見て、すべての文字位置を正確に調べ、制御ファイルを作成します。このような状況でデータをロードしようとすると、無効な数字および欠落フィールドによる多数のエラーが発生し、ロードが失敗する場合があります。

これらの種類のエラーはデータにタブが含まれる場合に発生します。用紙上では、各タブは複数の列に分かれて印刷されます。一方、データ・ファイル内では各タブは単なる1文字です。この結果、データ・ファイルがSQL*Loaderによって読み取られるとき、POSITION指定が正しくなくなります。

この問題を解決するには、データ・ファイル中のタブを探して該当箇所のPOSITION指定を調整するか、フィールドをデリミタで区切ります。

関連項目:

デリミタの指定

親トピック: データ・フィールドの位置指定

10.2.2 複数表のロードでのPOSITIONの使用

この項では、複数表のロードでPOSITIONを使用する方法について説明します。

複数表のロードでは、複数のINTO TABLE句を指定します。このとき最初の表の最初の列に対してPOSITION(*)を使用すると、論理レコードの先頭から相対的に位置が計算されます。2番目以降の表の最初の列に対してPOSITION(*)が使用された場合は、その時点で最後にロードされた表の最終列から相対的に位置が計算されます。

したがって、2番目以降のINTO TABLE句の処理開始時に、位置が自動的に論理レコードの先頭に設定されるわけではありません。このため、複数のINTO TABLE句を指定して、同一物理レコード中の異なる箇所を処理できます。例については、「複数の論理レコードの抽出」を参照してください。

論理レコード中のデータには、2つの表の両方ではなく、片方の表のみにロードするデータもあります。その場合は、POSITIONを設定しなおす必要があります。このとき、位置指定を省略するか、またはINTO TABLE句で先頭フィールドに対するPOSITION(*+n)を指定するかわりに、POSITION(1)またはPOSITION(n)を指定します。

親トピック: データ・フィールドの位置指定

10.2.3 POSITIONを使用した例

この項では、POSITIONを使用する例を示します。

```
siteid  POSITION (*) SMALLINT
siteloc POSITION (*) INTEGER
```

これらが最初の2つの列を指している場合、siteidは1列目から始まり、sitelocがその次の列から始まります。

```
ename  POSITION (1:20)  CHAR
empno  POSITION (22-26) INTEGER EXTERNAL
allow  POSITION (*+2)   INTEGER EXTERNAL TERMINATED BY "/"
```

列enameは、位置1から20までを占める文字データで、その次の列empnoは、位置22から26までを占める数値型データです。列allowは、empnoの終了直後の位置(27)から+2の位置、つまり列29から、スラッシュが検出されるまでのデータとなります。

親トピック: データ・フィールドの位置指定

10.3 列およびフィールドの指定

表の列はいくつでもロードできます。

データベース中に定義されていて制御ファイル中で指定されていない列には、NULL値が割り当てられます。

列指定には、列名とその列に入る値の指定を記述します。これらの列指定はカンマで区切って、全体を小カッコで囲みます。

```
(columnspec,columnspec, ...)
```

(FILLERのマークが付けられていない)それぞれの列名は、INTO TABLE句で指定した表中の列名に対応させてください。列名にSQLやSQL*Loaderの予約語または特殊文字が含まれているか、大/小文字の区別がある場合は、列名を引用符で囲みます。

列の値をSQL*Loaderで生成する場合は、列指定の中でRECNUMパラメータ、SEQUENCEパラメータまたはCONSTANTパラメータを指定します。詳細は、「SQL*Loaderを使用した入力データの生成」を参照してください。

データ・ファイルから列の値を読み込む場合は、列の値に対応するデータ・フィールドを指定します。このとき、列指定には、データベース表中の列を示す列名(column name)およびデータ・レコード中のフィールドを示すフィールド指定(field specification)を指定します。フィールド指定には、フィールドの位置、データ型、NULL値の制限およびデフォルト値を指定します。

列オブジェクトのロード時に、必ずしもすべての属性を指定する必要はありません。指定しなかった属性には、NULLが設定されます。

- FILLERフィールドの指定
  BOUNDFILLERまたはFILLERで指定されたFILLERフィールドは、データ・ファイルをマップしたフィールドであり、データベースの列には対応しません。
- データ・フィールドのデータ型の指定
  フィールドのデータ型を指定することによって、SQL*Loaderでフィールドのデータが処理される方法が決まります。

親トピック: SQL*Loaderフィールド・リスト・リファレンス

10.3.1 FILLERフィールドの指定

BOUNDFILLERまたはFILLERで指定されたFILLERフィールドは、データ・ファイルをマップしたフィールドで、データベースの列とは対応しません。

FILLERフィールドは、データ・ファイルがマップされているデータ・フィールドから割り当てられた値です。

FILLERフィールドに関しては次のことに注意してください。

- FILLERフィールドの構文は、フィールド名の後にFILLERを付けること以外は、列ベースのフィールドと同じです。
- FILLERフィールドには名前がありますが、表にはロードされません。
- FILLERフィールドは、引数としてinit_specs(たとえば、NULLIFおよびDEFAULTIF)に使用できます。
- FILLERフィールドは、引数として指示句(たとえば、SID、OID、REFおよびBFILE)に使用できます。

  Fillerフィールドが BFILEなどの指示句で参照され、列オブジェクト内の制御ファイルで宣言されている場合にあいまいな指定を行わないようにするために、フィールド名にオブジェクトの名前を修飾する必要があります。これは次の例で説明します。

  ```
  LOAD DATA
  INFILE *
  ```

```
INTO TABLE BFILE10_TBL REPLACE
FIELDS TERMINATED BY ','
(
   emp_number char,
   emp_info_b column object
   (
   bfile_name FILLER char(12),
   emp_b BFILE(constant "SQLOP_DIR", emp_info_b.bfile_name) NULLIF
  emp_info_b.bfile_name = 'NULL'
   )
)
BEGINDATA
00001,bfile1.dat,
00002,bfile2.dat,
00003,bfile3.dat,
```

- FILLERフィールドは、NULLIF句、DEFAULTIF句およびWHEN句のフィールド条件指定で使用できます。ただし、SQL文字列では使用できません。
- FILLERフィールドの指定に、NULLIF句またはDEFAULTIF句を含めることはできません。
- FILLERフィールドは、TRAILING NULLCOLSが指定されて適用可能な場合、NULLで初期化されます。他のフィールドが、無効なFILLERフィールドを参照している場合は、エラーになります。
- FILLERフィールドは、オブジェクトのフィールド・リスト内部またはVARRAYの定義の内部を含む、データ・ファイルのどこにでも指定できます。
- バイナリ配列のFILLERには領域が割り当てられていないため、SQL文字列をFILLERフィールドの一部としては指定できません。

ノート:

この項で説明する内容は、BOUNDFILLER を使用したバウンド FILLER の指定にも適用されます。唯一の例外として、バイナリ配列のバウンド FILLER には領域が割り当てられているため、このバウンド FILLER を使用して、SQL 文字列をフィールドの一部として指定できます。

例10-2 FILLERフィールド指定

FILLERフィールド指定は、次のようになります。

```
field_1_count FILLER char,
field_1 varray count(field_1_count)
(
   filler_field1  char(2),
   field_1  column object
   (
     attr1 char(2),
     filler_field2  char(2),
     attr2 char(2),
   )
   filler_field3  char(3),
)
filler_field4 char(6)
```

親トピック: 列およびフィールドの指定

10.3.2 データ・フィールドのデータ型の指定

フィールドのデータ型を指定することによって、SQL*Loaderでフィールドのデータが処理される方法が決まります。

たとえば、INTEGERデータ型を指定するとバイナリ・データとして処理され、INTEGER EXTERNALを指定すると数字を表す文字データとして処理されます。CHARを指定したフィールドには、すべての文字データを含むことができます。

各フィールドに指定できるデータ型は1つのみで、データ型が指定されていない場合、CHARであるとみなされます。

データ型を指定する前に、フィールドの位置を指定する必要があります。

SQL*Loaderデータ型をOracleデータ型に変換する方法と、各SQL*Loaderデータ型の詳細は、「SQL*Loaderのデータ型」を参照してください。

関連トピック

- SQL*Loaderのデータ型

親トピック: 列およびフィールドの指定

10.4 SQL*Loaderのデータ型

SQL*Loaderのデータ型は、移植可能なデータ型と移植不能なデータ型にグループ化できます。

- 移植可能なデータ型と移植不能なデータ型の相違点
  SQL*Loaderの移植可能なデータ型は、プラットフォームに依存しません。移植不能なデータ型は、移植性に影響する複数の依存性を持つことがあります。
- 移植不能なデータ型
  このリファレンスを使用して、SQL*Loaderで移植不能なデータ型を使用する方法について理解してください。
- 移植可能なデータ型
  このリファレンスを使用して、SQL*Loaderで移植可能なデータ型を使用する方法について理解してください。
- データ型変換
  SQL*Loaderは、ほとんどのデータ型の変換を自動で実行しますが、エラーを防止するために変換ルールについて理解しておく必要があります。
- 日時データ型および期間データ型のデータ型変換
  Oracle DatabaseのデータとSQL*Loader制御ファイルの日時データ型および期間データ型の間でサポートされる変換とサポートされない変換について説明します。
- デリミタの指定
  CHAR型、日時型、期間型または数値EXTERNAL型フィールドの境界は、デリミタ文字を使用して、入力データ・レコード中でも指定できます。
- デリミタ付きデータの処理方法
  デリミタは、TERMINATED BY、ENCLOSED BYおよび OPTIONALLY ENCLOSED BY句の様々な組合せをフィールド定義で使用して定義できます。
- 文字データ型フィールド長の競合
  CHAR型、DATE型および数値EXTERNAL型の文字データ・フィールドの場合は、そのフィールド長を制御ファイルに複数指定できます。

親トピック: SQL*Loaderフィールド・リスト・リファレンス

10.4.1 移植可能なデータ型と移植不能なデータ型の相違点

SQL*Loaderの移植可能なデータ型は、プラットフォームに依存しません。移植不能なデータ型は、移植性に影響する複数の依存性を持つことがあります。

SQL*Loaderデータ型の場合、データ型はさらにVALUEデータ型とLENGTH-VALUEデータ型にグループ化されます。

移植可能なデータ型と**移植不能なデータ型**という用語は、そのデータ型がプラットフォームに依存しているかどうかを表しています。プラットフォーム依存性は、異なるプラットフォームのバイト順序スキーム(ビッグ・エンディアンとリトル・エンディアン)の違い、プラットフォームのビット数(16ビット、32ビット、64ビット)の違い、符号付き数表現のスキーマの違い(2の補数と1の補数)などの様々な理由から存在します。バイト順序スキーム、プラットフォームのワード長などの場合は、SQL*Loaderで、プラットフォーム依存性を解決するメカニズムが提供されます。これらのメカニズムの詳細は、該当するデータ型の説明を参照してください。

移植可能なデータ型および移植不能なデータ型の両方ともVALUEまたはLENGTH-VALUEをとることができます。VALUEデータ型のデータ・フィールド部分は単一であるとします。LENGTH-VALUEデータ型では、データ・フィールドが2つのサブフィールド(lengthサブフィールドがvalueサブフィールドの長さを指定)で構成される必要があります。

> ノート:
>
> Oracle Database 12c リリース 1 (12.1)以降のリリースでは、Oracle Database のデータ型 VARCHAR2、NVARCHAR2 および RAW の最大サイズは 32KB です。このサイズを確保するには、COMPATIBLE 初期化パラメータを 12.0 以降に設定して、MAX_STRING_SIZE 初期化パラメータを EXTENDED に設定する必要があります。SQL*Loader では、この最大サイズがサポートされます。

親トピック: SQL*Loaderのデータ型

10.4.2 移植不能なデータ型

このリファレンスを使用して、SQL*Loaderで移植不能なデータ型を使用する方法について理解してください。

- 移植不能なデータ型のカテゴリ
  移植不能なデータ型は、**VALUEデータ型**と**LENGTH-VALUEデータ型**の2つグループに分類されます。
- INTEGER(n)
  SQL*Loaderの移植不能なVALUEデータ型INTEGER(n)は長さ固有の整数フィールドです。
- SMALLINT
  SQL*Loaderの移植不能なVALUEデータ型SMALLINTは、ハーフ・ワードの2進整数です。
- FLOAT
  SQL*Loaderの移植不能なVALUEデータ型FLOATは、単精度浮動小数点2進数です
- DOUBLE
  SQL*Loaderの移植不能なVALUEデータ型DOUBLEは、倍精度浮動小数点2進数です。
- BYTEINT
  SQL*Loaderの移植不能なVALUEデータ型BYTEINTには、2進数表現のバイトの10進数の値をロードします。
- ZONED
  SQL*Loaderの移植不能なVALUEデータ型ZONEDは、ZONED型の10進数形式で表現されます。
- DECIMAL
- VARGRAPHIC
  SQL*Loaderの移植不能なLENGTH-VALUEデータ型VARGRAPHICは、可変長のダブルバイト文字セット

(DBCS)です。

- VARCHAR
  SQL*Loaderの移植不能なLENGTH-VALUEデータ型VARCHARは、バイナリのlengthサブフィールドとその長さの文字列です。
- VARRAW
  SQL*Loaderの移植不能なLENGTH-VALUEデータ型VARROWは、2バイトのバイナリのlengthサブフィールドとRAW文字列のVALUEサブフィールドです。
- LONG VARRAW
  SQL*Loaderの移植不能なLENGTH-VALUEデータ型LONG VARRAWは、lengthサブフィールドが4バイトのVARRAWです。

親トピック: SQL*Loaderのデータ型

10.4.2.1 移植不能なデータ型のカテゴリ

移植不能なデータ型は、**VALUEデータ型**と**LENGTH-VALUEデータ型**の2つグループに分類されます。

移植不能なVALUEデータ型は次のとおりです。

- INTEGER(n)
- SMALLINT
- FLOAT
- DOUBLE
- BYTEINT
- ZONED
- (パック) DECIMAL

移植不能なLENGTH-VALUEデータ型は次のとおりです。

- VARGRAPHIC
- VARCHAR
- VARRAW
- LONG VARRAW

移植不能なデータ型について詳しく理解するために、datatype_specの構文図を参照してください。

関連トピック

- SQL*Loaderの構文図

親トピック: 移植不能なデータ型

10.4.2.2 INTEGER(n)

SQL*Loaderの移植不能なVALUEデータ型INTEGER(n)は長さ固有の整数フィールドです。

定義

データは、フルワード2進整数(nは、1、2、4または8から選択して指定した長さ)です。長さが指定されていない場合、その長さはバイト単位で、ご使用のプラットフォームのCプログラミング言語におけるLONG INTのサイズに基づいて決まります。

使用上のノート

INTEGERは、バイト・サイズ、バイト順序および符号付きの値の表現がシステム間で異なるため、移植できません。ただし、符号付きの値の表現がシステム間で同じ場合は、SQL*LoaderがINTEGERデータにアクセスして、正しい結果を得る可能性があります。長さ指定(n)でINTEGERを指定し、必要に応じて適切な方法でデータのバイト順序を指定すると、SQL*Loaderを使用してシステム間で正しい結果でデータにアクセスできます。長さ指定なしにINTEGERを指定すると、Cプログラミング言語

のLONG INTの長さが両方のシステムで同じバイト数である場合のみ、SQL*Loaderを使用して正しい結果でデータにアクセスできます。その場合も、必要に応じて、適切な方法でデータのバイト順序を指定する必要があります。

2進整数の長さを明示的に指定すると、ワード長がSQL*Loaderで使用されているものとは異なるプラットフォーム上で入力データを作成する場合に有効です。たとえば、2進整数を含む入力データは、64ビットのプラットフォームで作成され、32ビットのプラットフォーム上のSQL*Loaderを使用しているデータベースにロードされます。この場合、INTEGER(8)を指定して、SQL*Loaderによってその整数を4バイトの量ではなく、8バイトの量として処理します。

デフォルトでは、INTEGERはSIGNED量として処理されます。SQL*Loaderで、符号なしの量として処理する場合は、UNSIGNEDを指定します。デフォルトの動作に戻すには、SIGNEDを指定します。

関連トピック

- 異なるプラットフォーム間でのデータのロード

親トピック: 移植不能なデータ型

10.4.2.3 SMALLINT

SQL*Loaderの移植不能なVALUEデータ型SMALLINTは、ハーフ・ワードの2進整数です。

定義

SMALLINTフィールド長には、使用しているシステムのハーフワード整数の長さが取られます。

使用上のノート

デフォルトでは、SMALLINTデータはSIGNED量として処理されます。SQL*Loaderで、符号なしの量として処理する場合は、UNSIGNEDを指定します。デフォルトの動作に戻すには、SIGNEDを指定します。

SMALLINTデータは、SHORT INTの長さが同じバイト数のシステム間のみで、正しい結果でロードできます。バイト順序がシステム間で異なる場合は、適切な方法でデータのバイト順序を指定します。

ノート:

これは、C プログラミング言語の SHORT INT データ型です。フィールド長を決定する 1 つの方法は、データを入れずに小さい制御ファイルを作成し、その結果のログ・ファイルを調べることです。このデータ型のサイズは、制御ファイルを使用しては変更できません。

関連トピック

- SQL*Loaderによるバイト順序の管理方法の理解

親トピック: 移植不能なデータ型

10.4.2.4 FLOAT

SQL*Loaderの移植不能なVALUEデータ型FLOATは、単精度浮動小数点2進数です

定義

FLOATデータの場合、このフィールド長にはシステムの単精度浮動小数点2進数の長さが取られます。(データ型はCのFLOATです。)このデータ型のサイズは、制御ファイルを使用しては変更できません。POSITION句でendを指定するとendは無視されます。

使用上のノート

FLOATは、FLOATの表現に互換性があり、長さも同じシステムの間でのみ、正しい結果でロードできます。バイト順序が2つのシステム間で異なる場合は、適切な方法でデータのバイト順序を指定します。

関連トピック

- SQL*Loaderによるバイト順序の管理方法の理解

親トピック: 移植不能なデータ型

10.4.2.5 DOUBLE

SQL*Loaderの移植不能なVALUEデータ型DOUBLEは、倍精度浮動小数点2進数です。

定義

DOUBLEフィールドの長さには、システムの倍精度浮動小数点2進数の長さが取られます。(データ型はCのDOUBLEまたはLONG FLOATです。)このデータ型のサイズは、制御ファイルを使用しては変更できません。POSITION句でendを指定するとendは無視されます。

使用上のノート

DOUBLEは、DOUBLEの表現に互換性があり、長さも同じシステムの間でのみ、正しい結果でロードできます。バイト順序が2つのシステム間で異なる場合は、適切な方法でデータのバイト順序を指定します。

関連トピック

- SQL*Loaderによるバイト順序の管理方法の理解

親トピック: 移植不能なデータ型

10.4.2.6 BYTEINT

SQL*Loaderの移植不能なVALUEデータ型BYTEINTには、2進数表現のバイトの10進数の値をロードします。

定義

2進数で表されている1バイト分のデータを10進数に直した値がロードされます。たとえば、入力文字x"1C"は28としてロードされます。BYTEINTフィールドの長さは、常に1バイトになります。POSITION (start:end)を指定すると、endは無視されます。(データ型はCのUNSIGNED CHARです。)

例

このデータ型の構文例は次のとおりです。

```
(column1 position(1) BYTEINT,
column2 BYTEINT,
...
)
```

親トピック: 移植不能なデータ型

10.4.2.7 ZONED

SQL*Loaderの移植不能なVALUEデータ型ZONEDは、ZONED型の10進数形式で表現されます。

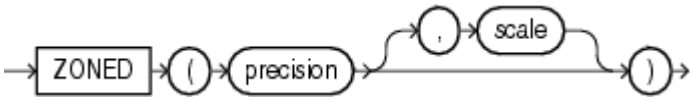
定義

ZONED型のデータは、ZONED型の10進数形式で表現されます。つまり、10進数の各1桁が1バイトで表され、最終バイトに

符号が入ります。(COBOLのSIGN TRAILINGフィールドに相当)このフィールド長には、精度(桁数)として指定された長さが取られます。

構文

ZONEDデータ型の構文は次のとおりです。



ここでのprecisionは数字の桁数です。scale(指定されている場合)は(暗黙の)小数点の右側の桁数です。

例

次の例では、位置32から始まる8桁の整数を表します。

```
sal  POSITION(32)  ZONED(8),
```

ZONED型のデータがASCIIベースのプラットフォームで生成される場合、Oracle DatabaseはVAX/VMS ZONED型10進数形式を使用します。EBCDICベースのプラットフォームで生成されるZONED型10進データのロードも可能です。この場合、Oracle Databaseでは、マニュアル『*ESA/390 Principles of Operations*』バージョン8.1で指定されているIBM形式を使用します。使用される形式は、入力データ・ファイルの文字セット・エンコーディングによって異なります。

関連トピック

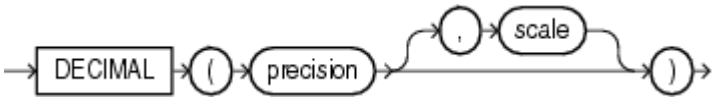
- CHARACTERSETパラメータ

親トピック: 移植不能なデータ型

10.4.2.8 DECIMAL

DECIMAL型のデータは、PACKED型の10進数形式で記述されます。つまり、10進数の各2桁が1バイトで表され、最終バイトに1桁と符号が入ります。DECIMALフィールドでは暗黙の小数点位置を指定できるため、分数の値を表すこともできます。

DECIMALデータ型の構文は次のとおりです。



precisionパラメータは、数値の桁数です。DECIMALフィールドのバイト長は、桁数から計算します。(N+1)/2を求め、その小数点以下を切り上げた値がバイト長となります。

scaleパラメータは、小数点の右側にくる桁数のことで、スケール変更係数と呼びます。デフォルト値は0(ゼロ)です(整数となります)。全体の桁数より大きい数は指定できますが、負数は指定できません。

次に例を示します。

```
sal DECIMAL (7,2)
```

この例では、+12345.67の形式の数値がロードされます。このフィールドは、データ・レコード中で4バイトを占有します。(DECIMALフィールドのバイト長は、(N+1)/2の小数点以下を切り上げた値になります。ここで、Nは数値の桁数です。また、1は符号用として追加されています。)

親トピック: 移植不能なデータ型

10.4.2.9 VARGRAPHIC

SQL*Loaderの移植不能なLENGTH-VALUEデータ型VARGRAPHICは、可変長のダブルバイト文字セット(DBCS)です。

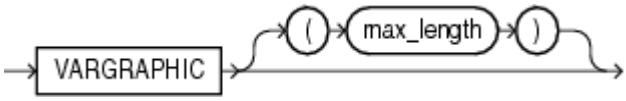
定義

VARGRAPHICデータはlengthサブフィールドとダブルバイト文字の文字列で構成されます。ダブルバイト文字セットは、Oracle Databaseではサポートされていないため、SQL*Loaderを使用して1バイトとして読み込んで、RAWデータとしてロードします。RAWデータ型と同様に、VARGRAPHICフィールドは変更されずに指定の列に格納されます。

> ノート:
>
> length サブフィールドのサイズには、システム上の SQL*Loader の SMALLINT データ型の長さ(C の SHORT INT 型に相当する長さ)が取られます。

構文

VARGRAPHICデータ型の構文は次のとおりです。



使用上のノート

VARGRAPHICデータは、SHORT INTの長さが同じバイト数のシステム間のみで、正しい結果でロードできます。バイト順序がシステム間で異なる場合は、適切な方法でlengthサブフィールドのバイト順序を指定します。

現行のフィールドの長さは、先頭の2バイトで示されます。VARGRAPHICデータ型に指定する最大長には、lengthサブフィールドの長さは含まれません。この最大長には、グラフィック(ダブルバイト)文字の文字数を指定します。フィールドのバイト単位の最大長を決定するには、このmaximum_lengthの値を2倍します。

フィールド最大長のデフォルトは、グラフィック文字で2KB、つまり4KB (2×2KB)です。必要なメモリーを最小限にするには、このような可変フィールドに対して、できるだけ最大長を指定します。

VARGRAPHIC文の前に(pos_specを使用して)位置指定が指定されている場合、グラフィック文字の1文字目ではなく、lengthサブフィールドの位置がわかります。pos_spec(start:end)を指定すると、endの位置によってそのフィールドの最大長が決まります。ここでのstartやendは、そのファイルにおける1バイト単位の文字位置を示します。したがって、(end+1)からstartの値を引くと、フィールドの実際のバイト長が求められます。最大長を指定した場合は、その最大長の方が、位置指定から計算された最大長より優先されます。

VARGRAPHICフィールドのフィールド長全体が、読み込まれる前に論理レコードの終わりで切り捨てられた場合、警告が出力されます。VARGRAPHIC型のフィールド長は、そのフィールドの各入力データ中に埋め込まれているため、そのフィールド長の方が正確であるとみなされます。

VARGRAPHICデータに対してはデリミタを使用できません。

関連トピック

- SQL*Loaderによるバイト順序の管理方法の理解

親トピック: 移植不能なデータ型

10.4.2.10 VARCHAR

SQL*Loaderの移植不能なLENGTH-VALUEデータ型VARCHARは、指定された長さの文字列が後に続くバイナリ長サブフィールドです。

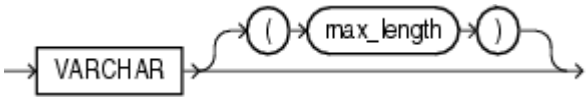
定義

VARCHARフィールドは、LENGTH-VALUEデータ型です。バイナリのlengthサブフィールドおよびその長さを持つ文字列で構成されます。データ・ファイルに文字長セマンティクスが使用されないかぎり、長さはバイト単位です。文字長セマンティクスが使用される場合は、文字単位になります。

> ノート:
>
> length サブフィールドのサイズには、システム上の SQL*Loader の SMALLINT データ型の長さ(C の SHORT INT 型に相当する長さ)が取られます。

構文

VARCHARデータ型の構文は次のとおりです。



使用上のノート

VARCHARフィールドは、SHORTデータ・フィールドのINTの長さが同じバイト数のシステム間でのみ、正しくロードできます。バイト順序がシステム間で異なる場合、またはVARCHARフィールドにUTF16文字セットのデータが含まれている場合は、適切な方法でlengthサブフィールドおよびデータのバイト順序を指定します。データのバイト順序は、UTF16文字セットに対してのみ問題となります。

制御ファイルに指定する最大長には、lengthサブフィールドのサイズを含むことはできません。VARCHARデータ型にオプションで最大長を指定すると、そのサイズ分のバッファがこのフィールドに対してバイト単位で割り当てられます。ただし、文字長セマンティクスがデータ・ファイルに使用される場合、バイト単位のバッファ・サイズは、文字セット内の最大限の文字のバイト単位のサイズのmax_length倍となります。

デフォルトの最大サイズは4KBです。データのロードに必要な最小限の値を最大値として指定することによって、SQL*Loaderで使用されるメモリーを最小限に抑えることができます。特に、VARCHARフィールドを多数使用する場合有効です。

POSITION句を使用する場合、指定する位置は、テキスト文字の先頭ではなく、lengthサブフィールドのバイト単位の位置になります。POSITION(start:end)と指定すると、endの位置によってそのフィールドの最大長が決まります。したがって、(end+1)からstartの値を引くと、フィールドの実際のバイト長が求められます。最大長を指定した場合は、その最大長の方がPOSITION句から計算された長さよりも優先されます。

VARCHARフィールドのフィールド長全体が、読み込まれる前に論理レコードの終わりで切り捨てられた場合、警告が出力されます。VARCHAR型のフィールド長は、そのフィールドの各入力データ中に埋め込まれているため、そのフィールド長の方が正確であるとみなされます。

VARCHARデータに対してはデリミタを使用できません。

関連トピック

- 文字長セマンティクス
- SQL*Loaderによるバイト順序の管理方法の理解

親トピック: 移植不能なデータ型

10.4.2.11 VARRAW

SQL*Loaderの移植不能なLENGTH-VALUEデータ型VARROWは、2バイトのバイナリのlengthサブフィールドとRAW文字

列のVALUEサブフィールドです。

定義

VARRAWは、2バイトのバイナリのlengthサブフィールドおよびその後に続くRAW文字列のVALUEサブフィールドで構成されています。

デフォルトでは、VARRAWは、lengthサブフィールドが2バイトで、最大サイズが4KBのVARRAWになります。VARRAW(65000)は、lengthサブフィールドが2バイトで、最大サイズが65000バイトのVARRAWになります。

使用上のノート

VARRAWフィールドは、適切な方法でlengthサブフィールドのバイト順序を指定すると、異なるバイト順序のシステム間でロードできます。

関連トピック

- SQL*Loaderによるバイト順序の管理方法の理解

親トピック: 移植不能なデータ型

10.4.2.12 LONG VARRAW

SQL*Loaderの移植不能なLENGTH-VALUEデータ型LONG VARRAWは、lengthサブフィールドが4バイトのVARRAWです。

定義

LONG VARRAWは、2バイトのlengthサブフィールドではなく、4バイトのlengthサブフィールドのあるVARRAWです。

デフォルトでは、LONG VARRAWは、lengthサブフィールドが4バイトで、最大サイズが4KBのVARRAWになります。LONG VARRAW(300000)は、lengthサブフィールドが4バイトで、最大サイズが300000バイトのVARRAWになります。

使用上のノート

LONG VARRAWフィールドは、適切な方法でlengthサブフィールドのバイト順序を指定すると、異なるバイト順序のシステム間でロードできます。

関連トピック

- SQL*Loaderによるバイト順序の管理方法の理解

親トピック: 移植不能なデータ型

10.4.3 移植可能なデータ型

このリファレンスを使用して、SQL*Loaderで移植可能なデータ型を使用する方法について理解してください。

移植可能なデータ型は、VALUEデータ型とLENGTH-VALUEデータ型にグループ化されます。移植可能なVALUEデータ型は、CHAR、日時および期間、GRAPHIC、GRAPHIC EXTERNAL、数値EXTERNAL (INTEGER、FLOAT、DECIMAL、ZONE)、RAWです。

移植可能なLENGTH-VALUEデータ型は、VARCHARCとVARRAWCです。

これらのデータ型の構文の詳細は、「datatype_spec」の構文図を参照してください。

- 移植可能なデータ型のカテゴリ
  移植可能なデータ型は、**VALUEデータ型**、**LENGTH-VALUEデータ型**および**文字データ型**にグループ化されます。

- CHAR
- 日時データ型および期間データ型
  SQL*Loaderでの、移植可能値の日時データ型(**datetime**)および期間データ型(interval)は、日付と時間間隔を記録するフィールドです。
- GRAPHIC
  SQL*Loaderの移植可能なVALUEデータ型GRAPHICには、ダブルバイト文字セット(DBCS)形式のデータが含まれます。
- GRAPHIC EXTERNAL
  SQL*Loaderの移植可能なVALUEデータ型GRAPHIC EXTERNALでは、外部表からロードされた図形データを指定します。
- 数値型EXTERNAL
  SQL*Loaderの移植可能なVALUE数値型EXTERNALデータ型は、判読可能な文字形式のデータです。
- RAW
  SQL*Loaderの移植可能なVALUEデータ型RAWでは、RAWバイナリ・データのロードを指定します。
- VARCHARC
  移植可能なLENGTH-VALUEデータ型VARCHARCでは、文字列の長さとサイズを指定します。
- VARRAWC
  移植可能なLENGTH-VALUEデータ型VARRAWCは、RAW文字列値のサブフィールドで構成されます。
- システム固有のデータ型フィールド長の競合
  各種のデータ型をロードする場合は、フィールド長の競合を管理するためにSQL*Loaderが従うルールを学習します。
- LENGTH-VALUEデータ型のフィールド長
  SQL*Loaderの移植可能なLENGTH-VALUEデータ型(VARCHAR、VARCHARC、VARGRAPHIC、VARRAW、VARRAWCなど)のフィールド長は、バイト単位または文字単位になります。

親トピック: SQL*Loaderのデータ型

10.4.3.1 移植不能なデータ型のカテゴリ

移植可能なデータ型は、**VALUEデータ型**、**LENGTH-VALUEデータ型**および**文字データ型**にグループ化されます。

移植可能なVALUEデータ型は次のとおりです。

- CHAR
- 日時データ型および期間データ型
- GRAPHIC
- GRAPHIC EXTERNAL
- 数値型EXTERNAL (INTEGER, FLOAT, DECIMAL, ZONE)
- RAW

移植可能なLENGTH-VALUEデータ型は次のとおりです。

- VARCHARC
- VARRAWC

文字データ型は、次のとおりです。

- CHAR
- DATE
- 数値型EXTERNAL

これらのフィールドにはデリミタを使用できます。また、制御ファイルでフィールド長(または最大長)を指定できます。

移植不能なデータ型について詳しく理解するために、datatype_specの構文図を参照してください。

関連トピック

- SQL*Loaderの構文図

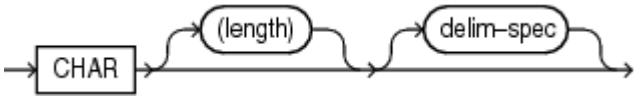
親トピック: 移植可能なデータ型

10.4.3.2 CHAR

このデータ・フィールドには、文字データが入ります。データ長には最大長を指定します(オプション)。長さに関して、次のことに注意してください。

- データ長が指定されていない場合、データ長はPOSITION句の指定から導出されます。
- データ長が指定されている場合は、POSITION句で指定したデータ長より優先されます。
- データ長もPOSITION句による位置も指定されていない場合、フィールドが区切られていないかぎり、CHARのデータ長は1文字とみなされます。
  - デリミタ付きCHARフィールドでは、長さが指定されている場合、その長さが最大長として使用されます。
  - 長さが指定されていないデリミタ付きCHARフィールドでは、デフォルトの255バイトが使用されます。
  - デリミタ付きで255バイトを超えるCHARフィールドには、最大長を指定する必要があります。指定しない場合は、データ・ファイルのフィールドが最大長を超えているというエラーを受信します。

CHARデータ型の構文は次のとおりです。



関連項目:

デリミタの指定

親トピック: 移植可能なデータ型

10.4.3.3 日時データ型および期間データ型

SQL*Loaderでの、移植可能値の日時データ型(**datetime**)および期間データ型(interval)は、日付と時間間隔を記録するフィールドです。

- 日時データ型と期間データ型のカテゴリ
  SQL*Loaderでは、移植可能値の日時データ型には日付フィールドおよび時間フィールドが記録され、期間データ型には時間間隔が記録されます。
- DATE
  SQL*Loaderの日時データ型DATEフィールドには、特定の日付を定義する文字データを格納します。
- TIME
  SQL*Loader日時データ型TIMEには、時、分および秒の値を格納します。
- TIME WITH TIME ZONE
  SQL*Loaderの日時データ型TIME WITH TIME ZONEは、その値にタイムゾーンの時差が含まれるTIMEの変形です。
- TIMESTAMP

SQL*Loaderの日時データ型TIMESTAMPは、DATEデータ型の拡張機能です。

- TIMESTAMP WITH TIME ZONE
  SQL*Loaderの日時データ型TIMESTAMP WITH TIME ZONEは、その値にタイムゾーンの時差が含まれるTIMESTAMPの変形です。
- TIMESTAMP WITH LOCAL TIME ZONE
  SQL*Loaderの日時データ型TIMESTAMP WITH LOCAL TIME ZONEは、その値にタイムゾーンの時差が含まれるTIMESTAMPデータ型の変形です。
- INTERVAL YEAR TO MONTH
  SQL*Loaderの期間データ型INTERVAL YEAR TO MONTHは、期間を格納します。
- INTERVAL DAY TO SECOND
  SQL*Loaderの間隔データ型INTERVAL DAY TO SECONDは、DAYおよびSECOND日時フィールドを使用して、一定の期間を格納します。

親トピック: 移植可能なデータ型

10.4.3.3.1 日時および間隔のデータ型のカテゴリ

SQL*Loaderでは、移植可能値の日時データ型には日付フィールドおよび時間フィールドが記録され、期間データ型には時間間隔が記録されます。

定義

DatatimesとIntervalsは、どちらも複数のフィールドで構成されます。これらのフィールドの値は、データ型の値によって決まります。

日時データ型は、次のとおりです。

- DATE
- TIME
- TIMEWITHTIMEZONE
- TIMESTAMP
- TIMESTAMPWITHTIMEZONE
- TIMESTAMPWITHLOCALTIMEZONE

期間データ型は次のとおりです。

- INTERVALYEARTOMONTH
- INTERVALDAYTOSECOND

使用上のノート

日時データ型の値を**Datetimes**と呼ぶこともあります。DATEの場合を除き、fractional_second_precisionには必要に応じて値を指定できます。fractional_second_precisionには、SECOND日時フィールドの小数部分に格納する桁数を指定します。このデータ型の列を作成する場合、値は0から9までの範囲内の数に設定できます。デフォルトは6です。

期間データ型の値は、「**期間**」とも呼ばれます。INTERVAL YEAR TO MONTHデータ型では、オプションでyear_precisionの値を指定できます。year_precisionには、YEAR日時フィールドの桁数を指定します。デフォルト値は2です。

INTERVAL DAY TO SECONDデータ型では、オプションでday_precisionおよびfractional_second_precisionの値を指定できます。day_precisionには、DAY日時フィールドの桁数を指定します。0から9までの値を使用できます。デフォルトは2です。fractional_second_precisionには、SECOND日時フィールドの小数部分に格納する桁数を指定します。このデータ型の列を作成する場合、値は0から9までの範囲内の数に設定できます。デフォルトは6です。

関連トピック

- 表レベルでの日時書式の指定
- 数値の優先順位

親トピック: 日時データ型および期間データ型

10.4.3.3.2 DATE

SQL*Loaderの日時データ型DATEフィールドには、特定の日付を定義する文字データを格納します。

構文

使用上のノート

DATEフィールドには文字データが入り、その文字データは、指定された日付マスクを使用してOracleの日付に変換されます。

可変長の日付マスクを指定していない場合、データ長の指定はオプションになります。データ・ファイルに文字長セマンティクスが使用されないかぎり、長さはバイト単位です。文字長セマンティクスが使用される場合は、文字単位になります。

長さを明示的に指定していない場合は、POSITION句から長さが求められます。マスクを使用するときは、データ長がマスクの長さ以下であることが確実でないかぎり、常に長さを指定することをお薦めします。

長さを明示的に指定した場合、この長さは、POSITION句で指定された長さよりも優先されます。これらの指定は、マスクから求められる長さをオーバーライドします。マスクについては、Oracle日付マスクとして有効なものを指定します。マスクの指定を省略すると、デフォルトのOracle日付マスク「dd-mon-yy」が使用されます。

データ長は小カッコで囲み、マスクは引用符で囲む必要があります。

また、デリミタを使用してDATEデータ型のフィールドを指定することもできます。

例

```
LOAD DATA
INTO TABLE dates (col_a POSITION (1:15) DATE "DD-Mon-YYYY")
BEGINDATA
1-Jan-2012
1-Apr-2012 28-Feb-2012
```

デリミタが見つかるまで、空白は無視され左から右に日付が解析されます。(DATEフィールドのデータがすべて空白の場合、NULLフィールドとしてロードされます)。

前述の例では、日付マスク"DD-Mon-YYYY"は、バイト長セマンティクスで11バイトあります。そのため、SQL*Loaderによってこのフィールドの最大文字数が11文字とみなされ、前述の指定は正しく処理されます。ただし、次のように指定される場合は注意が必要です。

```
DATE "Month dd, YYYY"
```

この場合、日付マスクは14バイトです。September 30, 2012などのように14バイトを超える長さの値が指定されている場合は、長さを指定する必要があります。

同様に、ユリウス日(日付マスク「J」)の場合も長さを指定する必要があります。日付文字列の長さがマスクの長さ(マスク内のバイト数)を超える可能性がある場合は、必ずフィールド長を指定してください。

関連トピック

- 文字長セマンティクス
- デリミタの指定

親トピック: 日時データ型および期間データ型

10.4.3.3.3 TIME

SQL*Loader日時データ型TIMEには、時、分および秒の値を格納します。

構文

```
TIME [(fractional_second_precision)]
```

親トピック: 日時データ型および期間データ型

10.4.3.3.4 TIMEWITHTIMEZONE

SQL*Loaderの日時データ型TIME WITH TIME ZONEは、その値にタイムゾーンの時差が含まれるTIMEの変形です。

定義

タイムゾーンの置換えは、ローカル時刻とUTCの差(時および分)です。

構文

```
TIME [(fractional_second_precision)] WITH [LOCAL] TIME ZONE
```

LOCALオプションが指定されている場合、データベースに格納されているデータはデータベース・タイムゾーンに指定されます。データが取得されると、ユーザーのローカル・セッション・タイムゾーンに返されます。

親トピック: 日時データ型および期間データ型

10.4.3.3.5 TIMESTAMP

SQL*Loaderの日時データ型TIMESTAMPは、DATEデータ型の拡張機能です。

定義

DATEデータ型の年、月および日に加えて、TIMEデータ型の時、分および秒の値を格納します。

構文

```
TIMESTAMP [(fractional_second_precision)]
```

日付値を指定する場合、時間構成要素を指定しないと、デフォルト時間の12:00:00 AM(真夜中)が採用されます。

親トピック: 日時データ型および期間データ型

10.4.3.3.6 TIMESTAMPWITHTIMEZONE

SQL*Loaderの日時データ型TIMESTAMP WITH TIME ZONEは、その値にタイムゾーンの時差が含まれるTIMESTAMPの変形です。

定義

タイムゾーンの置換えは、ローカル時刻とUTCの差(時および分)です。

構文

```
TIMESTAMP [(fractional_second_precision)] WITH TIME ZONE
```

親トピック: 日時データ型および期間データ型

10.4.3.3.7 TIMESTAMPWITHLOCALTIMEZONE

SQL*Loaderの日時データ型TIMESTAMP WITH LOCAL TIME ZONEは、その値にタイムゾーンの時差が含まれるTIMESTAMPデータ型の変形です。

定義

データベースに格納されているデータは、データベース・タイムゾーンに指定されます。データが取得されると、ユーザーのローカル・セッション・タイムゾーンに返されます。

構文

次のように指定します。

```
TIMESTAMP [(fractional_second_precision)] WITH LOCAL TIME ZONE
```

親トピック: 日時データ型および期間データ型

10.4.3.3.8 INTERVALYEARTOMONTH

SQL*Loaderの期間データ型INTERVAL YEAR TO MONTHは、期間を格納します。

定義

INTERVAL YEAR TO MONTHは、YEARおよびMONTH日時フィールドを使用して期間を格納します。

構文

```
INTERVAL YEAR [(year_precision)] TO MONTH
```

親トピック: 日時データ型および期間データ型

10.4.3.3.9 INTERVALDAYTOSECOND

SQL*Loaderの間隔データ型INTERVAL DAY TO SECONDは、DAYおよびSECOND日時フィールドを使用して、一定の期間を格納します。

定義

INTERVAL DAY TO SECONDデータ型は、DAYおよびSECOND日時フィールドを使用して一定期間を格納します。

構文

```
INTERVAL DAY [(day_precision)] TO SECOND [(fractional_second_precision)]
```

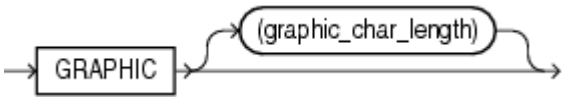
親トピック: 日時データ型および期間データ型

10.4.3.4 GRAPHIC

SQL*Loaderの移植可能なVALUEデータ型GRAPHICには、ダブルバイト文字セット(DBCS)形式のデータが含まれます。

定義

GRAPHICデータ型では、図形データを指定します。



使用上のノート

GRAPHICのデータは、ダブルバイト文字セット(DBCS)形式の文字列データです。ダブルバイト文字セットはOracle

Databaseではサポートされていないため、SQL*Loaderを使用して1バイトとして読み込みます。RAWデータ型と同様に、GRAPHICフィールドは変更されずに指定の列に格納されます。

GRAPHIC型およびGRAPHIC EXTERNAL型では、POSITION (start:end)を指定すると、論理レコードにおけるフィールドの正確な位置が決まります。

ただし、GRAPHIC (EXTERNAL)データ型にデータ長を指定する場合は、ダブルバイト・グラフィック文字の文字数を指定します。この値を2倍してフィールドのバイト長が求められます。グラフィック文字の長さを指定した場合は、POSITION句から求められたデータ長は無視されます。GRAPHICデータ型の指定では、データ・フィールドの区切りの指定はできません。

親トピック: 移植可能なデータ型

10.4.3.5 GRAPHIC EXTERNAL

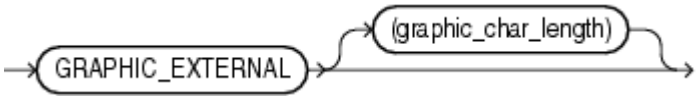
SQL*Loaderの移植可能なVALUEデータ型GRAPHIC EXTERNALでは、外部表からロードされた図形データを指定します。

説明

GRAPHICは、ダブルバイト文字(DBCA)のデータであることを示します。EXTERNALは、先頭と最後の文字が無視されることを示します。

DBCSフィールドがシフトイン/シフトアウト文字で囲まれている場合は、GRAPHIC EXTERNAL型を使用します。このデータ型は、データの先頭と最後の文字(シフトイン/シフトアウト文字)がロードされないことを除き、GRAPHIC型とほぼ同じです。

構文



GRAPHICは、ダブルバイト文字のデータであることを示します。EXTERNALは、先頭と最後の文字が無視されることを示します。graphic_char_length値はDBCSでの長さを指定します。

例

GRAPHIC EXTERNALの動作を示すために、[ ]がシフトイン/シフトアウト文字を表し、#がダブルバイト文字を表しているとします。

####を記述するには、POSITION(1:4) GRAPHICまたはPOSITION(1) GRAPHIC(2)を使用します。

[####]を記述するには、POSITION(1:6) GRAPHIC EXTERNALまたはPOSITION(1) GRAPHIC EXTERNAL(2)を使用します。

関連トピック

- GRAPHIC

親トピック: 移植可能なデータ型

10.4.3.6 数値型EXTERNAL

SQL*Loaderの移植可能なVALUE数値型EXTERNALデータ型は、判読可能な文字形式のデータです。

定義

数値型EXTERNALデータ型は、オプション指定の長さおよびデリミタ付きの、EXTERNALとして指定された数値データ型(INTEGER、 FLOAT、DECIMALおよびZONED)です。データ・ファイルに文字長セマンティクスが使用されないかぎり、長さはバイト単位です。文字長セマンティクスが使用される場合は、文字単位になります。

このデータ型は、判読可能な文字形式の数値データです。長さ、位置およびデリミタについては、CHARデータと同じ規則が数値型EXTERNALにも適用されます。これらの規則の詳細は、「CHAR」を参照してください。

数値型EXTERNALデータ型の構文は、datatype_spec SQL*Loaderデータ構文の一部として示されています。

FLOAT EXTERNALデータは科学表記法または通常表記法のどちらででも指定できます。「5.33」と「533E-2」は両方とも同じ値の正しい表現です。

> ノート:
>
> このデータは、バイナリ表現ではなく、文字形式の数字になります。したがって、これらのデータ型の処理方法は、DEFAULTIF を使用する場合を除き、CHAR と同じになります。デフォルトを NULL にする場合は CHAR を使用します。デフォルトを 0(ゼロ)にする場合は EXTERNAL を使用します。

関連トピック

- WHEN、NULLIFおよびDEFAULTIF句の使用
- 文字長セマンティクス
- CHAR
- SQL*Loaderの構文図

親トピック: 移植可能なデータ型

10.4.3.7 RAW

SQL*Loaderの移植可能なVALUEデータ型RAWでは、RAWバイナリ・データのロードを指定します。

説明

RAW型のバイナリ・データが無条件でRAW型のデータベース列にロードされた場合、Oracle Databaseファイルに配置されても変換されません。

データをCHAR列にロードした場合は、Oracle Databaseによって16進数に変換されます。DATE型や数値型の列にはロードできません。

構文

このフィールドのlengthは、制御ファイルに指定されたバイト数です。この長さは、データベース中のターゲット列の長さとメモリー・リソースによってのみ制限されます。データ・ファイルに文字長セマンティクスが使用される場合でも、長さは常にバイト単位です。RAWデータ・フィールドに対してはデリミタは使用できません。

親トピック: 移植可能なデータ型

10.4.3.8 VARCHARC

移植可能なLENGTH-VALUEデータ型VARCHARCでは、文字列の長さとサイズを指定します

説明

SQL*LoaderのVARCHARCデータ型は、文字のlengthサブフィールドとそれに続く文字列値サブフィールドで構成されます。

構文

```
VARCHARC(character_length,character_string)
```

使用上のノート

VARCHARCに対する宣言には、lengthサブフィールドの長さが含まれます。また、オプションで文字列の最大サイズがその後に続く場合もあります。データ・ファイルにバイト長セマンティクスが使用される場合、長さおよび最大サイズはともにバイト単位です。データ・ファイルに文字長セマンティクスが使用される場合、長さおよび最大サイズはともに文字単位です。最大サイズが指定されていない場合、バイト長セマンティクスまたは文字長セマンティクスのいずれが使用されていても、デフォルトは4KBです。

たとえば:

- 少なくとも長さサブフィールドに値を指定する必要があるため、VARCHARCはエラーになります。
- データ・ファイルにバイト長セマンティクスが使用されている場合、VARCHARC(7)は、lengthサブフィールドが7バイトで、最大サイズが4KB(デフォルト)のVARCHARCになります。文字長セマンティクスが使用されている場合は、lengthサブフィールドが7文字で、最大サイズが4KB(デフォルト)のVARCHARCになります。最大サイズが指定されていない場合、バイト長セマンティクスまたは文字長セマンティクスのいずれが使用されていても、常にデフォルトの4KBが使用されることに注意してください。
- データ・ファイルにバイト長セマンティクスが使用されている場合、VARCHARC(3,500)は、lengthサブフィールドが3バイトで、最大サイズが500バイトのVARCHARCになります。文字長セマンティクスが使用されている場合は、lengthサブフィールドが3文字で、最大サイズが500文字のVARCHARCになります。

例

```
CREATE TABLE emp_load
             (first_name CHAR(15),
              last_name CHAR(20),
              resume CHAR(2000),
              picture RAW (2000))
  ORGANIZATION EXTERNAL
  (TYPE ORACLE_LOADER
    DEFAULT DIRECTORY ext_tab_dir
    ACCESS PARAMETERS
      (FIELDS (first_name VARCHARC(5,12),
               last_name VARCHARC(2,20),
               resume VARCHARC(4,10000),
               picture VARRAWC(4,100000)))
  LOCATION ('info.dat'));
00007William05Ricca0035Resume for William Ricca is missing0000
```

関連トピック

- VARCHARCおよびVARRAWC
- 文字長セマンティクス

親トピック: 移植可能なデータ型

10.4.3.9 VARRAWC

移植可能なLENGTH-VALUEデータ型VARRAWCは、RAW文字列値のサブフィールドで構成されます。

説明

VARRAWCデータ型には、バイナリ・データが後に続く文字カウント・フィールドが含まれます。

構文

```
VARRAWC(character_length,binary_data)
```

使用上のノート

- VARRAWCはエラーになります。
- VARRAWC(7)は、lengthサブフィールドが7バイトで、最大サイズが4KB(デフォルト)のVARRAWCになります。
- VARRAWC(3,500)は、lengthサブフィールドが3バイトで、最大サイズが500バイトのVARRAWCになります。

例

次にVARRAWCの例を示します。pictureフィールドの長さは0です。これは、このフィールドがNULLに設定されていることを意味します。

```
CREATE TABLE emp_load
             (first_name CHAR(15),
              last_name CHAR(20),
              resume CHAR(2000),
              picture RAW (2000))
  ORGANIZATION EXTERNAL
  (TYPE ORACLE_LOADER
    DEFAULT DIRECTORY ext_tab_dir
    ACCESS PARAMETERS
      (FIELDS (first_name VARCHARC(5,12),
               last_name VARCHARC(2,20),
               resume VARCHARC(4,10000),
               picture VARRAWC(4,100000)))
  LOCATION ('info.dat'));
00007William05Ricca0035Resume for William Ricca is missing0000
```

関連トピック

- VARCHARCおよびVARRAWC

親トピック: 移植可能なデータ型

10.4.3.10 システム固有のデータ型フィールド長の競合

各種のデータ型をロードする場合は、フィールド長の競合を管理するためにSQL*Loaderが従うルールを学習します。

フィールド長を指定する方法は数通りあります。それぞれの指定方法で異なる値を指定して、値が競合する場合は、そのうちの1つの値が優先されます。競合が発生した時点で警告が出されます。どのフィールド長を採用するかは、次の規則に基づいて決定されます。

1. POSITION句で指定されたバイト数に関係なく、SMALLINT、FLOATおよびDOUBLEのデータ・サイズは固定長です。
2. DECIMAL、INTEGER、ZONED、GRAPHIC、GRAPHIC EXTERNALまたはRAWで指定されたフィールド長(または精度)が、POSITION(start:end)から計算されたサイズと異なる場合は、指定されたフィールド長(または精度)を採用します。
3. 文字フィールドまたはVARGRAPHICフィールドにおいて指定された最大長が、POSITION(start:end)から計算されたフィールド長と異なる場合は、指定された最大長を採用します。

たとえば、システム固有のデータ型INTEGERが4バイトであるときに、次のようなフィールドが指定されたとします。

```
column1 POSITION(1:6) INTEGER
```

この場合、警告が出力され、正しいフィールド長である4バイトが採用されます。実際に使用されたフィールド長は、ログ・ファイル内の列表の「Len」という見出しの箇所に記録されます。

```
Column Name              Position   Len  Term Encl Data Type
------------------------ --------- ----- ---- ---- ---------
```

```
COLUMN1                         1:6      4              INTEGER
```

親トピック: 移植可能なデータ型

10.4.3.11 LENGTH-VALUEデータ型のフィールド長

SQL*Loaderの移植可能なLENGTH-VALUEデータ型(VARCHAR、VARCHARC、VARGRAPHIC、VARRAW、VARRAWCなど)のフィールド長は、バイト単位または文字単位になります。

制御ファイルで、LENGTH-VALUEデータ型の最大長(VARCHAR、VARCHARC、VARGRAPHIC、VARRAWおよびVARRAWC)を指定できます。指定される最大長は、フィールドにバイト長セマンティクスが使用される場合はバイト単位で、文字長セマンティクスが使用される場合は文字単位です。最大長が指定されていない場合は、デフォルトで4096バイトとなります。フィールド長が最大長を超える場合、レコードは拒否され、次のエラーが表示されます。

```
Variable length field exceed maximum length
```

親トピック: 移植可能なデータ型

10.4.4 データ型の変換

SQL*Loaderは、ほとんどのデータ型の変換を自動で実行しますが、エラーを防止するために変換ルールについて理解しておく必要があります。

制御ファイルに指定したデータ型で、データ・ファイル中のデータをどのように解釈するかを、SQL*Loaderに対して指定します。一方、サーバーでは、これとは別にデータベース中の列に対してデータ型を定義します。これらの2つのデータ型を対応付ける手がかりとなるのが、制御ファイル中に指定している列名です。

SQL*Loaderでは、入力ファイル中のフィールドからデータが抽出されます。抽出処理は、制御ファイルに指定したデータ型に基づいて行われます。次にSQL*Loaderでは、そのフィールドがサーバーに送信されます。送信されたフィールドは、該当する列に(行挿入配列の一部として)格納されます。

SQL*Loaderまたはサーバーでは、変換の必要なデータに対してデータ変換が行われ、適切な内部形式でデータが格納されます。これには、データ・ファイルの文字セットとデータベースの文字セットが異なる場合に、データ・ファイルのデータをデータベース用に変換する機能も含まれます。

> ノート:
>
> SQL*Loader の従来型パスを使用して、データ・ファイルの文字データを LONG RAW 列にロードする場合、その文字データは HEX 文字列として解釈されます。SQL は、HEX 文字列をバイナリ表現に変換します。ここで、4000 バイトより長い文字列は、SQL HEXTORAW 変換演算子のバイト制限を超えていることに注意してください。文字列の長さがバイト制限を超えている場合は、エラーが返されます。SQL*Loader はエラーのある行を拒否して、ロードを続行します。

ファイルのデータのデータ型は、Oracle Database表の列のデータ型と同じである必要はありません。Oracle Databaseによって、自動的に変換が実行されます。ただし、その変換が適切であり、エラーを生成していないことを確認する必要があります。たとえば、CHARデータ型のデータ・ファイル・フィールドをNUMBERデータ型のデータベース列にロードする場合、文字フィールドの内容が有効な数値を表していることを確認する必要があります。

ノート:

SQL*Loader には、NUMBER または VARCHAR2 などの Oracle 内部データ型に関するデータ型指定は定義されていません。SQL*Loader のデータ型として扱えるのはテキスト・エディタで作成できるデータ(文字データ型)、および標準プログラミング言語で作成できるデータ(システム固有のデータ型)のみです。ただし、NUMBER および VARCHAR2 のようなデータ型は、SQL*Loader では認識されませんが、これらのデータ型またはその他のデータ型のデータベース列に、Oracle Database で変換可能なデータをロードすることはできます。

親トピック: SQL*Loaderのデータ型

10.4.5 日時データ型および期間データ型のデータ型変換

Oracle Databaseデータ型と、SQL*Loader制御ファイルの日時および期間データ型の間で変換がサポートされるかどうかを学習します。

日時データ型および期間データ型のデータ型変換の読み方

この表で使用されているOracle Databaseデータ型の略称は次のとおりです。

- N = NUMBER
- C = CHARまたはVARCHAR2
- D = DATE
- T = TIMEおよびTIME WITH TIME ZONE
- TS = TIMESTAMPおよびTIMESTAMP WITH TIME ZONE
- YM = INTERVAL YEAR TO MONTH
- DS = INTERVAL DAY TO SECOND

SQL*Loaderデータ型でも、D、T、TS、YMおよびDSに関しては、表の略称の定義は同じです。SQL*Loaderには、NUMBER、CHAR、VARCHAR2など、Oracle Databaseの内部データ型に関するデータ型指定は定義されていません。ただし、Oracleデータベースが変換できるデータは、これらまたは他のデータベース列にロードできます。

この表の読み方の例を、SQL*Loaderデータ型DATE (略称D)の行で示します。行を右に読み進めると、データ型変換がサポートされるOracle Databaseデータ型は、CHAR、VARCHAR2、DATE、TIMESTAMPおよびTIMESTAMP WITH TIME ZONEデータ型であることがわかります。ただし、Oracle Databaseデータ型のNUMBER、TIME、TIME WITH TIME ZONE、INTERVAL YEAR TO MONTHまたはINTERVAL DAY TO SECONDデータ型の変換はサポートされません。

表10-2 日時データ型および期間データ型のデータ型変換

SQL*Loaderデータ型	Oracle Databaseデータ型(変換のサポート)
N	N(可)、C(可)、D(不可)、T(不可)、TS(不可)、YM(不可)、DS(不可)
C	N(可)、C(可)、D(可)、T(可)、TS(可)、YM(可)、DS(可)
D	N(不可)、C(可)、D(可)、T(不可)、TS(可)、YM(不可)、DS(不可)
T	N(不可)、C(可)、D(不可)、T(可)、TS(可)、YM(不可)、DS(不可)

SQL*Loaderデータ型	Oracle Databaseデータ型(変換のサポート)
TS	N(不可)、C(可)、D(可)、T(可)、TS(可)、YM(不可)、DS(不可)
YM	N(不可)、C(可)、D(不可)、T(不可)、TS(不可)、YM(可)、DS(不可)
DS	N(不可)、C(可)、D(不可)、T(不可)、TS(不可)、YM(不可)、DS(可)

親トピック: SQL*Loaderのデータ型

10.4.6 デリミタの指定

CHAR、日時、期間または数値型EXTERNAL型のフィールドの境界は、デリミタ文字を使用して、入力データ・レコード中に指定することもできます。

デリミタ文字は、TERMINATED BY、ENCLOSED BYおよびOPTIONALLY ENCLOSED BY句を様々に組み合せて指定できます(ただし、TERMINATED BY句を使用する場合は、最初に指定する必要があります)。デリミタは、データ型の指定の後で指定します。

デリミタ句を様々に組み合せて使用した場合にデータがどのように処理されるかについては、「デリミタ付きデータの処理方法」を参照してください。

ノート:

RAW データ型でもデリミタを使用できます。ただし、RAW データ型が入力 LOBFILE にある場合、およびデリミタが TERMINATED BY EOF (ファイルの終わり)の場合のみです。

- 終了および囲みの指定に関する構文
  終了および囲みの指定に関する構文について説明します。
- データ中のデリミタ記号
  デリミタとして定義した句読点を、データの中でも使用する必要があります。
- デリミタ付きデータの最大長
  デリミタ付きフィールドでは、バインド配列に対して記憶域が大量に使用される場合があります。
- デリミタを使用した後続の空白のロード
  PRESERVE BLANKSを指定して末尾の空白をロードすることや、デリミタを使用してデータ・フィールドを宣言し、データ・ファイルにデリミタを追加することができます。

親トピック: SQL*Loaderのデータ型

10.4.6.1 終了および囲みの指定に関する構文

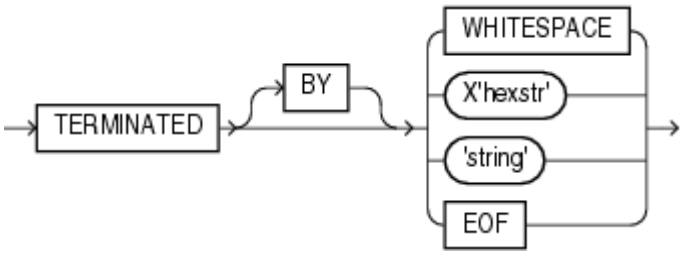
終了および囲みの指定に関する構文について説明します。

用途

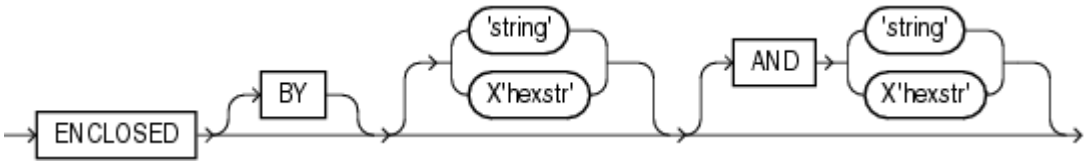
入力データ・レコードにデリミタ文字を指定します。

構文

次の図に、termination_specの構文を示します。



次の図に、enclosure_specの構文を示します。



次の表では、デリミタの指定に使用する、終了および囲みの指定に関する構文について説明します。

パラメータ

表10-3 デリミタの指定に使用するパラメータ

パラメータ	説明
TERMINATED	データは、最初にデリミタが現れるまで読み込まれます。
BY	可読性を向上させるために使用されるオプションのワードです。
WHITESPACE	デリミタには、スペース、タブ、空白、LF、改ページ、改行などの任意の空白文字を使用できます。(TERMINATED の場合のみ使用できます。ENCLOSED では使用できません。)
OPTIONALLY	指定する文字でデータを囲むこともできます。SQL*Loader では、この指定文字が最初に現れたところから、次に同じ文字が現れたところまでのデータ値が読み込まれます。データが囲まれていない場合は、終了デリミタ付きのフィールドとして読み込まれます。オプションで囲みデリミタを指定する場合は、必ず TERMINATED BY 句を指定してください。その場合、フィールド定義の一部としてローカルに指定しても、FIELDS 句の中でグローバルに指定してもかまいません。
ENCLOSED	データは、2 つのデリミタで囲まれます。
string	デリミタは文字列です。
X'hexstr'	デリミタは、文字コード体系における X'hexstr'によって指定される値を持つ文字で、X'1F'(10 進数の 31)などのように指定します。「X」は大文字でも小文字でも使用できます。
AND	後続の囲みデリミタを指定する場合に使用します。後続の囲みデリミタには、先頭の囲みデリミタとは異なる文字を指定できます。AND 句を指定しないと、先頭の囲みデリミタ

パラメータ	説明
	と後続の囲みデリミタは同じ文字とみなされます。
EOF	ファイル全体が LOB にロードされたことを示します。これは、LOB ファイルからデータがロードされる場合のみ有効です。EOF によって終了するフィールドは囲むことができません。

例

次に、終了と囲みに関する一連の例と、それらを説明するデータの例を示します。

```
TERMINATED BY ','                        a data string,
ENCLOSED BY '"'                          "a data string"
TERMINATED BY ',' ENCLOSED BY '"'        "a data string",
ENCLOSED BY '(' AND ')'                  (a data string)
```

親トピック: デリミタの指定

10.4.6.2 データ中のデリミタ記号

デリミタとして定義した句読点を、データの中でも使用する必要があります。

このような場合、デリミタ文字を2つ続けて記述すると、この文字は1文字のみ指定されたものと解釈され、データの一部として組み込まれます。たとえば、データベースに次の文字列を格納するとします。

```
(The delimiters are left parentheses, (, and right parentheses, )).)
```

フィールド指定は次のようにします。

```
ENCLOSED BY "(" AND ")"
```

この場合、データベースには次の文字列が格納されます。

```
The delimiters are left parentheses, (, and right parentheses, ).
```

このため、隣接するフィールドが同じデリミタを使用すると、問題が発生します。たとえば、次のように指定されている場合、

```
field1 TERMINATED BY "/"
field2 ENCLOSED by "/"
```

次のデータは正しく解釈されます。

```
This is the first string/      /This is the second string/
```

ただし、field1およびfield2が次のように隣接している場合、誤った処理が行われます。

```
This is the first string//This is the second string/
```

この場合、このデータ全体が、中央に1つの「/」のみを持つ単一の文字列とみなされ、field1に属するものと解釈されてしまいます。

親トピック: デリミタの指定

10.4.6.3 デリミタ付きデータの最大長

デリミタ付きフィールドでは、バインド配列に対して記憶域が大量に使用される場合があります。

デリミタ付きデータの最大長のデフォルトは、255バイトです。したがって、デリミタ付きフィールドでは、バインド配列に対して記憶域が大量に使用される場合があります。フィールドが255バイトより短い場合、最大長にはできるだけ小さい値を指定してください。フィールドが255バイトより長い場合は、フィールド長指定子またはPOSITION句を使用して、フィールドに最大長を指定する必要があります。

たとえば、文字列リテラルが255バイトより長い場合は、SUBSTR( )およびCHAR( )を使用して、フィールドのすべてのレコードの中で最も長い文字列を指定します。たとえば、field1のすべてのレコードの中で最も長い文字列が600バイトの場合は、次のようになります。

```
field1 CHAR(600) SUBSTR(:field, 1, 240)
```

親トピック: デリミタの指定

10.4.6.4 デリミタを使用した後続の空白のロード

PRESERVE BLANKSを指定して末尾の空白をロードすることや、デリミタを使用してデータ・フィールドを宣言し、データ・ファイルにデリミタを追加することができます。

デフォルトでは、制御ファイル内でPRESERVE BLANKSを指定しないかぎり、デリミタなしのデータ型の末尾の空白はロードされません。

データ・フィールドが9文字の長さで、値DANIELbbb (bbbは3つの空白)を含んでいる場合、デリミタなしでCHAR(9)として宣言されていると、Oracle Databaseには"DANIEL"としてロードされます。

デリミタを付けて末尾の空白を含めるには、このデータ・フィールドをCHAR(9) TERMINATED BY ':'として宣言し、データ・ファイルにコロンを追加してフィールドをDANIELbbb:とします。この変更の結果、フィールドは"DANIEL "としてロードされ、末尾の空白が含まれます。TERMINATED BY句を使用しないでPRESERVE BLANKSを指定すると、同じ結果が得られます。

関連トピック

- 空白の切捨て
- 空白の切捨てに対するPRESERVE BLANKSオプションの影響

親トピック: デリミタの指定

10.4.7 デリミタ付きデータの処理方法

デリミタは、TERMINATED BY、ENCLOSED BYおよび OPTIONALLY ENCLOSED BY句の様々な組合せをフィールド定義で使用して定義できます。

SQL*Loaderによるこれらのフィールド定義の各ケースの処理方法を理解するには、次の各トピックを参照してください。

- TERMINATED BYのみを使用するフィールド
  TERMINATED BYのみを使用するデータ・フィールドは、デリミタの位置に影響を受けます。
- TERMINATED BYなしでENCLOSED BYを使用するフィールド
  データ・フィールドでTERMINATED BYなしでENCLOSED BYを使用する場合は、SQL*Loaderでこれらのフィールドに使用される一連の処理があります。
- TERMINATED BYとともにENCLOSED BYを使用するフィールド
  データ・フィールドでTERMINATED BYとともにENCLOSED BYを使用する場合は、SQL*Loaderでこれらのフィールドに使用される一連の処理があります。
- TERMINATED BYとともにOPTIONALLY ENCLOSED BYを使用するフィールド
  データ・フィールドでTERMINATED BYとともにOPTIONALLY ENCLOSED BYを使用する場合は、SQL*Loader

でこれらのフィールドに使用される一連の処理があります。

親トピック: SQL*Loaderのデータ型

10.4.7.1 TERMINATED BYのみを使用するフィールド

TERMINATED BYのみを使用するデータ・フィールドは、デリミタの位置に影響を受けます。

フィールドに対してTERMINATED BYが指定され、ENCLOSED BYは指定されていない場合、フィールドの開始位置から最初のTERMINATED BYデリミタまでがフィールドのデータとして読み込まれます(デリミタ自体は読み込まれません)。終了デリミタがフィールドの最初の列位置にある場合、そのフィールドはNULLとなります。TERMINATED BYデリミタが検出される前にレコードの終わりが検出された場合は、レコードの末尾までのすべてのデータがフィールドの要素とみなされます。

TERMINATED BY WHITESPACEを指定すると、最初に空白文字(スペース、タブ、空白、LF、改ページまたは改行)が現れるまでデータが読み込まれます。空白文字が現れると、隣接する空白文字が検出されなくなるまで現在の位置が進められます。この処理の動作により、フィールド値がいくつかの空白で区切られます。

ただし、フィールドの最初の列の位置が判明しており、空白の終端文字がそこにある場合は、空白以外の終端文字の場合と異なり、フィールドはNULLとして扱われません。この処理によって、レコードが拒否される可能性や、正しくない列にフィールドがロードされる可能性があります。

親トピック: デリミタ付きデータの処理方法

10.4.7.2 TERMINATED BYなしでENCLOSED BYを使用するフィールド

データ・フィールドでTERMINATED BYなしでENCLOSED BYを使用する場合は、SQL*Loaderでこれらのフィールドに使用される一連の処理があります。

TERMINATED BY句を使用せずにENCLOSED BY句を使用するフィールドの場合は、次のステップが行われます。

1. フィールドの先頭に空白がある場合は、その空白がすべてスキップされます。
2. 最初に検出される空白以外の文字は、最初のENCLOSED BYデリミタに一致する文字列の先頭である必要があります。そうでない場合、行は拒否されます。
3. 最初のENCLOSED BYデリミタが検出された場合は、2番目のENCLOSED BYデリミタの検索が開始されます。
4. 2番目のENCLOSED BYデリミタが最初のデリミタと隣合せで検出された場合は、1つのデリミタが記述されていると解釈した上で、フィールドのデータ要素に含まれます。引き続き、2番目のENCLOSED BYデリミタが検索されます。
5. 2番目のENCLOSED BYデリミタが検出される前にレコードの終わりが検出された場合、その行は拒否されます。

親トピック: デリミタ付きデータの処理方法

10.4.7.3 TERMINATED BYとともにENCLOSED BYを使用するフィールド

データ・フィールドでTERMINATED BYとともにENCLOSED BYを使用する場合は、SQL*Loaderでこれらのフィールドに使用される一連の処理があります。

ENCLOSED BY句を使用し、TERMINATED BY句も使用するフィールドの場合は、次のステップが行われます。

1. フィールドの先頭に空白がある場合は、その空白がすべてスキップされます。
2. 最初に検出される空白以外の文字は、最初のENCLOSED BYデリミタに一致する文字列の先頭である必要があります。そうでない場合、行は拒否されます。
3. 最初のENCLOSED BYデリミタが検出された場合は、2番目のENCLOSED BYデリミタの検索が開始されます。
4. 2番目のENCLOSED BYデリミタが最初のデリミタと隣合せで検出された場合は、1つのデリミタが記述されていると解

釈した上で、フィールドのデータ要素に含まれます。引き続き、次にある2番目のENCLOSED BYデリミタが検索されます。

5. 2番目のENCLOSED BYデリミタが検出される前にレコードの終わりが検出された場合、その行は拒否されます。
6. 2番目のENCLOSED BYデリミタが検出された場合は、パーサーによりTERMINATED BYデリミタが検索されます。TERMINATED BYデリミタがWHITESPACE以外の場合、2番目のENCLOSED BYデリミタの最後とTERMINATED BYデリミタの間にある空白はスキップされます。

注意:

2 番目の ENCLOSED BY デリミタと TERMINATED BY デリミタの間に使用できるのは、WHITESPACE のみです。それ以外の文字があると、エラーが発生します。

7. TERMINATED BYデリミタが検出される前にレコードの終わりが検出された場合でも、行は拒否されません。

親トピック: デリミタ付きデータの処理方法

10.4.7.4 TERMINATED BYとともにOPTIONALLY ENCLOSED BYを使用するフィールド

データ・フィールドでTERMINATED BYとともにOPTIONALLY ENCLOSED BYを使用する場合は、SQL*Loaderでこれらのフィールドに使用される一連の処理があります。

OPTIONALLY ENCLOSED BY句およびTERMINATED BY句を使用するフィールドの場合は、次のステップが行われます。

1. フィールドの先頭に空白がある場合は、その空白がすべてスキップされます。
2. 最初に検出された空白以外の文字が、最初のOPTIONALLY ENCLOSED BYデリミタに一致する文字列の先頭であるかどうかがパーサーによって確認されます。そうでない場合に、OPTIONALLY ENCLOSED BYデリミタがデータ内に存在しないと、フィールドの現在の位置から最初のTERMINATED BYデリミタまでがフィールドのデータとして読み込まれます(デリミタ自体は読み込まれません)。TERMINATED BYデリミタが最初の列位置にある場合、そのフィールドはNULLとなります。TERMINATED BYデリミタが検出される前にレコードの終わりが検出された場合は、レコードの末尾までのすべてのデータがフィールドの要素とみなされます。
3. 最初のOPTIONALLY ENCLOSED BYデリミタが検出された場合は、2番目のOPTIONALLY ENCLOSED BYデリミタの検索が開始されます。
4. 2番目のOPTIONALLY ENCLOSED BYデリミタが2つ隣合せで検出された場合は、1つのデリミタが記述されていると解釈した上で、フィールドのデータ要素に含まれます。引き続き、2番目のOPTIONALLY ENCLOSED BYデリミタが検索されます。
5. 2番目のOPTIONALLY ENCLOSED BYデリミタが検出される前にレコードの終わりが検出された場合、その行は拒否されます。
6. OPTIONALLY ENCLOSED BYデリミタがデータ内に存在する場合は、パーサーによりTERMINATED BYデリミタが検索されます。TERMINATED BYデリミタがWHITESPACE以外の場合、2番目のOPTIONALLY ENCLOSED BYデリミタの最後とTERMINATED BYデリミタの間にある空白はスキップされます。
7. TERMINATED BYデリミタが検出される前にレコードの終わりが検出された場合でも、行は拒否されません。

注意:

空白文字を TERMINATED BY デリミタとして指定し、OPTIONALLY ENCLOSED BY も使用する場合は注意が必要です。SQL*Loader では、OPTIONALLY ENCLOSED BY デリミタを検索するときに、先頭の空白が取り除かれます。2 つの TERMINATED BY デリミタがレコードの途中に隣り合って含まれている場合(通常は、レコードのフィールドを NULL に設定する目的でこのように記述します)、最初の TERMINATED BY デリミタの空白を使用してフィールドが終了されますが、残りの空白は、次のフィールドの先頭の空白とみなされ、次のフィールドの TERMINATED BY デリミタとはみなされません。NULL 値をロードする場合は、データ内に ENCLOSED BY デリミタを含める必要があります。

親トピック: デリミタ付きデータの処理方法

10.4.8 文字データ型フィールド長の競合

CHAR型、DATE型および数値型EXTERNAL型の文字データ・フィールドの場合は、そのフィールド長を制御ファイルに複数指定できます。

複数指定したときの長さが異なり、値が競合する場合は、そのうちの1つが優先されます。競合が発生すると、警告が出力されます。 この項では、指定された長さのうちのどれが優先されるかについて説明します。

- 事前にサイズが決まっているフィールド
  サイズが決まっているフィールドでは、フィールド長は指定した値によって決まります。指定に競合がある場合は、そのフィールド長指定が使用されます。
- デリミタ付きフィールド
  デリミタ付きフィールドでは、フィールドの長さは、フィールド・セマンティクスおよび位置指定によって決まります。
- 日付フィールド・マスク
  DATEデータ型フィールドの長さは、マスクで指定された書式パターンによって異なりますが、位置指定または長さ指定で上書きできます。

親トピック: SQL*Loaderのデータ型

10.4.8.1 事前にサイズが決まっているフィールド

サイズが決まっているフィールドでは、フィールド長は指定した値によって決まります。指定に競合がある場合は、そのフィールド長指定が使用されます。

事前に決められているフィールドに対して、開始位置と終了位置を指定した場合、そのフィールド長は、データ型に対する指定によって決まります。データ型指定の中で長さを指定し、終了位置は指定しない場合、データ型指定の中で指定された長さがそのフィールド長になります。

開始位置、終了位置および長さがすべて指定されていて、その長さが異なる場合は、データ型指定の中で指定されている長さがフィールド長として使用されます。たとえば:

```
POSITION(1:10) CHAR(15)
```

この場合、このフィールド長は15になります。

親トピック: 文字データ型フィールド長の競合

10.4.8.2 デリミタ付きフィールド

デリミタ付きフィールドでは、フィールドの長さは、フィールド・セマンティクスおよび位置指定によって決まります。

デリミタ付きフィールドに長さを指定した場合、または開始位置と終了位置から長さを計算できる場合は、その長さがフィールドの**最大長**となります。指定される最大長は、フィールドにバイト長セマンティクスが使用される場合はバイト単位で、文字長セマンティクスが使用される場合は文字単位です。長さが指定されていない場合、または開始位置と終了位置から長さを計算できない場合、最大長のデフォルトは255バイトです。実際の長さはデリミタの位置によって変わりますが、長さの上限はこの最大長の値となります。

フィールドにデリミタのみでなく、開始位置および終了位置が指定されている場合は、位置指定のみが影響します。囲みデリミタまたは終了デリミタは無視されます。

デリミタが見つからない場合は、レコードの終わりがフィールドの終端となります。`TRAILING NULLCOLS`を指定した場合、SQL*Loaderでは、レコード内に存在しない、相対位置で指定した列はNULL列として処理されるため、残りのフィールドはNULLになります。デリミタまたはレコードの終端で区切った結果、フィールド長が最大長よりも大きくなる場合は、SQL*Loaderによってレコードが拒否され、エラーが返されます。

関連トピック

- TRAILING NULLCOLS句

親トピック: 文字データ型フィールド長の競合

10.4.8.3 日付フィールド・マスク

DATEデータ型フィールドの長さは、マスクで指定された書式パターンによって異なりますが、位置指定または長さ指定で上書きできます。

マスクを指定した場合、日付フィールド長は、使用するマスクによって異なります。指定されたマスクによって形式が決定され、SQL*Loaderではその形式に基づいてレコード中のデータが解釈されます。たとえば、次のようなマスクを指定したとします。

```
"Month dd, yyyy"
```

この場合、May 3, 2012はレコード(バイト長セマンティクスを含む)中で11バイトを占有し、January 31, 2012は16バイトを占有することになります。

ただし、開始位置および終了位置を指定すると、この位置指定から計算されるフィールド長は、マスクから求められるフィールド長よりも優先されます。`DATE(12)`のようにフィールド長が指定された場合は、このフィールド長が最優先となります。日付フィールドが、終了デリミタまたは囲みデリミタでも区切られている場合は、制御ファイル中で指定された長さがそのフィールドの最大長と解釈されます。

関連トピック

- 日時および間隔のデータ型のカテゴリ

親トピック: 文字データ型フィールド長の競合

10.5 フィールド条件の指定

フィールド条件とは、論理レコード中のフィールドに関して、それが真か偽かを評価する条件を記述したものです。

- フィールド条件の構文図
  フィールド条件の構文図を使用して、SQL*Loaderでのフィールド条件の使用方法を確認します。

- フィールドとBLANKSの比較
  BLANKSパラメータを使用すると、長さが不明なフィールドのデータが空白かどうかを知ることができます。
- フィールドとリテラルの比較
  リテラル文字列と比較されるデータ・フィールドには、文字列への空白のパディングを使用できます。

親トピック: SQL*Loaderフィールド・リスト・リファレンス

10.5.1 フィールド条件の構文図

フィールド条件の構文図を使用して、SQL*Loaderでのフィールド条件の使用方法を確認します。

用途

フィールド条件句は、WHEN句、NULLIF句およびDEFAULTIF句で使用します。

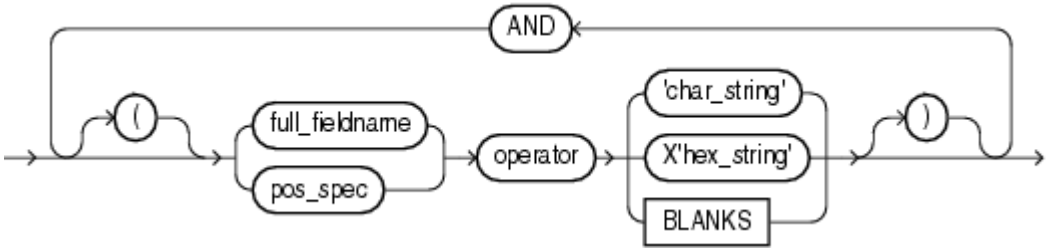
> ノート:
>
> 句の評価に使用するフィールドに NULL 値が含まれている場合、その句は必ず FALSE と評価されます。

フィールド条件は、CONTINUEIF句の中で指定する条件と同様ですが、次の2つの点で異なります。第1に、フィールド条件で指定する位置は、物理レコードではなく論理レコードの位置を示します。第2に、論理レコード内の位置またはデータ・ファイル内のフィールド名(FILLERフィールドを含む)のいずれかを指定できます。

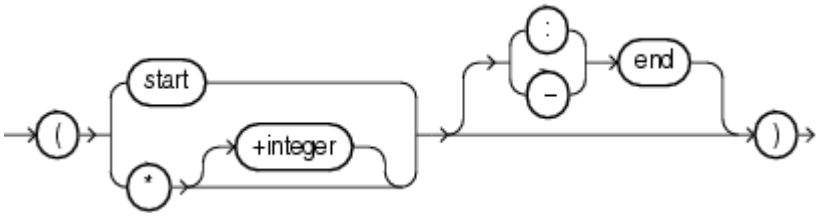
> ノート:
>
> フィールド条件は、セカンダリ・データ・ファイル(SDF)のフィールドに基づくことができません。

構文

field_condition句の構文は次のとおりです。



pos_spec句の構文は次のとおりです。



パラメータ

次の表では、フィールド条件句のパラメータについて説明します。

表10-4 フィールド条件句のパラメータ

パラメータ	説明
pos_spec	論理レコード中の比較対象フィールドの開始および終了位置です。それらの位置は小カッコで囲んでください。start-end と表記することも、start:end と表記することもできます。 開始位置の指定は、列番号、*(次の列)または*+n(次の列にオフセット分を加算)の形式で指定できます。 終了位置を省略した場合、フィールド長は、比較文字列の長さから判断されます。対象フィールドと比較文字列の長さが異なるときは、短い方を埋めるために文字列が追加されます。文字列の場合は空白が追加され、16 進数のバイト列の場合は 0(ゼロ)が追加されます。
start	論理レコード中の比較対象フィールドの開始位置です。
end	論理レコード中の比較対象フィールドの終了位置です。
full_fieldname	full_fieldname には、ドット表記法を使用してフィールドのフルネームを指定します。フィールド col2 が列オブジェクト col1 の属性の場合、指示句の中で col2 を参照するときは、col1.col2 と表記してください。同じエンティティを参照または命名している列名およびフィールド名があっても、列名には、エンティティのフルネームを指定できない(ドット表記法がない)ため、別のものとして認識されます。
operator	比較演算子として、等価または不等価を示す記号を指定します。
char_string	比較フィールドとの比較に使用する文字列で、一重引用符または二重引用符で囲んで指定します。比較の結果が真の場合は、現在のレコードが表に挿入されます。
X'hex_string'	16 進数の文字列で、16 進数 2 桁がフィールドの 1 バイトに相当します。一重引用符または二重引用符で囲まれます。比較の結果が真の場合は、現在のレコードが表に挿入されます。
BLANKS	フィールドが完全に空白かどうかをテストできます。BLANKS の指定は、デリミタ付きデータのロード時にフィールド長が予測できない場合、または空白が複数あるマルチバイト文字セットを使用する場合に必要となります。

親トピック: フィールド条件の指定

10.5.2 フィールドとBLANKSの比較

BLANKSパラメータを使用すると、長さが不明なフィールドのデータが空白かどうかを知ることができます。

次の指定を実行すると、空白のフィールドにNULL値を設定できます。

```
full_fieldname ... NULLIF column_name=BLANKS
```

BLANKSパラメータが認識できるのは空白のみです。タブは認識できません。これは、どのようなフィールド比較の場合でも、比較文字列のかわりに指定できます。列の値がすべて空白のときにのみ条件が真となります。

BLANKSは、固定長フィールドに対しても指定できます。その場合は、対象フィールドに合った長さの空白文字列を指定したのと同じことになります。たとえば、次の指定はどちらも同じことを意味します。

```
fixed_field CHAR(2) NULLIF fixed_field=BLANKS
fixed_field CHAR(2) NULLIF fixed_field="  "
```

マルチバイト文字セットには複数の空白が存在することもあります。このような文字セットには、空白文字列を指定するかわりにBLANKSを使用します。

文字列は特定の空白文字の組合せのみに一致しますが、BLANKSパラメータは様々な空白文字の組合せに一致します。マルチバイト文字セットの詳細は、「マルチバイト(アジア系言語)文字セット」を参照してください。

親トピック: フィールド条件の指定

10.5.3 フィールドと文字列の比較

リテラル文字列と比較されるデータ・フィールドには、文字列への空白のパディングを使用できます。

データ・フィールドが、それより短い比較文字列と比較された場合は、その文字列が埋め込まれます。文字データ型の文字列には、空白が埋め込まれます。たとえば:

```
NULLIF (1:4)=" "
```

この例では、位置1:4にあるデータが4つの空白と比較されます。位置1:4のデータが空白4つであれば、この句は真となります。

次の句のように、16進文字列には16進数のゼロが埋め込まれます。

```
NULLIF (1:4)=X'FF'
```

この句で、データ位置1:4を16進数のバイト列'FF000000'と比較します。

親トピック: フィールド条件の指定

10.6 WHEN、NULLIFおよびDEFAULTIF句の使用

SQL*Loaderでの、スカラー・フィールドを使用したWHEN、NULLIFおよびDEFAULTIF句の処理方法について学習します。

次の説明は、スカラー・フィールドに適用されます。非スカラー・フィールド(列オブジェクト、LOBおよびコレクション)はより複雑なため、WHEN、NULLIFおよびDEFAULTIF句は異なる方法で処理されます。

WHEN、NULLIFまたはDEFAULTIF句は、フィールド名を指定するか、フィールド位置を指定するかによって、結果が異なります。

- WHEN、NULLIFまたはDEFAULTIF句でフィールド名を指定すると、これらの句は、SQL*Loaderによってフィールドの評価値と比較されます。評価値には、切り捨てられた空白文字が考慮されます。空白とスペースのトリミングについては、次を参照してください。

  空白の切捨て

- WHEN、NULLIFまたはDEFAULTIF句で位置を指定すると、これらの句は、SQL*Loaderによってデータ・ファイル内の元の論理レコードと比較されます。この場合、論理レコードでは空白の切捨ては行われません。

フィールドに切り捨てられた空白がある場合、あるいはWHEN、NULLIFまたはDEFAULTIF句に空白およびタブが含まれているか、BLANKSパラメータが使用されている場合は、異なる結果が得られます。名前で指定されているフィールドおよび位置で指定されているフィールドに対して同じ結果が必要な場合は、PRESERVE BLANKSオプションを使用します。PRESERVE BLANKSオプションによって、フィールド値の評価時にSQL*Loaderによって空白文字が切り捨てられないようにします。

WHEN、NULLIFまたはDEFAULTIF句は、SQL*Loaderの処理ステップによっても結果に影響します。SQL*Loaderは次のステップを順番に実行します。ただし、すべてのステップを実行するわけではありません。フィールドが設定されると、設定作業の残りのステップは無視されます。たとえば、ステップ5でフィールドが設定された場合、SQL*Loaderはステップ6には進みません。

1. SQL*Loaderで、入力レコードの各フィールドの値が評価され、空白およびタブの切り捨てに関する既存のガイドラインに従って、切り捨てる必要のある空白が切り捨てられます。
2. 各レコードに対して、SQL*Loaderで表のWHEN句が評価されます。
3. レコードが表のWHEN句を満たす場合、またはWHENが指定されていない場合は、SQL*Loaderで、NULLIF句の各フィールドが確認されます。
4. NULLIF句が存在する場合は、SQL*Loaderで評価されます。
5. NULLIF句が満たされている場合は、SQL*LoaderではフィールドがNULLに設定されます。
6. NULLIF句が満たされていない場合、またはNULLIF句がない場合は、SQL*Loaderのフィールド評価によってフィールド長が確認されます。フィールドがフィールド評価の結果、0の長さを持っている場合(たとえば、NULLフィールドまたは結果としてNULLフィールドとなる空白の切捨て)は、SQL*LoaderでフィールドがNULLに設定されます。この場合、フィールドに指定されたDEFAULTIF句は評価されません。
7. 指定されたNULLIF句が偽の場合、またはNULLIF句がない場合、およびフィールドがフィールド評価の結果、0の長さを持たない場合は、SQL*Loaderで、DEFAULTIF句に対するフィールドが確認されます。
8. DEFAULTIF句が存在する場合は、SQL*Loaderで評価されます。
9. DEFAULTIF句が満たされていて、データ・ファイルのフィールドが数値フィールドの場合、フィールドは0に設定されます。フィールドが数値フィールドではない場合、NULLに設定されます。次のフィールドは数値フィールドで、DEFAULTIF句を満たす場合は、0に設定されます。
   - BYTEINT
   - SMALLINT
   - INTEGER
   - FLOAT
   - DOUBLE
   - ZONED
   - (パック) DECIMAL
   - 数値型 EXTERNAL(INTEGER、FLOAT、DECIMALおよびZONED)
10. DEFAULTIF句が満たされていない場合、またはDEFAULTIF句がない場合は、SQL*Loaderによって、ステップ1の評価値でフィールドが設定されます。

SQL*Loaderの操作順序が原因で、予期しない結果となる場合があります。たとえば、DEFAULTIF句は、数値フィールドを0ではなくNULLに設定しているように見える場合があります。

ノート:

これらのステップに示したとおり、NULLIF および DEFAULTIF 句を使用した場合は、SQL*Loader による処理が増えます。そのためパフォーマンスに影響する可能性があります。ステップ 1 では、長さが 0 と評価されたフィールド

は、SQL*Loader によって NULL 値に設定されます。パフォーマンスを向上させるために、この処理シーケンスを利用するようにデータを変更できるかどうかを検討してください。ステップ 1 での NULL の検出は、NULLIF 句または DEFAULTIF 句の処理よりはるかに迅速に行われます。

たとえば、CHAR(5)は、論理レコードの終わりまでに格納されない場合、または含まれているのがすべて空白で、空白の切捨てが有効になっている場合は、データ長が 0 になります。デリミタで区切られたフィールドは、フィールドの開始と終了記号の間に文字がない場合、データ長が 0 になります。

また、文字フィールドの場合は、通常、NULLIF の方が DEFAULTIF より迅速に処理されます(文字フィールドのデフォルト値は NULL です)。

関連トピック

- 表レベルでのNULLIF句の指定

親トピック: SQL*Loaderフィールド・リスト・リファレンス

10.7 WHEN、NULLIFおよびDEFAULTIF句の使用例

これらの例では、WHEN、NULLIFおよびDEFAULTIF句を使用できる様々な状況の結果について説明します。

これらの例では、空白またはスペースはピリオド(.)で示されています。col1およびcol2は、データベースのVARCHAR2(5)列です。

例10-3 DEFAULTIF句の無評価

制御ファイルが次のように指定されています。

```
(col1 POSITION (1:5),
 col2 POSITION (6:8) CHAR INTEGER EXTERNAL DEFAULTIF col1 = 'aname')
```

データ・ファイルには次の情報が格納されています。

```
aname...
```

この例では、行のcol1がanameと評価され、col2が長さが0 (「...」ですが、後続の空白は位置フィールドでは切り捨てられます)であるNULLと評価されます。

SQL*Loaderでcol2にロードされた最終値が確認される場合、WHEN句およびNULLIF句は評価されません。フィールド長(フィールド評価の結果、0と評価された長さ)が確認されます。したがって、SQL*Loaderでは、col2の最終値にNULLが設定されます。DEFAULTIF句は評価されず、行はcol1の場合はaname、col2の場合はNULLとしてロードされます。

例10-4 DEFAULTIF句の評価

制御ファイルが次のように指定されています。

```
.
.
.
PRESERVE BLANKS
.
.
.
(col1 POSITION (1:5),
 col2 POSITION (6:8) INTEGER EXTERNAL DEFAULTIF col1 = 'aname'
```

データ・ファイルには次の情報が格納されています。

```
aname...
```

この例では、行のcol1が再度anameと評価されます。col2は、PRESERVE BLANKSが指定された場合、後続の空白が切り捨てられないため、「...」と評価されます。

SQL*Loaderでcol2にロードされた最終値が確認される場合、WHEN句およびNULLIF句は評価されません。0でなく3と評価されたフィールド評価からフィールド長が確認されます。

次に、SQL*LoaderではDEFAULTIF句が評価されます。col1がanameでanameと同じであるため、真と評価されます。

col2は数値フィールドであるため、SQL*Loaderではcol2の最終値に0が設定されます。行は、col1の場合は「aname」、col2の場合は0としてロードされます。

例10-5 DEFAULTIF句を使用した位置の指定

制御ファイルが次のように指定されています。

```
(col1 POSITION (1:5),
 col2 POSITION (6:8) INTEGER EXTERNAL DEFAULTIF (1:5) = BLANKS)
```

データ・ファイルには次の情報が格納されています。

```
.....123
```

この例では、行のcol1が長さが0 (.....ですが、後続の空白は切り捨てられます)であるNULLと評価されます。col2は、123と評価されます。

SQL*Loaderでcol2にロードされる最終値が設定される場合、WHEN句およびNULLIF句は評価されません。0でなく3と評価されたフィールド評価からフィールド長が確認されます。

次に、SQL*LoaderではDEFAULTIF句が評価されます。.....である(1:5)を、真と評価されているBLANKSと比較します。したがって、col2が数値フィールド(integer EXTERNALは数値)のため、SQL*Loaderではcol2の最終値に「0」が設定されます。行は、col1の場合はNULL、col2の場合は0としてロードされます。

例10-6 DEFAULTIF句を使用したフィールド名の指定

制御ファイルが次のように指定されています。

```
(col1 POSITION (1:5),
 col2 POSITION(6:8) INTEGER EXTERNAL DEFAULTIF col1 = BLANKS)
```

データ・ファイルには次の情報が格納されています。

```
.....123
```

この例では、行のcol1が長さが0 (.....ですが、後続の空白は切り捨てられます)であるNULLと評価されます。col2は、123と評価されます。

SQL*Loaderでcol2の最終値が確認される場合、WHEN句およびNULLIF句は評価されません。0でなく3と評価されたフィールド評価からフィールド長が確認されます。

次に、SQL*LoaderではDEFAULTIF句が評価されます。評価の一部として、col1のフィールド評価がNULLであることが確認されます。NULLのため、DEFAULTIF句は偽と評価されます。したがって、SQL*Loaderではcol2の最終値に、フィールド評価の元の値である123が設定されます。行は、col1の場合はNULL、col2の場合は123としてロードされます。

親トピック: SQL*Loaderフィールド・リスト・リファレンス

10.8 異なるプラットフォーム間でのデータのロード

データ・ファイルを作成するプラットフォームと、そのデータ・ファイルのロード先となるプラットフォームが異なる場合は、ターゲット・システムが読取り可能な形式でデータを作成する必要があります。

たとえば、ソース・システムではシステム固有の浮動小数点の内部表現に16バイトを使用し、ターゲット・システムでは浮動小数点を12バイトで表現しているとします。この場合、ソース・システムで生成されたデータを、ターゲット・システムに直接読み込ませることはできません。

この問題を解決する方法として、Oracle Netデータベース・リンクを使用してデータをロードし、データ型の自動変換機能を利用する方法があります。前述のような問題が発生した場合は、できるだけこの方法を使用してください。この場合、SQL*Loaderはソース・システムで実行する必要があります。

プラットフォーム間のロードに関する問題は、通常、システム固有のデータ型に関連して発生します。フィールドに0(ゼロ)を追加してフィールド長を長くするか、またはフィールドの一部分のみを読み込んでフィールド長を短くする(4バイト整数を使用しているシステム上に8バイト整数を読み込む場合、またはその逆のパターンがこれに相当)ことによって、問題を回避できる場合もあります。ただし、データ型の実装に互換性がない場合は、この方法で問題の解決はできません。

Oracle Netデータベース・リンクが使用できず、ターゲット・システム上で実行しているSQL*Loaderを使用してデータ・ファイルにアクセスする必要がある場合は、移植可能なSQL*Loaderデータ型(たとえば、CHAR、DATE、VARCHARC、数値型EXTERNAL)のみを使用してください。これらのデータ型を使用して書き込まれたデータ・ファイルは、システム固有のデータ型を使用して書き込まれたデータ・ファイルより大きくなる可能性があります。そのため、ロードに時間がかかりますが、異なるプラットフォームに直接転送することができます。

バイト順序スキームまたはシステム固有の整数の長さが、入力データが作成されるプラットフォームとSQL*Loaderを実行するプラットフォームの間で異なることが事前にわかっている場合は、適切な方法で、データのバイト順序またはシステム固有の整数の長さを指定します。バイト順序を指定する方法には、BYTEORDERパラメータを使用する方法、またはファイルにバイト順序マーク(BOM)を設定する方法があります。この2つの方法の詳細は、「バイト順序」を参照してください。これらの方法を実行すると、非互換性を排除し、プラットフォーム間での正常なデータ・ロードを実現できます。バイト順序がSQL*Loaderのデフォルトと異なる場合は、バイト順序を指定する必要があります。

親トピック: SQL*Loaderフィールド・リスト・リファレンス

10.9 バイト順序

SQL*Loaderを実行するシステムとは異なるバイト順序スキームを持つシステムで入力データを作成する予定の場合は、バイト順序指定を確認してください。

> ノート:
>
> この項で示す情報は、ビッグ・エンディアンからリトル・エンディアン・システムへのデータの移動など、SQL*Loader を実行するシステムとはバイト順序スキームが異なるシステム上で入力データを作成する場合のみに当てはまります。各システムのバイト順序スキームが同じである場合、この項を読む必要はありません。

- SQL*Loaderによるバイト順序の管理方法の理解
  SQL*Loaderでは、SQL*Loaderが実行されているシステムとはバイト順序が異なるシステム上で作成されたデータ・ファイルから、(移植不能な特定のデータ型がデータ・ファイルに含まれている場合であっても)データをロードできます。
- バイト順序の構文

BYTEORDERの構文図を使用して、SQL*Loaderでのデータのバイト順序の指定方法を確認します。

- バイト順序マーク(BOM)の使用
  この項では、バイト順序マークの使用方法について説明します。
- BOMの確認の抑止
  この項では、BOMの確認の抑止方法について説明します。

親トピック: SQL*Loaderフィールド・リスト・リファレンス

10.9.1 SQL*Loaderによるバイト順序の管理方法の理解

SQL*Loaderを使用して、SQL*Loaderが実行されているシステムとはバイト順序が異なるシステム上で作成されたデータ・ファイルから、(データ・ファイルに移植不能な特定のデータ型が含まれている場合であっても)データをロードできます。

デフォルトでは、すべてのデータ・ファイルに対するバイト順序として実行されているシステムのバイト順序が、SQL*Loaderで使用されます。たとえば、Oracle Solarisシステムでは、SQL*Loaderはビッグ・エンディアン・バイト順序を使用します。IntelまたはIntelと互換性のあるPC上では、SQL*Loaderでリトル・エンディアン・バイト順序が使用されます。

バイト順序は、データが一度に偶数バイト(通常、2バイト、4バイトまたは8バイト)書き込まれる場合、および読み込まれる場合の結果に影響します。次に例をいくつか示します。

- 2バイト整数値の1は、ビッグ・エンディアン・システムでは0x0001、リトル・エンディアン・システムでは0x0100として書き込まれます。
- 4バイト整数の66051は、ビッグ・エンディアン・システムでは0x00010203、リトル・エンディアン・システムでは0x03020100として書き込まれます。

バイト順序は、UTF16文字セットの文字データが2バイトのエントリとして書き込まれる場合、および読み込まれる場合の結果にも影響します。たとえば、文字「a」(ASCIIでは0x61)は、ビッグ・エンディアン・システムでは0x0061、リトル・エンディアン・システムでは0x6100として書き込まれます。

Oracleでサポートされているすべての文字セット(UTF16を除く)は一度に1バイトずつ書き込まれます。そのため、UTF8などのマルチバイト文字セットの場合も、文字の書込み、読取りには、システムのバイト順序に関係なく、すべてのシステムで同じ方法が使用されます。したがって、UTF16文字セットのデータは、バイト順序依存のため移植不能です。Oracleでサポートされる他のすべての文字セットのデータは移植可能です。

データ・ファイルのバイト順序が問題となるのは、バイト順序依存データを含むデータ・ファイルが、SQL*Loaderが実行されているシステムとは異なるバイト順序のシステムで作成される場合のみです。SQL*Loaderでデータのバイト順序を識別できる場合、必要に応じてバイトを入れ替えて、データが正常にターゲット・データベースにロードされることを確認します。バイト・スワップとは、ビッグ・エンディアン形式のデータをリトル・エンディアン形式に変換すること、またはその逆の変換を意味します。

SQL*Loaderにデータのバイト順序を指定するには、BYTEORDERパラメータを使用するか、またはそのファイルにバイト順序マーク(BOM)を設定します。この2つの方法のいずれも使用しない場合、SQL*Loaderではデータを正常にデータ・ファイルにロードできません。

関連トピック

- SQL*Loaderの事例

関連項目:

SQL*Loaderによるバイト・スワップの処理方法の例は、SQL*Loader事例11「Unicode文字セットのデータのロード」を参

照してください。

親トピック: バイト順序

10.9.2 バイト順序の構文

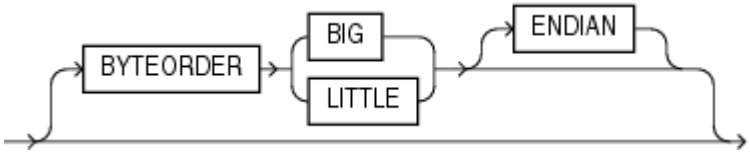
BYTEORDERの構文図を使用して、SQL*Loaderでのデータのバイト順序の指定方法を確認します。

用途

入力データ・ファイル内のデータのバイト順序を指定するには

構文

SQL*Loader制御ファイル内で次の構文を使用します。



使用上のノート

BYTEORDERパラメータには、次の特長があります。

- BYTEORDERは、SQL*Loader制御ファイルのLENGTHパラメータの後に置かれます。
- 異なるデータ・ファイルに異なるバイト順序を指定できます。ただし、INFILEパラメータの前のBYTEORDER指定が、プライマリ・データ・ファイルのすべてのリストに適用されます。
- プライマリ・データ・ファイルに対するBYTEORDER指定は、LOBFILEおよびSDFに対するデフォルトとしても使用されます。このデフォルトを上書きするには、LOBFILEまたはSDFの指定を使用してBYTEORDERを指定します。
- BYTEORDERパラメータは、制御ファイル内のデータには適用されません。
- BYTEORDERパラメータは次のものに適用されます。
    - 2進INTEGERデータおよびSMALLINTデータ
    - 可変長フィールドのバイナリ長(VARCHAR、VARGRAPHIC、VARRAWおよびLONG VARRAWデータ型用)
    - UTF16文字セットのデータ・ファイルの文字データ
    - FLOATおよびDOUBLEデータ型(データが書き込まれたシステムに、SQL*Loaderが実行されているシステム上のデータと互換性のある浮動小数点の表現がある場合)
- BYTEORDERパラメータは、次のものには適用されません。
    - RAWデータ型(RAW、VARRAWまたはVARRAWC)
    - 図形データ型(GRAPHIC、VARGRAPHICまたはGRAPHIC EXTERNAL)
    - UTF16以外の文字セットのデータ・ファイルの文字データ
    - ZONEDまたは(PACKED) DECIMALデータ型

親トピック: バイト順序

10.9.3 バイト順序マーク(BOM)の使用

この項では、バイト順序マークの使用方法について説明します。

Unicodeエンコーディング(UTF-16またはUTF-8)が使用されているデータ・ファイルには、ファイルの最初の数バイトにバイト順

序マーク(BOM)が含まれている場合があります。文字セットUTF-16が使用されているデータ・ファイルでは、ファイルの最初の2バイトの値{0xFE,0xFF}は、ファイルがビッグ・エンディアンのデータを含んでいることを示すBOMです。{0xFF,0xFE}という値は、ファイルにリトル・エンディアンのデータが含まれていることを示すBOMです。

第1プライマリ・データ・ファイルでUTF16文字セットが使用され、BOMで開始されている場合は、SQL*Loaderでそのマークを読み込んで解釈し、すべてのプライマリ・データ・ファイルのバイト順序を決定します。SQL*Loaderで、BOMを読み込んで解釈し、スキップして、BOMの直後のバイトからデータを処理し始めます。BOM設定は、第1プライマリ・データ・ファイルに対するBYTEORDER指定より優先されます。第1プライマリ・データ・ファイル以外のデータ・ファイルのBOMは、バイト順序競合の確認のみに使用されます。データ・ファイルの処理中にSQL*Loaderで使用されるバイト順序の設定は変更されません。

つまり、第1プライマリ・データ・ファイルに対するバイト順序インジケータの優先順位は次のようになります。

- 第1プライマリ・データ・ファイルのBOM(データ・ファイルでバイト順序に依存したUnicode文字セット(UTF16)を使用し、BOMが存在する場合)
- BYTEORDERパラメータの値(INFILEパラメータの前に指定された場合)
- SQL*Loaderが実行されているシステムのバイト順序

UTF8文字セットが使用されているデータ・ファイルでは、最初の3バイトの{0xEF,0xBB,0xBF}というBOMによって、ファイルにUTF8データが含まれていることが示されます。UTF8のデータはバイト順序依存ではないため、BOMではデータのバイト順序を指定しません。UTF8のBOMが検出された場合は、SQL*LoaderでBOMをスキップしますが、データ・ファイルの処理のためのバイト順序の設定変更は実行しません。

SQL*Loaderによって、まず、定義済の優先順位を使用して第1プライマリ・データ・ファイルのバイト順序設定を確立します。このバイト順位設定は、すべてのプライマリ・データ・ファイルに使用されます。別のプライマリ・データ・ファイルで文字セットUTF16が使用され、BOMも含まれている場合、そのBOMの値は、第1プライマリ・データ・ファイルで確立されたバイト順序設定と比較されます。BOMの値が第1プライマリ・データ・ファイルのバイト順序設定と一致する場合は、SQL*LoaderでそのBOMをスキップし、そのバイト順序設定を使用して、BOMの直後のバイトからデータを処理し始めます。BOMの値が、第1プライマリ・データ・ファイルで確立されたバイト順序設定と一致しない場合は、SQL*Loaderでエラー・メッセージを発行し、処理を停止します。

LOBFILEまたはセカンダリ・データ・ファイル(SDF)が制御ファイルで指定される場合、ファイルを処理する準備が整うと、SQL*Loaderで各LOBFILEおよびSDFのバイト順序設定を確立します。LOBFILEおよびSDFのデフォルトのバイト順序設定は、第1プライマリ・データ・ファイルで確立されたバイト順序設定です。これは、BYTEORDERパラメータがLOBFILEまたはSDFで指定される場合は、上書きされます。いずれの場合も、LOBFILEまたはSDFでUTF16文字セットが使用され、BOMが含まれている場合、BOMの値はファイルのバイト順序と比較されます。BOMの値がファイルのバイト順序設定と一致する場合は、SQL*LoaderでそのBOMをスキップし、そのバイト順序設定を使用して、BOMの直後のバイトからデータを処理し始めます。BOMの値が一致しない場合、SQL*Loaderからエラーが発行され、処理が停止します。

つまり、LOBFILEおよびSDFに対するバイト順序インジケータの優先順位は次のようになります。

- LOBFILEまたはSDFで指定されたBYTEORDERパラメータ値
- 第1プライマリ・データ・ファイルで確立されたバイト順序設定

ノート:

データ・ファイルの文字セットが Unicode 文字セットで、ファイルの最初の数バイトにバイト順序マークが設定されている場合、SKIP パラメータは使用しないでください。使用した場合、バイト順序マークは読み込

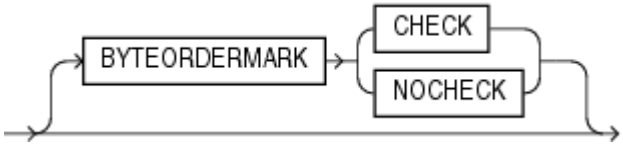
まれず、バイト順序マークとして解釈されません。

親トピック: バイト順序

10.9.4 BOMの確認の抑止

この項では、BOMの確認の抑止方法について説明します。

Unicode文字セットのデータ・ファイルには、ファイルの最初のバイトにBOMと一致するバイナリ・データが含まれています。たとえば、integer(2)型の値0xFEFF = 65279小数点は、UTF16のビッグ・エンディアンBOMと一致します。この場合、SQL*Loaderを使用してデータ・ファイルの最初のバイトをデータとして読み込むことができます。また、値NOCHECKでBYTEORDERMARKパラメータを指定して、SQL*LoaderによってBOMを確認しないようにできます。BYTEORDERMARKパラメータの構文は次のとおりです。



BYTEORDERMARK NOCHECKを指定すると、SQL*LoaderではBOMが確認されず、データ・ファイルのすべてのデータがデータとして読み込まれます。

BYTEORDERMARK CHECKを指定すると、BOMを確認するようSQL*Loaderに指示します。これはUnicode文字セットのデータ・ファイルについてのデフォルト動作です。ただし、この指定は明確化の目的で制御ファイル内で使用されることがあります。Unicode以外の文字セットを使用するデータ・ファイルについてBYTEORDERMARK CHECKを指定すると、エラーとなります。

BYTEORDERMARKパラメータには、次の特長があります。

- SQL*Loader制御ファイル内のオプションのBYTEORDERパラメータの後に置かれます。
- プライマリ・データ・ファイルの構文指定に適用される以外に、LOBFILEとセカンダリ・データ・ファイル(SDF)にも適用されます。
- 異なるデータ・ファイルに異なるBYTEORDERMARK値を指定できます。ただし、INFILEパラメータの前のBYTEORDERMARK指定が、プライマリ・データ・ファイルのリスト全体に適用されます。
- プライマリ・データ・ファイルに対するBYTEORDERMARK指定は、LOBFILEおよびSDFに対するデフォルトとしても使用されます。ただし、LOBFILEまたはSDFでUnicode以外の文字セットが使用されている場合は、値CHECKは無視されます。LOBFILEおよびセカンダリ・データ・ファイルに対するこのデフォルト設定は、LOBFILEまたはSDF指定にBYTEORDERMARKを指定することによって上書きできます。

親トピック: バイト順序

10.10 すべてが空白のフィールドのロード

フィールドが完全に空白のレコードは拒否されます。これらのフィールドのいずれかをNULLとしてロードするには、BLANKSパラメータを持つNULLIF句を使用します。

空白のフィールドがCHAR型で、囲みデリミタによって囲まれている場合は、囲みデリミタで囲まれている空白部分がロードされます。囲みデリミタで囲まれていない場合は、そのフィールドはNULLとしてロードされます。

DATEフィールドまたは数値フィールドのデータがすべて空白の場合は、NULLフィールドとしてロードされます。

関連トピック

- SQL*Loaderの事例
- 空白の切捨て
- 空白の切捨てに対するPRESERVE BLANKSオプションの影響

関連項目:

NULLIF句を使用してNULLとしてすべての空白フィールドをロードする方法の例は、SQL*Loaderの事例での事例6「ダイレクト・パス・ロード方式を使用したデータのロード」を参照してください。

親トピック: SQL*Loaderフィールド・リスト・リファレンス

10.11 空白の切捨て

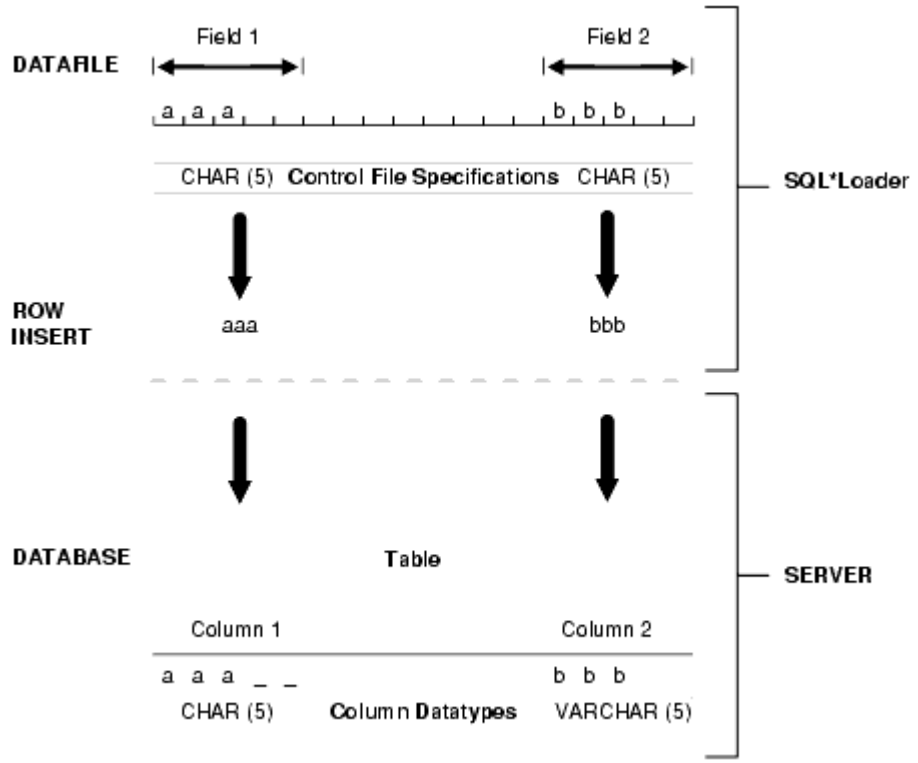
空白、タブおよびその他の印字されない文字(改行、LFなど)が空白を構成します。

先頭の空白は、フィールドの開始位置にあります。後続の空白は、フィールドの終了位置にあります。フィールドの指定方法にもよりますが、空白は、フィールドのデータベースへの挿入時に、データの一部として含めることも切り捨てることもできます。これについて、図「フィールド変換の例」に示します。この図では、2つのCHARフィールドがデータ・レコードに定義されています。

フィールド指定は制御ファイルに含まれています。制御ファイルのCHAR指定は、データベースのCHAR指定と同じではありません。制御ファイル内でCHARとして定義されたデータ・フィールドは、SQL*Loaderに行挿入の作成方法を指定するのみです。Oracle Databaseで必要な変換が行われ、データベースのCHAR、VARCHAR2、NCHAR、NVARCHAR2、NUMBERまたはDATE列にデータを挿入できるようになります。

デフォルトでは、SQL*Loaderを使用してCHARデータから後続の空白を削除した後、このデータをデータベースに渡します。したがって、図「フィールド変換の例」では、フィールド1とフィールド2の両方が3バイトのフィールドとしてデータベースに渡されます。ただし、データを表に挿入する場合は処理が異なります。

図10-1 フィールド変換の例



列1は、データベース内で長さ5の固定長CHAR列として定義されます。そのため、データ(aaa)は5バイトの幅を保持したまま、

その列で左揃えにされます。余った右側の部分は空白で埋められます。一方、列2は、最大長5バイトの可変長フィールドとして定義されています。その列(bbb)のデータも左揃えにされますが、長さは3バイトのままです。

表「空白の切捨てに関する動作のサマリー」に、PRESERVE BLANKSが指定されていない場合に空白が入力データ・フィールドから切り捨てられるケースおよびその処理方法を示します。空白の切捨てを回避する方法の詳細は、「空白の切捨てに対するPRESERVE BLANKSオプションの影響」を参照してください。

表10-5 空白の切捨てに関する動作のサマリー

指定	データ	結果	先頭の空白(すべてが空白のフィールドが切り捨てられた場合、その値はNULLになります。)	後続の空白(すべてが空白のフィールドが切り捨てられた場合、その値はNULLになります。)
サイズ指定あり	__aa__	__aa	あり	いいえ
終了	__aa__,	__aa__	あり	はい。空白で終了するフィールドを除きます。
囲み	"__aa__"	__aa__	あり	あり
終了と囲み	"__aa__",	__aa__	あり	あり
オプションの囲み(あり)	"__aa__",	__aa__	あり	あり
オプションの囲み(なし)	__aa__,	aa__	いいえ	あり
前のフィールドが空白で区切られている場合	__aa__	aa (後続の空白があるかどうかは、表中の他の項目に示すとおり、現行のフィールドの指定によって異なります。)	いいえ	後続の空白があるかどうかは、表中の他の項目に示すとおり、現行のフィールドの指定によって異なります。

この項の残りの部分では、空白の切捨てに関する次の項目について説明します。

- 空白の切捨てが可能なデータ型
  次の説明は、データ型が文字データ型であるフィールドにのみ適用できます。
- 空白の切捨てが可能なデータ型に対するフィールド長指定
  この項では、フィールド長の指定方法について説明します。
- フィールドの相対的な位置指定
  この項では、フィールドの相対的な位置の指定方法について説明します。
- 先頭の空白

この項では、先頭の空白について説明します。

- 後続の空白の切捨て
  後続の空白は、フィールドが文字データ型で事前にフィールド・サイズが決まっている場合、常に切り捨てられます。
- 囲まれたフィールドの切捨て
  この項では、囲まれたフィールドの切捨て方法について説明します。

親トピック: SQL*Loaderフィールド・リスト・リファレンス

10.11.1 空白の切捨てが可能なデータ型

次の説明は、データ型が文字データ型であるフィールドにのみ適用できます。

- CHARデータ型
- 日時および期間データ型
- 数値型EXTERNALデータ型:
  - INTEGER EXTERNAL
  - FLOAT EXTERNAL
  - (パック) DECIMAL EXTERNAL
  - ZONED (10進) EXTERNAL

ノート:

VARCHAR 型および VARCHARC 型フィールドも文字データを含みますが、フィールド中の空白は切り捨てられません。これらのフィールドは、データ・ファイルのフィールド中の空白文字をすべて含みます。

親トピック: 空白の切捨て

10.11.2 空白の切捨てが可能なデータ型に対するフィールド長指定

この項では、フィールド長を指定する方法について説明します。

フィールド長の指定には2通りの方法があります。制御ファイルで位置指定またはデータ型および長さについてフィールド長の定数を定義した場合、そのフィールドは、事前にサイズが決まっていることになります。一方、フィールド長を事前に指定しないでレコード中のインジケータでフィールド長を決める場合は、そのフィールドを囲みデリミタまたは終了デリミタのいずれかを使用して区切ります。

開始値および終了値での位置指定は、囲みデリミタまたは終了デリミタが定義されるフィールドに対して定義されます。囲みデリミタまたは終了デリミタは無視されます。

- 事前にサイズが決まっているフィールド
  事前にサイズが決まっているフィールドは、開始位置と終了位置、または長さで指定されます。
- デリミタ付きフィールド
  デリミタとは、フィールドの境界を指定する文字のことです。

親トピック: 空白の切捨て

10.11.2.1 事前にサイズが決まっているフィールド

事前にサイズが決まっているフィールドは、開始位置と終了位置、または長さで指定されます。

たとえば:

```
loc POSITION(19:31)
loc CHAR(14)
```

2番目の例では、フィールド位置は指定されていませんが、フィールド長は事前に指定されています。

親トピック: 空白の切捨てが可能なデータ型に対するフィールド長指定

10.11.2.2 デリミタ付きフィールド

デリミタとは、フィールドの境界を指定する文字のことです。

囲みデリミタは、次の例("__"が空白またはタブを表す)中の引用符のように、フィールドを囲みます。

```
"__aa__"
```

一方、終了デリミタは、フィールドの終わりを示します。次の例中のカンマがこれに相当します。

```
__aa__,
```

デリミタに使用する文字は、制御句TERMINATED BYおよびENCLOSED BYを使用して指定します。次に指定例を示します。

```
loc TERMINATED BY "." OPTIONALLY ENCLOSED BY '|'
```

親トピック: 空白の切捨てが可能なデータ型に対するフィールド長指定

10.11.3 フィールドの相対的な位置指定

この項では、フィールドの相対的な位置の指定方法について説明します。

SQL*Loaderでは、次の場合にフィールドの開始位置が決定されます。

- フィールドの開始位置が指定されていない場合
  フィールドの開始位置が指定されていない場合は、前のフィールドの終了位置の直後の位置が、そのフィールドの開始位置となります。
- 前のフィールドの終端がデリミタで指定されている場合
  前のフィールド(フィールド1)の終端がデリミタで指定されている場合、次のフィールドはそのデリミタの直後から開始します。
- 前のフィールドに囲みデリミタおよび終了デリミタの両方が含まれる場合
  フィールドが囲みデリミタと終了デリミタの両方で指定された場合、その次のフィールドは終了デリミタの直後の位置から開始します。

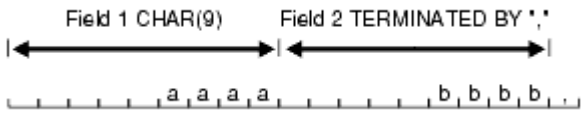
親トピック: 空白の切捨て

10.11.3.1 フィールドの開始位置が指定されていない場合

フィールドの開始位置が指定されていない場合は、前のフィールドの終了位置の直後の位置が、そのフィールドの開始位置となります。

次の図に、前のフィールド(フィールド1)のサイズが事前に決定されている場合の例を示します。

図10-2 固定長フィールドの後の相対的な位置指定



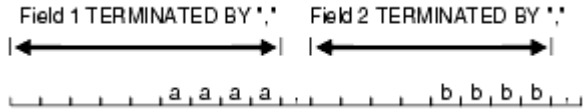
親トピック: フィールドの相対的な位置指定

10.11.3.2 前のフィールドの終端がデリミタで指定されている場合

前のフィールド(フィールド1)の終端がデリミタで指定されている場合、次のフィールドはそのデリミタの直後から開始します。

たとえば、図10-3のようになります。

図10-3 デリミタ付きフィールドの後の相対的な位置指定



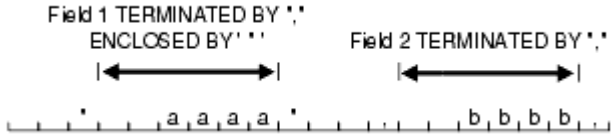
親トピック: フィールドの相対的な位置指定

10.11.3.3 前のフィールドに囲みデリミタおよび終了デリミタの両方が含まれる場合

フィールドが囲みデリミタと終了デリミタの両方で指定された場合、その次のフィールドは終了デリミタの直後の位置から開始します。

たとえば、図10-4のようになります。囲みデリミタから終了デリミタまでの間に空白以外の文字がある場合は、エラーが出力されます。

図10-4 囲みデリミタの後の相対的な位置指定



親トピック: フィールドの相対的な位置指定

10.11.4 先頭の空白

この項では、先頭の空白について説明します。

図10-4の例では、2つのフィールドはともに先頭の空白が付いた状態で格納されます。ただし、次のような場合、先頭の空白はフィールドのデータに含まれません。

- 前のフィールドが空白で区切られて(終了して)いて、現行のフィールドの開始位置が指定されていない場合。
- フィールドに対してオプションの囲みデリミタが指定されているにもかかわらず、その囲みデリミタが使用されていない場合。

これらの事例については、次の項で例示します。

- 前のフィールドが空白で区切られている場合
  前のフィールドがTERMINATED BY WHITESPACEで区切られていると、そのフィールドの後に続く空白はすべてデリミタとみなされます。
- オプションの囲みデリミタ付きフィールド
  オプションの囲みデリミタが指定されていて、それが使用されていない場合も、先頭の空白は切り捨てられます。

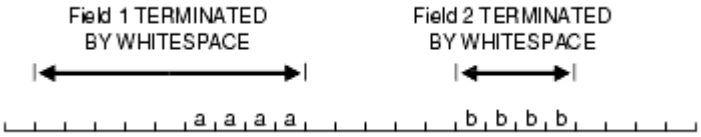
親トピック: 空白の切捨て

10.11.4.1 前のフィールドが空白で区切られている場合

前のフィールドがTERMINATED BY WHITESPACEで区切られていると、そのフィールドの後に続く空白はすべてデリミタとみなされます。

この場合、次のフィールドは、次に空白以外の文字が現れた位置から開始します。この例を図10-5に示します。

図10-5 空白で区切られたフィールド



このようなケースは、前述の例のとおり前のフィールドがTERMINATED BY WHITESPACE句で明示的に指定された場合に発生します。グローバルにFIELDS TERMINATED BY WHITESPACE句が指定された場合も、このケースに相当します。

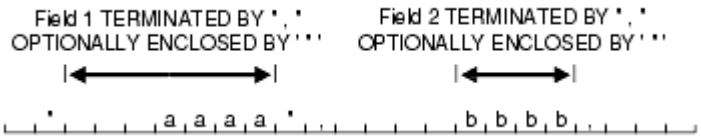
親トピック: 先頭の空白

10.11.4.2 オプションの囲みデリミタ

オプションの囲みデリミタが指定されているにもかかわらず、それが使用されていない場合も、先頭の空白は切り捨てられます。

オプションの囲みデリミタが指定されると、SQL*Loaderによって前方スキャンが実行され、最初の囲みデリミタが検索されます。囲みデリミタがない場合は、空白がスキップされ、フィールドから削除されます。最初に見つかった空白以外の文字がフィールドの開始と判断されます。この例を図10-6のフィールド2に示します。(フィールド1では、SQL*Loaderによってフィールドの囲みデリミタが検出されたため空白が含まれます)。

図10-6 オプションの囲みデリミタ付きフィールド



前のフィールドがTERMINATED BY WHITESPACEで区切られた場合と異なり、前述のように指定された場合は、現行のフィールドの開始位置が指定されていても先頭の空白は切り捨てられます。

> ノート:
>
> 囲みデリミタが存在する場合は、最初の囲みデリミタの後の先頭の空白はそのままデータとして保持されますが、この囲みデリミタの前にある空白は切り捨てられます。図 10-6 のフィールド 1 の最初の引用符がこのケースに相当します。

親トピック: 先頭の空白

10.11.5 後続の空白の切捨て

後続の空白は、フィールドが文字データ型で事前にフィールド・サイズが決まっている場合、常に切り捨てられます。

後続の空白が常に切り捨てられるのは、これらのフィールドのみです。

親トピック: 空白の切捨て

10.11.6 囲まれたフィールドの切捨て

この項では、囲まれたフィールドの切捨て方法について説明します。

フィールドが囲みデリミタで囲まれている場合、または、図10-6の最初のフィールドのように終了デリミタと囲みデリミタの両方で区切られている場合、囲みデリミタの外側にある空白は、フィールドの一部とはみなされません。囲みデリミタの内側に空白があれば、それが先頭の空白または後続の空白のいずれであっても、フィールドの一部とみなされます。

親トピック: 空白の切捨て

10.12 空白の切捨てに対するPRESERVE BLANKSオプションの影響

すべてのCHARフィールド、DATEフィールド、および数値型EXTERNALフィールドで空白が切り捨てられないようにするには、制御ファイル内のLOAD文でPRESERVE BLANKSを指定します。

ただし、すべてのCHAR、DATEまたは数値型EXTERNALフィールドに空白を残す必要がない場合があります。その場合は、SQL*Loaderを使用して、PRESERVE BLANKSを、LOAD文でグローバルに指定するのではなく、個々のフィールドのデータ型指定で指定することもできます。

次の例では、PRESERVE BLANKSがLOAD文で指定されていない場合に、空白を含むc1フィールドをデフォルトでゼロにします。これは、個々のフィールドに対してPRESERVE BLANKSを指定することで実現できます。指定したフィールドのみで空白が切り捨てられ、その他のフィールドでは空白はそのまま残されます。

```
c1 INTEGER EXTERNAL(10) PRESERVE BLANKS DEFAULTIF c1=BLANKS
```

この例では、PRESERVE BLANKSがフィールドに対して指定されていない場合、そのフィールドがNULL(0ではなく)として誤ってロードされます。

LOAD文に対するオプションとしてPRESERVE BLANKSを指定して、ほとんどのCHAR、DATEおよび数値型EXTERNALフィールドに適用する必要がある場合があります。このオプションの指定は、個々のフィールドのデータ型指定でNO PRESERVE BLANKSを指定して上書きできます。次に例を示します。

```
c1 INTEGER EXTERNAL(10) NO PRESERVE BLANKS
```

親トピック: SQL*Loaderフィールド・リスト・リファレンス

10.13 [NO] PRESERVE BLANKSとデリミタ句の併用

PRESERVE BLANKSオプションは、デリミタ句の存在の影響を受けます。

デリミタ句は、次の場合にPRESERVE BLANKSに影響します。

- オプションの囲みデリミタがない場合、先頭の空白はそのまま残ります。
- 事前にフィールド・サイズが指定された場合にも、後続の空白を残します。

たとえば、次のフィールドについて考えてみます。ここでは、アンダースコアは空白を表します。

```
__aa__,
```

このフィールドが次のデリミタ句でロードされるとします。

```
TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
```

この場合、PRESERVE BLANKSを指定すると、先頭の空白および後続の空白がともに保存されます。PRESERVE BLANKS

を指定しない場合は、先行する空白は切り捨てられます。

次に、このフィールドが次の句でロードされるとします。

```
TERMINATED BY WHITESPACE
```

この場合、PRESERVE BLANKSを指定すると、次のフィールドの先頭の空白は、このフィールドが空白を含むPOSITION句で指定されていないかぎり保存されません。このような指定がない場合は、SQL*Loaderによって前のフィールドの終わりにあるすべての空白を読み込まずにスキャンが実行され、次に空白以外の文字またはタブ以外の文字が現れた位置が次のフィールドの開始位置と認識されます。

関連トピック

- 空白の切捨て

親トピック: SQL*Loaderフィールド・リスト・リファレンス

10.14 フィールドへのSQL演算子の適用

この項では、SQL演算子をフィールドに適用する方法について説明します。

SQL文字列を使用することによって、様々なSQL演算子をフィールド・データに適用できます。SQL文字列には、任意に組み合せたSQL式を組み込むことができます。ただし、このSQL式は、Oracle DatabaseによってINSERT文中のVALUES句に対して有効であると認識されたものにかぎります。通常、ターゲット列のデータ型と互換性のある単一の値を返すSQL関数を使用できます。SQL文字列は、列オブジェクトおよびコレクションなどのユーザー定義の複合型に加えて、単純なスカラー列型にも適用できます。

列名とSQL文字列のバインド変数における列名は、SQL識別子のルールによる解釈の結果、同じ列に対応している必要があります。ただし、次の例に示すとおり、2つの名前を完全に同じように記述する必要はありません。

```
LOAD DATA
INFILE *
APPEND INTO TABLE XXX
( "Last"   position(1:7)     char   "UPPER(:¥"Last¥")"
   first   position(8:15)    char   "UPPER(:first || :FIRST || :¥"FIRST¥")"
)
BEGINDATA
Grant  Phil
Taylor Jason
```

前述の例では、次のことに注意してください。

- 表の作成時に、(前述の"Last"という列のように)列識別子に小文字または特殊文字(あるいはその両方)が含まれているために二重引用符を使用して宣言されている場合、バインド変数の列名は、CREATE TABLE文で使用される列名と完全に一致する必要があります。
- 表の作成時、(前述のfirstという列のように)列識別子が二重引用符を使用しないで宣言されている場合は、first、FIRSTおよび"FIRST"のすべてが大文字として処理され、FIRSTに解決されるため、SQL文字列のバインド変数では、これらのいずれの書式も使用できます。

SQL文字列を使用する場合は、次のことに注意してください。

- SQL文字列を実行することは、フィールド設定の一部とはみなされません。そのかわり、SQL文字列の実行時には、フィールド設定およびNULLIFやDEFAULTIF句の結果が使用されます。したがって、評価の順序は次のようになります(ステップ1と2はWHEN、NULLIFおよびDEFAULTIF句の使用で説明されているステップの要約です)。

1. フィールド設定が実行されます。
2. NULLIFまたはDEFAULTIF句が適用されます(これにより、このような句が含まれるフィールドのフィールド設定の結果が変更されることがあります)。NULLIFおよびDEFAULTIF句をSQL式で使用すると、FINAL型の列の結果ではなく、フィールド設定の結果に影響があります。
3. SQL式は、ステップ1と2の終了後に得られたフィールドの結果を使用して評価されます。結果は、SQL式を含む列に従って割り当てられます。(SQL式が存在しない場合、ステップ1と2から得られた結果が列に割り当てられます。)

- 関連付けられたSQL文字列を含む文字入力が制御ファイルによって指定されている場合、SQL*Loaderによるデータ変更は試行されません。これは、SQL*Loaderでは、SQL演算子を使用して変更した文字入力データは、データベースの挿入に対し正しい結果を生成すると仮定されているためです。
- SQL文字列を指定する位置は、その列に関するその他の指定がすべて記述された後になります。
- SQL文字列は、二重引用符で囲んで記述します。
- SQL文字列内の列名を引用符で囲むには、エスケープ文字を使用する必要があります。

  前述の例では、Lastを二重引用符で囲んで大文字および小文字の混在を保持しています。また、二重引用符にはバックスラッシュ(エスケープ)文字を使用する必要があります。

- SQL文字列に列オブジェクト属性を参照する列名が含まれている場合は、バインド変数にオブジェクト属性をフルネームで指定する必要があります。フルネームの各属性名は、個々の識別子です。各識別子は、SQL識別子の引用符ルールに従い、フルネームの他の識別子からは独立しています。たとえば、"HEIGHT_%TILE"という名前の属性を持つCHILDという列オブジェクトがあるとします。(その属性名は、二重引用符で囲まれています。)バインド変数にオブジェクト属性をフルネームで指定するには、次のいずれかの書式を使用します。

  1. :CHILD.¥"HEIGHT_%TILE¥"
  2. :child.¥"HEIGHT_%TILE¥"

  フルネームを引用符で囲む(:¥"CHILD.HEIGHT_%TILE¥")と、バインド変数で使用されるオブジェクト属性名の引用符ルールは変更されたという警告メッセージが生成されます。この警告は、バインド変数を正しく記述する必要があることを提案するものにすぎません。ロードが失敗することを示すものではありません。完全な名前を引用符で囲むと、SQLでは、名前が複数の識別子で構成される完全な列オブジェクト属性名ではなく、1つの識別子として解釈されるため、引用規則が変更されました。

- SQL文字列の評価は、NULLIF句またはDEFAULTIF句の後、日時マスクよりも前に行われます。
- Oracle Databaseで文字列を認識できない場合は、エラーが発生してロードは終了します。文字列が認識されても、データベース・エラーが発生すると、エラーの発生した行は拒否されます。
- フィールド指定でEXPRESSIONパラメータを使用する場合は、SQL文字列が必要です。
- SQL文字列では、OID、SID、REFまたはBFILEを使用してロードしたフィールドは参照できません。また、SQL文字列では、FILLERフィールドやSQL文字列を使用する他のフィールドも参照できません。
- ダイレクト・パス・モードでは、SQL文字列でのVARRAY、ネストした表またはLOB列の参照はできません。この制限は、列オブジェクトの属性であるVARRAY、ネストした表またはLOB列にも適用されます。
- SQL文字列は、RECNUM、SEQUENCE、CONSTANTまたはSYSDATEフィールドでは使用できません。
- SQL文字列は、LOB、BFILE、XML列またはコレクションの要素であるファイルでは使用できません。
- ダイレクト・パス・モードでは、SQL文字列の式の評価後に返される最後の結果はスカラー・データ型である必要があります。つまり、ダイレクト・パス・ロードの実行時、式によってはオブジェクトまたはコレクション・データ型を返せません。

- フィールドの参照
  レコード中のフィールドを参照する場合は、フィールド名の前にコロン(:)を付けます。
- フィールド指定でのSQL演算子の一般的使用

小数点が暗黙的に指定されている外部データをロードする場合、または長いフィールドを切り捨てる場合、フィールド指定でSQL演算子を使用すると、データの管理に役立ちます。

- SQL演算子の組合せ
  SQL*LoaderでSQL演算子を組み合せてデータ・ロードでの複数のステップを実行する方法について説明します。
- 日付マスク付きのSQL文字列の使用
  日付マスクと併用する場合、日付マスクはSQL文字列の後で評価されます。
- 書式化されたフィールドの解析
  TO_CHAR演算子を使用して、書式化された日付および数値を格納できます。
- SQL文字列を使用したANYDATAデータベース型へのロード
  ANYDATAデータベース型には、異なる型のデータを格納できます。

関連トピック

- WHEN、NULLIFおよびDEFAULTIF句の使用

親トピック: SQL*Loaderフィールド・リスト・リファレンス

10.14.1 フィールドの参照

レコード中のフィールドを参照する場合は、フィールド名の前にコロン(:)を付けます。

このように指定すると、現在のレコードのフィールド値が代入されます。SQL文字列で、前にコロン(:)が付いたフィールド名もバインド変数として参照されます。一重引用符で囲まれたバインド変数は、バインド変数としてではなくテキスト・リテラルとして扱われることに注意してください。

次の例では、制御ファイルの現行のフィールドおよび他のフィールドの参照方法を示します。また、バインド変数を一重引用符で囲むことにより、テキスト・リテラルとして扱われることを示しています。示されている概念を十分に理解するために、次の例のノートも参照してください。

```
LOAD DATA
INFILE *
APPEND INTO TABLE YYY
(
 field1  POSITION(1:6) CHAR "LOWER(:field1)"
 field2  CHAR TERMINATED BY ','
         NULLIF ((1) = 'a') DEFAULTIF ((1)= 'b')
         "RTRIM(:field2)",
 field3  CHAR(7) "TRANSLATE(:field3, ':field1', ':1')",
 field4  COLUMN OBJECT
 (
  attr1  CHAR(3) NULLIF field4.attr2='ZZ' "UPPER(:field4.attr3)",
  attr2  CHAR(2),
  attr3  CHAR(3)  ":field4.attr1 + 1"
 ),
 field5  EXPRESSION "MYFUNC(:FIELD4, SYSDATE)"
)
BEGINDATA
ABCDEF1234511  ,:field1500YYabc
abcDEF67890    ,:field2600ZZghl
```

この例に関するノート:

- 次の行では、:field1は一重引用符で囲まれていないため、バインド変数として解釈されます。
  field1 POSITION(1:6) CHAR "LOWER(:field1)"
- 次の行では、':field1'と':1'は一重引用符で囲まれているため、テキスト・リテラルとして解釈され、変更されず

にTRANSLATE関数に渡されます。

```
field3 CHAR(7) "TRANSLATE(:field3, ':field1', ':1')"
```

引用符で囲まれた文字列中で引用符を使用する方法の詳細は、「ファイル名およびオブジェクト名の指定」を参照してください。

- 各入力データ読取りでは、バインド変数で参照されたフィールドの値はバインド変数に置き換えられます。たとえば、最初のレコードの値ABCDEFは、最初のフィールド:field1にマップされます。この値は、引数としてLOWER関数に渡されます。
- SQL文字列のバインド変数によって現行のフィールドを参照する必要はありません。前述の例では、フィールドfield4.attr1に対するSQL文字列のバインド変数によって、フィールドfield4.attr3を参照しています。field4.attr1フィールドは、入力レコードの値500およびNULL (2番目のレコードのNULLIF field4.attr2='ZZ'句がTRUEであるため)にマップされます。ただし、対応する列に格納された最終値はABCおよびGHLです。

  field4.attr3フィールドは、入力レコードの値ABCおよびGHLにマップされます。ただし、SQL式がfield4.attr1を参照しているため、対応する列に格納された最終値は500 + 1 = 501およびGHLです。(NULLフィールドに1を足しても、結果はNULLフィールドになります。)
- field5フィールドは、入力レコードのどのフィールドにもマップされません。ターゲット列に格納されている値は、2つの引数をとるPL/SQL関数MYFUNCの実行結果です。EXPRESSIONパラメータの使用には、入力データがフィールドにマップされないため、SQL文字列を使用して列の最終値を計算する必要があります。

親トピック: フィールドへのSQL演算子の適用

10.14.2 フィールド指定でのSQL演算子の共通使用

小数点が暗黙的に指定されている外部データをロードする場合、または長いフィールドを切り捨てる場合、フィールド指定でSQL演算子を使用すると、データの管理に役立ちます。

次のタスクでは、一般的にSQL演算子が使用されています。

- 暗黙の小数点が付いているEXTERNALデータをロードするには、次のように指定します。

```
field1 POSITION(1:9) DECIMAL EXTERNAL(8) ":field1/1000"
```

- また、長すぎるフィールドを切り捨てるには、次のように指定します。

```
field1 CHAR TERMINATED BY "," "SUBSTR(:field1, 1, 10)"
```

親トピック: フィールドへのSQL演算子の適用

10.14.3 SQL演算子の組合せ

SQL*LoaderでSQL演算子を組み合せてデータ・ロードでの複数のステップを実行する方法について説明します。

次の例では、SQL*Loaderでフィールド指定において複数のSQL演算子を適用する方法を示します。

```
field1 POSITION(*+3) INTEGER EXTERNAL
       "TRUNC(RPAD(:field1,6,'0'), -2)"
field1 POSITION(1:8) INTEGER EXTERNAL
       "TRANSLATE(RTRIM(:field1),'N/A', '0')"
field1 CHAR(10)
       "NVL( LTRIM(RTRIM(:field1)), 'unknown' )"
```

親トピック: フィールドへのSQL演算子の適用

10.14.4 日付マスク付きのSQL文字列の使用

日付マスクと併用する場合、日付マスクはSQL文字列の後で評価されます。

次のように指定されるフィールドがあるとします。

```
field1 DATE "dd-mon-yy" "RTRIM(:field1)"
```

SQL*Loaderの内部で次の文が生成され、挿入されます。

```
TO_DATE(RTRIM(<field1_value>), 'dd-mon-yyyy')
```

SQL文字列でDATEフィールド・データ型を使用する場合、日付マスクが必要です。これは、SQL*Loaderで、DATEパラメータの後の最初の引用符付き文字列が日付マスクであるとみなされるためです。たとえば、次のフィールド指定では、エラー(ORA-01821: 日付書式コードが無効です)が発生します。

```
field1 DATE "RTRIM(TO_DATE(:field1, 'dd-mon-yyyy'))"
```

この場合の簡単な解決策は、CHARデータ型を使用することです。

親トピック: フィールドへのSQL演算子の適用

10.14.5 書式化されたフィールドの解析

TO_CHAR演算子を使用して、書式化された日付および数値を格納できます。

たとえば:

```
field1 ... "TO_CHAR(:field1, '$09999.99')"
```

この例では、数値型の入力データを、書式化された形式で格納することができます。この場合、field1はデータベース中ではキャラクタ列です。この指定にある書式化文字(ドル記号やピリオドなど)は、データとともにそのままフィールドに格納されます。

ただし、このような値を数量や日付として格納すると、より柔軟な処理を行うことができます。この場合、データベース内の値に算術関数を指定しても、書式設定された値を選択してレポートを作成することができます。

SQL文字列を使用して、書式化されたレポートからデータをロードする例は、「事例7: 書式化されたレポートからのデータの抽出」にあります。(事例の使用方法については、「SQL*Loaderの事例」を参照してください。)

親トピック: フィールドへのSQL演算子の適用

10.14.6 SQL文字列を使用したANYDATAデータベース型へのロード

ANYDATAデータベース型には、異なる型のデータを格納できます。

SQL*loaderを使用してANYDATA型をロードするには、ファンクション・コールによって明示的に構築する必要があります。ファンクションは、この項で説明したとおり、SQL文字列のサポートを使用してコールします。

たとえば、ANYDATA型のmiscellaneousという名前の列を含む表があるとします。この列は、次のようにしてロードできます。この場合、番号を含むANYDATA型が作成されます。

```
LOAD DATA
INFILE *
APPEND INTO TABLE  ORDERS
(
miscellaneous CHAR "SYS.ANYDATA.CONVERTNUMBER(:miscellaneous)"
```

```
)
BEGINDATA
4
```

レコード内の値によっては、さらに複雑な条件で、異なる型を含むANYDATA型を作成する場合もあります。そのためには、レコード内の値に基づいて、ANYDATA型にする型を決定する独自のPL/SQLファンクションを記述し、適切なANYDATA.Convert*()ファンクションをコールしてその型を作成する必要があります。

関連項目:

- ANYDATAデータベース型の詳細は、『Oracle Database SQL言語リファレンス』を参照してください。
- PL/SQLでのANYDATAの使用方法の詳細は、『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』を参照してください。

親トピック: フィールドへのSQL演算子の適用

10.15 SQL*Loaderを使用した入力データの生成

この項では、データ・ファイルからデータを読み込むのではなく、SQL*Loaderでデータを生成してデータベース・レコードに格納するパラメータについて説明します。

ここで説明するパラメータは次のとおりです。

- ファイルを使用しないデータのロード
  この項では、ファイルを使用しないデータのロード方法について説明します。
- 列への定数値の設定
  生成されたデータの最も単純な形式は、列への定数値の設定です。
- 列への式の値の設定
  列名の後にEXPRESSIONパラメータを使用して、SQL演算子または特別に書かれたPL/SQL関数によって返された値に、その列を設定します。
- データファイル・レコード番号への1列の設定
  RECNUMパラメータを列名の後に指定すると、レコードのロード元である論理レコードの番号が、その列に設定されます。
- 列への現在の日付の設定
  SYSDATEを使用して列を指定すると、SQL言語のSYSDATEパラメータで定義されているとおりに現在のシステム日付が取得されます。
- 列への一意の順序番号の設定
  SEQUENCEパラメータは、特定の列に対して一意の値を設定します。SEQUENCEの値は、ロードされたレコードまたは拒否されたレコードが発生するたびに増加します。
- 複数の表に対する順序番号の生成
  一意の順序番号は、各表への挿入時に生成されるのではなく、各論理入力レコードに対して生成されます。したがって、データを複数の表に挿入する場合に、同じ順序番号を使用できます。

親トピック: SQL*Loaderフィールド・リスト・リファレンス

10.15.1 ファイルを使用しないデータのロード

この項では、ファイルを使用しないデータのロード方法について説明します。

フィールドの指定として順序、レコード番号、システム日付、定数およびSQL文字列式のみを指定してSQL*Loaderでデータ

を生成できます。

SQL*Loaderを使用して、LOAD文で指定された数のレコードを挿入します。この場合、SKIPパラメータは指定できません。

SQL*Loaderは、このような場合用に最適化されています。生成された指定のみが使用されていることを検出すると、SQL*Loaderでは、常に、指定されたすべてのデータ・ファイルが無視されます。I/Oの読取りは実行されません。

バインド配列用のメモリーも不要です。制御ファイル中にWHEN句が指定されている場合は、SQL*Loaderでデータの評価が必要であると判断され、入力レコードが読み込まれます。

親トピック: SQL*Loaderを使用した入力データの生成

10.15.2 列への定数値の設定

生成されたデータの最も単純な形式は、列への定数値の設定です。

ロード中であっても、何回ロードしても、このデータは変わりません。

- CONSTANTパラメータ

親トピック: SQL*Loaderを使用した入力データの生成

10.15.2.1 CONSTANTパラメータ

列に定数値を設定するには、CONSTANTを指定して、その後に値を指定します。

```
CONSTANT  value
```

CONSTANTデータは、SQL*Loaderでは、文字入力として認識されます。このデータは、必要に応じてデータベース列のデータ型に変換されます。

値を引用符で囲むこともできます。特に、指定する値に空白や予約語が含まれている場合は、必ず引用符で囲んでください。また、ターゲット列に対して必ず有効な値を指定してください。指定した値が無効な場合は、すべてのレコードが拒否されてしまいます。

2の32乗-1(4,294,967,295)より大きい数値は引用符で囲んでください。

> ノート:
>
> CONSTANT パラメータを使用して列に NULL を設定することはできません。列を NULL に設定する場合は、その列については何も指定しないでください。これによって、Oracle でレコードをロードするときに、その列に自動的に NULL が設定されます。CONSTANT および値の組合せを指定すると、列指定は完結します。

親トピック: 列への定数値の設定

10.15.3 列への式の値の設定

列名の後にEXPRESSIONパラメータを使用して、SQL演算子または特別に書き込まれたPL/SQL関数によって返された値に、その列を設定します。

EXPRESSIONパラメータの後に続くSQL文字列に、演算子または関数が指定されます。演算子または関数に必要なパラメータが適切に指定され、その演算子または関数によって返された結果が、ロード中の列のデータ型と互換性がある場合、任意の式を使用できます。

- EXPRESSIONパラメータ

親トピック: SQL*Loaderを使用した入力データの生成

10.15.3.1 EXPRESSIONパラメータ

列名、EXPRESSIONパラメータおよびSQL文字列の組合せは、完全なフィールド指定です。

```
column_name EXPRESSION "SQL string"
```

従来型パス・モードおよびダイレクト・パス・モードの両方で、EXPRESSIONパラメータを使用して、デフォルト値をcolumn_nameにロードできます。

```
column_name EXPRESSION "DEFAULT"
```

DEFAULTを使用し、モードがダイレクト・パスの場合、順序をデフォルトとして使用しても機能しません。

親トピック: 列への式の値の設定

10.15.4 データファイル・レコード番号への1列の設定

RECNUMパラメータを列名の後に指定すると、レコードのロード元である論理レコードの番号が、その列に設定されます。

レコードの番号は、最初のデータ・ファイルの先頭をレコード1として、順番にカウントされます。RECNUMは、各論理レコードが構成されるたびに増えていきます。廃棄、スキップ、拒否またはロードされたレコードもカウントされます。SKIP=10と指定した場合、最初にロードされるレコードのRECNUMの値は11になります。

- RECNUMパラメータ
  列名とRECNUMの組合せにより、列指定は完全なものになります。

親トピック: SQL*Loaderを使用した入力データの生成

10.15.4.1 RECNUMパラメータ

列名およびRECNUMの組合せを指定すると、列指定は完結します。

たとえば:

```
column_name RECNUM
```

親トピック: データファイル・レコード番号への1列の設定

10.15.5 列への現在の日付の設定

SYSDATEを使用して列を指定すると、SQL言語のSYSDATEパラメータで定義されているとおりに現在のシステム日付が取得されます。

詳細は、『Oracle Database SQL言語リファレンス』のDATEデータ型に関する項を参照してください。

- SYSDATEパラメータ
  列名とSYSDATEパラメータの組合せにより、列指定は完全なものになります。

親トピック: SQL*Loaderを使用した入力データの生成

10.15.5.1 SYSDATEパラメータ

列名とSYSDATEパラメータの組合せを指定すると、列指定は完結します。

たとえば:

```
column_name SYSDATE
```

データベースの列のデータ型は、CHARまたはDATE型にしてください。列がCHAR型の場合、日付は'dd-mon-yy'の書式でロードされます。ロード後は、この書式でのみ日付のロードができます。システム日付をDATE列にロードすると、そのシステム日付は、時間と日付を含む様々な書式でロードできます。

新しいシステム日付/時間は、従来型パス・ロードで挿入されたレコードの各配列や、ダイレクト・パス・ロード中にロードされた各レコード・ブロックで使用されます。

親トピック: 列への現在の日付の設定

10.15.6 列への一意の順序番号の設定

SEQUENCEパラメータは、特定の列に対して一意の値を設定します。SEQUENCEの値は、ロードされたレコードまたは拒否されたレコードが発生するたびに増加します。

廃棄またはスキップされたレコードに対しては、値は増加しません。

- SEQUENCEパラメータ

親トピック: SQL*Loaderを使用した入力データの生成

10.15.6.1 SEQUENCEパラメータ

列名とSEQUENCEパラメータの組合せを指定すると、列指定は完結します。

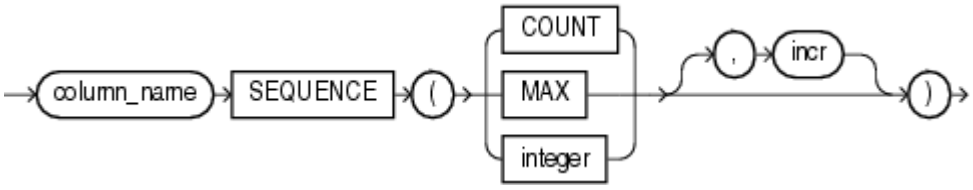


表10-6では、列指定に使用されるパラメータについて説明します。

表10-6 列指定に使用されるパラメータ

パラメータ	説明
column_name	データベース内の、順序を割り当てる列の名前。
SEQUENCE	列の値を指定するには、この SEQUENCE を使用します。
COUNT	順序番号は表中にすでにあるレコード数で始まり、以降は増分値を加えた値が設定されます。
MAX	順序番号は列の現在の最大値で始まり、以降は増分値を加えた値が設定されます。
integer	先頭となる特定の順序番号を指定します。
incr	レコードがロードまたは拒否された後の順序番号の増分値。これはオプションです。デ

パラメータ	説明
	フォルトは 1 です。

レコードが拒否された場合(書式エラーがあるか、またはOracleエラーが発生した場合)でも、その欠落を埋めるために生成された順序番号が変更されることはありません。たとえば、ある列に関して、4つの行に順序番号10、12、14および16が割り当てられ、番号12の行が拒否されたとします。この場合、挿入された3つの行の番号は10、14および16であって、10、12および14ではありません。このような処理が行われることによって、データ・エラーが発生しても、挿入時の順番を保持できます。拒否されたデータを修正して再度挿入する場合は、列の順序番号に一致するようにデータを手動で設定できます。

「事例3: 自由区分形式ファイルのロード」に、SEQUENCEパラメータの使用例があります。(事例の使用方法については、「SQL*Loaderの事例」を参照してください。)

親トピック: 列への一意の順序番号の設定

10.15.7 複数の表に対する順序番号の生成

一意の順序番号は、各表への挿入時に生成されるのではなく、各論理入力レコードに対して生成されます。したがって、データを複数の表に挿入する場合、同じ順序番号を使用することができます。

この仕様は、多くの場合に有効です。

ただし、INTO TABLE句ごとに別の順序番号を生成する場合もあります。たとえば、各入力レコード中に3つの論理レコードが定義された形式のデータについて考えます。この場合、INTO TABLE句を3つ使用して、レコードの3つの異なる部分を同じ表に挿入するように指定できます。SEQUENCE(MAX)を使用すると、各表の最大値が採用されるため、順序番号に一貫性がなくなります。

このレコードの順序番号を生成する場合は、挿入する3つの論理レコードそれぞれに対して、重複しない番号を生成する必要があります。1レコード当たりの表挿入の回数を順序番号の増分値として使用し、各挿入の順序番号をその続き番号で始めるようにします。

- 例: 挿入ごとの順序番号の生成

親トピック: SQL*Loaderを使用した入力データの生成

10.15.7.1 例: 挿入ごとの順序番号の生成

次に示す部門名をdept表にロードするとします。各入力レコードには部門名が3つ入っています。この部門番号を自動生成する方法について考えます。

```
Accounting      Personnel       Manufacturing
Shipping        Purchasing      Maintenance
...
```

部門番号を重複しないように生成するには、次のような制御ファイル・エントリを作成します。

```
INTO TABLE dept
(deptno  SEQUENCE(1, 3),
 dname   POSITION(1:14) CHAR)
INTO TABLE dept
(deptno  SEQUENCE(2, 3),
 dname   POSITION(16:29) CHAR)
INTO TABLE dept
(deptno  SEQUENCE(3, 3),
 dname   POSITION(31:44) CHAR)
```

最初のINTO TABLE句で生成される部門番号は1で、2番目では2が、3番目では3が生成されます。これらすべてで増分値として3が使用されます(増分値は各レコードに含まれる部門名の数と一致します)。この制御ファイルでロードを実行すると、Accounting部門は部門番号1、Personnel部門は部門番号2、Manufacturing部門は部門番号3でロードされます。

また、次のレコードになると、順序番号は増分値分のみ増加するので、Shipping部門は部門番号4、Purchasing部門は部門番号5でロードされ、以降も同様にロードされます。

親トピック: 複数の表に対する順序番号の生成

11 SQL*Loaderを使用したオブジェクト、LOBおよびコレクションのロード

SQL*Loaderを使用して、様々な形式で列オブジェクトがロードでき、オブジェクト表、REF列、LOBおよびコレクションのロードができます。

- 列オブジェクトのロード
  制御ファイルの列オブジェクトは、その属性によって記述されています。
- オブジェクト表のロード
  オブジェクト表のロードに必要な制御ファイルの構文は、典型的なリレーショナル表のロードの場合とほぼ同じです。
- REF列のロード
  SQL*Loaderは、システム生成OIDのREF列、主キーベースのREF列および主キーを使用できる有効範囲のないREF列をロードできます。
- SQL*Loaderを使用したLOBのロード
  SQL*Loaderでロードでロードできるラージ・オブジェクト型について調べて、LOBデータのロード方法を示す例を確認してください。
- SQL*Loaderを使用したBFILE列のロード
  BFILEデータ型には、データベースの外側にあるオペレーティング・システム・ファイルの非構造化バイナリ・データが格納されます。
- コレクション(ネストした表およびVARRAY)のロード
  LOBと同様、コレクションも、プライマリ・データ・ファイル(データ・インライン)またはSDF(データ・アウトライン)のいずれかからロードできます。
- 動的または静的SDF指定の選択
  SQL*Loaderを使用すると、SDFを静的に指定(実際のファイル名を指定)したり、または動的に指定(FILLERフィールドをファイル名のソースとして使用)したりできます。
- 親表を子表から分割してのロード
  ネストした表の列を含む表をロードする場合、親表を子表から分割してロードする場合があります。

親トピック: SQL*Loader

11.1 列オブジェクトのロード

制御ファイルの列オブジェクトは、その属性によって記述されています。

列オブジェクトの基になるオブジェクト型がNOT FINALであると宣言されると、制御ファイルの列オブジェクトは、基になるオブジェクト型から導出されたサブタイプの(導出および宣言された)属性によって記述されます。データ・ファイルでは、列オブジェクトの各属性に対応するデータは、単純なリレーショナル列に対応するデータ・フィールドと同様の形式でデータ・ファイルに記述されています。

ノート:

SQL*Loader による列オブジェクトなどの複合データ型のサポートによって、制御ファイル内に、2 つの同じフィールド名が存在する可能性があります。1 つは列に対応し、もう 1 つは列オブジェクトの属性に対応する場合です。制御ファイルに同名のフィールドが存在する場合、特定の句がフィールド(たとえば、WHEN、NULLIF、DEFAULTIF、

SID、OID、REF、BFILE など)を参照できるため、名前の重複が発生する場合があります。

したがって、フィールドを参照する句を使用する場合、フルネームを使用する必要があります。たとえば、フィールド fld1 が COLUMN OBJECT であると指定され、フィールド fld2 を含む場合、NULLIF などの句で fld2 を指定するときは、完全なフィールド名 fld1.fld2 を使用する必要があります。

次に、列オブジェクトのロードに関する例を示します。

- ストリーム・レコード形式での列オブジェクトのロード
  ストリーム・レコード形式を使用すると、列オブジェクトにデリミタを指定することで、複数行のフィールドが含まれているレコードをSQL*Loaderでロードできます。
- 可変レコード形式での列オブジェクトのロード
  列オブジェクトを可変レコード形式でロードできます。
- ネストした列オブジェクトのロード
  ネストした列オブジェクトをロードできます。
- 導出サブタイプを使用した列オブジェクトのロード
  導出サブタイプを使用した列オブジェクトをロードできます。
- オブジェクトに対するNULL値の指定
  オブジェクトに対してNULL値を指定できます。
- ユーザー定義コンストラクタを使用した列オブジェクトのロード
  ユーザー定義コンストラクタを使用した列オブジェクトをロードできます。

親トピック: SQL*Loaderを使用したオブジェクト、LOBおよびコレクションのロード

11.1.1 ストリーム・レコード形式への列オブジェクトのロード

ストリーム・レコード形式を使用すると、列オブジェクトにデリミタを指定することで、複数行のフィールドが含まれているレコードをSQL*Loaderでロードできます。

ストリーム・レコード形式の場合、SQL*Loaderはレコードの終了記号をスキャンすることでレコードを形成します。ストリーム・レコード形式の使用方法を示すために、次の例について考えてみます。この例では、事前にサイズが決まっているフィールドにデータがあります。終了文字は、物理レコードの終わりを示します。オペレーティング・システムのファイル処理句(os_file_proc_clause)のカスタム・レコード・セパレータを使用して、物理レコードの終わりを示すこともできます。

例11-1 ストリーム・レコード形式への列オブジェクトのロード

制御ファイルの内容

```
LOAD DATA
INFILE 'example.dat'
INTO TABLE departments
   (dept_no     POSITION(01:03)    CHAR,
    dept_name   POSITION(05:15)    CHAR,
1   dept_mgr    COLUMN OBJECT
      (name     POSITION(17:33)    CHAR,
       age      POSITION(35:37)    INTEGER EXTERNAL,
       emp_id   POSITION(40:46)    INTEGER EXTERNAL) )
```

データ・ファイル(example.dat)

```
101 Mathematics  Johnny Quest       30   1024
237 Physics      Albert Einstein   65   0000
```

この例では、dept_mgr COLUMN OBJECTのコールアウト1に注目してください。このタイプの列オブジェクト指定は、ネストした列オブジェクトの記述に再帰的に適用できます。

親トピック: 列オブジェクトのロード

11.1.2 可変レコード形式への列オブジェクトのロード

可変レコード形式で列オブジェクトをロードできます。

例11-2に、デリミタ付きフィールドにデータがある例を示します。

例11-2 可変レコード形式への列オブジェクトのロード

制御ファイルの内容

```
LOAD DATA
1 INFILE 'sample.dat' "var 6"
INTO TABLE departments
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
2  (dept_no
   dept_name,
   dept_mgr        COLUMN OBJECT
      (name        CHAR(30),
      age          INTEGER EXTERNAL(5),
      emp_id       INTEGER EXTERNAL(5)) )
```

データ・ファイル(sample.dat)

```
3  000034101,Mathematics,Johny Q.,30,1024,
   000039237,Physics,"Albert Einstein",65,0000,
```

ノート:

例の左に付けた太字の数字は、次のノートと対応しています。

1. "var"文字列には、各レコードの先頭にある長さフィールドのバイト数(この例では 6)が含まれます。値が指定されない場合、デフォルトは 5 バイトです。可変レコードの最大サイズは、2^32-1 です。それ以上の値を指定するとエラーになります。

2. 位置を指定しなくても、一般構文では同じ結果(列オブジェクトの名前の後に、カッコで囲まれた属性のリストが続く)になります。また、省略された型指定については、デフォルトで長さが 255 の CHAR 型になります。

3. 最初の 6 バイト(斜体で示した部分)に、次のレコードの長さを指定します。これらの長さ指定には、emp_id フィールドの後の終了記号のために無視される改行文字も含まれます。

親トピック: 列オブジェクトのロード

11.1.3 ネストした列オブジェクトのロード

ネストした列オブジェクトをロードできます。

例11-3に、ネストした列オブジェクト(他の列オブジェクト内にネストした1つの列オブジェクト)の制御ファイルの記述方法を示します。

例11-3 ネストした列オブジェクトのロード

制御ファイルの内容

```
LOAD DATA
INFILE `sample.dat'
INTO TABLE departments_v2
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
   (dept_no      CHAR(5),
   dept_name     CHAR(30),
   dept_mgr      COLUMN OBJECT
      (name      CHAR(30),
      age        INTEGER EXTERNAL(3),
      emp_id     INTEGER EXTERNAL(7),
1     em_contact COLUMN OBJECT
         (name      CHAR(30),
         phone_num  CHAR(20))))
```

データ・ファイル(sample.dat)

```
101,Mathematics,Johny Q.,30,1024,"Barbie",650-251-0010,
237,Physics,"Albert Einstein",65,0000,Wife Einstein,654-3210,
```

ノート:

例の左に付けた太字の数字は、次のノートと対応しています。

1. このエントリでは、列オブジェクトにネストした列オブジェクトを指定します。

親トピック: 列オブジェクトのロード

11.1.4 導出サブタイプを使用した列オブジェクトのロード

導出サブタイプを使用して列オブジェクトをロードできます。

例11-4に、NOT FINALのベース・オブジェクト型を拡張して新しく導出されたサブタイプを作成する例を示します。表定義の列オブジェクトは、ベース・オブジェクト型であると宣言されますが、サブタイプがベース・オブジェクト型から導出される場合は、SQL*Loaderによってサブタイプを列オブジェクトにロードできます。

例11-4 サブタイプを使用した列オブジェクトのロード

オブジェクトの型定義

```
CREATE TYPE person_type AS OBJECT
  (name     VARCHAR(30),
   ssn      NUMBER(9)) not final;
CREATE TYPE employee_type UNDER person_type
  (empid    NUMBER(5));
CREATE TABLE personnel
  (deptno   NUMBER(3),
   deptname VARCHAR(30),
   person   person_type);
```

制御ファイルの内容

```
LOAD DATA
INFILE 'sample.dat'
INTO TABLE personnel
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
```

```
   (deptno        INTEGER EXTERNAL(3),
    deptname      CHAR,
1   person        COLUMN OBJECT TREAT AS employee_type
      (name       CHAR,
       ssn        INTEGER EXTERNAL(9),
2      empid      INTEGER EXTERNAL(5)))
```

データ・ファイル(sample.dat)

```
101,Mathematics,Johny Q.,301189453,10249,
237,Physics,"Albert Einstein",128606590,10030,
```

ノート:

例の左に付けた太字の数字は、次のノートと対応しています。

1. TREAT AS 句で指定すると、SQL*Loader では、実際の宣言型が person_type である列オブジェクト person が、導出型 employee_type であると宣言されたかのように処理されます。

2. empid 属性は employee_type の属性であるため、ここで使用できます。TREAT AS 句が指定されていない場合、この属性は列の宣言型の属性ではないため、エラーが発生します。

親トピック: 列オブジェクトのロード

11.1.5 オブジェクトに対するNULL値の指定

オブジェクトに対してNULL値を指定できます。

非スカラー・データ型に対するNULL値の指定は、スカラー・データ型に対する指定より多少複雑になります。オブジェクトは、その属性のサブセットをNULLにするか、すべての属性をNULL(NULLオブジェクトにかぎります)にするか、またはオブジェクト自身をNULL(アトミックNULLオブジェクト)にできます。

- NULL属性の指定
  NULL属性を指定できます。
- アトミックNULLの指定
  アトミックNULLを指定できます。

親トピック: 列オブジェクトのロード

11.1.5.1 NULL属性の指定

NULL属性を指定できます。

列オブジェクトに対応するフィールドでは、NULLIF句を使用して、特殊属性をNULLに初期化するフィールド条件を指定できます。例11-5に、これを示します。

例11-5 NULLIF句を使用したNULL属性の指定

制御ファイルの内容

```
LOAD DATA
INFILE 'sample.dat'
INTO TABLE departments
  (dept_no       POSITION(01:03)    CHAR,
  dept_name      POSITION(05:15)    CHAR NULLIF dept_name=BLANKS,
```

```
  dept_mgr       COLUMN OBJECT
1    ( name      POSITION(17:33)    CHAR NULLIFdept_mgr.name=BLANKS,
1    age         POSITION(35:37)    INTEGEREXTERNAL NULLIFdept_mgr.age=BLANKS,
1    emp_id      POSITION(40:46)    INTEGEREXTERNAL NULLIF dept_mgr.empid=BLANKS))
```

データ・ファイル(sample.dat)

```
2  101              Johny Quest            1024
   237   Physics   Albert Einstein   65   0000
```

ノート:

例の左に付けた太字の数字は、次のノートと対応しています。

1. 各属性に対応する NULLIF 句は、属性値を NULL にする条件を示します

2. dept_mgr の age 属性の値は NULL です。dept_name の値も NULL です。

親トピック: オブジェクトに対するNULL値の指定

11.1.5.2 アトミックNULLの指定

アトミックNULLを指定できます。

列オブジェクトがNULL値(アトミックNULL)を取る条件を制御ファイルで指定するには、NULLIF句で使用するオブジェクトの名前は、マップ・フィールドの論理的な組合せに基づいている必要があります(たとえば、例11-5では、指定されたマップ・フィールドは、dept_no、dept_name、name、ageおよびemp_idです。dept_mgrは、データ・ファイルのどのフィールドにも対応していない(マップされていない)ため、指定されたマップ・フィールドではありません)。

前述の方法は使用可能ですが、オブジェクトがNULL値を取るための条件がマップ・フィールドに依存していない場合は、理想的な方法ではありません。このような場合は、FILLERフィールドを使用できます。

FILLERフィールドをデータ・ファイルのフィールドにマップし(列オブジェクトがアトミックNULLかどうかを示す)、そのFILLERフィールドを列オブジェクトのNULLIF句のフィールド条件で使用できます。この例を例11-6に示します。

例11-6 FILLERフィールドを使用したデータのロード

制御ファイルの内容

```
LOAD DATA
INFILE 'sample.dat'
INTO TABLE departments_v2
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
   (dept_no          CHAR(5),
   dept_name         CHAR(30),
1  is_null           FILLER CHAR,
2  dept_mgr          COLUMN OBJECT NULLIF is_null=BLANKS
      (name          CHAR(30) NULLIF dept_mgr.name=BLANKS,
      age            INTEGER EXTERNAL(3) NULLIF dept_mgr.age=BLANKS,
      emp_id         INTEGER EXTERNAL(7)
                     NULLIF dept_mgr.emp_id=BLANKS,
      em_contact     COLUMN OBJECT NULLIF is_null2=BLANKS
         (name       CHAR(30)
                     NULLIF dept_mgr.em_contact.name=BLANKS,
         phone_num   CHAR(20)
                     NULLIF dept_mgr.em_contact.phone_num=BLANKS)),
1  is_null2          FILLER CHAR)
```

データ・ファイル(sample.dat)

```
101,Mathematics,n,Johny Q.,,1024,"Barbie",608-251-0010,,
237,Physics,,"Albert Einstein",65,0000,,650-654-3210,n,
```

ノート:

例の左に付けた太字の数字は、次のノートと対応しています。

1. FILLER フィールド(データ・ファイルがマップされており、対応する列がない)は CHAR 型(デリミタ付きフィールドであるため、CHAR のデフォルトは CHAR(255))のフィールドです。NULLIF 句は、FILLER フィールド自体には使用できません

2. is_null フィールドが空白の場合は、NULL(アトミック NULL)の値を取得します。

親トピック: オブジェクトに対するNULL値の指定

11.1.6 ユーザー定義コンストラクタを使用した列オブジェクトのロード

ユーザー定義コンストラクタを使用して列オブジェクトをロードできます。

Oracle Databaseでは、すべてのオブジェクト型に対してデフォルトのコンストラクタが自動的に提供されます。このコンストラクタを使用するには、コンストラクタに対するコールで、その型のすべての属性を引数として指定する必要があります。オブジェクトの新しいインスタンスが作成されると、その属性は引数リスト内の対応する値を取ります。このコンストラクタは、属性値コンストラクタと呼ばれます。SQL*Loaderでは、列オブジェクトのロード時に、デフォルトで属性値コンストラクタが使用されます。

1つ以上のユーザー定義コンストラクタを作成することによって、属性値コンストラクタを上書きできます。ユーザー定義コンストラクタを作成する場合、オブジェクトの新しいインスタンスが作成されると、常に、ユーザー定義論理を実行する型本体を指定する必要があります。ユーザー定義コンストラクタの引数リストは、属性値コンストラクタと同じ場合もありますが、型本体で実装される論理は異なります。

ユーザー定義コンストラクタのファンクションの引数リストが属性値コンストラクタの引数リストと一致する場合、従来型パスとダイレクト・パスではSQL*Loaderの動作が異なります。従来型パス・モードでは、ユーザー定義コンストラクタがコールされます。ダイレクト・パス・モードでは、属性値コンストラクタがコールされます。例11-7に、この違いを示します。

例11-7 一致するコンストラクタを使用した列オブジェクトのロード

オブジェクトの型定義

```
CREATE TYPE person_type AS OBJECT
     (name     VARCHAR(30),
      ssn      NUMBER(9)) not final;
   CREATE TYPE employee_type UNDER person_type
     (empid    NUMBER(5),
   -- User-defined constructor that looks like an attribute-value constructor
      CONSTRUCTOR FUNCTION
        employee_type (name VARCHAR2, ssn NUMBER, empid NUMBER)
        RETURN SELF AS RESULT);
   CREATE TYPE BODY employee_type AS
     CONSTRUCTOR FUNCTION
        employee_type (name VARCHAR2, ssn NUMBER, empid NUMBER)
      RETURN SELF AS RESULT AS
   --User-defined constructor makes sure that the name attribute is uppercase.
      BEGIN
        SELF.name  := UPPER(name);
```

```
        SELF.ssn   := ssn;
        SELF.empid := empid;
        RETURN;
      END;
   CREATE TABLE personnel
     (deptno   NUMBER(3),
      deptname VARCHAR(30),
      employee employee_type);
```

制御ファイルの内容

```
LOAD DATA
   INFILE *
   REPLACE
   INTO TABLE personnel
   FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
      (deptno        INTEGER EXTERNAL(3),
       deptname      CHAR,
       employee      COLUMN OBJECT
         (name       CHAR,
          ssn        INTEGER EXTERNAL(9),
          empid      INTEGER EXTERNAL(5)))
   BEGINDATA
1  101,Mathematics,Johny Q.,301189453,10249,
   237,Physics,"Albert Einstein",128606590,10030,
```

ノート:

例の左に付けた太字の数字は、次のノートと対応しています。

1. この制御ファイルが従来型パス・モードで実行された場合、名前フィールドの Johny Q.および Albert Einstein は、両方とも大文字でロードされます。これは、従来型パス・モードではユーザー定義コンストラクタがコールされるためです。これに対し、この制御ファイルがダイレクト・パス・モードで実行された場合、名前フィールドは入力データに表示されているとおりにロードされます。これは、ダイレクト・パス・モードでは属性値コンストラクタがコールされるためです。

引数リストが属性値コンストラクタと一致しないユーザー定義コンストラクタの作成もできます。この場合は、従来型パス・モードおよびダイレクト・パス・モードの両方で、属性値コンストラクタがコールされます。例 11-8 に示す定義について考えてみます。

例11-8 一致しないコンストラクタを使用した列オブジェクトのロード

オブジェクトの型定義

```
CREATE SEQUENCE employee_ids
    START     WITH  1000
    INCREMENT BY    1;
   CREATE TYPE person_type AS OBJECT
     (name     VARCHAR(30),
      ssn      NUMBER(9)) not final;
   CREATE TYPE employee_type UNDER person_type
     (empid    NUMBER(5),
   -- User-defined constructor that does not look like an attribute-value
   -- constructor
      CONSTRUCTOR FUNCTION
        employee_type (name VARCHAR2, ssn NUMBER)
        RETURN SELF AS RESULT);
   CREATE TYPE BODY employee_type AS
```

```
     CONSTRUCTOR FUNCTION
        employee_type (name VARCHAR2, ssn NUMBER)
      RETURN SELF AS RESULT AS
   -- This user-defined constructor makes sure that the name attribute is in
   -- lowercase and assigns the employee identifier based on a sequence.
        nextid     NUMBER;
        stmt       VARCHAR2(64);
      BEGIN
        stmt := 'SELECT employee_ids.nextval FROM DUAL';
        EXECUTE IMMEDIATE stmt INTO nextid;
        SELF.name  := LOWER(name);
        SELF.ssn   := ssn;
        SELF.empid := nextid;
        RETURN;
      END;
   CREATE TABLE personnel
     (deptno   NUMBER(3),
      deptname VARCHAR(30),
      employee employee_type);
```

例11-7で説明した制御ファイルがこれらの定義に従って使用された場合、名前フィールドは、入力データに表示されているとおりに(大文字と小文字の組合せで)ロードされます。これは、従来型パス・モードおよびダイレクト・パス・モードの両方で属性値コンストラクタがコールされるためです。

SQL式でユーザー定義コンストラクタを明示的に参照することによって、従来型パス・モードを使用してこの表をロードすることもできます。例11-9に、この方法を示します。

例11-9 コンストラクタが一致しない場合のSQLを使用した列オブジェクトのロード

制御ファイルの内容

```
LOAD DATA
   INFILE *
   REPLACE
   INTO TABLE personnel
   FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
      (deptno        INTEGER EXTERNAL(3),
       deptname      CHAR,
       name          BOUNDFILLER CHAR,
       ssn           BOUNDFILLER INTEGER EXTERNAL(9),
1      employee      EXPRESSION "employee_type(:NAME, :SSN)")
   BEGINDATA
1  101,Mathematics,Johny Q.,301189453,
   237,Physics,"Albert Einstein",128606590,
```

ノート:

例の左に付けた太字の数字は、次のノートと対応しています。

1. この制御ファイルが従来型パス・モードで実行された場合、名前フィールドの `Johny Q.`および `Albert Einstein` は、両方とも大文字でロードされます。これは、従来型パス・モードではユーザー定義コンストラクタがコールされるためです。これに対し、この制御ファイルがダイレクト・パス・モードで実行された場合、名前フィールドは入力データに表示されているとおりにロードされます。これは、ダイレクト・パス・モードでは属性値コンストラクタがコールされるためです。

例11-9に示した制御ファイルがダイレクト・パス・モードで使用された場合は、次のエラーが通知されます。

```
SQL*Loader-951: Error calling once/load initialization
```

```
ORA-26052: Unsupported type 121 for SQL expression on column EMPLOYEE.
```

親トピック: 列オブジェクトのロード

11.2 オブジェクト表のロード

オブジェクト表のロードに必要な制御ファイルの構文は、典型的なリレーショナル表のロードの場合とほぼ同じです。

例11-10に、主キーベースのオブジェクト識別子(OID)を使用したオブジェクト表のロード例を示します。

例11-10 主キーOIDを使用したオブジェクト表のロード

制御ファイルの内容

```
LOAD DATA
INFILE 'sample.dat'
DISCARDFILE 'sample.dsc'
BADFILE 'sample.bad'
REPLACE
INTO TABLE employees
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
   (name    CHAR(30)                NULLIF name=BLANKS,
   age      INTEGER EXTERNAL(3)     NULLIF age=BLANKS,
   emp_id   INTEGER EXTERNAL(5))
```

データ・ファイル(sample.dat)

```
Johny Quest, 18, 007,
Speed Racer, 16, 000,
```

前述の制御ファイルを見ただけでは、ロードされる表がシステム生成OIDを持つオブジェクト表か、主キーOIDを持つオブジェクト表か、またはリレーショナル表かを判断できません。

すでにシステム生成OIDを含むデータをロードし、新しいOIDを生成するのではなく、データ・ファイル内の既存のOIDを使用するように指定する必要がある場合もあります。そのような場合は、`INTO TABLE`句に続けて`OID`句を使用します。

```
OID (fieldname)
```

この場合、`fieldname`には、システム生成OIDを含むデータ・フィールドにマップされた、フィールド指定リストのフィールド名(通常はFILLERフィールド)を指定します。SQL*Loaderでは、その指定されたOIDが、正しい形式で、グローバルな独自性を保持したOIDであるとみなされます。そのため、OracleのOIDジェネレータを使用してOIDを生成し、ロードされたOIDの一意性を確保する必要があります。

また、その`OID`句は、主キーOIDではなく、システム生成のOIDでのみ使用できます。

例11-11に、行オブジェクトを使用したシステム生成OIDのロード例を示します。

例11-11 OIDのロード

制御ファイルの内容

```
   LOAD DATA
   INFILE 'sample.dat'
   INTO TABLE employees_v2
1  OID (s_oid)
   FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
      (name    CHAR(30)                NULLIF name=BLANKS,
      age      INTEGER EXTERNAL(3)    NULLIF age=BLANKS,
      emp_id   INTEGER EXTERNAL(5),
2     s_oid    FILLER CHAR(32))
```

データ・ファイル(sample.dat)

```
3  Johny Quest, 18, 007, 21E978406D3E41FCE03400400B403BC3,
   Speed Racer, 16, 000, 21E978406D4441FCE03400400B403BC3,
```

ノート:

例の左に付けた太字の数字は、次のノートと対応しています。

1. OID 句には、s_oid のロード・フィールドが OID を含むように指定します。カッコが必要です
2. s_oid に有効な 16 進数が含まれていない場合、そのレコードは拒否されます。
3. データ・ファイルの OID は文字列で、32 バイトの 16 進数として解釈されます。32 バイトの 16 進数は、後で 16 バイトの RAW に変換されてオブジェクト表に格納されます。

- サブタイプを使用したオブジェクト表のロード
  オブジェクト表の列オブジェクトがNOT FINALに基づいている場合、SQL*Loaderによって導出サブタイプをオブジェクト表にロードできます。

親トピック: SQL*Loaderを使用したオブジェクト、LOBおよびコレクションのロード

11.2.1 サブタイプを使用したオブジェクト表のロード

オブジェクト表の列オブジェクトがNOT FINALに基づいている場合、SQL*Loaderによって導出サブタイプをオブジェクト表にロードできます。

オブジェクト表に導出サブタイプをロードする場合に必要な構文は、典型的なリレーショナル表のロードの場合とほぼ同じです。ただし、この場合、実際に使用するサブタイプがオブジェクト表で有効であるかどうかをSQL*Loaderで判断できるように、サブタイプに名前を指定する必要があります。相違点については、次の例を使用して示します。

例11-12 サブタイプを使用したオブジェクト表のロード

オブジェクト型の定義を確認して、制御ファイルの構成方法を理解するためにコールアウト(bold)を確認してください。

オブジェクトの型定義

```
CREATE TYPE employees_type AS OBJECT
  (name     VARCHAR2(30),
   age      NUMBER(3),
   emp_id   NUMBER(5)) not final;
CREATE TYPE hourly_emps_type UNDER employees_type
  (hours    NUMBER(3));
CREATE TABLE employees_v3 of employees_type;
```

制御ファイルの内容

```
   LOAD DATA
   INFILE 'sample.dat'
   INTO TABLE employees_v3
1  TREAT AS hourly_emps_type
   FIELDS TERMINATED BY ','
     (name     CHAR(30),
      age      INTEGER EXTERNAL(3),
      emp_id   INTEGER EXTERNAL(5),
2     hours    INTEGER EXTERNAL(2))
```

データ・ファイル(sample.dat)

```
Johny Quest, 18, 007, 32,
Speed Racer, 16, 000, 20,
```

ノート:

例の左に付けた太字の数字は、次のノートと対応しています。

1. TREAT AS 句はオブジェクト表を、実際に宣言された型(employee_type)ではなく、hourly_emps_type 型として宣言されたものとして扱うように SQL*Loader に指示します。
2. hours 属性は hourly_emps_type の属性であるため、ここで使用できます。TREAT AS 句が指定されていない場合にこの属性を使用すると、オブジェクト表の宣言された型の属性ではないため、エラーが発生します。

親トピック: オブジェクト表のロード

11.3 SQL*Loaderを使用したREF列のロード

SQL*Loaderは、システム生成OIDのREF列、主キーベースのREF列および主キーを使用可能な有効範囲なしREF列をロードできます。

REFとはOracleの組込みデータ型で、オブジェクト表内のオブジェクトを指す論理的なポインタです。これらのREF列の型ごとに、その型に応じた適切な表名を指定する必要があります。

- REF句における表名の指定
  次の例を使用して、SQL*Loader制御ファイルでのREF句の記述方法を確認し、大/小文字の区別について理解してください。
- システム生成OID REF列
  システム生成のREF列をロードする場合、SQL*LoaderはREF列を構築する実際のOIDが、データとともにデータ・ファイル内にあるとみなします。
- 主キーREF列
  主キーREF列をロードするには、SQL*Loader制御ファイルのフィールドで列名の後にREF句を記述する必要があります。
- 主キーが使用可能な有効範囲なしREF列
  主キーが使用可能な有効範囲なしREF列は、システム生成と主キーの両方のREFを参照できます。

親トピック: SQL*Loaderを使用したオブジェクト、LOBおよびコレクションのロード

11.3.1 REF句における表名の指定

次の例を使用して、SQL*Loader制御ファイルでのREF句の記述方法を確認し、大/小文字の区別について理解してください。

ノート:

この項で説明する内容は、SQL*Loader および Oracle Database のリリースがいずれも 11g リリース 1(11.1)以上の環境にのみ適用されます。SQL*Loader と Oracle Database のどちらか(またはその両方)が以前のリリースの環境には適用されません。

例11-13 SQL*Loader制御ファイルでのREF句の記述

SQL*Loader制御ファイルでは、REF列に対応するフィールドの記述は、列名の後にREF句を記述することによって行います。REF句には、引数として表名とロードされるREF列の型に適用可能な属性を取ります。表名は、(FILLERフィールドを使用して),動的に、または定数として指定できます。また、表名は、スキーマ名の有無にかかわらず指定できます。

REF句で定数として表名を指定するか、FILLERフィールドを使用して表名を指定するかに関係なく、SQL*Loaderでは、この指定を大/小文字の区別があるものとして解釈します。この点に注意していないと、次の問題が発生する可能性があります。

- ユーザーSCOTTが、小文字で表名を引用符で囲まずにtable2という名前の表を作成する場合、その表は次のいずれの方法でもREF句で使用できます。
  - REF(constant 'TABLE2', ...)
  - REF(constant '"TABLE2"', ...)
  - REF(constant 'SCOTT.TABLE2', ...)
- ユーザーSCOTTが、大文字と小文字を組み合せた名前を引用符で囲んで"Table2"という名前の表を作成する場合、その表は次のいずれの方法でもREF句で使用できます。
  - REF(constant 'Table2', ...)
  - REF(constant '"Table2"', ...)
  - REF(constant 'SCOTT.Table2', ...)

いずれの状況においても、constantがFILLERフィールドに置換された場合、同じ値がデータ・セクションに書き込まれると、例に示したとおり値が処理されます。

親トピック: SQL*Loaderを使用したREF列のロード

11.3.2 システム生成OID型のREF列

システム生成のREF列をロードする場合、SQL*LoaderはREF列を構築する実際のOIDが、データとともにデータ・ファイル内にあるとみなします。

REF列に対応するフィールドの記述は、列名の後にREF句を記述することによって行います。

REF句には、引数として表名とOIDを取ります。その引数は、定数として、または(FILLERフィールドを使用して)動的に指定できます。詳細は、SQL*Loaderのref_specの構文を参照してください。

例11-14 システム生成REF列のロード

次の例は、システム生成のOID REF列をロードする方法を示しています。boldのコールアウトに注目してください。

制御ファイルの内容

```
LOAD DATA
INFILE 'sample.dat'
INTO TABLE departments_alt_v2
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
  (dept_no      CHAR(5),
   dept_name    CHAR(30),
1 dept_mgr     REF(t_name, s_oid),
   s_oid        FILLER CHAR(32),
   t_name       FILLER CHAR(30))
```

データ・ファイル(sample.dat)

```
22345, QuestWorld, 21E978406D3E41FCE03400400B403BC3, EMPLOYEES_V2,
23423, Geography, 21E978406D4441FCE03400400B403BC3, EMPLOYEES_V2,
```

ノート:

例の左に付けた太字の数字は、次のノートと対応しています。

1. 指定した表が存在しない場合、レコードは拒否されます。また、dept_mgr フィールド自体には、データ・ファイルのフィールドはマップされません。

関連トピック

- SQL*Loaderの構文図

親トピック: SQL*Loaderを使用したREF列のロード

11.3.3 主キーREF列

主キーREF列をロードするには、SQL*Loader制御ファイルのフィールドで列名の後にREF句を記述する必要があります。

REF句には、カンマで区切ったフィールド名および定数値のリストが引数として必要です。最初の引数は表名で、その後にロードするREF列の基になる主キーOIDを指定する引数を記述します。詳細は、ref_specのSQL*Loader構文を参照してください。

SQL*Loaderでは、引数の順序は、参照されている表で主キーOIDを作成する列の相対順序に一致しているとみなされます。

例11-15 主キーREF列のロード

次の例は、主キーREF列のロード方法を示しています。

制御ファイルの内容

```
LOAD DATA
INFILE 'sample.dat'
INTO TABLE departments_alt
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
 (dept_no        CHAR(5),
 dept_name       CHAR(30),
 dept_mgr        REF(CONSTANT 'EMPLOYEES', emp_id),
 emp_id          FILLER CHAR(32))
```

データ・ファイル(sample.dat)

```
22345, QuestWorld, 007,
23423, Geography, 000,
```

関連トピック

- SQL*Loaderの構文図

親トピック: SQL*Loaderを使用したREF列のロード

11.3.4 主キーが使用可能な有効範囲なしREF列

主キーが使用可能な有効範囲なしREF列によって、システム生成および主キーの両方の型のREFを参照できます。

有効範囲なしのREF列にデータをロードする構文は、システム生成のOID REF列または主キーベースのREF列にデータをロードするときに使用する構文と同じです。

主キーが使用可能な有効範囲なしREF列をロードする場合は、次の制限が適用されます。

- 単一表へのロード中は、この列からシステム生成または主キーのいずれかの型のREFのみを参照できます。両方は参照できません。両方の型の参照を試行すると、データ行が拒否され、参照表名が無効であることを示すエラー・メッセージが表示されます。
- この列に有効範囲なし主キーREFをロードする場合、単一表へのロード中は、1つのオブジェクト表のみ参照できます。複数の有効範囲なし主キーREF(いくつかのREFはオブジェクト表Xを指し、その他のREFはオブジェクト表Yを指す)をロードする場合は、次のいずれかの操作を実行する必要があります。
  - 単一表へのロードを2回実行します。
  - 有効範囲なし主キーREFのオブジェクト表名など、WHEN句によってデータのいくつかの要素が指定されている複数のINTO TABLE句を使用して、ロードを1回実行します。

  これらの方法のいずれも使用しない場合は、データ行が拒否され、参照されている表の名前が無効であることを示すエラー・メッセージが表示されます。
- SQL*Loaderは、コレクション内の有効範囲なし主キーREFをサポートしていません。
- このREF列にシステム生成のREFをロードする場合は、システム生成のOID REF列に適用される制限も適用されます。
- このREF列に主キーREFをロードする場合は、主キーREF列に適用される制限も適用されます。

ノート:

主キーが使用可能な有効範囲なし REF 列の場合、SQL*Loader は最初の有効なオブジェクト表(解析されたもの)を取得します(REF ディレクティブまたはデータ行から取得します)。次に SQL*Loader は、そのオブジェクト表の OID タイプを使用して、単一表ロードで参照可能な REF タイプを判断します。

例11-16 複数のINTO TABLE句を使用した単一ロードの方法

この例では、WHEN句は、有効範囲なし主キーREF表のcust_tblとcust_noのオブジェクト表名で指定される"CUSTOMERS_PK"データからキーを外します。

```
LOAD DATA
INFILE 'data.dat'
INTO TABLE orders_apk
APPEND
when CUST_TBL = "CUSTOMERS_PK"
fields terminated by ","
(
  order_no   position(1)  char,
  cust_tbl FILLER     char,
  cust_no  FILLER     char,
  cust   REF (cust_tbl, cust_no) NULLIF order_no='0'
)
INTO TABLE orders_apk
APPEND
when CUST_TBL = "CUSTOMERS_PK2"
fields terminated by ","
(
  order_no  position(1)  char,
  cust_tbl FILLER     char,
  cust_no  FILLER     char,
```

```
  cust   REF (cust_tbl, cust_no) NULLIF order_no='0'
)
```

親トピック: SQL*Loaderを使用したREF列のロード

11.4 SQL*Loaderを使用したLOBのロード

SQL*Loaderでロードでロードできるラージ・オブジェクト型について調べて、LOBデータのロード方法を示す例を確認してください。

- プライマリ・データ・ファイルからのLOBデータのロード
  プライマリ・データ・ファイルから、内部LOB (BLOB、CLOB、NCLOB)またはXML列をロードできます。
- LOBFILEからのLOBデータのロード
  LOBデータは、非常に長いデータであるため、プライマリ・データ・ファイルからではなく、LOBFILEからロードすると有効です。
- LLSフィールドを含むデータ・ファイルのロード
  データ・ファイルのフィールドがLOBロケーション指定子(LLS)フィールドの場合、LLS句を使用してこのことを示すことができます。

親トピック: SQL*Loaderを使用したオブジェクト、LOBおよびコレクションのロード

11.4.1 プライマリ・データ・ファイルからのLOBデータのロード

プライマリ・データ・ファイルから、内部LOB (BLOB、CLOB、NCLOB)またはXML列をロードできます。

プライマリ・データ・ファイルから内部LOBまたはXML列をロードするには、次の標準SQL*Loader形式を使用できます。

- 事前にサイズが決まっているフィールド
- デリミタ付きフィールド
- Length-Value Pairフィールド
- 事前に決められたサイズのフィールドのLOBデータ
  事前にサイズが決まっているフィールドにLOBをロードすることで、LOBのロードが非常に高速になることと、概念的に単純な形式になることを示します。
- デリミタ付きフィールドのLOBデータ
  SQL*Loaderで同じ列(データ・ファイル・フィールド)内に異なるサイズのLOBをロードするときには、デリミタ付きフィールドの使用を検討してください。
- Length-Value PairフィールドのLOBデータ
  Length-Value Pairフィールドで編成されたLOBデータのロードには、VARCHAR、VARCHARCまたはVARRAWデータ型を使用できます。

親トピック: SQL*Loaderを使用したLOBのロード

11.4.1.1 事前に決められたサイズのフィールドのLOBデータ

事前にサイズが決まっているフィールドにLOBをロードすることで、LOBのロードが非常に高速になることと、概念的に単純な形式になることを示します。

ノート:

ロードする LOB データはサイズが異なるため、サイズが小さいデータ・フィールドに空白を埋め込み、全 LOB データが同じサイズになるようにできます。

事前にサイズが決まっているフィールドを使用してLOBをロードするには、ロードするデータ型としてCHARまたはRAWを使用する必要があります。

例11-17 事前に決められたサイズのフィールドのLOBデータ

制御ファイルとデータ・ファイルのサンプルを確認して、boldのコールアウトに注目してください。

制御ファイルの内容

```
LOAD DATA
INFILE 'sample.dat' "fix 501"
INTO TABLE person_table
   (name        POSITION(01:21)       CHAR,
1  "RESUME"     POSITION(23:500)      CHAR   DEFAULTIF "RESUME"=BLANKS)
```

データ・ファイル(sample.dat)

```
Julia Nayer       Julia Nayer
              500 Example Parkway
              jnayer@us.example.com ...
```

ノート:

例の左に付けた太字の数字は、次のノートと対応しています。

1. DEFAULTIF 句が使用されているため、RESUME が含まれているデータ・フィールドが空の場合、NULL の LOB ではなく、空の LOB になります。ただし、DEFAULTIF ではなく NULLIF 句が使用されている場合は、空のデータ・フィールドではなく NULL になります。

   また、ロード時に、CHAR 以外にも SQL*Loader のデータ型を使用できます。たとえば、BLOB のロード時、RAW データ型を使用する場合があります。

親トピック: プライマリ・データ・ファイルからのLOBデータのロード

11.4.1.2 デリミタ付きフィールドのLOBデータ

SQL*Loaderで同じ列(データ・ファイル・フィールド)内に異なるサイズのLOBをロードするときには、デリミタ付きフィールドの使用を検討してください。

デリミタ付きフィールド形式で、同じ列(データ・ファイルのフィールド)で異なるサイズのLOBを問題なく処理できます。ただし、このような柔軟性によって、SQL*Loaderでデリミタ文字列を探してデータをスキャンする必要があるため、パフォーマンスに影響します。

単一キャラクタのデリミタで、文字列のデリミタを指定する場合は、データ・ファイルの文字セットに注意してください。データ・ファイルの文字セットが制御ファイルの文字セットと異なる場合は、デリミタを16進数の表記法(つまりX'hexadecimal string')で指定できます。デリミタを実際に16進数の表記法で指定する場合は、入力データ・ファイルの文字セット中の有効な文字で指定する必要があります。一方、16進数の表記法で指定しない場合、デリミタは、クライアント(制御ファイル)の文

字セットで指定してください。この場合、デリミタは、SQL*Loaderによってデータ・ファイル内で検索される前に、データ・ファイルの文字セットに変換されます。

次の点に注意してください。

- 文字列デリミタを使用した区切り構文がサポートされます(囲みデリミタを区切りとします)。
- マルチ・キャラクタの囲みデリミタの前に空白は入れられません。
- フィールドがWHITESPACEで終わる場合、先頭の空白は切り捨てられます。

ノート:

SQL*Loader による CLOB データ移動時のフィールドのデフォルト最大長は 255 バイトですが、最大 2GB までの値を指定できます。デリミタ付きフィールドでは、長さが指定されている場合、その長さが最大長として使用されます。値が指定されない場合、デフォルトは 255 バイトです。デリミタ付きで 255 バイトを超える CHAR フィールドの場合、最大長を指定する必要があります。CHAR データ型の詳細は、「CHAR」を参照してください。

例11-18 デリミタ付きフィールドのLOBデータのロード

この例を確認して、デリミタ付きフィールドにLOBデータをロードする方法を理解してください。boldのコールアウトに注目してください。

制御ファイルの内容

```
LOAD DATA
INFILE 'sample.dat' "str '|'"
INTO TABLE person_table
FIELDS TERMINATED BY ','
   (name        CHAR(25),
1  "RESUME"     CHAR(507) ENCLOSED BY '<startlob>' AND '<endlob>')
```

データ・ファイル(sample.dat)

```
Julia Nayer,<startlob>        Julia Nayer
                          500 Example Parkway
                          jnayer@example.com ...   <endlob>
2  |Bruce Ernst, .......
```

ノート:

例の左に付けた太字の数字は、次のノートと対応しています。

1. <startlob>および<endlob>は、囲み文字列です。デフォルトのバイト長セマンティクスでは、CHAR(507)を使用して読み込むことができる LOB の最大長は 507 バイトです。文字長セマンティクスが使用された場合、最大長は 507 文字になります。詳細は、文字長セマンティクスを参照してください。
2. レコード・セパレータ'|'は、<endlob>のすぐ後にあり、その後に改行文字が続く場合、改行は、次のレコードの一部として解釈されます。代替方法は、レコード・セパレータに改行部分を作成することです(たとえば'|¥n'、または 16 進では X'7C0A')。

関連トピック

- 文字長セマンティクス

親トピック: プライマリ・データ・ファイルからのLOBデータのロード

11.4.1.3 Length-Value PairフィールドのLOBデータ

Length-Value Pairフィールドで編成されたLOBデータのロードには、VARCHAR、VARCHARCまたはVARRAWデータ型を使用できます。

Length-Value Pairフィールドでデータをロードすると、デリミタ付きフィールドを使用するよりもパフォーマンスが向上します。ただし、この方法では柔軟性が低下することもあります(たとえば、ロード前に各LOBのLOB長を知っている必要があります)。

例11-19 Length-Value PairフィールドへのLOBデータのロード

制御ファイルとデータ・ファイルの例を確認して、Length-Value PairフィールドにLOBデータをロードする方法を理解してください。boldのコールアウトに注目してください。

制御ファイルの内容

```
  LOAD DATA
1 INFILE 'sample.dat' "str '<endrec>¥n'"
  INTO TABLE person_table
  FIELDS TERMINATED BY ','
     (name       CHAR(25),
2    "RESUME"    VARCHARC(3,500))
```

データ・ファイル(sample.dat)

```
Julia Nayer,479 Julia Nayer
500 Example Parkway
jnayer@us.example.com... <endrec>
3    Bruce Ernst,000<endrec>
```

ノート:

例の左に付けた太字の数字は、次のノートと対応しています。

1. バックスラッシュ・エスケープ文字がサポートされていない場合、例の中でレコード・セパレータとして使用されている文字列は、16 進数の表記法で表現されます。
2. "RESUME"は、CLOB 列に対応するフィールドです。制御ファイルでは、VARCHARC がそのフィールドで、フィールド長は 3 バイト、最大サイズは 500 バイト(バイト長セマンティクスで)です。文字長セマンティクスが使用された場合、長さは 3 文字で最大サイズは 500 文字です。詳細は、「文字長セマンティクス」を参照してください。
3. VARCHARC の length サブフィールドは、0(サブフィールドの値が空)です。このため、LOB インスタンスは、空に初期化されます。

関連トピック

- 文字長セマンティクス

親トピック: プライマリ・データ・ファイルからのLOBデータのロード

11.4.2 LOBFILEからのLOBデータのロード

LOBデータは、非常に長いデータであるため、プライマリ・データ・ファイルからではなく、LOBFILEからロードすると有効です。

LOBFILEでは、LOBデータのインスタンスは、フィールド(事前に決められたサイズ、デリミタ付き、Length-Value Pair)内にあるとみなされますが、これらのフィールドは、レコードに編成されていません(LOBFILEにはレコードの概念がありません)。そのため、レコードを扱うことによって発生するオーバーヘッドを回避できます。このようなデータの編成方法は、LOBのロードにとって理想的です。

LOBFILEからロードしたLOBをメモリーに合わせる必要はありません。SQL*Loaderでは、64KB単位でLOBFILEが読み込まれます。

LOBFILEのデータ・フィールドは、次のいずれかの型です。

- ファイルの内容全体を読み込む単一のLOBフィールド
- サイズが決められたフィールド(固定長フィールド)
- デリミタ付きフィールド(TERMINATED BYまたはENCLOSED BY)

  PRESERVE BLANKS句は、LOBFILEから読み込むフィールドには使用できません。
- Length-Value Pairフィールド(可変長フィールド)

  この型のフィールドからデータをロードするには、VARRAW、VARCHARまたはVARCHARC SQL*Loaderデータ型を使用します。

これらの各フィールド型の使用例については、「LOBFILEからのLOBデータのロードの例」を参照してください。前述のすべてのフィールド型は、XML列のロードに使用できます。

LOBFILE構文については、「lobfile_spec」を参照してください。

- 動的および静的LOBFILE指定
  LOBFILEを静的に指定(制御ファイルにファイル名を指定)するか、または動的に指定(FILLERフィールドをファイル名のソースとして使用)できます。
- LOBFILEからのLOBデータのロードの例
  この項では、LOBFILEの様々なタイプのフィールドからデータをロードする例を示します。
- LOBFILEからLOBをロードする際の考慮事項
  SQL*LoaderでLOBFILESからラージ・オブジェクト型(LOB)をロードする際に適用される制限事項とガイドラインに注意してください。

親トピック: SQL*Loaderを使用したLOBのロード

11.4.2.1 動的および静的LOBFILE指定

LOBFILEを静的に指定(制御ファイルにファイル名を指定)するか、または動的に指定(FILLERフィールドをファイル名のソースとして使用)できます。

いずれの場合も、LOBFILEがEOFで終了しない場合は、LOBFILEの終わりに到達するとファイルがクローズされ、そのファイルからさらにデータを読み込む場合は、空のフィールドからデータを読み込むことになります。

ただし、LOBFILEをEOFで終了する場合は、ファイルからデータを読み込むと、常に、ファイル全体が戻されます。

同じLOBFILEを、2つの異なるフィールドのソースとして指定しないでください。指定すると、通常、2つのフィールドは、データを別々に読み取ります。

親トピック: LOBFILEからのLOBデータのロード

11.4.2.2 LOBFILEからのLOBデータのロードの例

この項では、LOBFILEの異なるフィールド型からデータをロードする例を示します。

- ファイルごとに1つのLOB
  ラージ・オブジェクト型(LOB)データをロードするときには、各LOBFILEが1つのLOBのソースになります。
- 事前に決められたサイズのLOB
  事前に決められたサイズのLOBを使用してLOBデータをロードできます。
- デリミタ付きLOB
  サイズが異なるラージ・オブジェクト型(LOB)には、事前に決められたサイズのLOBが使用できないため、デリミタ付きLOBをSQL*Loaderで使用することを検討してください。
- Length-Value Pairで指定したLOB
  ラージ・オブジェクト型(LOB)のロードにLength-Value Pairの指定を使用するとパフォーマンスが向上します。ただし、柔軟性は低下します。

親トピック: LOBFILEからのLOBデータのロード

11.4.2.2.1 ファイルごとに1つのLOB

ラージ・オブジェクト型(LOB)データをロードするときには、各LOBFILEが1つのLOBのソースになります。

この例を使用して、各LOBFILEが1つのLOBのソースになるように編成されたLOBをロードする方法について理解してください。

例11-20 LOBFILE当たり1つのLOBを使用したLOBデータのロード

この例では、列名またはフィールド名の後にLOBFILEデータ型の指定が続いている点に注目してください。boldのコールアウトに注目してください。

制御ファイルの内容

```
LOAD DATA
INFILE 'sample.dat'
   INTO TABLE person_table
   FIELDS TERMINATED BY ','
   (name      CHAR(20),
1  ext_fname    FILLER CHAR(40),
2  "RESUME"     LOBFILE(ext_fname) TERMINATED BY EOF)
```

データ・ファイル(sample.dat)

```
Johny Quest,jqresume.txt,
Speed Racer,'/private/sracer/srresume.txt',
```

セカンダリ・データ・ファイル(jqresume.txt)

```
Johny Quest 500 Oracle Parkway ...
```

セカンダリ・データ・ファイル(srresume.txt)

```
         Speed Racer
     400 Oracle Parkway
        ...
```

ノート:

例の左に付けた太字の数字は、次のノートと対応しています。

1. FILLER フィールドは、SQL*Loader の CHAR データ型を使用して読み込まれる、40 バイトのデータ・フィールドにマップされています。ここでは、デフォルトのバイト長セマンティクスの使用を想定しています。文字長セマンティクスが使用された場合、フィールドは 40 文字データ・フィールドにマップされます

2. SQL*Loader では、FILLER フィールド ext_fname の LOBFILE 名が使用されます。(CHAR データ型を使用する) LOBFILE の最初のバイトから EOF 文字までのデータがロードされます。既存の LOBFILE が指定されていない場合、RESUME フィールドは空に初期化されます。

親トピック: LOBFILEからのLOBデータのロードの例

11.4.2.2.2 事前に決められたサイズのLOB

事前に決められたサイズのLOBを使用してLOBデータをロードできます。

例11-21では、制御ファイルの特定の列にロードするLOBのサイズを指定します。ロード時、列にロードしたLOBデータは、指定したサイズとみなされます。事前に決められたサイズのフィールドでは、データ解析機能を最適に実行できます。ただし、すべてのLOBデータが必ずしも同じサイズであるとはかぎりません。

例11-21 事前に決められたサイズのLOBを使用したLOBデータのロード

制御ファイルの内容

```
LOAD DATA
INFILE 'sample.dat'
INTO TABLE person_table
FIELDS TERMINATED BY ','
   (name     CHAR(20),
1  "RESUME"    LOBFILE(CONSTANT '/usr/private/jquest/jqresume.txt')
               CHAR(2000))
```

データ・ファイル(sample.dat)

```
Johny Quest,
Speed Racer,
```

セカンダリ・データ・ファイル(jqresume.txt)

```
             Johny Quest
         500 Oracle Parkway
            ...
             Speed Racer
         400 Oracle Parkway
            ...
```

ノート:

例の左に付けた太字の数字は、次のノートと対応しています。

1. このエントリでは、現行のロード・セッション中、最後にロードされたバイト位置に続けてロードを開始し、CHAR データ型を使用して、jqresume.txt LOBFILE から 2000 バイトのデータをロードするように指定しています。ここでは、デフォルトのバイト長セマンティクスの使用を想定しています。文字長セマンティクスが使用された場合、SQL*Loader では、最後にロードされた文字の直後の文字から順番に、2000 文字

のデータがロードされます。詳細は、「文字長セマンティクス」を参照してください。

親トピック: LOBFILEからのLOBデータのロードの例

11.4.2.2.3 デリミタ付きLOB

サイズが異なるラージ・オブジェクト型(LOB)には、事前に決められたサイズのLOBが使用できないため、デリミタ付きLOBをSQL*Loaderで使用することを検討してください。

デリミタ付きのLOBデータ・インスタンスをロードする場合は、同じ列にサイズの異なるLOBをロードしても問題は発生しません。ただし、このような柔軟性によって、SQL*Loaderでデリミタ文字列を探してデータをスキャンする必要があるため、パフォーマンスに影響します。

例11-22 デリミタ付きLOBを使用したLOBデータのロード

この例では、boldのコールアウトに注目してください。

制御ファイルの内容

```
LOAD DATA
INFILE 'sample.dat'
INTO TABLE person_table
FIELDS TERMINATED BY ','
   (name     CHAR(20),
1  "RESUME"    LOBFILE( CONSTANT 'jqresume') CHAR(2000)
               TERMINATED BY "<endlob>¥n")
```

データ・ファイル(sample.dat)

```
Johny Quest,
Speed Racer,
```

セカンダリ・データ・ファイル(jqresume.txt)

```
             Johny Quest
         500 Oracle Parkway
            ... <endlob>
             Speed Racer
         400 Oracle Parkway
            ... <endlob>
```

ノート:

例の左に付けた太字の数字は、次のノートと対応しています。

1. CHAR の最大長に 2000 が指定されているため、SQL*Loader で、フィールドの最大長を推測でき、メモリーの使用量を最適化できます。最大長を指定する場合、小さすぎる値は指定しないように注意してください。TERMINATED BY 句は、LOB を終了する文字列を指定します。かわりに、ENCLOSED BY 句も使用できます。ENCLOSED BY 句を使用すると、LOBFILE 内の LOB の相対的な配置により柔軟性が少し高まります。これは、LOBFILE 内の LOB が連続している必要がないためです。

親トピック: LOBFILEからのLOBデータのロードの例

11.4.2.2.4 Length-Value Pairで指定したLOB

ラージ・オブジェクト型(LOB)のロードにLength-Value Pairの指定を使用するとパフォーマンスが向上します。ただし、柔軟性

は低下します。

Length-Value Pairを指定したLOBの場合は、LOBFILE内の各LOBの前に長さが付加されます。このように編成されたLOBデータのロードには、VARCHAR、VARCHARCまたはVARRAWデータ型を使用できます。

このロード方法を使用すると、デリミタ付きフィールドを使用するより高いパフォーマンスを得ることができます。ただし、柔軟性は損なわれます(たとえば、各LOBのロード前に、LOBの長さの確認が必要です)。

例11-23 Length-Value Pairを指定したLOBを使用したLOBデータのロード

制御ファイルの内容

次の例では、boldのコールアウトに注目してください。

```
LOAD DATA
INFILE 'sample.dat'
INTO TABLE person_table
FIELDS TERMINATED BY ','
   (name           CHAR(20),
1  "RESUME"        LOBFILE(CONSTANT 'jqresume') VARCHARC(4,2000))
```

データ・ファイル(sample.dat)

```
Johny Quest,
Speed Racer,
```

セカンダリ・データ・ファイル(jqresume.txt)

```
2       0501Johny Quest
        500 Oracle Parkway
           ...
3       0000
```

ノート:

例の左に付けた太字の数字は、次のノートと対応しています。

1. VARCHARC(4,2000)のエントリによって、LOBFILE の LOB が Length-Value Pair 形式であり、最初の 4 バイトが長さを示すことを SQL*Loader に指定します。この 2000 という値は、フィールドの最大サイズが 2000 であることを示します。ここでは、デフォルトのバイト長セマンティクスの使用を想定しています。文字長セマンティクスが使用された場合、最初の 4 文字は文字単位の長さとして解釈されます。フィールドの最大サイズは 2000 文字です。詳細は、「文字長セマンティクス」を参照してください。

2. Johny Quest の前の 0501 は、次の 501 文字が LOB のデータであることを示します。

3. このエントリは、LOB が空である(NULL ではない)ことを示します。

親トピック: LOBFILEからのLOBデータのロードの例

11.4.2.3 LOBFILEからLOBをロードする場合の考慮点

SQL*LoaderでLOBFILESからラージ・オブジェクト型(LOB)をロードする際に適用される制限事項とガイドラインに注意してください。

LOBFILEを使用してデータをロードする際には、次の点に注意が必要です。

- LOBおよびXML列のみがLOBFILEからロードできます。
- 特定のLOBのロードが失敗した場合、そのLOBを含むレコードは拒否されません。かわりに、結果は空のLOBを含むレコードになります。XML列の場合、LOBのロードに失敗するとNULL値が挿入されます。
- LOB列に対応するフィールドの最大長を指定する必要はありません。最大長を指定すると、SQL*Loaderでは最大長をメモリー使用量の最適化のヒントに使用します。そのため、最大長の指定では、実際の最大長よりも小さい値を指定しないでください。
- LOBFILEからデータをロード中、位置指定(pos_spec)はできません。
- NULLIFまたはDEFAULTIFのフィールド条件は、LOBFILEから読み込まれたフィールドに基づくことはできません。
- 存在しないLOBFILEをフィールドのデータ・ソースに指定すると、そのフィールドは初期化されて空になります。そのフィールドを空にできない場合は、NULLに初期化されます。
- 表レベル・デリミタは、LOBFILEから読み込まれるフィールドには指定できません。
- 従来型パス・モードでSQL式を使用してXML列をロードするか、LOB列を参照する場合は、SQL*LoaderでLOBデータを一時LOBとして処理する必要があります。こうした場合に最高のロード・パフォーマンスを得られるようにするために、一時LOBのパフォーマンスに関するガイドラインを参照してください。

関連トピック

- 一時LOBパフォーマンスのガイドライン

親トピック: LOBFILEからのLOBデータのロード

11.4.3 LLSフィールドを含むデータ・ファイルのロード

データ・ファイルのフィールドがLOBロケーション指定子(LLS)フィールドの場合、LLS句を使用してこのことを示すことができます。

用途

LLSフィールドには、ファイル名、データ・ファイル内のLOBデータのオフセットおよび長さが含まれます。SQL*Loaderでは、この情報を使用してLOB列のデータを読み取ります。

構文

LLS句の構文は次のとおりです。

使用上のノート

LOBは一部または全体をロードでき、任意の位置から任意の長さで開始できます。SQL LoaderではLLSフィールドの内容は filename.ext.nnn.mmm/であると予期され、各要素の定義は次のとおりです。

- filename.extは、LOBを含むファイルの名前です。
- nnnは、ファイル内のLOBのバイト単位のオフセットです。
- mmmは、バイト単位のLOBの長さです。-1の値は、LOBがNULLであることを示します。0の値は、LOBは存在するものの、空であることを示します。
- スラッシュ(/)は、フィールドの終端となります。

SQL*LoaderパラメータSDF_PREFIXを指定した場合、SQL*LoaderはSDF_PREFIXで指定されたディレクトリでファイル

を検索します。指定しない場合、SQL*Loaderはデータ・ファイルと同じディレクトリを検索します。

次のいずれかに該当する場合、エラーが報告され、行は拒否されます。

- ファイル名に相対パスまたは絶対パスの指定が含まれています。
- ファイルが見つからない、オフセットが無効である、または長さがファイルの終わりを超えています。
- フィールドの内容が予期した書式ではありません。
- LLSフィールドに関連付けられた列のデータ型がCLOB、BLOBまたはNCLOBではありません。

制限事項

- LLSフィールドが制御ファイルの他のフィールドの句(NULLIF句など)によって参照される場合、句の評価に使用される値は、データ・ファイル内の文字列であり、その文字列で示されるファイルのデータではありません。
- LLS句で示されるファイルのデータの文字セットは、データ・ファイルと同じ文字セットとみなされます。
- SQL*Loaderを実行するユーザーには、データ・ファイルへの読取りアクセス権限が必要です。

LLS句の指定例

次に、LLS句を含むSQL*Loader制御ファイルの例を示します。列はLOB型である必要があるため、列指定ではデータ型は必要ありません。

```
LOAD DATA
INFILE *
TRUNCATE
INTO TABLE tklglls
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' TRAILING NULLCOLS
(col1 , col2 NULLIF col1 = '1' LLS)
BEGINDATA
1,"tklglls1.dat.1.11/"
```

親トピック: SQL*Loaderを使用したLOBのロード

11.5 SQL*Loaderを使用したBFILE列のロード

BFILEデータ型には、データベースの外側にあるオペレーティング・システム・ファイルの非構造化バイナリ・データが格納されます。

Oracle BFILEデータ型は、最大サイズ4GBのバイナリ・データへの参照を格納するOracle LOBデータ型です。

BFILE列または属性には、データを含む外部ファイルを示すロケータが格納されます。BFILEとしてロードする必要があるファイルは、ロード時に存在している必要はなく、後で作成できます。BFILEを使用するには、いくつかのデータベース管理タスクの実行が必要になります。ディレクトリ・オブジェクトとBFILEオブジェクトに対する制限もあります。そうした制限には、オペレーティング・システム・ファイルの構成方法に関する要件や、オペレーティング・システムのディレクトリ・パスに関する要件などがあります。Oracle Database 18c以降のリリースでは、BFILEデータ型で使用するディレクトリ・オブジェクトのパス名に、シンボリック・リンクを使用できません。SQL*Loaderでは、必要なオブジェクト(サーバーのファイル・システム上の物理ディレクトリに対する論理的な別名)がすでに作成されていると想定しています。

BFILE列に対応する制御ファイルのフィールドの記述は、列名の後にBFILE句を記述して行います。BFILE句には、引数としてディレクトリ・オブジェクト(server_directoryの別名)名の後にBFILE名が必要です。両方の引数を文字列の定数として指定することも、別のフィールドを通じて動的にロードすることもできます。

次にBFILEのロード例を示します。最初の例ではファイル名のみを動的に指定しますが、2番目の例ではBFILEとディレクトリ・オブジェクトの両方を動的に指定する方法を示します。

例11-24 BFILEを使用したデータのロード: ファイル名のみを動的に指定

制御ファイルの内容を次に示します。ディレクトリ名scott_dir1は引用符に囲まれており、そのまま使用されるため、文字列は大文字にせずそのまま指定します。

```
LOAD DATA
INFILE sample.dat
INTO TABLE planets
FIELDS TERMINATED BY ','
   (pl_id    CHAR(3),
   pl_name   CHAR(20),
   fname     FILLER CHAR(30),
   pl_pict   BFILE(CONSTANT "scott_dir1", fname))
```

データ・ファイルsample.datの内容を次に示します。

```
1,Mercury,mercury.jpeg,
2,Venus,venus.jpeg,
3,Earth,earth.jpeg,
```

例11-25 BFILEを使用したデータのロード: ファイル名およびディレクトリを動的に指定

制御ファイルの内容を次に示します。dnameは、ロードしたファイルに対応するディレクトリ名を含む、データ・ファイルのフィールドにマップされていることに注意してください。

```
LOAD DATA
INFILE sample.dat
INTO TABLE planets
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
   (pl_id    NUMBER(4),
   pl_name   CHAR(20),
   fname     FILLER CHAR(30),
   dname     FILLER CHAR(20),
   pl_pict   BFILE(dname, fname) )
```

データ・ファイルsample.datの内容を次に示します。

```
1, Mercury, mercury.jpeg, scott_dir1,
2, Venus, venus.jpeg, scott_dir1,
3, Earth, earth.jpeg, scott_dir2,
```

関連トピック

- Oracle Database SecureFiles and Large Objects開発者ガイド
- Oracle Database SQL言語リファレンス

親トピック: SQL*Loaderを使用したオブジェクト、LOBおよびコレクションのロード

11.6 コレクション(ネストした表およびVARRAY)のロード

LOBと同様、コレクションも、プライマリ・データ・ファイル(データ・インライン)またはSDF(データ・アウトライン)のいずれかからロードできます。

コレクション・データをロードする場合、コレクションに属するデータのインスタンスが終了したことをSQL*Loaderに伝える機能が必要です。これには、2つの方法があります。

- それぞれのネストした表またはVARRAYインスタンスにロードされる行または要素の数を指定するには、DDL構文のCOUNT関数を使用します。COUNTに指定する値は、数字または数字を含む文字列のいずれかで、制御ファイルでCOUNT句よりも先に記述する必要があります。この位置の依存性は、COUNT句に固有です。COUNT(0)または

COUNT(cnt_field)を指定して、カレント行のcnt_fieldが0の場合は、NULLIF句によって上書きされないかぎり、空のコレクション(NULLではない)になります。詳細は、「count_spec」を参照してください。

制御ファイルでCOUNT句によってフィールドが指定され、そのフィールドがカレント行でNULLに設定されている場合、そのカウントを使用するコレクションは、カレント行でも空に設定されます。

- TERMINATED BYおよびENCLOSED BY句を使用して、一意のコレクション・デリミタを指定します。SDF句が使用されている場合、この方法は使用できません。

制御ファイルでは、コレクションは、列オブジェクトと同様に記述します。一部、次のような相違点があります。

- コレクションの記述には、前述の2つの機能を使用します。
- コレクションの記述には、セカンダリ・データ・ファイル(SDF)を指定できます。
- 同じSDFのフィールド上にないかぎり、NULLIFまたはDEFAULTIF句ではSDFのフィールドを参照できません。
- フィールド名を引数として使用する句には、同じコレクションのフィールドに対するDDL指定でないかぎり、コレクション内のフィールド名を使用できません。
- フィールド・リストには、非FILLERフィールドが1つと、複数のFILLERフィールドが含まれている必要があります。VARRAYが列オブジェクトのVARRAYの場合、列オブジェクトの属性は、ネストしたフィールド・リストに記述されます。

- ネストした表およびVARRAYでの制限事項
  ネストした表およびVARRAYに対しては制限があります。
- セカンダリ・データ・ファイル(SDF)
  大きなネストした表とVARRAYのロードが必要なときには、セカンダリ・データ・ファイル(SDF)を使用できます。概念としてはプライマリ・データ・ファイルと同様です。

関連項目:

SDFの詳細は、「セカンダリ・データ・ファイル(SDF)」を参照してください。

列オブジェクトのロードの詳細は、「列オブジェクトのロード」を参照してください。

親トピック: SQL*Loaderを使用したオブジェクト、LOBおよびコレクションのロード

11.6.1 ネストした表およびVARRAYでの制限事項

ネストした表およびVARRAYに対しては制限があります。

ネストした表およびVARRAYには次の制限事項があります。

- field_listには、collection_fld_specを含めることはできません。
- VARRAY内にネストしたcol_obj_specには、collection_fld_specを含めることはできません。
- field_listの一部として指定したcolumn_nameは、VARRAYパラメータを前に付けたcolumn_nameと同一である必要があります。

また、ネストした表を含む表にロードする場合、複数のロードへの細分化およびSIDの生成は自動的に行われないため注意してください。

例11-26に、VARRAYおよびネストした表のロード例を示します。

例11-26 VARRAYおよびネストした表のロード

制御ファイルの内容

```
   LOAD DATA
   INFILE 'sample.dat' "str '¥n' "
   INTO TABLE dept
   REPLACE
   FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
   (
     dept_no       CHAR(3),
     dname         CHAR(25) NULLIF dname=BLANKS,
1    emps          VARRAY TERMINATED BY ':'
     (
       emps        COLUMN OBJECT
       (
         name      CHAR(30),
         age       INTEGER EXTERNAL(3),
2        emp_id    CHAR(7) NULLIF emps.emps.emp_id=BLANKS
     )
   ),
3   proj_cnt      FILLER CHAR(3),
4   projects      NESTED TABLE SDF (CONSTANT "pr.txt" "fix 57") COUNT (proj_cnt)
  (
    projects    COLUMN OBJECT
    (
      project_id        POSITION (1:5) INTEGER EXTERNAL(5),
      project_name      POSITION (7:30) CHAR
                        NULLIF projects.projects.project_name = BLANKS
    )
  )
)
```

データ・ファイル(sample.dat)

```
 101,MATH,"Napier",28,2828,"Euclid", 123,9999:0
 210,"Topological Transforms",:2
```

セカンダリ・データ・ファイル(SDF)(pr.txt)

```
21034 Topological Transforms
77777 Impossible Proof
```

ノート:

例の左に付けた太字の数字は、次のノートと対応しています。

1. TERMINATED BY 句では、VARRAY のインスタンス終了記号(COUNT 句は使用されていないことに注意)を指定します。
2. この FILLER フィールドの存在によって発生するフィールド名の競合は、フルネームによるフィールド参照(ドット表記法を使用)によって解決されます。
3. proj_cnt は、COUNT 句に対する引数として使用する FILLER フィールドです。
4. このエントリでは、次の内容が指定されます。
   - pr.txt と呼ばれる SDF をデータのソースとして指定します。また、SDF 内で固定レコード形式を指定します。

- COUNT 句が 0 の場合、コレクションは空に初期化されます。コレクションを空に初期化するもう 1 つの方法は、DEFAULTIF 句を使用する方法です。ネストした表のフィールドの記述に対応するメイン・フィールド名は、そのネストした非 FILLER フィールドのフィールド名、特に、列オブジェクトのフィールド名の記述と同じです。

親トピック: コレクション(ネストした表およびVARRAY)のロード

11.6.2 セカンダリ・データ・ファイル(SDF)

大きなネストした表とVARRAYのロードが必要なときには、セカンダリ・データ・ファイル(SDF)を使用できます。概念としてはプライマリ・データ・ファイルと同様です。

プライマリ・データ・ファイルと同様に、SDFは、レコードおよびフィールドで構成されたレコードの集まりです。SDFは、制御ファイルのフィールドごとに指定されます。これは、大きいネストした表およびVARRAYをロードする場合に有効です。

ノート:

SDF をデータ・ソースとして命名できるのは、collection_fld_spec のみです。

SDFを指定するには、SDFパラメータを使用します。SDFパラメータの後に、ファイル指定文字列、または1つ以上のファイル指定文字列を含むデータ・フィールドにマップされたFILLERフィールドを指定します。

プライマリ・データ・ファイルの場合、各SDFについて次を指定できます。

- レコード形式(固定、ストリームまたは可変)の指定。また、ストリーム・レコード形式が使用される場合、レコード・セパレータを指定できます。
- レコード・サイズの指定。
- CHARACTERSET句を使用した、SDFの文字セットの指定(「異なる文字コード体系の処理」を参照)。
- 特にSDF指定(SDF指定を含むコレクションのすべてのメンバー・フィールドまたは属性で、LOBFILEフィールドを含むフィールドを除く)のあるフィールドに対するデフォルトのデリミタの指定(デリミタ指定を使用)

SDFについては、次のことにも注意してください。

- 存在しないSDFをフィールドのデータ・ソースに指定すると、そのフィールドは初期化されて空になります。そのフィールドを空にできない場合は、NULLに初期化されます。
- 表レベル・デリミタは、SDFから読み込まれるフィールドには指定できません。
- 64KBを超えるSDFをロードするには、READSIZEパラメータを使用してより大きな物理レコード・サイズを指定できます。コマンドラインからでも、OPTIONS句の一部としてでもREADSIZEパラメータを指定できます。

関連項目:

- READSIZE
- OPTIONS Clause
- sdf_spec

親トピック: コレクション(ネストした表およびVARRAY)のロード

11.7 動的または静的SDF指定の選択

SQL*Loaderを使用すると、SDFを静的に指定(実際のファイル名を指定)したり、動的に指定(FILLERフィールドをファイル名のソースとして使用)したりできます。

動的または静的SDF指定では、SDFのend-of-file (EOF)に達するとファイルが閉じられます。そのファイルから、さらにデータを読み取ってみると、空のフィールドからデータを読み取ることと同じ結果が得られます。

動的セカンダリ・ファイル指定では、動作は多少異なります。参照ファイルの指定が変更されると、常に、古いファイルはクローズされ、データは新しい参照先ファイルの最初から読み込まれます。

データ・ソース・ファイルの動的な切替えは、リセットの効果があります。たとえば、SQL*Loaderを使用して、現行のファイルから前回オープンしていたファイルに切り替える場合、前回オープンしていたファイルを再オープンし、そのファイルの最初からデータが読み込まれます。

同じSDFを、2つの異なるフィールドのソースとして指定しないでください。指定すると、通常、2つのフィールドは、データを別々に読み取ります。

親トピック: SQL*Loaderを使用したオブジェクト、LOBおよびコレクションのロード

11.8 親表を子表から分割してのロード

ネストした表の列を含む表をロードする場合、親表を子表から分割してロードする場合があります。

SID(システム生成またはユーザー定義)がロード時にわかっている場合(SIDがデータとともにデータ・ファイルにある場合)、親表と子表を別々にロードできます。

次の例は、ユーザー指定SIDを持つ親および子表をロードする方法を示しています。

例11-27 ユーザー定義SIDを使用した親表のロード

制御ファイルの内容

```
   LOAD DATA
   INFILE 'sample.dat' "str '|¥n' "
   INTO TABLE dept
   FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
   TRAILING NULLCOLS
   ( dept_no   CHAR(3),
   dname       CHAR(20) NULLIF dname=BLANKS ,
   mysid       FILLER CHAR(32),
1  projects    SID(mysid))
```

データ・ファイル(sample.dat)

```
101,Math,21E978407D4441FCE03400400B403BC3,|
210,"Topology",21E978408D4441FCE03400400B403BC3,|
```

ノート:

例の左に付けた太字の数字は、次のノートと対応しています。

1. mysid は、実際の SID を含むデータ・ファイルのフィールドにマップされている FILLER フィールドで、

SID 句に対する引数として指定できます。

例11-28 ユーザー定義SIDを使用した子表のロード

制御ファイルの内容

```
   LOAD DATA
   INFILE 'sample.dat'
   INTO TABLE dept
   FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
   TRAILING NULLCOLS
1  SID(sidsrc)
   (project_id     INTEGER EXTERNAL(5),
   project_name   CHAR(20) NULLIF project_name=BLANKS,
   sidsrc FILLER  CHAR(32))
```

データ・ファイル(sample.dat)

```
21034, "Topological Transforms", 21E978407D4441FCE03400400B403BC3,
77777, "Impossible Proof", 21E978408D4441FCE03400400B403BC3,
```

ノート:

例の左に付けた太字の数字は、次のノートと対応しています。

1. 表レベルの SID 句を指定すると、SQL*Loader によって、ネストした表の記憶表がロードされます。sidsrc は、実際の SID のソースである、FILLER フィールド名です。

- VARRAY列ロード時のメモリーの問題
  VARRAY列のロード時にメモリーの問題が発生する場合があります。

親トピック: SQL*Loaderを使用したオブジェクト、LOBおよびコレクションのロード

11.8.1 VARRAY列ロード時のメモリーの問題

VARRAY列のロード時にメモリーの問題が発生する場合があります。

次に、VARRAY列をロードする場合の注意事項を示します。

- VARRAYは、データベースにロードされる前にクライアント・メモリーに作成されます。VARRAYの各要素には、データベースにロードする前に、4バイトのクライアント・メモリーが必要です。そのため、1000個の要素を持つVARRAYをロードする場合は、VARRAYをデータベースにロードする前に、それぞれのVARRAYインスタンスに、4000バイト以上のクライアント・メモリーが必要です。多くの場合、SQL*Loaderでは、VARRAYの構築やロードに、2倍から3倍のメモリー量を必要とします。
- BINDSIZEパラメータを使用して、SQL*Loaderでレコードのロードに割り当てるメモリーの量を指定します。BINDSIZEに指定された値に応じて、SQL*Loaderで、ロード中の各フィールドのサイズを考慮し、1回のトランザクションでロードできる行数を判断します。行数が多ければトランザクションは少なくなり、パフォーマンスは向上します。ただし、システムのメモリー量に制限がある場合、パフォーマンスを優先せず、SQL*Loaderで算出した値より低い値をROWSに指定できます。
- 非常に大きいVARRAYまたは多数の小さいVARRAYが原因で、ロード中にメモリーが不足する場合があります。この場合は、BINDSIZEまたはROWSにより小さい値を指定し、再ロードします。

親トピック: 親表を子表から分割してのロード

12 従来型パス・ロードおよびダイレクト・パス・ロード

SQL*Loaderには、従来のパス・ロード方法とダイレクト・パス・ロード方法を使用してデータをロードするオプションがあります。

- データのロード方法
  SQL*Loaderでは、従来型パス・ロードまたはダイレクト・パス・ロードのいずれかを使用してデータをロードできます。
- ROWID列のロード
  従来型パス・ロードおよびダイレクト・パス・ロードの両方において、ROWID列のテキスト値を指定できます。
- 従来型パス・ロード
  従来型パス・ロード(デフォルト)では、SQL*LoaderはSQL INSERT文とバインド配列バッファを使用して、データをデータベース表にロードします。
- ダイレクト・パス・ロード
  SQL*Loaderのダイレクト・パス・ロードの概要、ダイレクト・パス・ロードを使用してデータを渡すタイミングと方法、この機能に適用される制限について説明します。
- ダイレクト・パス・ロードの使用
  SQL*Loaderのダイレクト・パス・ロード方法を使用してデータをロードする方法について説明します。
- ダイレクト・パス・ロードのパフォーマンスの最適化
  SQL*Loaderダイレクト・パス・ロードの高速実行を可能にし、使用する領域を減らす方法について説明します。
- 複数CPUシステムのダイレクト・パス・ロードの最適化
  複数CPUシステムでダイレクト・パス・ロードを実行する場合、SQL*Loaderではデフォルトでマルチスレッドが使用されます。この場合の複数CPUシステムは、2つ以上のCPUを持つ単一のシステムとして定義されます。
- 索引メンテナンスの回避
  従来型パスとダイレクト・パスの両方について、SQL*Loaderでは表のすべての既存の索引がメンテナンスされます。
- ダイレクト・パス・ロード、整合性制約およびトリガー
  従来型パス・ロードと比較して、ダイレクト・パス・ロードを使用したトリガーの設定方法が異なる場合があります
- ダイレクト・パス・ロードのパフォーマンスの最適化
  SQL*Loaderダイレクト・パス・ロードの高速実行を可能にし、使用する領域を減らす方法について説明します。
- 一般的なパフォーマンス改善のヒント
  パラレル・データ・ロードでSQL*Loaderを使用する場合に一般的なパフォーマンス改善を有効にする方法について説明します。

関連トピック

- SQL*Loaderの事例

親トピック: SQL*Loader

12.1 データのロード方法

SQL*Loaderでは、従来型パス・ロードまたはダイレクト・パス・ロードのいずれかを使用してデータをロードできます。

従来型パス・ロードでは、SQL INSERT文を実行してOracle Databaseに表を移入します。ダイレクト・パス・ロードでは、Oracleデータ・ブロックをフォーマットしてからデータ・ブロックを直接データ・ファイルに書き込むことによって、Oracle Databaseのオーバーヘッドを大幅に削減します。ダイレクト・パス・ロードでは、データベース・リソースに対して他のユーザーとの競合が発生しないため、ディスク速度に近い速度でデータをロードできます。

ロードする表は、すでにデータベースに存在する必要があります。SQL*Loaderでは、表は作成されません。すでにデータが含ま

れているか、または空である既存の表にロードされます。

ロードには次の権限が必要です。

- ロードする表についてのINSERT権限。
- REPLACEオプションまたはTRUNCATEオプションを使用して古いデータを削除してから新しくデータをロードする場合には、ロードする表についてのDELETE権限。

関連トピック

- 従来型パス・ロード
- ダイレクト・パス・ロード

親トピック: 従来型パス・ロードおよびダイレクト・パス・ロード

12.2 ROWID列のロード

従来型パス・ロードおよびダイレクト・パス・ロードの両方において、ROWID列のテキスト値を指定できます。

これは、SELECT ROWID FROM table_nameの処理の実行時に取得するテキストと同じです。ROWIDの文字列解釈は、表内の列に対してはROWID型に変換されます。

親トピック: 従来型パス・ロードおよびダイレクト・パス・ロード

12.3 従来型パス・ロード

従来型パス・ロード(デフォルト)では、SQL*LoaderはSQL INSERT文とバインド配列バッファを使用して、データをデータベース表にロードします。

SQL*Loaderで従来型パス・ロードを実行する場合、バッファ・リソースに関して他のすべてのプロセスと同等の処理が行われるため、競合が発生します。このメソッドを使用すると、ロードにかなりの時間がかかります。また、SQL文が生成され、Oracle Databaseに渡されてから実行されるため、さらにオーバーヘッドが発生します。

挿入が発生すると、常に、Oracle Databaseで空き領域のあるブロック(ディスク内に散在して、部分的に書込み可能なブロック)が検索され、そこにデータが書き込まれます。通常のデータベース使用の場合はそれほどでもありませんが、このメソッドは大量データのロード速度を大幅に低下させることがあります。

- 単一パーティションの従来型パス・ロード
  SQL*Loaderでは、INSERT文のパーティション拡張構文を使用します。
- 従来型パスを使用する場合
  ダイレクト・パス・ロードではなく従来型のパス・ロードを使用する場合を決定するには、ユースケース・シナリオのオプションを確認します。

関連トピック

- 中断された従来型パス・ロード

親トピック: 従来型パス・ロードおよびダイレクト・パス・ロード

12.3.1 単一パーティションの従来型パス・ロード

SQL*Loaderでは、INSERT文のパーティション拡張構文を使用します。

従来型パス・ロードでは、SQL INSERT文を使用します。ただし、従来型パスで単一パーティションに対してロードする場合は、

SQL*Loaderで次のような形式のINSERT文のパーティション拡張構文を使用します。

```
INSERT INTO TABLE T PARTITION (P) VALUES ...
```

OracleカーネルのSQLレイヤーでは、挿入される行が指定のパーティションに対応するかどうかを判断します。行が指定のパーティションに対応しない場合、その行は拒否され、そのことを示すエラー・メッセージがSQL*Loaderログ・ファイルに記録されます。

親トピック: 従来型パス・ロード

12.3.2 従来型パスを使用する場合

ダイレクト・パス・ロードではなく従来型のパス・ロードを使用する場合を決定するには、ユースケース・シナリオのオプションを確認します。

ロードを高速にするには、従来型パス・ロードよりダイレクト・ パス・ロードを使用します。ただし、ダイレクト・パス・ロードにはいくつかの制限があるため、従来型パス・ロードを使用することが必要な場合もあります。次のような場合には、従来型パス・ロードを使用します。

- ロードと並行して索引付き表にアクセスする場合、またはロードと並行して索引なしの表に挿入または更新を行う場合。

  ノート: ダイレクト・パス・ロード(パラレル・ロードは除く)を使用するには、SQL*Loaderに、表への排他的書込み権限と、すべての索引への排他的読取り権限および書込み権限が必要です。

- データをクラスタ表にロードする場合。

  理由: ダイレクト・パス・ロードでは、クラスタ表に対するロードをサポートしていません。

- 比較的少数の行を索引付きの大きな表にロードする場合。

  理由: ダイレクト・パス・ロードでは、既存の索引は新しい索引キーとマージするときにコピーされます。既存の索引が非常に大きく、新しいキーの数が非常に少ない場合は、索引をコピーする時間が、ダイレクト・パス・ロードで節約できる時間を相殺してしまうことがあります。

- 参照整合性制約および列チェック整合性制約のある大きな表に、比較的少数の行をロードする場合。

  理由: これらの制約は、ダイレクト・パスでロードした行には適用できないため、ロードが継続している行には適用されません。そして、ロードが完了した時点で表全体に適用されます。表が非常に大きく、新しい行の数が少ない場合は、この処理にかかる時間が節約した時間より多くなる可能性があります。

- レコードのロード時に、次のような状況でレコードが拒否されることを確認する場合。
  - レコードの挿入でOracleエラーが発生する場合
  - レコードが間違ってフォーマットされたため、SQL*Loaderでフィールドの境界を見つけられない場合。
  - レコードが制約に違反した場合、または一意の索引を非一意にしようとした場合

親トピック: 従来型パス・ロード

12.4 ダイレクト・パス・ロード

SQL*Loaderのダイレクト・パス・ロードの概要、ダイレクト・パス・ロードを使用してデータを渡すタイミングと方法、この機能に適用される制限を学習します。

- SQL*Loaderダイレクト・パス・ロードについて
  SQL*Loaderダイレクト・パス・ロード・オプションでは、ダイレクト・パスAPIを使用して、サーバーのロード・エンジンに

ロードされるデータを渡します。

- ダイレクト・パス・ロード時のデータ変換
  SQL*Loaderのダイレクト・パス・ロード時には、サーバー側ではなくクライアント側でデータ変換が発生します。
- パーティション表またはサブパーティション表のダイレクト・パス・ロード
  パーティション表またはサブパーティション表をロードする場合、SQL*Loaderによって、行がパーティション化され、索引がメンテナンスされます(索引もパーティション化できます)。
- 単一パーティションまたはサブパーティションのダイレクト・パス・ロード
  ダイレクト・パスで単一パーティションに対してロードする場合、SQL*LoaderではLOAD文のパーティション拡張構文を使用します。
- ダイレクト・パス・ロードの利点
  通常、ダイレクト・パス・ロードは従来型パス・ロードを使用するよりも高速です。
- ダイレクト・パス・ロードの使用の制限
  ダイレクト・パス・ロード方式を使用するには、表およびセグメントが特定の要件を満たしている必要があります。一部の機能は、ダイレクト・パス・ロードでは使用できません。
- 単一パーティションのダイレクト・パス・ロードでの制限
  単一パーティションのダイレクト・パス・ロードを使用する場合は、ダイレクト・パス・ロードに指定するパーティションが追加の要件を満たす必要があります。
- ダイレクト・パスを使用する場合
  SQL*Loaderをダイレクト・パス・ロードで実行する必要がある状況について説明します。
- 整合性制約
  ダイレクト・パス・ロード時には、すべての整合性制約が適用されます。ただし、すべての制約が同時に適用されるとはかぎりません。
- ダイレクト・パスのフィールド・デフォルト
  ダイレクト・パス・ロードを使用する場合、データベースに定義されているデフォルトの列指定は使用できません。
- シノニムへのロード
  ダイレクト・パス・ロードでは、SQL*Loaderを使用して表のシノニムにデータをロードできます。ただし、そのシノニムが表または単純な表のビューを直接指している必要があります。

親トピック: 従来型パス・ロードおよびダイレクト・パス・ロード

12.4.1 SQL*Loaderダイレクト・パス・ロードについて

SQL*Loaderダイレクト・パス・ロード・オプションでは、ダイレクト・パスAPIを使用して、サーバーのロード・エンジンにロードされるデータを渡します。

SQL*Loaderのダイレクト・パス・ロード機能を使用すると、バインド配列バッファに書き込むかわりにSQL INSERT文を使用して、バインド配列をOracle Databaseに渡します。ダイレクト・パス・ロードは、ダイレクト・パスAPIを使用して、ロードされるデータをサーバーのロード・エンジンに渡します。ロード・エンジンは、渡されたデータから列配列構造体を作成します。

ダイレクト・パス・ロード・エンジンは、列配列構造体を使用してOracle Databaseのデータ・ブロックをフォーマットし、索引キーを作成します。新しくフォーマットされたデータベース・ブロックが直接データベースに書き込まれます(ホスト・プラットフォームが非同期I/Oをサポートしている場合、非同期書込みを使用して1つのI/O要求に対して複数のブロックを書き込むことができます)。

内部的には、フォーマットされたブロック用に複数のバッファが使用されます。ホスト・プラットフォームで非同期I/Oが可能な場合は、あるバッファに書き込んでいる間に1つ以上のバッファへの書込みが行われます。この場合、I/Oを伴う処理がオーバーラップするため、ロード・パフォーマンスが向上します。

関連トピック

- 中断されたダイレクト・パス・ロード

親トピック: ダイレクト・パス・ロード

12.4.2 ダイレクト・パス・ロード時のデータ変換

SQL*Loaderのダイレクト・パス・ロード時には、サーバー側ではなくクライアント側でデータ変換が発生します。

クライアント側のデータ変換が意味しているのは、データベース初期化パラメータ・ファイルのNLSパラメータ(サーバー側の言語ハンドル)が使用されないことです。この動作を変更するには、SQL*Loaderの制御ファイルに書式マスクを指定するか(初期化パラメータ・ファイルでのNLSパラメータの設定と同じ)、または適切な環境変数を設定できます。たとえば、フィールドに日付書式を指定するには、「SQL*Loaderの制御ファイルの日付書式の設定」に示されているように、SQL*Loaderの制御ファイルに日付書式を設定するか、または「NLS_DATE_FORMAT環境変数の設定」に示されているようにNLS_DATE_FORMAT環境変数を設定できます。

例12-1 SQL*Loaderの制御ファイルに対する日付書式の設定

```
LOAD DATA
INFILE 'data.dat'
INSERT INTO TABLE emp
FIELDS TERMINATED BY "|"
(
EMPNO NUMBER(4) NOT NULL,
ENAME CHAR(10),
JOB CHAR(9),
MGR NUMBER(4),
HIREDATE DATE 'YYYYMMDD',
SAL NUMBER(7,2),
COMM NUMBER(7,2),
DEPTNO NUMBER(2)
)
```

例12-2 NLS_DATE_FORMAT環境変数の設定

UnixのBourneまたはKornシェルの場合:

```
% NLS_DATE_FORMAT='YYYYMMDD'
% export NLS_DATE_FORMAT
```

UnixのCシェル(csh):

```
%setenv NLS_DATE_FORMAT='YYYYMMDD'
```

親トピック: ダイレクト・パス・ロード

12.4.3 パーティション表またはサブパーティション表のダイレクト・パス・ロード

パーティション表またはサブパーティション表へロードする場合は、SQL*Loaderによって、行がパーティション化され、索引(索引もパーティション化できます)がメンテナンスされます。

パーティション表またはサブパーティション表のダイレクト・パス・ロードは、パーティションまたはサブパーティションの多い表の場合、非常に多くのリソースを使用することに注意してください。

ノート:

複数のパーティションに対してダイレクト・パス・ロードを実行する場合に領域エラーが発生すると、ロードは直前のコミット・ポイントにロールバックされます。コミット・ポイントが存在しない場合、ロード全体がロールバックされます。これによって、領域エラー後に検出されたデータが異なるパーティションに書き込まれることがなくなります。

ROWS パラメータを使用して、コミット・ポイントの頻度を指定できます。ROWS パラメータが指定されていない場合、ロード全体がロールバックされます。

親トピック: ダイレクト・パス・ロード

12.4.4 単一パーティションまたはサブパーティションのダイレクト・パス・ロード

ダイレクト・パスで単一パーティションに対してロードする場合、SQL*LoaderではLOAD文のパーティション拡張構文を使用します。

パーティション表またはサブパーティション表の単一パーティションをロードするとき、SQL*Loaderによって行がパーティション化され、SQL*Loader制御ファイルに指定されたパーティションまたはサブパーティションにマップされていない行はすべて拒否されます。ロードされるデータのパーティションまたはサブパーティションに対応するローカル索引パーティションは、SQL*Loaderによってメンテナンスされます。単一パーティションまたはサブパーティションのダイレクト・パス・ロードでは、グローバル索引はメンテナンスされません。ただし、ダイレクト・パスで単一パーティションに対してロードする場合、SQL*Loaderでは次のような形式のLOAD文のパーティション拡張構文を使用します。

```
LOAD INTO TABLE T PARTITION (P) VALUES ...
LOAD INTO TABLE T SUBPARTITION (P) VALUES ...
```

パーティション表またはサブパーティション表の1つのパーティションをロードしている間も、その表の他のパーティションに対してはDML操作やダイレクト・パス・ロードを行うことができます。

ダイレクト・パス・ロードはデータベース処理を最小化しますが、ロードを初期化してから終了するために、ロードの開始時と終了時にOracle Databaseに対する何回かのコールが必要です。また、ロードの初期化中に特定のDMLロックが必要です。ロードが完了すると、これらのDMLロックは解除されます。ロード中に次の操作が発生します。

- 索引キーが作成されてソートされます
- 領域管理ルーチンが、必要に応じて新しいエクステントを取得し、データ・セーブポイントの上限(最高水位標)を調整するために使用されます。

データの保護の詳細は、「データ・セーブを使用したデータ損失の防止」を参照してください。

関連トピック

- データ・セーブを使用したデータ損失の防止

親トピック: ダイレクト・パス・ロード

12.4.5 ダイレクト・パス・ロードのメリット

通常、ダイレクト・パス・ロードは従来型パス・ロードを使用するよりも高速です。

ダイレクト・パス・ロードは、従来型パス・ロードよりも処理が高速です。その理由は次のとおりです。

- 一部使用中の部分ブロックを使用しないため、空きブロックを検索するための読取り処理が不要で、書込みも少なくなります。
- SQL*LoaderではSQL INSERT文を実行する必要がないため、Oracle Databaseでの処理負荷が軽減されます。

- 表と索引をロード開始時にロックするようにOracle Databaseに要求し、ロード完了時にロック解除を要求します。従来型パス・ロードでは、行配列ごとにOracle Databaseコールを1回発行して、SQL INSERT文を処理します。
- マルチ・ブロックの非同期I/Oを使用してデータベース・ファイルに書き込みます。
- ダイレクト・パス・ロード中に、プロセスは、Oracle Databaseのバッファ・キャッシュを使用するのではなく、そのプロセス独自の書込みI/Oを実行します。この処理方法により、他のOracle Databaseユーザーとの競合を最少にします。
- SORTED INDEXESオプションを指定すると、使用システムまたはインストール環境に固有の高性能ソート・ルーチンを使用して、データを事前にソートしておくことができます。
- ロードするように指定した表が空の場合、事前ソート・オプションにより索引作成のソート段階とマージ段階が排除されます。索引は、データの挿入時に作成されます。
- インスタンス障害からのリストアに、REDOログ・ファイル・エントリは必要ありません。したがって、次の場合は、ロード時のログを記録する時間は不要です。
    - Oracle Databaseで、SQL NOARCHIVELOGパラメータが使用可能な場合
    - SQL*LoaderのUNRECOVERABLE句が使用可能な場合
    - ロードするオブジェクトのSQL NOLOGGINGパラメータが設定されている場合

関連トピック

- インスタンス・リカバリおよびダイレクト・パス・ロード

親トピック: ダイレクト・パス・ロード

12.4.6 ダイレクト・パス・ロード使用上の制限

ダイレクト・パス・ロード方式を使用するには、表およびセグメントが特定の要件を満たしている必要があります。一部の機能は、ダイレクト・パス・ロードでは使用できません。

ダイレクト・パス・ロードを使用するには、次の条件を満たしている必要があります。

- ロードする表はクラスタ化できません。
- ロードする表は、INSERTでアクティブなOracle Virtual Private Database (VPD)ポリシーを含むことができません。
- ロードするセグメントは、未処理のアクティブなトランザクションを含むことができません。

  アクティブなトランザクションを確認するには、MONITOR TABLEというOracle Enterprise Managerコマンドを使用して、ロードする表のオブジェクトIDを検索します。次に、MONITOR LOCKコマンドを使用して、表にロックがかけられているかどうか調べます。
- Oracle9iより前のOracle Databaseリリースでは、クライアントとサーバーが同じリリースである場合のみ、SQL*Loaderのダイレクト・パス・ロードを実行できます。この制限は、Oracle9iのデータを以前のOracle Databaseリリースにダイレクト・パス・ロードできないことも意味しています。たとえば、ダイレクト・パス・ロードを使用して、Oracle Database 9iリリース1 (9.0.1)からOracle 8i (8.1.7)のOracle Databaseにデータをロードすることはできません。

  Oracle Database 9i以降では、クライアントとサーバーのリリースが異なる場合にも、SQL*Loaderのダイレクト・パス・ロードを実行できます。ただし、両方がOracle Database 9iリリース1 (9.0.1)以降で、クライアントのリリースは、サーバーのリリースと同じかそれ未満である必要があります。たとえば、Oracle Database 9iリリース1 (9.0.1)データ

ベースからOracle Database 9iリリース2 (9.2)へのダイレクト・パス・ロードを実行できます。ただし、Oracle Database 10gからOracle Database 9iリリースへのデータのロードに、ダイレクト・パス・ロードを使用することはできません。

次の機能は、ダイレクト・パス・ロードでは使用できません。

- BFILE列のロード
- ロード時のCREATE SEQUENCEの使用。これは、ダイレクト・パスがINSERT文を生成しないため、ダイレクト・パス・ロードでは、次の値をフェッチするSQLが生成されないことが理由です。

親トピック: ダイレクト・パス・ロード

12.4.7 単一パーティションのダイレクト・パス・ロードでの制限

単一パーティションのダイレクト・パス・ロードを使用する場合は、ダイレクト・パス・ロードに指定するパーティションが追加の要件を満たす必要があります。

前述の制限に加え、単一のパーティションをロードするときには次の制限があります。

- パーティションのある表に、グローバル索引が定義されていないこと。
- パーティションのある表に対して、参照制約およびCHECK制約が使用禁止であること。
- トリガーが使用禁止であること。

親トピック: ダイレクト・パス・ロード

12.4.8 ダイレクト・パスを使用する場合

ダイレクト・パス・ロードでSQL*Loaderを実行する必要がある状況を学習します。

ビュー、フィールド・デフォルトまたは整合性制約によって制限されていない場合は、次の状況でダイレクト・パス・ロードを使用する必要があります。

- 短時間で大量のデータをロードする必要がある場合。ダイレクト・パス・ロードによって、大量のデータを高速ロードし、索引付けできます。表が空であるかどうかにかかわらず、データをロードできます。
- 最大のパフォーマンスを得るため、データをパラレルでロードする場合。

親トピック: ダイレクト・パス・ロード

12.4.9 整合性制約

ダイレクト・パス・ロード時には、すべての整合性制約が適用されます。ただし、すべての制約が同時に適用されるとはかぎりません。

NOT NULL制約は、SQL*Loaderのロード中に適用されます。これらの制約に従っていないレコードは拒否されます。

ロード中およびロード後に、UNIQUE制約が施行されます。UNIQUE制約に違反するレコードは拒否されます(制約違反が検出されると、そのレコードはメモリー内で使用不可になります)。

他の行または表に依存する整合性制約(参照制約など)は、ダイレクト・パス・ロード実行前に使用禁止になります。そのため、ロード後に再び使用可能にする必要があります。REENABLEを指定すると、SQL*Loaderにより、これらの制約はロード終了時に自動的に使用可能に戻されます。制約が再び使用可能になった時点で、表全体(すべての行)に対してチェックが行われます。このチェックでエラーが見つかった行は、指定されたエラー・ログに書き込まれます。

関連トピック

- ダイレクト・パス・ロード、整合性制約およびトリガー

親トピック: ダイレクト・パス・ロード

12.4.10 ダイレクト・パスのフィールド・デフォルト

ダイレクト・パス・ロードを使用する場合、データベースに定義されているデフォルトの列指定は使用できません。

デフォルト値を設定するフィールドに対しては、DEFAULTIF句を使用して指定する必要があります。DEFAULTIF句が指定されておらず、フィールドがNULLである場合は、NULL値がデータベースに挿入されます。

親トピック: ダイレクト・パス・ロード

12.4.11 シノニムへのロード

ダイレクト・パス・ロードでは、SQL*Loaderを使用して表のシノニムにデータをロードできます。ただし、そのシノニムが表または単純な表のビューを直接指している必要があります。

以下の制限に注意してください。

- ユーザー定義の型またはXMLデータのいずれかを持つ表にビューが存在する場合、ダイレクト・パス・モードは使用できません。
- ダイレクト・パス・モードでは、SQL式を含むSQL*Loader制御ファイルを使用してビューをロードすることはできません。

親トピック: ダイレクト・パス・ロード

12.5 ダイレクト・パス・ロードの使用

SQL*Loaderのダイレクト・パス・ロード方法を使用してデータをロードする方法を学習します。

- ダイレクト・パス・ロードの設定
  ダイレクト・パス・ロード用にデータベースを準備するために必要なビューを作成するには、セットアップ・スクリプト`catldr.sql`を実行する必要があります。
- ダイレクト・パス・ロードの指定
  SQL*Loaderをダイレクト・パス・ロード・モードで起動するには、コマンドラインまたはパラメータ・ファイルのDIRECTパラメータをTRUEに設定します。
- 索引の作成
  一時記憶域を使用すると、ダイレクト・パス・ロードのパフォーマンスが向上します。各ブロックがフォーマットされた後、新しい索引キーがソート(一時)セグメントに挿入されます。
- 使用禁止状態(Index Unusable)のままの索引
  ロードされているデータ・セグメントが、その索引の索引セグメントより新しいものになると、SQL*Loaderによって索引が索引使用禁止状態になります。
- データ・セーブによるデータ損失の防止
  データ・セーブを使用して、インスタンス障害によるデータ損失を回避できます。
- ダイレクト・パス・ロード時のデータ・リカバリ
  SQL*Loaderでは、ダイレクト・パス・ロード・メソッドを使用する場合にデータ・リカバリが完全にサポートされます。
- 長いデータ・フィールドのロード
  ラージ・オブジェクト・タイプ(LOB)を使用することにより、SQL*Loaderの最大バッファ・サイズがダイレクト・パスにロード

できるよりも長いデータをロードできます。

- PIECEDとしてのデータのロード
  データが論理レコードの最終列である場合、PIECEDパラメータを使用すると、データをセクションごとにロードできます。
- ダイレクト・パス・モードを使用するSQL*Loader操作の監査
  選択したユーザー・データベース・アクションを監視および記録するため、SQL*Loaderのダイレクト・パス・ロードの監査を実行できます。

親トピック: 従来型パス・ロードおよびダイレクト・パス・ロード

12.5.1 ダイレクト・パス・ロードのセットアップ

ダイレクト・パス・ロード用にデータベースを準備するために必要なビューを作成するには、セットアップ・スクリプトcatldr.sqlを実行する必要があります。

catldr.sqlを実行する必要があるのは、ダイレクト・ロードを実行する予定の各データベースに対して1回のみです。データベースのインストール時に、ダイレクト・ロードを実行することがわかっている場合は、データベースのインストール中にこのスクリプトを実行することもできます。

親トピック: ダイレクト・パス・ロードの使用

12.5.2 ダイレクト・パス・ロードの指定

SQL*Loaderをダイレクト・パス・ロード・モードで起動するには、コマンドラインまたはパラメータ・ファイルのDIRECTパラメータをTRUEに設定します。

たとえば、ダイレクト・パス・ロード・モードでSQL*Loaderを起動するようにパラメータ・ファイルを構成するには、パラメータ・ファイルに次の行を含めます。

```
DIRECT=TRUE
```

関連トピック

- ダイレクト・パス・ロードに必要な時間と領域の最小化
- 複数CPUシステムのダイレクト・パス・ロードの最適化

親トピック: ダイレクト・パス・ロードの使用

12.5.3 索引の作成

一時記憶域を使用すると、ダイレクト・パス・ロードのパフォーマンスが向上します。各ブロックがフォーマットされた後、新しい索引キーがソート(一時)セグメントに挿入されます。

ロードが終了すると、古い索引と新しいキーがマージされ、新しい索引が作成されます。古い索引、ソート(一時)セグメント、新しい索引セグメントでは、すべてのマージが完了するまで記憶域が必要です。最後に、古い索引と一時セグメントが削除されます。

従来型パス・ロードでは、行が挿入されるたびに索引が更新されます。この方法では一時記憶域は不要ですが、処理に時間がかかります。

- パフォーマンスの向上
  メモリーに制限があるシステムでSQL*Loaderダイレクト・ロードのパフォーマンスを向上するには、SINGLEROWパラメータを使用します。

- 一時セグメント記憶域要件の計算
  新規索引キーを格納するためにダイレクト・パス・ロード中に必要な一時セグメント領域の量を見積もるには、次の式を使用します。

親トピック: ダイレクト・パス・ロードの使用

12.5.3.1 パフォーマンスの向上

メモリーに制限があるシステムでSQL*Loaderダイレクト・ロードのパフォーマンスを向上するには、SINGLEROWパラメータを使用します。

> ノート:
>
> ダイレクト・ロード時にデータの事前ソートを指定してあり、既存の索引が空である場合、一時セグメントは不要でマージもされません。この場合は、索引にキーが直接付加されます。
>
> ダイレクト・パス・ロードのパフォーマンスの最適化を参照してください

複数の索引が作成されると、古い索引の他に、各索引に対応する一時セグメントが同時に存在するようになります。次に、新しいキーは一度に1索引ずつ古い索引とマージされます。新しい各索引が作成されると、古い索引とそれに対応する一時セグメントは削除されます。

関連トピック

- SINGLEROWオプション
- 索引サイズの見積りと記憶域パラメータの設定

親トピック: 索引の作成

12.5.3.2 一時セグメント記憶域要件の計算

新規索引キーを格納するためにダイレクト・パス・ロード中に必要な一時セグメント領域の量を見積もるには、次の式を使用します。

新しい索引キーの格納に必要な一時セグメント領域の大きさ(バイト単位)を計算するには、次の式を使用します。

```
1.3 * key_storage
```

この式では、キー・ストレージは次のように定義されます。number_rowsは行数、sum_of_column_sizesは列サイズの合計、number_of_columnsは索引内の列数です。

```
key_storage = (number_rows) *
( 10 + sum_of_column_sizes + number_of_columns )
```

この式における列(column)とは、索引の列を意味します。ここでは、1列につき1バイトを使用しています。さらに、ROWID,やその他のオーバーヘッドとして1行につき10バイトを計算に入れています。

定数1.3は、ソートに必要な追加領域の平均的な大きさを反映しています。この値は、データの順序がきわめてランダムな場合に有効です。データが逆の順序に並んでいると、ソートには2倍のキー記憶域が必要となるため、そのときは定数値を2.0とします。ただし、これは最悪の場合です。

データが完全にソートされている場合は、索引エントリを格納できる領域のみが必要なため、そのときの定数の値は1.0に下がります。

関連トピック

- 高速索引付けのためのデータの事前ソート

親トピック: 索引の作成

12.5.4 使用禁止状態(Index Unusable)のままの索引

ロードされているデータ・セグメントが、その索引の索引セグメントより新しいものになると、SQL*Loaderによって索引が索引使用禁止状態になります。

SQL文が索引使用禁止状態の索引を参照しようとすると、エラーが発生します。ダイレクト・パス・ロードの実行時に、次のような状況が発生すると、索引またはパーティション索引のパーティションは索引使用禁止状態になります。

- SQL*Loaderで索引のための領域が少なくなり、索引が更新できない場合。
- データがSORTED INDEXES句で指定した順序になっていない場合。
- インスタンス障害が発生したか、または索引作成中にOracleシャドウ・プロセスが失敗した場合。
- 一意の索引内に重複キーがある場合。
- データ・セーブポイントを使用中、データ・セーブポイント発生後にロードが正常に実行されないか、またはキーボードからの中断によって終了された場合。

ある索引が索引使用禁止状態かどうかを調べるには、次に示す簡単な問合せを実行します。

```
SELECT INDEX_NAME, STATUS
   FROM USER_INDEXES
   WHERE TABLE_NAME = 'tablename';
```

表の所有者でない場合は、USER_INDEXESのかわりに、ALL_INDEXESまたはDBA_INDEXESを検索してください。

ある索引パーティションが使用禁止状態かどうかを調べるには、次に示す問合せを実行します。

```
SELECT INDEX_NAME,
       PARTITION_NAME,
       STATUS FROM USER_IND_PARTITIONS
       WHERE STATUS != 'VALID';
```

表の所有者でない場合は、USER_IND_PARTITIONSのかわりに、ALL_IND_PARTITIONSおよびDBA_IND_PARTITIONSを検索します。

親トピック: ダイレクト・パス・ロードの使用

12.5.5 データ・セーブによるデータ損失の防止

データ・セーブを使用して、インスタンス障害によるデータ損失を回避できます。

- データ・セーブを使用したデータ損失の防止
  セーブポイントがある場合、SQL*Loaderのロード中にインスタンス障害が発生した場合は、SKIPパラメータを使用してロードを続行します。
- ROWSパラメータの使用
  ROWSパラメータには、ダイレクト・パス・ロードでデータ・セーブを行う間隔を設定します。
- データ・セーブとコミット
  従来型ロードでは、ROWSはコミット操作の前に読み込む行数を意味します。ダイレクト・ロードにおけるデータ・セーブは、従来型ロードにおけるコミットと同様ですが、異なる部分もあります。

親トピック: ダイレクト・パス・ロードの使用

12.5.5.1 データ・セーブを使用したデータ損失の防止

セーブポイントがある場合、SQL*Loaderのロード中にインスタンス障害が発生した場合は、SKIPパラメータを使用してロードを続行します。

最後のセーブポイントにロードされたすべてのデータが、インスタンス障害から保護されます。

インスタンス障害後にロードを続行する場合は、障害発生前に入力ファイルから何行処理されたか判別し、SKIPパラメータを使用して、処理済の行をスキップします。

表に索引がある場合には、ロードを続行する前にそれらの索引を削除し、ロード後に索引を再作成してください。メディア障害およびインスタンス・リカバリの詳細は、「ダイレクト・パス・ロード時のデータ・リカバリ」を参照してください。

ノート:

SQL*Loader ではデータのロードが完了するまで索引が作成されないため、索引はデータ・セーブでは保護されません(事前ソートされたデータを空の表にロードする場合にかぎり、ロード中に索引が作成されますが、その場合も索引は保護されません。)

関連トピック

- ダイレクト・パス・ロード時のデータ・リカバリ

親トピック: データ・セーブによるデータ損失の防止

12.5.5.2 ROWSパラメータの使用

ROWSパラメータには、ダイレクト・パス・ロードでデータ・セーブを行う間隔を設定します。

ROWSに指定する値は、データベースへの挿入を保存する前に、SQL*Loaderによって入力ファイルから読み込まれる行数です。

データ・セーブは負荷の高い操作です。データ・セーブの間隔が15分以上になるように、ROWSを十分高い値に設定してください。これによって、長時間のダイレクト・パス・ロードの実行中にインスタンス障害が発生したときに、損失する作業量の上限(最高水位標)を指定できます。ROWSに小さい値を設定すると、パフォーマンスおよびデータ・ブロック領域使用率が低下します。

親トピック: データ・セーブによるデータ損失の防止

12.5.5.3 データ・セーブとコミット

従来型ロードでは、ROWSはコミット操作の前に読み込む行数を意味します。ダイレクト・ロードにおけるデータ・セーブは、従来型ロードにおけるコミットと同様ですが、異なる部分もあります。

類似点は次のとおりです。

- データ・セーブを行うと他のユーザーもその行を参照できます。
- データ・セーブ後は、行をロールバックできません。

一方、従来型との主な相違点は、ダイレクト・パス・ロードのデータ・セーブではロードが完了するまで索引が使用できない(索引使用禁止状態)ということです。

親トピック: データ・セーブによるデータ損失の防止

12.5.6 ダイレクト・パス・ロード時のデータ・リカバリ

ダイレクト・パス・ロード方法を指定すると、SQL*Loaderのデータ・リカバリ機能が完全にサポートされます。

リカバリには大きく分けて2種類あります。

- メディア・リカバリは、損失したデータベース・ファイルをリカバリします。データベース・ファイルを損失した場合に、それをリカバリできるようにするには、ARCHIVELOGモードで実行する必要があります。
- インスタンス・リカバリは、障害が発生する前にインメモリー・データが変更されたが、ディスクに書き込まれる前に障害のため失われてしまったというシステム障害をリカバリします。Oracle Databaseでは、REDOログ・ファイルがアーカイブされていない場合も、インスタンス障害をリカバリできます。

- メディア・リカバリおよびダイレクト・パス・ロード
  REDOログ・ファイル・アーカイブ機能が使用可能になっている(ARCHIVELOGモードで実行している)場合、ダイレクト・パスでロードしたデータは、SQL*Loaderによってログに記録されます。それによって、メディア・リカバリが可能になります。
- インスタンス・リカバリおよびダイレクト・パス・ロード
  データベース・ファイルは、SQL*Loaderによって直接書き込まれます。そのため、インスタンスを再起動すると、最後にデータをセーブした時点までに挿入したすべての行が、自動的にデータベース・ファイルに存在します。

親トピック: ダイレクト・パス・ロードの使用

12.5.6.1 メディア・リカバリおよびダイレクト・パス・ロード

REDOログ・ファイル・アーカイブ機能が使用可能になっている(ARCHIVELOGモードで実行している)場合、ダイレクト・パスでロードしたデータは、SQL*Loaderによってログに記録されます。それによって、メディア・リカバリが可能になります。

REDOログ・ファイル・アーカイブの機能が使用可能になっていない(NOARCHIVELOGモードで実行している)場合、メディア・リカバリはできません。

ロード中に失われたデータベース・ファイルをリカバリするには、従来型パスでロードしたデータをリカバリするときと同じ方法を使用してください。

1. 影響を受けたデータベース・ファイルの最新のバックアップをリストアします。
2. RMAN RECOVERコマンドを使用して、表領域をリカバリします。

関連トピック

- Oracle Databaseバックアップおよびリカバリ・ユーザーズ・ガイドの表領域の完全リカバリの実行

親トピック: ダイレクト・パス・ロード時のデータ・リカバリ

12.5.6.2 インスタンス・リカバリおよびダイレクト・パス・ロード

データベース・ファイルは、SQL*Loaderによって直接書き込まれます。そのため、インスタンスを再起動すると、最後にデータをセーブした時点までに挿入したすべての行が、自動的にデータベース・ファイルに存在します。

変更がREDOログ・ファイルに記録されていなくても、インスタンス・リカバリは可能です。

インスタンス障害が発生すると、作成中の索引は索引使用禁止状態のままになります。使用禁止状態の索引は、表またはパーティションを使用する前に再構築する必要があります。索引が索引使用禁止状態のままであるかどうかを調べる方法については、「使用禁止状態(Index Unusable)のままの索引」を参照してください。

関連トピック

- 使用禁止状態(Index Unusable)のままの索引

親トピック: ダイレクト・パス・ロード時のデータ・リカバリ

12.5.7 LONG型データ・フィールドのロード

ラージ・オブジェクト・タイプ(LOB)を使用することにより、SQL*Loaderの最大バッファ・サイズがダイレクト・パスにロードできるよりも長いデータをロードできます。

長いデータ・フィールドのロード方法を検討する際には、次の点に注意してください。

- 長いデータ・フィールドをLOBとしてロードするパフォーマンスを向上させるために、大きなSTREAMSIZE値を使用することをお薦めします。
- LOBのかわりに、PIECEDパラメータを使用して、最大バッファ・サイズよりも長いデータを読み込むこともできます。ただし、このシナリオでは、PIECEDのかわりにLOBを使用することをお薦めします。

関連トピック

- SQL*Loaderを使用したLOBのロード
- 列配列の行数およびストリーム・バッファ・サイズの指定

親トピック: ダイレクト・パス・ロードの使用

12.5.8 PIECEDとしてのデータのロード

データが論理レコードの最終列である場合、PIECEDパラメータを使用すると、データをセクションごとにロードできます。

列をPIECEDと宣言することによって、LONGフィールドを複数の物理レコード(ピース)に分割することが、ダイレクト・パス・ローダーに通知されます。この場合、SQL*Loaderでは、 LONGフィールドの各ピースが物理レコード内で検索された順序で処理されます。レコードが処理される前に、フィールドのすべてのピースが読み込まれます。SQL*Loaderでは、格納前のLONGフィールドは具体化されません。ただし、レコードが処理される前に、フィールドのすべての部分が読み込まれます。

列をPIECEDと宣言する場合、次の制約が適用されます。

- このオプションはダイレクト・パスでのみ有効です。
- 1つの表につき1フィールドのみをPIECEDにできます。
- PIECEDフィールドは論理レコードの最終フィールドである必要があります。
- WHEN句、NULLIF句またはDEFAULTIF句では、PIECEDフィールドを使用できません。
- 論理レコード内のPIECEDフィールドの領域は、他のフィールドの領域と重複してはいけません。
- PIECEDに対応するデータベースの列を索引に含めることはできません。
- 拒否されたレコードにPIECEDフィールドが含まれている場合は、不良ファイルからそのレコードをロードできません。

  たとえば、1つのPIECEDフィールドが3つのレコードにまたがっているとします。SQL*Loaderでは、最初のレコードからPIECEDフィールドの第一分割がロードされ、次に同じバッファを使用して2番目のレコードから第二分割がロードされます。その後、同じバッファを使用して同様に3番目のレコードがロードされます。ここでエラーが検出されると、最初の2つのレコードはすでにバッファには存在しないため、3番目のレコードのみが不良ファイルに書き込まれます。その結果、不良ファイルにあるレコードが無効となります。

親トピック: ダイレクト・パス・ロードの使用

12.5.9 ダイレクト・パス・モードを使用するSQL*Loader操作の監査

選択したユーザー・データベース・アクションを監視および記録するため、SQL*Loaderのダイレクト・パス・ロードの監査を実行できます。

SQL*Loaderでは、すべての監査レコードが1つの場所に一元化される統合監査を使用します。

統合監査を設定するには、統合監査ポリシーを作成するか、既存のポリシーを変更します。監査ポリシーは、データベースにおけるユーザー動作の特定の部分を監査できる監査設定の名前付きグループです。ポリシーを作成するには、SQL `CREATE AUDIT POLICY`文を使用します。

監査ポリシーを作成したら、`AUDIT`および`NOAUDIT` SQL文を使用してポリシーをそれぞれ有効化および無効化します。

関連トピック

- CREATE AUDIT POLICY (統合監査)
- Oracle SQL*Loaderダイレクト・ロード・パス・イベントの監査

親トピック: ダイレクト・パス・ロードの使用

12.6 ダイレクト・パス・ロードのパフォーマンスの最適化

SQL*Loaderダイレクト・パス・ロードの高速実行を可能にし、使用する領域を減らす方法を学習します。

- ダイレクト・パス・ロードに必要な時間と領域の最小化
  ダイレクト・パス・ロードでは、使用する時間と一時記憶域を制御できます。
- 高速ロードのための記憶域の事前割当て
  SQL*Loaderでは、表のエクステントが自動的に追加されますが、このプロセスには時間がかかります。新しい表へ高速にロードするには、表の作成に必要なエクステントを事前に割り当ててください。
- 高速索引付けのためのデータの事前ソート
  索引付き列を基準にしてデータを事前ソートすると、SQL*Loaderダイレクト・パス・ロードのパフォーマンスを改善できます。
- データ・セーブの回数の削減
  `ROWS`値が小さいために、データ・セーブの回数が多くなると、ダイレクト・パス・ロードのパフォーマンスに悪影響があります。
- REDOログの最小限の使用
  ダイレクト・ロードを大幅に高速化する1つの方法は、REDOログの使用を最小限に抑えることです。
- 列配列の行数およびストリーム・バッファ・サイズの指定
  列配列の行数によって、ストリーム・バッファが作成される前にロードされる行数が決まります。T
- DATE_CACHEの値の指定
  ダイレクト・パス・ロード中に同じ日付またはタイムスタンプが何回も使用されるパフォーマンスを向上させるには、SQL*Loaderの日付キャッシュを使用します。

親トピック: 従来型パス・ロードおよびダイレクト・パス・ロード

12.6.1 ダイレクト・パス・ロードに必要な時間と領域の最小化

ダイレクト・パス・ロードでは、使用する時間と一時記憶域を制御できます。

時間を最小化するには:

- 記憶域の事前割当て
- データの事前ソート
- データ・セーブの回数の削減
- REDOログの最小限の使用
- 配列行の列数およびストリーム・バッファのサイズを指定
- 日付キャッシュの値の指定
- DB_UNRECOVERABLE_SCN_TRACKING=FALSEと設定します。リカバリ不能な(nologging)ダイレクト書込みは、SCNと最終ダイレクト書込みの時間を定期的に格納することによって、制御ファイル内で追跡されます。制御ファイルへのこれらの更新がパフォーマンスに悪影響を及ぼす場合は、DB_UNRECOVERABLE_SCN_TRACKINGパラメータをFALSEに設定すると、パフォーマンスが向上することがあります。

領域を最小化するには:

- ロード前のデータのソート時に、最も多くの一時記憶域を必要とする索引でデータをソートします。
- ロード中の索引メンテナンスを回避します。

親トピック: ダイレクト・パス・ロードのパフォーマンスの最適化

12.6.2 高速ロードのための記憶域の事前割当て

SQL*Loaderでは、必要に応じて自動的に表にエクステントが追加されますが、これには時間がかかります。新しい表へ高速にロードするには、表の作成に必要なエクステントを事前に割り当ててください。

表に必要な領域を計算するには、Oracle Database管理者ガイドのデータベース・ファイルの管理の説明を参照してください。必要な領域を割り当てるには、SQLのCREATE TABLE文でINITIALまたはMINEXTENTS句を使用します。

別の方法として、エクステントの割当て回数が減るようにエクステントのサイズを十分に大きくする方法もあります。

親トピック: ダイレクト・パス・ロードのパフォーマンスの最適化

12.6.3 高速索引付けのためのデータの事前ソート

索引付き列を基準にしてデータを事前ソートすると、SQL*Loaderのダイレクト・パス・ロードのパフォーマンスを改善できます。

- データの事前ソートの利点
  事前ソートによって、SQL*Loaderでのロード・パフォーマンスを向上させる方法について説明します
- SORTED INDEXES句
  SORTED INDEXESは、データを事前ソートしている索引を指定します。
- 未ソートのデータ
  SORTED INDEXES句で索引を指定しても、データがその索引でソートされていない場合は、ロード終了時に索引は索引使用禁止状態のままになります。
- 複数列索引
  SORTED INDEXES句で複数列の索引を指定する場合は、まず索引の最初の列で順序付けが行われ、次に2番目の列で順序付けが行われるように、データをソートしてください。
- 最適ソート順序の選択方法
  ダイレクト・パス・ロードのパフォーマンスを最大限に引き出すには、最も大きな一時セグメント領域を必要とする索引に基づいて、データを事前ソートしてください。

親トピック: ダイレクト・パス・ロードのパフォーマンスの最適化

12.6.3.1 データの事前ソートの利点

事前ソートによって、SQL*Loaderでのロード・パフォーマンスを向上させる方法について学習します

事前ソートを行うと、ロード時の一時記憶要件を最小限に抑えることができます。また、事前ソートでは、ご使用のオペレーティング・システムまたはアプリケーション用に最適化された高性能ソート・ルーチンを利用できます。

データが事前ソートされていて既存の索引が空でない場合は、事前ソートによって、新しいキーに必要な一時セグメント領域の大きさを最小にできます。ソート・ルーチンは、新しい各キーをキー・リストに追加します。ソート用の追加領域は必要なく、キーのための領域のみが必要となります。必要な記憶域の大きさを計算するには、ソート係数として1.3ではなく1.0を使用してください。必要な記憶域要件の見積りについては、「一時セグメント記憶域要件」を参照してください。

事前ソートを指定していて既存の索引が空である場合は、最大効率が実現します。新しいキーが索引に挿入されるのみです。一時セグメントと新しい索引が古い空の索引と同時に存在するのではなく、新しい索引のみが存在します。その結果、ロード中に一時的な記憶域が不要になり、時間が節約されます。

関連トピック

- 一時セグメント記憶域要件の計算

親トピック: 高速索引付けのためのデータの事前ソート

12.6.3.2 SORTED INDEXES句

`SORTED INDEXES`は、データを事前ソートしている索引を指定します。

この句は、ダイレクト・パス・ロードでのみ使用できます。例については、「事例6: ダイレクト・パス・ロード方式を使用したデータのロード」を参照してください。(事例の使用方法については、「SQL*Loaderの事例」を参照してください。)

一般に、`SORTED INDEXES`句では1つの索引のみを指定します。通常、これは、ある索引でソートされたデータは、別の索引にとって正しい順序とはかぎらないためです。ただし、複数の索引のデータの順序が同じである場合は、索引すべてを同時に指定できます。

`SORTED INDEXES`句で指定した索引はすべて、ダイレクト・パス・ロードを開始する前に作成する必要があります。

親トピック: 高速索引付けのためのデータの事前ソート

12.6.3.3 未ソートのデータ

`SORTED INDEXES`句で索引を指定しても、データがその索引でソートされていない場合は、ロード終了時に索引は索引使用禁止状態のままになります。

データは存在していますが、索引を使用しようとするとエラーになります。索引使用禁止状態の索引がある場合は、ロード後に再構築してください。

親トピック: 高速索引付けのためのデータの事前ソート

12.6.3.4 複数列索引

`SORTED INDEXES`句で複数列の索引を指定する場合は、まず索引の最初の列で順序付けが行われ、次に2番目の列で順序付けが行われるように、データをソートしてください。

たとえば、索引の最初の列に都市名があり、2番目の列に名前の名字がある場合、次のリストのように都市別順で、同じ都市の中では名字順に並ぶようにデータをソートします。

```
Albuquerque       Adams
Albuquerque       Hartstein
```

```
Albuquerque      Klein
...          ...
Boston           Andrews
Boston           Bobrowski
Boston           Heigham
...              ...
```

親トピック: 高速索引付けのためのデータの事前ソート

12.6.3.5 最適ソート順序の選択方法

ダイレクト・パス・ロードのパフォーマンスを最大限に引き出すには、最も大きな一時セグメント領域を必要とする索引に基づいて、データを事前ソートしてください。

たとえば、主キーが1つの数値列で、2次キーが3つのテキスト列で構成される場合、2次キーで事前ソートすることによって、ソート時間と記憶要件の両方を最小にできます。

最も大きな記憶域を必要とする索引がどれであるかを知るには、次の手順に従ってください。

1. 各索引について、その索引のすべての列の幅を加算します。
2. 単一表へのロードの場合は、最大幅を持つ索引を選択します。
3. 複数表へのロードの場合は、各表に対して最大幅を持つ索引を調べます。各表にロードされる行数が同じ場合は、最大幅を持つ索引を選択します。通常は、各表にロードされる行数は同じです。
4. 複数表へのロードにおいて、索引付きの表にロードされる行数が表によって異なる場合は、ステップ3で確認した各索引の幅と、その索引にロードされる行数を掛け合せます。結果が最も大きい値の索引を選択します。

親トピック: 高速索引付けのためのデータの事前ソート

12.6.4 データ・セーブの回数の削減

ROWS値が小さいことが原因でデータ・セーブが頻繁に発生する場合、ダイレクト・パス・ロードのパフォーマンスは低下します。

ROWS値が小さい場合、セーブ後に最後のデータ・ブロックには書込みが行われないため、データ・ブロック領域が無駄になります。

ダイレクト・パス・ロードは従来型ロードより何倍も高速なため、ダイレクト・ロードの場合には、ROWSの値は従来型ロードの場合よりかなり大きくする必要があります。

データ・セーブ時には、SQL*Loaderのすべてのバッファへの書込みが正常に終了するまで、ロードは停止します。ROWSの値は、安全性を確保できる範囲で、できるだけ大きくしてください。数千行をロードしてみて、1行当たりの平均ロード時間を計ってみることをお薦めします。その値から、ROWSに設定する値が求められます。

たとえば、1分当たり20,000行がロードされるとします。この場合、処理途中に実行するセーブの間隔を10分以内にするには、ROWSを200,000(20,000行/分×10分間)に設定してください。

親トピック: ダイレクト・パス・ロードのパフォーマンスの最適化

12.6.5 REDOログの最小限の使用

ダイレクト・ロードを大幅に高速化する1つの方法は、REDOログの使用を最小限に抑えることです。

それには3通りの方法があります。アーカイブを使用禁止にする方法、ロードをリカバリ不可能に指定する方法、ロードされるオブジェクトに対してSQLのNOLOGGINGパラメータを設定する方法です。この項では、すべての方法を説明します。

- アーカイブの使用禁止
  アーカイブが使用禁止の場合、ダイレクト・パス・ロードでは全体イメージのREDOログは生成されません。

- SQL*LoaderのUNRECOVERABLE句の指定
  時間およびREDOログ・ファイルの領域を節約するには、データのロード時に制御ファイルでSQL*LoaderのUNRECOVERABLE句を使用します。
- SQL NOLOGGINGパラメータの設定
  データまたは索引のセグメントにSQLのNOLOGGINGパラメータが設定されていると、そのセグメントに対する全体イメージのREDOログは使用できません(無効REDOログが生成されます)。

親トピック: ダイレクト・パス・ロードのパフォーマンスの最適化

12.6.5.1 アーカイブの使用禁止

アーカイブが使用禁止の場合、ダイレクト・パス・ロードでは全体イメージのREDOログは生成されません。

SQLのARCHIVELOGおよびNOARCHIVELOGパラメータを使用して、アーカイブ・モードを設定します。

関連トピック

- Oracle Database管理者ガイドのアーカイブREDOログ・ファイルの管理

親トピック: REDOログの最小限の使用

12.6.5.2 SQL*LoaderのUNRECOVERABLE句の指定

時間およびREDOログ・ファイルの領域を節約するには、データのロード時に制御ファイルでSQL*LoaderのUNRECOVERABLE句を使用します。

リカバリ不可能を指定したロードの場合は、ロードされたデータはREDOログ・ファイルに記録されません。かわりに、操作を無効にするために必要なREDOログ(無効REDOログ)が生成されます。

UNRECOVERABLE句は、ロード・セッション中にロードされたすべてのオブジェクト(データ・セグメントおよび索引セグメントの両方)に適用されます。このため、ロードされた表についてはメディア・リカバリはできません。ただし、他のユーザーが行ったデータベース変更のログは、引き続き記録されます。

ノート:

データ・ロードは記録されないため、必要な場合はロード後にデータのバックアップを取ってください。

UNRECOVERABLE句を指定してロードしたデータについてメディア・リカバリが必要になった場合、ロードしたデータ・ブロックには、論理的に破損したというマークが付けられます。

データをリカバリするには、データを削除して再作成します。データがリカバリ不能にならないように、データのロード後、すぐにバックアップを取ってください。

デフォルトでは、ダイレクト・パス・ロードはRECOVERABLEです。

次に、制御ファイルのUNRECOVERABLE句の指定例を示します。

```
UNRECOVERABLE
LOAD DATA
INFILE 'sample.dat'
INTO TABLE emp
(ename VARCHAR2(10), empno NUMBER(4));
```

親トピック: REDOログの最小限の使用

12.6.5.3 SQL NOLOGGINGパラメータの設定

データまたは索引のセグメントにSQLのNOLOGGINGパラメータが設定されていると、そのセグメントに対する全体イメージのREDOログは使用できません(無効REDOログが生成されます)。

NOLOGGINGパラメータを使用すると、ログが記録されないオブジェクトに対しより優れた制御が可能です。

親トピック: REDOログの最小限の使用

12.6.6 列配列の行数およびストリーム・バッファ・サイズの指定

列配列の行数によって、ストリーム・バッファが作成される前にロードされた行数を判断します。T

STREAMSIZEパラメータで、クライアントからサーバーへ送ったデータのストリーム・サイズ(バイト単位)を指定します。

列配列の行数の値を指定するには、COLUMNARRAYROWSパラメータを使用します。ダイレクト・パスを使用してVARRAYをロードすると、COLUMNARRAYROWSパラメータはデフォルトで100に設定され、クライアント・オブジェクトのキャッシュ・スラッシングを回避します。

ダイレクト・パス・ストリーム・バッファのサイズを指定するには、STREAMSIZEパラメータを使用します。

これらのパラメータの最適値は、使用されるシステム、入力データ型およびOracleの列データ型に応じて異なります。独自の構成用に最適な値を使用することで、SQL*Loaderのログ・ファイルでの経過時間が少なくなります。

> ノート:
>
> ページングが過剰に発生すると、パフォーマンスが大幅に低下するため、ページング・アクティビティのプロセスを監視する必要があります。過剰なページングを回避するには、READSIZE、STREAMSIZE および COLUMNARRAYROWS の値を小さくする必要があります。

複数CPUシステムでダイレクト・パス・ロードを実行する場合、列配列の行数およびストリーム・バッファのサイズを指定すると、特に有効です。

関連項目:

- 複数CPUシステムのダイレクト・パス・ロードの最適化
- COLUMNARRAYROWS
- STREAMSIZE

親トピック: ダイレクト・パス・ロードのパフォーマンスの最適化

12.6.7 DATE_CACHEの値の指定

ダイレクト・パス・ロード中に同じ日付またはタイムスタンプが何回も使用されるパフォーマンスを向上させるには、SQL*Loaderの日付キャッシュを使用します。

同じ日付値またはタイムスタンプ値のロードが何度も行われるダイレクト・パス・ロードを実行する場合、総ロード時間の大部分が日付およびタイムスタンプのデータの変換に使用される可能性があります。特に、複数の日付列がロードされる場合にこのような状況が発生します。この場合、SQL*Loaderの日付キャッシュを使用することによってパフォーマンスを向上できます。

日付キャッシュを使用すると、入力データ内に多数の重複する日付値が存在する場合、日付変換が実行される回数が減ります。この機能を使用すると、ロード中に予測される一意の日付の数を指定できます。

日付キャッシュは、デフォルトで使用可能です。日付キャッシュ機能を使用禁止にするには、0(ゼロ)に設定します。

デフォルトの日付キャッシュ・サイズは1000要素です。デフォルトのサイズを使用し、1000を超える一意の入力値がロードされると、日付キャッシュはこの表に対して自動的に使用禁止となります。これによって、過剰および不要なルックアップ時間によって、パフォーマンスが低下する可能性がなくなります。ただし、デフォルトを使用するかわりに0(ゼロ)以外の値を日付キャッシュに指定し、キャッシュ量がこの値を超えた場合、日付キャッシュは使用禁止になりません。最大値を超えた入力データは、適切な変換ルーチンによって明示的に変換されます。

日付キャッシュは、1つの表のみに対応付けできます。複数の表で日付キャッシュの共有はできません。次のすべての条件を満たす場合にのみ、表に対して日付キャッシュが作成されます。

- DATE_CACHEパラメータが0(ゼロ)以外に設定されている
- 表への格納のためにデータ型変換が必要な、1つ以上の日付値またはタイムスタンプ値(あるいはその両方)がロードされている
- ダイレクト・パス・ロードでロードされている

日付キャッシュの統計はログ・ファイルに書き込まれます。これらの統計を使用して、次のとおりダイレクト・パス・ロードのパフォーマンスを向上できます。

- キャッシュ・エントリの数がキャッシュ・サイズより小さく、キャッシュ・ミスがない場合は、キャッシュ・サイズをより小さい値に設定できます。
- キャッシュ・ヒット(重複値が存在するエントリ)の数が小さく、キャッシュ・ミスの数が大きい場合は、キャッシュ・サイズを大きくする必要があります。キャッシュ・サイズを大きくしすぎると、過剰なページング、過度のメモリー使用量などの他の問題が発生する場合があります。
- ほぼすべての入力日付値が一意の場合、日付キャッシュを使用してもパフォーマンスは向上しないため、使用する必要はありません。

ノート:

日付キャッシュがデフォルトで使用可能な場合、最大値を超えたため使用禁止になると、日付キャッシュの統計は SQL*Loader のログ・ファイルに書き込まれません。

キャッシュ・サイズを大きくしてもパフォーマンスが向上しない場合は、デフォルトの動作に戻すか、またはキャッシュ・サイズを0(ゼロ)に設定します。パフォーマンス全体の向上は、ロードされる他の列のデータ型によっても異なります。ロードされる日付列の総数がロードされる他のデータ型より大きい場合、パフォーマンスは大幅に向上します。

関連トピック

- DATE_CACHE

親トピック: ダイレクト・パス・ロードのパフォーマンスの最適化

12.7 複数CPUシステムのダイレクト・パス・ロードの最適化

複数CPUシステムでダイレクト・パス・ロードを実行する場合、SQL*Loaderではデフォルトでマルチスレッドが使用されます。こ

の場合の複数CPUシステムは、2つ以上のCPUを持つ単一のシステムとして定義されます。

マルチスレッド・ロードとは、可能な場合、列配列をストリーム・バッファに変換し、ストリーム・バッファ・ロードがパラレルで実行されることを表します。この最適化は、次の場合に最も効果的です。

- 列配列が、ロード用に複数のダイレクト・パス・ストリーム・バッファを生成できる十分な大きさの場合。
- 入力フィールドのデータ型からOracleの列データ型へのデータ変換が必要な場合。

  変換は、ストリーム・バッファのロード時にパラレルで実行されます。

この処理の状態は、次の例に示すとおり、SQL*Loaderのログ・ファイルに記録されます。

```
Total stream buffers loaded by SQL*Loader main thread:         47
Total stream buffers loaded by SQL*Loader load thread:        180
Column array rows:                                           1000
Stream buffer bytes:                                       256000
```

この例では、SQL*Loaderのロード・スレッドがメイン・スレッドをオフロードしています。これによって、ロード・スレッドがサーバーで現行のストリームをロードする一方で、メイン・スレッドは、次のストリーム・バッファを作成できます。

ロード・スレッドを使用して、できるだけ多くのストリーム・バッファ・ロードを実行することが目的です。これによって、列配列の行数の増加、またはストリーム・バッファのサイズの削減(あるいはその両方)を実現できます。SQL*Loaderのログ・ファイルの経過時間を監視することで、変更による効果を確認できます。詳細は、「列配列の行数およびストリーム・バッファ・サイズの指定」を参照してください。

単一CPUシステム上では、最適化はデフォルトで無効になります。サーバーが他のシステム上にある場合は、マルチスレッドを手動で有効にするとパフォーマンスが向上します。

マルチスレッド・オプションを有効または無効にするには、SQL*LoaderのコマンドラインでMULTITHREADINGパラメータを使用するか、またはSQL*Loaderの制御ファイルにMULTITHREADINGパラメータを指定します。

関連項目:

ダイレクト・パス・ロードの概要は、『Oracle Call Interfaceプログラマーズ・ガイド』を参照してください

親トピック: 従来型パス・ロードおよびダイレクト・パス・ロード

12.8 索引メンテナンスの回避

従来型パスとダイレクト・パスの両方について、SQL*Loaderでは表のすべての既存の索引がメンテナンスされます。

索引のメンテナンスを回避するには、次のいずれかの方法を使用します。

- ロードを始める前に索引を削除します。
- ロードを始める前に、選択した索引または索引パーティションを索引使用禁止状態に設定し、SKIP_UNUSABLE_INDEXESパラメータを使用します。
- SKIP_INDEX_MAINTENANCEパラメータを使用します(ダイレクト・パスの場合に限定されるため、注意して使用してください)。

索引のメンテナンスを回避すると、ダイレクト・パス・ロード中に必要な領域を最小限にできます。その方法は次のとおりです。

- 一度に索引を作成できるため、各索引を別々に作成する場合に必要なソート用の(一時)セグメント領域を削減できます。

- 索引の作成時に、索引セグメントは1つのみ存在します(これに対し、新しいキーを古いキーにマージして新しい索引を作成するときには、一時的に3つのセグメントが存在します)。

表の全行数に対してロードする行数が多い場合、索引のメンテナンスを避けることは合理的です。ただし、比較的少数の行を大きな表に追加する場合は、索引の再ソートに非常に時間がかかることがあります。そのような場合は、従来型パス・ロードを使用するか、SQL*LoaderのSINGLEROWパラメータを使用します。詳細は、「SINGLEROWオプション」を参照してください。

親トピック: 従来型パス・ロードおよびダイレクト・パス・ロード

12.9 ダイレクト・パス・ロード、整合性制約およびトリガー

従来型パス・ロードと比較して、ダイレクト・パス・ロードを使用したトリガーの設定方法が異なる場合があります

従来型パス・ロードでは、行配列の挿入には標準SQL INSERT文を使用します。このとき、整合性制約および挿入トリガーは自動的に適用されます。ただし、ダイレクト・パスでデータをロードする場合は、SQL*Loaderでは一部の整合性制約およびすべてのデータベース・トリガーが使用禁止になります。

- 整合性制約
  SQL*Loaderを使用したダイレクト・パス・ロード中に、一部の整合性制約は自動的に使用禁止になりますが、その他は使用禁止になりません。
- データベース挿入トリガー
  ダイレクト・パス・ロードが始まると、表挿入トリガーも無効化されます。
- 永続的に使用禁止のトリガーおよび制約
  SQL*Loaderでは、トリガーおよび制約を使用禁止にするために、ロードされる表にいくつかのロックを獲得する必要があります。
- 従来型パスの同時ロードによるパフォーマンスの向上
  トリガーまたは整合性制約の問題があっても、より高速なロードを実現する場合は、従来型パスによる同時ロードの使用を考えてください。

親トピック: 従来型パス・ロードおよびダイレクト・パス・ロード

12.9.1 整合性制約

SQL*Loaderを使用したダイレクト・パス・ロード中に、一部の整合性制約は自動的に使用禁止になりますが、その他は使用禁止になりません。

整合性制約によってデータベースに関連付けられたビジネス・ルールがどのように強制されるかという概念をよく理解し、様々な手法を使用して表に無効な情報が入力されないようにするには、「データ整合性」を参照してください。

- 使用可能な制約
  ダイレクト・パス・ロード中に、一部の制約は使用可能なままになります。
- 使用禁止の制約
  ダイレクト・パス・ロード中に、一部の制約は使用不可になります。
- 制約を使用可能に戻す方法
  SQL*Loaderのロードが完了すると、REENABLE句が指定されている場合、整合性制約が自動的に使用可能に戻されます。

関連トピック

- データ整合性

親トピック: ダイレクト・パス・ロード、整合性制約およびトリガー

12.9.1.1 使用可能な制約

ダイレクト・パス・ロード中に、一部の制約は使用可能なままになります。

ダイレクト・パス・ロード時でも使用可能な制約は、次のとおりです。

- NOT NULL
- UNIQUE
- 主キー制約(NOT NULL列における一意制約)

NOT NULL制約は列配列の作成時にチェックされます。NOT NULL制約に違反する行はすべて拒否されます。

ダイレクト・パス・ロード時にUNIQUE制約が使用可能であっても、その制約に違反する行もロードされます。(これは、そのような行が拒否される従来型パスとは異なります。)UNIQUE制約は、ダイレクト・パス・ロードの最後で索引が再作成されるときに検証されます。違反が検出されると、索引は索引使用禁止状態のままになります。詳細は、「使用禁止状態(Index Unusable)のままの索引」を参照してください。

親トピック: 整合性制約

12.9.1.2 使用禁止の制約

ダイレクト・パス・ロード中に、一部の制約は使用不可になります。

次の制約は、ダイレクト・パス・ロード時にデフォルトで自動的に使用禁止になります。

- CHECK制約
- 参照制約(外部キー)

EVALUATE CHECK_CONSTRAINTS句を指定すると、CHECK制約の自動的な使用禁止を変更できます。SQL*Loaderでは、ダイレクト・パス・ロード中にCHECK制約が評価されます。CHECK制約に違反する行はすべて拒否されます。次の例に、SQL*Loader制御ファイルでのEVALUATE CHECK_CONSTRAINTS句の使用方法を示します。

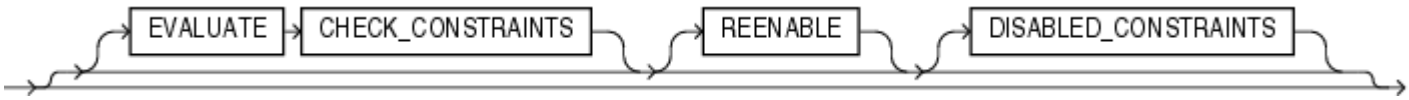
```
LOAD DATA
INFILE *
APPEND
INTO TABLE emp
EVALUATE CHECK_CONSTRAINTS
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
(c1 CHAR(10) ,c2)
BEGINDATA
Jones,10
Smith,20
Brown,30
Taylor,40
```

親トピック: 整合性制約

12.9.1.3 制約を使用可能に戻す方法

SQL*Loaderのロードが完了すると、REENABLE句が指定されている場合、整合性制約が自動的に使用可能に戻されます。

REENABLE句の構文は次のとおりです。



DISABLED_CONSTRAINTSパラメータはオプションで、読みやすくするために使用します。EXCEPTIONS句が含まれている場合は、例外表(デフォルト名: EXCEPTIONS)がすでに存在していて、それに挿入できる必要があります。この表には、整合性制約のいずれかに違反したすべての行のROWID値が格納されます。また、違反があった制約名も格納されます。

例外表を作成する手順は、*Oracle Database SQL*言語リファレンスのconstraintにあるexceptions_clauseを参照してください。

SQL*Loaderログ・ファイルには、使用禁止となっていた制約、および使用可能に戻された制約が記録され、エラーが原因で制約を使用可能に戻せなかった場合または検査できなかった場合はそのエラーが記録されます。さらに、ロードした表のそれぞれに指定された例外表の名前もログ・ファイルに書き込まれます。

REENABLE句を使用しない場合は、表のすべての行が検査されるときに制約を手動で使用可能に戻す必要があります。ここで新しいデータにエラーが見つかると、エラー・メッセージが生成されます。例外表を指定した場合は、違反のあった制約名と不良データのROWIDが、その表に書き込まれます。

REENABLE句を使用すると、制約を自動的に使用可能に戻し、新しい行すべてを検証できます。新しいデータにエラーが見つからなかった場合は、検証された制約に自動的にマークを付けられます。新しいデータにエラーが見つかった場合は、ログ・ファイルにエラー・メッセージが書き込まれ、SQL*Loaderによって制約の状態にENABLE NOVALIDATEというマークが付けられます。例外表を指定した場合は、違反のあった制約名と不良データのROWIDが、その表に書き込まれます。

ノート:

通常、表制約が ENABLE NOVALIDATE の状態のままになっている場合、新しいデータは表へ挿入されますが、新しい無効なデータは挿入されません。ただし、SQL*Loader のダイレクト・パス・ロードでは、この規則は施行されません。したがって、次のダイレクト・パス・ロードが無効なデータで実行される場合、無効なデータは挿入されますが、前述されたのと同じエラー・レポートおよび例外表の処理が実行されます。この例では、各ロードの前に外部表がクリーン・アウトされていない場合、その表には、重複するエントリが含まれます。重複するエントリは、次のような問合せを実行することによって、簡単に削除できます。

```
SELECT UNIQUE * FROM exceptions_table;
```

ノート:

参照整合性の再検証は、表全体に対して実行する必要があります。そのため、少数の行を非常に大きい表にロードするときは、ダイレクト・パスではなく従来型パスを使用した方がパフォーマンスが向上することがあります。

関連トピック

- Oracle Database SQL言語リファレンスのconstraint

親トピック: 整合性制約

12.9.2 データベース挿入トリガー

ダイレクト・パス・ロードが始まると、表挿入トリガーも使用禁止になります。

行のロードおよび索引の再作成が完了すると、使用禁止になっていたトリガーはすべて使用可能に戻されます。ログ・ファイルには、ロード時に使用禁止になっていたすべてのトリガーのリストが示されます。トリガーを使用可能に戻すときに1つでもエラーがあると、使用可能にできません。

整合性制約と異なり、挿入トリガーは、使用可能に戻っても表全体に対して再び適用されません。つまり、ダイレクト・パスでロードされた行に対しては、挿入トリガーは起動しません。ダイレクト・パスでロードした場合は、新しい行への挿入トリガーに相当する処理はすべて、アプリケーション側で実行する必要があります。

- 挿入トリガーの整合性制約への置換
  アプリケーションが整合性制約を実行する場合、通常は挿入トリガーが使用されます。
- 自動制約が使用できない場合
  挿入トリガーは、Oracleの自動整合性制約に置き換えられない場合があります。
- データベース・トリガーの準備
  挿入トリガーまたは自動制約方法を使用する前に、Oracle Database表を準備する必要があります。
- 更新トリガーの使用
  一般に、挿入トリガーと同じ処理を実現する場合は、データベース更新トリガーを使用します。
- 例外処理と同じ処理の実現
  挿入トリガーの中で例外を呼び出している場合、それと同じ処理を行うには、さらに作業が必要です。
- ストアド・プロシージャの使用
  挿入トリガーを使用すると例外が発生する場合は、ストアド・プロシージャを使用して挿入トリガーと同じ処理を実現することを検討してください。

親トピック: ダイレクト・パス・ロード、整合性制約およびトリガー

12.9.2.1 挿入トリガーの整合性制約への置換

アプリケーションは整合性制約を実行する場合、通常は挿入トリガーを使用します。

アプリケーションが使用する挿入トリガーのほとんどは単純なため、Oracleの自動整合性制約に置き換えることができます。

親トピック: データベース挿入トリガー

12.9.2.2 自動制約が使用できない場合

挿入トリガーは、Oracleの自動整合性制約に置き換えられない場合があります。

たとえば、挿入トリガーの中で表のルックアップ関数を使用して整合性チェックを行っている場合、自動制約は使用できません。これは、自動制約はカレント行における定数および列以外は参照できないためです。このようなトリガーと同じ処理を実現する方法が2つあります。この項ではその方法について説明します。

親トピック: データベース挿入トリガー

12.9.2.3 データベース・トリガーの準備

挿入トリガーまたは自動制約方法を使用する前に、Oracle Database表を準備する必要があります。

次に示す一般的なガイドラインに従って、表を準備してください。

1. ロードの前に、表に1バイトまたは1文字分の列を追加します。この列は、各行が「旧データ」か「新データ」かを示すためのものです。
2. この列の値がNULLの場合は「旧データ」を示すことにします。これは、NULL列であれば領域を使用せずに済むためです。

3. ロード時にSQL*LoaderのCONSTANTパラメータを使用して、ロードしたすべての行に「新データ」を示すフラグを付けます。

この手順に従って準備すると、新しくロードした行が識別できるため、古い行に影響を与えずに新しいデータを操作できます。

親トピック: データベース挿入トリガー

12.9.2.4 更新トリガーの使用

一般に、挿入トリガーと同じ処理を実現する場合は、データベース更新トリガーを使用します。

これは最も単純な方法です。例外を呼び出さない挿入トリガーの場合は、常にこの方法を使用できます。

1. 挿入トリガーと同じ処理を行う更新トリガーを作成します。

   トリガーをコピーします。「`new.column_name`」というすべての箇所を「`old.column_name`」に変更してください。

2. 現行の更新トリガーがある場合は、それを新しい更新トリガーに置き換えます。
3. 「新データ」のフラグをNULLに変更して、表を更新します。これによって、更新トリガーが起動します。
4. 元の更新トリガーがある場合は、それをリストアします。

トリガーの動作によっては、この操作中に表に対する排他的更新アクセスが必要となる場合もあります。排他的更新アクセスを実行すると、他のユーザーが修正する行に、誤ってこのトリガーが適用されることはありません。

親トピック: データベース挿入トリガー

12.9.2.5 例外処理と同じ処理の実現

挿入トリガーの中で例外を呼び出している場合、それと同じ処理を行うには、さらに作業が必要です。

例外を呼び出すということは、表にその行を挿入しないということです。この処理を更新トリガーで実現するには、ロードした行に削除フラグを付けておく必要があります。

この場合、「新データ」列を削除フラグに使用することはできません。更新トリガーでは、その起動元である列を修正できないためです。したがって、表にもう1列追加する必要があります。ここで追加する列は、削除する行を示すためのものです。NULL値の場合は、行が有効であることを示します。挿入トリガーで例外を呼び出すと、常に、更新トリガーによって追加列にフラグが設定されます。これによって、その行が無効であることが示されます。

要約すると、挿入トリガーで例外を呼び出している場合は、次の条件を満たすと、同じ処理を更新トリガーで実現できます。

- 表に列を2つ追加する(通常はNULL)。
- 表を排他的に更新できる(必要な場合)。

親トピック: データベース挿入トリガー

12.9.2.6 ストアド・プロシージャの使用

挿入トリガーを使用すると例外が発生する場合は、ストアド・プロシージャを使用して挿入トリガーと同じ処理を実現することを検討してください。

次に示すプロシージャは、どのような場合でも使用できますが、その実装はより複雑になります。このプロシージャは、挿入トリガーで例外を呼び出すときに使用できます。そのとき、2番目の列を追加する必要はありません。また、更新トリガーとは処理が異なるため、表に対する排他的アクセス権限がなくても使用できます。

1. 挿入トリガーと同じ処理を実現するストアド・プロシージャを作成します。
   a. 表から新しい行をすべて選択するように、カーソルを宣言します。

b. 処理ループの中でカーソルをオープンし、1回に1行ずつフェッチします。
c. 挿入トリガーでの操作を実行します。
d. 操作が正常に終了した場合は、「新データ」のフラグをNULLに変更します。
e. 操作が失敗した場合は、「新データ」のフラグを「不良データ」に変更します。

2. SQL*Plusなどの管理ツールを使用して、このストアド・プロシージャを実行します。
3. プロシージャの実行後、表の中に「不良データ」フラグの付いた行がないかどうかを調べます。
4. 不良の行を更新または削除します。
5. 挿入トリガーを使用可能に戻します。

親トピック: データベース挿入トリガー

12.9.3 永続的に使用禁止のトリガーおよび制約

SQL*Loaderでは、トリガーおよび制約を使用禁止にするために、ロードされる表にいくつかのロックを獲得する必要があります。

競合するプロセスが表のトリガーまたは制約を使用可能にしているときに、SQL*Loaderでその表のトリガーまたは制約を使用禁止にしようとした場合、SQL*Loaderではその表に関して排他的アクセス権を獲得することはできません。

この場合、SQL*Loaderでは、できるかぎり問題のないように処理が実行されます。ロード終了前に、使用禁止のトリガーおよび制約を使用可能に戻そうとします。ただし、表ロックが原因でSQL*Loaderの処理を継続できなくなった場合は、SQL*Loaderでトリガーや制約を使用可能にする処理も実行されないことがあります。この場合、トリガーおよび制約は、手動で使用可能にするまで永続的に使用できない状態になります。

このような状況はまれですが、発生する可能性はあります。このような状況を回避するには、ダイレクト・ロードの処理中は、表のトリガーまたは制約を使用可能にするアプリケーションを実行しないことをお薦めします。

適切なロックの獲得に失敗したことによってダイレクト・ロードが終了した場合は、ログを確認してください。ログには、使用禁止のすべてのトリガーおよび制約と、それらを使用可能に戻すための各試行が示されます。SQL*Loaderによって使用可能に戻されなかったすべてのトリガーおよび制約は、『Oracle Database SQL言語リファレンス』で説明するALTER TABLE文のENABLE句を使用して、手動で使用可能にする必要があります。

親トピック: ダイレクト・パス・ロード、整合性制約およびトリガー

12.9.4 従来型パスの同時ロードによるパフォーマンスの向上

トリガーまたは整合性制約の問題があっても、より高速なロードを実現する場合は、従来型パスによる同時ロードの使用を考えてください。

つまり、複数CPUシステムで同時に複数のセッションでロードを実行します。入力データ・ファイルを論理レコードの境界で別々のファイルに分割し、それらの各入力データ・ファイルを従来型パス・ロード・セッションでロードします。このロードには、次のような特長があります。

- 複数CPUシステムでの単一従来型パス・ロードよりは速くなりますが、ダイレクト・ロードほど速くはありません。
- トリガーが起動されて整合性制約がロードされた行に適用され、標準DML実行ロジックによって索引がメンテナンスされます。

親トピック: ダイレクト・パス・ロード、整合性制約およびトリガー

12.10 ダイレクト・パス・ロードのパフォーマンスの最適化

SQL*Loaderダイレクト・パス・ロードの高速実行を可能にし、使用する領域を減らす方法を学習します。

- SQL*Loaderのパラレル・データ・ロード・モデルについて
  データのロードに必要な経過時間を最小限に抑えるために使用できる、同時実行性の基本モデルが3つあります。
- 従来型パスの同時ロード
  このトピックでは、従来型パスによる同時ロードの使用について説明します。
- ダイレクト・パスによるセグメント間同時処理
  セグメント間同時処理は、異なるオブジェクトを同時にロードする場合に使用できます。
- ダイレクト・パスによるセグメント内同時処理
  SQL*Loaderでは、複数のセッションを同時に実行して、同一の表またはパーティション表の同一パーティションに対してダイレクト・パス・ロードを実行できます。
- パラレル・ダイレクト・パス・ロードの制限
  パラレル・ダイレクト・パス・ロードには制限があります。
- 複数のSQL*Loaderセッションの開始
  入力元となるデータ・ファイルは、SQL*Loaderセッションごとに異なります。同じ表にダイレクト・ロードを実行するセッションすべてに対して、PARALLEL句にTRUEを設定する必要があります。
- パラレル・ダイレクト・パス・ロードのパラメータ
  パラレル・ダイレクト・パス・ロードを実行する場合、ローダーによって割り当てられる一時セグメントの属性を指定してできるオプションがあります。
- パラレル・ダイレクト・パス・ロード後の制約の使用可能化
  すべてのデータのロード完了後に、制約およびトリガーを手動で使用可能にしてください。
- 主キー制約および一意キー制約
  このトピックでは、主キー制約および一意キー制約の使用について説明します。

親トピック: 従来型パス・ロードおよびダイレクト・パス・ロード

12.10.1 SQL*Loaderのパラレル・データ・ロード・モデルについて

データのロードに必要な所要時間を最小限にするために使用される、同時処理の3つの基本モデルがあります。

同時実行性モデルは次のとおりです。

- 従来型パスによる同時ロード
- ダイレクト・パス・ロードによるセグメント間同時処理
- ダイレクト・パス・ロードによるセグメント内同時処理

親トピック: ダイレクト・パス・ロードのパフォーマンスの最適化

12.10.2 従来型パスによる同時ロード

このトピックでは、従来型パスによる同時ロードの使用について説明します。

同時に複数の従来型パス・ロード・セッションを実行する方法の詳細は、「従来型パスの同時ロードによるパフォーマンスの向上」を参照してください。同一または異なるオブジェクトを制限なしで同時にロードする場合に、この方法を使用できます。

親トピック: ダイレクト・パス・ロードのパフォーマンスの最適化

12.10.3 ダイレクト・パスによるセグメント間同時処理

セグメント間同時処理は、異なるオブジェクトを同時にロードする場合に使用できます。

この方法は、異なる表の同時ダイレクト・パス・ロード、または同じ表の異なるパーティションの同時ダイレクト・パス・ロードに適用できます。

1つのパーティションのダイレクト・パス・ロードを行う場合は、次のことを考慮します。

- ローカル索引は、ロードによってメンテナンスされます。
- グローバル索引は、ロードではメンテナンスできません。
- 参照整合性およびCHECK制約は使用禁止にする必要があります。
- トリガーは使用禁止にする必要があります。
- 入力データは事前にパーティション化する必要があります(パーティション化しない場合、多くのレコードが拒否され、パフォーマンスが低下します。)

親トピック: ダイレクト・パス・ロードのパフォーマンスの最適化

12.10.4 ダイレクト・パスによるセグメント内同時処理

SQL*Loaderでは、複数のセッションを同時に実行して、同一の表またはパーティション表の同一パーティションに対してダイレクト・パス・ロードを実行できます。

複数のSQL*Loaderセッションを実行すると、システムで使用可能なリソースを与えられればダイレクト・パス・ロードのパフォーマンスが向上します。

このデータ・ロード方法は、DIRECTおよびPARALLELパラメータにTRUEを設定することによって使用でき、「パラレル・ダイレクト・パス・ロード」とも呼ばれます。

並列化はユーザーによって管理されるものだということを理解しておいてください。PARALLELパラメータにTRUEを設定した場合、複数の同時ダイレクト・パス・ロード・セッションのみが可能になります。

親トピック: ダイレクト・パス・ロードのパフォーマンスの最適化

12.10.5 パラレル・ダイレクト・パス・ロードの制限

パラレル・ダイレクト・パス・ロードには制限があります。

パラレル・ダイレクト・パス・ロードには次の制限があります。

- ローカル索引もグローバル索引もロードによってメンテナンスできません。
- 行は追加(APPEND)のみできます。REPLACE、TRUNCATEおよびINSERTは使用できません(これは、個別のロードの間には整合性がないためです)。パラレル・ロードの前に表を切り捨てる必要がある場合は、手動で行ってください。

さらに、パラレル・ダイレクト・パス・ロードでは次のオブジェクトを使用禁止にする必要があります。これらを使用禁止にするための操作は不要です。SQL*Loaderではロードの開始前にこれらを使用禁止にし、ロードの完了後にこれらを使用可能に戻します。

- 参照整合性制約
- トリガー
- CHECK制約(ENABLE_CHECK_CONSTRAINTS制御ファイル・オプションが使用される場合を除く)

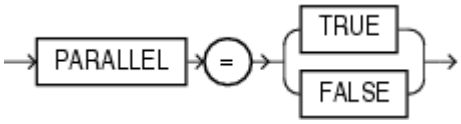
1つのパーティションのパラレル・ダイレクト・パス・ロードを行う場合は、まず、データをパーティション化してください(そうしない場合は、パーティション不一致によるレコード拒否のオーバーヘッドのため、ロード速度が遅くなります)。

親トピック: ダイレクト・パス・ロードのパフォーマンスの最適化

12.10.6 複数のSQL*Loaderセッションの初期化

入力元となるデータ・ファイルは、SQL*Loaderセッションごとに異なります。同じ表にダイレクト・ロードを実行するセッションすべてに対して、PARALLEL句にTRUEを設定する必要があります。

構文は次のとおりです。



PARALLELは、コマンドラインまたはパラメータ・ファイルに指定できます。また、OPTIONS句を使用して制御ファイルに指定することもできます。

たとえば、1つの表について3つのSQL*Loaderダイレクト・パス・ロード・セッションを開始するには、オペレーティング・システムのプロンプトで、次の各コマンドを実行します。コマンドをそれぞれ入力した後に、パスワードを入力するように要求されます。

```
sqlldr USERID=scott CONTROL=load1.ctl DIRECT=TRUE PARALLEL=TRUE
sqlldr USERID=scott CONTROL=load2.ctl DIRECT=TRUE PARALLEL=TRUE
sqlldr USERID=scott CONTROL=load3.ctl DIRECT=TRUE PARALLEL=TRUE
```

このコマンドは、別々のセッションで実行するか、またはオペレーティング・システムがサポートしている場合には別々のバックグラウンド・ジョブとして実行してください。複数の制御ファイルを使用していることに注意してください。そうすることによって、ダイレクト・パス・ロードで使用するファイルをより柔軟に指定できます。

> ノート:
>
> パラレル・ロード時には、索引はメンテナンスされません。ロード完了後に、索引をすべて手動で(再)作成または再構築する必要があります。パラレル・ロード後に大きな索引を構築する場合、パラレル索引作成機能またはパラレル索引再構築機能を使用すると処理を高速化できます。

PARALLEL句を使用してロードを実行すると、SQL*Loaderによって同時実行セッションごとに一時セグメントが作成されます。それらの一時セグメントは、ロード完了時に マージされます。マージによって作成されたセグメントは、セグメントの最高水位標より上にあるデータベースの既存のセグメントに追加されます。各ローダー・セッションの各セグメントで使用された最後のエクステントは、空き領域をすべて切り捨ててから、SQL*Loaderセッションの他のエクステントと組み合せることができます。

親トピック: ダイレクト・パス・ロードのパフォーマンスの最適化

12.10.7 パラレル・ダイレクト・パス・ロードのパラメータ

パラレル・ダイレクト・パス・ロードを実行する場合、ローダーによって割り当てられる一時セグメントの属性を指定してできるオプションがあります。

これらのオプションは、FILEおよびSTORAGEパラメータを使用して指定します。これらのパラメータはパラレル・ロードに対してのみ有効です。

- FILEパラメータを使用した一時セグメントの指定
  最大の入出力スループットを得るために、同時実行のダイレクト・パス・ロード・セッションで、各ファイルを別のディスクに置いて使用することをお薦めします。
- FILEパラメータの使用

このトピックでは、FILEパラメータの使用について説明します。

- STORAGEパラメータの使用方法
  STORAGEパラメータを使用して、パラレル・ダイレクト・パス・ロード用に割り当てられる一時セグメントの記憶域属性を指定できます。

親トピック: ダイレクト・パス・ロードのパフォーマンスの最適化

12.10.7.1 FILEパラメータを使用した一時セグメントの指定

最大の入出力スループットを得るために、同時実行のダイレクト・パス・ロード・セッションで、各ファイルを別のディスクに置いて使用することをお薦めします。

SQL*Loader制御ファイルでは、ロードされるオブジェクト(表またはパーティション)の表領域にある有効なデータ・ファイルであれば、OPTIONS句のFILEパラメータを使用してファイル名を指定できます。

たとえば:

```
LOAD DATA
INFILE 'load1.dat'
INSERT INTO TABLE emp
OPTIONS(FILE='/dat/data1.dat')
(empno POSITION(01:04) INTEGER EXTERNAL NULLIF empno=BLANKS
...
```

同時実行する各SQL*Loaderセッションのコマンドラインでも、FILEパラメータを指定できます。ただし、その指定は、そのセッションでロードされるすべてのオブジェクトにグローバルに適用されます。

親トピック: パラレル・ダイレクト・パス・ロードのパラメータ

12.10.7.2 FILEパラメータの使用

このトピックでは、FILEパラメータの使用について説明します。

パラレル・ダイレクト・パス・ロードでは、Oracle DatabaseのFILEパラメータに次の制限があります。

- 非パーティション表の場合: 指定されたファイルは、ロードする表と同じ表領域に存在する必要があります。
- パーティション表の1つのパーティションをロードする場合: 指定したファイルは、ロードするパーティションの表領域に存在する必要があります。
- パーティション表の表全体をロードする場合: 指定されたファイルは、ロードするすべてのパーティションと同じ表領域に存在する必要があります。つまり、すべてのパーティションは同じ表領域に存在する必要があります。

親トピック: パラレル・ダイレクト・パス・ロードのパラメータ

12.10.7.3 STORAGEパラメータの使用方法

STORAGEパラメータを使用して、パラレル・ダイレクト・パス・ロード用に割り当てられる一時セグメントの記憶域属性を指定できます。

STORAGEパラメータが使用されない場合は、ロードされるオブジェクト(表、パーティション)が存在するセグメントの記憶域属性が使用されます。また、STORAGEパラメータが指定されてない場合は、SQL*LoaderでEXTENTS用にデフォルトの2KBが使用されます。

たとえば、STORAGEパラメータを指定するためには、次のOPTIONS句が使用されます。

```
OPTIONS (STORAGE=(INITIAL 100M NEXT 100M PCTINCREASE 0))
```

STORAGEパラメータを使用できるのは、SQL*Loader制御ファイル内のみでコマンドラインでは使用できません。STORAGEパラメータは、PCTINCREASEを0(ゼロ)に設定する、INITIALまたはNEXT値を設定する以外には、使用しないでください。暗黙的に無視される可能性があります。

親トピック: パラレル・ダイレクト・パス・ロードのパラメータ

12.10.8 パラレル・ダイレクト・パス・ロード後の制約の使用可能化

すべてのデータのロード完了後に、制約およびトリガーを手動で使用可能にしてください。

それぞれのSQL*Loaderセッションで、ダイレクト・パス・ロードの後に表の制約が使用可能に戻されることがあるため、あるセッションで、他のセッションがデータのロードを終える前に、制約が使用可能に戻される可能性があります。この場合、ロードを完了する最初のセッションでは、残りのセッションで表のロックが共有されているため、制約を使用可能にできなくなります。

ダイレクト・パス・ロードの後、一部の制約が使用可能に戻っていない可能性があるため、ロードの完了後、制約の状態を調べ、制約が使用可能になったことを確認する必要があります。

親トピック: ダイレクト・パス・ロードのパフォーマンスの最適化

12.10.9 主キー制約および一意キー制約

このトピックでは、主キー制約および一意キー制約の使用について説明します。

PRIMARY KEY制約およびUNIQUE KEY制約が有効である場合、表に索引が作成されます。表が非常に大きい場合は、ダイレクト・パス・ロード後に索引を使用可能にするまでに非常に長い時間がかかる可能性があります。ロード後に、これらの制約を手動で有効にしてください(また、自動で有効にする機能を指定しないでください)。これによって、必要な索引をパラレルに手動で作成し、制約を有効にするまでの時間を節約できます。

親トピック: ダイレクト・パス・ロードのパフォーマンスの最適化

12.11 一般的なパフォーマンス改善のヒント

パラレル・データ・ロードでSQL*Loaderを使用する場合に一般的なパフォーマンスの改善を有効にする方法について説明します。

ロードするデータの形式について制御が可能な場合は、次に示すヒントを利用してロード・パフォーマンスを改善できます。

- 論理レコードの処理を効率化します。
  - 物理レコードと論理レコードの1対1のマップを使用します(CONTINUEIFおよびCONCATENATEを使用しない)。
  - ソフトウェアで物理レコードの境界を簡単に判断できるようにします。ファイル処理オプション文字列「FIX nnn」または「VAR」を使用します。ほとんどのプラットフォーム(UNIX、NTなど)では、デフォルト(ストリーム・モード)を使用する場合、ローダーで各物理レコードをスキャンしてレコード終了記号(改行文字)を探す必要があります。
- フィールド設定を効率化します。

  フィールド設定とは、データ・ファイルのフィールドを、ロードされる表の対応する列にマップする処理です。マップ機能は、制御ファイルのフィールド記述によって制御されます。フィールド設定は(データ変換とともに)、ほとんどのロードでCPUサイクルを最も使用する処理です。
  - デリミタ付きのフィールドを使用しないようにします。固定位置のフィールドを使用します。デリミタ付きフィールド

を使用すると、ローダーで入力データをスキャンしてデリミタを見つけなければなりません。固定位置フィールドを使用すると、フィールド設定は単純なポインタの算出によって(非常に速く)行われます。

- (PRESERVE BLANKSを使用する)必要がない場合は、空白を切り捨てないでください。

- 変換を効率化します。

  SQL*Loaderでは、文字セット変換、データ型変換などのいくつかの変換が行われます。このような変換が行われない場合、処理は最も速くなります。

  - 可能な場合は、シングルバイト文字セットを使用します。
  - 文字セット変換はできるだけ行わないようにします。SQL*Loaderでは、次の4つの文字セットがサポートされています。
    - クライアント文字セット(クライアントsqlldrプロセスのNLS_LANG)
    - データ・ファイル文字セット(通常、クライアント文字セットと同じ)
    - データベース文字セット
    - データベース文字セット

    すべての文字セットが同じ場合、パフォーマンスは最適化されます。ダイレクト・パスでは、データ・ファイル文字セットとデータベース・サーバー文字セットが同じである場合、最適なパフォーマンスが得られます。文字セットが同じ場合、文字セット変換用バッファは割り当てられません。

- ダイレクト・パス・ロードを使用します。
- SORTED INDEXS句を使用します。
- NULLIF句およびDEFAULTIF句は必要な場合以外は使用しないようにします。これらの句は、関連する句を持つ各列にロードされるすべての行について、評価される必要があります。
- 可能な場合は、パラレル・ダイレクト・パス・ロードおよびパラレル索引作成を使用します。
- CONCATENATE句およびCOLUMNARRAYROWS句の両方に大きい値を指定する場合は、パフォーマンスへの影響に注意します。

関連トピック

- CONCATENATEを使用した論理レコードの作成

親トピック: 従来型パス・ロードおよびダイレクト・パス・ロード

13 SQL*Loaderエクスプレス

SQL*Loaderエクスプレス・モードでは、SQL*Loaderを使用して単純なデータ型を迅速かつ簡単にロードできます。

- SQL*Loaderエクスプレス・モードとは
  SQL*Loaderエクスプレス・モードでは、表の列がすべて文字、数値または日時データ型で、入力データ・ファイルにデリミタ付き文字データのみが含まれる場合、表名のみを指定することで迅速にロードを実行できます。
- SQL*Loaderエクスプレス・モードの使用方法
  SQL*Loaderエクスプレス・モードをアクティブ化するには、ユーザー名と表名を指定するだけです。
- SQL*Loaderエクスプレス・モードのパラメータ・リファレンス
  この項では、SQL*Loaderエクスプレス・モードで使用可能なパラメータについて説明します。
- SQL*Loaderエクスプレス・モードの構文図
  SQL*Loaderエクスプレス・モードのオプションについて理解するために、ここに示した図形式の構文ガイド(線路図またはDDL構文図と呼ぶこともあります)を参照してください。

親トピック: SQL*Loader

13.1 SQL*Loaderエクスプレス・モードとは

SQL*Loaderエクスプレス・モードでは、表の列がすべて文字、数値または日時データ型で、入力データ・ファイルにデリミタ付き文字データのみが含まれる場合、表名のみを指定することで迅速にロードを実行できます。

エクスプレス・モードでは、SQL*Loader制御ファイルは使用されません。かわりに、SQL*Loaderでは、`ALL_TAB_COLUMNS`ビューにある表の列定義を使用して入力フィールドの順序とデータ型を判別します。他のほとんどの設定については、デフォルト値が想定されますが、これはコマンドライン・パラメータで上書きできます。

> ノート:
>
> SQL*Loader エクスプレス・モードで使用できる有効なパラメータは、この章で説明しているパラメータのみです。他の任意のパラメータは、無視されるか、状況によってはエラーの原因となります。

親トピック: SQL*Loaderエクスプレス

13.2 SQL*Loaderエクスプレス・モードの使用方法

SQL*Loaderエクスプレス・モードをアクティブ化するには、ユーザー名と表名を指定するだけです。

次のように、SQL*Loaderによってパスワードの入力を要求されます。

```
> sqlldr username TABLE=employees
Password:
.
.
.

SQL*Loader: Release 18.0.0.0.0 - Production on Mon Oct 16 127:19:39 2017
Version 18.1.0.0.0
Copyright (c) 1982, 2017, Oracle and/or its affiliates.  All rights reserved.

Express Mode Load, Table: EMPLOYEES
.
```

```
.
.
```

TABLEパラメータのみを指定してSQL*Loaderエクスプレス・モードをアクティブ化すると、他の複数のパラメータにはデフォルト設定が使用されます。ほとんどのデフォルト値は、コマンドラインで追加パラメータを指定して上書きできます。

SQL*Loaderエクスプレス・モードでは、SQL*Loader制御ファイルを含むログ・ファイルが生成されます。ログ・ファイルには、外部表を作成し、SQL INSERT AS SELECT文を使用してロードを実行するためのSQLスクリプトも含まれます。制御ファイルもSQLスクリプトもSQL*Loaderエクスプレス・モードでは使用されません。これらは、通常のSQL*Loaderまたはスタンドアロンの外部表を使用して操作を実行する場合の開始ポイントとして使用できます(制御ファイルはSQL*Loaderで使用し、SQLスクリプトはスタンドアロンの外部表の操作で使用します)。

- SQL*Loaderエクスプレス・モードで使用されるデフォルト値
  SQL*Loaderエクスプレスで表をロードする方法、使用されるデフォルト値、デフォルト値を変更する条件について説明します。
- SQL*Loaderエクスプレス・モードによるバイト順序の処理方法
  データ・ファイルで使用されている文字セットのタイプは、SQL*Loaderエクスプレスで使用されるバイト順序に影響します。

関連項目:

- SQL*Loaderエクスプレス・モードのパラメータ・リファレンス
- 制御ファイルの詳細は、「SQL*Loader制御ファイル・リファレンス」を参照してください

親トピック: SQL*Loaderエクスプレス

13.2.1 SQL*Loaderエクスプレス・モードで使用されるデフォルト値

SQL*Loaderエクスプレスで表をロードする方法、使用されるデフォルト値、デフォルト値を変更する条件について学習します。

デフォルトでは、特に指定しないかぎり、SQL*Loaderエクスプレス・モードを使用して実行されるロードでは次のように想定されます。

- データ・ファイルを指定しない場合、カレント・ディレクトリでtable-name.datというファイルが検索されます。
- デフォルトでは、SQL*Loaderエクスプレスは、外部表のロード方法を使用します。ただし、一部のエラーでは、SQL*Loaderエクスプレス・モードによって、デフォルトの外部表のロード方法からダイレクト・パス・ロードに自動的に切り替えられます。これが発生する例として、権限違反が原因でCREATE DIRECTORY SQLコマンドが失敗した場合があげられます。
- SQL*Loaderエクスプレスのフィールドは、次のように設定されます。
  - 表の列名に基づく名前(フィールドの順序は表の列の順序に一致)
  - 表の列型に基づく型
  - 改行(レコード・デリミタとして)
  - カンマ(フィールド・デリミタとして)
  - 囲みなし
  - 左右切捨て

- DEGREE_OF_PARALLELISMパラメータはAUTOに設定されます。
- 日付およびタイムスタンプの書式ではNLS設定が使用されます。
- NLSクライアント文字セットが使用されます。
- すでに表にデータがある場合、新しいデータは表に追加されます。
- データ・ファイルを指定していない場合、データ・ファイル、ログ・ファイルおよび不良ファイルには、次のデフォルト名が使用されます(%pは、Oracle Database子プロセスのプロセスIDに置き換えられます)。
  - table-name.dat (データ・ファイル)
  - table-name.log (SQL*Loaderログ・ファイル)
  - table-name_%p.log_xt (Oracle Databaseログ・ファイル。emp_17228.log_xtなど)
  - table-name_%p.bad (不良ファイル)
- DATAパラメータを使用して、1つ以上のデータファイルを指定すると、ログ・ファイルと不良ファイルには次のデフォルト名が付けられます(%pは、サーバーの子プロセスのプロセスIDに置き換えられます)。
  - table-name.log (SQL*Loaderログ・ファイル)
  - table-name_%p.log_xt (Oracle Databaseログ・ファイル)
  - first-data-file_%p.bad (不良ファイル)

関連トピック

- DATA

親トピック: SQL*Loaderエクスプレス・モードの使用方法

13.2.2 SQL*Loaderエクスプレス・モードによるバイト順序の処理方法

データ・ファイルで使用されている文字セットのタイプは、SQL*Loaderエクスプレスで使用されるバイト順序に影響します。

一般的に、SQL*Loaderエクスプレス・モードでは、SQL*Loader制御ファイルを使用して実行されるロードと同じ方法でバイト順序マークが処理されます。

まとめると、次のようになります。

- Unicode文字セットを含むデータ・ファイルでは、SQL*Loaderエクスプレス・モードによってファイルの先頭にあるバイト順序マークが確認されます。
- UTF16データ・ファイルでは、バイト順序マークが見つかると、バイト順序マークによってデータ・ファイルのバイト順序が設定されます。バイト順序マークが見つからない場合、SQL*Loaderが実行されているシステムのバイト順序がデータ・ファイルで使用されます。
- UTF16データ・ファイルは、バイト順序(エンディアン)が、SQL*Loaderエクスプレスが実行されているシステムと同じバイト順序であるかどうかにかかわらず、ロードできます。
- UTF8データ・ファイルでは、どのバイト順序マークが見つかってもスキップされます。
- 複数のデータ・ファイルが関与し、それらが異なるバイト順序を使用している場合、ロードは中断されます。

関連トピック

- SQL*Loaderによるバイト順序の管理方法の理解

13.3 SQL*Loaderエクスプレス・モードのパラメータ・リファレンス

この項では、SQL*Loaderエクスプレス・モードで使用できるパラメータについて説明します。

一部のパラメータの名前は通常のSQL*Loaderで使用されるパラメータと同じですが、動作が異なる場合があります。説明をよく読んで、想定される動作を確認してください。

> ノート:
>
> パラメータ値に引用符が含まれる場合、それらはパラメータ・ファイルで指定することをお薦めします。データ・ポンプ・エクスポート・コマンドライン・モードで使用可能なパラメータの「データ・ポンプ・コマンドラインでの引用符の使用」を参照してください。ここで説明する内容は、SQL*Loader エクスプレス・モードにも関係します。

- BAD
  SQL*Loaderエクスプレス・モードのBADパラメータでは、不良ファイルの場所と名前を指定します。
- CHARACTERSET
  SQL*Loaderエクスプレス・モードのCHARACTERSETパラメータでは、ロードに使用する文字セットを指定します。
- CSV
  SQL*Loaderエクスプレス・モードのCSVパラメータを使用すると、CSV形式のファイルに埋込みのレコード終了記号が付いたフィールドが含まれているかどうかを指定できます。
- DATA
  SQL*Loaderエクスプレス・モードのDATAパラメータでは、ロードするデータが含まれているデータ・ファイルの名前を指定します。
- DATE_FORMAT
  SQL*Loaderエクスプレス・モードのDATE_FORMATパラメータは、すべての日付フィールドのデフォルト値をオーバーライドする日付書式を指定します。
- DEGREE_OF_PARALLELISM
  SQL*Loaderエクスプレス・モードのDEGREE_OF_PARALLELISMパラメータは、ロードで使用する並列度を指定します。
- DIRECT
  SQL*Loaderエクスプレス・モードのDIRECTパラメータは、従来型パスまたはダイレクト・パスのどちらのロード方法を使用するかを指定します。
- DNFS_ENABLE
  SQL*Loaderエクスプレス・モードのDNFS_ENABLEパラメータを使用すると、SQL*Loader操作中の入力データ・ファイルに対するDirect NFSクライアントの使用を有効化および無効化できます。
- DNFS_READBUFFERS
  SQL*Loaderエクスプレス・モードのDNFS_READBUFFERSパラメータを使用すると、Direct NFSクライアントで使用される読取りバッファの数を制御できます。
- ENCLOSED_BY
  SQL*Loaderエクスプレス・モードのENCLOSED_BYパラメータでは、フィールドの囲み文字列を指定します。
- EXTERNAL_TABLE
  SQL*Loaderエクスプレス・モードのEXTERNAL_TABLEパラメータは、外部表のオプションを使用してデータをロード

するかどうかを決定します。

- FIELD_NAMES
  SQL*Loaderエクスプレス・モードのFIELD_NAMESパラメータでは、データベース表の列の順序に従っているフィールドを上書きします。
- LOAD
  SQL*Loaderエクスプレス・モードのLOADパラメータでは、ロードするレコードの数を指定します。
- NULLIF
  SQL*Loaderエクスプレス・モードのNULLIFパラメータは、フィールドをNULL列としてロードするかどうかを決定するために使用する値を指定します。
- OPTIONALLY_ENCLOSED_BY
  SQL*Loaderエクスプレス・モードのOPTIONALLY_ENCLOSED_BYでは、オプションのフィールドの囲み文字列を指定します。
- PARFILE
  SQL*Loaderエクスプレス・モードのPARFILEパラメータは、頻繁に使用するコマンドライン・パラメータを格納したファイルの名前を指定します。
- SILENT
  SQL*Loaderエクスプレス・モードのSILENTパラメータでは、SQL*Loaderの操作中に画面に書き込まれるコンテンツを抑止します。
- TABLE
  SQL*Loaderエクスプレス・モードのTABLEパラメータでは、SQL*Loaderエクスプレス・モードをアクティブ化します。
- TERMINATED_BY
  SQL*Loaderエクスプレス・モードのTERMINATED_BYでは、デフォルトに優先するフィールド終端文字を指定します。
- TIMESTAMP_FORMAT
  TIMESTAMP_FORMATパラメータでは、ロードの際に使用するタイムスタンプの書式を指定します。
- TRIM
  SQL*Loaderエクスプレス・モードのTRIMパラメータでは、ロード中に使用するフィールド切捨てのタイプを指定します。
- USERID
  SQL*Loaderエクスプレス・モードのUSERIDを使用すると、Oracleユーザー名とパスワードを指定できるようになり、その入力要求が表示されなくなります。

親トピック: SQL*Loaderエクスプレス

13.3.1 BAD

SQL*Loaderエクスプレス・モードのBADパラメータでは、不良ファイルの場所と名前を指定します。

デフォルト

デフォルトは、DATAパラメータを使用してデータ・ファイルが指定されているかどうかによって異なります。

用途

BADパラメータでは、不良ファイルの場所と名前を指定します。

構文

```
BAD=[directory/][filename]
```

使用上のノート

不良ファイルには、挿入時にエラーの原因になったレコードや、不適切に書式設定されたレコードが格納されます。BADパラメータを指定する場合は、ディレクトリとファイル名(またはそのどちらか)を指定する必要があります。BADパラメータを指定しない場合に、拒否レコードが存在すると、デフォルトのファイル名が使用されます。

directory変数には、不良ファイルの書込み先となるディレクトリを指定します。指定には、デバイスまたはネットワーク・ノードの名前を含めることができます。

filename変数には、プラットフォームで有効とみなされるファイル名を指定します。名前(および.bad以外のものを使用する場合は拡張子)のみを指定する必要があります。ファイル名に空白やその他の句読点文字が含まれている場合は、一重引用符で囲む必要があります。

directoryおよびfilenameの値は、次のように決定されます。

- BADパラメータでファイル名を指定しているがディレクトリはない場合、ディレクトリはデフォルトで現在のディレクトリになります。
- BADパラメータでディレクトリを指定しているがファイル名はない場合は、指定されたディレクトリが使用され、ファイル名と拡張子にはデフォルトが使用されます。

BADパラメータは、指定したDATAパラメータと一致するファイルのすべてに適用されます(DATAパラメータを指定していた場合)。DATAパラメータを指定していない場合、BADパラメータは1つのデータ・ファイル(table-name.dat)に適用されます。

注意:

- ファイル名(デフォルトまたは指定した名前)がすでに存在する場合、そのファイル名は使用しているオペレーティング・システムに応じて、上書きされるか、新しいバージョンが生成されます。
- 複数のデータ・ファイルがロードされる状況では、BAD パラメータを指定しないか、または不良ファイルのディレクトリのみを指定することをお薦めします。

例

次の指定では、カレント・ディレクトリにemp1.badという不良ファイルを作成します。

```
> sqlldr hr TABLE=employees BAD=emp1
```

親トピック: SQL*Loaderエクスプレス・モードのパラメータ・リファレンス

13.3.2 CHARACTERSET

SQL*LoaderエクスプレスのCHARACTERSETパラメータでは、ロードに使用する文字セットを指定します。

デフォルト

環境変数NLS_LANGで指定されたNLSクライアント文字セット

用途

CHARACTERSETパラメータでは、ロードに使用するデフォルト以外の文字セットを指定します。

構文

```
CHARACTERSET=character_set_name
```

character_set_name変数で、文字セット名を指定します。通常、指定された名前は、Oracle Databaseがサポートしている文字セットの名前である必要があります。

使用上のノート

CHARACTERSETパラメータでは、SQL*Loaderの入力データ・ファイルの文字セットを指定します。CHARACTERSETパラメータが指定されていない場合、すべてのデータ・ファイルに対するデフォルトの文字セットは、環境変数NLS_LANGで定義されたセッション文字セットです。文字データ(SQL*Loaderデータ型のCHAR、VARCHAR、VARCHARC、数値型EXTERNALおよび日時データ型と期間データ型のフィールド)のみが、データ・ファイルの文字セットに影響されます。

UnicodeのUTF-16エンコーディングの場合は、名前はAL16UTF16でなくUTF16を使用します。AL16UTF16 (UTF-16エンコードされたデータに対してサポートされる文字セット名)は、ビッグ・エンディアンのバイト順序のUTF-16データにのみ使用されます。ただし、データ・ファイルを作成したシステムのバイト順序を使用してデータを設定することが許可されているため、データ・ファイル内のデータはビッグ・エンディアンまたはリトル・エンディアンのいずれかにできます。このため、異なる文字セット名(UTF16)が使用されます。また、文字セット名AL16UTF16もサポートされています。ただし、リトル・エンディアンのバイト順序のデータ・ファイルにAL16UTF16を指定すると、SQL*Loaderによって警告メッセージが発行され、データ・ファイルはリトル・エンディアンとして処理されます。

CHARACTERSETパラメータ値は、すべてのデータ・ファイルで同じであるとみなされます。

> ノート:
>
> UTF-16 という用語は、一般的に Unicode の UTF-16 エンコーディングを意味します。UTF16(ハイフンなし)という用語は、文字セットの固有の名前で、UTF-16 エンコーディングの使用時に CHARACTERSET パラメータに指定する必要のある用語です。UTF-8 および UTF8 の場合も同様です。

制限事項

ありません。

例

次の例では、UTF-8の文字セットを指定します。

```
> sqlldr hr TABLE=employees CHARACTERSETNAME=utf8
```

親トピック: SQL*Loaderエクスプレス・モードのパラメータ・リファレンス

13.3.3 CSV

SQL*Loaderエクスプレス・モードのCSVパラメータを使用すると、CSV形式のファイルに埋込みのレコード終了記号が付いたフィールドが含まれているかどうかを指定できます。

デフォルト

コマンドラインでCSVパラメータを指定しない場合、SQL*Loaderエクスプレスでは、ロードされるCSVファイルに埋込み文字や囲み文字を含まないデータが存在するものと想定されます。

コマンドラインでCSV=WITHOUT_EMBEDDEDを指定する場合、SQL*Loaderエクスプレスでは、ロードされるCSVファイルに埋込み文字を含まないデータと、オプションで「"」に囲まれたデータが存在するものと想定されます。

用途

CSVパラメータによって、ロードされるカンマ区切り値(CSV)形式のファイルにレコード終了記号の埋込みフィールドが含まれるかどうかを指定するオプションが提供されます。

構文

```
CSV=[WITH_EMBEDDED | WITHOUT_EMBEDDED]
```

- WITH_EMBEDDED — このオプションは、レコードのフィールドにレコード終了記号が含まれる(埋め込まれている)場合があることを示します。レコード終了記号は改行です。デフォルトのデリミタは、TERMINTATED by ","およびOPTIONALLY_ENCLOSED_BY '"'です。埋込みのレコード終了記号は囲む必要があります。
- WITHOUT_EMBEDDED—このオプションは、レコードのフィールドにレコード終了記号が含まれない(埋め込まれていない)ことを示します。レコード終了記号は改行です。デフォルトのデリミタは、TERMINATED BY ","およびOPTIONALLY_ENCLOSED_BY ' " 'です。

使用上のノート

CSVファイルに埋込みのレコード終了記号が多く含まれると、このパラメータによってパフォーマンスが悪影響を受ける可能性があります。

制限事項

- 通常、ファイルはパラレルに処理できます(同時に複数の実行サーバーによって分割および処理されます)。ただし、埋込みのレコード終了記号を含むCSV形式のファイルの場合、そのファイルはただ1つの実行サーバーによって処理される必要があります。そのため、CSVパラメータをCSV=WITH_EMBEDDEDに設定すると、データ・ファイル内のパラレル処理が無効になります。

例

次の例では、埋込みのレコード終了記号を含むCSV形式のファイルとしてデータ・ファイルを処理します。

```
> sqlldr hr TABLE=employees CSV=WITH_EMBEDDED
```

親トピック: SQL*Loaderエクスプレス・モードのパラメータ・リファレンス

13.3.4 DATA

SQL*LoaderエクスプレスのDATAパラメータでは、ロードするデータが含まれているデータ・ファイルの名前を指定します。

デフォルト

表名と同じ名前ですが、拡張子は.datになります。

用途

DATAパラメータは、ロードするデータが含まれているデータ・ファイルの名前を指定します。

構文

```
DATA=data-file-name
```

ファイル拡張子を指定しない場合、デフォルトで.datになります。

使用上のノート

ファイルの指定にはワイルド・カードを含めることができますが、ファイル名とファイル拡張子にのみ可能で、デバイス名やディレクトリ名には不可能です。アスタリスク(*)は、複数の文字を表します。疑問符(?)は、任意の1文字を表します。たとえば:

```
DATA='emp*.dat'
DATA='m?emp.dat'
```

複数のデータ・ファイルの指定(それぞれにワイルド・カードを含めることが可能)をリストするには、カンマでファイル名を区切る必要があります。

ファイル名に任意の特殊文字(空白、*、?など)が含まれる場合、名前全体を一重引用符で囲む必要があります。

次に、DATAパラメータの有効な3つの使用例を示します(ファイル名に特殊文字が含まれる場合のみ、一重引用符が必要です)。

```
DATA='file1','file2','file3','file4','file5','file6'

DATA='file1','file2'
DATA='file3,'file4','file5'
DATA='file6'

DATA='file1'
DATA='file2'
DATA='file3'
DATA='file4'
DATA='file5'
DATA='file6'
```

注意:

複数のデータ・ファイルがロードされる状況で BAD パラメータも指定する場合、ファイル名ではなく、不良ファイルのディレクトリのみを指定することをお薦めします。指定した名前を持つファイルがすでに存在する場合、使用しているオペレーティング・システムに応じて、そのファイルが上書きされるか、新しいバージョンが生成されます。

例

カレント・ディレクトリに emp1.dat、emp2.dat、m1emp.datおよびm2emp.datという名前のデータ・ファイルが含まれている状態で、次のコマンドを発行するとします。

```
> sqlldr hr TABLE=employees DATA='emp*','m1emp'
```

コマンドによって、ファイルemp1.dat、emp2.datおよびm1emp.datがロードされます。ファイルm2emp.datは、どのワイルドカード基準にも一致しないため、ロードされません。

親トピック: SQL*Loaderエクスプレス・モードのパラメータ・リファレンス

13.3.5 DATE_FORMAT

SQL*LoaderエクスプレスのDATE_FORMATパラメータは、すべての日付フィールドのデフォルト値をオーバーライドする日付書式を指定します。

デフォルト

DATE_FORMATパラメータが指定されていない場合、環境変数NLS_DATE_FORMAT、NLS_LANGUAGEまたはNLS_DATE_LANGUAGEの設定が使用されます(SQL*Loaderセッションに対して定義されている場合)。NLS_DATE_FORMATが定義されていない場合、日付は、NLS_TERRITORYの設定によって定義されるデフォルト書式であるとみなされます。

用途

DATE_FORMATパラメータでは、すべての日付フィールドのデフォルト値に優先する日付書式を指定します。

構文

```
DATE_FORMAT=mask
```

maskは、通常は二重引用符で囲まれる日付書式マスクです。

例

データ・ファイルの日付が2019年6月25日の場合、日付書式は次のように指定します。

```
> sqlldr hr TABLE=employees DATE_FORMAT="DD-Month-YYYY"
```

親トピック: SQL*Loaderエクスプレス・モードのパラメータ・リファレンス

13.3.6 DEGREE_OF_PARALLELISM

SQL*LoaderエクスプレスモードのDEGREE_OF_PARALLELISMパラメータは、ロードで使用する並列度を指定します。

デフォルト

```
AUTO
```

用途

DEGREE_OF_PARALLELISMパラメータでは、ロードで使用する並列度を指定します。

構文

```
DEGREE_OF_PARALLELISM=[degree-num|DEFAULT|AUTO|NONE]
```

オプション:

- degree-numでは、並列度を指定します。1からnまでの整数値にする必要があります。
- DEFAULTでは、データベースのデフォルトの並列度(デフォルトのパラメータ値AUTO以外)を指定します。
- AUTOでは、ロードの並列度をOracle Databaseが自動的に設定するように指定します。DEGREE_OF_PARALLELISMパラメータを指定していない場合、これが並列度のデフォルト・オプションになります。
- NONEでは、ロードの並列実行が不要なことを指定します。値NONEは、値1と同じです。

制限事項

- DEGREE_OF_PARALLELISMパラメータは、従来型パスまたはダイレクト・パスのロード方法を強制的に実行する場合は無視されます(NONEオプションが使用されます)。どの値でDEGREE_OF_PARALLELISMパラメータを指定しても、常にこの内容を示すメッセージが表示されます。
- ロードがデフォルトの外部表ロードで、SQL*Loaderエクスプレス・モードでかわりにダイレクト・パス・ロードを使用することになるエラーが発生した場合、並列度を指定したり、AUTOで外部表のデフォルトを受け入れていても、ジョブはパラレルに実行されません。この変更を示すアラート・メッセージが表示されます。

例

次の例では、並列度を使用せずにデータをロードします。

```
> sqlldr hr TABLE=employees DEGREE_OF_PARALLELISM=NONE
```

関連トピック

- パラレル実行の概念

親トピック: SQL*Loaderエクスプレス・モードのパラメータ・リファレンス

13.3.7 DIRECT

SQL*LoaderエクスプレスのDIRECTパラメータは、従来型パスまたはダイレクト・パスのどちらのロード方法を使用するかを指定します。

デフォルト

デフォルトなし。

用途

DIRECTパラメータでは、従来型パスまたはダイレクト・パスのどちらのロード方法を使用するかを指定します。

構文

```
DIRECT=[TRUE|FALSE]
```

TRUE値はダイレクト・パス・ロードを指定します。FALSE値は従来型パス・ロードを指定します。

使用上のノート

このパラメータは、SQL*Loaderエクスプレス・モードの外部表のデフォルトのロード方法を上書きします。

一部のエラーでは、SQL*Loaderエクスプレス・モードによって、デフォルトの外部表のロード方法からダイレクト・パス・ロードに自動的に切り替えられます。これが発生する例として、権限違反が原因でCREATE DIRECTORY SQLコマンドが失敗した場合があげられます。

DIRECTパラメータを使用し、従来型パス・ロードまたはダイレクト・パス・ロードを指定する場合、次の標準のSQL*Loaderパラメータをエクスプレス・モードで使用できます。

- BINDSIZE
- COLUMNARRAYROWS (ダイレクト・パス・ロードのみ)
- DATE_CACHE
- ERRORS
- MULTITHREADING (ダイレクト・パス・ロードのみ)
- NO_INDEX_ERRORS (ダイレクト・パス・ロードのみ)
- RESUMABLE
- RESUMABLE_NAME
- RESUMABLE_TIMEOUT
- ROWS
- SKIP
- STREAMSIZE

例

次の例では、SQL*Loaderは、ロードで外部表のかわりにダイレクト・パス・ロード方法を使用します。

```
> sqlldr hr TABLE=employees DIRECT=TRUE
```

親トピック: SQL*Loaderエクスプレス・モードのパラメータ・リファレンス

13.3.8 DNFS_ENABLE

SQL*LoaderエクスプレスのDNFS_ENABLEパラメータを使用すると、SQL*Loader操作中の入力データ・ファイルに

対するDirect NFSクライアントの使用を有効化および無効化できます。

デフォルト

TRUE

用途

DNFS_ENABLEパラメータでは、SQL*Loader操作中の入力データ・ファイルに対するDirect NFSクライアントの使用を有効化および無効化できます。

Direct NFSクライアントは、Oracleがファイル・サーバー上のファイルにアクセスする際のパフォーマンスを向上するために、ファイル・サーバーで実装できるAPIです。

構文

構文は次のとおりです。

```
DNFS_ENABLE=[TRUE|FALSE]
```

使用上のノート

SQL*Loaderでは、1GBを超えるデータ・ファイルを読み取る場合にデフォルトでDirect NFSクライアント・インタフェースが使用されます。よりサイズの小さいファイルでは、オペレーティング・システムのI/Oインタフェースが使用されます。すべての入力データ・ファイルでDirect NFSクライアントを使用するには、DNFS_ENABLE=TRUEを使用します。

すべてのデータ・ファイルでDirect NFSクライアントの使用を無効にするには、DNFS_ENABLE=FALSEを指定します。

DNFS_ENABLEパラメータをDNFS_READBUFFERSパラメータと組み合せて使用すると、Direct NFSクライアントによって使用される読取りバッファの数を指定できます。

親トピック: SQL*Loaderエクスプレス・モードのパラメータ・リファレンス

13.3.9 DNFS_READBUFFERS

SQL*LoaderエクスプレスモードのDNFS_READBUFFERSパラメータを使用すると、Direct NFSクライアントで使用される読取りバッファの数を制御できます。

デフォルト

4

用途

DNFS_READBUFFERSパラメータでは、Direct NFSクライアントによって使用される読取りバッファの数を制御できます。Direct NFSクライアントは、Oracleがファイル・サーバー上のファイルにアクセスする際のパフォーマンスを向上するために、ファイル・サーバーで実装できるAPIです。

構文

構文は次のとおりです。

```
DNFS_READBUFFERS = n
```

使用上のノート

デフォルトより大きい値を使用すると、Direct NFSクライアント・ファイル・サーバーからの一貫性のないI/Oに対処できますが、大きな値を使用すると、メモリーの使用量が増加する可能性もあります。

DNFS_ENABLEパラメータを同時に指定せずにこのパラメータを使用するには、入力ファイルが1GBを超えている必要があります。

親トピック: SQL*Loaderエクスプレス・モードのパラメータ・リファレンス

13.3.10 ENCLOSED_BY

SQL*LoaderエクスプレスモードのENCLOSED_BYパラメータでは、フィールドの囲み文字列を指定します。

デフォルト

デフォルトは、囲み文字なしです。

用途

ENCLOSED_BYパラメータでは、フィールドの囲み文字列を指定します。

構文

```
ENCLOSED_BY=['string'|x'hex-string']
```

囲み文字は1つの文字列または16進文字列である必要があります。

使用上のノート

同じ文字列を使用して囲み文字の最初と最後の両方を示す必要があります。

例

次の例では、フィールド・データは「/」文字(フォワード・スラッシュ)で囲まれます。

```
> sqlldr hr TABLE=employees ENCLOSED_BY='/'
```

親トピック: SQL*Loaderエクスプレス・モードのパラメータ・リファレンス

13.3.11 EXTERNAL_TABLE

SQL*LoaderエクスプレスモードのEXTERNAL_TABLEパラメータは、外部表のオプションを使用してデータをロードするかどうかを決定します。

デフォルト

EXECUTE

用途

EXTERNAL_TABLEパラメータは、外部表オプションを使用してデータをロードするかどうかをSQL*Loaderに指定します。

構文

```
EXTERNAL_TABLE=[NOT_USED | GENERATE_ONLY | EXECUTE]
```

次の3つの値を指定できます。

- NOT_USED—従来型パス・モードまたはダイレクト・パス・モードのいずれかを使用して、ロードを行います。
- GENERATE_ONLY—SQL*Loaderログ・ファイル内の外部表を使用してロードを行うために必要なすべてのSQL文を書き込みます。これらのSQL文は、編集およびカスタマイズできます。実際のロードは、SQL*Loaderを使用せずに、SQL*Plusでこれらの文を実行して、後で行うことができます。

- EXECUTE—SQL*Loaderエクスプレス・モードのデフォルト値。外部表を使用してロードを行うために必要なSQL文を実行します。ただし、SQL文からエラーが返されると、ロードは停止します。SQL文は、実行されたとおりログ・ファイルに書き込まれます。つまり、SQL文からエラーが返されると、ロードに必要な残りのSQL文はログ・ファイルには書き込まれません。

使用上のノート

外部表オプションで、データベース内のディレクトリ・オブジェクトを使用して、すべてのデータ・ファイルがどこに格納されるか、および出力ファイル(不良ファイルや廃棄ファイルなど)がどこで作成されるかを指定します。データ・ファイルを含むディレクトリ・オブジェクトには、READアクセスが、出力ファイルが作成されるディレクトリ・オブジェクトには、WRITEアクセスが必要です。データ・ファイルまたは出力ファイル格納用の既存のディレクトリ・オブジェクトがない場合は、SQL*LoaderでSQL文を生成して作成します。したがって、EXECUTEオプションを指定する場合は、CREATE ANY DIRECTORY権限が必要です。ロード処理の最後にディレクトリ・オブジェクトを削除する場合は、DROP ANY DIRECTORY権限も必要です。

ノート:

SQL*Loader で EXTERNAL_TABLE=EXECUTE 修飾子を指定すると、データのロードに使用可能な外部表が作成された後、INSERT 文が実行され、データがロードされます。外部表のすべてのファイルがディレクトリ・オブジェクト内に存在すると識別される必要があります。SQL*Loader は、ユーザーがアクセス権限を所有している既存のディレクトリ・オブジェクトを使用します。ただし、SQL*Loader により一致するディレクトリ・オブジェクトが検出されない場合は、一時ディレクトリ・オブジェクトの作成が試行されます。新しいディレクトリ・オブジェクトを作成する権限を所有していない場合、この操作は正常に実行されません。

この問題を解決するには、EXTERNAL_TABLE=GENERATE_ONLY を使用して、SQL*Loader で実行が試行される SQL 文を作成します。これらの SQL 文を抽出し、参照するディレクトリ・オブジェクトを、アクセス権限を所有しているディレクトリ・オブジェクトに変更します。その後、これらの SQL 文を実行します。

例

```
sqlldr hr TABLE=employees EXTERNAL_TABLE=NOT_USED
```

親トピック: SQL*Loaderエクスプレス・モードのパラメータ・リファレンス

13.3.12 FIELD_NAMES

SQL*LoaderエクスプレスのFIELD_NAMESパラメータでは、データベース表の列の順序に従っているフィールドを上書きします。

デフォルト

NONE

用途

FIELD_NAMESパラメータを使用して、データベース表の列の順序に従っているフィールドを上書きします。(デフォルトで、SQL*Loaderでは、ALL_TAB_COLUMNSビューにある表の列定義を使用して入力フィールドの順序とデータ型を判別します。)

このパラメータが役に立つ場合の例は、入力ファイルのデータが表の列と同じ順序になっていない場合です。このような場合、データ・ファイルに(表の列ヘッダー行のように)フィールド名のレコードを含め、FIELD_NAMESパラメータを使用して、最初のレ

コードのフィールド名を処理してフィールドの順序を決定するようにSQL*Loaderに指示できます。

構文

```
FIELD_NAMES=[ALL | ALL_IGNORE | FIRST | FIRST_IGNORE | NONE]
```

このパラメータの有効なオプションは次のとおりです。

- ALL—すべてのデータ・ファイルでフィールド名レコードが処理されます。
- ALL_IGNORE—すべてのデータ・ファイルで最初の(フィールド名)レコードを無視し、通常どおりデータ・レコードを処理します。
- FIRST—最初のデータ・ファイルで最初の(フィールド名)レコードを処理します。他のすべてのデータ・ファイルでは、フィールド名レコードがないため、データ・ファイルは通常どおり処理されます。
- FIRST_IGNORE—最初のデータ・ファイルで最初の(フィールド名)レコードを無視し、フィールド順序として表の列順序を使用します。
- NONE—どのデータ・ファイルにもフィールド名レコードが存在しないため、データ・ファイルは通常どおり処理されます。これはデフォルトです。

使用上のノート

- フィールド名に大文字と小文字が混在する場合、または空白などの特殊文字が含まれる場合、OPTIONALLY_ENCLOSED_BYパラメータまたはENCLOSED_BYパラメータを使用して、大/小文字を保持することと、特殊文字をフィールド名の一部として含めることを指定します。

例

列ヘッダーを含むCSVファイルを表にロードする際に、入力ファイルの各行のフィールドが表の列と同じ順序である場合、次のようにします。

```
> sqlldr hr TABLE=employees CSV=WITHOUT_EMBEDDED FIELD_NAMES=FIRST_IGNORE
```

親トピック: SQL*Loaderエクスプレス・モードのパラメータ・リファレンス

13.3.13 LOAD

SQL*LoaderエクスプレスモードのLOADパラメータでは、ロードするレコードの数を指定します。

デフォルト

すべてのレコードがロードされます。

用途

LOADパラメータでは、ロードするレコード数を指定します。

構文

```
LOAD=n
```

使用上のノート

ロードに指定したすべてのパラメータが正しく設定されているかどうかをテストする場合、LOADパラメータを使用して、すべてのレコードをロードするかわりに、かぎられた数のレコードを指定します。実際のレコード件数が指定された最大数より少ない場合、エラーは発生しません。

例

次の例では、最大10個のレコードをロードするように指定します。

```
> sqlldr hr TABLE=employees LOAD=10
```

外部表の方法によるロード(エクスプレス・モードのデフォルトのロード方法)では、ロードに成功したレコードのみが合計に加算されます。そのため、ファイルに15個のレコードがあり、レコード2および4が不良の場合、レコード1、3、5、6、7、8、9、10、11および12が合計10個のレコードとして表にロードされます。

従来型パス・ロードとダイレクト・パス・ロードでは、成功と失敗の両方のロード試行が合計に加算されます。そのため、ファイルに15個のレコードがあり、レコード2および4が不良の場合、レコード1、3、5、6、7、8、9および10の8個のみが実際に表にロードされます。

親トピック: SQL*Loaderエクスプレス・モードのパラメータ・リファレンス

13.3.14 NULLIF

SQL*LoaderエクスプレスモードのNULLIFパラメータは、フィールドをNULL列としてロードするかどうかを決定するために使用する値を指定します。

デフォルト

デフォルトでは、NULLIFのチェックは実行されません。

構文

```
NULLIF = "string"
```

または

```
NULLIF != "string"
```

使用上のノート

SQL*Loaderは、レコードのフィールドの値に基づいて、指定された値を確認します。等号(=)または不等号(!=)の指定を使用して一致した場合、その行ではフィールドがNULLに設定されます。空白の切捨て後に長さが0になったフィールドも、すべてNULLに設定されます。

例

次の例では、ピリオドの値が含まれるフィールドが存在する場合、各行でそれらのフィールドをNULLに設定します。

```
> sqlldr hr TABLE=employees NULLIF="."
```

親トピック: SQL*Loaderエクスプレス・モードのパラメータ・リファレンス

13.3.15 OPTIONALLY_ENCLOSED_BY

SQL*LoaderエクスプレスモードのOPTIONALLY_ENCLOSED_BYでは、オプションのフィールドの囲み文字列を指定します。

デフォルト

デフォルトでは、オプションのフィールド囲み文字はありません。

用途

OPTIONALLY_ENCLOSED_BYパラメータでは、オプションのフィールド囲み文字列を指定します。

構文

```
OPTIONALLY_ENCLOSED_BY=['string'| x'hex-string']
```

囲み文字は1つの文字列または16進文字列です。

使用上のノート

囲みの開始と終了の両方を示すために同じ文字列を使用する必要があります。

例

次の例では、オプションの囲み文字として二重引用符(")を指定します。

```
> sqlldr hr TABLE=employees OPTIONALLY_ENCLOSED_BY='"'
```

次の例では、16進形式でオプションの囲み文字を指定します。

```
> sqlldr hr TABLE=employees OPTIONALLY_ENCLOSED_BY=x'22'
```

親トピック: SQL*Loaderエクスプレス・モードのパラメータ・リファレンス

13.3.16 PARFILE

SQL*LoaderエクスプレスのPARFILEパラメータは、頻繁に使用するコマンドライン・パラメータを格納したファイルの名前を指定します。

デフォルト

デフォルトはありません

構文

```
PARFILE=parameter_file_name
```

使用上のノート

パラメータ値に引用符が含まれているものがある場合は、パラメータ・ファイルの使用をお薦めします。

ノート:

通常は問題ありませんが、システムによっては、パラメータ指定の中で等号(=)の前後に空白を挿入できないものもあります。

制限事項

- セキュリティ上の理由から、パラメータ・ファイルにUSERIDのパスワードを含めないことをお薦めします。コマンドラインでパラメータ・ファイルを指定すると、SQL*Loaderからパスワードの入力を求められます。たとえば:

  ```
  > sqlldr hr TABLE=employees PARFILE=daily_report.par
  Password:
  ```

例

次のパラメータ・ファイルtest.parを作成したとします。

```
table=employees
data='mydata*.dat'
enclosed_by='"'
```

次のコマンドを実行すると、mydata*.datと一致するデータ・ファイル内の二重引用符で囲まれたすべてのフィールドがemployees表にロードされます。

```
> sqlldr hr PARFILE=test.par
Password:
```

親トピック: SQL*Loaderエクスプレス・モードのパラメータ・リファレンス

13.3.17 SILENT

SQL*LoaderエクスプレスモードのSILENTパラメータは、SQL*Loaderの操作中に画面に書き込まれるコンテンツを抑止します。

デフォルト

¥¥このパラメータが指定されていない場合、内容は抑止されません。

用途

SILENTパラメータは、SQL*Loaderの操作中に画面に書き込まれる内容の一部を抑止します。

構文

構文は次のとおりです。

SILENT={HEADER | FEEDBACK | ERRORS | DISCARDS | PARTITIONS | ALL}

適切な値を使用して、次の1つ以上の内容を抑止します(複数のオプションを指定する場合、カンマで区切る必要があります)。

- HEADER—画面に通常表示されるSQL*Loaderのヘッダー・メッセージを非表示にします。ただし、ヘッダー・メッセージはログ・ファイルに出力されます。
- FEEDBACK—ロードで画面に通常表示される「commit point reached」メッセージとステータス・メッセージを非表示にします。
- ERRORS—レコードにOracleエラーが発生したためそのレコードが不良ファイルに書き込まれた場合、データ・エラー・メッセージがログ・ファイルに出力されないようにします。ただし、拒否レコード件数は出力されます。
- DISCARDS—レコードが廃棄ファイルに書き込まれた場合に、そのことを示すメッセージがログ・ファイルに出力されないようにします。このオプションは、エクスプレス・モードでは無視されます。
- PARTITIONS—パーティション表のダイレクト・ロード中、ログ・ファイルに対するパーティションごとの統計情報の書込みを使用禁止にします。このオプションは、強制ダイレクト・パス操作でのみ有効です。
- ALL—すべての抑止オプションを実装します。

例

たとえば、画面に通常表示されるヘッダーとフィードバック・メッセージが表示されないようにするには、コマンドラインの引数で次のように指定します。

```
> sqlldr hr TABLE=employees SILENT=HEADER, FEEDBACK
```

親トピック: SQL*Loaderエクスプレス・モードのパラメータ・リファレンス

13.3.18 TABLE

SQL*LoaderエクスプレスモードのTABLEパラメータでは、SQL*Loaderエクスプレスモードをアクティブ化します。

デフォルト

デフォルトはありません。

構文

```
TABLE=[schema-name.]table-name
```

使用上のノート

スキーマ名または表名に小文字、空白または他の特殊文字が含まれる場合、名前を二重引用符で囲み、その文字列全体を一重引用符で囲む必要があります。たとえば:

```
TABLE='"hr.Employees"'
```

制限事項

TABLEパラメータは、SQL*Loaderエクスプレス・モードでのみ有効です。

例

次の例では、エクスプレス・モードでemployees表をロードします。

```
> sqlldr hr TABLE=employees
```

親トピック: SQL*Loaderエクスプレス・モードのパラメータ・リファレンス

13.3.19 TERMINATED_BY

SQL*LoaderエクスプレスのTERMINATED_BYでは、デフォルトに優先するフィールド終端文字を指定します。

デフォルト

デフォルトでは、カンマがフィールド終端文字です。

用途

TERMINATED_BYパラメータでは、デフォルトに優先するフィールド終端文字を指定します。

構文

```
TERMINATED_BY=['string'| x'hex-string' | WHITESPACE]
```

フィールド終端文字は1つの文字列または16進文字列である必要があります。

使用上のノート

TERMINATED_BY=WHITESPACEを指定すると、最初に空白文字(スペース、タブ、空白、LF、改ページまたは改行)が現れるまでデータが読み込まれます。空白文字が現れると、隣接する空白文字が検出されなくなるまで現在の位置が進められます。この方法では、様々な数の空白によってフィールド値を区切ることができます。

TERMINATED_BY=WHITESPACEを指定する場合、NULLフィールドに空白だけを含めることはできません。空白がスキップされ、エラーが報告される可能性があるためです。このオプションでは、データにNULLフィールドが含まれる場合は別の文字列を使用してNULLフィールドを示すことを検討します。また、NULLIFパラメータを使用してNULLIF文字列を示します。たとえば、文字列"NoData"を使用してNULLフィールドを示し、データに文字列"NoData"を挿入してNULLフィールドを示すことができます。NULLIF="NoData"を指定して、SQL*Loaderに対し、文字列"NoData"を含むフィールドをNULLに設定するように指示します。

例

次の例では、フィールドが「|」文字で終了します。

```
> sqlldr hr TABLE=employees TERMINATED_BY="|"
```

親トピック: SQL*Loaderエクスプレス・モードのパラメータ・リファレンス

13.3.20 TIMESTAMP_FORMAT

TIMESTAMP_FORMATパラメータは、ロードの際に使用する必要があるタイムスタンプの書式を指定します。

デフォルト

デフォルトは、環境変数NLS_TIMESTAMP_FORMATの値から取得されます。NLS_TIMESTAMP_FORMATが設定されていない場合、タイムスタンプでは、6桁の小数点精度を持つ環境変数NLS_TERRITORYで定義されたデフォルト書式を使用します。

構文

```
TIMESTAMP_FORMAT="timestamp_format"
```

例

次の例では、タイムスタンプ書式を指定します。

```
> sqlldr hr TABLE=employees TIMESTAMP_FORMAT="MON-DD-YYYY HH:MI:SSXFF AM"
```

親トピック: SQL*Loaderエクスプレス・モードのパラメータ・リファレンス

13.3.21 TRIM

SQL*LoaderエクスプレスのTRIMパラメータでは、ロード中に使用するフィールド切捨てのタイプを指定します。

デフォルト

従来型パス・ロードとダイレクト・パス・ロードのデフォルトはLDRTRIMです。外部表ロードのデフォルトはLRTRIMです。

用途

TRIMパラメータは、ロード中に使用するフィールド切捨てのタイプを指定します。TRIMは、テキスト・フィールドの先頭から空白を切り捨てるか、末尾から切り捨てるか、その両方から切り捨てるかを指定するために使用します。空白には、空白文字およびその他の印字されない文字(タブ、行送り、改行など)が含まれます。

構文

```
TRIM=[LRTRIM | NOTRIM | LTRIM | RTRIM |LDRTRIM]
```

オプション:

- LRTRIMでは、先頭と末尾のどちらの空白も切り捨てることを指定します。
- NOTRIMは、フィールドから文字を切り捨てないように指定します。この設定では、通常、最高のパフォーマンスを得ることができます。
- LTRIMでは、先頭の空白の切捨てを指定します。
- RTRIMでは、末尾の空白の切捨てを指定します。
- LDRTRIMはNOTRIMと同じです。ただし、フィールドがOPTIONALLY_ENCLOSED_BYで指定したデリミタ付きのフィールドで、オプションの囲みデリミタが特定のインスタンスで欠落している場合を除きます。その場合は、空白が左側から切り捨てられます。

使用上のノート

すべてが空白のフィールドに切捨てを指定すると、そのフィールドはNULLに設定されます。

制限事項

- 強制従来型パス・ロードおよび強制ダイレクト・パス・ロードでは、LDRTRIMのみがサポートされます。どの値でTRIMパラメータを指定しても、常にこの内容を示すメッセージが表示されます。
- ロードがデフォルトの外部表ロードで、SQL*Loaderエクスプレス・モードでかわりにダイレクト・パス・ロードを使用することになるエラーが発生した場合、別の方法を指定したり、LRTRIMで外部表のデフォルトを受け入れていても、LDRTRMが切捨て方法として使用されます。この変更を示すアラート・メッセージが表示されます。

  NOTRIMを使用する場合、PRESERVE BLANKS句を含む制御ファイルを使用します。

例

次の例では、右側のすべての空白(後続の空白)を切り捨ててフィールドを読み取ります。

```
> sqlldr hr TABLE=employees TRIM=RTRIM
```

親トピック: SQL*Loaderエクスプレス・モードのパラメータ・リファレンス

13.3.22 USERID

SQL*LoaderエクスプレスのUSERIDを使用すると、Oracleユーザー名とパスワードを指定できるようになり、その入力要求が表示されなくなります。

デフォルト

ありません。

用途

USERIDパラメータを使用すると、Oracleユーザー名とパスワードを指定できます。

構文

```
USERID = [username | / | SYS]
```

使用上のノート

USERIDパラメータを指定しない場合、その入力を求められます。スラッシュのみを入力すると、デフォルトとしてオペレーティング・システムのログイン名がUSERIDに適用されます。

ユーザーSYSとして接続する場合は、接続文字列にAS SYSDBAも指定する必要があります。

制限事項

- 文字列AS SYSDBAには空白が含まれるため、一部のオペレーティング・システムでは、接続文字列全体をなんらかの方法でリテラルとしてマーク(一重引用符で囲むなど)する必要があります。また、一部のオペレーティング・システムでは、バックスラッシュなどのエスケープ文字をコマンドラインの引用符の前に使用する必要があります。

  システムの特殊文字および予約文字の詳細は、対象オペレーティング・システムのマニュアルを参照してください。

例

次の例では、ユーザーhrのジョブを開始します。

```
> sqlldr USERID=hr TABLE=employees
```

```
Password:
```

親トピック: SQL*Loaderエクスプレス・モードのパラメータ・リファレンス

13.4 SQL*Loaderエクスプレス・モードの構文図

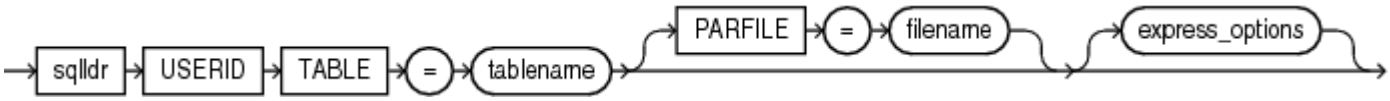
SQL*Loaderエクスプレス・モードのオプションについて理解するために、ここに示した図形式の構文ガイド(線路図またはDDL構文図と呼ぶこともあります)を参照してください。

図形式の構文表記法の理解

使用される構文の表記法については、次を参照してください。

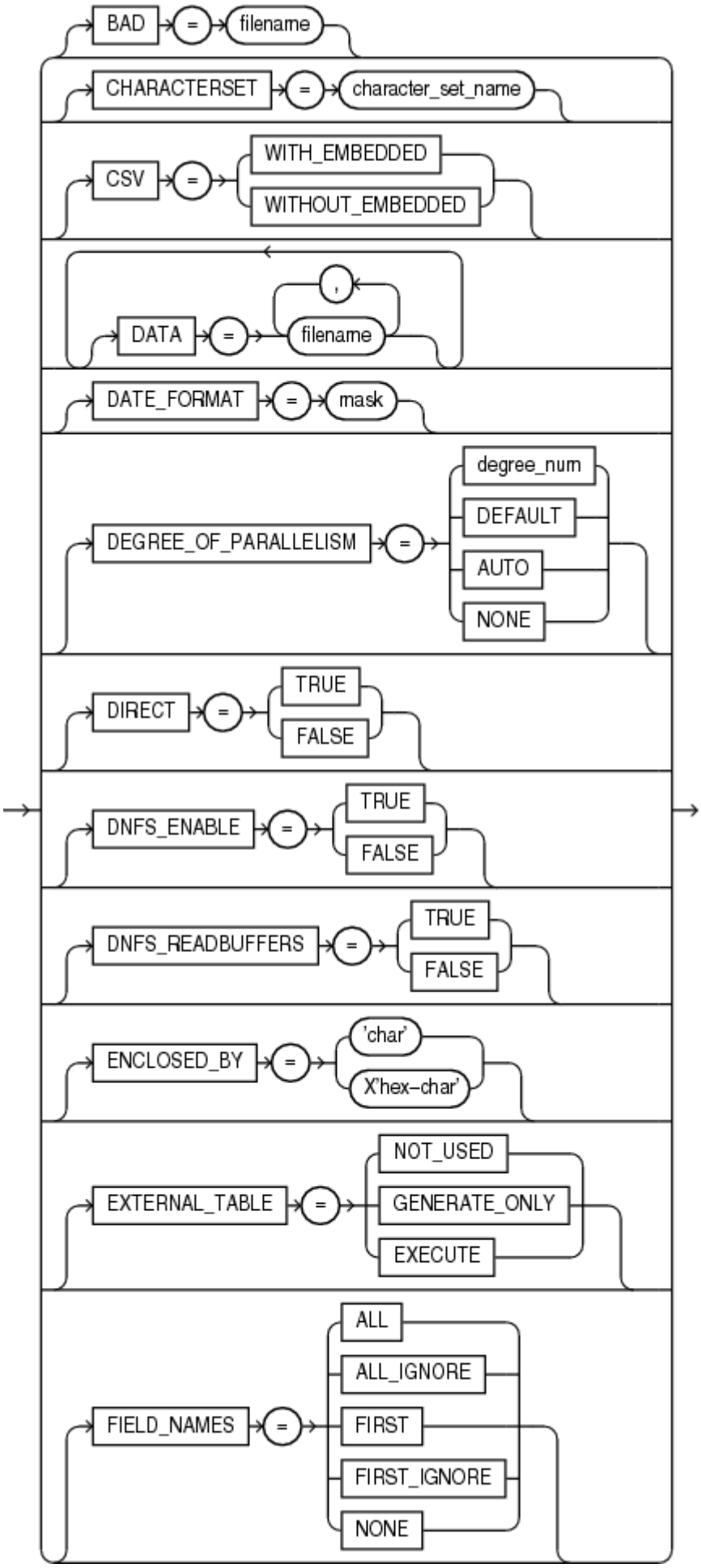
構文図の読み方

express_init

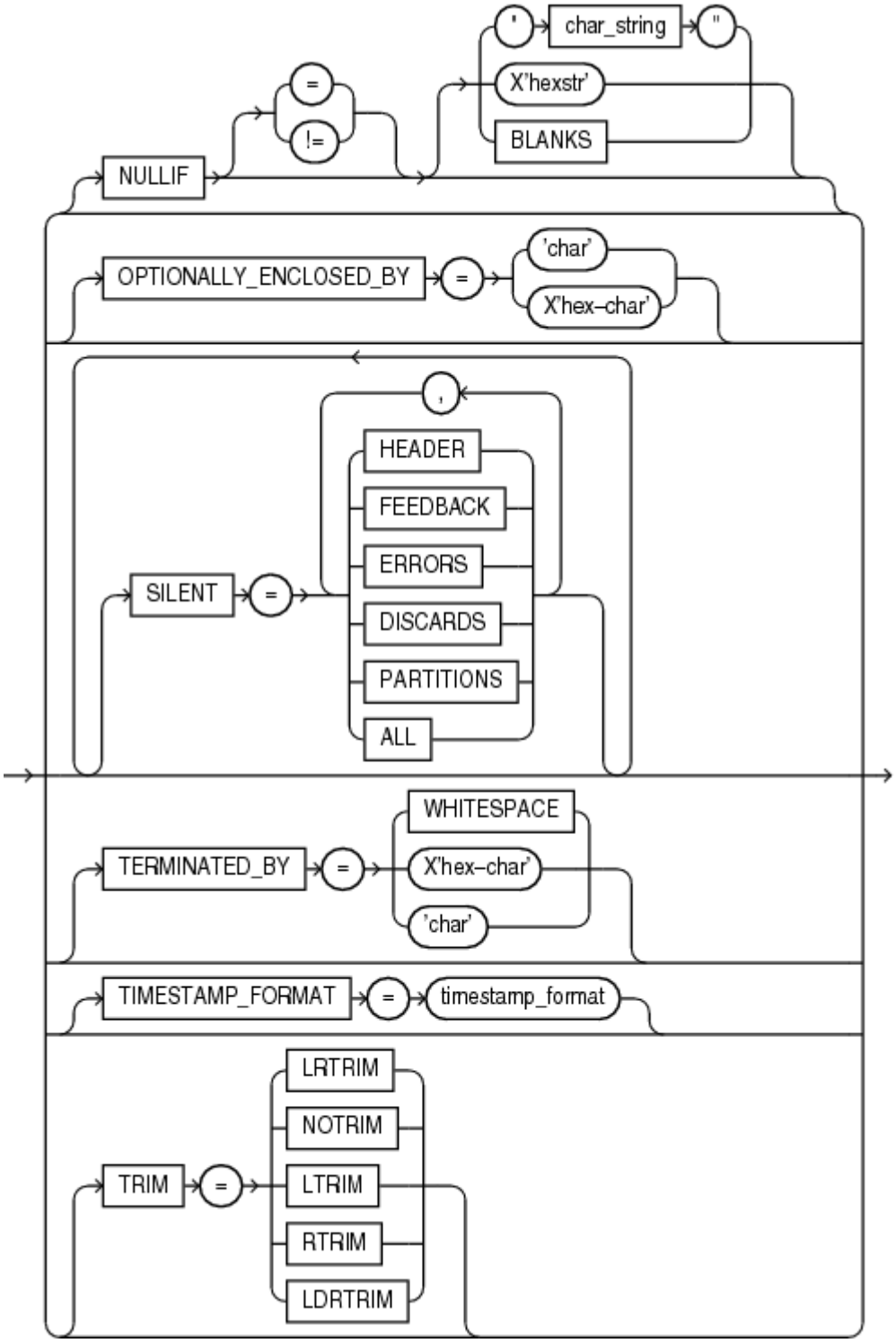


次の構文図は、前の構文図の`express_options`に含まれるパラメータを示します。次の構文図に示されているSQL*Loaderエクスプレス・モードのパラメータは、すべてオプションで、SQL*Loaderコマンドラインに任意の順序で指定できます。そのため、これらは単純なアルファベット順で示します。

express_options



express_options_cont



親トピック: SQL*Loaderエクスプレス

第III部 外部表

外部表を正常に使用するには、外部表の概念を調べ、Oracle Databaseで外部表を使用するために使用できるオプションの例を参照してください。

- 外部表の概要
  外部表機能は、既存のSQL*Loader機能を補足する機能です。この機能により、データベース内の表にあるデータと同様に、外部ソースのデータにアクセスできるようになります。
- ORACLE_LOADERアクセス・ドライバ
  ORACLE_LOADERアクセス・ドライバのパラメータを使用して外部表にアクセスする方法を制御し、アクセス・ドライバのデフォルト動作を変更する方法について説明します。
- ORACLE_DATAPUMPアクセス・ドライバ
  ORACLE_DATAPUMPアクセス・ドライバは、ORACLE_DATAPUMP型の外部表に固有のアクセス・パラメータのセットを提供します。
- ORACLE_HDFSおよびORACLE_HIVEアクセス・ドライバ
  外部表を使用すると、Oracle Databaseの表に格納されたデータであるかのように、HadoopおよびHiveクライアント上のHDFSおよびHive表に格納されているデータにアクセスできます。
- 外部表の例
  Oracle DatabaseおよびBig Dataのデータの問合せにアクセス・ドライバのORACLE_LOADER、ORACLE_DATAPUMP、ORACLE_HDFSおよびORACLE_HIVEを使用する方法の例を示します。
- ORACLE_BIGDATAアクセス・ドライバ
  ORACLE_BIGDATAアクセス・ドライバを使用すると、Oracle Databaseの表に格納されたデータであるかのように、オブジェクト・ストアに格納されているデータにアクセスできます。

14 外部表の概要

外部表機能は、既存のSQL*Loader機能を補足する機能です。この機能により、データベース内の表にあるデータと同様に、外部ソースのデータにアクセスできるようになります。

- 外部表の作成方法
  外部表は、SQLのCREATE TABLE...ORGANIZATION EXTERNAL文を使用して作成されます。
- データ・ファイルおよび出力ファイルの位置
  データ・ファイルおよび出力ファイルはサーバーに配置する必要があります。ファイルの読取りおよび書込みを行う位置を指定するディレクトリ・オブジェクトが必要です。
- 外部表のアクセス・パラメータ
  外部表のアクセス・ドライバのデフォルトの動作を変更するには、アクセス・パラメータを指定します。
- 外部表使用中のデータ型変換
  ソースとターゲットのデータ型が一致しない場合、Oracle Databaseが外部表から読み取るとき、および外部表に書き込むときに変換エラーが発生する可能性があります。

親トピック: 外部表

14.1 外部表の作成方法

外部表は、SQLのCREATE TABLE...ORGANIZATION EXTERNAL文を使用して作成されます。

データ・ロード時にステージング表の追加の索引付けが必要な場合、SQL*Loaderを使用する方が有効です。SQL*Loaderと外部表との処理内容の違いについては、「SQL*Loaderと外部表との処理内容の違い」を参照してください。

Oracle Database 12cリリース2 (12.2.0.1)からは、外部表に含まれるデータをパーティション化でき、これにより、データベースに格納された表をパーティション化した場合(パーティション・プルーニングなど)と同様にパフォーマンスを高めることができます。

関連項目:

外部表の作成と管理の詳細、および外部表のパーティション化の詳細は、『Oracle Database管理者ガイド』を参照してください。

> ノート:
>
> 外部表は SQL 文のインライン外部表として使用できるため、データ・ディクショナリ内に永続データベース・オブジェクトとして外部表を作成する必要がなくります。詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

外部表の作成時に、次の属性を指定します。

- TYPE—外部表の型を指定します。外部表の各型は、固有のアクセス・ドライバによってサポートされます。
  - ORACLE_LOADER—これがデフォルトのアクセス・ドライバです。これによって、外部表から内部表にデータがロードされます。データはテキスト・データ・ファイルからのものである必要があります。(ORACLE_LOADERアクセス・ドライバは、アンロードを実行できないため、内部表から外部表にデータを移動することはできません。)

- ORACLE_DATAPUMP—このアクセス・ドライバは、ロードとアンロードの両方を実行できます。データは、バイナリのダンプ・ファイルからロードする必要があります。内部表から外部表へのロードは、バイナリのダンプ・ファイルからフェッチすることによって実行されます。内部表から外部表へのアンロードは、外部表のバイナリのダンプ・ファイルを移入することによって実行されます。ORACLE_DATAPUMPアクセス・ドライバは、SQLのCREATE TABLE AS SELECT文を使用した外部表の作成の一環としてのみダンプ・ファイルにデータを書き込むことができます。作成されたダンプ・ファイルは、何回でも読み取ることができますが、変更はできません(つまり、DML操作は実行できません)。
- ORACLE_HDFS—Hadoop Distributed File System (HDFS)に格納されたデータを抽出します。
- ORACLE_HIVE—Apache HIVEに格納されたデータを抽出します。

- DEFAULT DIRECTORY—明示的にディレクトリ・オブジェクトに名前を付けないすべての入出力ファイルに対して、使用するデフォルトのディレクトリを指定します。位置はディレクトリ・パスではなく、ディレクトリ・オブジェクトで指定されます。外部表を作成する前にディレクトリ・オブジェクトを作成しておく必要があります(これを行わない場合はエラーが生成されます)。詳細は、「データ・ファイルおよび出力ファイルの位置」を参照してください。
- ACCESS PARAMETERS—外部データ・ソースを記述し、指定された外部表の型を実装します。各外部表の型には、その外部表の型に固有のアクセス・パラメータを提供する専用のアクセス・ドライバがあります。アクセス・パラメータはオプションです。詳細は、「アクセス・パラメータ」を参照してください。
- LOCATION—外部表のデータ・ファイルを指定します。
  - ORACLE_LOADERおよびORACLE_DATAPUMPの場合、ファイルの名前はdirectory:fileという形式になります。directoryの部分はオプションです。この部分を指定しないと、デフォルトのディレクトリがファイルのディレクトリとして使用されます。ORACLE_LOADERアクセス・ドライバを使用している場合には、ファイル名にはワイルドカードを使用でき、アスタリスク(*)は複数文字を、疑問符(?)は1文字を表します。
  - ORACLE_HDFSの場合、LOCATION句は、ディレクトリまたはファイルのUniform Resource Identifier (URI)のリストです。URIに関連付けられたディレクトリ・オブジェクトはありません。
  - ORACLE_HIVEの場合、LOCATION句は使用されません。かわりに、Hadoop HCatalog表を読み取ってデータ・ソースの場所(ファイルや別のデータベース)に関する情報を取得します。

次の各例に、各アクセス・ドライバの属性の簡単な使用方法を示します。

例14-1 ORACLE_LOADERアクセス・ドライバの属性の指定

次の例では、ORACLE_LOADERアクセス・ドライバを使用して、これらの各属性の使用方法を示しています(デフォルト・ディレクトリdef_dir1はすでに存在するものとします)。

```
SQL> CREATE TABLE emp_load
  2    (employee_number      CHAR(5),
  3     employee_dob         CHAR(20),
  4     employee_last_name   CHAR(20),
  5     employee_first_name  CHAR(15),
  6     employee_middle_name CHAR(15),
  7     employee_hire_date   DATE)
  8  ORGANIZATION EXTERNAL
  9    (TYPE ORACLE_LOADER
 10     DEFAULT DIRECTORY def_dir1
 11     ACCESS PARAMETERS
 12       (RECORDS DELIMITED BY NEWLINE
 13        FIELDS (employee_number      CHAR(2),
 14                employee_dob         CHAR(20),
 15                employee_last_name   CHAR(18),
 16                employee_first_name  CHAR(11),
```

```
 17                 employee_middle_name CHAR(11),
 18                 employee_hire_date   CHAR(10) date_format DATE mask "mm/dd/yyyy"
 19                )
 20        )
 21      LOCATION ('info.dat')
 22     );

Table created.
```

アクセス・ドライバで指定する情報により、データ・ソースのデータが、外部表の定義と一致するように処理されます。CREATE TABLE emp_loadの後にリストされるフィールドが、実際にinfo.datソース・ファイル内のデータのメタデータを定義します。

例14-2 ORACLE_DATAPUMPアクセス・ドライバの属性の指定

この例では、inventories_xtという名前の外部表を作成し、外部表のダンプ・ファイルにoeサンプル・スキーマ内のinventories表のデータを移入します。

```
SQL> CREATE TABLE inventories_xt
2 ORGANIZATION EXTERNAL
3 (
4 TYPE ORACLE_DATAPUMP
5 DEFAULT DIRECTORY def_dir1
6 LOCATION ('inv_xt.dmp')
7 )
8 AS SELECT * FROM inventories;
Table created.
```

例14-3 ORACLE_HDFSアクセス・ドライバの属性の指定

```
CREATE TABLE sales_external
(    time_id            DATE NOT NULL, …
     amount_sold        NUMBER(10,2)
)
ORGANIZATION EXTERNAL
(   TYPE ORACLE_HDFS
    ACCESS PARAMETERS (com.oracle.bigdata.cluster=hadoop1)
    LOCATION (“hdfs:/usr/sales_1.csv”, “hdfs:/usr/my_sales_*.csv”)
)
```

例14-4 ORACLE_HIVEアクセス・ドライバの属性の指定

```
CREATE TABLE sales_external
(    time_id            DATE NOT NULL, …
     amount_sold        NUMBER(10,2)
)
ORGANIZATION EXTERNAL
(    TYPE ORACLE_HIVE
     ACCESS PARAMETERS (com.oracle.bigdata.cluster=hadoop1
                        com.oracle.bigdata.tablename=default.ratings_hive_table)
);
```

親トピック: 外部表の概要

14.2 データ・ファイルおよび出力ファイルの位置

データ・ファイルおよび出力ファイルはサーバーに配置する必要があります。ファイルの読取りおよび書込みを行う位置を指定するディレクトリ・オブジェクトが必要です。

ノート:

この項に示すディレクトリ・オブジェクト関連の情報は、ORACLE_HDFS アクセス・ドライバまたは ORACLE_HIVE アクセス・ドライバには適用されません。ORACLE_HDFS ドライバを使用する場合、データの場所はディレクトリまたはファイルの URI のリストで指定します。URI に関連付けられたディレクトリ・オブジェクトはありません。ORACLE_HIVE ドライバではデータ・ソースの場所を指定しません。その情報は Hive メタストア表を読み取って取得するため、ディレクトリ・オブジェクトは必要ありません。

アクセス・ドライバは、データベース・サーバー内で実行されます。この動作は、SQL*Loaderが、ロードするデータをサーバーに送信するクライアント・プログラムであるという点で、SQL*Loaderとは異なります。この違いは、次のことを意味しています。

- アクセス・ドライバがロード可能なファイルにサーバーがアクセスできる必要があります。
- サーバーでは、アクセス・ドライバによって作成された出力ファイル(ログ・ファイル、不良ファイルおよび廃棄ファイル)、およびORACLE_DATAPUMPアクセス・ドライバによって作成されたダンプ・ファイルの作成と書込みを行う必要があります。

ファイルの読取りおよび書込みを実行する場所を指定するには、アクセス・ドライバでディレクトリ・オブジェクトを使用する必要があります。ディレクトリ・オブジェクトは、ファイル・システムのディレクトリ名に名前をマップします。たとえば、次の文は/usr/apps/datafilesにあるディレクトリにマップされる、ext_tab_dirという名前のディレクトリ・オブジェクトを作成します。

```
CREATE DIRECTORY ext_tab_dir AS '/usr/apps/datafiles';
```

DBAまたはユーザーがCREATE ANY DIRECTORY権限でディレクトリ・オブジェクトを作成できます。

ノート:

Oracle Real Applications Cluster(Oracle RAC)構成で外部表を使用するには、ディレクトリ・オブジェクトのパスがクラスタ・ファイル・システム上に存在するようにする必要があります。

ディレクトリの作成後、ディレクトリ・オブジェクトを作成するユーザーは、そのディレクトリのREAD権限およびWRITE権限を他のユーザーに付与する必要があります。これらの権限は、ロールを使用して割り当てるのではなく、明示的に付与する必要があります。たとえば、ext_tab_dirで指定されたディレクトリのユーザーscottのかわりに、サーバーがファイルを読み込むことができるようにするには、ディレクトリ・オブジェクトを作成したユーザーが、次のコマンドを実行する必要があります。

```
GRANT READ ON DIRECTORY ext_tab_dir TO scott;
```

SYSユーザーのみがディレクトリ・オブジェクトを所有できます。ただし、SYSユーザーは、ディレクトリ・オブジェクトを作成する権限を他のユーザーに付与できます。ディレクトリ・オブジェクトに対するREAD権限またはWRITE権限は、Oracle Databaseがそのファイルに読取りまたは書込みを実行できるという意味にすぎません。適切なオペレーティング・システム権限がないかぎり、Oracle Databaseの外部にあるファイルには直接アクセスできません。同様に、Oracle Databaseには、ディレクトリのファイルに対して読取りおよび書込みを行うオペレーティング・システム権限が必要です。

親トピック: 外部表の概要

14.3 外部表のアクセス・パラメータ

外部表のアクセス・ドライバのデフォルトの動作を変更するには、アクセス・パラメータを指定します。

特定の型の外部表を作成する場合、アクセス・パラメータを指定して、アクセス・ドライバのデフォルトの動作を変更できます。各アクセス・ドライバには、アクセス・パラメータ用の固有の構文があります。外部表に使用するためのアクセス・ドライバORACLE_LOADER、ORACLE_DATAPUMP、ORACLE_HDFSおよびORACLE_HIVEが提供されています。

ノート:

これらのアクセス・パラメータは、SQL 文の CREATE TABLE...ORGANIZATION EXTERNAL の opaque_format_spec としてまとめて参照されます。ACCESS パラメータ句を使用すると、SQL コメントを使用できます。

関連項目:

- ORACLE_LOADERアクセス・ドライバ
- ORACLE_DATAPUMPアクセス・ドライバ
- ORACLE_HDFSおよびORACLE_HIVEアクセス・ドライバ
- SQL CREATE TABLE文を使用するときにopaque_format_specを指定する方法の詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

親トピック: 外部表の概要

14.4 外部表の使用時のデータ型の変換

ソースとターゲットのデータ型が一致しない場合、Oracle Databaseが外部表から読み取るとき、および外部表に書き込むときに、変換エラーが発生する可能性があります。

外部表の読取り時の変換エラー

外部表から行を選択すると、アクセス・ドライバによって、データ・ソースのデータを外部表の対応する列のデータ型と一致させるために必要な変換が実行されます。データおよび必要な変更タイプによっては、変換でエラーが発生する可能性があります。

外部表からの読取り時に発生する可能性のあるデータ変換の問題のタイプを示すために、2つの列(データ型がVARCHAR2(4)のKEYと、データ型がNUMBERのVAL)を使用して次の外部表KV_TAB_XTを作成するとします。

```
SQL> CREATE TABLE KV_TAB_XT (KEY VARCHAR2(4), VAL NUMBER)
2 ORGANIZATION EXTERNAL
3 (DEFAULT DIRECTORY DEF_DIR1 LOCATION ('key_val.csv'));
```

外部表KV_TAB_XTは、アクセス・パラメータのデフォルト値を使用します。したがって、次のようになります。

- レコードは改行で区切られます。
- データ・ファイルとデータの文字セットは同じです。
- データ・ファイル内のフィールドの名前と順序は、外部表の列と同じです。
- フィールドのデータ型はCHAR(255)です。
- 各フィールドのデータはカンマで終了します。

KV_TAB_XT外部表のデータ・ファイル内のレコードの要件は、次のとおりです。

- 最大4バイトの文字列。文字列が空の場合、フィールドの値はNULLです。
- 終了のカンマ。
- 数字の文字列。文字列が空の場合、このフィールドの値はNULLです。
- オプションの終了カンマ。

アクセス・ドライバでは、データ・ファイルからレコードを読み取るとき、データ・ファイル内のKEYフィールドの値の長さが4以下であることを確認し、データ・ファイル内のVALフィールドの値をOracle Databaseの数値に変換しようとします。

KEYフィールドの値の長さが4より大きい場合、またはVALの値に数値以外の文字がある場合、ORACLE_LOADERアクセス・ドライバはその行を拒否します。その結果、行のコピーが不良ファイルに書き込まれ、エラー・メッセージがログ・ファイルに書き込まれます。

すべてのアクセス・ドライバは、外部表のソースにあるフィールドのデータ型からの変換、および外部表の列のデータ型からの変換を処理する必要があります。アクセス・ドライバが実行する変換およびチェックのタイプについて、いくつかの例を次に示します。

- 文字データを、ソース・データで使用されている文字セットからデータベースで使用される文字セットに変換します。
- 文字データから数値データに変換します。
- 数値データから文字データに変換します。
- 文字データから日付またはタイムスタンプに変換します。
- 日付またはタイムスタンプから文字データに変換します。
- 文字データから内部データ型に変換します。
- 内部データ型から文字データに変換します。
- キャラクタ列のデータ値の長さがその列の長さ制限を超えていないことを確認します。

必要な変換または検証の実行中にエラーが発生した場合、アクセス・ドライバではエラーの処理方法を決定できます。ORACLE_LOADERおよびORACLE_DATAPUMPアクセス・ドライバでは、エラーを検出すると、レコードを拒否し、エラー・メッセージをログ・ファイルに書き込みます。その場合、そのレコードがデータ・ソースになかったかのようになります。ORACLE_HDFSおよびORACLE_HIVEアクセス・ドライバがエラーを検出すると、エラーが見つかったフィールドの値がNULLに設定されます。この動作は、HiveがHadoopでエラーを処理する際の動作と一致しています。

外部表列のデータ型と一致させるために、データ・ソースからのデータがアクセス・ドライバによって変換された後も、その外部表にアクセスするSQL文で、追加のデータ型変換が必要になる場合があります。この追加変換のいずれかでエラーが発生した場合は、文全体が失敗します。(ただし、SQL文でDMLエラー・ロギング機能を使用してこれらのエラーを処理する場合は例外です。)これらの変換は、通常、SQL文の実行時に必要になる可能性がある変換と同じです。たとえば、KV_TAB_XT外部表の定義を文字データ型の列のみが含まれるように変更した後、INSERT文を実行して、外部表のデータを列VALのデータ型がNUMBERの別の表にロードするとします。

```
SQL> CREATE TABLE KV_TAB_XT (KEY VARCHAR2(20), VAL VARCHAR2(20))
2 ORGANIZATION EXTERNAL
3 (DEFAULT DIRECTORY DEF_DIR1 LOCATION (‘key_val.csv’));
4 CREATE TABLE KV_TAB (KEY VARCHAR2(4), VAL NUMBER);
5 INSERT INTO KV_TAB SELECT * FROM KV_TAB_XT;
```

この例において、VALのデータに数字以外の文字が含まれている場合、アクセス・ドライバはレコードを拒否しません。これは、外部表のVALが(NUMBERではなく) VARCHAR2になったためです。ただし、SQLの処理でKV_TAB_XTの文字データ型からKV_TABの数値データ型への変換処理が必要になります。外部表のVALの値に数字以外の文字がある場合、SQLで変換エラーが発生し、挿入された行はすべてロールバックされます。SQLでの変換エラーを回避するため、外部表の列のデータ型が、これらの列の値を使用する他の表または関数で要求されるデータ型と一致させることをお薦めします。

外部表への書込み時の変換エラー

ORACLE_DATAPUMPアクセス・ドライバを使用すると、CREATE TABLE AS SELECT文を使用してデータを外部表にアンロードできます。SELECT式における列のデータ型が外部表の列のデータ型と一致しない場合、データ変換が発生します。SQLでデータ型変換のエラーが検出されると、SQLは文を停止し、データ・ファイルは読み取ることができなくなります。

変換エラーによる操作の失敗に関連する問題を回避するには、外部表の列のデータ型を、外部表への書込みに使用される

ソース表または式の列のデータ型と一致させる必要があります。ただし、外部表はすべてのデータ型をサポートするわけではないため、必ず成功するとはかぎりません。そのような場合、ソース表でサポートされていないデータ型を、外部表でサポートするデータ型に変換する必要があります。次のCREATE TABLE文は、この変換の例を示しています。

```
CREATE TABLE LONG_TAB_XT (LONG_COL CLOB)
ORGANIZATION EXTERNAL...SELECT TO_LOB(LONG_COL) FROM LONG_TAB;
```

LONG_TABという名前のソース表には、LONG列があります。そのため、作成する外部表の対応する列LONG_TAB_XTはCLOBである必要があり、外部表の移入に使用されるSELECT副問合せは、TO_LOB演算子を使用して列をロードする必要があります。

ノート:

LONG データ型(LONG、LONG RAW、LONG VARCHAR、LONG VARRAW)のすべての形式は、Oracle8i リリース 8.1.6 で非推奨になりました。以降のリリースでは、LONG データ型は既存のアプリケーションとの下位互換性のために提供されていました。以降のリリースで開発された新しいアプリケーションでは、大量の文字データに CLOB および NCLOB データ型を使用することをお薦めします。

親トピック: 外部表の概要

15 ORACLE_LOADERアクセス・ドライバ

ORACLE_LOADERアクセス・ドライバのパラメータを使用して外部表にアクセスする方法を制御し、アクセス・ドライバのデフォルト動作を変更する方法を学習します。

- ORACLE_LOADERアクセス・ドライバについて
  ORACLE_LOADERアクセス・ドライバは、ORACLE_LOADER型の外部表に固有のアクセス・パラメータのセットを提供します。
- access_parameters句
  access_parameters句には、コメント、レコード形式およびフィールド形式の情報が含まれています。
- record_format_info句
  record_format_info句とその副次句を使用して、レコード情報を解析、ラベル付け、管理する方法について説明します。
- field_definitions句
  field_definitions句では、FIELDSパラメータを使用してデータ・ファイルのフィールドに名前を付け、レコード内でのそれらの検索方法を指定します。
- column_transforms句
  オプションのORACLE_LOADERアクセス・ドライバCOLUMN TRANSFORMS句を使用した変換では、データ・ファイル内の列に直接マップされない外部表内の列をロードする方法を指定できます。
- ORACLE_LOADERアクセス・ドライバのパラレル・ロードに関する考慮点
  ORACLE_LOADERアクセス・ドライバは、大きいデータ・ファイルを、個別に処理できるチャンクに分割しようとします。
- ORACLE_LOADERアクセス・ドライバを使用する際のパフォーマンスのヒント
  このトピックでは、ORACLE_LOADERアクセス・ドライバを使用する際のパフォーマンスのヒントについて説明します。
- ORACLE_LOADERアクセス・ドライバ使用時の制限事項
  この項では、ORACLE_LOADERアクセス・ドライバを使用する際に知っておく必要がある制限事項について説明します。
- ORACLE_LOADERアクセス・ドライバの予約語
  外部表アクセス・パラメータで、列名や表名などの識別子が指定された場合、特定の値はアクセス・パラメータ・パーサーにより予約語とみなされます。

親トピック: 外部表

15.1 ORACLE_LOADERアクセス・ドライバについて

ORACLE_LOADERアクセス・ドライバは、ORACLE_LOADER型の外部表に固有のアクセス・パラメータのセットを提供します。

アクセス・ドライバのデフォルトの動作を変更するためにアクセス・パラメータを使用できます。アクセス・ドライバで指定する情報により、データ・ソースのデータが、外部表の定義と一致するように処理されます。

ORACLE_LOADERアクセス・パラメータが提供する外部テーブル管理機能を使用するには、使用するプラットフォームのデータ・ファイルのファイル形式とレコード形式(文字セット、フィールドのデータ型など)についての知識が必要です。また、外部表を作成し、その外部表に問合せを実行するためのSQLの知識も必要です。

SQL*LoaderでEXTERNAL_TABLE=GENERATE_ONLYパラメータを使用すると、任意のSQL*Loader制御ファイルに適正なアクセス・パラメータを取得できます。GENERATE_ONLYを指定すると、制御ファイルに記述されているとおり、SQL*Loaderログ・ファイル内の外部表を使用してロードを行うために必要なすべてのSQL文が書き込まれます。それらのSQL文は編集およびカスタマイズできます。それらの文をSQL*Plusで実行すると、SQL*Loaderを使用せずに実際のロードを後か

ら実行できます。

ノート:

- ORACLE_LOADER アクセス・ドライバのパラメータ構文は、別の ORACLE_LOADER アクセス・ドライバのパラメータを参考にしないと理解することが困難になることがあります。特定のパラメータの構文が理解しにくいときには、そのパラメータを別の参考パラメータと関連付けて参照してください。
- CREATE TABLE...ORGANIZATION EXTERNAL 文について示す ORACLE_LOADER アクセス・ドライバのパラメータ例では、その後に外部表用のデータ・ファイルのコンテンツの例が続いていますが、その例のデータ・ファイルのコンテンツは CREATE TABLE 文の一部分ではない点に注意してください。それらは、例が完全なものになるようにするために例中にのみ存在するものです。
- 外部表アクセス・パラメータにおいて列名や表名などの識別子が指定される場合、特定の値はアクセス・パラメータ・パーサーによって予約語とみなされます。予約語を識別子として使用するには、二重引用符で囲む必要があります。

関連トピック

- EXTERNAL_TABLE
- ORACLE_LOADERアクセス・ドライバの予約語
- Oracle Database管理者ガイド

親トピック: ORACLE_LOADERアクセス・ドライバ

15.2 access_parameters句

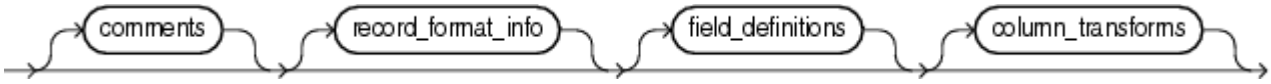
access_parameters句には、コメント、レコード形式およびフィールド形式の情報が含まれています。

デフォルト

ありません。

構文

access_parameters句の構文は次のとおりです。



用途

データ・ソースのデータの記述は外部表の定義とは別です。これは、次のことを意味します。

- ソース・ファイルに含まれるフィールドの数は、外部表の列数と異なる場合があります。
- データ・ソースのフィールドのデータ型は、外部表の列のデータ型と異なる場合があります。

アクセス・ドライバによって、データ・ソースのデータが、外部表の定義と一致するように処理されます。

ノート:

これらのアクセス・パラメータは、SQL 文の CREATE TABLE...ORGANIZATION EXTERNAL の

opaque_format_spec としてまとめて参照されます。

関連項目:

SQL CREATE TABLE...ORGANIZATION EXTERNAL文を使用するときにopaque_format_specを指定する方法の詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

コメント

コメントは、2つのハイフンで始まり、その後にテキストが続く行です。コメントは、アクセス・パラメータより前に位置する必要があります。たとえば:

```
--This is a comment.
--This is another comment.
RECORDS DELIMITED BY NEWLINE
```

二重ハイフンの右側のすべてのテキストは行末まで無視されます。

record_format_info

record_format_info句はオプションであり、レコード(形式など)、データの文字セット、およびレコードをロードの対象外とする規則についての情報が含まれます。構文の詳細は、「record_format_info句」を参照してください。

field_definitions

field_definitions句を使用して、データ・ファイルのフィールドを指定します。データ・ファイルのフィールドが外部表の列と同じ名前の場合、フィールドのデータはその列に使用されます。構文の詳細は、「field_definitions句」を参照してください。

column_transforms

column_transforms句はオプションであり、データ・ファイルの列に直接マップされない外部表の列のロード方法を記述するために使用します。NULL、CONSTANT、CONCATおよびLOBFILE変換を使用して行います。構文の詳細は、「column_transforms句」を参照してください。

親トピック: ORACLE_LOADERアクセス・ドライバ

15.3 record_format_info句

record_format_info句とその副次句を使用して、レコード情報を解析、ラベル付け、管理する方法を学習します。

- record_format_info句の概要
  record_format_info句には、レコード(形式など)、データの文字セットおよびレコードをロード対象外とする規則についての情報が含まれます。
- FIXED長
  record_format_info FIXED句は、外部ファイルのすべてのレコードを固定のバイト長サイズとして識別するために使用します。
- VARIABLEサイズ
  record_format_info VARIABLE句は、レコードが可変長であること示すために使用します
- DELIMITED BY
  record_format_info DELIMITED BY句は、レコード終了文字を区切りにするために使用します。
- XMLTAG
  record_format_info XMLTAG句は、XMLドキュメントからサブドキュメントをロードするために使用するXMLタグ

の指定に使用します。

- CHARACTERSET
  record_format_info CHARACTERSET句は、データ・ファイルの文字セットを指定するために使用します。
- EXTERNAL VARIABLE DATA
  ORACLE_DATAPUMPアクセス・ドライバで生成されたダンプ・ファイルをOracle SQL Connector for HDFSにロードするには、EXTERNAL VARIABLE DATA句を使用します。
- PREPROCESSOR
  すべてのデータ・ファイルに対して実行する独自のプリプロセッサ・プログラムを指定するには、record_format_info PREPROCESSOR句を使用します。
- LANGUAGE
  LANGUAGE句を使用すると、言語名(FRENCHなど)を指定できます。この言語名から、データのロケールに依存する情報が得られます。
- TERRITORY
  TERRITORY句を使用すると、入力データの特性をより詳細に判断するための地域名を指定できます。
- DATA IS...ENDIAN
  DATA IS...ENDIAN句を使用して、データ・ファイルを生成したプラットフォームによってバイト順序が異なるデータのエンディアンを指定します。
- BYTEORDERMARK [CHECK | NOCHECK]
  record_format_info BYTEORDERMARK句を使用すると、データ・ファイルのバイト順序マーク(BOM)の有無をチェックするかどうかを指定できます。
- STRING SIZES ARE IN
  record_format_info STRING SIZES ARE IN句を使用すると、文字列に指定された長さがバイト単位か文字単位かを指定できます。
- LOAD WHEN
  record_format_info LOAD WHEN句を使用すると、データベースに渡すレコードを識別できます。
- BADFILE | NOBADFILE
  record_format_info BADFILE句を使用すると、エラーのためにレコードをロードできない場合にレコードが書き込まれるファイルの名前を指定できます。
- DISCARDFILE | NODISCARDFILE
  record_format_info DISCARDFILE句を使用すると、LOAD WHEN句の条件を満たさないレコードが書き込まれるファイルの名前を指定できます。
- LOGFILE | NOLOGFILE
  record_format_info LOGFILE句を使用すると、データ・ファイル内のデータへのアクセス中に外部表ユーティリティによって生成されたメッセージを含むファイルに名前を付けることができます。
- SKIP
  record_format_info SKIP句を使用すると、ロードする前にデータファイル内の指定した数のレコードをスキップできます。
- FIELD NAMES
  record_format_info FIELD NAMES句を使用して、データファイル内のフィールド順序を指定します。
- READSIZE
  READSIZEパラメータでは、レコードの処理に使用する読取りバッファのサイズを指定します。
- DISABLE_DIRECTORY_LINK_CHECK
  DISABLE_DIRECTORY_LINK_CHECKパラメータは、ORACLE_LOADERアクセス・ドライバにシンボリック・リンク・

チェックをバイパスするように指示します。

- DATE_CACHE
- 文字列

  文字列は、引用符で囲まれた一連の文字または16進数字です。
- condition_spec

  condition_specでは、ブール演算子で結合される1つ以上の条件を指定します。
- [directory object name:] [filename]

  [directory object name:] [filename]句を使用して、出力ファイルの名前(BADFILE、DISCARDFILEまたはLOGFILE)を指定します。
- condition
- IO_OPTIONS句
- DNFS_DISABLE | DNFS_ENABLE
- DNFS_READBUFFERS

  record_format_info句のDNFS_READBUFFERSパラメータを使用して、Direct NFSクライアントによって使用される読取りバッファの数を制御します。

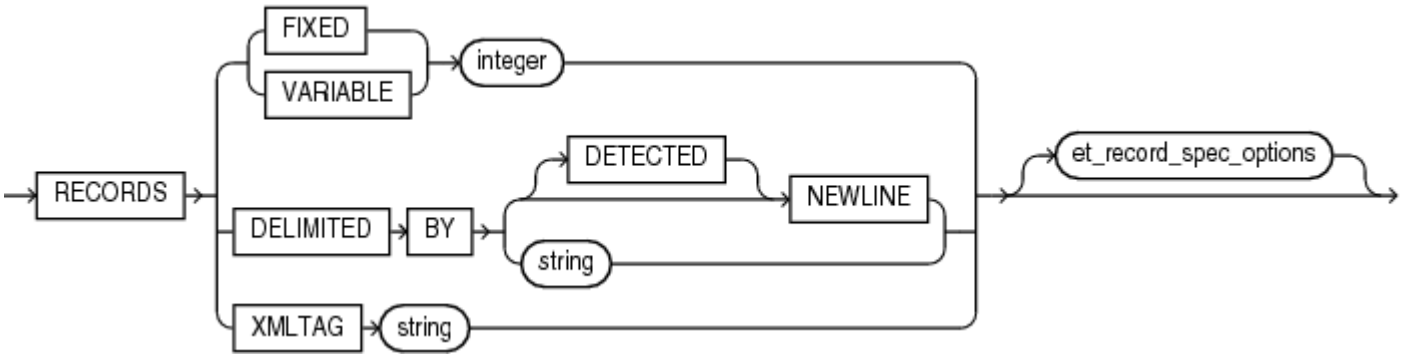
親トピック: ORACLE_LOADERアクセス・ドライバ

15.3.1 record_format_info句の概要

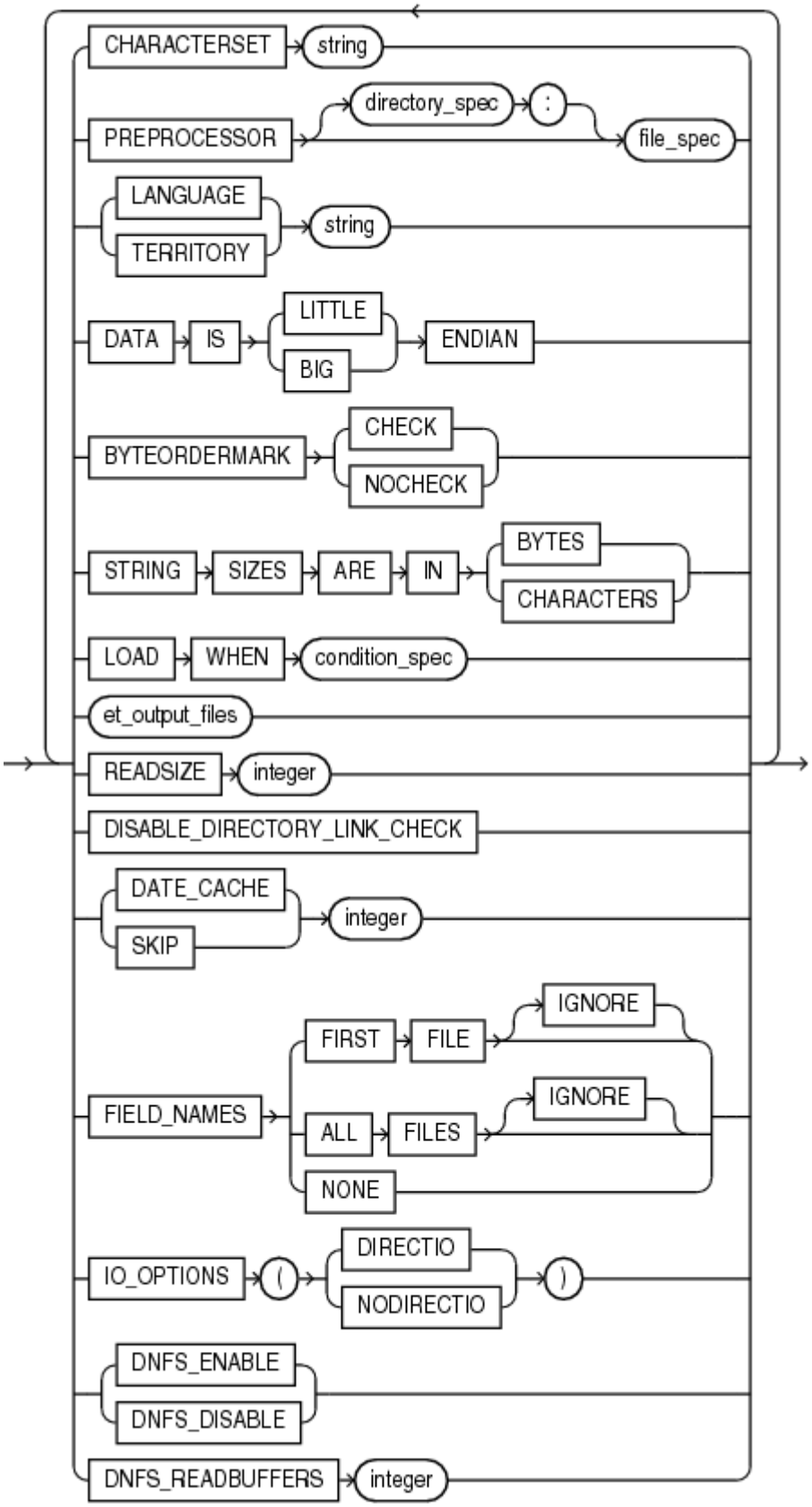
record_format_info句には、レコード(形式など)、データの文字セットおよびレコードをロード対象とする規則についての情報が含まれます。

PREPROCESSOR句を使用すると、ユーザーが用意したプログラムの名前を指定し、そのプログラムを実行して、ORACLE_LOADERアクセス・ドライバで解析できるようにデータ・ファイルの内容を変更できます。

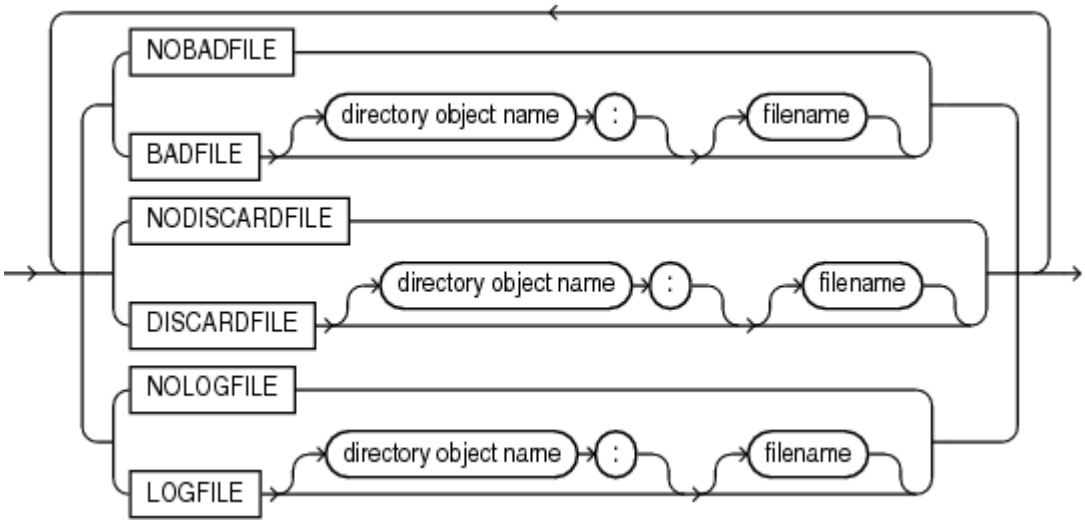
record_format_info句はオプションです。record_format_info句の構文は次のとおりです。



et_record_spec_options句を使用すると、追加の書式情報をオプションで指定できます。書式オプションは必要な数だけ任意の順序で指定できます。オプションの構文は次のとおりです。



次の`et_output_files`の図に、不良ファイル、廃棄ファイルおよびログ・ファイルを指定するオプションを示します。これらの句ごとに、ディレクトリ・オブジェクト名またはファイル名(あるいはその両方)を指定する必要があります。



親トピック: record_format_info句

15.3.2 FIXED長

record_format_info FIXED句は、外部ファイルのすべてのレコードを固定のバイト長サイズとして識別するために使用します。

デフォルト

ありません。

用途

外部表のすべてのレコードを固定のバイト長サイズとして識別できます。

使用上のノート

FIXEDレコードに対して指定したサイズには、改行などのレコード終了文字を含める必要があります。他のレコード型と比較して、固定長レコードの固定長フィールドは、アクセス・ドライバを最も簡単に処理できるフィールドおよびレコード形式です。

例

次に、FIXEDレコードが使用されている例を示します。この例では、データ・ファイルの各レコードの末尾に1バイトの改行文字があるとします。FIXEDを使用したcreate tableコマンドの後に、その表にロードできるデータ・ファイルの例を示しています。

```
CREATE TABLE emp_load (first_name CHAR(15), last_name CHAR(20), year_of_birth
CHAR(4))
  ORGANIZATION EXTERNAL (TYPE ORACLE_LOADER DEFAULT DIRECTORY ext_tab_dir
                         ACCESS PARAMETERS (RECORDS FIXED 20 FIELDS (first_name
CHAR(7),
                                                                     last_name CHAR(8),
                                                                     year_of_birth
CHAR(4)))
                         LOCATION ('info.dat'));
Alvin   Tolliver1976
KennethBaer     1963
Mary    Dube    1973
```

親トピック: record_format_info句

15.3.3 VARIABLEサイズ

record_format_info VARIABLE句は、レコードが可変長であること示すために使用します

デフォルト

ありません。

用途

VARIABLE句を使用して、レコードを可変長として識別します。各レコードの先頭に、レコードのバイト数を示す文字列が付きます。カウント・フィールドを含む文字列の長さは、VARIABLEパラメータの後に続くサイズ引数となります。サイズは、文字数ではなく、バイト数で表されることに注意してください。レコードの先頭の数値にレコード終了文字の分が含まれる必要があります。ただし、カウント・フィールド自身のサイズは含まれません。レコード終了文字のバイト数は、ファイルの作成方法および作成時のプラットフォームによって異なります。

例

次のVARIABLEレコードの使用例では、データ・ファイルの各レコードの末尾に1バイトの改行文字があります。SQL例の後に、ロードに使用できるデータ・ファイルの例を示します。

```
CREATE TABLE emp_load (first_name CHAR(15), last_name CHAR(20), year_of_birth
CHAR(4))
  ORGANIZATION EXTERNAL (TYPE ORACLE_LOADER DEFAULT DIRECTORY ext_tab_dir
                         ACCESS PARAMETERS (RECORDS VARIABLE 2 FIELDS TERMINATED BY
','
                                            (first_name CHAR(7),
                                             last_name CHAR(8),
                                             year_of_birth CHAR(4)))
                         LOCATION ('info.dat'));
21Alvin,Tolliver,1976,
19Kenneth,Baer,1963,
16Mary,Dube,1973,
```

親トピック: record_format_info句

15.3.4 DELIMITED BY

record_format_info DELIMITED BY句は、レコード終了文字を区切りにするために使用します。

デフォルト

なし

用途

DELIMITED BY句を使用して、レコードの終わりを識別する文字を指定します。

DELIMITED BY NEWLINEを指定したときに、実際に使用される値はプラットフォームに依存します。UnixまたはLinuxのオペレーティング・システムでは、NEWLINEは¥nとみなされます。Microsoft Windowsオペレーティング・システムでは、NEWLINEは¥r¥nとみなされます。

データ・ファイルの作成時に使用されたレコード・デリミタがわからないときに、DELIMITED BY NEWLINEを指定した外部表問合せを実行すると、ファイルが不正確にロードされることがあります。この問合せは、データ・ファイルの作成時に使用されたレコード・デリミタを特定することなく実行できます。たとえば、UnixまたはLinuxのオペレーティング・システムで作業するときに、Windows形式で作成されたファイルを使用することもあります。UNIXまたはLinuxのオペレーティング・システムでRECORDS DELIMITED BY NEWLINEを指定すると、デリミタは自動的に¥nとみなされます。ただし、ファイルはWindows形式で作成され、レコードが¥r¥nで区切られているため、このファイルはUNIXまたはLinuxのオペレーティング・システムに正しくアップロードされません。

異なるレコード・デリミタの問題を解決するには、次の構文を使用します。

```
RECORDS DELIMITED BY DETECTED NEWLINE
```

この構文により、ORACLE_LOADERアクセス・ドライバはデータをスキャンして最初にWindowsのデリミタ('¥r¥n')を探します。Windowsのデリミタが見つからなかった場合、アクセス・ドライバはUnixまたはLinuxのデリミタ('¥n')を探します。見つかった最初のデリミタは、レコード・デリミタとして使用されます。

レコード・デリミタが見つかると、そのデリミタはアクセス・ドライバによってレコードの末尾として識別されます。このため、レコードの末尾より前のデータに埋込みデリミタ文字が含まれていると、DETECTEDキーワードが使用できなくなります。これは、ORACLE_LOADERアクセス・ドライバが、フィールド内のデリミタをレコードの末尾を示しているものと誤って想定するためです。結果として、ファイル内の現行のレコードと後続のレコードが正しく解析されなくなります。

同じファイル内に改行デリミタを混在させることはできません。ORACLE_LOADERが最初のデリミタを見つけると、それがファイル内のレコードを識別するデリミタになります。アクセス・ドライバは、同じ改行文字をデリミタとして使用して、ファイル内の後続のすべてのレコードを処理します。

DELIMITED BY stringを指定する場合、stringはテキストにするか、引用符で囲んだ0XまたはXで始まる一連の16進数にします。文字列がテキストの場合は、データ・ファイルの文字セットに変換され、その結果がレコードの境界の識別に使用されます。

次の条件を満たす場合は、デリミタの識別には16進数字を使用する必要があります。

- アクセス・パラメータの文字セットがデータ・ファイルの文字セットとは異なる場合。
- 区切り文字列中にデータ・ファイルの文字セットに変換できない文字がある場合。

16進数字はバイトに変換されます。16進文字列では文字セットの変換は実行されません。

ファイルの終わりがレコード終了記号の前で検出された場合、アクセス・ドライバは、終了記号が検出された場合と同様に、ファイルの終わりまでの処理されていないすべてのデータをレコードの部分とみなします。

> ノート:
>
> デリミタ付きのレコードには、VARCHAR および VARRAW のバイナリ数値を含むバイナリ・データを含めないでください。バイナリ・データを含めると、そのバイナリ・データがデリミタの検索中に文字として解釈されるため、エラーまたは破損が発生します。

例

次に、DELIMITED BYレコードの使用例を示します。

```
CREATE TABLE emp_load (first_name CHAR(15), last_name CHAR(20), year_of_birth
CHAR(4))
  ORGANIZATION EXTERNAL (TYPE ORACLE_LOADER DEFAULT DIRECTORY ext_tab_dir
                         ACCESS PARAMETERS (RECORDS DELIMITED BY '|' FIELDS
TERMINATED BY ','
                                            (first_name CHAR(7),
                                             last_name CHAR(8),
                                             year_of_birth CHAR(4)))
                         LOCATION ('info.dat'));
Alvin,Tolliver,1976|Kenneth,Baer,1963|Mary,Dube,1973
```

関連トピック

- string

親トピック: record_format_info句

15.3.5 XMLTAG

record_format_info XMLTAG句は、XMLドキュメントからサブドキュメントをロードするために使用するXMLタグの指定に使用します。

デフォルト

なし

用途

ORACLE_LOADERアクセス・ドライバのXMLTAG句を使用して、XMLドキュメントからのサブドキュメントのロードに使用するXMLタグを指定できます。アクセス・ドライバは、句で識別するタグで囲まれたドキュメントのデータ・ファイルを検索し、それらのドキュメントを個別の行として外部表にロードします。

XMLTAG句は、1つ以上の文字列のリストを受け入れます。これらの文字列を使用して、ORACLE_LOADERがデータ・ファイル内のサブドキュメントを検索するときに使用するタグが作成されます。アクセス・パラメータで指定するタグには、"<"および">"デリミタは含めません。

ORACLE_LOADERアクセス・ドライバは、ファイルの先頭からXMLTAG句にリストされたタグが最初に出現する箇所検索します。一致が見つかると、対応する閉じタグを検索します。たとえば、タグが"ORDER_ITEM"の場合、ORACLE_LOADERは、ファイルの先頭からテキスト文字列"<ORDER_ITEM>"を検索します。"<ORDER_ITEM>"が出現する箇所が見つかると、次に、"</ORDER_ITEM>"を検索します。タグ<ORDER_ITEM>と</ORDER_ITEM>の間のすべての内容が、ドキュメントから行にロードされる部分となります。その後、ORACLE_LOADERは、閉じタグの後の最初の文字から、次に出現するタグを検索します。

ORACLE_LOADERアクセス・ドライバは、タグ名と一致する要素に対してXMLドキュメントを解析するのではなく、単に、テキスト・ファイル全体の文字列検索を実行します。外部表がパラレルにアクセスされている場合、ORACLE_LOADERは、各セクションが独立して読み取られるように、大きいファイルを分割します。データ・ファイルのセクションの読取りを開始するとき、XMLTAGで指定されたタグの1つの検索を開始します。セクションの終わりに到達したが、まだ一致する終了タグを検索している場合、ORACLE_LOADERは、一致する終了タグが見つかるまで引き続き次のセクションを読み取ります。

XMLTAGの使用時の制限事項

- 要素が同じ要素のドキュメントの内側にネストされているデータ・ファイルは、XMLTAG句を使用してロードすることはできません。たとえば、XMLTAG(‘FOO’)を使用してロードするデータ・ファイルに次のデータが含まれているとします。

  ```
  <FOO><BAR><FOO></FOO></BAR></FOO>
  ```

  この場合、ORACLE_LOADERでは最初の<FOO>と最初の</FOO>の間にあるすべての内容をドキュメントとして抽出しますが、これは有効なドキュメントを構成していません。

  同様に、XMLTAG(“FOO”,”BAR”)を指定した場合、データ・ファイルに次のデータが含まれているとします。

  ```
  <FOO><BAR></BAR></FOO>
  ```

  この場合は、<BAR>と</BAR>がロードされますが、"FOO"のドキュメントとしてロードされます。

- 抽出されるサブドキュメントの大きさの制限は、READSIZEアクセス・パラメータによって決定されます。ORACLE_LOADERアクセス・ドライバがREADSIZEよりも大きいサブドキュメントを検出した場合、エラーが返されます。

XMLTAG句の使用例

たとえば、次のようにして外部表T_XTを作成するとします。

```
CREATE TABLE "T_XT"
```

```
(
  "C0" VARCHAR2(2000)
)
ORGANIZATION external
(
  TYPE oracle_loader
  DEFAULT DIRECTORY DMPDIR
  ACCESS PARAMETERS
  (
    RECORDS
    XMLTAG ("home address", "work address"," home phone ")
    READSIZE 1024
    SKIP 0
    FIELDS NOTRIM
    MISSING FIELD VALUES ARE NULL
    (
      "C0" (1:2000) CHAR(2000)
    )
  )
  location
  (
    't.dat'
  )
)REJECT LIMIT UNLIMITED
/
exit;
```

データ・ファイルの内容が次のとおりだとします。

```
<first name>Lionel</first name><home address>23 Oak St, Tripoli, CT</home
address><last name>Rice</last name>
```

この場合、次のSQL問合せを実行できます。

```
SQL> SELECT C0 FROM T_XT;
C0
--------------------------------------------------------------------
<home address>23 Oak St, Tripoli, CT</home address>
```

親トピック: record_format_info句

15.3.6 CHARACTERSET

`record_format_info CHARACTERSET`句は、データ・ファイルの文字セットを指定するために使用します。

デフォルト

ありません。

用途

`CHARACTERSET string`句を使用して、データ・ファイルの文字セットを識別します。文字セットを指定しない場合、データベースのデフォルトの文字セットが使用されます。

ノート:

クライアントの NLS 環境変数設定は、データベースに使用される文字セットに影響しません。

関連トピック

- string
- Oracle Databaseグローバリゼーション・サポート・ガイド

親トピック: record_format_info句

15.3.7 EXTERNAL VARIABLE DATA

ORACLE_DATAPUMPアクセス・ドライバで生成されたダンプ・ファイルをOracle SQL Connector for HDFSにロードするには、EXTERNAL VARIABLE DATA句を使用します。

デフォルト

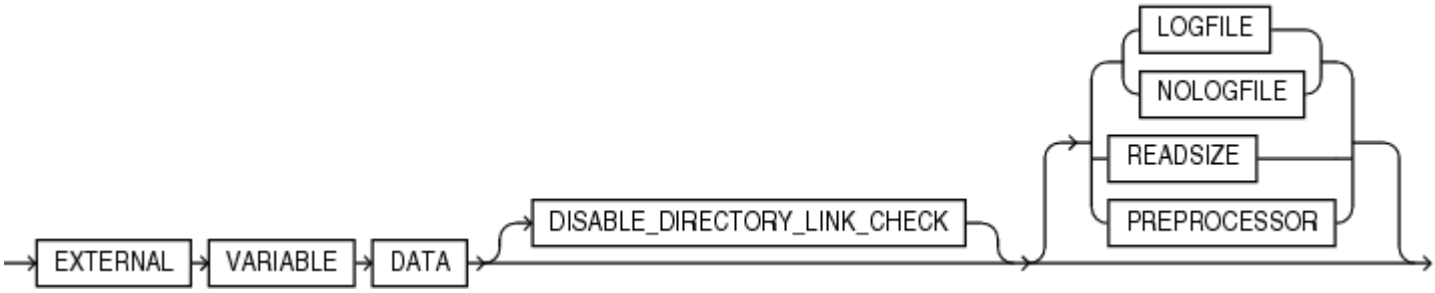
ありません。

用途

EXTERNAL VARIABLE DATA句を指定し、ORACLE_LOADERアクセス・ドライバを使用してORACLE_DATAPUMPアクセス・ドライバで生成されたダンプ・ファイルをロードします。

> ノート:
>
> EXTERNAL VARIABLE DATA 句が Oracle SQL Connector for Hadoop Distributed File System (HDFS)とともに使用する場合にのみ、有効です。Oracle SQL Connector for HDFS の詳細は、『*Oracle Big Data Connectors* ユーザーズ・ガイド』を参照してください。

構文および説明



EXTERNAL VARIABLE DATA句には、次のアクセス・パラメータのみが使用できます。

- LOGFILE | NOLOGFILE
- READSIZE
- PREPROCESSOR

> ノート:
>
> パラメータ DISABLE_DIRECTORY_LINK_CHECK はサポート対象外となりました。

例

ここに示すEXTERNAL VARIABLE DATA句の使用例は、次のシナリオが該当します。

- deptxt1.dmpダンプ・ファイルは、事前にORACLE_DATAPUMPアクセス・ドライバで生成されています。
- PREPROCESSORパラメータで指定されたtkexcatプログラムは、入力データの処理に使用するためにユーザーが用

意したプログラムです。

```
CREATE TABLE deptxt1
(
   deptno  number(2),
   dname   varchar2(14),
   loc     varchar2(13)
)
ORGANIZATION EXTERNAL
(
  TYPE ORACLE_LOADER
  DEFAULT DIRECTORY dpump_dir
  ACCESS PARAMETERS
  (
    EXTERNAL VARIABLE DATA
    LOGFILE 'deptxt1.log'
    READSIZE=10000
    PREPROCESSOR tkexcat
  )
  LOCATION ('deptxt1.dmp')
)
REJECT LIMIT UNLIMITED
;
```

関連トピック

- LOGFILE | NOLOGFILE
- READSIZE
- PREPROCESSOR

親トピック: record_format_info句

15.3.8 PREPROCESSOR

すべてのデータ・ファイルに対して実行する独自のプリプロセッサ・プログラムを指定するには、`record_format_info` `PREPROCESSOR`句を使用します。

デフォルト

ありません。

用途

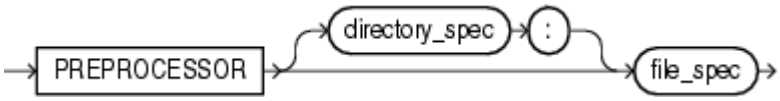
> 注意:
>
> `PREPROCESSOR` 句を使用するときに考慮する必要があるセキュリティ上の注意事項があります。

ロードするファイルに`ORACLE_LOADER`アクセス・ドライバでサポートしていない形式のデータ・レコードが含まれる場合は、`PREPROCESSOR`句を使用して、ユーザーが用意するプリプロセッサ・プログラムを指定し、そのプログラムをすべてのデータ・ファイルに対して実行します。プログラムの指定は、引数を使用する場合にシェル・スクリプトに入れる必要があります(「`file_spec`」の説明を参照)。

プリプロセッサ・プログラムは、アクセス・ドライバがサポートするレコード形式にデータを変換し、変換したレコード・データを標準出力(stdout)に書き出します。アクセス・ドライバはそのデータを入力として読み取ることになります。

構文

PREPROCESSOR句の構文は次のとおりです。



directory_spec

各データ・ファイルに対して実行するプリプロセッサ・プログラムの名前が含まれるディレクトリ・オブジェクトを指定します。外部表にアクセスするユーザーには、使用されるディレクトリ・オブジェクトのEXECUTE権限が必要です。directory_specを省略した場合は、外部表に対して指定されたデフォルトのディレクトリが使用されます。

> 注意:
>
> セキュリティ上の理由から、プリプロセッサ・プログラムの格納には個別のディレクトリを使用するようにしてください。デフォルト・ディレクトリは使用しないでください。プリプロセッサ・プログラムが格納されているディレクトリには別のファイルを保管しないでください。

セキュリティを維持するために、プリプロセッサ・プログラムは、ディレクトリ・オブジェクト内に存在する必要があります。これは、プログラムへのアクセスを制御できるようにするためです。オペレーティング・システム管理者は、そのディレクトリ・オブジェクトに対応するディレクトリを作成する必要があります。また、データベース用のオペレーティング・システムOracleユーザーが、そのディレクトリにアクセスできることを確認する必要もあります。データベース管理者は、承認されたユーザーのみがディレクトリ・パスに関連付けられたディレクトリ・オブジェクトへのアクセス許可を付与されるようにする必要があります。ディレクトリ・オブジェクトのアクセス権は、複数のデータベース・ユーザーに付与できますが、そのディレクトリ内のプリプロセッサを実行できるのはEXECUTE権限を持つユーザーのみです。ディレクトリ・オブジェクトへの読取り/書込み権限を持つ既存データベース・ユーザーでは、前処理の機能を使用できません。DBAとして、どのユーザーにもディレクトリ・オブジェクトのEXECUTE権限を付与しなければ、プリプロセッサの使用を禁止できます。EXECUTE権限の付与方法の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

file_spec

プリプロセッサ・プログラムの名前です。この名前は、使用されているディレクトリ・オブジェクト(directory_specまたは外部表のデフォルト・ディレクトリのいずれか)に関連付けられたパス名の後ろに追加されます。file_specに絶対ディレクトリ・パスや相対ディレクトリ・パスを含めることはできません。

プリプロセッサ・プログラムに引数(gunzip -cなど)が必要な場合は、プログラム名とその引数を実行可能シェル・スクリプト(Microsoft Windowsオペレーティング・システムではバッチ(.bat)・ファイル)で指定する必要があります。次の各項で説明するように、シェル・スクリプトとバッチ・ファイルには特定の要件があります。

オペレーティング・システム・ディレクトリに正しいバージョンのプリプロセッサ・プログラムがあることを必ず確認してください。

シェルやバッチ・ファイルを使用しないで、PREPROCESSOR句を指定する例を次に示します。

```
SQL> CREATE TABLE xtab (recno varchar2(2000))
     2    ORGANIZATION EXTERNAL (
     3    TYPE ORACLE_LOADER
     4    DEFAULT DIRECTORY data_dir
     5    ACCESS PARAMETERS (
     6    RECORDS DELIMITED BY NEWLINE
     7    PREPROCESSOR execdir:'zcat'
     8    FIELDS (recno char(2000)))
     9    LOCATION ('foo.dat.gz'))
   10    REJECT LIMIT UNLIMITED;
Table created.
```

Linuxオペレーティング・システムにおけるPREPROCESSOR句でのシェル・スクリプトの使用

Linuxでシェル・スクリプトを使用する場合は、次の条件が満たされている必要があります。

- シェル・スクリプトは、directory_specに存在する必要があります。
- gunzipなどのシステム・コマンドのフルパス名を指定する必要があります。
- プリプロセッサのシェル・スクリプトには、EXECUTE権限が必要です。
- 外部表のLOCATION句にリストされているデータ・ファイルは、$1で参照する必要があります。

次の例は、外部表作成時のPREPROCESSOR句でのシェル・スクリプトの指定方法を示しています。

```
SQL> CREATE TABLE xtab (recno varchar2(2000))
     2    ORGANIZATION EXTERNAL (
     3    TYPE ORACLE_LOADER
     4    DEFAULT DIRECTORY data_dir
     5    ACCESS PARAMETERS (
     6    RECORDS DELIMITED BY NEWLINE
     7    PREPROCESSOR execdir:'uncompress.sh'
     8    FIELDS (recno char(2000)))
     9    LOCATION ('foo.dat.gz'))
   10    REJECT LIMIT UNLIMITED;
Table created.
```

Windowsオペレーティング・システムにおけるPREPROCESSOR句でのバッチ・ファイルの使用

Microsoft Windowsでシェル・スクリプトを使用するには、次の条件が満たされている必要があります。

- バッチ・ファイルは、directory_specに存在する必要があります。
- gunzipなどのシステム・コマンドのフルパス名を指定する必要があります。
- プリプロセッサのバッチ・ファイルには、EXECUTE権限が必要です。
- バッチ・ファイルの最初の行には、@echo offを含める必要があります。これが必要になるのは、バッチ・ファイルが実行されるとき、デフォルトでは実行中のコマンドが表示されるため、このことによって、エコー処理されたコマンドが外部表のアクセス・ドライバへの入力として処理されるという意図しない副次的な影響を回避するためです。
- location句からの入力を表すには、%1を使用する必要があります。(これは、location句が$1により参照されるUNIXおよびLinuxスタイルのシェル・スクリプトとは異なります。)
- バッチ・ファイル内のすべての実行可能ファイル(次の例ではsed.exe)のフルパスを指定する必要があります。すべてのMicrosoft WindowsインストールにMKSツールキットが存在するとはかぎらないため、sed.exeなどのコマンドが使用できない場合があることにも注意してください。

  Microsoft Windowsで使用するバッチ・ファイルは、.batまたは.cmd拡張子を持つ必要があります。そうでない場合(たとえば、プリプロセッサ・スクリプトをsed.shとして指定しようとした場合)は、次のエラーが発生します。

  ```
  SQL> select * from foo ;
  select * from foo
  *
  ERROR at line 1:
  ORA-29913: error in executing ODCIEXTTABLEFETCH callout
  ORA-29400: data cartridge error
  KUP-04095: preprocessor command
  C:/Temp¥sed.sh encountered error
  "CreateProcess Failure for Preprocessor:
  C:/Temp¥sed.sh, errorcode: 193
  ```

Windowsでバッチ・ファイルを外部表のPREPROCESSORオプションとともに使用する単純な例を、次に示します。この例では、

バッチ・ファイルはストリーム・エディタ(sed.exe)ユーティリティを使用して、入力データの単純な変換を実行します。

```
SQL> create table deptXT (deptno char(2),
  2  dname char(14),
  3  loc char(13)
  4  )
  5  organization external
  6  (
  7  type ORACLE_LOADER
  8  default directory def_dir1
  9  access parameters
 10  (
 11  records delimited by newline
 12  badfile 'deptXT.bad'
 13  logfile 'deptXT.log'
 14  preprocessor exec_dir:'sed.bat'
 15  fields terminated by ','
 16  missing field values are null
 17  )
 18  location ('deptXT.dat')
 19  )
 20  reject limit unlimited ;
Table created.
select * from deptxt ;
Where deptxt.dat contains:
20,RESEARCH,DALLAS
30,SALES,CHICAGO
40,OPERATIONS,BOSTON
51,OPERATIONS,BOSTON
```

プリプロセッサ・プログラムsed.batの内容は次のとおりです。

```
@echo off
c:/mksnt/mksnt/sed.exe -e 's/BOSTON/CALIFORNIA/' %1
```

PREPROCESSORオプションは、入力データ(deptxt.dat)をsed.batに渡します。次にdeptxt表から選択すると、最後の2行のLOC列(以前はBOSTONであった列)がCALIFORNIAになっています。

```
SQL> select * from deptxt ;
DE DNAME           LOC
-- -------------- -------------
20 RESEARCH,      DALLAS
30 SALES          CHICAGO
40 OPERATIONS     CALIFORNIA
51 OPERATIONS     CALIFORNIA
4 rows selected.
```

PREPROCESSOR句によるパラレル処理の使用上のノート

外部表では、LOCATION句で指定した各データ・ファイルが単一グラニュルとして扱われます。PREPROCESSOR句によるパラレル処理を最大限に活用するために、ロードするデータを複数ファイル(グラニュル)に分割することをお薦めします。外部表における並列度が、存在するデータ・ファイル数にまで制限されます。たとえば、並列度を16に指定し、データ・ファイルが10個のみ存在する場合、10個の子プロセスがビジー、6個がアイドル状態になるため、実際の並列度は10になります。データの処理の効率を向上するために、子プロセスがアイドル状態にならないようにします。並列度を指定する場合は、すべての子プロセスがビジー状態を維持するようにして、指定する並列度がデータ・ファイルの数を超えないようにしてください。並列度のグラニュルの詳細は、『*Oracle Database VLDB*およびパーティショニング・ガイド』を参照してください。

また、オブジェクト・ストア・データは、ファイル・システム・ファイルで使用するものと同じプリプロセッサ・スクリプトを使用して処理できないことにも注意してください。オブジェクト・ストア・データにプリプロセッサを使用する場合は、オブジェクト・ストア・データにアクセスできるプリプロセッサ・スクリプトを作成し、データを変更する必要があります。たとえば、LinuxまたはUNIXシステムではこの

場合、$1はhttps://www.yoururl.example.com/yourdataなどのソースを表します。

```
@echo off
#!/bin/sh/your_script_or_plsql_function_to_display_objectstore_contents($1) | sed -e
's/BOSTON/CALIFORNIA/'
```

この構文を使用すると、プリプロセッサがデータを取得してstdoutに送信し、これをパイプしてアクセス・ドライバが読み取れるようにします。

PREPROCESSOR句を使用する際の制限事項

- Oracle Database Vault機能を使用するデータベースでは、PREPROCESSOR句を使用できません。
- PREPROCESSOR句は、COLUMN TRANSFORMS句とともには使用できません。

- PREPROCESSOR句によるパラレル処理の使用
  PREPROCESSOR句によるパラレル処理の使用。
- PREPROCESSOR句を使用する際の制限事項
  PREPROCESSOR句を使用する際の制限事項について説明します。

関連トピック

- ORACLE_LOADERアクセス・ドライバの保護に関するガイドライン
- Oracle Database SQL言語リファレンスGRANT

親トピック: record_format_info句

15.3.8.1 PREPROCESSOR句によるパラレル処理の使用

PREPROCESSOR句によるパラレル処理の使用。

外部表では、LOCATION句で指定した各データ・ファイルが単一グラニュルとして扱われます。PREPROCESSOR句によるパラレル処理を最大限に活用するには、ロードされるデータを複数ファイル(グラニュル)に分割する必要があります。これは、外部表における並列度が、存在するデータ・ファイル数までに制限されるためです。たとえば、並列度を16に指定し、データ・ファイルが10個のみ存在する場合には、10個のスレーブ・プロセスがビジーとなり、6個がアイドル状態になるため、実際の並列度は10となります。アイドル状態のスレーブ・プロセスがないことが最適な状態です。したがって、並列度を指定する場合、すべてのスレーブ・プロセスがビジーになるように、並列度がデータ・ファイル数を超えないことが理想的です。

関連項目:

- 並列度のグラニュルの詳細は、『Oracle Database VLDBおよびパーティショニング・ガイド』を参照してください。

親トピック: PREPROCESSOR

15.3.8.2 PREPROCESSOR句を使用する際の制限事項

PREPROCESSOR句を使用する際の制限事項について説明します。

- Oracle Database Vault機能を使用するデータベースでは、PREPROCESSOR句を使用できません。
- PREPROCESSOR句は、COLUMN TRANSFORMS句とともには使用できません。

親トピック: PREPROCESSOR

15.3.9 LANGUAGE

LANGUAGE句を使用すると、言語名(FRENCHなど)を指定できます。この言語名から、データのロケールに依存する情報が得られます。

次に、言語名から得られる情報のタイプの例を示します。

- 月と日の名前およびその略称
- A.M.、P.M.、A.D.およびB.C.と同じ内容を表す記号
- ORDER BY SQL句が指定されている場合の文字データのデフォルトのソート順序
- 筆記方向(右から左、左から右)
- 肯定および否定応答の文字列(たとえばYESとNO)

関連項目:

Oracleでサポートされる言語のリストは、『Oracle Databaseグローバリゼーション・サポート・ガイド』を参照してください。

親トピック: record_format_info句

15.3.10 TERRITORY

TERRITORY句を使用すると、地域名を指定して、入力データの特性をより詳細に指定することができます。

たとえば、カンマではなく小数点(531,298のかわりに531.298など)が数字に使用される国もあります。

関連項目:

Oracleでサポートされる地域のリストは、『Oracle Databaseグローバリゼーション・サポート・ガイド』を参照してください。

親トピック: record_format_info句

15.3.11 DATA IS...ENDIAN

DATA IS...ENDIAN句を使用して、データ・ファイルを生成したプラットフォームによってバイト順序が異なるデータのエンディアンを指定します。

用途

データ・ファイルを生成したプラットフォームによってバイト順序が異なるデータのエンディアンを指定します。

使用上のノート

次の型のフィールドは、この句の影響を受けます。

- INTEGER
- UNSIGNED INTEGER
- FLOAT
- BINARY_FLOAT
- DOUBLE
- BINARY_DOUBLE
- VARCHAR(数値のみ)
- VARRAW(数値のみ)

- UTF16文字セットの任意の文字データ型
- RECORDS DELIMITED BY string,によって指定するUTF16文字セット文字列

Microsoft Windowsベースのプラットフォームでは、リトル・エンディアン・データが生成されます。ビッグ・エンディアン・プラットフォームには、Oracle SolarisおよびIBM zSeriesベースのLinuxが含まれます。DATA IS...ENDIAN句を指定しない場合、データは、アクセス・ドライバが実行されているプラットフォームと同じエンディアンになります。UTF-16データ・ファイルには、ファイルの先頭にデータのエンディアンを示すマークがある場合があります。存在する場合、このマークはDATA IS...ENDIAN句をオーバーライドします。

親トピック: record_format_info句

15.3.12 BYTEORDERMARK [CHECK | NOCHECK]

record_format_info BYTEORDERMARK句を使用すると、バイト順序マーク(BOM)の有無をデータファイルでチェックするかどうかを指定できます。

デフォルト

CHECK

構文

```
BYTEORDERMARK [CHECK | NOCHECK]
```

用途

BYTEORDERMARK句を使用すると、バイト順序マーク(BOM)の有無をデータファイルでチェックするかどうかを指定できます。この句は、文字セットがUnicodeの場合にのみ有効です。

BYTEORDERMARK NOCHECKを指定すると、データ・ファイルにBOMが存在するかどうかが確認されず、データ・ファイルのすべてのデータがデータとして読み込まれます。

BYTEORDERMARK CHECKを指定すると、データ・ファイルにBOMがあるかどうかを確認します。これはUnicode文字セットのデータ・ファイルについてのデフォルト動作です。

使用上のノート

次に、使用例をいくつか示します。

- データをリトル・エンディアンまたはビッグ・エンディアンとして指定し、CHECKを指定したときにそのエンディアンがデータ・ファイルと一致していないと判断された場合は、エラーが返されます。たとえば、次のパラメータを指定したとします。

  ```
  DATA IS LITTLE ENDIAN
  BYTEORDERMARK CHECK
  ```

  Unicodeデータ・ファイル内にBOMが存在するかどうかが確認され、そのデータが実際にはビッグ・エンディアンであった場合は、リトル・エンディアンを指定していたため、エラーが返されます。

- BOMが存在せず、DATA IS...ENDIANパラメータを使用してエンディアンを指定しない場合は、プラットフォームのエンディアンが使用されます。

- BYTE ORDER MARK NOCHECKを指定し、DATA IS...ENDIANパラメータを使用してエンディアンを指定した場合は、そのエンディアン値が使用されます。それ以外の場合は、プラットフォームのエンディアンが使用されます。

関連トピック

- SQL*Loaderによるバイト順序の管理方法の理解

親トピック: record_format_info句

15.3.13 STRING SIZES ARE IN

record_format_info STRING SIZES ARE IN句を使用すると、文字列に指定された長さがバイト単位か文字単位かを指定できます。

デフォルト

ありません。

構文

```
STRING SIZES ARE IN [BYTES | CHARACTERS]
```

用途

STRING SIZES ARE IN句を使用して、文字列の長さがバイト単位であるか、または文字単位であるかを指定します。この句を指定しない場合、アクセス・ドライバは、データベースが使用するモードを使用します。長さが埋め込まれた文字型(VARCHARなど)も、この句の影響を受けます。この句を指定すると、埋め込まれた長さは、バイト数ではなく、文字数となります。UTF16などのマルチバイト文字セットのロード時には、STRING SIZES ARE IN CHARACTERSを指定する必要があります。

親トピック: record_format_info句

15.3.14 LOAD WHEN

record_format_info LOAD WHEN句を使用すると、データベースに渡されるレコードを識別できます。

デフォルト

構文

LOAD WHEN句の構文は、次のとおりです。ここで、condition_specは条件指定です。

```
LOAD WHEN condition_spec
```

用途

LOAD WHEN condition_spec句を使用して、データベースに渡すレコードを識別します。評価の方法は様々です。

- condition_spec句がレコードのフィールドを参照する場合、この句は、すべてのフィールドがレコードから解析された後で、NULLIF句またはDEFAULTIF句の評価が行われる前にのみ評価されます。
- 条件指定が範囲のみを参照する(フィールド名は参照しない)場合、フィールドが解析される前に句が評価されます。このユースケースは、ロードしないファイル内のレコードをエラーなしで現在のレコード定義に解析できない場合に役立ちます。

例

次に、LOAD WHENの使用例を示します。

```
LOAD WHEN (empid != BLANKS)
LOAD WHEN ((dept_id = "SPORTING GOODS" OR dept_id = "SHOES") AND total_sales != 0)
```

関連トピック

- condition_spec

親トピック: record_format_info句

15.3.15 BADFILE | NOBADFILE

record_format_info BADFILE句を使用すると、エラーのためにレコードをロードできない場合にレコードが書き込まれるファイルに名前を付けることができます。

デフォルト

デフォルト名で不良ファイルを作成します。詳細は、「用途」を参照してください。

構文

```
BADFILE name | NOBADFILE
```

用途

BADFILE句を使用して、エラーのためにロードできない場合にレコードが書き込まれるファイルを指定します。たとえば、データ・ファイルのフィールドを外部表の列のデータ型に変換できなかった場合、不良ファイルにレコードが書き込まれます。不良ファイルの目的は、すべての拒否されたデータを調査および修正して、ファイルをロードできるようにすることです。不良レコードがあってもデータを修正しない場合は、NOBADFILEオプションを使用して不良ファイルの作成を回避できます。

BADFILE句を指定する場合、ディレクトリ・オブジェクト名またはファイル名(あるいはその両方)を指定する必要があります。詳細は、「[directory object name:] [filename]」を参照してください。

NOBADFILEを指定すると、不良ファイルは作成されません。

BADFILEまたはNOBADFILEのいずれも指定しない場合、デフォルトでは1つ以上のレコードが拒否されると、不良ファイルが作成されます。このファイルの名前は、表名の後に_%pが付いたものになります(%pは、ファイル作成プロセスのPIDで置き換えられます)。ファイルには拡張子.badが付けられます。ディレクトリ・ナビゲーションとして解釈できない文字(%、/、*など)が表名に含まれる場合、これらの文字は出力ファイル名に含められません。

LOAD WHEN句が正常に実行されない場合、レコードは不良ファイルには書き込まれず、かわりに、廃棄ファイルに書き込まれます。また、外部表のレコードを使用中にエラーが発生する場合は(外部表に対してINSERT INTO...AS SELECT...を使用した場合の制約違反など)、レコードは不良ファイルに書き込まれません。

関連トピック

- [directory object name:] [filename]

親トピック: record_format_info句

15.3.16 DISCARDFILE | NODISCARDFILE

record_format_info DISCARDFILE句を使用すると、LOAD WHEN句の条件を満たさないレコードが書き込まれるファイルに名前を付けることができます。

デフォルト

デフォルト名で廃棄ファイルを作成します。詳細は、「用途」を参照してください。

構文

```
DISCARDFILE name | NODISCARDFILE
```

用途

DISCARDFILE句を使用して、レコードがLOAD WHEN句の条件を満たすことができないことが書き込まれるファイルを指定します。この廃棄ファイルは、廃棄される最初のレコードが検出されると作成されます。同じ外部表が複数回アクセスされる場合、廃棄ファイルはそのたびに再度書き込まれます。廃棄レコードを個別のファイルに保存する必要がない場合、NODISCARDFILEを使用します。

DISCARDFILEを指定する場合、ディレクトリ・オブジェクト名またはファイル名(あるいはその両方)を指定する必要があります。詳細は、「[directory object name:] [filename]」を参照してください。

NODISCARDFILEを指定すると、廃棄ファイルは作成されません。

DISCARDFILE またはNODISCARDFILEのいずれも指定しない場合、デフォルトでは1つ以上のレコードでLOAD WHEN句が失敗すると、廃棄ファイルが作成されます。このファイルの名前は、表名の後に_%pが付いたものになります(%pは、ファイル作成プロセスのPIDで置き換えられます)。ファイルには拡張子.dcsが付けられます。ディレクトリ・ナビゲーションとして解釈できない文字(%、/、*など)が表名に含まれる場合、これらの文字はファイル名に含められません。

関連トピック

- [directory object name:] [filename]

親トピック: record_format_info句

15.3.17 LOGFILE | NOLOGFILE

record_format_info LOGFILE句を使用して、データ・ファイル内のデータへのアクセス中に外部表ユーティリティによって生成されたメッセージを含むファイルに名前を付けます。

デフォルト

既存のファイルを使用するか、デフォルト名でログ・ファイルを作成します。詳細は、「用途」を参照してください。

構文

```
LOGFILE name | NOLOGFILE
```

用途

LOGFILE句を使用して、データ・ファイルのデータへのアクセス中に外部表のユーティリティによって生成されたメッセージを含むファイルを指定します。ログ・ファイルがすでに同じ名前で存在する場合は、アクセス・ドライバによってそのログ・ファイルが再びオープンされ、新しいログ情報がファイルの終わりに追加されます。この点では、既存のファイルを上書きする不良ファイルおよび廃棄ファイルとは異なります。NOLOGFILE句を使用してログ・ファイルの作成を回避できます。

LOGFILEを指定する場合、ディレクトリ・オブジェクト名またはファイル名(あるいはその両方)を指定する必要があります。詳細は、「[directory object name:] [filename]」を参照してください。

NOLOGFILEを指定した場合、ログ・ファイルは作成されません。

LOGFILEまたはNOLOGFILEのいずれも指定しない場合、デフォルトではログ・ファイルが作成されます。このファイルの名前は、表名の後に_%pが付いたものになります(%pは、ファイル作成プロセスのPIDで置き換えられます)。ファイルには拡張子.logが付けられます。ディレクトリ・ナビゲーションとして解釈できない文字(%、/、*など)が表名に含まれる場合、これらの文字はファイル名に含められません。

関連トピック

- [directory object name:] [filename]

親トピック: record_format_info句

15.3.18 SKIP

record_format_info SKIP句を使用すると、ロードする前にデータファイル内の指定した数のレコードをスキップできます。

デフォルト

なし (0)

構文

構文は次のとおりです。ここで、numはスキップするレコードの数です(デフォルトは0)。

```
SKIP = num
```

用途

SKIP句は、ロードの前に、データ・ファイルに含まれる指定した件数のレコードをスキップします。この句を指定できるのは、データにパラレルにアクセスしない場合にのみです。同じ表について同じ場所に複数のデータ・ファイルがある場合、SKIP句を指定すると、ORACLE_LOADERドライバが最初のデータ・ファイルに含まれる指定した件数のレコードのみをスキップします。

親トピック: record_format_info句

15.3.19 FIELD NAMES

record_format_info FIELD NAMES句を使用して、データファイルのフィールド順序を指定します。

デフォルト

NONE

構文

```
FIELD NAMES {FIRST FILE | FIRST IGNORE | ALL FILES | ALL IGNORE| NONE}
```

FIELD NAMESのオプションは次のとおりです。

- FIRST FILE—最初のデータ・ファイルに、最初のレコードのデータに対応するフィールド名のリストが含まれることを示します。このリストでは、データ・ファイルのデータと同じデリミタが使用されます。このレコードは、データ・ファイルのフィールドとターゲット表の列とのマッピングを設定するために読み取られ、使用されます。このレコードは、データが処理されるとスキップされます。このオプションは、データ・ファイルのフィールドの順序が表の列の順序とは異なる場合に役立ちます。
- FIRST IGNORE—最初のデータ・ファイルに、最初のレコードのデータに対応するフィールド名のリストが含まれるが、その情報は無視する必要があることを示します。このレコードは、データが処理されるとスキップされますが、フィールドの設定には使用されません。
- ALL FILES — すべてのデータ・ファイルに、最初のレコードのデータに対応するフィールド名のリストが含まれることを示します。データ・ファイル内のフィールドの順序は、どのような順序でもかまいません。順序は、各ファイルの最初の行によって指定されます。アクセス・ドライバでは、外部表の列とは異なる順序でフィールドが指定されます。
- ALL IGNORE—すべてのデータ・ファイルに、最初のレコードのデータに対応するフィールド名のリストが含まれるが、その情報は無視する必要があることを示します。このレコードは、すべてのデータ・ファイルでデータが処理されるとスキップされますが、フィールドの設定には使用されません。
- NONE—データ・ファイルに、最初のレコードの通常のデータが含まれることを示します。このオプションがデフォルトです。

用途

FIELD NAMES句を使用すると、いずれかのオプションを使用して、データ・ファイルの最初の行のデータ・ファイルのフィールド順序を指定できます。たとえば、FIELD NAMES FIRST FILEが指定されている場合、最初のデータ・ファイルのみが行ヘッ

ダーを持ちます。FIELD NAMES ALL FILESが指定されている場合、すべてのデータ・ファイルに行ヘッダーが含まれます。

制限事項

- FIELD NAMES句では、データ・ファイル内のフィールド名の間の空白は切り捨てられません。

  たとえば、データ・ファイルにフィールド名deptno, dname, LOC (フィールド名の間に空白を含む)がある場合、FIELD NAMESの指定に失敗し、「KUP-04117: フィールド名LOCがアクセス・パラメータ・フィールド・リストまたは表に見つかりませんでした。」と表示されます。

- データ・ファイルのフィールド名は、引用符を使用できません。たとえば、次の列のフィールド名はサポートされていません:

  deptno,"dname",loc

- 最初の列ヘッダー行では、埋込みデリミタはサポートされていません。

例

通常、列(COL1、COL2、COL3)を含む表を生成するデータ・ファイルのフィールドはデータ・ファイル内にあり、表と同じ順序です。ただし、次の例では、deptxt1.datおよびdeptxt2.datのデータ・ファイル・フィールドの順序は異なります。FIELD NAMES ALL FILESを指定すると、1つ以上のデータ・ファイルのフィールド名の順序が異なるデータ・フィールドを正しく問い合せることができます:

```
[admin@example]$ cat /tmp/deptxt1.dat
deptno,dname,loc
10,ACCOUNTING,NEW YORK
20,RESEARCH,DALLAS
30,SALES,CHICAGO
40,OPERATIONS,BOSTON
[admin@example]$ cat /tmp/deptxt2.dat
dNamE,lOc,DEPTNO
ACCOUNTING,NEW YORK,11
RESEARCH,DALLAS,21
SALES,CHICAGO,31
OPERATIONS,BOSTON,41
[admin@example]$ sql @xt
Connected.
Directory created.
SQL> create table deptXT
   2  (
   3      deptno    number(2),
   4      dname    varchar2(14),
   5      loc    varchar2(13)
   6  )
   7  organization external
   8  (
   9    type ORACLE_LOADER
  10    DEFAULT DIRECTORY DATA_DIR
  11    access parameters
  12    (
  13       records delimited by newline
  14       field names all files
  15       logfile 'deptxt.log'
  16       badfile 'deptxt.bad'
  17       fields terminated by ','
  18       missing field values are null
  19    )
  20    location ('deptxt?.dat')
  21  )
  22  reject limit unlimited
  23  ;
Table created.
SQL> Rem returns all 8 rows
SQL> select deptno, dname, loc from deptxt order by deptno;
```

```
DEPTNO     DNAME          LOC
---------- -------------- -------------
     10    ACCOUNTING     NEW YORK
     11    ACCOUNTING     NEW YORK
     20    RESEARCH       DALLAS
     21    RESEARCH       DALLAS
     30    SALES          CHICAGO
     31    SALES          CHICAGO
     40    OPERATIONS     BOSTON
     41    OPERATIONS     BOSTON
```

親トピック: record_format_info句

15.3.20 READSIZE

READSIZEパラメータは、レコードの処理に使用する読取りバッファのサイズを指定します。

読取りバッファのサイズは、アクセス・ドライバで処理する最大入力レコードのサイズ以上にする必要があります。サイズは、整数のバイト数で指定します。デフォルト値は512KB(524288バイト)です。データ・ファイル内に512KBより大きいレコードがある場合は、デフォルト値より大きい値を指定する必要があります。READSIZEのサイズに上限はありませんが、アクセス・ドライバで割当て可能なメモリーの最大量が事実上の上限となります。

追加のバッファが割り当てられる場合もあるため、割当てに使用可能なメモリーの量によっても制限されます。追加のバッファは、(データ内、デリミタ、またはマルチ・キャラクタ/バイトのデリミタが使用されている場合はそのデリミタのいずれかで)分割された可能性のあるレコードの処理を正常に完了するために使用されます。

親トピック: record_format_info句

15.3.21 DISABLE_DIRECTORY_LINK_CHECK

DISABLE_DIRECTORY_LINK_CHECKパラメータは、ORACLE_LOADERアクセス・ドライバにシンボリック・リンク・チェックをバイパスするように指示します。

デフォルトでは、ORACLE_LOADERアクセス・ドライバは、データやログ・ファイルを開く前にチェックを行い、使用するディレクトリがシンボリック・リンクでないことを確認します。DISABLE_DIRECTORY_LINK_CHECKパラメータ(引数なし)を指定して、このチェックを回避するようにアクセス・ドライバに指示すると、親ディレクトリがシンボリック・リンクの可能性があるファイルも使用できます。

> ノート:
>
> シンボリック・リンクは外部表のロード操作での入力や出力をリダイレクトするために使用される可能性もあるため、このパラメータの使用にはセキュリティのリスクが伴います。

親トピック: record_format_info句

15.3.22 DATE_CACHE

デフォルトでは、(1000要素に対して)日付キャッシュ機能が使用できます。日付キャッシュ機能を完全に使用禁止にするには、0(ゼロ)に設定します。

DATE_CACHEは、日付キャッシュ・サイズ(エントリ数)を指定します。たとえば、DATE_CACHE=5000を指定すると、作成された日付キャッシュごとに最大5000の一意の日付エントリが含まれます。必要に応じて、すべての表に固有の日付キャッシュが作成されます。日付キャッシュは、表への格納のためにデータ型変換が必要な日付値またはタイムスタンプ値が1つ以上ロードされ

た場合にのみ作成されます。

日付キャッシュ機能は、デフォルトで使用可能です。デフォルトの日付キャッシュ・サイズは1000要素です。デフォルトのサイズを使用し、1000を超える一意の入力値をロードすると、日付キャッシュ機能は、この表に対して自動的に使用禁止となります。ただし、デフォルトを変更して0以外の日付キャッシュ・サイズを指定し、キャッシュ量がこのサイズを超えた場合、キャッシュは使用禁止になりません。

ログ・ファイルに含まれている日付キャッシュ統計(エントリ数、ヒット数、ミス数)を使用して、将来、同様のロードを行うときのためにキャッシュのサイズを調整できます。

関連項目:

日付キャッシュの値の指定

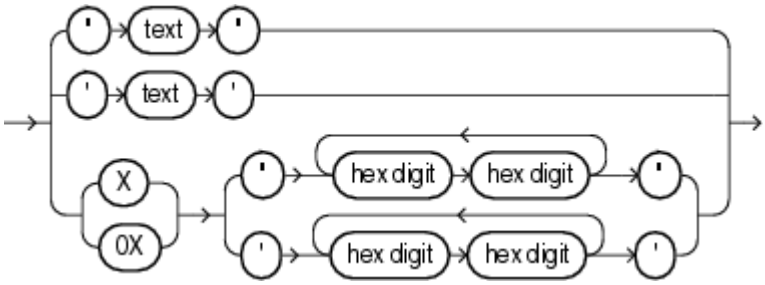
親トピック: record_format_info句

15.3.23 string

stringは、引用符で囲まれた一連の文字または16進数字です。

構文

`string`の構文は次のとおりです。



用途

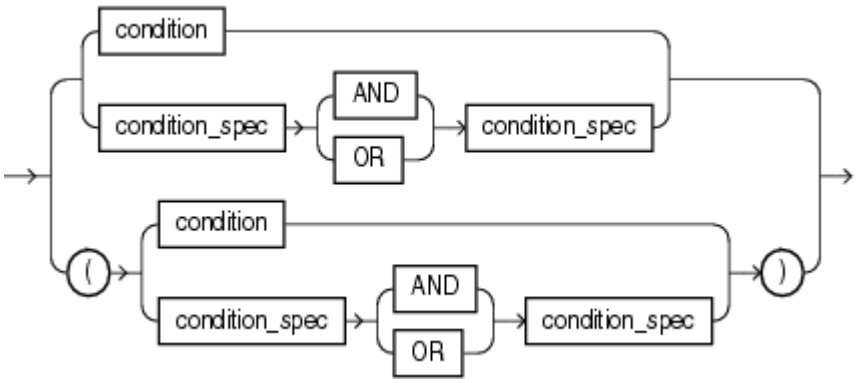
一連の文字の場合、これらの文字はデータ・ファイルの文字セットに変換されます。また、16進数字の場合には、偶数にする必要があります。16進数字は、バイナリに翻訳されたものに変換され、その翻訳結果はデータ・ファイルの文字セット内の文字列として処理されます。これは、16進数字がバイナリに翻訳されたものに変換された後では、他の文字セットの翻訳が実行されないことを意味します。

親トピック: record_format_info句

15.3.24 condition_spec

`condition_spec`では、ブール演算子で結合される1つ以上の条件を指定します。

この句は、真または偽のいずれかに評価される式です。条件およびブール演算子は、左から右へと評価されます。(ブール演算子は、条件が評価された後に適用されます。)ブール演算子のデフォルトの評価順序を上書きするには、カッコを使用します。`condition_spec`句の評価にはより多くのレコード処理時間が必要であるため、多くの句を使用しないようにする必要があります。`condition_spec`の構文は次のとおりです。



条件指定にフィールド名を参照する条件が含まれている場合、条件指定は、すべてのフィールドがレコードで検出され、空白の切捨てが行われた後のみに評価されます。空白がフィールドから切り捨てられている場合、フィールドとBLANKSの比較は有効ではありません。

次に、condition_specの使用例を示します。

```
empid = BLANKS OR last_name = BLANKS
(dept_id = SPORTING GOODS OR dept_id = SHOES) AND total_sales != 0
```

関連項目:

condition

親トピック: record_format_info句

15.3.25 [directory object name:] [filename]

[directory object name:] [filename]句を使用して、出力ファイルの名前(BADFILE、DISCARDFILEまたはLOGFILE)を指定します。

構文

```
[directory object name:] [filename]
```

- directory object name: ファイルの読取りおよび書込みのためのデータベース・サーバー上のオペレーティング・システム・ディレクトリの別名。
- filename: ディレクトリ・オブジェクト内に作成するファイルの名前。

パラレル・ロードでファイル名を一意にするために、アクセス・ドライバは記号置換を行います。Linux、UNIXおよびMicrosoft Windowsオペレーティング・システムでサポートされる記号置換は、次のとおりです(その他のプラットフォームでは、別の記号が使用される場合があります)。

- %pは、現行のプロセスのプロセスIDに置換されます。

  たとえば、アクセス・ドライバのプロセスIDが12345の場合、exttab_%p.logとして指定されたファイル名はexttab_12345.logとなります。

- %aは、現行のプロセスのエージェント番号に置換されます。エージェント番号は、外部表にアクセスしている各パラレル・プロセスに割り当てられた一意の番号です。この番号には、3文字になるように、左側に0が埋められます。

  たとえば、3番目のパラレル・エージェントがファイルを作成する場合、bad_data_%a.badをファイル名として指定した場合、エージェントはbad_data_003.badというファイルを作成します。

- %%は、%に置換されます。ファイル名にパーセント符号が必要な場合、この記号置換が使用されます。

%文字が検出され、前述の文字以外の文字がその後に続く場合、エラーが返されます。

用途

出力ファイルの名前(BADFILE、DISCARDFILEまたはLOGFILE)を指定します。

使用上のノート

この句を使用するには、ディレクトリ・オブジェクト名またはファイル名(あるいはその両方)を指定する必要があります。そのディレクトリ・オブジェクト名は、外部表にアクセスしているユーザーが書込み権限を所有しているディレクトリ・オブジェクトの名前です。このディレクトリ・オブジェクト名を指定しない場合、CREATE TABLE...ORGANIZATION EXTERNAL文のDEFAULT DIRECTORY句に対して指定した値が使用されます。

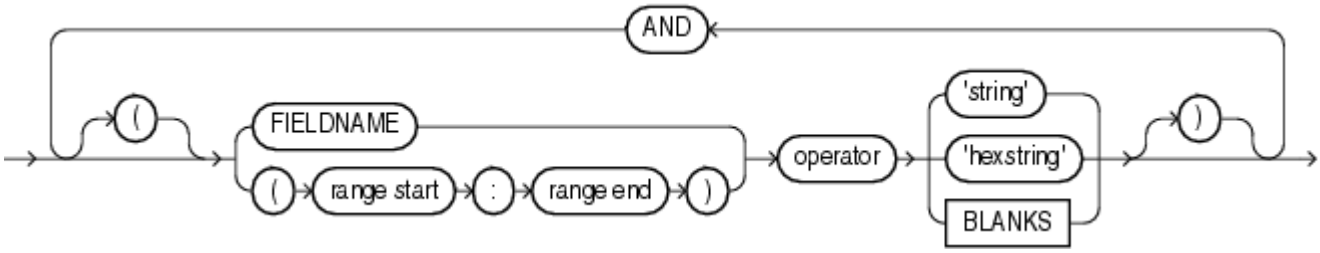
%pまたは%aを使用しないで出力ファイルに対して一意のファイル名を作成し、外部表にパラレルでアクセス中の場合、出力ファイルが破損するか、エージェントがファイルに書込みできないことがあります。

BADFILE (あるいはDISCARDFILEまたはLOGFILE)を指定しない場合、SQL_LOADERアクセス・ドライバでは、表の名前に_%pを付けたものがファイル名として使用されます。ファイルに拡張子がない場合は、デフォルトの拡張子が使用されます。デフォルトの拡張子は、不良ファイルでは.bad、廃棄ファイルでは.dsc、ログ・ファイルでは.log となります。

親トピック: record_format_info句

15.3.26 condition

conditionを使用して、定数文字列とレコードのバイト範囲またはフィールドを比較します。比較のソースは、レコードのフィールドまたはレコードのバイト範囲のいずれかです。比較はバイト単位で実行されます。文字列を比較のターゲットとして指定すると、データ・ファイルの文字セットに変換されます。フィールドに文字以外のデータ型が含まれる場合、フィールド値または文字列のどちらでもデータ型変換は実行されません。conditionの構文は次のとおりです。



- range start : range end
  range start:range end句は、レコードに含まれるバイト数または文字数の範囲を記述して条件に使用します。

親トピック: record_format_info句

15.3.26.1 range start : range end

range start:range end句は、レコードに含まれるバイト数または文字数の範囲を記述して条件に使用します。

STRING SIZES ARE句に入力した値によって、rangeがバイト数を示すか、文字数を示すかを決定します。range startおよびrange endは、レコードへのバイト・オフセットまたは文字オフセットです。range startは、range end以下である必要があります。文字範囲の検索は、可変幅文字セットのデータに対してより固定幅文字セットのデータに対しての方が速く処理されます。範囲が、存在しないレコードの一部を指す場合、その範囲を参照しようとするとレコードは拒否されます。range start:range endは、カッコで囲む必要があります。たとえば: (10:13)。

ノート:

データ・ファイルには、バイナリ・データ(VARCHAR などのバイナリ・カウントのあるデータ型を含む)および文字データ(可変幅文字セットが使用されているか、または文字幅が 1 バイトより大きいデータ)が混在しないようにする必要があります。この場合、アクセス・ドライバは、開始位置の検索時にバイナリ・データを文字データとして処理するため、フィールドの適切な開始位置を検索できない場合があります。

次に、conditionの使用例を示します。

```
LOAD WHEN empid != BLANKS
LOAD WHEN (10:13) = 0x'00000830'
LOAD WHEN PRODUCT_COUNT = "MISSING"
```

親トピック: 条件

15.3.27 IO_OPTIONS句

IO_OPTIONS句によって、オペレーティング・システムがデータ・ファイルを読み取るために使用するI/Oオプションを指定できます。指定可能なオプションはDIRECTIO (デフォルト)とNODIRECTIOのみです。

DIRECTIOオプションはデフォルトで使用され、直接I/Oを使用してデータ・ファイルのオープンおよび読取りが試行されます。試行が成功した場合、オペレーティング・システムとNFSサーバー(ファイルがNFSサーバー上にあるとき)では、ファイルから読み取ったデータをキャッシュしません。これにより、特にファイルが大きい場合に、データ・ファイルの読取りパフォーマンスが向上します。読取り対象のデータ・ファイルについて直接I/Oがサポートされない場合、ファイルは開かれて読み取られますがDIRECTIOオプションは無視されます。

IO_OPTIONS句がNODIRECTIOオプションを使用して指定されている場合、データ・ファイルの読取りに直接I/Oは使用されません。

IO_OPTIONS句がまったく指定されていない場合、デフォルトのDIRECTIOオプションが使用されます。

親トピック: record_format_info句

15.3.28 DNFS_DISABLE | DNFS_ENABLE

これらのパラメータを使用して、外部表操作中の入力データ・ファイルに対するDirect NFSクライアントの使用を有効化および無効化できます。

Direct NFSクライアントは、Oracleがファイル・サーバー上のファイルにアクセスする際のパフォーマンスを向上するために、ファイル・サーバーで実装できるAPIです。

外部表では、1GBを超えるデータ・ファイルを読み取る場合にデフォルトでDirect NFSクライアント・インタフェースが使用されます。よりサイズの小さいファイルでは、オペレーティング・システムのI/Oインタフェースが使用されます。すべての入力データ・ファイルでDirect NFSクライアントを使用するには、DNFS_ENABLEを使用します。

すべてのデータ・ファイルでDirect NFSクライアントの使用を無効にするには、DNFS_DISABLEを指定します。

関連項目:

- NFSでDirect NFSクライアントのOracle Disk Manager Controlを有効化する方法の詳細は、『Oracle Grid Infrastructureインストレーション・ガイドfor Linux』を参照してください。

親トピック: record_format_info句

15.3.29 DNFS_READBUFFERS

record_format_info句のDNFS_READBUFFERSパラメータを使用して、Direct NFSクライアントによって使用される読取りバッファの数を制御します。

DNFS_READBUFFERSを使用して、Direct NFSクライアントによって使用される読取りバッファの数を制御できます。Direct NFSクライアントは、Oracleがファイル・サーバー上のファイルにアクセスする際のパフォーマンスを向上するために、ファイル・サーバーで実装できるAPIです。

DNFS_READBUFFERSのデフォルト値は4です。

より大きい値を使用すると、Direct NFSクライアント・ファイル・サーバーからの一貫性のないI/Oに対処できますが、メモリー使用量が増大する可能性があります。

関連項目:

- NFSでDirect NFSクライアントのOracle Disk Manager Controlを有効化する方法の詳細は、『Oracle Grid Infrastructureインストレーション・ガイドfor Linux』を参照してください。

親トピック: record_format_info句

15.4 field_definitions句

field_definitions句では、FIELDSパラメータを使用してデータ・ファイルのフィールドに名前を付け、それらのレコード内での検索方法を指定します。

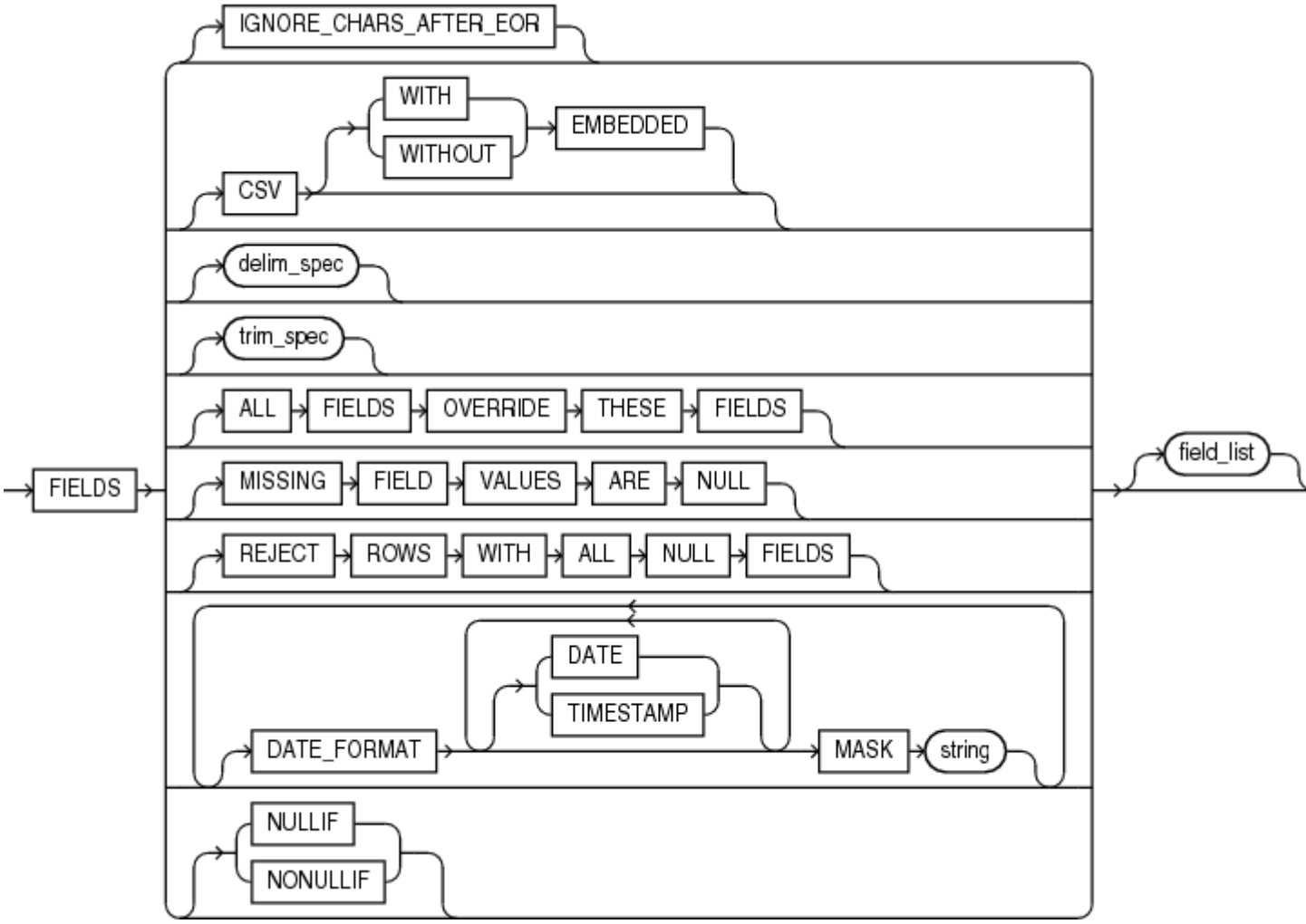
field_definitions句が省略された場合、次のことが想定されます。

- フィールドは「,」で区切られる
- フィールドのデータ型はCHARである
- フィールドの最大長は255である
- データ・ファイルのフィールドの順序は、外部表で定義されたフィールドの順序となる
- 空白はフィールドから切り捨てられない。

次に、アクセス・パラメータを含まずに作成する外部表の例を示します。その後に、外部表をロードするために使用できるデータ・ファイルのサンプルinfo.datを示します。

```
CREATE TABLE emp_load (first_name CHAR(15), last_name CHAR(20), year_of_birth
CHAR(4))
 ORGANIZATION EXTERNAL (TYPE ORACLE_LOADER DEFAULT DIRECTORY ext_tab_dir LOCATION
('info.dat'));
Alvin,Tolliver,1976
Kenneth,Baer,1963
```

field_definitions句の構文は次のとおりです。



IGNORE_CHARS_AFTER_EOR

オプションのこのパラメータは、最後のエンドオブレコードの後、かつエンドオブファイルより前に、レコードの定義に一致しない不要な文字が見つかった場合、その文字を無視するように指定します。

次の4つの条件がすべて当てはまると、エラー・メッセージが外部表のログ・ファイルに書き込まれます。

- IGNORE_CHARS_AFTER_EORパラメータが設定されているか、フィールドが自由書式可能になっている(自由書式可能とは、フィールドが可変長であるか、フィールドがデリミタまたは囲み文字で指定されていて可変長であるかのいずれかです)。
- 文字がファイル内の最後のエンドオブレコードの後にある。
- アクセス・パラメータMISSING FIELD VALUES ARE NULLが設定されていない。
- フィールドに絶対位置が設定されていない。

外部表のログ・ファイルに書き込まれるエラー・メッセージは次のとおりです。

```
KUP-04021: field formatting error for field Col1
KUP-04023: field start is after end of record
KUP-04101: record 2 rejected in file /home/oracle/datafiles/example.dat
```

CSV

外部表でカンマ区切り値形式のファイルであるデータ・ファイルにアクセスするには、FIELDS CSV句を使用します。これによって、ファイルは、通常の改行文字列(たとえば、UNIXまたはLinuxオペレーティング・システムの場合は¥n、Windowsオペレーティング・システムの場合は¥nまたは¥r¥n)を含むストリーム・レコード形式のファイルであるとみなされます。データ値にはレコード終

了記号含める(埋め込む)ことができます。FIELDS CSV句の構文は次のとおりです。

```
FIELDS CSV [WITH EMBEDDED | WITHOUT EMBEDDED] [TERMINATED BY ','] [OPTIONALLY
ENCLOSED BY '"']
```

FIELDS CSV句の主なポイントは次のとおりです。

- デフォルトでは、FIELDS CSV句を使用しません。
- WITH EMBEDDEDおよびWITHOUT EMBEDDEDオプションでは、データにレコード終了記号が含まれている(埋め込まれている)かどうかを指定します。WITH EMBEDDEDオプションがデフォルトです。
- WITH EMBEDDEDを使用する場合、埋込みのレコード終了記号は囲む必要があり、データ・ファイル内の並列度は外部表ロードでは無効になります。
- TERMINATED BY ','およびOPTIONALLY ENCLOSED BY '"'オプションはデフォルトのため、指定する必要はありません。これらは別の終了記号や囲み文字で上書きできます。
- CSV句を使用する場合、フィールド・レベルでのデリミタ指定は許可されず、区切ることが可能なデータ型のみ許可されます。区切ることが可能なデータ型には、CHAR、日時、期間および数値型EXTERNALが含まれます。
- CSV句を指定する場合、TERMINATED BYおよびENCLOSED BY句は、フィールド・レベルでは使用できません。
- CSV句を指定する場合、デフォルトの切捨て動作はLDRTRIMです。これを上書きするには、外部表の他の切捨てオプション(NOTRIM、LRTRIM、LTRIMまたはRTRIM)のいずれかを指定します。
- CSV句は、IGNORE_CHARS_AFTER_EOR句の後で、delim_spec句の前に指定する必要があります。

delim_spec句

delim_spec句を使用して、レコード内のすべてのフィールドの終了位置を識別します。すべてのフィールドに指定されるdelim_specは、特定のフィールドに対してfield_list句の一部として上書きできます。構文の詳細は、「delim_spec」を参照してください。

trim_spec句

trim_spec句を使用して、すべての文字フィールドでデフォルトとして実行される空白の切捨てタイプを指定します。すべてのフィールドに指定されるtrim_spec句は、個々のフィールドに対してtrim_spec句を指定して上書きできます。構文の詳細は、「trim_spec」を参照してください。

ALL FIELDS OVERRIDE

ALL FIELDS OVERRIDE句では、アクセス・ドライバに対し、すべてのフィールドが存在し、それらが外部表の列と同じ順序であることを示します。特殊な定義が含まれるフィールドを指定するだけで済みます。この句は、オプションのtrim_spec句の後で、オプションのMISSING FIELD VALUES ARE NULL句の前に指定する必要があります。

次に、ALL FIELDS OVERRIDE句の使用例を示します。指定する必要のあるフィールドはHIREDATEのみで、これは日付書式マスクを必要とします。他のすべてのフィールドはデフォルト値を使用します。

```
FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY '"' LDRTRIM
ALL FIELDS OVERRIDE
REJECT ROWS WITH ALL NULL FIELDS
(
 HIREDATE CHAR(20) DATE_FORMAT DATE MASK "DD-Month-YYYY"
)
```

MISSING FIELD VALUES ARE NULL

MISSING FIELD VALUES ARE NULLは、位置が明示的に示されておらず、入力するのに十分なデータがないフィールドをNULLに設定します。詳細は、「MISSING FIELD VALUES ARE NULL」を参照してください。

REJECT ROWS WITH ALL NULL FIELDS

REJECT ROWS WITH ALL NULL FIELDSは、行内で参照されるすべてのフィールドがNULLの場合、その行が外部表にロードされないことを示します。このパラメータを指定しない場合、デフォルト値が使用され、すべてのフィールドがNULLの行が外部表にロードされます。このパラメータの設定は、「reject rows with all null fields」または「rows with all null fields are accepted」としてログ・ファイルに書き込まれます。

DATE_FORMAT

DATE_FORMAT句では、フィールド・レベルで日時書式マスクを一度指定することで、指定された独自のマスクのないそのタイプのすべてのフィールドにそれを適用できます。日時書式マスクは、オプションのREJECT ROWS WITH ALL NULL FIELDS句の後で、fields_list句の前に指定する必要があります。

DATE_FORMATは、日時データ型のDATE、TIME、TIME WITH TIME ZONE、TIMESTAMPおよびTIMESTAMP WITH TIME ZONEで指定できます。

次に、任意のDATE型フィールドにDD-Month-YYYYの日付マスクを適用するDATE_FORMAT句の使用例を示します。

```
FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY '"' LDRTRIM
REJECT ROWS WITH ALL NULL FIELDS
DATE_FORMAT DATE MASK "DD-Month-YYYY"

    (
       EMPNO,
       ENAME,
       JOB,
       MGR,
       HIREDATE CHAR(20),
       SAL,
       COMM,
       DEPTNO,
       PROJNO,
       ENTRYDATE CHAR(20)
    )
```

NULLIF | NO NULLIF

NULLIF句は、すべての文字フィールド(CHAR、VARCHAR、VARCHARC、外部NUMBER、日時など)に適用されます。

構文は次のとおりです。

```
NULLIF {=|!=}{"char_string"|x'hex_string'|BLANKS}
```

フィールドに対する等号または不等号の指定を使用して一致した場合、フィールドはその行でNULLに設定されます。

char_stringおよびhex_stringは、一重引用符または二重引用符で囲む必要があります。

NULLIFは、フィールド・レベルで指定されると、このNULLIF句に優先します。

NULLIF句を適用しないフィールドがある場合、フィールド・レベルでNO NULLIFを指定できます(次の例を参照)。

NULLIF句は、オプションのREJECT ROWS WITH ALL NULL FIELDS句の後で、fields_list句の前に指定する必要があります。

次に、NULLIF句の使用例を示します。MGRフィールドは、NULLIF="NONE"句をそのフィールドに適用しないようにするNO NULLIFに設定されます。

```
FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY '"' LDRTRIM
REJECT ROWS WITH ALL NULL FIELDS
NULLIF = "NONE"
(
  EMPNO,
  ENAME,
  JOB,
  MGR
 )
```

field_list句

field_list句を使用して、データ・ファイルのフィールドおよびそのデータ型を識別します。構文の詳細は、「field_list」を参照してください。

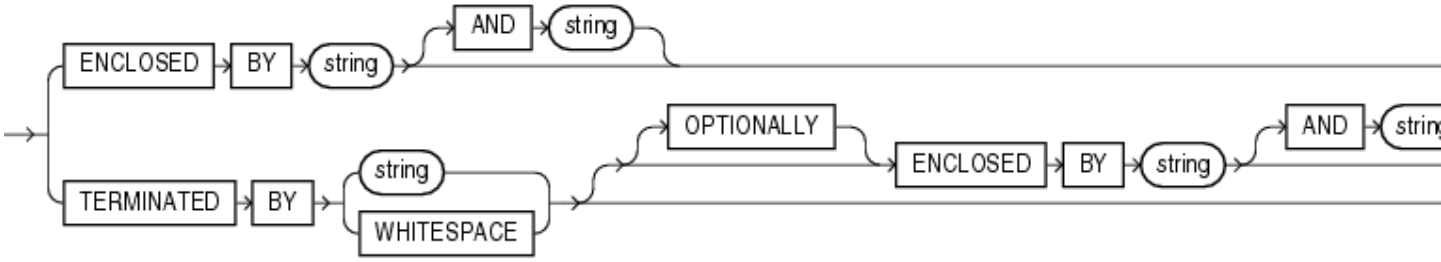
- delim_spec
  delim_spec句を使用して、フィールドの終了位置(ENCLOSED BYを指定する場合は、開始位置)を検索します。
- trim_spec
  trim_spec句を使用して、空白をテキスト・フィールドの始めから切り捨てるか、終わりから切り捨てるか、またはその両方から切り捨てるかを指定します。
- MISSING FIELD VALUES ARE NULL
  MISSING FIELD VALUES ARE NULLの影響は、フィールド位置を明示的に示すためにPOSITIONが使用されているかどうかによって異なります。
- field_list
  field_definitions field_list句は、データ・ファイル内のフィールドとそのデータ型を識別します。
- pos_spec句
  pos_spec句を使用して、レコード内の列の位置を指定します。
- datatype_spec句の構文
  ORACLE_LOADER datatype_spec句の構文は次のとおりです。
- init_spec句
  外部表にinit_spec句を使用して、フィールドをNULLまたはデフォルト値に設定するタイミングを指定します。
- LLS句

親トピック: ORACLE_LOADERアクセス・ドライバ

15.4.1 delim_spec

delim_spec句を使用して、フィールドの終了位置(ENCLOSED BYを指定する場合は、開始位置)を検索します。

構文



使用上のノート

ENCLOSED BYを指定すると、ORACLE_LOADERアクセス・ドライバで、レコードの現在の位置から最初のデリミタまでの間のすべての空白がスキップされます。現在の位置と最初のデリミタの間のすべての空白が無視されます。次に、アクセス・ドライバで

は、2番目の囲みデリミタが検索されます(または、2番目のデリミタが指定されていない場合は、最初のデリミタがもう1度検索されます)。これら2つのデリミタの間にあるすべての文字がフィールド部分とみなされます。

TERMINATED BY stringをENCLOSED BY句で指定する場合、終了記号文字列は、2番目の囲みデリミタの直後に置く必要があります。2番目の囲みデリミタと終了デリミタの間の空白はスキップされます。2つのデリミタの間で空白以外の文字が検索される場合、正しく書式設定されていないため行が拒否されます。

ENCLOSED BY句を使用せずにTERMINATED BYを指定する場合、レコードの現在の位置と次に検索される終了記号文字列の間にあるすべての文字がフィールド部分とみなされます。

OPTIONALLYを指定する場合は、TERMINATED BYも指定する必要があります。OPTIONALLYパラメータによって、ENCLOSED BYデリミタは、両方存在するかまたは両方存在しないかのいずれかであることが示されます。終了デリミタは、ENCLOSED BYデリミタの有無にかかわらず存在する必要があります。OPTIONALLYを指定する場合、アクセス・ドライバは、最初の空白以外の文字までのすべての空白をスキップします。最初の空白以外の文字が検索されると、アクセス・ドライバは、現在の位置に最初の囲みデリミタが含まれているかどうかを確認します。含まれている場合は、アクセス・ドライバによって2番目の囲み文字列が検索されます。最初の囲みデリミタと2番目の囲みデリミタの間のすべてが、フィールドの一部とみなされます。終了デリミタは、2番目の囲みデリミタの直後に置く必要があります(2番目の囲みデリミタと終了デリミタの間にオプションで空白を置くことも可能)。最初の空白以外の文字が最初の囲み文字列ではない場合、アクセス・ドライバは終了デリミタを検索します。この場合、先頭の空白は切り捨てられます。

デリミタが検出された後、レコードの現在の位置は、フィールドの最後のデリミタの後に設定されます。TERMINATED BY WHITESPACEを指定した場合、レコードの現在の位置は、フィールドの後に続くすべての空白の後に設定されます。

アクセス・ドライバのデフォルトの切捨て動作の詳細は、「空白の切捨て」を参照してください。LTRIMおよびRTRIMを使用すると、この動作をオーバーライドできます。

レコードの最後のフィールドで終了記号が欠落している場合は、エラーではありません。アクセス・ドライバは、終了記号が検出された場合と同様に処理を行います。2番目の囲みデリミタが欠落している場合は、エラーとなります。

2番目の囲みに使用される文字列は、2番目の囲みを2回続けることによって、データ・フィールドに含むことができます。たとえば、フィールドが一重引用符で囲まれる場合、次の例のdon'tという単語で示すような方法で一重引用符を2つ並べて指定することによって、フィールドに一重引用符を含むことができます。

```
'I don''t like green eggs and ham'
```

囲みデリミタを使用せずに、データ・フィールドの終了文字列を引用符で囲む方法はありません。フィールドに終了デリミタを含むことができるのは、フィールド・パーサーが囲みデリミタを検出するまで終了デリミタを検索しないためです。

通常、1文字の文字列は、複数文字の文字列より速く指定できます。また、固定幅文字セットのデータは、可変幅の文字セットよりも速く検索できます。

ノート:

外部表では、文字列内のバックスラッシュ(¥)の使用はサポートされていません。

- 例: 終了デリミタを含む外部表
  終了デリミタを使用する外部表、および終了デリミタがあるデータ・ファイルを作成する方法を説明します。
- 例: 囲みデリミタおよび終了デリミタを含む外部表
  囲みデリミタと終了デリミタを両方とも使用する外部表の作成方法を説明します。
- 例: オプションの囲みデリミタを含む外部表

オプションの囲みデリミタを使用する外部表の作成方法を説明します。

関連トピック

- 空白の切捨て

親トピック: field_definitions句

15.4.1.1 例: 終了デリミタを含む外部表

終了デリミタを使用する外部表、および終了デリミタがあるデータ・ファイルを作成する方法を説明します。

この表は、終了デリミタを使用するように作成されます。その後に、その表のロードに使用できるデータ・ファイルの例が示されます。

```
CREATE TABLE emp_load (first_name CHAR(15), last_name CHAR(20), year_of_birth
CHAR(4))
  ORGANIZATION EXTERNAL (TYPE ORACLE_LOADER DEFAULT DIRECTORY ext_tab_dir
                         ACCESS PARAMETERS (FIELDS TERMINATED BY WHITESPACE)
                         LOCATION ('info.dat'));
Alvin Tolliver 1976
Kenneth Baer 1963
Mary Dube 1973
```

親トピック: delim_spec

15.4.1.2 例: 囲みデリミタおよび終了デリミタを含む外部表

囲みデリミタと終了デリミタを両方とも使用する外部表の作成方法を説明します。

次に、囲みデリミタと終了デリミタの両方を使用する外部表の例を示します。2番目の囲みデリミタと終了記号の間のすべての空白が無視されるのと同様に、終了文字列と最初の囲み文字列の間のすべての空白も無視されます。この例の後に、ロードが可能なデータ・ファイルのサンプルを示します。

```
CREATE TABLE emp_load (first_name CHAR(15), last_name CHAR(20), year_of_birth
CHAR(4))
  ORGANIZATION EXTERNAL (TYPE ORACLE_LOADER DEFAULT DIRECTORY ext_tab_dir
                        ACCESS PARAMETERS (FIELDS TERMINATED BY "," ENCLOSED BY "("
AND ")")
                        LOCATION ('info.dat'));
(Alvin) ,   (Tolliver),(1976)
(Kenneth),  (Baer) ,(1963)
(Mary),(Dube) ,   (1973)
```

親トピック: delim_spec

15.4.1.3 例: オプションの囲みデリミタを含む外部表

オプションの囲みデリミタを使用する外部表の作成方法を説明します。

この表は、オプションの囲みデリミタを使用するように作成される外部表です。フィールドの先頭および後続の空白を切り捨てるために、LRTRIMを使用していることに注意してください。この例の後に、その表のロードに使用できるデータ・ファイルの例が示されます。

```
CREATE TABLE emp_load (first_name CHAR(15), last_name CHAR(20), year_of_birth
CHAR(4))
  ORGANIZATION EXTERNAL (TYPE ORACLE_LOADER DEFAULT DIRECTORY ext_tab_dir
                         ACCESS PARAMETERS (FIELDS TERMINATED BY ','
                                            OPTIONALLY ENCLOSED BY '(' and ')'
                                            LRTRIM)
                         LOCATION ('info.dat'));
Alvin ,   Tolliver , 1976
(Kenneth),  (Baer), (1963)
```

```
( Mary ), Dube ,    (1973)
```

親トピック: delim_spec

15.4.2 trim_spec

trim_spec句を使用して、空白をテキスト・フィールドの始めから切り捨てるか、終わりから切り捨てるか、またはその両方から切り捨てるかを指定します。

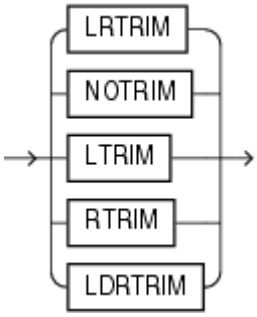
説明

テキスト・フィールドの先頭またはテキスト・フィールドの末尾、あるいはその両方の空白を切り捨てるように、ORACLE_LOADERアクセス・ドライバに指示します。空白には、空白文字およびその他の印字されない文字(タブ、行送り、改行など)が含まれます。

デフォルト

デフォルトは、LDRTRIMです。NOTRIMを指定すると、パフォーマンスが向上します。

構文



オプション

- NOTRIMは、フィールドから文字を切り捨てないように指定します。
- LRTRIMは、先頭と末尾のどちらの空白も切り捨てるように指定します。
- LTRIMは、先頭の空白を切り捨てるように指定します。
- RTRIMは、末尾の空白を切り捨てるように指定します。
- LDRTRIMは、SQL*Loaderの切捨て機能との互換性を提供します。次の場合を除いて、NOTRIMと同様です。
    - フィールドがデリミタ付きのフィールドではない場合、空白は右から切り捨てられる。
    - フィールドがOPTIONALLY ENCLOSED BYで指定されたデリミタ付きフィールドで、オプションの囲みが特定のインスタンスで欠落している場合、空白は左から切り捨てられます。

使用上のノート

trim_spec句をフィールド・リストの前に指定して、デフォルトの切捨てをすべてのフィールドに設定できます。trim_specがフィールド・リストの前で指定されない場合、LDRTRIMが、デフォルトの切捨て設定となります。デフォルトの切捨ては、個々のフィールドに対してdatatype_specの一部として上書きできます。

すべてが空白のフィールドに対して切捨てを指定する場合、そのフィールドはNULLに設定されます。

次の例では、すべてのデータが固定長です。ただし、先頭に空白がある文字データはロードできません。この例の後に、ロードが可能なデータ・ファイルのサンプルを示します。

```
CREATE TABLE emp_load (first_name CHAR(15), last_name CHAR(20),
year_of_birth CHAR(4))
  ORGANIZATION EXTERNAL (TYPE ORACLE_LOADER DEFAULT DIRECTORY ext_tab_dir
                         ACCESS PARAMETERS (FIELDS LTRIM)
                         LOCATION ('info.dat'));
Alvin,           Tolliver,1976
```

```
Kenneth,          Baer,     1963
Mary,             Dube,     1973
```

親トピック: field_definitions句

15.4.3 MISSINGFIELDVALUESARENULL

MISSING FIELD VALUES ARE NULLの影響は、フィールド位置を明示的に示すためにPOSITIONが使用されているかどうかによって異なります。

たとえば:

- デフォルトの動作では、フィールド位置が明示的に示されておらず、レコードのすべてのフィールドに十分なデータがない場合、レコードは拒否されます。この動作を変更するには、MISSING FIELD VALUES ARE NULLを使用して、使用可能なデータがないフィールドをNULLとして定義します。
- フィールド位置が明示的に示されている場合、MISSING FIELD VALUES ARE NULLが使用されているかどうかにかかわらず、常に、値のないフィールドはNULLとして定義されます。

次の例で、2番目のレコードは、生まれた年のデータがデータ・ファイルから欠落していても、year_of_birth列に対しNULLに設定されて格納されます。MISSING FIELD VALUES ARE NULL句をアクセス・パラメータで指定しない場合、year_of_birth列の値が含まれていない2番目のレコードが拒否されます。この例の後に、ロードが可能なデータ・ファイルのサンプルを示します。

```
CREATE TABLE emp_load (first_name CHAR(15), last_name CHAR(20), year_of_birth INT)
  ORGANIZATION EXTERNAL (TYPE ORACLE_LOADER DEFAULT DIRECTORY ext_tab_dir
                         ACCESS PARAMETERS (FIELDS TERMINATED BY ","
                                            MISSING FIELD VALUES ARE NULL)
                         LOCATION ('info.dat'));

Alvin,Tolliver,1976
Baer,Kenneth
Mary,Dube,1973
```

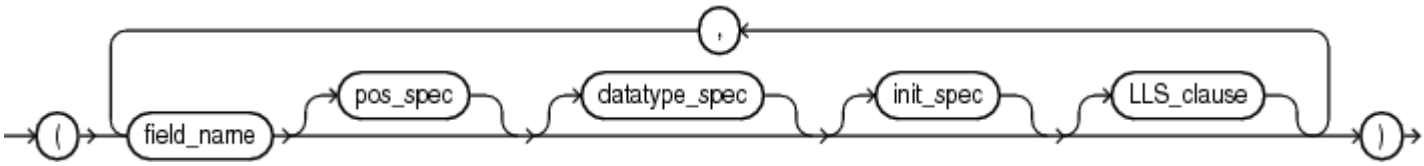
親トピック: field_definitions句

15.4.4 field_list

field_definitions field_list句を使用して、データ・ファイルのフィールドおよびそのデータ型を識別します。

構文

field_list句の構文は次のとおりです。



field list句は、次のとおりです。

- field_name: データファイルのフィールドの名前を識別する文字列です。文字列が引用符内にない場合、フィールド名は外部表の列名に一致され、大文字になります。

  field_nameが問合せで参照される外部表の列名と一致する場合は、このフィールド値が外部表列の値に使用されます。名前が外部表で参照されたいずれの名前にも一致しない場合、フィールドはロードされません。ただし、この

フィールドは句の評価(たとえば、WHENまたはNULLIF)には使用できます。

- pos_spec: レコード内の列の位置を示します。構文の詳細は、「pos_spec句」を参照してください。
- datatype_spec: フィールドのデータ型を示します。datatype_specが指定されない場合、アクセス・ドライバは、データ型はCHAR(255)であると想定します。構文の詳細は、「datatype_spec句」を参照してください。
- init_spec: フィールドがNULLであるか、デフォルト値を持っているかを示します。構文の詳細は、「init_spec句」を参照してください。
- LLS: フィールドにLLSを指定した場合、ORACLE_LOADERはフィールドの値を対応する列にロードしません。かわりに、値の情報を使用して、フィールドの値を見つける場所を判別します。「LLS句」を参照してください。

用途

field_list句を使用して、データ・ファイルのフィールドおよびそのデータ型を識別します。field_list句では、次のように評価します。

- フィールドにいずれのデータ型も指定されない場合、データ型は、デリミタなしフィールドではCHAR(1)、デリミタ付きフィールドではCHAR(255)である。
- いずれのフィールド・リストも指定されない場合、データ・ファイルのフィールドは外部表と同じ順序である。データベースの列がCHARまたはVARCHARではない場合、すべてのフィールドのデータ型は、CHAR(255)となる。データベースの列がCHARまたはVARCHARの場合、フィールドのデータ型はCHARのままであるが、その長さは255か列の長さでいずれか長い方の値となる。
- いずれのフィールド・リストも指定されず、delim_spec句も指定されない場合、データ・ファイルのフィールドは外部表のフィールドと同じ順序である。すべてのフィールドは、CHAR(255)であり、カンマで終了する。

例

次の例では、field_listおよびdelim_specを含まない外部表の定義を示します。その後に、ロードが可能なデータ・ファイルのサンプルを示します。

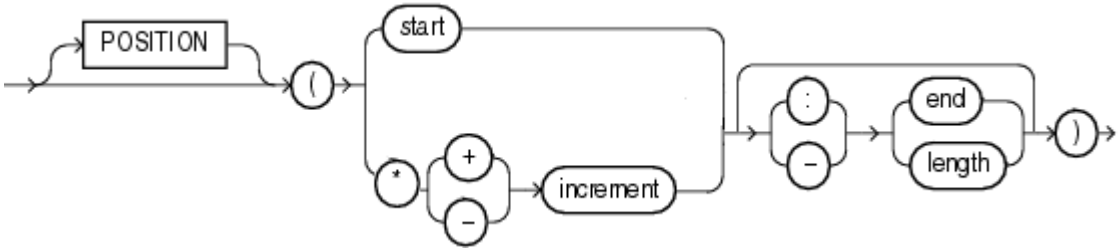
```
CREATE TABLE emp_load (first_name CHAR(15), last_name CHAR(20), year_of_birth INT)
  ORGANIZATION EXTERNAL (TYPE ORACLE_LOADER DEFAULT DIRECTORY ext_tab_dir
                         ACCESS PARAMETERS (FIELDS TERMINATED BY "|")
                         LOCATION ('info.dat'));
Alvin|Tolliver|1976
Kenneth|Baer|1963
Mary|Dube|1973
```

親トピック: field_definitions句

15.4.5 pos_spec句

pos_spec句を使用して、レコード内の列の位置を指定します。

STRING SIZES ARE IN句を設定して、pos_specがバイト位置と文字位置のどちらを参照するかを決定します。可変幅文字セットで文字位置を使用すると、固定幅文字セットで文字位置を使用するより大幅に時間がかかります。pos_specが文字位置に使用されると、バイナリ文字データとマルチバイト文字データは、同じデータ・ファイルには指定できません。指定した場合の結果は保証されません。pos_spec句の構文は次のとおりです。



- start
  pos_spec句のstartパラメータは、レコードの開始位置からフィールドの開始位置までのバイト数を示します。
- *
  pos_spec句の*パラメータは、フィールドが前のフィールドの直後のバイトから始まることを示します。
- increment
  pos_spec句のincrementパラメータは、フィールドの開始位置を前のフィールドの終了位置からの固定バイト数で設定します。
- end
  pos_spec句のendパラメータは、フィールドの最終バイトをレコード内の絶対バイト・オフセットで示します。
- length
  pos_spec句のlengthパラメータは、フィールドの終了位置を開始位置からの固定バイト数で示します。

親トピック: field_definitions句

15.4.5.1 start

pos_spec句のstartパラメータは、レコードの開始位置からフィールドの開始位置までのバイト数を示します。

startパラメータを使用すると、前のフィールドの位置からの相対ではなく、レコードの絶対位置でフィールドの開始位置を設定できます。

親トピック: pos_spec句

15.4.5.2 *

pos_spec句の*パラメータは、フィールドが前のフィールドの直後のバイトから始まることを示します。

*パラメータは、ORACLE_LOADERで可変長フィールドの後に固定長フィールドが続く場合に役立ちます。このオプションは、レコードの最初のフィールドには使用できません。

親トピック: pos_spec句

15.4.5.3 increment

pos_spec句のincrementパラメータは、フィールドの開始位置を前のフィールドの終了位置からの固定バイト数で設定します。

incrementパラメータは、フィールドの開始位置を前のフィールドの終了位置からの固定バイト数で設定します。*-incrementを使用して、フィールドの開始位置をレコードの現在の位置の前に指定します。現在の位置の後に開始を移動するには、*+incrementを使用します

親トピック: pos_spec句

15.4.5.4 end

pos_spec句のendパラメータは、フィールドの最終バイトをレコード内の絶対バイト・オフセットで示します。

endパラメータは、フィールドの最終バイトのレコード内での絶対バイト・オフセットを設定するために使用します。startをendとともに指定する場合、endは、startより小さくできません。*またはincrementをendとともに指定し、startが特定のレコードのendより大きいオフセットと評価された場合、レコードは拒否されます。

親トピック: pos_spec句

15.4.5.5 length

pos_spec句のlengthパラメータは、フィールドの終了位置を開始位置からの固定バイト数で示します。

lengthパラメータは、開始が*で指定されているときに、固定長フィールドを設定する場合に使用します。次に、pos_specの使用例を示します。その後で、ロードに使用できるデータ・ファイルのサンプルを示します。

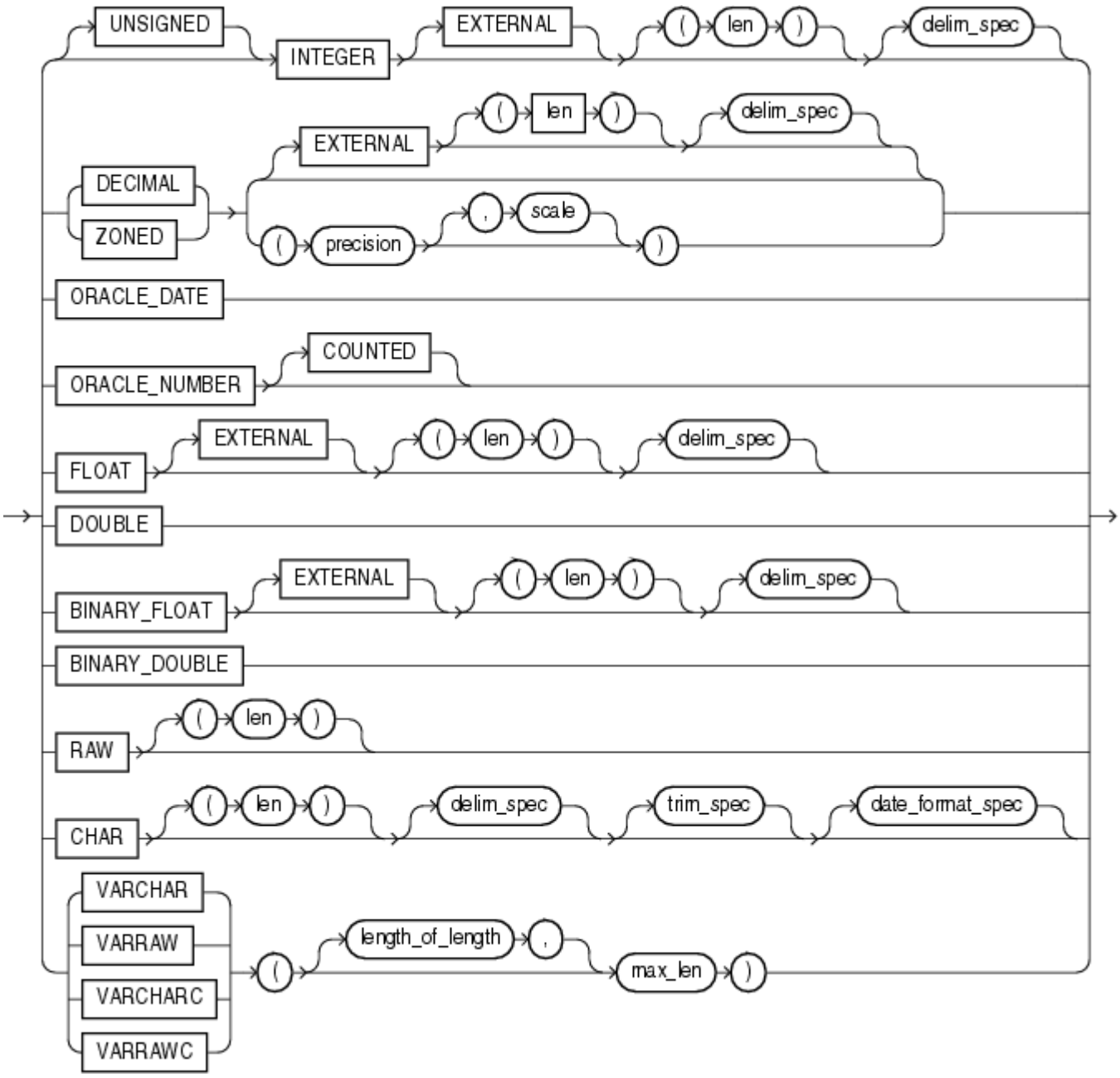
```
CREATE TABLE emp_load (first_name CHAR(15),
                       last_name CHAR(20),
                       year_of_birth INT,
                       phone CHAR(12),
                       area_code CHAR(3),
                       exchange CHAR(3),
                       extension CHAR(4))
  ORGANIZATION EXTERNAL
  (TYPE ORACLE_LOADER
   DEFAULT DIRECTORY ext_tab_dir
   ACCESS PARAMETERS
     (RECORDS CHARACTERSET we8iso8859p1
      FIELDS RTRIM
            (first_name (1:15) CHAR(15),
             last_name (*:+20),
             year_of_birth (36:39),
             phone (40:52),
             area_code (*-12: +3),
             exchange (*+1: +3),
             extension (*+1: +4)))
   LOCATION ('info.dat'));
Alvin          Tolliver            1976415-922-1982
Kenneth        Baer                1963212-341-7912
Mary           Dube                1973309-672-2341
```

この例では、宣言されたRECORDS CHARACTERSET we8iso8859p1はマルチバイト文字セットではありません。すべての文字が1バイトで表されることが保証されています。データ・フィールドの開始位置と終了位置(*演算子および+演算子を含む)を決定するためのPOSITION句の計算は、文字ではなくバイトに基づきます(つまり、Oracle文字セットWE8ISO8859P1のように、文字を表すには1バイトのみが必要です)。可変長文字セット(Unicodeバリアント、JIS X 0208-1990、またはフィールド・データに1つ以上のマルチバイト文字が含まれているその他のマルチバイト文字セットなど)を使用すると、計算は不正確になります。

親トピック: pos_spec句

15.4.6 datatype_spec句の構文

ORACLE_LOADER datatype_spec句の構文は次のとおりです。



フィールドのバイト数または文字数が0の場合、フィールドはNULLであると想定されます。オプションのDEFAULTIF句を使用して、フィールドをデフォルトの値に設定するタイミングを指定します。また、オプションのNULLIF句で、フィールドに対応付けられた列をNULLに設定するタイミングに関するその他の条件を指定します。DEFAULTIF句またはNULLIF句がTRUEの場合、これらの句を使用すると、データ・ファイルから読み込まれるすべての値が上書きされます。

- [UNSIGNED] INTEGER [EXTERNAL] [(len)]
  datatype_spec句[UNSIGNED] INTEGER [EXTERNAL] [(len)]では、フィールドが整数として定義されます。
- DECIMAL [EXTERNAL]およびZONED [EXTERNAL]
  DECIMAL句を使用して、フィールドがPACKED型の10進数であることを指定します。ZONED句を使用して、フィールドがZONED型の10進数であることを指定します。
- ORACLE_DATE
  ORACLE_DATEは、Oracleバイナリ日付形式の日付を格納しているフィールドです。
- ORACLE_NUMBER
  ORACLE_NUMBERは、Oracle数値形式の数値格納しているフィールドです。
- 浮動小数点数

DOUBLE、FLOAT、BINARY_DOUBLEおよびBINARY_FLOATの4つのデータ型は、浮動小数点数です。

- DOUBLE
  DOUBLE句では、フィールドがアクセス・ドライバを実行しているプラットフォームのC言語のDOUBLEデータ型と同じ形式であることを指定します。
- FLOAT [EXTERNAL]
  FLOAT句を使用して、フィールドが、アクセス・ドライバが実行されているプラットフォーム上のC言語のFLOATデータ型と同じ形式であることを指定します。
- BINARY_DOUBLE
  datatype_spec句の値BINARY_DOUBLEは、64ビットの倍精度浮動小数点数データ型です。
- BINARY_FLOAT
  datatype_spec句の値BINARY_FLOATは、32ビットの単精度浮動小数点数データ型です。
- RAW
  RAW句では、ソース・データがバイナリ・データであることを示します。
- CHAR
  datatype_spec句のデータ型CHAR句を使用して、フィールドが文字データ型であることを示します。
- date_format_spec
  date_format_spec句を使用して、特定の形式の日付データまたは時刻データ(またはその両方)が文字列フィールドに含まれることを指定します。
- VARCHARおよびVARRAW
  datatype_spec句のVARCHARデータ型は文字データを定義し、VARRAWデータ型はバイナリ・データを定義します。
- VARCHARCおよびVARRAWC
  datatype_spec句のVARCHARCデータ型は文字データを定義し、VARRAWCデータ型ではバイナリ・データを定義します。

関連トピック

- init_spec句
- Oracle Database SQL言語リファレンス

親トピック: field_definitions句

15.4.6.1 [UNSIGNED] INTEGER [EXTERNAL] [(len)]

datatype_spec句[UNSIGNED] INTEGER [EXTERNAL] [(len)]では、フィールドが整数として定義されます。

この句を使用して、フィールドを整数として定義します。EXTERNALを指定する場合、数値は文字列で指定します。EXTERNALを指定しない場合、数値はバイナリ・フィールドです。2進整数フィールドのlenに対する有効な値は、1、2、4および8です。lenが2進整数で指定されていない場合、デフォルトの値は、アクセス・ドライバが実行されているプラットフォーム上のsizeof(int)の値です。DATA IS {BIG|LITTLE} ENDIAN句を使用すると、データは格納される前にバイト・スワップされます。

EXTERNALを指定する場合、lenの値は、(STRING SIZES ARE IN BYTES句またはCHARACTERS句の設定に応じて)バイト数または文字数を数値で指定します。長さを指定しない場合、デフォルト値は255になります。

[UNSIGNED] INTEGER [EXTERNAL] [(len)]データ型のデフォルト値は、次のように決定されます。

- 長さを指定しない場合、デフォルトの長さは1になります。
- 長さを指定せず、フィールドがDELIMITED BY NEWLINE句で区切られている場合、デフォルトの長さは1になります。

- 長さを指定せず、フィールドがDELIMITED BY句で区切られている場合、デフォルトの長さは255になります(ただし、前述のとおりデリミタがNEWLINEでない場合)。

親トピック: datatype_spec句の構文

15.4.6.2 DECIMAL [EXTERNAL]およびZONED [EXTERNAL]

DECIMAL句を使用して、フィールドがPACKED型の10進数であることを指定します。ZONED句を使用して、フィールドがZONED型の10進数であることを指定します。

precisionフィールドで、数値の桁数を指定します。scaleフィールドで、数値の小数点の位置を指定します。つまり、小数点の右側にくる桁数を指定します。scaleを指定しない場合、値は0となります。

使用中の文字セットがEBCDICベースかASCIIベースかによって、ZONED型の10進数には異なるエンコーディング形式があることに注意してください。ソース・データの言語がEBCDICの場合、そのファイルのZONED型の10進数は、EBCDICエンコーディングと一致する必要があります。言語がASCIIベースの場合、その数値はASCIIエンコーディングと一致する必要があります。

EXTERNALパラメータを指定する場合、データ・フィールドは、その長さがフィールドの精度と一致する文字列です。

親トピック: datatype_spec句の構文

15.4.6.3 ORACLE_DATE

ORACLE_DATEは、Oracleバイナリ日付フォーマットの日付を含むフィールドであることを指定します。

これは、Oracle Call Interface (OCI)プログラムでは、DTYDATデータ型として使用される形式です。固定長7のフィールドです。

親トピック: datatype_spec句の構文

15.4.6.4 ORACLE_NUMBER

ORACLE_NUMBERは、Oracle数値書式の数値を含むフィールドであることを指定します。

COUNTEDを指定しないかぎり、フィールドは固定長(Oracle数値フィールドの最大サイズ)です。その場合、フィールドの最初のバイトには残りのフィールドのバイト数が含まれます。

ORACLE_NUMBERは、固定長22バイトのフィールドです。ORACLE_NUMBER COUNTEDフィールドの長さは、カウント・バイト用の1バイトに、カウント・バイトで指定されたバイト数を加えた長さです。

親トピック: datatype_spec句の構文

15.4.6.5 浮動小数点数

DOUBLE、FLOAT、BINARY_DOUBLEおよびBINARY_FLOATのデータ型は、浮動小数点数です。

DOUBLE、FLOAT、BINARY_DOUBLEおよびBINARY_FLOATのデータ型は、浮動小数点数です。

DOUBLEおよびFLOATは、プラットフォームで固有に使用される浮動小数点形式です。これらは、そのプラットフォームのCプログラムのDOUBLEおよびFLOATデータ型でデフォルトで使用されるものと同じデータ型です。BINARY_FLOATおよびBINARY_DOUBLEは浮動小数点数であり、実質的にInstitute for Electrical and Electronics Engineers (IEEE) Standard for Binary Floating-Point Arithmetic、IEEE規格754-1985に準拠しています。ほとんどのプラットフォームでは、固有の浮動小数点形式としてIEEE規格を使用しているため、FLOATとBINARY_FLOATは、プラットフォーム上では同様となります。また、DOUBLEとBINARY_DOUBLEも同様となります。

ノート:

浮動小数点数の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

親トピック: datatype_spec句の構文

15.4.6.6 DOUBLE

DOUBLE句を使用して、フィールドが、アクセス・ドライバが実行されているプラットフォーム上のC言語のDOUBLEデータ型と同じ形式であることを指定します。

DATA IS {BIG | LITTLE} ENDIAN句を使用すると、データは格納される前にバイト・スワップされます。このデータ型は特定のプラットフォーム間では移植できません。

親トピック: datatype_spec句の構文

15.4.6.7 FLOAT [EXTERNAL]

FLOAT句を使用して、フィールドが、アクセス・ドライバが実行されているプラットフォーム上のC言語のFLOATデータ型と同じ形式であることを指定します。

FLOAT句を使用して、フィールドが、アクセス・ドライバが実行されているプラットフォーム上のC言語のFLOATデータ型と同じ形式であることを指定します。DATA IS {BIG | LITTLE} ENDIAN句を使用すると、データは格納される前にバイト・スワップされます。このデータ型は特定のプラットフォーム間では移植できません。

EXTERNALパラメータを指定する場合、フィールドは、最大長255の文字列です。

親トピック: datatype_spec句の構文

15.4.6.8 BINARY_DOUBLE

datatype_spec句の値BINARY_DOUBLEは、64ビットの倍精度浮動小数点数データ型です。

各BINARY_DOUBLE値では、長さを示すバイトを含め9バイトを必要とします。浮動小数点数の詳細は、FLOATデータ型のノートに関する情報を参照してください。

親トピック: datatype_spec句の構文

15.4.6.9 BINARY_FLOAT

datatype_spec句の値BINARY_FLOATは、32ビットの単精度浮動小数点数データ型です。

各BINARY_FLOAT値では、長さを示すバイトを含め5バイトを必要とします。浮動小数点数の詳細は、FLOATデータ型のノートに関する情報を参照してください。

親トピック: datatype_spec句の構文

15.4.6.10 RAW

RAW句を使用して、ソース・データがバイナリ・データであることを指定します。

RAWフィールドに対するlenは常にバイト単位です。RAWフィールドがキャラクタ列にロードされると、列に書き込まれるデータは、RAWフィールドのバイトの16進表現となります。

親トピック: datatype_spec句の構文

15.4.6.11 CHAR

datatype_spec句のデータ型CHAR句を使用して、フィールドが文字データ型であることを指定します。

CHARフィールドの長さ(len)で、フィールドの最大バイト数または最大文字数を指定します。lenは、STRING SIZES ARE IN句の設定に応じて、バイト単位または文字単位になります。

CHARデータ型のフィールドに長さを指定しない場合、フィールドが区切られていないかぎり、フィールド・サイズは1になります。

- デリミタ付きCHARフィールドでは、長さが指定されている場合、その長さが最大長として使用されます。
- 長さが指定されていないデリミタ付きCHARフィールドでは、デフォルトの255バイトが使用されます。
- デリミタ付きで255バイトを超えるCHARフィールドには、最大長を指定する必要があります。指定しない場合は、データ・ファイルのフィールドが最大長を超えているというエラーを受信します。

次に、CHAR句の使用例を示します。

```
SQL> CREATE TABLE emp_load
  2    (employee_number      CHAR(5),
  3     employee_dob         CHAR(20),
  4     employee_last_name   CHAR(20),
  5     employee_first_name  CHAR(15),
  6     employee_middle_name CHAR(15),
  7     employee_hire_date   DATE)
  8  ORGANIZATION EXTERNAL
  9    (TYPE ORACLE_LOADER
 10     DEFAULT DIRECTORY def_dir1
 11     ACCESS PARAMETERS
 12       (RECORDS DELIMITED BY NEWLINE
 13        FIELDS (employee_number      CHAR(2),
 14                employee_dob         CHAR(20),
 15                employee_last_name   CHAR(18),
 16                employee_first_name  CHAR(11),
 17                employee_middle_name CHAR(11),
 18                employee_hire_date   CHAR(10) date_format DATE mask "mm/dd/yyyy"
 19               )
 20       )
 21     LOCATION ('info.dat')
 22    );

Table created.
```

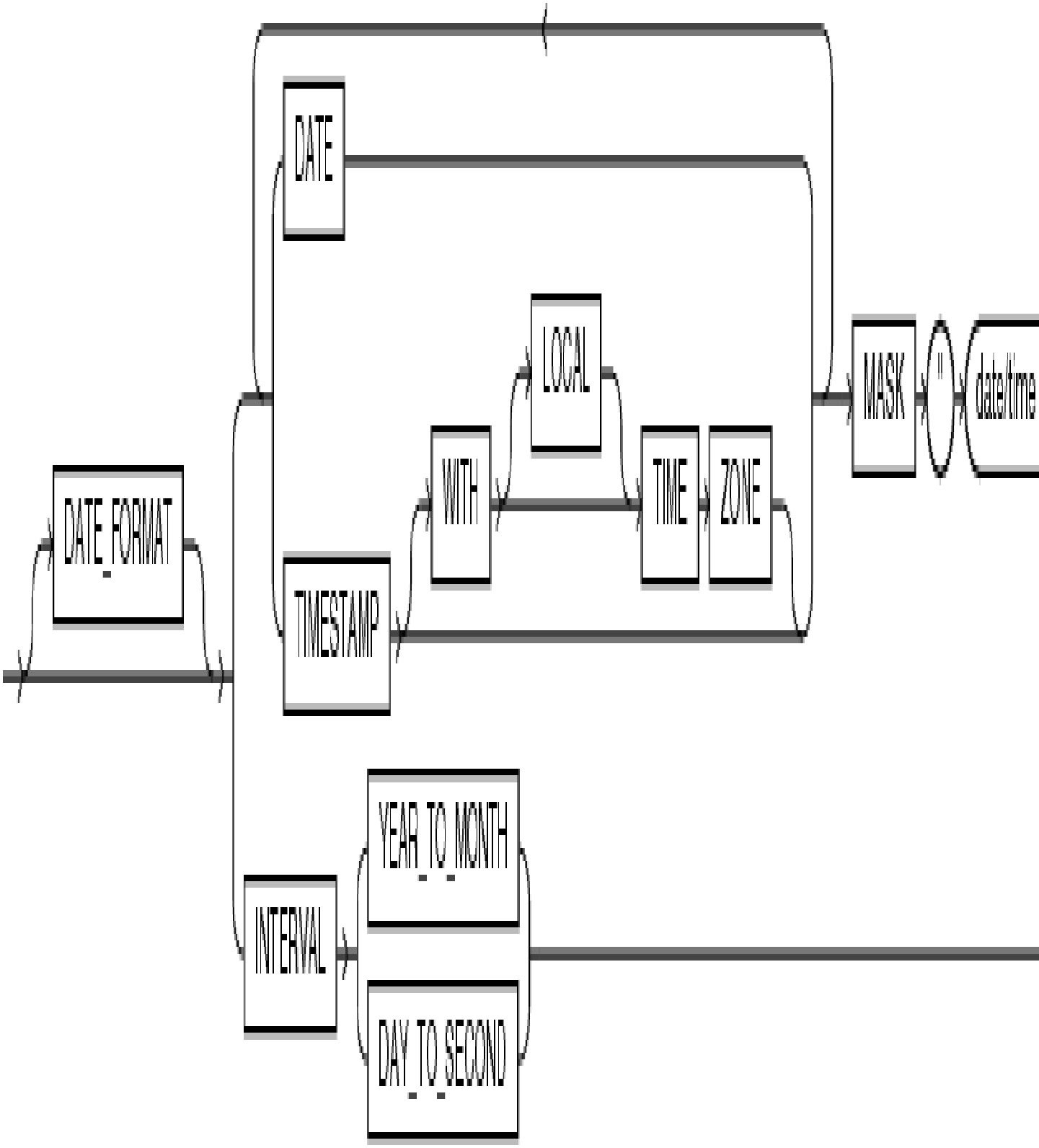
親トピック: datatype_spec句の構文

15.4.6.12 date_format_spec

date_format_spec句を使用して、特定の形式の日付データまたは時刻データ(またはその両方)が文字列フィールドに含まれることを指定します。

この情報は、文字フィールドが日付データ型または時刻データ型に変換される場合、および文字列フィールドが日付列にマップされる場合のみに使用されます。

date_format_spec句の構文は次のとおりです。



日付と時刻の書式を正しく指定する方法の詳細は、*Oracle Database SQL*リファレンスを参照してください。

- DATE
  DATE句では、文字列に日付が含まれることを示します。
- MASK
  MASK句は、データ型に対するデフォルトのグローバリゼーション形式のマスクをオーバーライドするために使用します。
- TIMESTAMP
  TIMESTAMP句では、フィールドに書式設定されたタイムスタンプが格納されていることを示します。

- INTERVAL
  INTERVAL句では、フィールドに書式設定された間隔が格納されていることを示します。

関連トピック

- Oracle Database SQL言語リファレンス

親トピック: datatype_spec句の構文

15.4.6.12.1 DATE

DATE句を使用して、文字列に日付が含まれることを指定します。

親トピック: date_format_spec

15.4.6.12.2 MASK

MASK句を使用して、データ型に対するデフォルトのグローバリゼーション書式マスクを上書きします。

日付マスクを指定しない場合は、データ型に対する適切なグローバリゼーション・パラメータのデータベースのNLSパラメータ設定(セッションの設定ではない)が使用されます。NLS_DATABASE_PARAMETERSビューに、これらの設定が表示されます。

- NLS_DATE_FORMAT (DATEデータ型)
- NLS_TIMESTAMP_FORMAT (TIMESTAMPデータ型)
- NLS_TIMESTAMP_TZ_FORMAT (TIMESTAMP WITH TIME ZONEデータ型)

次の点に注意してください。

- NLS_NUMERIC_CHARACTERS初期化パラメータのデータベース設定(NLS_DATABASE_PARAMETERSビューの設定)によって、文字データ型から数値データ型への暗黙的な変換に使用される小数点区切りが制御されます。
- デフォルトの書式では、桁区切りは使用できません。

親トピック: date_format_spec

15.4.6.12.3 TIMESTAMP

TIMESTAMP句を使用して、書式設定されたタイムスタンプがフィールドに含まれることを指定します。

親トピック: date_format_spec

15.4.6.12.4 INTERVAL

INTERVAL句を使用して、フィールドに書式設定された期間が含まれることを指定します。

INTERVAL句を使用して、フィールドに書式設定された期間が含まれることを指定します。期間の型は、YEAR TO MONTHまたはDAY TO SECONDのいずれかです。

次に、複雑なDATE文字列とTIMESTAMP文字列の使用例を示します。その後に、ロードが可能なデータ・ファイルのサンプルを示します。

```
SQL> CREATE TABLE emp_load
  2    (employee_number      CHAR(5),
  3     employee_dob         CHAR(20),
  4     employee_last_name   CHAR(20),
  5     employee_first_name  CHAR(15),
  6     employee_middle_name CHAR(15),
  7     employee_hire_date   DATE,
  8     rec_creation_date    TIMESTAMP WITH TIME ZONE)
  9  ORGANIZATION EXTERNAL
```

```
 10    (TYPE ORACLE_LOADER
 11     DEFAULT DIRECTORY def_dir1
 12     ACCESS PARAMETERS
 13       (RECORDS DELIMITED BY NEWLINE
 14        FIELDS (employee_number      CHAR(2),
 15                employee_dob         CHAR(20),
 16                employee_last_name   CHAR(18),
 17                employee_first_name  CHAR(11),
 18                employee_middle_name CHAR(11),
 19                employee_hire_date   CHAR(22) date_format DATE mask "mm/dd/yyyy
hh:mi:ss AM",
 20                rec_creation_date    CHAR(35) date_format TIMESTAMP WITH TIME ZONE
mask "DD-MON-RR HH.MI.SSXFF AM TZH:TZM"
 21               )
 22       )
 23     LOCATION ('infoc.dat')
 24    );

Table created.

SQL> SELECT * FROM emp_load;

EMPLO EMPLOYEE_DOB         EMPLOYEE_LAST_NAME   EMPLOYEE_FIRST_ EMPLOYEE_MIDDLE
----- -------------------- -------------------- --------------- ---------------
EMPLOYEE_
---------
REC_CREATION_DATE
---------------------------------------------------------------------------
56    november, 15, 1980   baker                mary            alice
01-SEP-04
01-DEC-04 11.22.03.034567 AM -08:00

87    december, 20, 1970   roper                lisa            marie
01-JAN-02
01-DEC-02 02.03.00.678573 AM -08:00


2 rows selected.
```

info.dat ファイルの内容は、次のようになります。これは、2つの長いレコードです。日付フィールド(09/01/2004、01/01/2002)とそれに続く時刻フィールドの間には、空白が1つあります。

```
56november, 15, 1980  baker             mary       alice      09/01/2004 08:23:01
AM01-DEC-04 11.22.03.034567 AM -08:00
87december, 20, 1970  roper             lisa       marie      01/01/2002 02:44:55
PM01-DEC-02 02.03.00.678573 AM -08:00
```

親トピック: date_format_spec

15.4.6.13 VARCHARおよびVARRAW

datatype_spec句のVARCHARデータ型は文字データを定義し、VARRAWデータ型はバイナリ・データを定義します。

VARCHARデータ型には、文字データが後に続くバイナリ・カウント・フィールドが含まれます。バイナリ・カウント・フィールドの値は、フィールドのバイト数または文字数のいずれかです。数値が、文字数とバイト数のどちらで解釈されるかを指定する方法の詳細は、「STRING SIZES ARE IN」を参照してください。

VARRAWデータ型には、バイナリ・データが後に続くバイナリ・カウント・フィールドが含まれます。バイナリ・カウント・フィールドの値は、バイナリ・データのバイト数です。VARRAWフィールドのデータは、DATA IS...ENDIAN句の影響を受けません。

ACCESS PARAMETERS句の中のVARIABLE 2句は、長さを含むバイナリ・フィールドのサイズを規定します。

オプションのlength_of_lengthフィールドは、カウント・フィールドのバイト数です。VARCHARに対するlength_of_lengthの有効な値は、1、2、4および8です。length_of_lengthを指定しない場合、値に2が使用されます。カウント・フィールドは、DATA IS...ENDIAN句で指定されたとおりの同じエンディアンを持ちます。

max_lenフィールドを使用して、データ・ファイルのフィールドのインスタンスの最大サイズを指定します。VARRAWフィールドでは、max_lenはバイト数です。VARCHARフィールドでは、max_lenは、STRING SIZES ARE IN句の設定に応じて、文字数またはバイト数のいずれかになります。

次に、VARCHARおよびVARRAWの使用例を示します。例の後に、データ・ファイルinfo.datの内容を示します。

```
CREATE TABLE emp_load
             (first_name CHAR(15),
              last_name CHAR(20),
              resume CHAR(2000),
              picture RAW(2000))
  ORGANIZATION EXTERNAL
  (TYPE ORACLE_LOADER
   DEFAULT DIRECTORY ext_tab_dir
   ACCESS PARAMETERS
     (RECORDS
        VARIABLE 2
        DATA IS BIG ENDIAN
        CHARACTERSET US7ASCII
      FIELDS (first_name VARCHAR(2,12),
              last_name VARCHAR(2,20),
              resume VARCHAR(4,10000),
              picture VARRAW(4,100000)))
    LOCATION ('info.dat'));
```

info.datデータ・ファイルの内容

この例で使用するデータ・ファイルの内容は次のとおりです。

```
0005Alvin0008Tolliver0000001DAlvin Tolliver's Resume etc.
0000001013f4690a30bc29d7e40023ab4599ffff
```

わかりやすくするため、カウント・バイトのバイナリ値およびRAWデータの値は、1バイナリ・バイトをイタリック体の2文字で表示していることに注意してください。実際のデータ・ファイルでは、ASCII形式ではなくバイナリ形式の値になります。したがって、この例をカット・アンド・ペーストで使用すると、エラーが返されます。

関連トピック

- STRING SIZES ARE IN

親トピック: datatype_spec句の構文

15.4.6.14 VARCHARCおよびVARRAWC

datatype_spec句のVARCHARCデータ型は文字データを定義し、VARRAWCデータ型はバイナリ・データを定義します。

VARCHARCデータ型には、文字データが後に続く文字カウント・フィールドが含まれます。カウント・フィールドの値は、フィールドのバイト数または文字数のいずれかです。数値が、文字数とバイト数のどちらで解釈されるかを指定する方法の詳細は、STRING SIZES ARE INを参照してください。オプションのlength_of_lengthは、長さが文字とバイトのどちらで解釈されるかに応じて、VARCHARCに対するカウント・フィールドのバイト数または文字数のいずれかになります。

VARCHARCに対するlength_of_lengthsの最大値は、文字列のサイズが文字単位の場合は10で、文字列のサイズがバイト単位の場合は20です。length_of_lengthのデフォルトの値は5です。

VARRAWCデータ型には、バイナリ・データが後に続く文字カウント・フィールドが含まれます。カウント・フィールドの値は、バイナリ・

データのバイト数です。length_of_lengthは、カウント・フィールドのバイト数です。

max_lenフィールドを使用して、データ・ファイルのフィールドのインスタンスの最大サイズを指定します。VARRAWCフィールドでは、max_lenはバイト数です。VARCHARCフィールドでは、max_lenは、STRING SIZES ARE IN句の設定に応じて、文字数またはバイト数のいずれかになります。

次に、VARCHARCおよびVARRAWCの使用例を示します。pictureフィールドの長さは0です。これは、このフィールドがNULLに設定されていることを意味します。

```
CREATE TABLE emp_load
             (first_name CHAR(15),
              last_name CHAR(20),
              resume CHAR(2000),
              picture RAW (2000))
  ORGANIZATION EXTERNAL
  (TYPE ORACLE_LOADER
    DEFAULT DIRECTORY ext_tab_dir
    ACCESS PARAMETERS
      (FIELDS (first_name VARCHARC(5,12),
               last_name VARCHARC(2,20),
               resume VARCHARC(4,10000),
               picture VARRAWC(4,100000)))
  LOCATION ('info.dat'));
00007William05Ricca0035Resume for William Ricca is missing0000
```

関連トピック

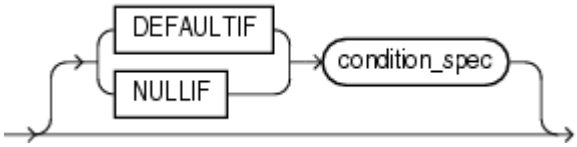
- STRING SIZES ARE IN

親トピック: datatype_spec句の構文

15.4.7 init_spec句

外部表にinit_spec句を使用して、フィールドをNULLまたはデフォルト値に設定するタイミングを指定します。

init_spec句の構文は次のとおりです。



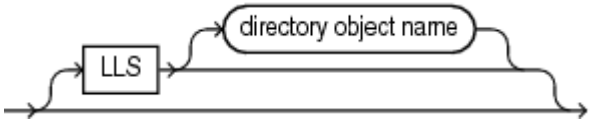
NULLIF句およびDEFAULTIF句は、フィールドに各1回のみ指定できます。これらの句を使用して、次の処理を実行できます。

- NULLIF condition_specを指定し、TRUEと評価された場合、フィールドはNULLに設定されます。
- DEFAULTIF condition_specを指定し、TRUEと評価された場合、フィールドの値はデフォルトの値に設定されます。デフォルトの値は、フィールドのデータ型によって次のように異なります。
  - 文字データ型の場合は、デフォルトの値は空の文字列
  - 数値データ型の場合は、デフォルトの値は0
  - 日付データ型の場合は、デフォルトの値はNULL
- NULLIF句とDEFAULTIF句の両方をフィールドに指定する場合、まずNULLIF句が評価され、DEFAULTIF句は、NULLIF句がFALSEと評価された場合のみに評価されます。

親トピック: field_definitions句

15.4.8 LLS句

データ・ファイルのフィールドがLOBロケーション指定子(LLS)フィールドの場合、LLS句を使用してこのことを示すことができます。LLSフィールドには、ファイル名、データ・ファイル内のLOBデータのオフセットおよび長さが含まれます。SQL*Loaderでは、この情報を使用してLOB列のデータを読み取ります。ORACLE_LOADERのLLS句の構文は、次のとおりです。



LLS句を使用した場合、ORACLE_LOADERはフィールドの値を対応する列にロードしません。かわりに、値の情報を使用して、フィールドの値を見つける場所を判別します。LOBは一部または全体をロードでき、任意の位置から任意の長さで開始できます。ORACLE_LOADERは、フィールドの内容がfilename.ext.nnn.mmm/であることを要求します。各要素は次のように定義されます。

- filename.extは、LOBを含むファイルの名前です。
- nnnは、ファイル内のLOBのバイト単位のオフセットです。
- mmmは、LOBの長さ(バイト)です。-1の値は、LOBがNULLであることを意味します。0の値は、LOBは存在するものの、空であることを示します。
- スラッシュ(/)は、フィールドの終端となります。

LLS句には、Oracleディレクトリ・オブジェクトを指定するオプションのDIRECTORY句があります。

- DIRECTORYを指定した場合、ファイルがその場所に存在する必要があり、また、そのディレクトリ・オブジェクトへのREADアクセス権限が必要です。
- DIRECTORYを指定しない場合、ファイルがデータ・ファイルと同じディレクトリ内に存在する必要があります。

次のいずれかに該当する場合、エラーが返され、行は拒否されます。

- ファイル名に相対パスまたは絶対パスの指定が含まれています。
- ファイルが見つからない、オフセットが無効である、または長さがファイルの終わりを超えています。
- フィールドの内容が予期した書式ではありません。
- LLSフィールドに関連付けられた列のデータ型がCLOB、BLOB、NCLOBのいずれでもない場合。

LLSフィールドが他のフィールドの句(NULLIF句など)によって参照されている場合、アクセス・パラメータ内で、その句の評価に使用される値はデータ・ファイル内の文字列であり、その文字列により示されるファイル内のデータではありません。

LLS句で示されるファイルのデータの文字セットは、データ・ファイルと同じ文字セットとみなされます。

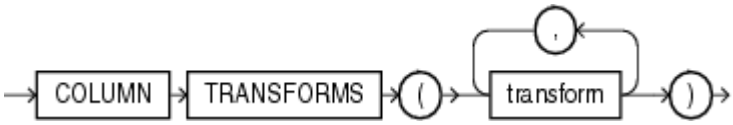
親トピック: field_definitions句

15.5 column_transforms句

オプションのORACLE_LOADERアクセス・ドライバCOLUMN TRANSFORMS句を使用した変換では、データ・ファイル内の列に直接マップされない外部表内の列をロードする方法を指定できます。

構文

column_transforms句の構文は次のようになります。



ノート:

COLUMN TRANSFORMS 句と PREPROCESSOR 句を同時に使用することはできません。

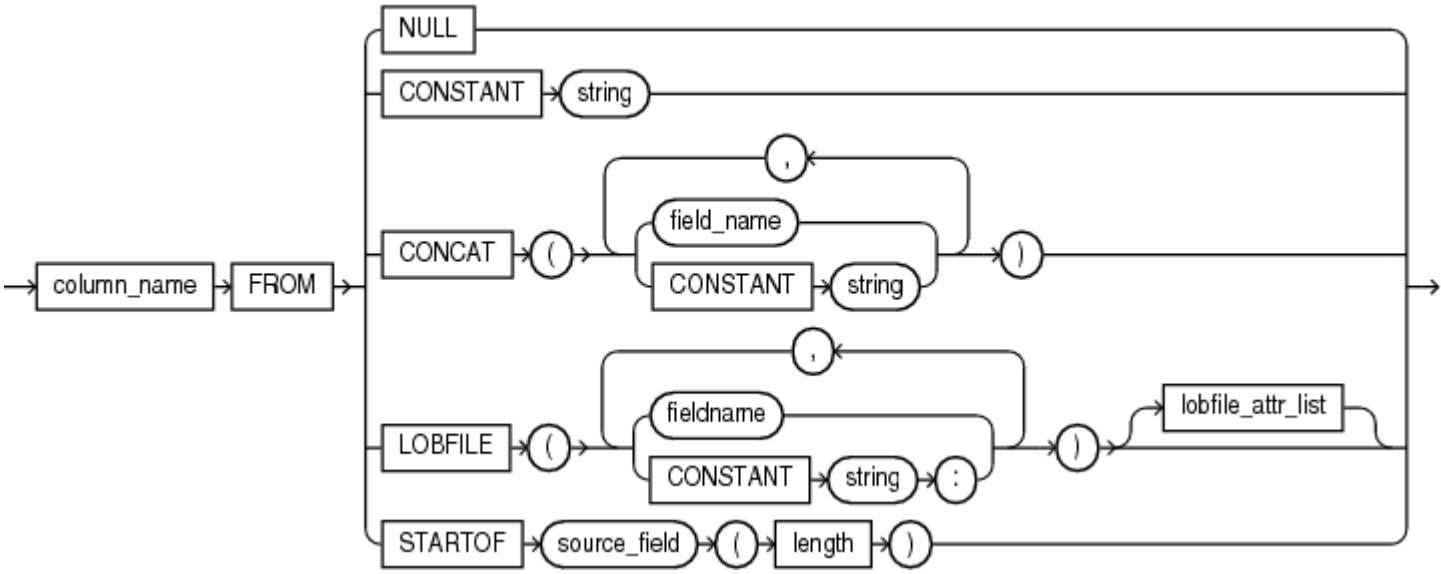
- transform
  transform句で指定した各変換では、外部表の列を特定してから列の値の計算方法を指定します。

親トピック: ORACLE_LOADERアクセス・ドライバ

15.5.1 transform

transform句で指定された各変換では、外部表の列を識別し、次に列の値の計算方法を指定します。

構文は次のとおりです。



NULL変換は、外部表の列を各行でNULLに設定します。CONSTANT変換は、外部表の列をすべての行で同じ値に設定するために使用されます。CONCAT変換は、外部表の列をデータ・ファイルからの現在のレコード内の定数文字列またはフィールド(あるいはその両方)の連結に設定するために使用されます。LOBFILE変換は、別のデータ・ファイルからのレコードのフィールドにデータをロードするために使用されます。これらの各変換については、次の項で詳しく説明します。

- column_name FROM
  column_nameにより、ロードする外部表内の列を一意に特定します。
- NULL
  NULL変換が指定されていると、すべてのレコードでフィールドのすべての値がNULLに設定されます。
- CONSTANT
  CONSTANT句変換では、レコード内の列の値として指定された文字列の値が使用されます。
- CONCAT
  CONCAT変換は、データ・ファイル内の定数文字列とフィールドを連結して1つの文字列にします。
- LOBFILE
  LOBFILE変換は、外部表の列の値として使用する内容が含まれているファイルを特定するために使用します。

- lobfile_attr_list
  lobfile_attr_listは、LOBFILEの追加属性をリストします。
- STARTOF source_field (長さ)
  STARTOFキーワードを使用すると、列がソース・フィールドのデータの部分文字列になることが可能な外部表を作成できます。

親トピック: column_transforms句

15.5.1.1 column_name FROM

column_nameにより、ロードする外部表内の列を一意に特定します。

transform句で列名が参照される場合、その名前は、データ・ファイルのフィールドとしてFIELDS句で指定することはできません。

親トピック: transform

15.5.1.2 NULL

NULL変換が指定された場合、フィールドのすべての値は各レコードに対してNULLに設定されます。

親トピック: transform

15.5.1.3 CONSTANT

CONSTANT句変換では、レコード内の列の値として指定された文字列の値が使用されます。

外部表の列が文字列型でない場合、定数文字列は列のデータ型に変換されます。この変換は各行に対して行われます。

データ型変換で使用される文字列の文字セットは、データベースの文字セットです。

親トピック: transform

15.5.1.4 CONCAT

CONCAT変換は、データ・ファイルの定数文字列とフィールドを連結して1つの文字列にします。

連結の一部として使用できるフィールドは、文字データ型でfields句にリストされているフィールドのみです。他の列変換は、連結の一部として指定できません。

親トピック: transform

15.5.1.5 LOBFILE

LOBFILE変換は、外部表の列の値として内容を使用するファイルを識別します。

すべてのLOBFILEは、directory object:filename形式のオプションのディレクトリ・オブジェクトおよびファイル名で識別されます。LOBFILE変換には次の規則が適用されます。

- ディレクトリ・オブジェクトおよびファイル名は、定数文字列またはフィールド句のフィールド名のいずれかです。
- 定数文字列を指定すると、その文字列を使用して表の各行のLOBFILEが検索されます。
- フィールド名を指定すると、データ・ファイル内のそのフィールドの値を使用してLOBFILEが検索されます。
- ディレクトリ・オブジェクトまたはファイル名のどちらかに対してフィールド名が指定されていて、そのフィールドの値がNULLの場合は、LOBFILEによってロードされる列もNULLに設定されます。

- ディレクトリ・オブジェクトが指定されない場合、外部表に対して指定されたデフォルトのディレクトリが使用されます。
- ディレクトリ・オブジェクトに対してフィールド名が指定された場合、FROM句も指定する必要があります。

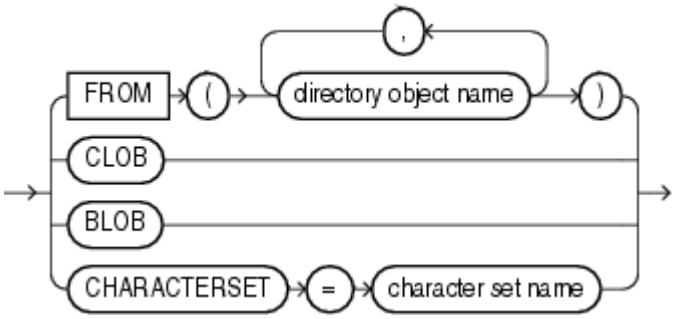
ファイル全体がLOB列の値として使用されます。複数の行で同じファイルが参照されると、各列を移入するためにファイルが再びオープンし、再び読み込まれます。

親トピック: transform

15.5.1.6 lobfile_attr_list

lobfile_attr_listは、LOBFILEの追加属性をリストします。

構文は次のとおりです。



FROM句は、LOBFILEで使用されるすべてのディレクトリ・オブジェクトの名前をリストします。LOBFILEの名前のディレクトリ・オブジェクトに対して、フィールド名が指定された場合にのみ使用します。FROM句の目的は、初期化時に名前付きのディレクトリ・オブジェクトに許可されたアクセス権の種類を識別することにあります。フィールドの値に指定されたディレクトリ・オブジェクトが、リスト内のディレクトリ・オブジェクトではない場合、行は拒否されます。

CLOB属性は、LOBFILE内のデータが文字データであること(RAWデータではないこと)を示します。場合によって文字データは、データベース内にLOBを格納するために使用される文字セットに変換する必要があります。

CHARACTERSET属性には、LOBFILE内のデータの文字セット名が格納されます。

BLOB属性は、LOBFILE内のデータが生データであることを示します。

CLOBまたはBLOBのいずれも指定されない場合、CLOBであるとみなされます。文字LOBFILEに対して文字セットが指定されない場合、データ・ファイルの文字セットであるとみなされます。

親トピック: transform

15.5.1.7 STARTOF source_field (長さ)

STARTOFキーワードを使用すると、列がソース・フィールドのデータの部分文字列になることが可能な外部表を作成できます。

長さは、部分文字列の先頭からの長さです。長さとはバイト数のことであり、変換される外部表の列は文字長セマンティクスではなく、バイト長セマンティクスを使用するとみなされます。(文字長セマンティクスを使用すると、予期しない結果になる可能性があります。)

完全な文字コードのみが移動され、文字は分割されません。したがって、部分文字列がマルチバイト文字の途中で終わると、結果の文字列は短くなります。たとえば、長さに10を指定し、10番目のバイトがマルチバイト文字の最初のバイトである場合、最初の9バイトのみが返されます。

次の例では、部署名(dname)フィールドの最初の4バイトのみが必要となる場合の、STARTOFキーワードの使用方法を示しています。

```
SQL> CREATE TABLE dept (deptno  NUMBER(2),
```

```
  2                     dname   VARCHAR2(14),
  3                     loc     VARCHAR2(13)
  4                          )
  5  ORGANIZATION EXTERNAL
  6  (
  7    DEFAULT DIRECTORY def_dir1
  8    ACCESS PARAMETERS
  9    (
 10      RECORDS DELIMITED BY NEWLINE
 11      FIELDS TERMINATED BY ','
 12      (
 13        deptno           CHAR(2),
 14        dname_source     CHAR(14),
 15        loc              CHAR(13)
 16      )
 17      column transforms
 18      (
 19         dname FROM STARTOF dname_source (4)
 20      )
 21    )
 22    LOCATION ('dept.dat')
 23  );

Table created.
```

dept表からSELECT操作を実行する場合、dnameフィールドの最初の4バイトが返されます。

```
SQL> SELECT * FROM dept;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCO           NEW YORK
        20 RESE           DALLAS
        30 SALE           CHICAGO
        40 OPER           BOSTON

4 rows selected.
```

親トピック: transform

15.6 ORACLE_LOADERアクセス・ドライバのパラレル・ロードに関する考慮点

ORACLE_LOADERアクセス・ドライバでは、大きいデータ・ファイルを、個別に処理できるチャンクに分割します。

次のファイル、レコードおよびデータ特性によって、ファイルのパラレル処理が禁止されます。

- 順次データ・ソース(テープ・ドライブ、パイプなど)
- 文字の境界が文字列中の任意のバイトで始まり、境界を判断できないマルチバイト文字セットのデータ

  この制限事項は、1レコード当たりのバイト数が固定のデータ・ファイルには適用されません。
- VAR形式のレコード

PARALLEL句の指定は、大量のデータを扱う場合にのみ有効です。

親トピック: ORACLE_LOADERアクセス・ドライバ

15.7 ORACLE_LOADERアクセス・ドライバを使用する際のパフォーマンスのヒント

このトピックでは、ORACLE_LOADERアクセス・ドライバを使用する際のパフォーマンスのヒントについて説明します。

パフォーマンスを監視する場合、最も重要なことは、ロードの経過時間の測定です。また、CPU使用量、メモリー使用量およびI/O率の測定も重要です。

並列度を増減することによって、パフォーマンスを変更できます。並列度は、データ・ファイルの処理に起動できるアクセス・ドライバの数を示します。並列度によって、リソース使用率を低くした遅いロードと、すべてのリソースを使用した速いロードを選択できます。アクセス・ドライバは、アクセス・ドライバ専用に使用するリソース量を判断できないため、自動的にはチューニングされません。

さらに考慮する必要があるのは、アクセス・ドライバで大きなI/Oバッファを使用すると、パフォーマンスが向上するという点です(バッファのサイズは、READSIZE句をアクセス・パラメータで使用して指定できます)。共有サーバーを使用するデータベースでは、アクセス・ドライバが使用するすべてのメモリーはSystem Global Area(SGA)から割り当てられます。そのため、共有サーバー上の外部表を使用する際には注意が必要です。

また、パフォーマンスは、日付キャッシュ機能を使用して向上できる場合もあります。ロード中に予測される一意の日付の数を指定するために日付キャッシュを使用すると、入力データ内に多数の重複する日付またはタイムスタンプ値が存在する場合に日付変換が実行される回数を減らすことができます。外部表で提供される日付キャッシュ機能は、SQL*Loaderで提供されるものと同じです。詳細は、「DATE_CACHE」を参照してください。

パフォーマンスを向上させるには、並列度の変更および日付キャッシュの使用に加えて、次のことを考慮してください。

- 固定長レコードは、文字列で終了しているレコードより速く処理される。
- 固定長フィールドは、デリミタ付きフィールドより速く処理される。
- シングルバイト文字セットは、最も速く処理される。
- 固定幅文字セットは、可変幅文字セットより速く処理される。
- 可変幅文字セットのバイト長セマンティクスは、文字長セマンティクスより早く処理される。
- 1文字のレコード終了デリミタおよびフィールド・デリミタは、複数文字のデリミタより速く処理される。
- 文字セットを変換するより、データ・ファイルの文字セットをデータベースの文字セットに一致させる方が速く処理される。
- データ型を変換するより、データ・ファイルのデータ型をデータベースのデータ型に一致させる方が速く処理される。
- 拒否された行を拒否ファイルに書き込まない場合は、オーバーヘッドが削減されるため、処理速度が速くなる。
- 条件句(WHEN、NULLIFおよびDEFAULTIFを含む)を使用すると、処理速度が遅くなる。
- アクセス・ドライバは、マルチスレッドを使用して作業をできるだけ簡素化します。

親トピック: ORACLE_LOADERアクセス・ドライバ

15.8 ORACLE_LOADERアクセス・ドライバ使用時の制限事項

この項では、ORACLE_LOADERアクセス・ドライバを使用する際に認識しておく必要がある制限事項について説明します。

具体的には次のとおりです。

- 暗号化された列がある外部表のエクスポートとインポートは、サポートされていません。

- 列処理: デフォルトでは、外部表の機能は、外部表に定義されたすべての列をフェッチします。これによって、すべての問合せに対し、一貫した結果が保証されます。ただし、パフォーマンスを考慮して、外部表内で参照される列だけを処理することができます。これによって、データ変換の量および問合せの実行で必要なデータ処理の量を最小限にします。この場合、データ型変換エラーが発生した列を含んでいたために拒否された行も、その列が参照されないかぎり、別の問合せでは拒否されません。ALTER TABLEコマンドを使用すると、この列処理動作を変更できます。
- 外部表は、LONG列にデータをロードできません。
- ORACLE_LOADERアクセス・ドライバのアクセス・パラメータにSQL文字列を指定できません。解決策として、外部表を読み取る文のSELECT句にDECODE句を使用できます。また、DECODE句を使用する外部表のビューを作成して、外部表ではなくそのビューから選択することもできます。
- 外部表では、文字列内のバックスラッシュ(¥)の使用はサポートされていません。詳細は、「バックスラッシュ・エスケープ文字の使用」を参照してください。
- 外部表アクセス・パラメータにおいて列名や表名などの識別子が指定される場合、特定の値はアクセス・パラメータ・パーサーによって予約語とみなされます。予約語を識別子として使用するには、二重引用符で囲む必要があります。

親トピック: ORACLE_LOADERアクセス・ドライバ

15.9 ORACLE_LOADERアクセス・ドライバの予約語

外部表アクセス・パラメータにおいて列名や表名などの識別子が指定される場合、特定の値はアクセス・パラメータ・パーサーによって予約語とみなされます。

予約語を識別子として使用するには、二重引用符で囲む必要があります。ORACLE_LOADERアクセス・ドライバの予約語は、次のとおりです。

- ALL
- AND
- ARE
- ASTERISK
- AT
- ATSIGN
- BADFILE
- BADFILENAME
- BACKSLASH
- BENDIAN
- BIG
- BLANKS
- BY
- BYTES
- BYTESTR

- CHAR
- CHARACTERS
- CHARACTERSET
- CHARSET
- CHARSTR
- CHECK
- CLOB
- COLLENGTH
- COLON
- COLUMN
- COMMA
- CONCAT
- CONSTANT
- COUNTED
- DATA
- DATE
- DATE_CACHE
- DATE_FORMAT
- DATEMASK
- DAY
- DEBUG
- DECIMAL
- DEFAULTIF
- DELIMITBY
- DELIMITED
- DISCARDFILE
- DNFS_ENABLE
- DNFS_DISABLE
- DNFS_READBUFFERS
- DOT
- DOUBLE
- DOUBLETYPE

- DQSTRING
- DQUOTE
- DSCFILENAME
- ENCLOSED
- ENDIAN
- ENDPOS
- EOF
- EQUAL
- EXIT
- EXTENDED_IO_PARAMETERS
- EXTERNAL
- EXTERNALKW
- EXTPARM
- FIELD
- FIELDS
- FILE
- FILEDIR
- FILENAME
- FIXED
- FLOAT
- FLOATTYPE
- FOR
- FROM
- HASH
- HEXPREFIX
- IN
- INTEGER
- INTERVAL
- LANGUAGE
- IS
- LEFTCB
- LEFTTXTDELIM

- LEFTP
- LENDIAN
- LDRTRIM
- LITTLE
- LOAD
- LOBFILE
- LOBPC
- LOBPCCONST
- LOCAL
- LOCALTZONE
- LOGFILE
- LOGFILENAME
- LRTRIM
- LTRIM
- MAKE_REF
- MASK
- MINUSSIGN
- MISSING
- MISSINGFLD
- MONTH
- NEWLINE
- NO
- NOCHECK
- NOT
- NOBADFILE
- NODISCARDFILE
- NOLOGFILE
- NOTEQUAL
- NOTERMBY
- NOTRIM
- NULL
- NULLIF

- OID
- OPTENCLOSE
- OPTIONALLY
- OPTIONS
- OR
- ORACLE_DATE
- ORACLE_NUMBER
- PLUSSIGN
- POSITION
- PROCESSING
- QUOTE
- RAW
- READSIZE
- RECNUM
- RECORDS
- REJECT
- RIGHTCB
- RIGHTTXTDELIM
- RIGHTP
- ROW
- ROWS
- RTRIM
- SCALE
- SECOND
- SEMI
- SETID
- SIGN
- SIZES
- SKIP
- STRING
- TERMBY
- TERMEOF

- TERMINATED
- TERMWS
- TERRITORY
- TIME
- TIMESTAMP
- TIMEZONE
- TO
- TRANSFORMS
- UNDERSCORE
- UINTEGER
- UNSIGNED
- VALUES
- VARCHAR
- VARCHARC
- VARIABLE
- VARRAW
- VARRAWC
- VLENELN
- VMAXLEN
- WHEN
- WHITESPACE
- WITH
- YEAR
- ZONED

親トピック: ORACLE_LOADERアクセス・ドライバ

16 ORACLE_DATAPUMPアクセス・ドライバ

ORACLE_DATAPUMPアクセス・ドライバは、ORACLE_DATAPUMP型の外部表に固有のアクセス・パラメータのセットを提供します。

- ORACLE_DATAPUMPアクセス・ドライバの使用
  アクセス・ドライバのデフォルト動作を変更するには、ORACLE_DATAPUMPアクセス・パラメータを使用します。アクセス・ドライバで指定する情報により、データ・ソースのデータが、外部表の定義と一致するように処理されます。
- access_parameters句
  ORACLE_DATAPUMPアクセス・ドライバの外部表を作成する場合、access_parameters句に特定のパラメータを指定できます。
- ORACLE_DATAPUMPアクセス・ドライバを使用したデータのアンロードとロード
  ORACLE_DATAPUMPアクセス・ドライバは、SQLのCREATE TABLE AS SELECT文を使用した外部表の作成の一環として、ダンプ・ファイルにデータを書き込むことができます。
- サポートされるデータ型
  ORACLE_DATAPUMPアクセス・ドライバは、ロードおよびアンロード時に多くのデータ型を自動的に解決します。
- サポートされないデータ型
  ORACLE_DATAPUMPアクセス・ドライバを使用すると、サポートされていないデータ型のデータをアンロードおよび再ロードできます。
- ORACLE_DATAPUMPアクセス・ドライバを使用する際のパフォーマンスのヒント
  ORACLE_DATAPUMPアクセス・ドライバのパフォーマンスを向上させる方法について説明します。
- ORACLE_DATAPUMPアクセス・ドライバ使用時の制限事項
  ORACLE_DATAPUMPアクセス・ドライバを使用した外部表へのアクセスに適用される制限事項に注意してください。
- ORACLE_DATAPUMPアクセス・ドライバの予約語
  ORACLE_DATAPUMPアクセス・ドライバで予約されている識別子の語を使用する場合は、二重引用符で囲む必要があります。

親トピック: 外部表

16.1 ORACLE_DATAPUMPアクセス・ドライバの使用

アクセス・ドライバのデフォルト動作を変更するには、ORACLE_DATAPUMPアクセス・パラメータを使用します。アクセス・ドライバで指定する情報により、データ・ソースのデータが、外部表の定義と一致するように処理されます。

ORACLE_DATAPUMPアクセス・ドライバを正しく使用するには、使用するプラットフォームのデータ・ファイルのファイル形式およびレコード形式(文字セット、フィールドのデータ型など)についての知識が必要です。また、SQLを使用して外部表を作成し、作成した表に対して問合せを実行できる必要もあります。

ノート:

- 章の後半で説明されているその他の構文を使用しなければ、わかりにくい場合があります。構文によって行われる処理が明確でない場合は、先に進み、その説明を参照できます。
- 外部表アクセス・パラメータにおいて列名や表名などの識別子が指定される場合、特定の値はアクセス・パラメータ・パーサーによって予約語とみなされます。予約語を識別子として使用するには、二重引用符で囲

む必要があります。詳細は、「ORACLE_DATAPUMP アクセス・ドライバの予約語」を参照してください。

親トピック: ORACLE_DATAPUMPアクセス・ドライバ

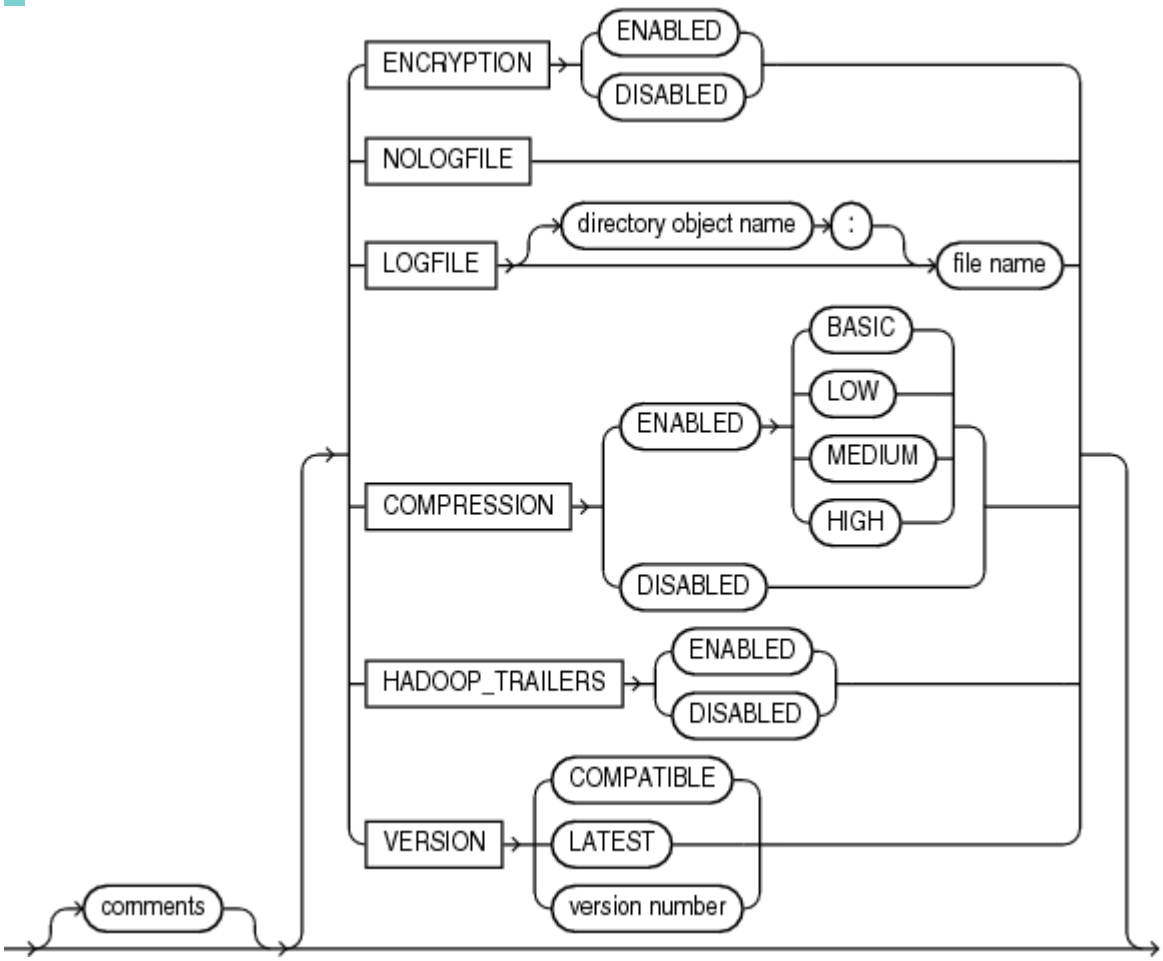
16.2 access_parameters句

ORACLE_DATAPUMPアクセス・ドライバの外部表を作成する場合、access_parameters句に特定のパラメータを指定できます。

この句およびその個別のパラメータはオプションです。たとえば、LOGFILEを指定し、VERSIONは指定しない、またはその逆も可能です。access_parameters句の構文は次のとおりです。

ノート:

これらのアクセス・パラメータは、SQL 文の CREATE TABLE...ORGANIZATION EXTERNAL の opaque_format_spec としてまとめて参照されます。



- コメント
- COMPRESSION
- ENCRYPTION
- LOGFILE | NOLOGFILE
- VERSION句
  VERSION句は、ダンプ・ファイルを読み取るOracle Databaseの最小リリースを指定するために使用します。
- HADOOP_TRAILERS句

ORACLE_DATAPUMPアクセス・ドライバにはHADOOP_TRAILERS句が用意されており、ENABLEDまたはDISABLED (デフォルト)に設定できます。

- SQL ENCRYPT句の使用による影響
  ORACLE_DATAPUMPアクセス・ドライバのENCRYPT句を使用して列を暗号化する場合の外部表の要件およびガイドラインを確認します。

関連トピック

- CREATE TABLE

関連項目:

SQL CREATE TABLE...ORGANIZATION EXTERNAL文を使用するときにopaque_format_specを指定する方法の詳細は、*Oracle Database SQL*言語リファレンスのCREATE TABLEに関する項を参照してください。

親トピック: ORACLE_DATAPUMPアクセス・ドライバ

16.2.1 コメント

コメントは、2つのハイフンで始まり、その後にテキストが続く行です。コメントは、アクセス・パラメータより前に位置する必要があります。たとえば:

```
--This is a comment.
--This is another comment.
NOLOG
```

二重ハイフンの右側のすべてのテキストは行末まで無視されます。

親トピック: access_parameters句

16.2.2 COMPRESSION

デフォルト: DISABLED

用途

データをダンプ・ファイル・セットに書き込む前にデータを圧縮するかどうかと、使用する圧縮アルゴリズム(オプション)を指定します。

構文および説明

```
COMPRESSION [ENABLED {BASIC | LOW| MEDIUM | HIGH} | DISABLED]
```

- ENABLEDを指定すると、アンロード操作全体ですべてのデータが圧縮されます。追加で次の圧縮オプションのいずれかを指定できます。
  - BASIC - 圧縮率と速度の適度な組合せが実現し、使用されるアルゴリズムはOracle Data Pumpの以前のバージョンと同じです。
  - LOW - アンロード・スループットへの影響を最小限に抑えます。CPUリソースが限定要因となっている環境に適しています。
  - MEDIUM - ほとんどの環境に適しています。このオプションはBASICオプションと同様、圧縮率と処理速度を加味したよい組合せですが、BASICとは異なるアルゴリズムが使用されています。
  - HIGH - ネットワーク速度が限定要因となっている、速度が遅いネットワークを使用するアンロードに最適です。

ノート:

これらの圧縮アルゴリズムを使用するには、COMPATIBLE 初期化パラメータを 12.0.0 以上に設定する必要があります。この機能では、Oracle Advanced Compression オプションを有効にする必要があります。

圧縮アルゴリズムのパフォーマンスは、そのCPU使用量および圧縮率(非圧縮の入力に対する割合としての圧縮済の出力のサイズ)によって特徴付けられます。これらの測定値は、使用される圧縮アルゴリズムの速度に加え、入力のサイズおよびタイプに応じて変化します。基本的に圧縮率はLOWからHIGHへと高くなります。ただし、圧縮率の上昇に応じて、消費されるCPUリソースが多くなります。

ご使用の環境で、データに対して様々な圧縮レベルをテストすることをお薦めします。エクスポートされるダンプ・ファイル・セットの圧縮レベルがパフォーマンスおよび記憶域要件を満たすようにするには、環境、ワークロードの特性、およびデータのサイズとタイプに基づいて圧縮レベルを選択します。

- DISABLEDを指定すると、アップロード操作でデータは圧縮されません。

例

次の例では、COMPRESSIONパラメータがENABLEDに設定されています。したがって、dept.dmpダンプ・ファイルに書き込まれるすべてのデータは圧縮形式になります。

```
CREATE TABLE deptXTec3
 ORGANIZATION EXTERNAL (TYPE ORACLE_DATAPUMP DEFAULT DIRECTORY def_dir1
 ACCESS PARAMETERS (COMPRESSION ENABLED) LOCATION ('dept.dmp'));
```

親トピック: access_parameters句

16.2.3 ENCRYPTION

デフォルト

DISABLED

用途

ダンプ・ファイル・セットに書き込む前にデータを暗号化するかどうかを指定します。

構文および説明

```
ENCRYPTION [ENABLED | DISABLED]
```

ENABLEDを指定すると、すべてのデータが暗号化形式でダンプ・ファイル・セットに書き込まれます。

DISABLEDを指定すると、データは暗号化形式でダンプ・ファイル・セットに書き込まれません。

制限事項

このパラメータは、エクスポート操作でのみ使用します。

例

次の例では、ENCRYPTIONパラメータがENABLEDに設定されています。したがって、dept.dmpファイルに書き込まれるすべてのデータは暗号化形式になります。

```
CREATE TABLE deptXTec3
```

```
ORGANIZATION EXTERNAL (TYPE ORACLE_DATAPUMP DEFAULT DIRECTORY def_dir1
ACCESS PARAMETERS (ENCRYPTION ENABLED) LOCATION ('dept.dmp'));
```

親トピック: access_parameters句

16.2.4 LOGFILE | NOLOGFILE

デフォルト: LOGFILEが指定されない場合、デフォルトのディレクトリにログ・ファイルが作成され、ログ・ファイルの名前は、表名および.logの拡張子を持つプロセスIDから生成されます。ログ・ファイルがすでに同じ名前で存在する場合は、アクセス・ドライバによってそのログ・ファイルが再びオープンされ、新しいログ情報がファイルの終わりに追加されます。

用途

LOGFILEは、ダンプ・ファイルのアクセス中に生成されたすべてのメッセージを含むログ・ファイルの名前を指定します。NOLOGFILEを使用してログ・ファイルの作成を回避できます。

構文および説明

```
NOLOGFILE
```

または

```
LOGFILE [directory_object:]logfile_name
```

ログ・ファイル名の一部としてディレクトリ・オブジェクトを指定しない場合、DEFAULT DIRECTORY属性で指定されたディレクトリ・オブジェクトが使用されます。ディレクトリ・オブジェクトが指定されず、デフォルトのディレクトリも指定されない場合は、エラーが返されます。置換変数を使用してパラレル・ロードおよびパラレル・アンロード中に一意のファイル名を作成する方法については、「LOGFILEのファイル名」を参照してください。

例

次の例では、ダンプ・ファイルdept_dmpは、ディレクトリ・オブジェクトload_dirで識別されるディレクトリ内にありますが、ログ・ファイルdeptxt.logは、ディレクトリ・オブジェクトlog_dirで識別されるディレクトリ内にあります。

```
CREATE TABLE dept_xt (dept_no INT, dept_name CHAR(20), location CHAR(20))
ORGANIZATION EXTERNAL (TYPE ORACLE_DATAPUMP DEFAULT DIRECTORY load_dir
ACCESS PARAMETERS (LOGFILE log_dir:deptxt) LOCATION ('dept_dmp'));
```

- パラレル・ロードのログ・ファイル・ネーミング

親トピック: access_parameters句

16.2.4.1 パラレル・ロードのログ・ファイル・ネーミング

パラレル・ロードでファイル名を一意にするには、アクセス・ドライバで記号置換を行います。サポートされている記号置換は次のとおりです。

- %pは、現行のプロセスのプロセスIDに置換されます。たとえば、アクセス・ドライバのプロセスIDが12345の場合、exttab_%p.logは、exttab_12345.logとなります。
- %aは、現行のプロセスのエージェント番号に置換されます。エージェント番号は、外部表にアクセスしている各パラレル・プロセスに割り当てられた一意の番号です。この番号には、3文字になるように、左側に0が埋められます。たとえば、3番目のパラレル・エージェントがファイルを作成する際に、exttab_%a.logがファイル名として指定されている場合、エージェントはexttab_003.logというファイルを作成します。
- %%は、%に置換されます。ファイル名にパーセント符号が必要な場合、この記号置換を使用する必要があります。

%文字に前述のリスト内の文字以外の文字が続く場合、エラーが返されます。

%pまたは%aを使用しないで出力ファイルに対して一意のファイル名を作成し、外部表にパラレルでアクセス中の場合、出力ファイルが破損するか、エージェントがファイルに書込みをできないという問題が発生する場合があります。

ファイルに対して拡張子が指定されていない場合は、デフォルトの拡張子である.logが使用されます。生成された名前が有効なファイル名ではない場合、エラーが返され、データのロードまたはアンロードは行われません。

親トピック: LOGFILE | NOLOGFILE

16.2.5 VERSION句

VERSION句は、ダンプ・ファイルを読み取るOracle Databaseの最低のリリースを指定するために使用されます。

たとえば、11.1と指定すると、Oracle Database 11gリリース1 (11.1および11.2)データベースの両方でダンプ・ファイルを読み取ることができます。11.2と指定すると、Oracle Database 11gリリース2(11.2)データベースのみがダンプ・ファイルを読み取ることができます。

デフォルト値はCOMPATIBLEです。

親トピック: access_parameters句

16.2.6 HADOOP_TRAILERS句

ORACLE_DATAPUMPアクセス・ドライバにはHADOOP_TRAILERS句が用意されており、ENABLEDまたはDISABLED (デフォルト)に設定できます。

HADOOP_TRAILERS句を有効にすると、Hadoopトレーラがダンプ・ファイルに書き込まれます。Hadoopトレーラには、ファイルの様々な部分の場所およびサイズに関する情報が含まれます。情報は、先頭ではなく、ファイルの末尾のダンプ・トレーラ・ブロックと、ストリーム・データの末尾に書き込まれます。

親トピック: access_parameters句

16.2.7 SQL ENCRYPT句の使用による影響

ORACLE_DATAPUMPアクセス・ドライバのENCRYPT句を使用して列を暗号化する場合の外部表の要件およびガイドラインを確認します。

外部表の作成時にSQL ENCRYPT句を指定する場合は、次の点に注意してください。

- ENCRYPT句を指定した列は、暗号化されてからダンプ・ファイルに書き込まれます。
- ダンプ・ファイルを別のデータベースに移動する場合は、ダンプ・ファイル内の暗号化列、およびダンプ・ファイルの読込みに使用する外部表の両方で、同じ暗号化パスワードを使用する必要があります。
- 2つ目のデータベースの外部表で暗号化されている正しい列に対してパスワードを指定しなかった場合は、エラーが発生します。指定したパスワードが正しくない場合は、ダンプ・ファイルのデータが無効になります。
- 作成するダンプ・ファイルのリリースは10.2以上にする必要があります。それ以外の場合はエラーが戻されます。

関連トピック

- 『Oracle Database SQL言語リファレンス』の「CREATE TABLE」

関連項目:

CREATE TABLE文でのENCRYPT句の使用方法の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

親トピック: access_parameters句

16.3 ORACLE_DATAPUMPアクセス・ドライバを使用したデータのアンロードとロード

ORACLE_DATAPUMPアクセス・ドライバは、SQLのCREATE TABLE AS SELECT文を使用した外部表の作成の一環として、ダンプ・ファイルにデータを書き込むことができます。

ファイル内のデータは、バイナリ形式で記述され、ORACLE_DATAPUMPアクセス・ドライバによってのみ読取りが可能です。作成されたダンプ・ファイルは変更できません(つまり、ダンプ・ファイルに対してデータ操作言語(DML)操作は実行できません)。ただし、このファイルは何回でも読み取ることができ、別の外部表のダンプ・ファイルとして同じデータベースまたは異なるデータベースで使用できます。

次のステップではサンプル・スキーマoeを使用して、ORACLE_DATAPUMPアクセス・ドライバを使用したデータのアンロード方法およびロード方法の例を示します。(例では、ディレクトリ・オブジェクトdef_dir1がすでに存在し、ユーザーoeにはそれに対する読取りおよび書込み権限が付与されているとします。)

1. 外部表は、AS SELECT句を使用した外部表の作成の一環としてのみファイルにデータを移入します。次の例では、inventories_xtという名前の外部表を作成し、外部表のダンプ・ファイルにoeスキーマ内のinventories表のデータを移入します。

```
SQL> CREATE TABLE inventories_xt
  2  ORGANIZATION EXTERNAL
  3  (
  4    TYPE ORACLE_DATAPUMP
  5    DEFAULT DIRECTORY def_dir1
  6    LOCATION ('inv_xt.dmp')
  7  )
  8  AS SELECT * FROM inventories;
Table created.
```

2. inventoriesおよび新しい外部表の両方を、次のとおり記述します。その両方は一致している必要があります。

```
SQL> DESCRIBE inventories
 Name                                      Null?    Type
 ----------------------------------------- --------- ----------------
 PRODUCT_ID                                NOT NULL NUMBER(6)
 WAREHOUSE_ID                              NOT NULL NUMBER(3)
 QUANTITY_ON_HAND                          NOT NULL NUMBER(8)
SQL> DESCRIBE inventories_xt
 Name                                      Null?    Type
 ----------------------------------------- -------- -----------------
 PRODUCT_ID                                NOT NULL NUMBER(6)
 WAREHOUSE_ID                              NOT NULL NUMBER(3)
 QUANTITY_ON_HAND                          NOT NULL NUMBER(8)
```

3. 作成された外部表は、他の表と同様に問合せを行うことができます。たとえば、次のように外部表のレコード数を選択します。

```
SQL> SELECT COUNT(*) FROM inventories_xt;
  COUNT(*)
----------
      1112
```

4. 外部表のデータをinventoriesのデータと比較します。違いがないことを確認します。

```
SQL> SELECT * FROM inventories MINUS SELECT * FROM inventories_xt;
no rows selected
```

5. 外部表が作成され、CREATE TABLE AS SELECT文によりダンプ・ファイルが移入された後では、外部表に対する行の追加、更新または削除は実行できません。外部表のデータの変更を試みると、エラーが発生します。

   次の例では、既存の外部表へのデータ操作言語(DML)の使用を試みます。次のエラーが返されます。

```
SQL> DELETE FROM inventories_xt WHERE warehouse_id = 5;
DELETE FROM inventories_xt WHERE warehouse_id = 5
            *
ERROR at line 1:
ORA-30657: operation not supported on external organized table
```

6. 外部表に対して作成されたダンプ・ファイルは移動できるようになり、同じデータベースまたは異なるデータベース内の別の外部表のダンプ・ファイルとして使用できます。既存のファイルを使用する外部表を作成した場合、CREATE TABLE文のAS SELECT句は存在しません。

```
SQL> CREATE TABLE inventories_xt2
  2  (
  3    product_id          NUMBER(6),
  4    warehouse_id        NUMBER(3),
  5    quantity_on_hand    NUMBER(8)
  6  )
  7  ORGANIZATION EXTERNAL
  8  (
  9    TYPE ORACLE_DATAPUMP
 10    DEFAULT DIRECTORY def_dir1
 11    LOCATION ('inv_xt.dmp')
 12  );
Table created.
```

7. 新しい外部表のデータをinventories表のデータと比較します。比較する前に、product_idフィールドは互換性のあるデータ型に変換されます。違いがないことを確認します。

```
SQL> SELECT * FROM inventories MINUS SELECT * FROM inventories_xt2;
no rows selected
```

8. 3つのダンプ・ファイルおよび3の並列度の外部表を作成します。

```
SQL> CREATE TABLE inventories_xt3
  2  ORGANIZATION EXTERNAL
  3  (
  4    TYPE ORACLE_DATAPUMP
  5    DEFAULT DIRECTORY def_dir1
  6    LOCATION ('inv_xt1.dmp', 'inv_xt2.dmp', 'inv_xt3.dmp')
  7  )
  8  PARALLEL 3
  9  AS SELECT * FROM inventories;
Table created.
```

9. アンロードされたデータとinventoriesを比較します。違いがないことを確認します。

```
SQL> SELECT * FROM inventories MINUS SELECT * FROM inventories_xt3;
no rows selected
```

10. inventories表の数行を含む外部表を作成します。

```
SQL> CREATE TABLE inv_part_xt
  2  ORGANIZATION EXTERNAL
  3  (
```

```
  4  TYPE ORACLE_DATAPUMP
  5  DEFAULT DIRECTORY def_dir1
  6  LOCATION ('inv_p1_xt.dmp')
  7  )
  8  AS SELECT * FROM inventories WHERE warehouse_id < 5;

Table created.
```

11. inventoriesの残りの行を含む別の外部表を作成します。

```
SQL> drop table inv_part_xt;

Table dropped.

SQL>
SQL> CREATE TABLE inv_part_xt
  2  ORGANIZATION EXTERNAL
  3  (
  4  TYPE ORACLE_DATAPUMP
  5  DEFAULT DIRECTORY def_dir1
  6  LOCATION ('inv_p2_xt.dmp')
  7  )
  8  AS SELECT * FROM inventories WHERE warehouse_id >= 5;

Table created.
```

12. ステップ10およびステップ11で作成した2つのダンプ・ファイルを使用する外部表を作成します。

```
SQL> CREATE TABLE inv_part_all_xt
  2  (
  3  product_id NUMBER(6),
  4  warehouse_id NUMBER(3),
  5  quantity_on_hand NUMBER(8)
  6  )
  7  ORGANIZATION EXTERNAL
  8  (
  9  TYPE ORACLE_DATAPUMP
 10  DEFAULT DIRECTORY def_dir1
 11  LOCATION ('inv_p1_xt.dmp','inv_p2_xt.dmp')
 12  );

Table created.
```

13. 新しい外部表とinventories表を比較します。違いがないことを確認します。これは、外部表の作成で使用された2つのダンプ・ファイルに同じメタデータ(同じ表名inv_part_xt、同じ列情報など)があるためです。

```
SQL> SELECT * FROM inventories MINUS SELECT * FROM inv_part_all_xt;
no rows selected
```

- パラレル・ロードおよびパラレル・アンロード
  このトピックでは、パラレル・ロードおよびパラレル・アンロードについて説明します。
- ダンプ・ファイルの結合
  異なる外部表から移入されたすべてのダンプ・ファイルは、別の外部表のLOCATION句で指定できます。

親トピック: ORACLE_DATAPUMPアクセス・ドライバ

16.3.1 パラレル・ロードおよびパラレル・アンロード

このトピックでは、パラレル・ロードおよびパラレル・アンロードについて説明します。

ダンプ・ファイルは、記述されるすべてのデータを保持できるサイズのディスクに格納する必要があります。すべてのデータを保持で

きない容量の場合、CREATE TABLE AS SELECT文に対してエラーが返されます。この問題を軽減するには、CREATE TABLE AS SELECT文の実行時に複数のディレクトリ・オブジェクトを作成するという方法があります。ただし、これらのディレクトリは、異なるディスクに存在する必要があります。LOCATION句でdirectory:file形式を使用して複数の位置を指定し、PARALLEL句を指定することで複数のファイルを作成できます。外部表を移入するために作成された各パラレルI/Oサーバー・プロセスでは、固有のファイルに書込みを実行します。各I/Oサーバー・プロセスには固有のファイルが必要であるため、LOCATION句のファイル数と並列度が一致している必要があります。余分なファイルを指定した場合は、すべて無視されます。指定された並列度に対して必要数のファイルが存在しない場合、LOCATION句のファイル数と一致するよう並列度が下げられます。

inventories表の3つのファイルへのアンロードの例を次に示します。

```
SQL> CREATE TABLE inventories_XT_3
  2  ORGANIZATION EXTERNAL
  3  (
  4    TYPE ORACLE_DATAPUMP
  5    DEFAULT DIRECTORY def_dir1
  6    LOCATION ('inv_xt1.dmp', 'inv_xt2.dmp', 'inv_xt3.dmp')
  7  )
  8  PARALLEL 3
  9  AS SELECT * FROM oe.inventories;
Table created.
```

ORACLE_DATAPUMPアクセス・ドライバを使用してデータをロードする場合は、パラレル処理で、複数のダンプ・ファイルや同じダンプ・ファイルのチャンクも同時に読み取ることができます。したがって、ファイルが複数のファイル・オフセットを含むことができる大きさであれば、ダンプ・ファイルが1つであっても、データをパラレルでロードできます。ORACLE_DATAPUMP外部表から読み込む場合、並列度はLOCATION句のファイル数に関連付けられません。

親トピック: ORACLE_DATAPUMPアクセス・ドライバを使用したデータのアンロードとロード

16.3.2 ダンプ・ファイルの結合

異なる外部表から移入されたすべてのダンプ・ファイルは、別の外部表のLOCATION句で指定できます。

たとえば、異なる本番データベースのデータを個別のファイルにアンロードし、次にそれらのファイルをデータ・ウェアハウスで定義された外部表に含めることができます。これにより、複数のソースから容易にデータを集計できます。唯一の制限は、すべての外部表のメタデータが完全に同じでなければいけない必要がある点です。つまり、文字セット、タイムゾーン、スキーマ名、表名および列名がすべて一致する必要があります。また、列は同じ順序で定義され、そのデータ型も完全に同一である必要があります。1つ目の外部表を作成した後で、2つ目の外部表に同じ表名を使用できるようにするには、1つ目の外部表を削除する必要があります。こうすることで、2つのダンプ・ファイルにリストされたメタデータは同じ状態になり、これらのダンプ・ファイルを同時に使用して同じ外部表を作成できます。

```
SQL> CREATE TABLE inv_part_1_xt
  2  ORGANIZATION EXTERNAL
  3  (
  4    TYPE ORACLE_DATAPUMP
  5    DEFAULT DIRECTORY def_dir1
  6    LOCATION ('inv_p1_xt.dmp')
  7  )
  8  AS SELECT * FROM oe.inventories WHERE warehouse_id < 5;
Table created.
SQL> DROP TABLE inv_part_1_xt;
SQL> CREATE TABLE inv_part_1_xt
  2  ORGANIZATION EXTERNAL
  3  (
  4    TYPE ORACLE_DATAPUMP
  5    DEFAULT directory def_dir1
```

```
  6    LOCATION ('inv_p2_xt.dmp')
  7  )
  8  AS SELECT * FROM oe.inventories WHERE warehouse_id >= 5;
Table created.
SQL> CREATE TABLE inv_part_all_xt
  2  (
  3    PRODUCT_ID          NUMBER(6),
  4    WAREHOUSE_ID        NUMBER(3),
  5    QUANTITY_ON_HAND    NUMBER(8)
  6  )
  7  ORGANIZATION EXTERNAL
  8  (
  9    TYPE ORACLE_DATAPUMP
 10    DEFAULT DIRECTORY def_dir1
 11    LOCATION ('inv_p1_xt.dmp','inv_p2_xt.dmp')
 12  );
Table created.
SQL> SELECT * FROM inv_part_all_xt MINUS SELECT * FROM oe.inventories;
no rows selected
```

親トピック: ORACLE_DATAPUMPアクセス・ドライバを使用したデータのアンロードとロード

16.4 サポートされるデータ型

ORACLE_DATAPUMPアクセス・ドライバは、ロードおよびアンロード時に多くのデータ型を自動的に解決します。

外部表を使用してデータベース間でデータを移動する際には、次のような問題があります。

- 2つのプラットフォーム間で、データベースの文字セットとデータベースの各国語文字セットが異なる場合があります。
- 2つのデータベースで、プラットフォームのエンディアンが異なる場合があります。

ORACLE_DATAPUMPアクセス・ドライバは、これらの状況の一部を自動的に解消します。

次のデータ型は、ロード時およびアンロード時に自動的に変換されます。

- 文字(CHAR、NCHAR、VARCHAR2、NVARCHAR2)
- RAW
- NUMBER
- 日時
- BLOB
- CLOBおよびNCLOB
- ROWIDおよびUROWID

外部表でサポートされていないデータ型を使用するとエラーが返されます。サポートされていないデータ型であるLONGを使用した例を次に示します。

```
SQL> CREATE TABLE bad_datatype_xt
  2  (
  3    product_id             NUMBER(6),
  4    language_id            VARCHAR2(3),
  5    translated_name        NVARCHAR2(50),
  6    translated_description LONG
  7  )
  8  ORGANIZATION EXTERNAL
  9  (
 10    TYPE ORACLE_DATAPUMP
 11    DEFAULT DIRECTORY def_dir1
 12    LOCATION ('proddesc.dmp')
 13  );
  translated_description LONG
  *
```

```
ERROR at line 6:
ORA-30656: column type not supported on external organized table
```

親トピック: ORACLE_DATAPUMPアクセス・ドライバ

16.5 サポートされないデータ型

ORACLE_DATAPUMPアクセス・ドライバを使用すると、サポートされていないデータ型のデータをアンロードおよび再ロードできます。

外部表は、列に対して有効なすべてのデータ型のサブセットをサポートしています。特に、文字データ型(LONGを除く)、RAWデータ型、すべての数値データ型、およびすべての日付、タイムスタンプ、期間データ型をサポートしています。

ORACLE_DATAPUMPアクセス・ドライバを使用してデータをアンロードおよび再ロードできる、サポートされていないデータ型には、次のものが含まれます。

- BFILE
- LONGおよびLONG RAW
- FINALオブジェクト型
- FINALオブジェクト型の表

- BFILEデータ型のアンロードおよびロード
  BFILEデータ型には、ファイルのディレクトリ・オブジェクトとディレクトリ・オブジェクト内のファイル名という2つの情報が格納されます。
- LONGおよびLONG RAWデータ型のアンロード
  ORACLE_DATAPUMPアクセス・ドライバは、LONG列およびLONG RAW列のアンロードに使用できますが、データはLOBフィールドにのみ再ロードできます。
- FINALオブジェクト型を含む列のアンロードおよびロード
  FINAL型の列のオブジェクトは、オブジェクト型の各属性を外部表の列に移動することで移入されます。
- FINALオブジェクト型の表
  オブジェクト表には、表のすべての行を一意に識別するオブジェクト識別子があります。

親トピック: ORACLE_DATAPUMPアクセス・ドライバ

16.5.1 BFILEデータ型のアンロードおよびロード

BFILEデータ型には、ファイルのディレクトリ・オブジェクトとディレクトリ・オブジェクト内のファイル名という2つの情報が格納されます。

ORACLE_DATAPUMPアクセス・ドライバを使用して、ディレクトリ・オブジェクト名およびファイル名を外部表の2つの列に格納することにより、BFILE列をアンロードできます。DBMS_LOB.FILEGETNAMEプロシージャは両方の名前を返します。ただし、これはプロシージャであるため、SELECT文では使用できません。かわりに、2つのファンクションを使用します。最初のファンクションは、ディレクトリ・オブジェクト名を返し、2つ目のファンクションはファイル名を返します。

次のステップでは、BFILEデータ型のアンロードおよびロードの例を示します。

1. BFILE 列のディレクトリ・オブジェクトを抽出するファンクションを作成します。列がNULLの場合は、NULLが返されます。

```
SQL> CREATE FUNCTION get_dir_name (bf BFILE) RETURN VARCHAR2 IS
  2  DIR_ALIAS VARCHAR2(255);
  3  FILE_NAME VARCHAR2(255);
  4  BEGIN
  5    IF bf is NULL
  6    THEN
```

```
  7      RETURN NULL;
  8    ELSE
  9      DBMS_LOB.FILEGETNAME (bf, dir_alias, file_name);
 10      RETURN dir_alias;
 11    END IF;
 12  END;
 13  /
Function created.
```

2. BFILE列のファイル名を抽出するファンクションを作成します。

```
SQL> CREATE FUNCTION get_file_name (bf BFILE) RETURN VARCHAR2 is
  2  dir_alias VARCHAR2(255);
  3  file_name VARCHAR2(255);
  4  BEGIN
  5    IF bf is NULL
  6    THEN
  7      RETURN NULL;
  8    ELSE
  9      DBMS_LOB.FILEGETNAME (bf, dir_alias, file_name);
 10      RETURN file_name;
 11    END IF;
 12  END;
 13  /
Function created.
```

3. 次のように、BFILE列にNULL値を持つ行を追加できます。

```
SQL> INSERT INTO PRINT_MEDIA (product_id, ad_id, ad_graphic)
  2  VALUES (3515, 12001, NULL);
1 row created.
```

新しく作成したファンクションを使用して外部表を移入できます。BFILE列がNULLの場合、ファンクションはad_graphic_dir列およびad_graphic_file列をNULLに設定します。

4. print_media表からのデータを含める外部表を作成します。get_dir_nameおよびget_file_nameファンクションを使用して、BFILE列の構成要素を取得します。

```
SQL> CREATE TABLE print_media_xt
  2  ORGANIZATION EXTERNAL
  3  (
  4    TYPE oracle_datapump
  5    DEFAULT DIRECTORY def_dir1
  6    LOCATION ('pm_xt.dmp')
  7  ) AS
  8  SELECT product_id, ad_id,
  9         get_dir_name (ad_graphic) ad_graphic_dir,
 10         get_file_name(ad_graphic) ad_graphic_file
 11  FROM print_media;
Table created.
```

5. 外部表内のデータからBFILE列をロードするファンクションを作成します。外部表のad_graphic_dir列がNULLの場合、ファンクションはNULLを返します。

```
SQL> CREATE FUNCTION get_bfile (dir VARCHAR2, file VARCHAR2) RETURN
BFILE is
  2  bf BFILE;
  3  BEGIN
  4    IF dir IS NULL
  5    THEN
  6      RETURN NULL;
  7    ELSE
  8      RETURN BFILENAME(dir,file);
  9    END IF;
```

```
 10  END;
 11  /
Function created.
```

6. get_bfileファンクションを使用してBFILE列を含む新しい表を移入できます。

```
SQL> CREATE TABLE print_media_int AS
  2  SELECT product_id, ad_id,
  3         get_bfile (ad_graphic_dir, ad_graphic_file) ad_graphic
  4  FROM print_media_xt;
Table created.
```

7. 新しくロードされた表の列にあるデータと、print_media表の列のデータは一致している必要があります。

```
SQL> SELECT product_id, ad_id,
  2         get_dir_name(ad_graphic),
  3         get_file_name(ad_graphic)
  4  FROM print_media_int
  5  MINUS
  6  SELECT product_id, ad_id,
  7         get_dir_name(ad_graphic),
  8         get_file_name(ad_graphic)
  9  FROM print_media;
no rows selected
```

親トピック: サポートされていないデータ型

16.5.2 LONGおよびLONG RAWデータ型のアンロード

ORACLE_DATAPUMPアクセス・ドライバは、LONG列およびLONG RAW列のアンロードに使用できますが、データはLOBフィールドにのみ再ロードできます。

次のステップは、LONGおよびLONG RAWデータ型のアンロードの例を示します。

1. アンロードする表にLONG列またはLONG RAW列が含まれる場合、外部表の対応する列を、LONG列に対してはCLOBを、LONG RAW列に対してはBLOBを定義します。

```
SQL> CREATE TABLE long_tab
  2  (
  3    key                    SMALLINT,
  4    description            LONG
  5  );
Table created.
SQL> INSERT INTO long_tab VALUES (1, 'Description Text');
1 row created.
```

2. これで、LONG列からのデータを含めたCLOB列を含む外部表を作成できます。外部表をロードするときには、LONG列をCLOBに変換するためにTO_LOB演算子が使用されます。

```
SQL> CREATE TABLE long_tab_xt
  2  ORGANIZATION EXTERNAL
  3  (
  4    TYPE ORACLE_DATAPUMP
  5    DEFAULT DIRECTORY def_dir1
  6    LOCATION ('long_tab_xt.dmp')
  7  )
  8  AS SELECT key, TO_LOB(description) description FROM long_tab;
Table created.
```

3. 外部表のデータを使用して、アンロードされている表と同じ表を別に作成できますが、作成した表にはLONG列のかわりにLOB列が含まれます。

```
SQL> CREATE TABLE lob_tab
  2  AS SELECT * from long_tab_xt;
Table created.
```

4. 表が正しく作成されたことを確認します。

```
SQL> SELECT * FROM lob_tab;
       KEY  DESCRIPTION
--------------------------------------------------------------------------------
         1  Description Text
```

親トピック: サポートされていないデータ型

16.5.3 FINALオブジェクト型を含む列のアンロードおよびロード

FINAL型の列のオブジェクトは、オブジェクト型の各属性を外部表の列に移動することで移入されます。

また、外部表には、列オブジェクトがアトミックNULLであるかどうかを追跡するための、新規の列が必要です。次のステップでは、FINALオブジェクト型を含む列のアンロードおよびロードの方法を示します。

1. 次の例では、ソース表のwarehouse列がアトミックNULLであるかを追跡するために外部表のwarehouse列を使用します。

```
SQL> CREATE TABLE inventories_obj_xt
  2  ORGANIZATION EXTERNAL
  3  (
  4    TYPE ORACLE_DATAPUMP
  5    DEFAULT DIRECTORY def_dir1
  6    LOCATION ('inv_obj_xt.dmp')
  7  )
  8  AS
  9  SELECT oi.product_id,
 10         DECODE (oi.warehouse, NULL, 0, 1) warehouse,
 11         oi.warehouse.location_id location_id,
 12         oi.warehouse.warehouse_id warehouse_id,
 13         oi.warehouse.warehouse_name warehouse_name,
 14         oi.quantity_on_hand
 15  FROM oc_inventories oi;
Table created.
```

これで、オブジェクト型の属性を含む外部表の列を、その型の列をロードする際の型コンストラクタのファンクションの引数として使用できます。外部表のwarehouse列は、オブジェクトのコンストラクタ・ファンクションをコールするか、列をNULLに設定するかを判断するために使用されます。

2. oc_inventoriesビューと同様の新規の内部表をロードします。(WHERE 1=0句を使用すると、古い表と同様の表を作成しますが、データは古い表から新しい表にはコピーされません。)

```
SQL> CREATE TABLE oc_inventories_2 AS SELECT * FROM oc_inventories
WHERE 1 = 0;
Table created.
SQL> INSERT INTO oc_inventories_2
  2  SELECT product_id,
  3         DECODE (warehouse, 0, NULL,
  4                 warehouse_typ(warehouse_id, warehouse_name,
  5                 location_id)), quantity_on_hand
  6  FROM inventories_obj_xt;
1112 rows created.
```

親トピック: サポートされていないデータ型

16.5.4 FINALオブジェクト型の表

オブジェクト表には、表のすべての行を一意に識別するオブジェクト識別子があります。

そのため次のような問題があります。

- オブジェクト識別子をアンロードおよび再ロードする必要がない場合、外部表にはオブジェクト表に対する型の属性のフィールドのみを含める必要があります。
- オブジェクト識別子(OID)をアンロードおよび再ロードする必要があり、表のOIDが表の1つ以上のフィールドである場合(主キー・ベースOIDとも呼ばれる)、外部表には表の型のすべての属性に対して1つの列が存在します。
- OIDをアンロードする必要があり、表のOIDがシステム生成である場合、手順はより複雑になります。システム生成OIDを保持するには、型の属性の他に別の列を作成する必要があります。

次のステップは、前述の最後の状況の例を示します。

1. システム生成OIDを持つ型の表を作成します。

```
SQL> CREATE TYPE person AS OBJECT (name varchar2(20)) NOT FINAL
  2  /
Type created.
SQL> CREATE TABLE people OF person;
Table created.
SQL> INSERT INTO people VALUES ('Euclid');
1 row created.
```

2. システム生成OIDを含む列を保持するために、OID列を使用する外部表を作成します。

```
SQL> CREATE TABLE people_xt
  2  ORGANIZATION EXTERNAL
  3  (
  4    TYPE ORACLE_DATAPUMP
  5    DEFAULT DIRECTORY def_dir1
  6    LOCATION ('people.dmp')
  7  )
  8  AS SELECT SYS_NC_OID$ oid, name FROM people;
Table created.
```

3. システム生成OIDを持つ同じ型の別の表を作成します。次にINSERT文を実行して、新しい表に古い表からアンロードされたデータをロードします。

```
SQL> CREATE TABLE people2 OF person;
Table created.
SQL>
SQL> INSERT INTO people2 (SYS_NC_OID$, SYS_NC_ROWINFO$)
  2  SELECT oid, person(name) FROM people_xt;
1 row created.
SQL>
SQL> SELECT SYS_NC_OID$, name FROM people
  2  MINUS
  3  SELECT SYS_NC_OID$, name FROM people2;
no rows selected
```

親トピック: サポートされていないデータ型

16.6 ORACLE_DATAPUMPアクセス・ドライバを使用する際のパフォーマンスのヒント

ORACLE_DATAPUMPアクセス・ドライバのパフォーマンスを向上させる方法について説明します。

パフォーマンスを監視する場合、最も重要なことは、ロードの経過時間の測定です。また、CPU使用量、メモリー使用量およびI/O率の測定も重要です。

並列度を増減することによって、パフォーマンスを変更できます。並列度は、データ・ファイルの処理に起動できるアクセス・ドライバの数を示します。並列度によって、リソース使用率を低くした遅いロードと、すべてのリソースを使用した速いロードを選択できます。アクセス・ドライバは、アクセス・ドライバ専用に使用するリソース量を判断できないため、自動的にはチューニングされません。

アクセス・ドライバは、パフォーマンスを向上させるために大きなI/Oバッファを使用します。共有サーバーを使用するデータベースでは、アクセス・ドライバが使用するすべてのメモリーはSystem Global Area(SGA)から割り当てられます。そのため、共有サーバー上の外部表を使用する際には注意が必要です。

親トピック: ORACLE_DATAPUMPアクセス・ドライバ

16.7 ORACLE_DATAPUMPアクセス・ドライバ使用時の制限事項

ORACLE_DATAPUMPアクセス・ドライバを使用した外部表へのアクセスに適用される制限事項に注意してください。

外部表でのORACLE_DATAPUMPアクセス・ドライバの使用に適用される制限事項は、次のとおりです。

- 暗号化された列: 暗号化された列がある外部表のエクスポートとインポートは、サポートされていません。
- 列処理: デフォルトでは、外部表の機能は、外部表に定義されたすべての列をフェッチします。これによって、すべての問合せに対し、一貫した結果が保証されます。ただし、パフォーマンスを考慮して、外部表内で参照される列だけを処理することができます。これによって、データ変換の量および問合せの実行で必要なデータ処理の量を最小限にします。この場合、データ型変換エラーが発生した列を含んでいたために拒否された行も、その列が参照されないかぎり、別の問合せでは拒否されません。ALTER TABLEコマンドを使用すると、この列処理動作を変更できます。
- LONG列: 外部表は、LONG列にデータをロードできません。
- ロード時におけるバイト順序マークの処理。データ・ファイル文字セットがUTF8またはUTF16の外部表ロードでは、バイト順序マークの確認は抑止できません。バイト順序マークの確認は、データ・ファイルの先頭にバイト順序マークのエンコーディングと一致するバイナリ・データが含まれている場合にのみ抑止する必要があります。(SQL*Loaderを使用したロードでは、バイト順序マークの確認を抑止できます。)バイト順序マークを確認するということは、バイト順序マークがデータ・ファイル内に存在する必要があるという意味ではないことに注意してください。バイト順序マークがない場合は、サーバー・プラットフォームのバイト順序が使用されます。
- バックスラッシュ・エスケープ文字: 外部表機能では、文字列内のバックスラッシュ(¥)エスケープ文字の使用をサポートしません。
- 予約語: 外部表アクセス・パラメータにおいて列名や表名などの識別子が指定される場合、特定の値はアクセス・パラメータ・パーサーによって予約語とみなされます。予約語を識別子として使用するには、二重引用符で囲む必要があります。

関連トピック

- バックスラッシュ・エスケープ文字の使用

親トピック: ORACLE_DATAPUMPアクセス・ドライバ

16.8 ORACLE_DATAPUMPアクセス・ドライバの予約語

ORACLE_DATAPUMPアクセス・ドライバによって予約されている識別子で語を使用する場合は、二重引用符で囲む必要があります。

外部表アクセス・パラメータにおいて列名や表名などの識別子が指定される場合、特定の値はアクセス・パラメータ・パーサーによって予約語とみなされます。予約語を識別子として使用するには、二重引用符で囲む必要があります。

ORACLE_DATAPUMPアクセス・ドライバの予約語は、次のとおりです。

- BADFILE
- COMPATIBLE
- COMPRESSION
- DATAPUMP
- DEBUG
- ENCRYPTION
- INTERNAL
- JOB
- LATEST
- LOGFILE
- NOBADFILE
- NOLOGFILE
- PARALLEL
- TABLE
- VERSION
- WORKERID

親トピック: ORACLE_DATAPUMPアクセス・ドライバ

17 ORACLE_HDFSおよびORACLE_HIVEアクセス・ドライバ

外部表を使用すると、Oracle Databaseの表に格納されたデータであるかのように、HadoopおよびHiveクライアント上のHDFSおよびHive表に格納されているデータにアクセスできます。

ORACLE_HDFSまたはORACLE_HIVEアクセス・ドライバを使用する外部表を作成するために使用するプロパティは、SQL文CREATE TABLE ORGANIZATION EXTERNALでACCESS PARAMETERSのopaque_format_spec句の中に指定します。

- プロパティを指定するための構文ルール
  プロパティは、SQL CREATE TABLEのACCESS PARAMETERS句および構成ファイル内に、キーワード/値のペアを使用して設定します。
- ORACLE_HDFSアクセス・パラメータ
  ORACLE_HDFSアクセス・ドライバのアクセス・パラメータは、HDFS内のデータを特定し、それに基づいてHive表を作成するために必要なメタデータを提供します。
- ORACLE_HIVEアクセス・パラメータ
  ORACLE_HIVEは、Hiveカタログから外部データ・ソースに関するメタデータを取得します。
- com.oracle.bigdataパラメータの説明
  com.oracle.bigdataパラメータは、ORACLE_HIVE、ORACLE_HDFSおよびORACLE_BIGDATAアクセス・ドライバによって使用されます。

関連トピック

- 外部表の例

親トピック: 外部表

17.1 プロパティを指定するための構文ルール

プロパティは、SQL CREATE TABLEのACCESS PARAMETERS句および構成ファイル内に、キーワード/値のペアを使用して設定します。

構文は次のルールに従う必要があります。

- 各キーワード/値のペアの形式は、**キーワード**、コロンまたは等号と、**値**です。有効なキーワード/値のペアを次に示します。

  ```
  keyword=value
  keyword:value
  ```

  値は、セパレータの後の最初の非空白文字から行末までのすべてになります。セパレータと値の間の空白は無視されます。値の後の空白は保持されます。

- プロパティ定義は1行または複数行にできます。
- 行の終了文字は、LF、改行または改行とLFの組合せです。
- 1つのプロパティ定義が複数行にわたる場合は、最終行以外の行の終了文字の前にバックスラッシュ(エスケープ文字)を付けます。この例では、Keyword1プロパティの値はValue part 1 Value part 2 Value part 3で

す。

```
Keyword1=  Value part 1 ¥
           Value part 2 ¥
           Value part 3
```

- 先頭の空白の各物理行を削除して行を連結することで、**論理行**を作成できます。パーサーによって、プロパティ名および値が論理行から抽出されます。
- 文字の前に代入を示すバックスラッシュ(エスケープ文字)を付けることで、プロパティ名またはプロパティ値に特殊文字を埋め込むことができます。次の表で、特殊文字について説明します。

表17-1 プロパティの特殊文字

エスケープ・シーケンス	文字
¥b	バックスペース(¥u0008)
¥t	水平タブ(¥u0009)
¥n	LF (¥u000a)
¥f	改ページ(¥u000c)
¥r	改行(¥u000d)
¥"	二重引用符(¥u0022)
¥'	一重引用符(¥u0027)
¥¥	バックスラッシュ(¥u005c) 複数のバックスラッシュが行末にある場合、パーサーは奇数のバックスラッシュに対してのみ値を次の行に継続します。
¥uxxxx	2 バイト、ビッグ・エンディアンの Unicode コード・ポイント。 1 文字に 2 コード・ポイント(4 バイト)が必要な場合、パーサーは 2 番目のコード・ポイントに¥uを要求します。

親トピック: ORACLE_HDFSおよびORACLE_HIVEアクセス・ドライバ

17.2 ORACLE_HDFSアクセス・パラメータ

ORACLE_HDFSアクセス・ドライバのアクセス・パラメータは、HDFS内のデータを特定し、それに基づいてHive表を作成するために必要なメタデータを提供します。

- ORACLE_HDFSのデフォルトのパラメータ設定
  ORACLE_HDFSのデフォルトのパラメータ設定について説明します。
- ORACLE_HDFSのオプションのパラメータ設定
  ORACLE_HDFSでは、オプションのcom.oracle.bigdataパラメータがサポートされており、opaque_format_spec句で指定できます。

親トピック: ORACLE_HDFSおよびORACLE_HIVEアクセス・ドライバ

17.2.1 ORACLE_HDFSのデフォルトのパラメータ設定

ORACLE_HDFSのデフォルトのパラメータ設定について説明します。

CREATE TABLE文からすべてのアクセス・パラメータを省略した場合、ORACLE_HDFSは次のデフォルト値を使用します。

```
com.oracle.bigdata.rowformat=DELIMITED
com.oracle.bigdata.fileformat=TEXTFILE
com.oracle.bigdata.overflow={"action":"error"}
com.oracle.bigdata.erroropt={"action":"setnull"}
```

親トピック: ORACLE_HDFSアクセス・パラメータ

17.2.2 ORACLE_HDFSのオプションのパラメータ設定

ORACLE_HDFSは、オプションのcom.oracle.bigdataパラメータをサポートしています。これらのパラメータは、opaque_format_spec句で指定できます。

具体的には次のとおりです。

- com.oracle.bigdata.colmap
- com.oracle.bigdata.erroropt
- com.oracle.bigdata.fields
- com.oracle.bigdata.fileformat
- com.oracle.bigdata.log.exec
- com.oracle.bigdata.log.qc
- com.oracle.bigdata.overflow
- com.oracle.bigdata.rowformat

次の例に、複数のアクセス・パラメータが設定されているCREATE TABLE文を示します。

例17-1 ORACLE_HDFSの複数のアクセス・パラメータの設定

```
CREATE TABLE ORDER (CUST_NUM VARCHAR2(10),
                    ORDER_NUM VARCHAR2(20),
                    ORDER_DATE DATE,
                    ITEM_CNT NUMBER,
                    DESCRIPTION VARCHAR2(100),
                    ORDER_TOTAL (NUMBER8,2)) ORGANIZATION EXTERNAL
             (TYPE ORACLE_HDFS
 ACCESS PARAMETERS (
  com.oracle.bigdata.fields: (CUST_NUM,              ¥
                              ORDER_NUM,             ¥
                              ORDER_DATE,            ¥
                              ORDER_LINE_ITEM_COUNT, ¥
                              DESCRIPTION,           ¥
                              ORDER_TOTAL)
  com.oracle.bigdata.colMap:     {"col":"item_cnt", ¥
                                  "field":"order_line_item_count"}
```

```
 com.oracle.bigdata.overflow:   {"action":"TRUNCATE", ¥
                                 "col":"DESCRIPTION"}
 com.oracle.bigdata.errorOpt:  [{"action":"replace", ¥
                                 "value":"INVALID NUM", ¥
                                 "col":["CUST_NUM","ORDER_NUM"]} , ¥
                                {"action":"reject", ¥
                                 "col":"ORDER_TOTAL"}]
)
LOCATION ("hdfs:/usr/cust/summary/*"));
```

親トピック: ORACLE_HDFSアクセス・パラメータ

17.3 ORACLE_HIVEアクセス・パラメータ

ORACLE_HIVEは、Hiveカタログから外部データ・ソースに関するメタデータを取得します。

通常、外部データ表の列へのHiveデータのデフォルト・マッピングは適切です。ただし、状況によっては、特殊なパラメータ設定が必要になることや、独自の理由でデフォルト値を上書きすることがあります。

- ORACLE_HIVEのデフォルトのパラメータ設定
  ORACLE_HIVEのデフォルトのパラメータ設定について説明します。
- ORACLE_HIVEのオプションのパラメータ設定
  ORACLE_HIVEでは、オプションのcom.oracle.bigdataパラメータがサポートされており、opaque_format_spec句で指定できます。

親トピック: ORACLE_HDFSおよびORACLE_HIVEアクセス・ドライバ

17.3.1 ORACLE_HIVEのデフォルトのパラメータ設定

ORACLE_HIVEのデフォルトのパラメータ設定について説明します。

CREATE TABLE文からすべてのアクセス・パラメータを省略した場合、ORACLE_HIVEでは次のデフォルト値を使用します。

```
com.oracle.bigdata.tablename=name of external table
com.oracle.bigdata.overflow={"action":"error"}
com.oracle.bigdata.erroropt={"action":"setnull"}
```

親トピック: ORACLE_HIVEアクセス・パラメータ

17.3.2 ORACLE_HIVEのオプションのパラメータ設定

ORACLE_HIVEは、オプションのcom.oracle.bigdataパラメータをサポートしています。これらのパラメータは、opaque_format_spec句で指定できます。

具体的には次のとおりです。

- com.oracle.bigdata.colmap
- com.oracle.bigdata.erroropt
- com.oracle.bigdata.log.exec
- com.oracle.bigdata.log.qc
- com.oracle.bigdata.overflow
- com.oracle.bigdata.tablename

次の例に、複数のアクセス・パラメータが設定されているCREATE TABLE文を示します。

例17-2 ORACLE_HIVEの複数のアクセス・パラメータの設定

```
CREATE TABLE ORDER (cust_num VARCHAR2(10),
```

```
                    order_num VARCHAR2(20),
                    order_date DATE,
                    item_cnt NUMBER,
                    description VARCHAR2(100),
                    order_total (NUMBER8,2)) ORGANIZATION EXTERNAL
(TYPE oracle_hive
 ACCESS PARAMETERS (
  com.oracle.bigdata.tableName:  order_db.order_summary
  com.oracle.bigdata.colMap:     {"col":"ITEM_CNT", ¥
                                  "field":"order_line_item_count"}
  com.oracle.bigdata.overflow:   {"action":"ERROR", ¥
                                  "col":"DESCRIPTION"}
  com.oracle.bigdata.errorOpt:   [{"action":"replace", ¥
                                    "value":"INV_NUM" , ¥
                                    "col":["CUST_NUM","ORDER_NUM"]} ,¥
                                  {"action":"reject", ¥
                                    "col":"ORDER_TOTAL}]
));
```

親トピック: ORACLE_HIVEアクセス・パラメータ

17.4 com.oracle.bigdataパラメータの説明

com.oracle.bigdataパラメータは、ORACLE_HIVE、ORACLE_HDFSおよびORACLE_BIGDATAアクセス・ドライバによって使用されます。

- com.oracle.bigdata.colmap
  ソース・データの列をOracle外部表の列にマップします。
- com.oracle.bigdata.erroropt
  列の値の計算中に発生するエラーを処理する方法を示します。
- com.oracle.bigdata.fields
  データ・ソースのフィールド名およびデータ型をリストします。
- com.oracle.bigdata.fileformat
  ORACLE_HDFSによって生成されたHive表のROW FORMAT句に基づいて、データ・ソースの行形式を説明します。
- com.oracle.bigdata.log.exec
  アクセス・ドライバがCDH上のパラレル処理として実行されている場合、問合せのCコードによって生成されたログ・ファイルを生成する方法を指定します。
- com.oracle.bigdata.log.qc
  問合せのログ・ファイルをアクセス・ドライバで生成する方法を指定します。
- com.oracle.bigdata.overflow
  外部表の列に対して長すぎる文字列データの処理方法を示します。
- com.oracle.bigdata.rowformat
  アクセス・ドライバがファイル内のレコードからフィールドを抽出するために必要な情報を提供します。
- com.oracle.bigdata.tablename
  ソース・データを含む表の名前を識別します。

親トピック: ORACLE_HDFSおよびORACLE_HIVEアクセス・ドライバ

17.4.1 com.oracle.bigdata.colmap

ソース・データの列をOracle外部表の列にマップします。

1つまたは複数の列マッピングのペアを定義できます。ソース・フィールド名がOracle列名の最大長を超える場合、または外部

表で別の列名を使用する場合、このプロパティを使用します。

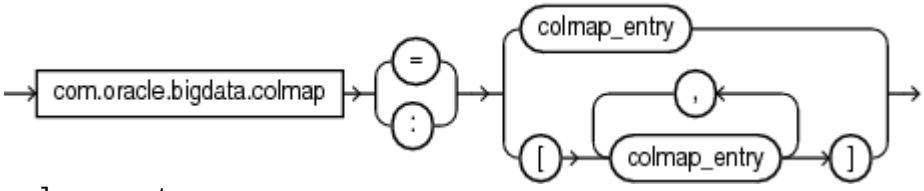
デフォルト値

Hive列と同じ名前を持つ外部表の列

構文

キーワード/値のペアを持つJSONドキュメントを次の図に示します。

colmap ::=



colmap_entry ::=

セマンティクス

"col":name

"col": キーワードは小文字にし、引用符で囲む必要があります。

name: Oracle外部表の列の名前。大/小文字が区別され、引用符で囲む必要があります。

"field":name

"field": キーワードは小文字にし、引用符で囲む必要があります。

name: データ・ソースのフィールドの名前。名前は大/小文字を区別しませんが、引用符で囲む必要があります。「プロパティを指定するための構文ルール」を参照してください。

例

この例では、ORDER_LINE_ITEM_COUNTという名前のHive列をITEM_CNTという名前のOracle列にマップします。

```
com.oracle.bigdata.colMap={"col":"ITEM_CNT", ¥
                           "field":"order_line_item_count"}
```

次の例は、複数の列のマッピングを示しています。

```
com.oracle.bigdata.colmap:[{"col":"KOL1",
"field":"PROJECT_NAME"},{ "col":"KOL2","field":"wsdl_name"},{"col":"KOL3",
"field":"method"}]
```

親トピック: com.oracle.bigdataパラメータの説明

17.4.2 com.oracle.bigdata.erroropt

列の値の計算中に発生するエラーを処理する方法を示します。

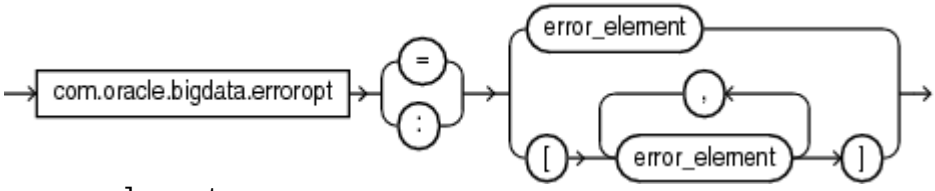
デフォルト値

{"action":"setnull"}

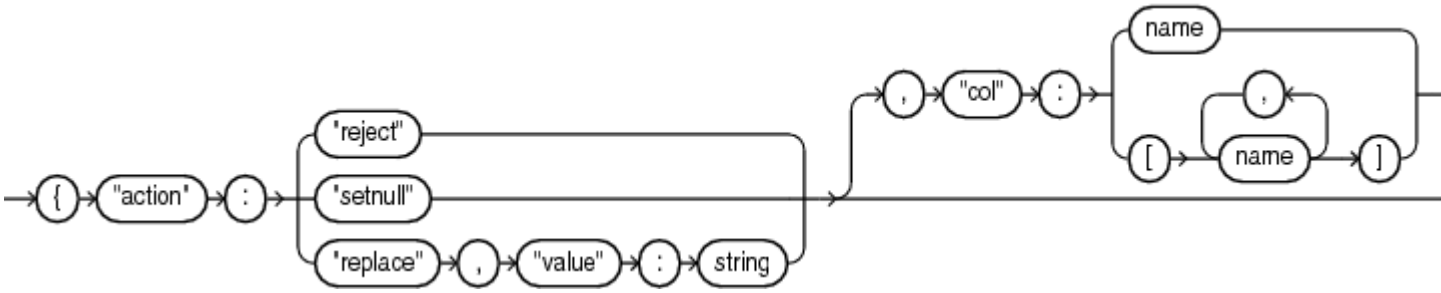
構文

キーワード/値のペアを持つJSONドキュメントを次の図に示します。

erroropt ::=



error_element ::=



セマンティクス

"action"、"reject"、"setnull"、"replace"、"value"および"col"キーワードは小文字を使用して、引用符で囲む必要があります。「プロパティを指定するための構文ルール」を参照してください。

"action":value

value: 次のキーワードのいずれかになります。

- "reject": 行をロードしません。
- "setnull": 列をNULLに設定します。
- "replace": 列を指定された値に設定します。

"value":string

string: 外部表の不正な値を置換します。引用符で囲む必要があります。

"col":name

name: 外部表の列を識別します。列名は大/小文字が区別され、引用符で囲む必要があり、1回のみ指定できます。

例

この例では、Hive値によってエラーが発生した場合、CUST_NUMまたはORDER_NUM列の値をINVALIDに設定します。その他の列では、エラーによってHive値が拒否されます。

```
com.oracle.bigdata.errorOpt: {"action":"replace",¥
                              "value":"INVALID", ¥
                              "col":["CUST_NUM","ORDER_NUM"]
```

親トピック: com.oracle.bigdataパラメータの説明

17.4.3 com.oracle.bigdata.fields

データ・ソースのフィールド名およびデータ型をリストします。

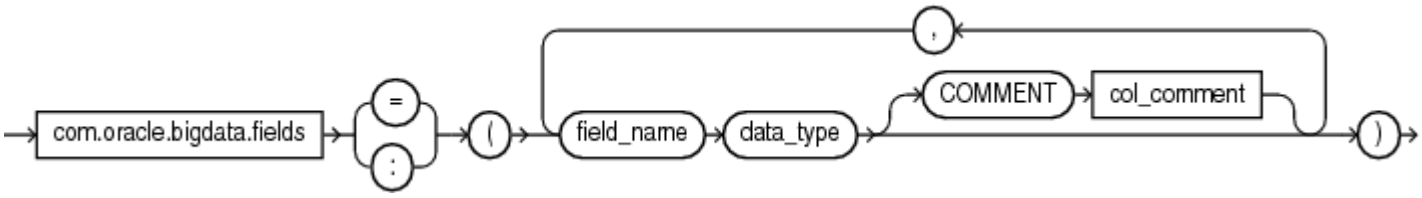
デフォルト値

未定義

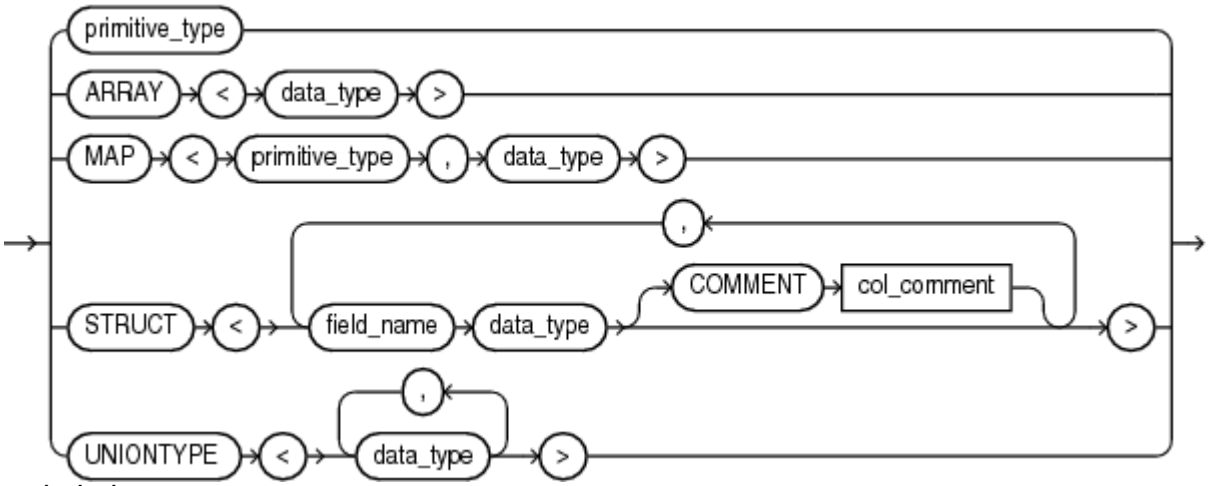
構文

キーワード/値のペアを持つJSONドキュメントを次の図に示します。

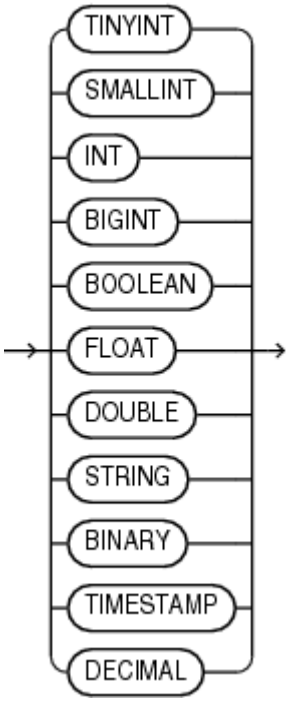
fields ::=



data_type ::=



primitive_type ::=



セマンティクス

構文は、Hive表のフィールド・リストと同じです。フィールド・リストを複数行に分割する場合、バックスラッシュを使用して改行文字をエスケープする必要があります。

field_name

Hiveフィールドの名前。英数字およびアンダースコア(_)のみ使用します。最大の長さは、128文字です。フィールド名の大/小文字は区別されます。

data_type

(オプション) Hiveフィールドのデータ型。デフォルトはSTRINGです。文字セットはUTF8である必要があります。

データ型は複合またはプリミティブになります。

Hive複合データ型

- ARRAY: 索引付け可能なリスト
- MAP: キーと値のタプル
- STRUCT: 要素のリスト
- UNIONTYPE: 複数のデータ型

Hiveプリミティブ・データ型

- INT: 4バイトの整数
- BIGINT: 8バイトの整数
- SMALLINT: 2バイトの整数
- TINYINT: 1バイトの整数
- BOOLEAN: TRUEまたはFALSE
- FLOAT: 単精度
- DOUBLE: 倍精度
- STRING: 文字列

関連項目:

*Apache Hive Language Manual*のデータ型に関する項を参照してください。

https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Types

COMMENT col_comment

一重引用符で囲まれた文字列リテラル。Hive表のメタデータとして格納されます(TBLPROPERTIESのcommentプロパティ)。

親トピック: com.oracle.bigdataパラメータの説明

17.4.4 com.oracle.bigdata.fileformat

ORACLE_HDFSによって生成されたHive表のROW FORMAT句に基づいて、データ・ソースの行形式を説明します。

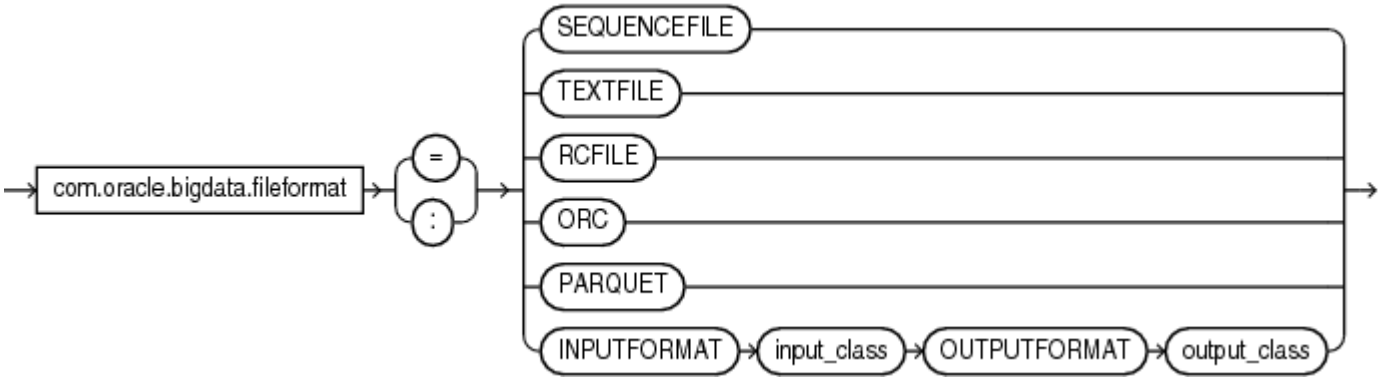
デフォルト値

TEXTFILE

構文

キーワード/値のペアを持つJSONドキュメントを次の図に示します。

fileformat::=



セマンティクス

ORC

最適化された行列ファイル形式

PARQUET

列指向、バイナリ・ファイル形式

RCFILE

レコード列ファイル形式

SEQUENCEFILE

圧縮ファイル形式

TEXTFILE

プレーン・テキスト・ファイル形式

INPUTFORMAT

データ・ファイルからレコードを抽出できるJavaクラスを識別します。

OUTPUTFORMAT

出力レコードを目的の形式に設定できるJavaクラスを識別します。

親トピック: com.oracle.bigdataパラメータの説明

17.4.5 com.oracle.bigdata.log.exec

アクセス・ドライバがCDH上のパラレル処理として実行されている場合、問合せのCコードによって生成されたログ・ファイルを生成する方法を指定します。

アクセス・ドライバは、Hadoopクラスタ・ノード上で実行されている場合、ログ・ファイルの作成または書込みを実行しません。書込みはパラレル問合せプロセスによって実行されます。Javaコードからのログ・ファイルは`log4j`プロパティによって制御されます。このプロパティは、構成ファイルまたはアクセス・パラメータで指定されます。

関連項目:

bigdata-log4j.properties

デフォルト値

未定義(ロギングなし)

構文

```
[directory_object:]file_name_template
```

セマンティクス

```
directory_object
```

ログ・ファイルが作成されるHadoopクラスタ上のHDFSパスのOracleディレクトリ・オブジェクト。

`file_name_template`

ファイル名の生成に使用する文字列。この表では、テンプレートで使用できるオプションの変数を示します。

表17-2 com.oracle.bigdata.log.execの変数

変数	値
%p	オペレーティング・システム・プロセス識別子(PID)
%a	プロセスを一意に識別する番号
%%	パーセント記号(%)

例

次の例では、PIDおよび一意の番号を含むログ・ファイル名(`xtlogp_hive14_3413_57`など)を生成します。

```
com.oracle.bigdata.log.exec= xtlogp_hive14_%p_%a
```

親トピック: com.oracle.bigdataパラメータの説明

17.4.6 com.oracle.bigdata.log.qc

アクセス・ドライバが問合せのログ・ファイルを生成する方法を指定します。

デフォルト値

未定義(ロギングなし)

構文

```
[directory_object:]file_name_template
```

セマンティクス

`directory_object`

ログ・ファイルが書き込まれるパスを指すOracleディレクトリ・オブジェクトの名前。この値を省略すると、ログは外部表のデフォルト・ディレクトリに書き込まれます。

`file_name_template`

ファイル名の生成に使用する文字列。表17-3に、文字列で使用できるオプションの変数を示します。

表17-3 com.oracle.bigdata.log.qcの変数

変数	値

変数	値
%p	オペレーティング・システム・プロセス識別子(PID)
%%	パーセント記号(%)

例

この例では、PIDおよびパーセント記号を含むログ・ファイル名(`xtlogp_hive213459_%`など)を作成します。

```
com.oracle.bigdata.log.qc= xtlogp_hive21%p_%%
```

親トピック: com.oracle.bigdataパラメータの説明

17.4.7 com.oracle.bigdata.overflow

外部表の列に対して長すぎる文字列データの処理方法を示します。

データ・ソースは文字またはバイナリにできます。Hiveの場合、データ・ソースは`STRUCT`、`UNIONTYPES`、`MAP`または`ARRAY`にすることもできます。

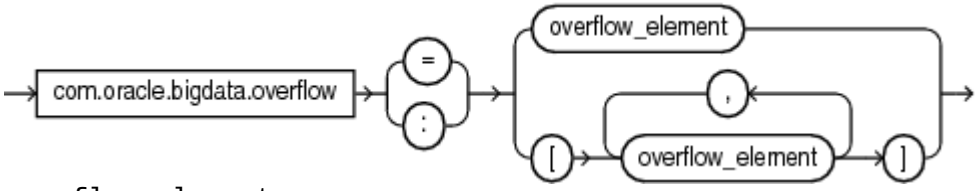
デフォルト値

`{"action":"error"}`

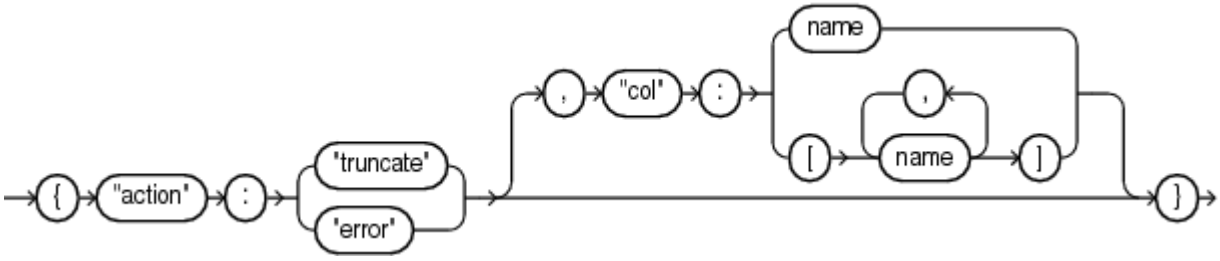
構文

キーワード/値のペアを持つJSONドキュメントを次の図に示します。

`overflow ::=`



`overflow_element ::=`



セマンティクス

`"action"`、`"truncate"`、`"error"`および`"col"`タグは小文字にし、引用符で囲む必要があります。「プロパティを指定するための構文ルール」を参照してください。

`"action":value`

`"action"`の値は次のキーワードのいずれかになります。

- `truncate:` 列に収まるようにデータを短くします。
- `error:` エラーをスローします。プロパティcom.oracle.bigdata.erroroptは、エラーの結果を制御します。

`"col":name`

name: 外部表の列を識別します。名前は大/小文字が区別され、引用符で囲む必要があります。

例

この例では、DESCRIPTION列のソース・データが列幅を超える場合、ソース・データを切り捨てます。

```
com.oracle.bigdata.overflow={"action":"truncate", ¥
                             "col":"DESCRIPTION"}
```

親トピック: com.oracle.bigdataパラメータの説明

17.4.8 com.oracle.bigdata.rowformat

アクセス・ドライバがファイル内のレコードからフィールドを抽出するために必要な情報を提供します。

> 注意:
>
> この属性値は使用しないでください com.oracle.bigdata.rowformat は、"type ORACLE_LOADER" を使用する従来型の外部表のアクセス・パラメータ構文とは関連していません。FIELDS、TERMINATED のようなキーワードなど、両方の句に出現するものもありますが、ネーミングの一致は偶然であり、機能が共通するということではありません。com.oracle.bigdata.rowformat アクセス・パラメータは、デフォルトの Hive SerDe に変更を加えることなく渡されます。行から列を抽出するために使用される Hive SerDe は意図的に制限されます。複雑なケースは、特別な SerDe を使用して処理されます。

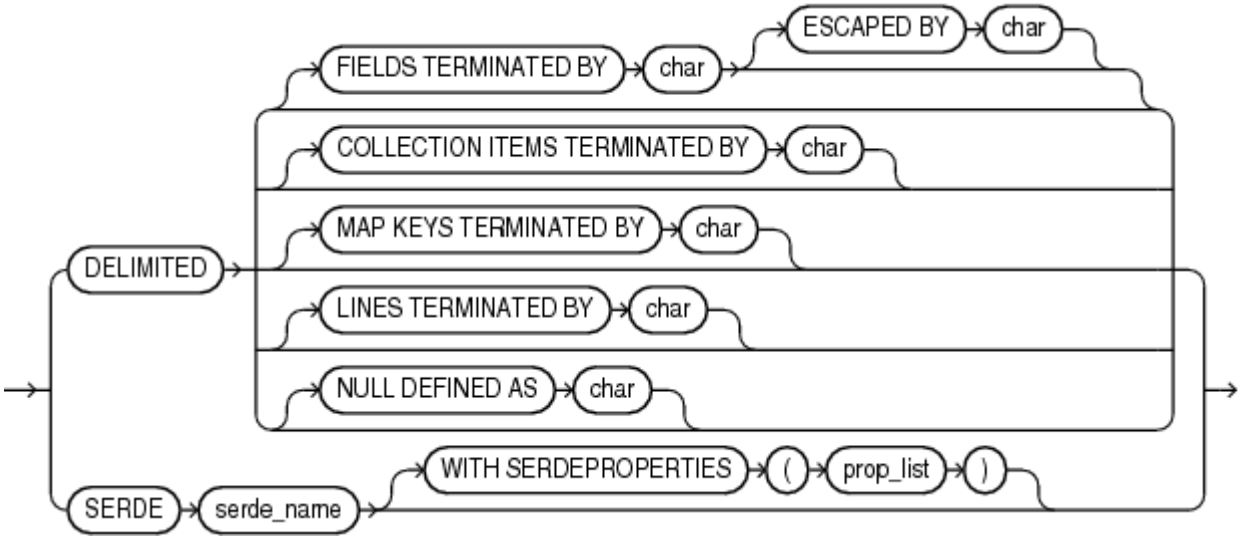
デフォルト値

DELIMITED

構文

キーワード/値のペアを持つJSONドキュメントを次の図に示します。

rowformat ::=



セマンティクス

DELIMITED

レコード内のフィールドを区切るために使用する文字を示します。

- FIELDS TERMINATED BY: レコード内の各フィールドを区切る文字。オプションのESCAPED BY文字は、フィー

ルド値内に指定する場合、デリミタ文字の前に付けます。

- `COLLECTION ITEMS TERMINATED BY`: 配列要素の終了を示す文字。列がコレクションまたはネストされたレコードである場合に使用します。この場合、結果の値はJSON配列になります。
- `MAP KEYS TERMINATED BY`: MAPフィールドのエントリの終了を示す文字。列がコレクションまたはネストされたレコードである場合に使用します。結果の値はJSONオブジェクトになります。
- `LINES TERMINATED BY`: レコードの終了を示す文字。
- `NULL DEFINED AS`: null値を示す文字。

`SERDE`

アクセス・ドライバが必要とするシリアライザ/デシリアライザ(SerDe)のデータおよびプロパティを解析できるSerDeを識別します。

例

この例では、Avroコンテナ・ファイルにSerDeを指定します。

```
com.oracle.bigdata.rowformat:
   SERDE'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
```

次の例では、正規表現を含むファイルのSerDeを指定します。

```
com.oracle.bigdata.rowformat=¥
    SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe' ¥
    WITH SERDEPROPERTIES ¥
    ("input.regex" = "(¥¥¥¥d{6}) (¥¥¥¥d{5}) (.{29}) .*")
```

親トピック: com.oracle.bigdataパラメータの説明

17.4.9 com.oracle.bigdata.tablename

ソース・データを含む表の名前を識別します。

デフォルト値

```
DEFAULT.external_table_name
```

構文

```
[hive_database_name.]table_name
```

セマンティクス

`hive_database_name`および`table_name`の最大長は128 UTF-8文字(512バイト)です。

`hive_database_name`: ソース・データが存在しているHiveデータベース。`DEFAULT`はHive初期データベースの名前です。

`table_name`: データが含まれているHive表。`table_name`を省略すると、`ORACLE_HIVE`は外部表と同じ名前のHive表を検索します。表名の大/小文字は区別されます。

例

この設定は、ソース・データがHiveの`ORDER_DB`データベース内の`ORDER_SUMMARY`という表にあることを示します。

```
com.oracle.bigdata.tablename ORDER_DB.ORDER_SUMMARY
```

親トピック: com.oracle.bigdataパラメータの説明

18 外部表の例

これらの例から、Oracle DatabaseおよびBig Dataのデータの問合せにアクセス・ドライバのORACLE_LOADER、ORACLE_DATAPUMP、ORACLE_HDFSおよびORACLE_HIVEを使用する方法を学習します。

- ORACLE_LOADERアクセス・ドライバを使用したパーティション化された外部表の作成
  このトピックでは、ORACLE_LOADERアクセス・ドライバを使用してパーティション化された外部表を作成する方法について説明します。
- ORACLE_LOADERアクセス・ドライバを使用したパーティション化されたハイブリッド表の作成
  このトピックでは、ORACLE_LOADERアクセス・ドライバを使用してパーティション化されたハイブリッド表を作成する方法について説明します。
- ORACLE_DATAPUMPアクセス・ドライバを使用したパーティション化された外部表の作成
  この項では、サブパーティション化された外部表を作成する方法の例を示します。
- ORACLE_HDFSアクセス・ドライバを使用したパーティション化された外部表の作成
  この項の例はORACLE_HDFSアクセス・ドライバを使用してパーティション化された外部表を作成する方法を示しています。
- ORACLE_HIVEアクセス・ドライバを使用したパーティション化された外部表の作成
  ORACLE_HIVE表に対してパーティション化された外部表を作成するには、パーティション化されたHive外部表が必要です。
- 外部表からのLOBのロード
  外部表は、単一のファイルから多数のレコードをロードし、各レコードが表の独自の行になるようにする場合に特に役立ちます。
- 外部表からのCSVファイルのロード
  このトピックでは、様々な条件で外部表からCSVファイルをロードする方法の例を示します。

親トピック: 外部表

18.1 ORACLE_LOADERアクセス・ドライバを使用したパーティション化された外部表の作成

このトピックでは、ORACLE_LOADERアクセス・ドライバを使用してパーティション化された外部表を作成する方法について説明します。

例18-1 ORACLE_LOADERを使用したパーティション化された外部表の作成

この例では、次の内容の4つのデータ・ファイルがあると仮定しています。

```
p1a.dat:
1, AAAAA Plumbing,01372,
28, Sparkly Laundry,78907,
13, Andi's Doughnuts,54570,
p1b.dat:
51, DIY Supplies,61614,
87, Fast Frames,22201,
89, Friendly Pharmacy,89901,
p2.dat:
121, Pleasant Pets,33893,
130, Bailey the Bookmonger,99915,
105, Le Bistrot du Chat Noir,94114,
p3.dat:
210, The Electric Eel Diner,07101,
```

```
222, Everyt'ing General Store,80118,
231, Big Rocket Market,01754,
```

データ・ファイルには、CUSTOMER_NUMBER、CUSTOMER_NAMEおよびPOSTAL_CODEの3つのフィールドがあります。外部表ではCUSTOMER_NUMBERに対してレンジ・パーティション化を使用して3つのパーティションを作成します。

- パーティション1、100未満のcustomer_number用
- パーティション2、200未満のcustomer_number用
- パーティション3、300未満のcustomer_number用

最初のパーティションのデータ・ファイルは2つで、他のパーティションでは1つのみであることに注意してください。次に、ファイルを作成する場合のSQLPlusからの出力を示します。

```
SQL> create table customer_list_xt
  2    (CUSTOMER_NUMBER number, CUSTOMER_NAME VARCHAR2(50), POSTAL_CODE CHAR(5))
  3  organization external
  4    (type oracle_loader default directory def_dir1)
  5  partition by range(CUSTOMER_NUMBER)
  6  (
  7    partition p1 values less than (100) location('p1a.dat', 'p1b.dat'),
  8    partition p2 values less than (200) location('p2.dat'),
  9    partition p3 values less than (300) location('p3.dat')
 10  );
Table created.
SQL>
```

次に、表全体に対するSELECT *からの出力を示します。

```
SQL> select customer_number, customer_name, postal_code
  2    from customer_list_xt
  3    order by customer_number;
CUSTOMER_NUMBER CUSTOMER_NAME                                          POSTA
--------------- -------------------------------------------------- -----
              1  AAAAA Plumbing                                      01372
             13  Andi's Doughnuts                                    54570
             28  Sparkly Laundry                                     78907
             51  DIY Supplies                                        61614
             87  Fast Frames                                         22201
             89  Friendly Pharmacy                                   89901
            105  Le Bistrot du Chat Noir                             94114
            121  Pleasant Pets                                       33893
            130  Bailey the Bookmonger                               99915
            210  The Electric Eel Diner                              07101
            222  Everyt'ing General Store                            80118
            231  Big Rocket Market                                   01754
12 rows selected.
SQL>
```

次の問合せでは、最初のパーティションのレコードのみが読み取られます。

```
SQL> select customer_number, customer_name, postal_code
  2    from customer_list_xt
  3    where customer_number < 20
  4    order by customer_number;
CUSTOMER_NUMBER CUSTOMER_NAME                                          POSTA
--------------- -------------------------------------------------- -----
              1  AAAAA Plumbing                                      01372
             13  Andi's Doughnuts                                    54570
2 rows selected.
SQL>
```

次の問合せでは、SELECT文の一部として読み取るパーティションを指定します。

```
SQL> select customer_number, customer_name, postal_code
  2    from customer_list_xt partition (p1)
  3    order by customer_number;
CUSTOMER_NUMBER CUSTOMER_NAME                                          POSTA
--------------- -------------------------------------------------- -----
              1  AAAAA Plumbing                                      01372
             13  Andi's Doughnuts                                    54570
             28  Sparkly Laundry                                     78907
             51  DIY Supplies                                        61614
             87  Fast Frames                                         22201
             89  Friendly Pharmacy                                   89901
6 rows selected.
SQL>
```

親トピック: 外部表の例

18.2 ORACLE_LOADERアクセス・ドライバを使用したパーティション化されたハイブリッド表の作成

このトピックでは、ORACLE_LOADERアクセス・ドライバを使用してパーティション化されたハイブリッド表を作成する方法について説明します。

ハイブリッド・パーティション表は、データベース・セグメント内および外部のファイルまたはソース内に一部のパーティションが存在できるようにすることで、Oracle Partitioningを拡張する機能です。これにより、表の大部分が外部パーティションに存在できるようになり、Big Data SQLのパーティション化の機能が大幅に向上します。

例18-2 例

次に、パーティション化されたハイブリッド表を作成する文の例を示します。

```
CREATE TABLE hybrid_pt (time_id date, customer number)
  TABLESPACE TS1
  EXTERNAL PARTITION ATTRIBUTES (TYPE ORACLE_LOADER
                                 DEFAULT DIRECTORY data_dir0
                                 ACCESS PARAMETERS(FIELDS TERMINATED BY ',')
                                 REJECT LIMIT UNLIMITED)
PARTITION by range (time_id)
(
 PARTITION century_18 VALUES LESS THAN ('01-01-1800')
    EXTERNAL,                                                <-- empty external
partition
  PARTITION century_19 VALUES LESS THAN ('01-01-1900')
    EXTERNAL DEFAULT DIRECTORY data_dir1 LOCATION (‘century19_data.txt'),
  PARTITION century_20 VALUES LESS THAN ('01-01-2000')
    EXTERNAL LOCATION (‘century20_data.txt'),
  PARTITION year_2000 VALUES LESS THAN ('01-01-2001') TABLESPACE TS2,
  PARTITION pmax VALUES LESS THAN (MAXVALUE)
);
```

この例では、表に内部パーティションと外部パーティションの両方が含まれています。表の内部パーティションのデフォルト表領域は、TS1です。表内の外部パーティションに表レベルで適用されるパラメータを指定するために、EXTERNAL PARTITION ATTRIBUTES句が追加されます。ハイブリッド・パーティション表の場合、この句は必須です。この場合、外部パーティションはORACLE_LOADERアクセス・ドライバを介してアクセスされ、アクセス・ドライバで必要なパラメータは句で指定されます。パーティション・レベルで、EXTERNAL句は、パーティションに適用されるすべての外部パラメータとともに、外部パーティションごとに指定されます。

この例では、century_18、century_19およびcentury_20は外部パーティションです。century_18は、場所が含まれていないため空のパーティションです。パーティションcentury_19のデフォルト・ディレクトリはdata_dir1であり、表レベルの

デフォルト・ディレクトリよりも優先されます。パーティションの場所はdata_dir1:century19_data.txtになります。パーティションにデフォルト・ディレクトリが指定されていない場合、表レベルのデフォルト・ディレクトリが場所に適用されるため、パーティションcentury_20の場所はdata_dir0:century20_data.txtになります。パーティションyear_2000およびpmaxは、内部パーティションです。パーティションyear_2000にはtablespace TS2が含まれます。パーティションにEXTERNAL句または外部パラメータが指定されていない場合、デフォルトで内部パーティションとみなされます。

親トピック: 外部表の例

18.3 ORACLE_DATAPUMPアクセス・ドライバを使用したパーティション化された外部表の作成

この項の例では、サブパーティション化された外部表を作成する方法を示しています。

また、仮想列を使用して表をパーティション化する方法も示しています。

例18-3 ORACLE_DATAPUMPアクセス・ドライバを使用したパーティション化された外部表の作成

この例で使用するダンプ・ファイルは、前の例でORACLE_LOADERアクセス・ドライバを使用して作成されたものと同じです。ただし、この例では、customer_numberを使用してデータをパーティション化するだけでなく、postal_codeを使用してデータをサブパーティション化します。すべてのパーティションに、postal_codeが50000未満のサブパーティションと、postal_codeが他のすべての値の別のサブパーティションがあります。それぞれ2つのサブパーティションを含む3つのパーティションでは、合計で6つのファイルが必要となります。ファイルを作成するために、SQL CREATE TABLE AS SELECTを使用してパーティションの正しい行を選択し、それらの行をORACLE_DATAPUMPドライバのファイルに書き込みます。

次の文では、パーティションp1 (customer_numberが100未満)の最初のサブパーティション(postal_codeが50000未満)のデータを含むファイルが作成されます。

```
SQL> create table customer_list_dp_p1_sp1_xt
  2  organization external
  3    (type oracle_datapump default directory def_dir1 location('p1_sp1.dmp'))
  4  as
  5    select customer_number, customer_name, postal_code
  6      from customer_list_xt partition (p1)
  7      where to_number(postal_code) < 50000;
Table created.
SQL>
```

この文では、パーティションp1 (customer_numberが100未満)の2番目のサブパーティション(postal_codeが他のすべての値)のデータを含むファイルが作成されます。

```
SQL> create table customer_list_dp_p1_sp2_xt
  2  organization external
  3    (type oracle_datapump default directory def_dir1 location('p1_sp2.dmp'))
  4  as
  5    select customer_number, customer_name, postal_code
  6      from customer_list_xt partition (p1)
  7      where to_number(postal_code) >= 50000;
Table created.
```

次のように、他のパーティションのファイルは同様に作成されます。

```
SQL> create table customer_list_dp_p2_sp1_xt
  2  organization external
  3    (type oracle_datapump default directory def_dir1 location('p2_sp1.dmp'))
  4  as
  5    select customer_number, customer_name, postal_code
  6      from customer_list_xt partition (p2)
```

```
  7      where to_number(postal_code) < 50000;
Table created.
SQL>
SQL> create table customer_list_dp_p2_sp2_xt
  2  organization external
  3    (type oracle_datapump default directory def_dir1 location('p2_sp2.dmp'))
  4  as
  5    select customer_number, customer_name, postal_code
  6      from customer_list_xt partition (p2)
  7      where to_number(postal_code) >= 50000;
Table created.
SQL>
SQL> create table customer_list_dp_p3_sp1_xt
  2  organization external
  3    (type oracle_datapump default directory def_dir1 location('p3_sp1.dmp'))
  4  as
  5    select customer_number, customer_name, postal_code
  6      from customer_list_xt partition (p3)
  7      where to_number(postal_code) < 50000;
Table created.
SQL>
SQL> create table customer_list_dp_p3_sp2_xt
  2  organization external
  3    (type oracle_datapump default directory def_dir1 location('p3_sp2.dmp'))
  4  as
  5    select customer_number, customer_name, postal_code
  6      from customer_list_xt partition (p3)
  7      where to_number(postal_code) >= 50000;
Table created.
SQL>
```

これらの各外部表から選択して、書き出すデータが含まれているかどうかを確認できます。SQL CREATE TABLE AS SELECT文を実行した後、これらの外部表を削除できます。

仮想列を使用して表をパーティション化するために、パーティション化されたORACLE_DATAPUMP表を作成します。この例でも、表はcustomer_number列でパーティション化され、postal_code列でサブパーティション化されます。postal_code列は数字を含む文字列フィールドですが、この例では、文字列ではなく数値に基づいてこの列をパーティション化します。このことを行うために、値がNUMBERデータ型に変換されるpostal_codeフィールドの仮想列postal_code_numを作成します。SUBPARTITION句では、行のサブパーティションを決定するために仮想列を使用します。

```
SQL> create table customer_list_dp_xt
  2  (customer_number    number,
  3   CUSTOMER_NAME      VARCHAR2(50),
  4   postal_code        CHAR(5),
  5   postal_code_NUM    as (to_number(postal_code)))
  6  organization external
  7    (type oracle_datapump default directory def_dir1)
  8  partition by range(customer_number)
  9  subpartition by range(postal_code_NUM)
 10  (
 11    partition p1 values less than (100)
 12     (subpartition p1_sp1 values less than (50000) location('p1_sp1.dmp'),
 13      subpartition p1_sp2 values less than (MAXVALUE) location('p1_sp2.dmp')),
 14    partition p2 values less than (200)
 15     (subpartition p2_sp1 values less than (50000) location('p2_sp1.dmp'),
 16      subpartition p2_sp2 values less than (MAXVALUE) location('p2_sp2.dmp')),
 17    partition p3 values less than (300)
 18     (subpartition p3_sp1 values less than (50000) location('p3_sp1.dmp'),
 19      subpartition p3_sp2 values less than (MAXVALUE) location('p3_sp2.dmp'))
 20  );
Table created.
SQL>
```

すべての行を選択した場合、返されるデータはORACLE_LOADERアクセス・ドライバを使用した前の例で返されたデータと同じになります。

```
SQL> select customer_number, customer_name, postal_code
  2    from customer_list_dp_xt
  3    order by customer_number;
customer_number CUSTOMER_NAME                                          POSTA
--------------- ---------------------------------------------------- -----
              1  AAAAA Plumbing                                       01372
             13  Andi's Doughnuts                                     54570
             28  Sparkly Laundry                                      78907
             51  DIY Supplies                                         61614
             87  Fast Frames                                          22201
             89  Friendly Pharmacy                                    89901
            105  Le Bistrot du Chat Noir                              94114
            121  Pleasant Pets                                        33893
            130  Bailey the Bookmonger                                99915
            210  The Electric Eel Diner                               07101
            222  Everyt'ing General Store                             80118
            231  Big Rocket Market                                    01754
12 rows selected.
SQL>
```

WHERE句では、読み取る行を1つのサブパーティションに制限できます。次の問合せでは、最初のパーティションの最初のサブパーティションのみが読み取られます。

```
SQL> select customer_number, customer_name, postal_code
  2    from customer_list_dp_xt
  3    where customer_number < 20 and postal_code_NUM < 39998
  4    order by customer_number;
customer_number CUSTOMER_NAME                                          POSTA
--------------- ---------------------------------------------------- -----
              1  AAAAA Plumbing                                       01372
1 row selected.
SQL>
```

次のように、問合せで特定のパーティションを指定することもできます。

```
SQL> select customer_number, customer_name, postal_code
  2    from customer_list_dp_xt subpartition (p2_sp2) order by customer_number;
customer_number CUSTOMER_NAME                                          POSTA
--------------- ---------------------------------------------------- -----
            105  Le Bistrot du Chat Noir                              94114
            130  Bailey the Bookmonger                                99915
2 rows selected.
SQL>
```

親トピック: 外部表の例

18.4 ORACLE_HDFSアクセス・ドライバを使用したパーティション化された外部表の作成

この項で説明する例では、ORACLE_HDFSアクセス・ドライバを使用してパーティション化された外部表を作成する方法を示します。

例18-4 ORACLE_HDFSアクセス・ドライバを使用したパーティション化された外部表の作成

この例では、HDFSディレクトリ・パス"hdfs_pet/"に格納された4つのデータ・ファイルがあります。

```
p1a.dat
1, AAAAA Plumbing,01372,
```

```
28, Sparkly Laundry,07101,
13, Andi'''s Doughnuts,01372,
p1b.dat
51, DIY Supplies,07101,
87, Fast Frames,01754,
89, Friendly Pharmacy,01372,
p2.dat
121, Pleasant Pets,01754,
130, Bailey the Bookmonger,01754,
105, Le Bistrot du Chat Noir,01754,
p3.dat
210, The Electric Eel Diner,07101,
222, Everyt'ing General Store,01372,
231, Big Rocket Market,01754,
```

この例の目的のために、データ・ファイルは次を使用してHDFSディレクトリに書き込まれます。

```
hadoop dfs -mkdir hdfs_pet
hadoop dfs -put p1a.dat hdfs_pet/p1a.dat
hadoop dfs -put p1b.dat hdfs_pet/p1b.dat
hadoop dfs -put p2.dat hdfs_pet/p2.dat
hadoop dfs -put p3.dat hdfs_pet/p3.dat
```

パーティション化された外部表を作成するためのCREATE TABLEコマンドは次のとおりです。

```
create table customer_list_hdfs
  (CUSTOMER_NUMBER number, CUSTOMER_NAME VARCHAR2(50), POSTAL_CODE CHAR(5))
organization external
  (type oracle_hdfs
   default directory def_dir1
   access parameters
   (com.oracle.bigdata.cluster   = hadoop_cl_1
    com.oracle.bigdata.rowformat = delimited fields terminated by ','))
partition by range(CUSTOMER_NUMBER)
(
  partition p1 values less than (100) location('hdfs_pet/p1a.dat',
'hdfs_pet/p1b.dat'),
  partition p2 values less than (200) location('hdfs_pet/p2.dat'),
  partition p3 values less than (300) location('hdfs_pet/p3.dat')
);
```

次の問合せは、外部表からのSELECT操作を示しています。

```
SQL> select * from customer_list_hdfs order by customer_number;
CUSTOMER_NUMBER CUSTOMER_NAME                                          POSTA
--------------- ---------------------------------------------------- -----
              1  AAAAA Plumbing                                       01372
             13  Andi's Doughnuts                                     01372
             28  Sparkly Laundry                                      07101
             51  DIY Supplies                                         07101
             87  Fast Frames                                          01754
             89  Friendly Pharmacy                                    01372
            105  Le Bistrot du Chat Noir                              01754
            121  Pleasant Pets                                        01754
            130  Bailey the Bookmonger                                01754
            210  The Electric Eel Diner                               07101
            222  Everyt'ing General Store                             01372
            231  Big Rocket Market                                    01754
12 rows selected.
SQL>
```

次のように、WHERE句の条件に一致しないパーティションを読取りの対象から除外するWHERE句を使用して問合せを実行することもできます。

```
SQL> select customer_number, customer_name, postal_code
```

```
  2    from customer_list_hdfs
  3    where customer_number < 20
  4    order by customer_number;
CUSTOMER_NUMBER CUSTOMER_NAME                                          POSTA
--------------- -------------------------------------------------- -----
              1  AAAAA Plumbing                                      01372
             13  Andi's Doughnuts                                    01372
2 rows selected.
SQL>
```

次に示すように、FROM句の一部として読み取るパーティションを指定することもできます。

```
SQL> select customer_number, customer_name, postal_code
  2    from customer_list_hdfs partition (p3)
  3    order by customer_number;
CUSTOMER_NUMBER CUSTOMER_NAME                                          POSTA
--------------- -------------------------------------------------- -----
            210  The Electric Eel Diner                              07101
            222  Everyt'ing General Store                            01372
            231  Big Rocket Market                                   01754
3 rows selected.
SQL>
```

親トピック: 外部表の例

18.5 ORACLE_HIVEアクセス・ドライバを使用したパーティション化された外部表の作成

ORACLE_HIVE表のパーティション化された外部表を作成するには、パーティション化されたHive外部表が必要です。

その後PL/SQLプロシージャDBMS_HADOOP.CREATE_EXTDDL_FOR_HIVE()を使用する必要があります。また、この例では、前の例で使用したHDFSファイルからパーティション化されたHive表を作成します。パーティション化されたHive表を作成する前に、データを格納するHDFSディクショナリを作成する必要があります。

例18-5 ORACLE_HIVEアクセス・ドライバを使用したパーティション化された外部表の作成

この例では、ORACLE_HDFSを使用してパーティション化された外部表を作成する方法を示した前の例のデータ・ファイルを使用して、Hive表を作成します。次のコマンドはすべてHive CLI内で実行されるため、Hive構文を使用します。

まず、次のように、Hiveを使用して、HDFSデータ・ファイルに基づくHive外部表を作成します。

```
create external table customer_list_no_part ( customer_number int,
                                              customer_name   string,
                                              postal_code string)
 row format delimited fields terminated by ','
 stored as textfile
 location '/user/doc/hdfs_pet'
```

次に、パーティションを動的に作成するようにHiveに指示するために、次のコマンドを実行します。

```
set hive.exec.dynamic.partition=true
set hive.exec.dynamic.partition.mode=nonstrict
```

パーティション化されたHive表の作成:

```
create table customer_list( CUSTOMER_NUMBER int,
                            CUSTOMER_NAME   string)
 partitioned by (postal_code string)
 row format delimited
 fields terminated by '¥t'
 location '/user/doc/doc_pet'
```

Hive表に表customer_list_no_partのデータを移入します。これにより、パーティションごとに、正しい行を含むファイルが作成されます。

```
insert overwrite table customer_list partition (postal_code) select * from
customer_list_no_part
```

次の問合せは、Hive customer_list表に移入された行を示しています。

```
select * from customer_list order by customer_number
1     AAAAA Plumbing    01372
13     Andi's Doughnuts    01372
28     Sparkly Laundry    07101
51     DIY Supplies    07101
87     Fast Frames    01754
89     Friendly Pharmacy    01372
105     Le Bistrot du Chat Noir    01754
121     Pleasant Pets    01754
130     Bailey the Bookmonger    01754
210     The Electric Eel Diner    07101
222     Everyt'ing General Store    01372
231     Big Rocket Market    01754
```

これで、SQL*Plusに戻り、Oracle Database内にパーティション化された外部表を作成できます。まず、PL/SQLファンクションDBMS_HADOOP.CREATE_EXTDDL_FOR_HIVE()を使用して外部表を作成します。引数は次のとおりです。

1. Hadoopクラスタの名前
2. 表を所有するHiveユーザーの名前
3. パーティション化されたHive表の名前
4. Hive表のパーティション列を外部表の列として含めるかどうかを示すブール値
5. 作成されるパーティション化されたORACLE_HIVE表の名前
6. CREATE DDLが実行されるかどうかを示すブール値
7. CREATE DDL文字列を含むCLOB

この例では、CREATE DDL文字列のCLOBを取得し、DBMS_OUTPUT.PUT_LINE()プロシージャを使用してこれを書き込みます。SERVEROUTPUT ONを設定することで、DBMS_OUTPUT.PUT_LINE()プロシージャからデータを表示するようSQL*Plusに指示します。LINESIZEを132に設定すると、DBMS_OUTPUT.PUT_LINE()プロシージャからデータを表示するときに不適切な場所に改行が挿入されません。

```
SQL> SET LINESIZE 132
SQL> SET SERVEROUTPUT ON
SQL>
SQL> DECLARE
  2    DDLtxt clob;
  3  BEGIN
  4    dbms_hadoop.create_extddl_for_hive
  5     ('hadoop_cl_1', 'default', 'customer_list',
  6      TRUE, 'CUSTOMER_LIST_HIVE', TRUE, DDLtxt);
  7    dbms_output.put_line('DDL Text is :  ' || DDLtxt);
  8  END;
  9  /
External table successfully created.
DDL Text is :  CREATE TABLE "DOC"."CUSTOMER_LIST_HIVE" (customer_number NUMBER,
customer_name VARCHAR2(4000), postal_code
VARCHAR2(4000))  ORGANIZATION EXTERNAL
 (TYPE ORACLE_HIVE
 DEFAULT DIRECTORY DEFAULT_DIR
 ACCESS PARAMETERS
```

```
(
com.oracle.bigdata.cluster=hadoop_cl_1
com.oracle.bigdata.tablename=default.customer_list)
) REJECT LIMIT UNLIMITED
PARTITION BY
LIST (postal_code)
(
PARTITION "P_293620257" VALUES ('01372'),
PARTITION "P_292175793" VALUES ('01754'),
PARTITION "P_717839126"
VALUES ('07101')
)
```

HiveではSTRING列の最大文字数は指定されないため、外部表の列定義はVARCHAR2(4000)となります。一部の列の長さをより短くする場合は、DBMS_HADOOP.CREATE_EXTDDL_FOR_HIVE()プロシージャをコールし、CREATE DDLを実行しないことを指定します。次に、CLOBで返されたCREATE文を編集して、VARCHAR2列の長さをより適切な値に設定できます。

また、パーティション名の数字は変動する場合があります。

これで、表が作成され、SELECT *文を実行すると、すべての行が返されます。前に実行したSET LINESIZEは、SQL*Plusでcustomer_nameおよびpostal_codeに対して132文字が使用されることを意味します。

```
SQL> select * from customer_list_hive order by customer_number;
CUSTOMER_NUMBER
---------------
CUSTOMER_NAME
--------------------------------------------------------------------------------
----------------------------------------POSTAL_CODE
--------------------------------------------------------------------------------
----------------------------------------              1
 AAAAA Plumbing
01372
             13
 Andi's Doughnuts
01372
             28
 Sparkly Laundry
07101
             51
 DIY Supplies
07101
             87
 Fast Frames
01754
             89
 Friendly Pharmacy
01372
            105
 Le Bistrot du Chat Noir
01754
            121
 Pleasant Pets
01754
            130
 Bailey the Bookmonger
01754
            210
 The Electric Eel Diner
07101
            222
 Everyt'ing General Store
01372
            231
```

```
 Big Rocket Market
01754
12 rows selected.
SQL>
```

SQLの実行では、パーティション情報を使用してWHERE句の条件に一致しないパーティションが除外されます。

```
SQL> select customer_number, customer_name, postal_code
  2    from customer_list_hive
  3    where postal_code = '01754'
  4    order by customer_number;
CUSTOMER_NUMBER
---------------
CUSTOMER_NAME
--------------------------------------------------------------------------------
----------------------------------------POSTAL_CODE
--------------------------------------------------------------------------------
----------------------------------------             87
 Fast Frames
01754
            105
 Le Bistrot du Chat Noir
01754
            121
 Pleasant Pets
01754
            130
 Bailey the Bookmonger
01754
            231
 Big Rocket Market
01754
5 rows selected.
SQL>
```

親トピック: 外部表の例

18.6 外部表からのLOBのロード

外部表は、単一のファイルから多数のレコードをロードし、各レコードが表の独自の行になるようにする場合に特に役立ちます。

次の例に、このようなロードを実行する方法を示します。

例18-6 外部表からのLOBのロード

次のように、外部表my_ext_tableを定義するとします。

```
CREATE TABLE my_ext_table ( id NUMBER, author VARCHAR2(30), created DATE, text CLOB )
ORGANIZATION EXTERNAL (
  TYPE ORACLE_LOADER
  DEFAULT DIRECTORY MY_DIRECTORY
  ACCESS PARAMETERS (
    RECORDS DELIMITED BY 0x'0A'
    FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
      ( id         CHAR(10),
        author     CHAR(30),
        created    DATE "YYYY-MM-DD",
        text       CHAR(131071)
      )
    )
    LOCATION (
      MY_DIRECTORY:'external.dmp'
    )
);
```

external.dmpファイルの内容は次のとおりです。

```
1,Roger,2015-08-08,The quick brown fox jumps over the lazy dog
2,John,2012-01-01,"The angry aligator, acting alone, ate the antelope"
```

ダンプ・ファイルの2番目の行には、全テキスト文字列を囲む引用符が必要です。引用符がない場合、フィールドはカンマで終了します。

ノート:

この例で使用されているダンプ・ファイルの問題ではありませんが、全テキスト文字列のいずれかの部分が引用符で囲まれている場合は、次のaloneという語のように、その部分をさらにもう1組の引用符で囲みます。

```
2,John,2012-01-01,"The angry aligator, acting ""alone"", ate the antelope"
```

全テキストにレコード・デリミタ文字(0x'0A'、または改行)が含まれている場合は、文書ごとに個別のファイルを指定できます。外部表ではFILLERフィールドはサポートされていないため、かわりにCOLUMN TRANSFORMS句を使用して、fnameフィールドに外部ファイルの名前が含まれることを指定する必要があります。

```
DROP TABLE my_ext_table2;
CREATE TABLE my_ext_table2 ( id NUMBER, author VARCHAR2(30), created DATE, text
CLOB )
ORGANIZATION EXTERNAL (
  TYPE ORACLE_LOADER
  DEFAULT DIRECTORY MY_DIRECTORY
  ACCESS PARAMETERS (
    RECORDS DELIMITED BY 0x'0A'
    FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
      ( id         CHAR(10),
        author     CHAR(30),
        created    DATE "YYYY-MM-DD",
        fname      char(100)
      )
    COLUMN TRANSFORMS (text FROM LOBFILE(fname) FROM (MY_DIRECTORY) )
    )
    LOCATION (
      'loader.txt'
    )
);
```

ノート:

FROM (MY_DIRECTORY)句はデフォルト・ディクショナリとしてすでに指定されているため、実際には必要ありません。ただし、loader.txt ファイルと個々の CLOB ファイルが異なる場所にある場合は必要となるため、ここでは例としての目的のために示しています。

外部表にデータが移入された後は、そのまま、その外部表に対して通常の表操作(DMLおよびほとんどのDDL)を実行するか、または外部表をステージング表として使用して標準の表にデータを取得できます。新しい標準の(外部でない)表を作成するには、次のSQL文を使用します。

```
CREATE TABLE normaltable AS SELECT * FROM externaltable;
```

同様に、次のSQL文を使用して新しい標準の表にデータを挿入できます。

```
INSERT INTO normaltable AS SELECT * FROM externaltable;
```

18.7 外部表からのCSVファイルのロード

このトピックでは、様々な条件で外部表からCSVファイルをロードする方法の例を示します。

一部の例は、前の例に基づいて作成されています。

例18-7 アクセス・パラメータを使用しないCSVファイルからのデータのロード

この例には、次の条件があります。

- 表の列の順序は、データ・ファイルのフィールドの順序に一致する必要があります。
- データ・ファイル内のレコードは、改行で終了する必要があります。
- データ・ファイル内のレコードのフィールドはカンマで区切る必要があります(フィールド値が引用符で囲まれている場合は、フィールドから引用符を削除しません)。
- フィールドの途中では改行文字を使用できません。

外部表のデータは次のとおりです。

```
events_all.csv
Winter Games,10-JAN-2010,10,
Hockey Tournament,18-MAR-2009,3,
Baseball Expo,28-APR-2009,2,
International Football Meeting,2-MAY-2009,14,
Track and Field Finale,12-MAY-2010,3,
Mid-summer Swim Meet,5-JUL-2010,4,
Rugby Kickoff,28-SEP-2009,6,
```

外部表の定義は次のとおりです。

```
SQL> CREATE TABLE EVENTS_XT_1
  2  (EVENT       varchar2(30),
  3   START_DATE  date,
  4   LENGTH      number)
  5  ORGANIZATION EXTERNAL
  6  (default directory def_dir1 location ('events_all.csv'));
Table created.
```

次に、外部表EVENTS_XT_1でのSELECT操作を示します。

```
SQL> select START_DATE, EVENT, LENGTH
  2    from EVENTS_XT_1
  3    order by START_DATE;
START_DAT EVENT                              LENGTH
--------- ------------------------------ ----------
18-MAR-09 Hockey Tournament                       3
28-APR-09 Baseball Expo                           2
02-MAY-09 International Football Meeting         14
28-SEP-09 Rugby Kickoff                           6
10-JAN-10 Winter Games                           10
12-MAY-10 Track and Field Finale                  3
05-JUL-10 Mid-summer Swim Meet                    4
7 rows selected.
SQL>
```

例18-8 データ・ファイルの日付フィールドの書式と一致しないセッションのデフォルトの日付マスク

この例は、セッションのデフォルトの日付マスクがデータ・ファイルの日付フィールドの書式と一致しないことを除いて前の例と同じで

す。次の例では、セッションの日付書式はDD-Mon-YYYYですが、データ・ファイルの日付書式はMM/DD/YYYYです。外部表定義に日付マスクがない場合、ORACLE_LOADERアクセス・ドライバはセッションの日付マスクを使用してデータ・ファイルの文字データを日付データ型に変換しようとします。または、外部表の日付列のロードに使用されるデータ・ファイルのすべてのフィールドに使用する日付マスクのアクセス・パラメータを指定します。

次に、外部表のデータ・ファイルの内容を示します。

```
events_all_date_fmt.csv
Winter Games,1/10/2010,10
Hockey Tournament,3/18/2009,3
Baseball Expo,4/28/2009,2
International Football Meeting,5/2/2009,14
Track and Field Finale,5/12/2009,3
Mid-summer Swim Meet,7/5/2010,4
Rugby Kickoff,9/28/2009,6
```

外部表の定義は次のとおりです。

```
SQL> CREATE TABLE EVENTS_XT_2
  2  (EVENT       varchar2(30),
  3   START_DATE  date,
  4   LENGTH      number)
  5  ORGANIZATION EXTERNAL
  6  (default directory def_dir1
  7   access parameters (fields date_format date mask "mm/dd/yyyy")
  8   location ('events_all_date_fmt.csv'));
Table created.
SQL>
```

次に、外部表EVENTS_XT_2でのSELECT操作を示します。

```
SQL> select START_DATE, EVENT, LENGTH
  2    from EVENTS_XT_2
  3    order by START_DATE;
START_DAT EVENT                              LENGTH
--------- ------------------------------ ----------
18-MAR-09 Hockey Tournament                       3
28-APR-09 Baseball Expo                           2
02-MAY-09 International Football Meeting         14
12-MAY-09 Track and Field Finale                  3
28-SEP-09 Rugby Kickoff                           6
10-JAN-10 Winter Games                           10
05-JUL-10 Mid-summer Swim Meet                    4
7 rows selected.
```

例18-9 データが2つのデータ・ファイルに分割されている場合

この例は、次のことを除き、この項の最初の例と同じです。

- データが2つのデータ・ファイルに分割されています。
- 各データ・ファイルにフィールドの名前を含む行が格納されています。
- データ・ファイルの一部のフィールドが引用符で囲まれています。

FIELD NAMES ALL FILESは、アクセス・ドライバに対し、各ファイルの最初の行にファイルのフィールドの名前を含む行が含まれることを示します。アクセス・ドライバはフィールドの名前を表の列の名前と照合します。したがって、ファイルのフィールドの順序が表の列の順序と同じである必要はありません。最初の行内のフィールド名が引用符で囲まれていない場合、アクセス・ドライバでは、表の列名と照合する前に、フィールド名を大文字に変更します。フィールド名が引用符で囲まれている場合、アクセス・ドライバでは、一致する名前を検索する前に、フィールド名を大文字に変更しません。

フィールドは引用符で囲まれているため、アクセス・パラメータを使用する場合は、CSV WITHOUT EMBEDDED RECORD TERMINATORS句が必要になります。この句では、次のことを指定しています。

- データ・ファイルのフィールドはカンマで区切られています。
- フィールドが二重引用符で囲まれている場合、アクセス・ドライバはフィールド値からそれらを削除します。
- フィールド値に改行は埋め込まれていません(このオプションを使用すると、アクセス・ドライバは外部表でのSELECT操作のパフォーマンスが低下する可能性がある一部のチェックをスキップできます)。

2つのデータ・ファイルは次のとおりです。

```
events_1.csv
"EVENT","START DATE","LENGTH",
"Winter Games", "10-JAN-2010", "10"
"Hockey Tournament", "18-MAR-2009", "3"
"Baseball Expo", "28-APR-2009", "2"
"International Football Meeting", "2-MAY-2009", "14"

events_2.csv
Event,Start date,Length,
Track and Field Finale, 12-MAY-2009, 3
Mid-summer Swim Meet, 5-JUL-2010, 4
Rugby Kickoff, 28-SEP-2009, 6
```

外部表の定義は次のとおりです。

```
SQL> CREATE TABLE EVENTS_XT_3
  2  ("START DATE"  date,
  3   EVENT         varchar2(30),
  4   LENGTH        number)
  5  ORGANIZATION EXTERNAL
  6  (default directory def_dir1
  7   access parameters (records field names all files
  8                      fields csv without embedded record terminators)
  9   location ('events_1.csv', 'events_2.csv'));
Table created.
```

次に、EVENTS_XT_3外部表でのSELECT操作の結果を示します。

```
SQL> select "START DATE", EVENT, LENGTH
  2    from EVENTS_XT_3
  3    order by "START DATE";
START DAT EVENT                              LENGTH
--------- ------------------------------ ----------
18-MAR-09 Hockey Tournament                       3
28-APR-09 Baseball Expo                           2
02-MAY-09 International Football Meeting         14
12-MAY-09 Track and Field Finale                  3
28-SEP-09 Rugby Kickoff                           6
10-JAN-10 Winter Games                           10
05-JUL-10 Mid-summer Swim Meet                    4
7 rows selected.
```

例18-10 データが2つのファイルに分割され、最初のファイルにのみフィールド名の行がある場合

この例は、最初のファイルにのみフィールド名の行があることを除き、例3と同じです。2番目のファイルの最初の行には、実際のデータが含まれています。RECORDS句は「field names first file」に変更されます。

2つのデータ・ファイルは次のとおりです。

```
events_1.csv (same as for example 3)
"EVENT","START DATE","LENGTH",
```

```
"Winter Games", "10-JAN-2010", "10"
"Hockey Tournament", "18-MAR-2009", "3"
"Baseball Expo", "28-APR-2009", "2"
"International Football Meeting", "2-MAY-2009", "14"

events_2_no_header_row.csv
Track and Field Finale, 12-MAY-2009, 3
Mid-summer Swim Meet, 5-JUL-2010, 4
Rugby Kickoff, 28-SEP-2009, 6
```

外部表の定義は次のとおりです。

```
SQL> CREATE TABLE EVENTS_XT_4
  2  ("START DATE"  date,
  3   EVENT         varchar2(30),
  4   LENGTH        number)
  5  ORGANIZATION EXTERNAL
  6  (default directory def_dir1
  7   access parameters (records field names first file
  8                      fields csv without embedded record terminators)
  9   location ('events_1.csv', 'events_2_no_header_row.csv'));
Table created.
```

次に、EVENTS_XT_4外部表でのSELECT操作を示します。

```
SQL> select "START DATE", EVENT, LENGTH
  2    from EVENTS_XT_4
  3    order by "START DATE";
START DAT EVENT                              LENGTH
--------- ------------------------------ ----------
18-MAR-09 Hockey Tournament                       3
28-APR-09 Baseball Expo                           2
02-MAY-09 International Football Meeting         14
12-MAY-09 Track and Field Finale                  3
28-SEP-09 Rugby Kickoff                           6
10-JAN-10 Winter Games                           10
05-JUL-10 Mid-summer Swim Meet                    4
7 rows selected.
```

例18-11 ファイルのフィールドの順序が表の列の順序と一致する場合

この例には、次の条件があります。

- ファイルのフィールドの順序は表の列の順序と一致します。
- フィールドは改行で区切られ、オプションで二重引用符で囲まれています。
- 値に改行が埋め込まれたフィールドがあり、それらのフィールドは二重引用符で囲まれています。

データ・ファイルの内容は次のとおりです。

```
event_contacts_1.csv
Winter Games, 10-JAN-2010, Ana Davis,
Hockey Tournament, 18-MAR-2009, "Daniel Dube
Michel Gagnon",
Baseball Expo, 28-APR-2009, "Robert Brown"
Internation Football Meeting, 2-MAY-2009,"Pete Perez
Randall Barnes
Melissa Gray",

event_contacts_2.csv
Track and Field Finale, 12-MAY-2009, John Taylor,
Mid-summer Swim Meet, 5-JUL-2010, "Louise Stewart
Cindy Sanders"
Rugby Kickoff, 28-SEP-2009, "Don Nguyen
```

```
Ray Lavoie"
```

表定義は次のとおりです。CSV WITH EMBEDDED RECORD TERMINATORS句では、二重引用符で囲まれ、改行も埋め込まれているフィールドの処理方法をアクセス・ドライバに指示します。

```
SQL> CREATE TABLE EVENTS_CONTACTS_1
  2  (EVENT       varchar2(30),
  3   START_DATE  date,
  4   CONTACT     varchar2(120))
  5  ORGANIZATION EXTERNAL
  6  (default directory def_dir1
  7   access parameters (fields CSV with embedded record terminators)
  8   location ('event_contacts_1.csv', 'event_contacts_2.csv'));
Table created.
```

次に、EVENT_CONTACTS_1外部表でのSELECT操作の結果を示します。

```
SQL> column contact format a30
SQL> select START_DATE, EVENT, CONTACT
  2    from EVENTS_CONTACTS_1
  3    order by START_DATE;
START_DAT EVENT                          CONTACT
--------- ------------------------------ ------------------------------
18-MAR-09 Hockey Tournament              Daniel Dube
                                         Michel Gagnon
28-APR-09 Baseball Expo                  Robert Brown
02-MAY-09 Internation Football Meeting   Pete Perez
                                         Randall Barnes
                                         Melissa Gray
12-MAY-09 Track and Field Finale         John Taylor
28-SEP-09 Rugby Kickoff                  Don Nguyen
                                         Ray Lavoie
10-JAN-10 Winter Games                   Ana Davis
05-JUL-10 Mid-summer Swim Meet           Louise Stewart
                                         Cindy Sanders
7 rows selected.
```

例18-12 データ・ファイルの一部のフィールドでアクセス・パラメータのデフォルト設定が使用されない場合

この例では、データ・ファイルのほとんどのフィールドでアクセス・パラメータのデフォルト設定が使用されるものの、少数のフィールドではデフォルト設定が使用されない場合に行う操作を示します。この例では、すべてのフィールドの設定を一覧表示するのではなく、デフォルトとは異なるフィールドの属性のみを設定する方法を示します。 相違点は以下のとおりです。

- 2つの日付フィールドがあり、その1つではセッションの書式が使用されますが、registration_deadlineでは異なる書式が使用されます。
- registration_deadlineでは、NULL値を示すためにNONEの値も使用されます。

データ・ファイルの内容は次のとおりです。

```
events_reg.csv
Winter Games,10-JAN-2010,10,12/1/2009,
Hockey Tournament,18-MAR-2009,3,3/11/2009,
Baseball Expo,28-APR-2009,2,NONE
International Football Meeting,2-MAY-2009,14,3/1/2009
Track and Field Finale,12-MAY-2010,3,5/10/010
Mid-summer Swim Meet,5-JUL-2010,4,6/20/2010
Rugby Kickoff,28-SEP-2009,6,NONE
```

表定義は次のとおりです。ALL FIELDS OVERRIDE句を使用すると、そのフィールドの情報を指定しながら、残りのフィールドでデフォルトを使用できます。残りのフィールドのデータ型はCHAR(255)で、フィールド・データはカンマで終了し、切捨てオプションはLDRTRIMになります。

```
SQL> CREATE TABLE EVENT_REGISTRATION_1
  2  (EVENT                 varchar2(30),
  3   START_DATE            date,
  4   LENGTH                number,
  5   REGISTRATION_DEADLINE date)
  6  ORGANIZATION EXTERNAL
  7  (default directory def_dir1
  8   access parameters
  9  (fields all fields override
 10     (REGISTRATION_DEADLINE CHAR (10) DATE_FORMAT DATE MASK "mm/dd/yyyy"
 11                           NULLIF REGISTRATION_DEADLINE = 'NONE'))
 12   location ('events_reg.csv'));
Table created.
```

次に、EVENT_REGISTRATION_1外部表でのSELECT操作の結果を示します。

```
SQL> select START_DATE, EVENT, LENGTH, REGISTRATION_DEADLINE
  2    from EVENT_REGISTRATION_1
  3    order by START_DATE;
START_DAT EVENT                              LENGTH REGISTRAT
--------- ------------------------------ ---------- ---------
18-MAR-09 Hockey Tournament                       3 11-MAR-09
28-APR-09 Baseball Expo                           2
02-MAY-09 International Football Meeting         14 01-MAR-09
28-SEP-09 Rugby Kickoff                           6
10-JAN-10 Winter Games                           10 01-DEC-09
12-MAY-10 Track and Field Finale                  3 10-MAY-10
05-JUL-10 Mid-summer Swim Meet                    4 20-JUN-10
7 rows selected.
```

親トピック: 外部表の例

19 ORACLE_BIGDATAアクセス・ドライバ

ORACLE_BIGDATAアクセス・ドライバを使用すると、Oracle Databaseの表に格納されたデータであるかのように、オブジェクト・ストアに格納されているデータにアクセスできます。

ORACLE_BIGDATAは、現在、Oracleオブジェクト・ストア、Amazon S3およびAzure Blob Storageへのアクセスをサポートしています。このドライバを使用すると、テストや小規模なデータ・セットに有用なローカル・データを問い合せることもできます。

- ORACLE_BIGDATAアクセス・ドライバの使用
  ORACLE_BIGDATAドライバを使用すると、外部オブジェクト・ストアにあるデータにアクセスできます。
- オブジェクト・ストアの資格証明の作成方法
  資格証明オブジェクトを使用すると、外部オブジェクト・ストアにアクセスできます。
- オブジェクト・ストアのアクセス・パラメータ
  これらのアクセス・パラメータを使用すると、オブジェクト・ストアに存在するファイルに関するプロパティを指定できます。

関連トピック

- 外部表の例

親トピック: 外部表

19.1 ORACLE_BIGDATAアクセス・ドライバの使用

ORACLE_BIGDATAドライバを使用すると、外部オブジェクト・ストアにあるデータにアクセスできます。

オブジェクト・ストア内のデータにアクセスするには、次の2つのステップが必要です。

- 資格証明オブジェクトを作成します(パブリック・バケットには不要です)。

  資格証明オブジェクトには、オブジェクト・ストア資格証明が暗号化された形式で格納されます。資格証明によって指定されたアイデンティティは、オブジェクト・ストアの基礎となるデータにアクセスできる必要があります。

- インライン外部表がサポートされています。これらの外部表は、単に問合せの一部として表されます。

  インライン外部表を使用して、外部表または問合せを作成します。アクセス・ドライバ・タイプはORACLE_BIGDATAである必要があります。CREATE TABLE文では、オブジェクト・ストアにアクセスする認証を提供する資格証明オブジェクトを参照する必要があります。作成する表には、オブジェクト・ストア内のファイルへのURIを提供するLOCATION句も必要です。

  パブリック・バケットの場合、CREDENTIALは必要ありません。

親トピック: ORACLE_BIGDATAアクセス・ドライバ

19.2 オブジェクト・ストアの資格証明の作成方法

資格証明オブジェクトを使用すると、外部オブジェクト・ストアにアクセスできます。

資格証明オブジェクトを作成するには、DBMS_CREDENTIAL.CREATE_CREDENTIALまたはDBMS_CLOUD.CREATE_CREDENTIALを使用します。このオブジェクトには、オブジェクト・ストアにアクセスするために必要なユーザー名とパスワード情報が含まれています。この資格証明パスワードは、クラウド・サービス内のユーザー名に対して作成された認証トークン(**認証トークン**)と一致する必要があります。

ノート:

DBMS_CLOUD パッケージをインストールしておく必要があります。

- DBMS_CREDENTIAL.CREATE_CREDENTIALを使用した資格証明オブジェクトの作成
  DBMS_CLOUDのサブプログラムDBMS_CREDENTIAL.CREATE_CREDENTIALを使用すると、外部オブジェクト・ストアへのアクセスを認証できます。
- DBMS_CLOUD.CREATE_CREDENTIALを使用した資格証明オブジェクトの作成
  DBMS_CLOUDのサブプログラムDBMS_CLOUD.CREATE_CREDENTIALを使用すると、外部オブジェクト・ストアへのアクセスを認証できます。
- オブジェクト・ストレージのLocation句の定義方法
  これらの例を使用して、ソースに応じてオブジェクト・ストアURIを指定する方法を確認します。
- ORACLE_BIGDATAアクセス・パラメータの理解
  ORACLE_BIGDATAを使用するには、データにアクセスして解析する方法を示す情報をアクセス・パラメータに指定します。

関連トピック

- My Oracle Supportノート2748362.1

親トピック: ORACLE_BIGDATAアクセス・ドライバ

19.2.1 DBMS_CREDENTIAL.CREATE_CREDENTIALを使用した資格証明オブジェクトの作成

DBMS_CLOUDのサブプログラムDBMS_CREDENTIAL.CREATE_CREDENTIALを使用すると、外部オブジェクト・ストアへのアクセスを認証できます。

次の例では、DBMS_CREDENTIAL.CREATE_CREDENTIALの使用方法を示します。

例19-1 クラウド・サービス資格証明

次の例では、my_credentialはOracle Cloud Infrastructureユーザー名、usernameはアカウント・ユーザー名、passwordはOracle Cloud Infrastructure認証トークンです。

```
execute dbms_credential.create_credential(
   credential_name => 'my_credential',
   username        => 'username',
   password        => 'password'
);
```

例19-2 ネイティブのOracle Cloud Infrastructure資格証明

次の例では、my_credentialはOracle Cloud Infrastructureユーザー名、user_ocidはOracle Cloud Identifier (OCID)、tenancy_ocidはOracle Cloudテナンシ識別子、private_keyはSSH秘密キー、fingerprintは公開キー・フィンガープリントです。

```
execute dbms_credential.create_credential(
   credential_name => 'my_credential',
   username => 'user_ocid',
   password => '',
   key =>
'{"tenancy_ocid":"tenancy_ocid","private_key":"private_key","fingerprint":"fingerprin
t"}');
```

資格証明を作成した後、パラメータcom.oracle.bigdata.credential.nameで資格証明オブジェクト名を指定します。このリリースの時点では、資格証明は表と同じスキーマ内にある必要があります。

関連トピック

- CREATE_CREDENTIALプロシージャ

親トピック: オブジェクト・ストアの資格証明の作成方法

19.2.2 DBMS_CLOUD.CREATE_CREDENTIALを使用した資格証明オブジェクトの作成

DBMS_CLOUDのサブプログラムDBMS_CLOUD.CREATE_CREDENTIALを使用すると、外部オブジェクト・ストアへのアクセスを認証できます。

次の例では、DBMS_CLOUD.CREATE_CREDENTIALの使用方法を示します。

> ノート:
>
> 資格証明パラメータは、OCI リソース・プリンシパル、Azure サービス・プリンシパル、Amazon リソース名(ARN)または Google サービス・アカウントにはできません。

例19-3 ネイティブのOracle Cloud Infrastructure資格証明

次の例では、my_credentialはOracle Cloud Infrastructureユーザー名、passwordはOracle Cloud Infrastructure認証トークン、user_ocidはOracle Cloud Identifier (OCID)、tenancy_ocidはOracle Cloudテナンシ識別子、private_keyはSSH秘密キー、fingerprintは公開キー・フィンガープリントです。

```
execute dbms_cloud.create_credential DBMS_CLOUD.CREATE_CREDENTIAL(
    credential_name => 'my_credential',
    username => 'user_ocid@example.com',
    password => 'password'
    key =>
'{"tenancy_ocid":"tenancy_ocid","private_key":"private_key","fingerprint":"fingerprin
t"}');
```

資格証明を作成した後、パラメータcom.oracle.bigdata.credential.nameで資格証明オブジェクト名を指定します。このリリースの時点では、資格証明は表と同じスキーマ内にある必要があります。

関連トピック

- CREATE_CREDENTIALプロシージャ

親トピック: オブジェクト・ストアの資格証明の作成方法

19.2.3 オブジェクト・ストレージのLocation句の定義方法

これらの例を使用して、ソースに応じてオブジェクト・ストアURIを指定する方法を確認します。

LOCATIONは、オブジェクト・ストア内のデータを指すURIです。現在サポートされているオブジェクト・ストアは、Oracleオブジェクト・ストア、Amazon S3およびAzure Blob Storageです。

この例では次の変数が使用されています。

- region – テナンシのリージョン

- `host` – サーバー・ホスト名
- `port` – ホスト上でリスニングしている、サービスに割り当てられたポート番号
- `container` – コンテナ・リソースの名前
- `namespace` – リージョン内のネームスペース
- `bucket` – リソースのグローバルに一意な名前
- `objectname` – バケット内のオブジェクトの一意の識別子
- `filename` – オブジェクト・ストアのファイル名

場所を定義するには、次の前提条件に注意してください。

- 資格証明オブジェクトは、プライベート・オブジェクト・ストアへのアクセスのために必要です。資格証明パラメータを省略する場合は、オブジェクトがパブリック・バケットにある必要があります。

  この資格証明に関連付けられているユーザーIDには、オブジェクト・ストレージからデータを読み取るためのアクセス権が必要です。

- ローカル・ストレージを使用してオブジェクト・ストレージ内のデータへのアクセスをテストする場合は、`ORACLE_LOADER`データ・ソースの場合と同じように、その場所のOracleディレクトリ・オブジェクトを指定する必要があります。

例19-4 ネイティブのOracle Cloud Infrastructureオブジェクト・ストレージ

```
location
('https://objectstorage.region.oraclecloud.com/n/namespace/b/bucket/o/objectname')
```

例19-5 Oracle Cloud Infrastructureオブジェクト・ストレージ

```
location
('https://swiftobjectstorage.region.oraclecloud.com/v1/namespace/bucket/filename'
```

例19-6 ホストスタイルのURI形式

```
location ('https://bucket.host/objectname')
```

例19-7 パススタイルのURI形式

```
location ('https://host/bucket/objectname')
```

たとえば、Amazonパススタイルの URIの形式は次のとおりです。

```
location ('https://s3-us-west-2.amazonaws.com/adwc/filename')
```

例19-8 Azure BLOB Storageの場所の形式

```
location ('https://host:port/container/blob')
```

たとえば、Azureパススタイルの URIの形式は次のとおりです。

```
location ('https://exampleacount.blob.core.windows.net/examplecontainer/exampleblob')
```

親トピック: オブジェクト・ストアの資格証明の作成方法

19.2.4 ORACLE_BIGDATAアクセス・パラメータの理解

`ORACLE_BIGDATA`を使用するには、データにアクセスして解析する方法を示す情報をアクセス・パラメータに指定します。

外部オブジェクト・ストアにアクセスするには、アクセス・パラメータcom.oracle.bigdata.fileformatで、`csv`、`textfile`、`avro`、`parquet`または`orc`のいずれかの値を使用してファイル形式タイプを定義します。

```
com.oracle.bigdata.fileformat=[csv|textfile|avro|parquet|orc]
```

ORACLE_BIGDATAを使用すると、テストや単純な問合せ目的でローカル・ファイルにアクセスすることもできます。この場合、LOCATIONフィールド値はORACLE_LOADERに使用する値と同じです。LOCATIONフィールドでは、Oracleディレクトリ・オブジェクトの後にファイルの名前を続けることができます。ローカル・ファイルの場合、資格証明オブジェクトは必須ではありません。ただし、ファイルにアクセスするには、ディレクトリ・オブジェクトに対する権限が必要です。

関連トピック

- Oracle Big Data SQLユーザーズ・ガイドのORACLE_BIGDATAアクセス・パラメータに関する項

親トピック: オブジェクト・ストアの資格証明の作成方法

19.3 オブジェクト・ストアのアクセス・パラメータ

これらのアクセス・パラメータを使用すると、オブジェクト・ストアに存在するファイルに関するプロパティを指定できます。

- プロパティを指定するための構文ルール
  プロパティは、SQL CREATE TABLEのACCESS PARAMETERS句および構成ファイル内に、キーワード/値のペアを使用して設定します。
- ORACLE_BIGDATAアクセス・パラメータ
  すべてのファイル形式に共通するアクセス・パラメータのセットがあります。特定のファイル形式に固有のパラメータもあります。
- GATHER_EXTERNAL_TABLE_STATS
  これは、外部表(ORACLE_HDFS、ORACLE_HIVE、ORACLE_BIGDATA)の統計を手動で収集するためのPL/SQLインタフェースです。

親トピック: ORACLE_BIGDATAアクセス・ドライバ

19.3.1 プロパティを指定するための構文ルール

プロパティは、SQL CREATE TABLEのACCESS PARAMETERS句および構成ファイル内に、キーワード/値のペアを使用して設定します。

構文は次のルールに従う必要があります。

- 各キーワード/値のペアの形式は、**キーワード**、コロンまたは等号と、**値**です。有効なキーワード/値のペアを次に示します。

  ```
  keyword=value
  keyword:value
  ```

  値は、セパレータの後の最初の非空白文字から行末までのすべてになります。セパレータと値の間の空白は無視されます。値の後の空白は保持されます。

- プロパティ定義は1行または複数行にできます。
- 行の終了文字は、LF、改行または改行とLFの組合せです。
- 1つのプロパティ定義が複数行にわたる場合は、最終行以外の行の終了文字の前にバックスラッシュ(エスケープ文字)を付けます。この例では、Keyword1プロパティの値はValue part 1 Value part 2 Value part 3です。

  ```
  Keyword1=  Value part 1 ¥
  ```

```
            Value part 2 ¥
            Value part 3
```

- 先頭の空白の各物理行を削除して行を連結することで、**論理行**を作成できます。パーサーによって、プロパティ名および値が論理行から抽出されます。
- 文字の前に代入を示すバックスラッシュ(エスケープ文字)を付けることで、プロパティ名またはプロパティ値に特殊文字を埋め込むことができます。次の表で、特殊文字について説明します。

表19-1 プロパティの特殊文字

エスケープ・シーケンス	文字
¥b	バックスペース(¥u0008)
¥t	水平タブ(¥u0009)
¥n	LF (¥u000a)
¥f	改ページ(¥u000c)
¥r	改行(¥u000d)
¥"	二重引用符(¥u0022)
¥'	一重引用符(¥u0027)
¥¥	バックスラッシュ(¥u005c) 複数のバックスラッシュが行末にある場合、パーサーは奇数のバックスラッシュに対してのみ値を次の行に継続します。
¥uxxxx	2 バイト、ビッグ・エンディアンの Unicode コード・ポイント。 1 文字に 2 コード・ポイント(4 バイト)が必要な場合、パーサーは 2 番目のコード・ポイントに¥uを要求します。

親トピック: オブジェクト・ストアのアクセス・パラメータ

19.3.2 ORACLE_BIGDATAアクセス・パラメータ

すべてのファイル形式に共通するアクセス・パラメータのセットがあります。特定のファイル形式に固有のパラメータもあります。

共通のアクセス・パラメータ

次の表に、ORACLE_BIGDATAを介してアクセスされるすべてのファイル形式に共通するパラメータを示します。1列目には、すべてのデータ・ファイル・タイプに共通する各アクセス・パラメータを示します。2列目は各パラメータの説明です。

表19-2 共通のアクセス・パラメータ

共通のアクセス・パラメータ	説明
com.oracle.bigdata.credential.name	オブジェクト・ストア内のデータ・ファイルにアクセスするときに使用する資格証明オブジェクトを指定します。 このアクセス・パラメータは、オブジェクト・ストアへのアクセスに必要です。ディレクトリ・オブジェクトを介したファイルへのアクセスの場合や、パブリック・バケットに格納されているデータの場合は必要ありません。 資格証明に指定する名前は、表の所有者と同じスキーマにある資格証明オブジェクトの名前である必要があります。この表に対する SELECT または READ のアクセス権をユーザーに付与すると、その資格証明が表へのアクセスに使用されます。 資格証明オブジェクトを作成するには、DBMS_CREDENTIAL PL/SQL パッケージの DBMS_CREDENTIAL.CREATE_CREDENTIAL を使用します。exec dbms_credential.create_credential(credential_name => 'MY_CRED',username =>'<username>', password => '<password>'); CREATE TABLE 文で、資格証明パラメータの値を資格証明オブジェクトの名前に設定します。 com.oracle.bigdata.credential.name=MY_CRED
com.oracle.bigdata.fileformat	ファイルの形式を指定します。このパラメータの値によって、ファイルを処理するリーダーが識別されます。各リーダーは追加のアクセス・パラメータをサポートでき、それらは他のリーダーでサポートされるものもサポートされないものもあります。 有効な値: parquet、orc、textfile、avro、csv デフォルト: PARQUET
com.oracle.bigdata.log.opt	ログ・メッセージをログ・ファイルに書き込むかどうかを指定します。none を指定すると、ログ・ファイルは作成されません。値が normal の場合、ファイル・リーダーでメッセージを書き込むようになっていると、ログ・ファイルが作成されます。ログ・ファイルに書き込む内容はファイル・リーダー次第です。

共通のアクセス・パラメータ	説明
	有効な値: normal、none デフォルト: none
com.oracle.bigdata.log.qc	パラレル問合せコーディネータによって作成されるログ・ファイルの名前を指定します。このパラメータは、com.oracle.bigdata.log.opt が normal に設定されている場合にのみ使用されます。有効な値は、ORACLE_HIVE および ORACLE_HDFS で com.oracle.bigdata.log.qc に指定されている値と同じです。
com.oracle.bigdata.log.exec	問合せの実行時に作成されるログ・ファイルの名前を指定します。この値は、com.oracle.bigdata.log.exec が normal に設定されている場合のみ使用されます(必須です)。有効な値は、ORACLE_HIVE および ORACLE_HDFS で指定されている値と同じです。 有効な値: normal、none デフォルト: none

Avro固有のアクセス・パラメータ

共通のアクセス・パラメータの他に、Avroファイル形式でのみ有効なパラメータもあります。この表では、1列目にAvroファイル形式に固有のアクセス・パラメータを示し、2列目でそのパラメータについて説明します。現時点では、Avro固有のパラメータは1つのみです。

表19-3 Avro固有のアクセス・パラメータ

Avro固有のパラメータ	説明
com.oracle.bigdata.avro.decimaltpe	バイト配列に格納されている 10 進数の表現を指定します。 有効な値: int、integer、str、string デフォルト: このパラメータを使用しない場合、Avro の 10 進数列は、バイト配列に格納されているのは Avro 仕様で定義されているとおりの値の数値表現(デフォルトは int)であるという想定で読み取られます。

Parquet固有のアクセス・パラメータ

一部のアクセス・パラメータは、Parquetファイル形式でのみ有効です。この表では、1列目にParquetファイル形式に固有のア

クセス・パラメータを示し、2列目でそのパラメータについて説明します。

表19-4 Parquet固有のアクセス・パラメータ

Parquet固有のアクセス・パラメータ	説明
com.oracle.bigdata.prq.binary_as_string	これは、binary を文字列として格納するかどうかを指定するブール型プロパティです。 有効な値: true、t、yes、y、l、false、f、no、n、0 デフォルト: true
com.oracle.bigdata.prq.int96_as_timestamp	これは、int96 がタイムスタンプを表すかどうかを指定するブール型プロパティです。 有効な値: true、t、yes、y、l、false、f、no、n、0 デフォルト: true

テキスト・ファイルおよびCSVに固有のアクセス・パラメータ

テキスト・ファイルとカンマ区切り値(csv)ファイル形式は、Hiveテキスト・ファイル形式に似ています。これは、デリミタ付きファイルからテキスト・データおよびcsvデータを読み取ります。ORACLE_BIGDATAは、行の終了文字(¥n、¥rまたは¥r¥n)を自動的に検出します。デフォルトでは、ファイル内のフィールドはカンマで区切られており、ファイル内のフィールドの順序は外部表内の列の順序と一致すると想定されています。

例19-9 CSVデータ・ファイル

これは単純なcsvの例です。このデータ・ファイルにはカンマ区切りの値が含まれ、オプションで引用符で囲まれています。

```
-----Source csv data in t.dat
t.dat:
1,"abc",
2,xyx,

----------Create an external table over the csv source data in t.dat
CREATE TABLE t
(
 c0 number,
 c1 varchar2(20)
)
ORGANIZATION external
(
 TYPE oracle_bigdata
 DEFAULT DIRECTORY DMPDIR
 ACCESS PARAMETERS
 (
  com.oracle.bigdata.fileformat=csv
 )
 location
 (
  't.dat'
 )
)REJECT LIMIT 1
```

```
;

-------Select data from external table
select c0, c1 from t;
 C0    C1
----  -----
 1     abc
 2     xyz
```

例19-10 CSVデータ・ファイル

この例では、フィールド・セパレータが'|'であるcsvデータ・ソースを使用して外部表を作成する方法を示します。データ・ファイルはgzipで圧縮され、空白にNULLが使用され、日付書式を含みます。

```
-----The source csv data in t.dat
t.dat:
 1|  |
 2|Apr-99-30|

-------Create an external table over the csv data source in t.dat
CREATE TABLE t(
 c0 number,
 c1 date
)
ORGANIZATION external
(
 TYPE oracle_bigdata
 DEFAULT DIRECTORY DMPDIR
 ACCESS PARAMETERS
 (
  com.oracle.bigdata.fileformat=textfile
  com.oracle.bigdata.compressiontype=gzip
  com.oracle.bigdata.csv.rowformat.separatorcharacter='|'
  com.oracle.bigdata.blankasnull=true
  com.oracle.bigdata.dateformat="MON-RR-DD HH:MI:SS"
 )
 location
 (
  't.dat.gz'
  )
 )REJECT LIMIT 1
;

--Select csv data from external table
QL> select c0, c1 from t;

 C0       C1
------ ---------
  1
  2    30-APR-99
```

例19-11 JSONデータ・ファイル

これは、各行が1つのJSONドキュメントであるJSONファイルです。各行に外部表がアクセスします。問合せでは、Oracle SQL JSON関数を使用してデータが解析されます。

```
{"id":"72","name":"George","lat":40.76727216,"lon":-
73.99392888,"segments":["wealthy","middle-aged"],"age":45}
{"id":"79","name":"Lucy","lat":40.71911552,"lon":-
74.00666661,"segments":["married","yes"],"age":33}

— Create the external table over Json source
CREATE TABLE people (
 doc varchar2(4000)
```

```
)
ORGANIZATION EXTERNAL (
  TYPE ORACLE_BIGDATA
  DEFAULT DIRECTORY DEFAULT_DIR
  ACCESS PARAMETERS(
    com.oracle.bigdata.credential.name=MY_CRED
    com.oracle.bigdata.fileformat=textfile
    com.oracle.bigdata.csv.rowformat.fields.terminator='¥n’    <— notice the
      delimiter is a new line (that is, not a comma).  So, it will read to the end of
the line and get
      the whole document.
    )
  LOCATION ('https://swftobjectstorage.us-phoenix-
1.oraclecloud.com/v1/mybucket/people.json')
)
REJECT LIMIT UNLIMITED;

----Select data from external table
select s.doc.id,
  s.doc.name,
  s.doc.segments[0] ---notice we’re getting the first item in the array of possible
individuals
from the table people;
id     Name     segments[0]
----   -----    -----------------------------------
  72   George   wealthy
  79   Lucy     married
```

テキスト・ファイルおよびCSVに固有のアクセス・パラメータ

表19-5 テキスト・ファイルおよびCSVに固有のアクセス・パラメータ

テキスト・ファイルに固有のアクセス・パラメータ	説明
com.oracle.bigdata.buffersize	ファイルの読取りに使用する入出力(I のサイズ(KB 単位)です。また、バッフ ります。フォーマット・リーダーは、この値 す。 デフォルト: 1024
com.oracle.bigdata.blankasnull	true に設定すると、空白で構成され 有効な値: true、false デフォルト: false 例: com.oracle.bigdata.bla
com.oracle.bigdata.characterset	ソース・ファイルの文字セットを指定しま 有効な値: UTF-8 デフォルト: UTF-8

テキスト・ファイルに固有のアクセス・パラメータ	説明
	例: com.oracle.bigdata.cha
com.oracle.bigdata.compressiontype	このパラメータを指定した場合、コード 凍しようとします。 有効な値: gzip、bzip2、zlib、 デフォルト: 圧縮なし detect を指定すると、フォーマット・ 使用してファイルが圧縮されたかを判別
com.oracle.bigdata.conversionerrors	行でデータ型の変換エラーが発生 納するか、その行を拒否します。 有効な値: reject_record、sto デフォルト: store_null 例: com.oracle.bigdata.con
com.oracle.bigdata.csv.rowformat.nulldefinedas	フィールドの値を示すために使用する文 指定しない場合、値はありません。
com.oracle.bigdata.csv.rowformat.fields.terminator	フィールド値を区切るために使用する文 必要があります。例: '|' デフォルト: ','
com.oracle.bigdata.csv.rowformat.fields.escapedby	埋込みフィールドの終了記号またはフ 使用する文字を指定します。文字値
com.oracle.bigdata.dateformat	ソース・ファイル内の日付書式を指定し チェックされます。 J、MM-DD-YYYYBC、MM-DD-YYY HHMISS、YYYY.DDD、YYYY-MM- デフォルト: yyyy-mm-dd hh24:m 例: com.oracle.bigdata.dat

テキスト・ファイルに固有のアクセス・パラメータ	説明
com.oracle.bigdata.fields	データ・ファイルのフィールドの順序を指 1 つの例外を除き、ORACLE_HDFS データ・ファイルはテキストであるため、テ 無視し、すべてのフィールドがテキストで め、このパラメータはフィールド名のリス
com.oracle.bigdata.ignoreblanklines	true に設定すると、空白行は無視さ 有効な値: true、false デフォルト: false 例: com.oracle.bigdata.ign
com.oracle.bigdata.ignoremissingcolumns	見つからない列が NULL として格納さ 有効な値: true デフォルト: true 例: com.oracle.bigdata.ign
com.oracle.bigdata.quote	フィールドの引用符文字を指定します されます。 有効な値: 文字 デフォルト: NULL (引用符なしを意味 例: com.oracle.bigdata.csv
com.oracle.bigdata.rejectlimit	指定した数の行が拒否された後で、操 コードが拒否される場合にのみ適用さ 有効な値: 数値 デフォルト: 0 例: com.oracle.bigdata.re
com.oracle.bigdata.removequotes	ソース・ファイル内のフィールドを囲む引

テキスト・ファイルに固有のアクセス・パラメータ	説明
	有効な値: true、false デフォルト: false 例: com.oracle.bigdata.rem
com.oracle.bigdata.csv.skip.header	ファイルの先頭からスキップする行数を 有効な値: 数値 デフォルト: 0 (指定しない場合) 例: com.oracle.bigdata.csv
com.oracle.bigdata.timestampformat	ソース・ファイル内のタイムスタンプ書式 書式がチェックされます。 YYYY-MM-DD HH:MI:SS.FF、Y MM/DD/YYYY HH:MI:SS.FF3 有効な値: auto デフォルト: yyyy-mm-dd hh24:m 例: com.oracle.bigdata.tim
com.oracle.bigdata.timestampltzformat	ソース・ファイル内のローカル・タイムゾー プション AUTO では、次の書式がチェッ DD Mon YYYY HH:MI:SS.FF YYYY-MM-DD HH:MI:SS+/-TZ DD.MM.YYYY HH:MI:SS TZR 有効な値: auto デフォルト: yyyy-mm-dd hh24:m 例: com.oracle.bigdata.tim
com.oracle.bigdata.timestamptzformat	ソース・ファイル内のタイムゾーン付きタ AUTO では、次の書式がチェックされま DD Mon YYYY HH:MI:SS.FF YYYY-MM-DD HH:MI:SS+/-TZ

テキスト・ファイルに固有のアクセス・パラメータ	説明
	DD.MM.YYYY HH:MI:SS TZR 有効な値: auto デフォルト: yyy-mm-dd hh24:mi 例: com.oracle.bigdata.tir
com.oracle.bigdata.trimspaces	フィールドの先頭と末尾の空白を切り 有効な値: rtrim、ltrim、notri デフォルト: notrim 例: com.oracle.bigdata.tri
com.oracle.bigdata.truncatecol	ファイルのデータがフィールドに対して長 フィールドを NULL に設定したりするか 有効な値: true、false デフォルト: false 例: com.oracle.bigdata.tru

親トピック: オブジェクト・ストアのアクセス・パラメータ

19.3.3 GATHER_EXTERNAL_TABLE_STATS

これは、外部表(ORACLE_HDFS、ORACLE_HIVE、ORACLE_BIGDATA)の統計を手動で収集するためのPL/SQLインタフェースです。

動作とパラメータはdbms_stats.gather_table_statsと同じですが、表の所有者はプロシージャを実行しているセッション・ユーザーである必要があり、このプロシージャを使用して収集された統計は再起動後も保持される点が異なります。このプロシージャは、Hiveから自動的に同期化される外部表では使用できません。

GATHER_TABLE_STATSプロシージャを参照してください。

構文

```
PROCEDURE gather_external_table_stats( tabname varchar2, partname varchar2
default null, estimate_percent number default
dbms_stats.DEFAULT_ESTIMATE_PERCENT, block_sample boolean default FALSE,
method_opt varchar2 default dbms_stats.DEFAULT_METHOD_OPT, degree number default
dbms_stats.DEFAULT_DEGREE_VALUE, granularity varchar2 default
dbms_stats.DEFAULT_GRANULARITY, cascade boolean default
dbms_stats.DEFAULT_CASCADE, stattab varchar2 default null, statid varchar2
default null, statown varchar2 default null, no_invalidate boolean default
dbms_stats.to_no_invalidate_type(dbms_stats.get_param('NO_INVALIDATE')),
```

```
stattype varchar2 default 'DATA', force boolean default FALSE, options varchar2
default dbms_stats.DEFAULT_OPTIONS )
```

```
PROCEDURE gather_external_table_stats(
    tabname varchar2,
    partname varchar2 default null,
    estimate_percent number default dbms_stats.DEFAULT_ESTIMATE_PERCENT,
    block_sample boolean default FALSE,
    method_opt varchar2 default dbms_stats.DEFAULT_METHOD_OPT,
    degree number default dbms_stats.DEFAULT_DEGREE_VALUE,
    granularity varchar2 default dbms_stats.DEFAULT_GRANULARITY,
    cascade boolean default dbms_stats.DEFAULT_CASCADE,
    stattab varchar2 default null,
    statid varchar2 default null,
    statown varchar2 default null,
    no_invalidate boolean default
dbms_stats.to_no_invalidate_type(dbms_stats.get_param('NO_INVALIDATE')),
    stattype varchar2 default 'DATA',
    force boolean default FALSE,
    options varchar2 default dbms_stats.DEFAULT_OPTIONS
  )
```

親トピック: オブジェクト・ストアのアクセス・パラメータ

第IV部 その他のユーティリティ

その他のOracleデータ管理ユーティリティには、ADRコマンド・インタプリタ、DBVERIFY、Oracle LogMiner、DBMS_METADATA APIおよびレガシー・データ移動ユーティリティが含まれます。

- クラウド移行前アドバイザ・ツール
  Oracle Cloudデータベースに移行する前にソース・データベースの互換性を評価するには、クラウド移行前アドバイザ・ツール(CPAT)を使用します。
- ADRCI: ADRコマンド・インタプリタ
  自動診断リポジトリ・コマンド・インタプリタ(ADRCI)ユーティリティは、Oracle Database診断データの管理に使用するコマンドライン・ツールです。
- DBVERIFY: オフライン・データベース検査ユーティリティ
  DBVERIFYは外部コマンドライン・ユーティリティで、物理データ構造に対する整合性チェックを実行します。
- DBNEWIDユーティリティ
  DBNEWIDは、オペレーショナル・データベースのDBIDおよびDBNAMEを変更できるデータベース・ユーティリティです。
- REDOログ・ファイル分析のためのLogMinerの使用
  Oracle LogMinerは、Oracle Databaseの一部で、SQLインタフェースを介してREDOログ・ファイル(オンラインおよびアーカイブ)を問い合せることができます。
- メタデータAPIの使用
  DBMS_METADATA APIの使用。
- オリジナルのエクスポート
  オリジナルのエクスポート・ユーティリティ(exp)が、Oracle Databaseからオペレーティング・システム・ファイルにバイナリ形式でデータを書き込むために使用されました。
- オリジナルのインポート
  オリジナルのエクスポート・ユーティリティ(imp)は、オリジナルのエクスポート・ユーティリティ(exp)を使用して作成されたダンプ・ファイルをインポートします。

20 クラウド移行前アドバイザ・ツール

Oracle Cloudデータベースに移行する前にソース・データベースの互換性を評価するには、クラウド移行前アドバイザ・ツール(CPAT)を使用します。

- クラウド移行前アドバイザ・ツールとは
  クラウド移行前アドバイザ・ツール(CPAT)は、Oracle Database内のデータベース・メタデータを分析する移行アシスタントであり、Oracle Cloud内のOracle Autonomous Databaseにデータを移動する際に役立つ情報を提供します。
- クラウド移行前アドバイザ・ツールを使用するための前提条件
  クラウド移行前アドバイザ・ツール(CPAT)を実行するために必要なJava環境、ユーザー権限およびセキュリティが設定されていることを確認します。
- クラウド移行前アドバイザ・ツールのダウンロードと構成
  クラウド移行前アドバイザ・ツール(CPAT)の最新の更新をダウンロードして、ディレクトリに抽出し、環境変数を設定します。
- クラウド移行前アドバイザ・ツール(CPAT)の開始
  Oracle SQLclまたはCPATをダウンロードした後、ソース・データベースに必要なJavaホームがあることを確認し、環境変数を設定して、実行するチェックの種類を決定します。
- クラウド移行前アドバイザ・ツールの接続文字列
  クラウド移行アドバイザ・ツール(CPAT)は、標準のOracle JDBC形式の接続文字列を受け入れます。
- クラウド移行前アドバイザ・ツールに必要なコマンドライン文字列
  ユースケースに応じて、クラウド移行前アドバイザ・ツール(CPAT)の実行には一部の文字列が必要です。
- FULLモードおよびSCHEMAモード
  クラウド移行前アドバイザ・ツール(CPAT)は、インスタンス全体に対して、またはスキーマに対して実行できます。
- クラウド移行前アドバイザ・ツール(CPAT)レポート・データの解釈
  CPATによって生成されるレポートには、サマリー情報と、正常に実行された各チェックの詳細が含まれます。
- コマンドライン構文およびプロパティ
  クラウド移行前アドバイザ・ツール(CPAT)のプロパティを使用して、CPATコマンドライン構文で実行するチェックおよびその他の操作を指定します。
- 移行前アドバイザ・ツールのログ・ファイル構造
  移行前アドバイザ・ツールでは、ジョブ・ステータスや構成ファイルを含む、ログ・ファイル構造が生成されます。
- 移行前アドバイザ・ツールによって実行されるチェックのリスト
  移行前アドバイザ・ツールのレポートに存在するチェックについて情報をご確認ください。
- 移行前アドバイザ・ツールを使用するためのベスト・プラクティス
  これらのクラウド移行前アドバイザ・ツール(CPAT)のヒントは、CPATをより効果的に使用するのに役立ちます。

親トピック: その他のユーティリティ

20.1 クラウド移行前アドバイザ・ツールを使用するための前提条件

クラウド移行前アドバイザ・ツール(CPAT)を実行するために必要なJava環境、ユーザー権限およびセキュリティが設定されていることを確認します。

Java Runtime Environment (JRE)の要件

CPATを実行するサーバーまたはクライアントにJava 7以降がインストールされている必要があります。Java 8 Java

Runtime Environment (JRE)を使用することをお薦めします。

CPATは、環境変数JAVA_HOMEおよびORACLE_HOMEを使用してJREを検索します。ソースOracle DatabaseがOracle 12cリリース1 (12.1.0.2)より後の場合は、CPATを実行できるJava JREのバージョンをOracleホームで使用できます。以前のリリースのOracle Databaseから移行する場合、または新しいJavaリリースのOracleホームを使用するように指定する場合は、環境変数がCPATの適切なJavaホームに設定されていることを確認してください。

Oracle Call InterfaceベースのシックJDBC接続文字列を使用する場合、CPATでは現在、ORACLE_SID、ORACLE_HOMEおよびLD_LIBRARY_PATHの環境変数が設定されていることが想定されます。

> ノート:
>
> Oracle Database ホーム内で使用可能な oraenv スクリプトを使用して、ORACLE_SID、ORACLE_HOME および LD_LIBRARY_PATH を設定することをお薦めします。

接続文字列および関連する環境変数の詳細は、「接続文字列」というタイトルの「高度な使用上のノート」セクションを参照してください。

ソース・データベースに対するユーザー権限

チェックのためにソース・データベースに接続するユーザーを指定し、そのユーザーにCPAT --usernameプロパティを指定する場合は、指定するユーザー名にSELECT ANY DICTIONARY権限が付与され、SYSTEM.DUM$COLUMNSおよびSYSTEM.DUM$DATABASEに対するSELECTが付与されている必要があります。

DUM$表へのアクセスが必要になるのは、ソースとターゲットの文字セットでOracle Database Migration Assistant for Unicode (DMU)が必要であることが示されている場合のみです。

> ノート:
>
> CPAT をインストールして実行しても、Oracle Database は変更されません。CPAT ではユーザーまたはパッケージは作成されず、CPAT ではロールまたは権限は付与されません。データベースへの CPAT アクセスは READ ONLY です。データベース・メタデータのみをチェックし、アプリケーションまたはビジネス・データはチェックしません。

セキュリティ構成

- --outdirプロパティを使用して、CPATログの出力場所を設定し、サーバーまたはクライアント上のセキュアな場所を使用します。
- LinuxおよびUnixシステムでユーザー・ファイル作成モード・マスク(umask)を設定して、CPATスクリプトのr|w|x権限のデフォルト値を認可ユーザーに制限します。

親トピック: クラウド移行前アドバイザ・ツール

20.2 クラウド移行前アドバイザ・ツールのダウンロードと構成

クラウド移行前アドバイザ・ツール(CPAT)の最新の更新をダウンロードして、ディレクトリに抽出し、環境変数を設定します。

CPATを実行するには、Oracle SQLclおよびSQLclコマンド- MIGRATEADVSORを使用することをお薦めします。次のURLからSQLclをダウンロードできます:

https://www.oracle.com/database/sqldeveloper/。

1. CPATパッチのダウンロードおよび抽出

バージョン11.2.0.4以上用のパッチ32613591クラウド移行前アドバイザ・ツール(CPAT)

CPATのMy Oracle Supportノートは、クラウド移行前アドバイザ・ツール(CPAT)によるクラウド移行の適性の分析(ドキュメントID 2758371.1)です。

2. Javaがインストールされており、JAVA_HOMEユーザー環境変数およびその他の環境変数が設定されていることを確認します。

   CPATをダウンロードして解凍した後、CPATが実行されるマシンに適切なJava Runtime Environment (JRE)がインストールされていることを確認します。CPATに必要なJREの最小バージョンはJava 7です。

   CPATは、環境変数JAVA_HOMEおよびORACLE_HOMEを使用してJREホームを検索します。ORACLE_HOMEのJavaのバージョンがJava 6以前のリリースの場合(Oracle Database 12gリリース1以前のホームの場合のみ)は、JAVA_HOMEをJava 7 (以上)のJREを指すように設定します。ORACLE_HOMEでJavaをアップグレードするには、https://support.oracle.comにアクセスし、Oracle Database 11gデータベースの場合はドキュメント2366614.1 (パッチID 25803774)、Oracle Database 12.1データベースの場合はドキュメント2495017.1 (パッチID 27301652)を検索します。

   Microsoft WindowsシステムでJAVA_HOMEを設定するには:

   a. 「マイ コンピュータ」を右クリックし、「プロパティ」を選択します。

   b. 「詳細設定」タブで「環境変数」を選択し、JAVA_HOMEを編集してJava Runtime Environment (JRE)の場所を指し示すようにします。

      たとえば:

      ```
      C:¥Program Files¥Java¥jdk1.8¥jre
      ```

      JREはJava Development Kit (JDK)の一部ですが、個別にダウンロードできます。

   LinuxまたはUnixシステム(KornまたはBashシェル)でJAVA_HOMEを設定するには:

   ```
   export JAVA_HOME=jdk-install-dir
   export PATH=$JAVA_HOME/bin:$PATH
   ```

   ノート:

   Linux および Unix では、Oracle Database に付属の oraenv スクリプトを使用して、ORACLE_SID、ORACLE_HOME および LD_LIBRARY_PATH 変数を設定することをお薦めします。

   ORACLE_HOMEを定義せずにCPATを使用する必要があり、Oracle CallインタフェースのJDBC接続文字列を使用する必要がない場合は、JAVA_HOMEがJava 7 (以上)のJREに設定されていることを確認します。可能な場合は、Java 8以上のJREを使用することをお薦めします。他にも、OJDBC8 jarの機能を使用すると、Oracle Cloudインスタンスへの接続時に使用される接続など、ウォレットベースの接続が簡素化されます。

関連トピック

- クラウド移行前アドバイザ・ツール(CPAT)によるクラウド移行の適切性についてのデータベースの分析(ドキュメントID 2758371.1)

親トピック: クラウド移行前アドバイザ・ツール

20.3 クラウド移行前アドバイザ・ツール(CPAT)の開始

Oracle SQLclまたはCPATをダウンロードした後、ソース・データベースに必要なJavaホームがあることを確認し、環境変数を設定して、実行するチェックの種類を決定します。

クラウド移行前アドバイザ・ツール(CPAT)を使用するためのワークフローは次のとおりです:

1. 移行先のクラウド・データベースのタイプを決定します。
2. CPATを実行し、`gettargetprops`を使用してCPATプロパティ・ファイルを生成します。このスイッチは、ターゲット・データベースのプロパティ(作成されている場合)を収集します。ターゲット・プロパティは、フォーカスするソース・データベースの分析時に使用され、実行するチェックをターゲット・データベースに必要なチェックに制限します。
3. 移行シナリオに必要なオプションを指定してCPATを実行します。CPATを実行して、様々な移行シナリオをテストできます。CPATを繰り返し実行する場合は、テストを区別するために、`--outfileprefix`および`--outdir`スイッチを使用して出力を整理し、レポートが上書きされないようにすることをお薦めします。

CPATパッチ配布キットには、LinuxおよびUnixプラットフォームでCPATを実行するための`premigration.sh`と、Microsoft WindowsプラットフォームでCPATを実行するための`premigration.cmd`が含まれています。CPATは、分析するデータベース・インスタンスへのネットワーク・アクセスを持つ任意のホストから実行できます。

> ノート:
>
> サーバーで移行前スクリプトを実行しても、Oracle Database は変更されません。CPAT 自体では、ユーザーまたはパッケージは作成されず、ロールまたは権限を付与する必要はありません。データベースは `READ ONLY` として処理されます。データベース・メタデータのみをチェックし、アプリケーションまたはビジネス・データはチェックしません。

この例では、`premigration.sh`が使用されています(Microsoft Windowsシステムでは`premigration.cmd`を使用します)

例20-1 CPATプロパティ・ファイルの生成

この例では、ソース・データベースがOracle Autonomous Database Shared for Transaction Processing and Mixed Workloads (ATP-S)に移行する準備ができているかどうかを確認し、要件のプロパティ・ファイルを生成します:

```
premigration.sh --connectstring ¥
'jdbc:oracle:thin:@db_tp_tunnel?TNS_ADMIN=/path/to/wallets/Wallet1' --username ADMIN
¥
--gettargetprops --outdir migration
```

このコマンドの出力は次のようになります:

```
Enter password for ADMIN user: Cloud Premigration Advisor Tool Version 22.10.0
Cloud Premigration Advisor Tool generated properties file location:
/home/oracle/migration/configprops/atps_premigration_advisor_analysis.properties
```

> ノート:
>
> CPAT を`--username` スイッチで実行する場合、指定する Oracle ユーザー名には `SELECT ANY DICTIONARY` 権限が必要であり、`SYSTEM.DUM$COLUMNS` および `SYSTEM.DUM$DATABASE` に `SELECT` が付与されている必要があります。`DUM$`表へのアクセスが必要になるのは、ソースとターゲットの文字セットで Oracle Database Migration Assistant for Unicode (DMU)が必要であることが示されている場合のみです。

20.4 クラウド移行前アドバイザ・ツールの接続文字列

クラウド移行前アドバイザ・ツール(CPAT)は、標準のOracle JDBC形式の接続文字列を受け入れます。

標準のOracle JDBC形式の接続文字列を使用すると、接続に「シック」または「シン」のOracle JDBCドライバを使用できます。

表20-1 JDBC接続文字列の例

接続の説明	接続文字列	ノート
シン・クライアント	`jdbc:oracle:thin:@host:port:sid`	変数 `host`、`port` および `sid` を、接続ポートのホストおよびソースのシステム識別子に置き換えます。
PDB サービスを使用したシン・クライアント	jdbc:oracle:thin:@host:port/pdb-service-name	変数 `host`、`port` および `PDB-service-name` を、接続ポートのホストおよびソースの PDB サービス名に置き換えます。
AWS RDS を使用したシン・クライアント	`jdbc:oracle:thin:@database-1.xxx.us-east-1.rds.amazonaws.com:port:sid`	データベースのエンドポイントおよびポートの詳細を検索する手順は、Amazon Web Services Relational Database (AWS RDS)のドキュメントを参照してください。
オペレーティング・システム認証	`jdbc:oracle:oci:@`	CPAT コマンドラインには、プロパティ`--sysdba` も含める必要があります
PDB を使用したオペレーティング・システム認証	`jdbc:oracle:oci:@`	CPAT コマンドラインには、プロパティ`--sysdba` および`--pdbname PDB-name` も含める必要があります。ここで、`PDB-name` は PDB の名前です。
Java 8 JRE を使用したウォレットベース	`jdbc:oracle:thin:@service-name?TNS_ADMIN=path-to-wallet`	`TNS_ADMIN` 接続プロパティでは、`path-to-wallet` で表される次のものを指定します: `tnsnames.ora` の場所。 Oracle Wallet (`ewallet.sso`、`ewallet.p12`)または Java キーストア(JKS)ファイル(`truststore.jks`、`keystore.jks`)の場所。 `ojdbc.properties` の場所。このファイルに

接続の説明	接続文字列	ノート
		は、Oracle Wallet または Java KeyStore (JKS)を使用するために必要な接続プロパティが含まれています。 キーストアの使用方法の詳細は、Oracle Autonomous Database のドキュメントを参照してください。

追加接続文字列情報

`--pdbname`プロパティを使用する必要があるのは、接続文字列が`CDB$ROOT`用である場合のみです。

`JDBC:oracle:thin:@service-name?TNS_ADMIN=path-to-wallet`などのキーストア接続文字列を使用する場合、JDBCでは次のいずれかがtrueである必要があります:

- `ojdbc.properties`ファイルはウォレット・ディレクトリにあり、`oracle.net.wallet_location=(SOURCE=(METHOD=FILE)(METHOD_DATA=(DIRECTORY=${TNS_ADMIN})))`などの値を持つ`oracle.net.wallet_location property` を含みます
- `JAVA_TOOL_OPTIONS`環境変数は、次のような適切な値で設定されます:

  ```
  export JAVA_TOOLS_OPTIONS='-Doracle.net.tns_admin=path-to-wallet-dir -Doracle.net.wallet_location=(SOURCE=(METHOD=FILE)(METHOD_DATA=(DIRECTORY=path-to-wallet-dir)))'
  ```

関連トピック

- Oracle Database Insider: データ・ポンプを介したAWS RDSからOracle Autonomous Databaseへの移行
- Oracle Autonomous Database on Shared Exadata Infrastructureの使用: JDBCシン・ドライバおよびウォレットでのJDBC URL接続文字列の使用

親トピック: クラウド移行前アドバイザ・ツール

20.5 クラウド移行前アドバイザ・ツールに必要なコマンドライン文字列

ユースケースに応じて、クラウド移行前アドバイザ・ツール(CPAT)の実行には一部の文字列が必要です。

CPATを使用してソース分析用のデータベースに接続する場合、コマンド文字列には3つの必須プロパティがあります。すなわち、クラウド・ターゲット(`targetcloud`)を指定するプロパティ、接続文字列(`connectstring`)を指定するプロパティ、ユーザー認証文字列(`sysdba`または`username`プロパティのいずれかで指定)を指定するプロパティです。

最初の2つのコマンド・プロパティは常に次である必要があります

- `--targetcloud type` (または`-t type`)。ここで、`type`は`Oracle Cloud`ターゲット・タイプです
- `--connectstring JDBC-connect-string`または`-c JDBC-connect-string`。ここで、`JDBC-connect-string`は、移行ソースのOracle Databaseへの接続に使用するJDBC接続文字列です。

その他の必須プロパティではユーザー資格証明が提供されるため、分析の開始に使用するユーザー資格証明によって異なります:

- ユーザー・アカウントによるオペレーティング・システム認証、または`SYS`ユーザーを使用したローカル・システムでの認可の

場合は、--sysdbaまたは-dを使用します。これにより、AS SYSDBAを使用してソース・データベースに接続することでCPATが起動します。SYSDBAが付与されているが、CPATで必要な他の権限が付与されていないユーザーとして接続する場合、この認証オプションも必要です。

- ウォレットまたはオペレーティング・システム認証を使用していないユーザー・アカウントによる認証の場合は、--username nameまたは-u nameを使用します。ここで、nameは、ソース・システムへのログインに使用するユーザー・アカウント名です。実行時に、そのユーザーのパスワードの入力が求められます。指定するユーザー名は、SYSDBAおよびADMIN権限が付与されたユーザー・アカウントである必要があります。

  --usernameプロパティを使用してCPATを認証する場合、指定するOracleユーザー名にはSELECT ANY DICTIONARY権限が必要であり、SYSTEM.DUM$COLUMNSおよびSYSTEM.DUM$DATABASEにSELECTが付与されている必要があります。DUM$表へのアクセスが必要になるのは、ソースとターゲットの文字セットでOracle Database Migration Assistant for Unicode (DMU)が必要であることが示されている場合のみです。

親トピック: クラウド移行前アドバイザ・ツール

20.6 FULLモードおよびSCHEMAモード

クラウド移行前アドバイザ・ツール(CPAT)は、インスタンス全体に対して、またはスキーマに対して実行できます。

FULLモード

FULLモードがデフォルト・モードです。このモードでは、選択した移行方法およびクラウド・ターゲット・タイプに関連するチェックが実行され、Oracleによって保守されていないすべてのスキーマのデータが分析されます。FULLモードでは、SCHEMA、INSTANCEおよびUNIVERSALスコープ・チェックが実行されます。

> ノート:
>
> FULL モードでも、デフォルトで、Oracle によって保守されることがわかっているスキーマ内のデータのチェックは除外されます。--excludeschemas プロパティを使用しても、CPAT のデフォルトの FULL モードは変更されません。

SCHEMAモード

SCHEMAモードは、--schemasプロパティを使用して設定されます。--schemasが設定されており、--fullも指定されていない場合、CPATはSCHEMAモードで実行されます。SCHEMAモードでは、SCHEMAおよびUNIVERSALスコープ・チェックが実行されます。INSTANCEスコープ・チェックは実行されません。

CPATモードの制御

CPATモードは、次の2つのオプションのプロパティを使用して制御されます:

- スキーマ・プロパティ(--schemas 'schemaname' ['schemaname''schemaname'])は、リストしたスキーマに対して、1つ以上のスキーマ名のスペース区切りのスキーマ名リストでチェックを実行します。この場合、名前は一重引用符で囲んで指定します。スキーマ・モードでは、SCHEMAおよびUNIVERSALスコープ・チェックが実行されます。INSTANCEスコープ・チェックは実行されません。
- 全体プロパティ(--full)は、ソース・データベース・インスタンス全体に対してチェックを実行します。

--schemasプロパティの値を指定しない場合、デフォルトはFULLモードです。

コマンドラインで--schemasを指定した場合、コマンドラインで--fullも指定しないかぎり、CPATはSCHEMAモードで実行されます。両方のプロパティを使用する場合、SCHEMA、INSTANCEおよびUNIVERSALスコープ・チェックは実行されますが、-

schemasリストのスキーマのリストでのみ実行されます。

スキーマ名が小文字であるか、大文字と小文字であるか、特殊文字を含んでいる場合は、二重引用符と一重引用符を使用してスキーマ名を指定します。たとえば:

```
premigration.sh --schemas 'PARdUS' '"ComEDIT"' '"faciem.$meam"' --targetcloud
ATPS --connectstring jdbc-connect-string"
```

親トピック: クラウド移行前アドバイザ・ツール

20.7 クラウド移行前アドバイザ・ツール(CPAT)レポート・データの解釈

CPATにより生成されたレポートには、要約情報および正常に実行された各チェックの詳細が含まれます。

チェックごとに、移行前アドバイザ・レポートに次の情報が含まれます:

- 説明: このフィールドは、チェックの対象、またはチェックの実行理由を示します。
- 影響: このフィールドは、合格以外の結果の重要性を説明しています。
- アクション: このチェックでは、チェック結果が合格でない場合に問題を修正するために移行前に実行する必要がある操作について説明します。

CPATのチェックを実行するたびに、レポート・ステータスは合格、「レビュー推奨」、「レビュー必須」または「アクション必須」になります。

CPATレポートの全体的な結果は、実行されたすべてのチェックの最も重大な結果になります。たとえば、30のチェックのステータスが合格、1つのチェックのステータスが「レビュー必須」の場合、全体的な結果は「レビュー必須」になります。

各CPATチェック結果の現在の定義は次のとおりです:

表20-2 移行前アドバイザ・ツール(CPAT)のチェック結果の定義

チェック	定義
合格	移行が成功し、アプリケーションの動作に違いがないことを示します。
レビュー推奨	移行が成功し、アプリケーションに機能的な違いがない可能性が高いことを示します。ただし、データベース管理者は、移行前に、このステータスの各チェックを評価して潜在的な問題を探す必要があります。
レビュー必須	移行が(少なくとも一部)成功する可能性があるが、ソース・データベースで行ったとおりにすべてが実行されるとは予想できないか、データベース管理者が移行後に追加の作業を完了してターゲット・インスタンスをソース・データベースと連携させる必要があることを示します。
アクション必須	移行が失敗する原因となる可能性が高いものを示します。通常、この結果でのチェックは、移行を試行する前に解決する必要があります。
失敗	クラウド移行前アドバイザは分析を完了できませんでした。Oracle サポート・サービスに連絡してください。

ノート: CPATの結果の「アクション必須」は、たとえば、データのインポート中にOracle Data Pumpのインポートが途中で終了することを必ずしも意味するわけではありません。これは、インポート中にエラーが発生する可能性が高いことを意味し、一部のデータが移行されていないことを示す可能性があります。データベースとデータベースでサポートされているアプリケーションの両

方をよく理解している管理者が、合格ではないチェックの結果を調べる必要があります。

チェックがスキップ済とマークされることがある理由

クラウド移行前アドバイザ・レポートでスキップ済としてマークされたチェックは、CPATコマンドで提供されるプロパティ(たとえば、`--targetcloud --migrationmethod`、またはその他のレポート値)のCPAT分析中に完了する必要がありましたが、この特定の移行前アドバイザ・レポートでは実行されませんでした。

次の2つのケースのいずれかがスキップ済ステータスの原因です:

- チェックは実行する必要がありますが、ソース・データベースの現在の内容または構成が原因で、レポートの生成時に実行できません。この場合、チェック結果は「レビュー推奨」以上の重大性になります。
- ソース・データベースの現在の内容または構成により、レポート時にチェックを完了する必要はありません。この場合のチェック結果は合格になります。

親トピック: クラウド移行前アドバイザ・ツール

20.8 移行前アドバイザ・ツールを使用するためのベスト・プラクティス

これらのクラウド移行前アドバイザ・ツール(CPAT)のヒントは、CPATをより効果的に使用するのに役立ちます。

- ターゲット・データベース・インスタンスでのプロパティ・ファイルの生成
  移行前アドバイザ・ツール(CPAT)のプロパティ・ファイルをターゲット・データベース・インスタンスで生成することをお薦めします。
- CPAT分析の焦点
  移行前アドバイザ・ツール(CPAT)分析に重点を置き、CPATで調べるスキーマを制限することをお薦めします。
- レポートのデータ量の削減
  一部のクラウド移行前アドバイザ・ツールのチェックでは、同じ懸念事項を持つ数千のオブジェクトが返されることがあります。レポート・サイズを小さくする方法をここで示します。
- JSONレポートの生成およびログの保存
  テキスト・レポートのみを使用する予定の場合でも、クラウド移行前アドバイザ・ツール(CPAT)を使用してJSON出力ファイルを生成し、診断のためにログ・ファイルを保存することを提案します。
- 出力接頭辞を使用した様々な移行シナリオの記録
  様々な移行オプションのレポートを追跡するには、CPATコマンドラインで`--outfileprefix`および`--outdir`プロパティを使用します。

親トピック: クラウド移行前アドバイザ・ツール

20.8.1 ターゲット・データベース・インスタンスでのプロパティ・ファイルの生成

移行前アドバイザ・ツール(CPAT)のプロパティ・ファイルをターゲット・データベース・インスタンスで生成することをお薦めします。

ソース・データベース・インスタンスの最も完全でターゲットを絞った分析を実行するには、ターゲット・データベース・インスタンスの特定のプロパティが必要です。このため、移行するデータベース・インスタンスでCPATプロパティ・ファイルを生成する必要があります。この関数を実行するために、`--gettargetprops`プロパティは、他の接続関連プロパティとともに使用することを目的としています。
次の例では、CPATスクリプトは、ターゲット・データベース・インスタンスでユーザーADMINによって実行されます:

```
./premigration.sh --gettargetprops -username ADMIN --connectstring
'jdbc:oracle:thin:@service-name?TNS_ADMIN=path-to-wallet'
```

このコマンドでは、ソース・インスタンスの分析に使用できるプロパティ・ファイル`premigration_advisor_analysis.properties`が生成されます。

必要に応じて、ターゲットで生成されたプロパティ・ファイルをソース・データベース分析が実行されるホストにコピーし、`--analysisprops`プロパティを使用してそのファイルをCPATに提供できます。

たとえば:

```
./premigration.sh --connectstring jdbc:oracle:oci:@ --targetcloud ATPD --sysdba --
analysisprops premigration_advisor_analysis.properties
```

ユーザー(またはOracle Zero Downtime Migration (ZDM)またはOracle Database Migration Service (DMS))が必要なすべての表領域をマッピング(または事前作成)することがわかっている場合は、CPATに指定したプロパティ・ファイルにプロパティ`MigrationMethodProp.ALL_METHODS.TABLESPACE_MAPPING=ALL`を追加します。このプロパティ設定により、表領域関連チェックの大部分(すべてではありません)が合格します。ただし、このオプションを選択した場合は、表領域マッピングを使用した割当て制限に関連する移行の問題が引き続き発生する可能性があることに注意してください。

親トピック: 移行前アドバイザ・ツールを使用するためのベスト・プラクティス

20.8.2 CPAT分析の焦点

移行前アドバイザ・ツール(CPAT)分析に重点を置き、CPATで調べるスキーマを制限することをお薦めします。

CPATが分析中に調べるスキーマを制限するには、`--schema`スイッチ・プロパティの使用を検討してください。`--schemas list` (`list`はスキーマのスペース区切りリスト)を使用してCPATを起動すると、それらのスキーマに対してのみチェックが実行されます。`--schemas`スイッチを使用しない場合、ソース・インスタンス内のすべてのスキーマ(Oracle管理スキーマを除く)が分析されます。この結果、移行する予定のないスキーマで問題が見つかる可能性があります。`--schemas`プロパティを使用してスコープを制限すると、ソース・インスタンスが複数のアプリケーションをホストしている場合に特に便利です。この場合、各アプリケーションが異なるOracle Autonomous Databaseインスタンスに移行される可能性があります。
次の例では、CPATスクリプトがターゲット・データベース・インスタンスでユーザー`ADMIN`によって実行され、スキーマ`schema1`および`schema2`に対する分析が実行されます:

```
./premigration.sh -username SYSTEM --connectstring 'jdbc:oracle:thin:@service-
name?TNS_ADMIN=path-to-wallet' --schemas schema1 schema2
```

`--schemas`スイッチ・プロパティでは、CPATにスキーマ(schema1およびschema2)のスペース区切りリストが提供されるため、実行されるチェックはこれら2つのスキーマのみに制限されます。

親トピック: 移行前アドバイザ・ツールを使用するためのベスト・プラクティス

20.8.3 レポートのデータ量の削減

一部のクラウド移行前アドバイザ・ツールのチェックでは、同じ懸念事項を持つ数千のオブジェクトが返されることがあります。レポート・サイズを小さくする方法をここで示します。

実行するチェックによっては、一部のCPATチェックによって、テキスト・レポートの複数のオブジェクトで同じ問題の結果が返される場合があります。結果の数を減らすには、`--maxtextdatarows n`関数を使用できます。ここで、nは、表示する行数を指定する整数です。

`--maxrelevantobjects n`プロパティはレポートに対して同じ関数を実行しますが、通常はJSONレポートのサイズを制限する必要はありません。

次の例では、CPATスクリプトは、ターゲット・データベース・インスタンスでユーザー`SYSTEM`によって実行され、出力は最大10行

のテキスト・ファイル・データを返すように設定されています:

```
./premigration.sh --username SYSTEM --connectstring 'jdbc:oracle:thin:@service-
name?TNS_ADMIN=path-to-wallet --maxtextdatarows 10"
```

親トピック: 移行前アドバイザ・ツールを使用するためのベスト・プラクティス

20.8.4 JSONレポートの生成およびログの保存

テキスト・レポートのみを使用する予定の場合でも、クラウド移行前アドバイザ・ツール(CPAT)を使用してJSON出力ファイルを生成し、診断のためにログ・ファイルを保存することを提案します。

JSONレポートとテキスト・レポートを生成し、常にログ・レポート・ファイルを保存することをお薦めします。なぜでしょうか。CPATの使用中に問題が発生し、Oracle Supportに連絡する必要がある場合は、Oracle Supportによる問題の解決に役立つ可能性のあるすべての情報を提供できます。Oracle Supportを支援するには、テキスト・レポートとJSONレポートの両方、およびCPATによって生成された`.log`レポートを発行する準備をします。`--reportformat`プロパティは、1つ以上のスペース区切りのレポート形式を受け入れます。`--reportformat`スイッチに許可される値は、`json`および`text`です。
たとえば:

```
./premigration.sh -username SYSTEM --connectstring 'jdbc:oracle:thin:@service-name --
reportformat json text
```

親トピック: 移行前アドバイザ・ツールを使用するためのベスト・プラクティス

20.8.5 出力接頭辞を使用した様々な移行シナリオの記録

移行オプションごとにレポートを追跡するには、CPATコマンドラインで`--outfileprefix`および`--outdir`プロパティを使用します。

様々なクラウド移行オプションのレポートを生成するには、クラウド移行前アドバイザ・ツール(CPAT)を`--outfileprefix`とともに使用して、生成したレポート・オプションを編成できるレポートおよびログ・ファイルに接頭辞を付けます。`--outdir`プロパティを使用して、様々なインスタンスのレポートを編成したり、様々なシナリオのレポートを編成することもできます。

> ノート:
>
> `--outdir` プロパティは、絶対フォルダ・パスまたは相対フォルダ・パスのいずれかを受け入れます。このプロパティを使用して、ログ・ファイル、レポート・ファイル、および生成するプロパティ・ファイルが作成される特定の場所を指定します。コマンドラインから`--outdir` を省略すると、ログ・ファイルおよびその他の生成ファイルがユーザーの現在のフォルダに作成される結果として、複数の分析が実行されるときにファイルが上書きされる可能性があります。

たとえば:

```
./premigration.sh --outfileprefix ATPS_RUN_01 --outdir /path/CPAT_output --
reportformat TEXT JSON ...
```

親トピック: 移行前アドバイザ・ツールを使用するためのベスト・プラクティス

20.1 クラウド移行前アドバイザ・ツールとは

クラウド移行前アドバイザ・ツール(CPAT)は、Oracle Database内のデータベース・メタデータを分析する移行アシスタントであり、Oracle Cloud内のOracle Autonomous Databaseにデータを移動する際に役立つ情報を提供します。

クラウド移行前アドバイザ・ツール(CPAT)は、Oracle Cloudまたはオンプレミス内のOracle Databaseへの正常な移行を計画するために役立つように用意されています。ソース・データベースとデータベース・ターゲットおよび選択した移行方法との互換性を分析し、潜在的な非互換性に対する一連のアクションを提案します。CPATでは、様々な移行ツールについて考慮する情報が提供されます。

クラウド移行前アドバイザ・ツールの実行には、ソース・データベースへの変更は必要ありません。ユーザーの追加、ロールの付与、またはパッケージのロードは必要ありません。

クラウド移行前アドバイザ・ツールのしくみ

クラウド移行前アドバイザ・ツールでは、ソース・データベース・メタデータに対するチェックが実行され、移行についての情報が提供されます。実際の移行は実行されません。その情報を移行計画の一環で使用します。CPATは、Java 7以降のリリースを使用して実行され、Java 8 Java Runtime Environment (JRE)が推奨されます。

ノート:

CPAT をインストールして実行しても、Oracle Database は変更されません。CPAT は、ユーザーやパッケージを作成したり、ロールや権限の付与を必要としません。データベースは READ ONLY として処理されます。

チェックにより、データベース、データベース・オブジェクト、ユーザーまたはコンポーネントについてプログラム的に判断できます。チェックは、特定のOracle Cloudデータベース・デプロイメント・オプションに移動するためのデータベースおよびデータベース・スキーマの適合性を判断することを目的としています。例: Oracle Data Pumpなどの特定の移行方法を使用するOracle Autonomous Database on Shared Exadata Infrastructure (ADB-S)。

ソース・データベースとは、Oracle Autonomous Databaseへの移行の適切性についての分析対象となるデータベースです。ターゲットとは、特定のOracle Autonomous Database、またはCPATの実行時に選択できる一般的なOracle Autonomous Databaseデプロイメント・オプションです。

CPATは、Javaコマンドライン・ツールとして、またはSQLclを使用してSQLコマンドライン・ツールとして実行することで起動します。次に、ソース・データベースとOracle Autonomous Databaseターゲットを指定するか、他のOracle Cloud Infrastructure (OCI)ターゲット・データベース(Exadata Cloud@Customer、Exadata Cloud Service、オンプレミス・データベースなど)にDEFAULTを指定します。ソース・データベースおよびスキーマの内容に対して多数のチェックが実行されます。これらのチェックは、選択したターゲット、およびユーザーが意図した移行オプションに基づきます。

ソース・データベースのチェックが完了すると、検出された内容を示すレポートが生成されます。レポートには、合格、「レビュー推奨」、「レビュー必須」または「アクション必須」など、各チェックの要約情報と詳細(チェックの結果を含む)の両方が含まれています。また、移行に関連する可能性があるソース・データベース内の追加メタデータも識別されます。

チェック結果は、まとめられて、レポートに表示されます。レポートには、マシン可読レポート(JSON)、判読可能な形式(プレーン・テキスト)があります。これらのレポートは、Oracle Zero Downtime Migration (ZDM) Cloud ServiceやOracle Cloud Infrastructure (OCI) Database Migration Serviceなどの他のOracle移行製品および機能で直接使用することもできます。

移行前アドバイザ・ツールのプロパティ

コマンドラインでプロパティを指定して分析チェックの情報を指定することにより、CPATの実行方法および実行するチェックを指定できます。

クラウド移行前アドバイザ・ツールのレポート

CPATでは、特定の移行コマンドの使用、特定のデータベース・パラメータの設定、ソース・インスタンスまたはターゲット・インスタンスでのSQLスクリプトの実行など、関連するアクションが推奨されます。ターゲット・デプロイメント・オプションおよび実際のデータベース・ターゲットでチェックを実行できるため、ソース・データベースまたはターゲット・データベースで何かを完了する必要がある場合、レポートでは「ターゲット」ではなく「場所」という用語を使用します。レポートで特定のパラメータおよびコマンドの使用が提案された場合は、レポート内のガイダンスに従うことをお薦めします。

関連トピック

- クラウド移行前アドバイザ・ツール(CPAT)によるクラウド移行の適切性についてのデータベースの分析(ドキュメントID 2758371.1)

親トピック: クラウド移行前アドバイザ・ツール

20.9 コマンドライン構文およびプロパティ

クラウド移行前アドバイザ・ツール(CPAT)のプロパティを使用して、CPATコマンドライン構文で実行する必要のあるチェックおよびその他の操作を指定します。

- 移行前アドバイザ・ツールのコマンドライン構文
  コマンドライン・シェル・スクリプトとして、移行前アドバイザ・ツールを実行します。
- 移行前アドバイザ・ツールのコマンドライン・プロパティ
  移行前アドバイザ・ツールのプロパティを確認して、Oracle Database移行シナリオのコマンド・ツリーおよびオプションを作成します。

親トピック: クラウド移行前アドバイザ・ツール

20.9.1 移行前アドバイザ・ツールのコマンドライン構文

コマンドライン・シェル・スクリプトとして、移行前アドバイザ・ツールを実行します。

前提条件

- ソース環境にJava Development Kit (JDK) 7以降がインストールされている必要があります。Oracleでは、Java 8 Runtime Environment (JRE)を使用することをお薦めします。

  JDK 8は、Oracle Database 12cリリース2 (12.2)以降のリリースごとにインストールされます。12.2より前のリリースでは、ターゲットOracle DatabaseのJavaリリースを使用して移行前アドバイザ・ツール(CPAT)を実行するか、ソース・データベース・サーバーにJDK 8をインストールする必要があります。

Javaファイル・パス

My Oracle Supportから最新のCPAT zipファイルを取得します。そのアプリケーション、およびそのアプリケーションのデプロイメント手順は、My Oracle Supportノート2758371.1から入手できますCPATはJavaベースのツールであるため、ツールを実行するマシンに適切なJava Runtime Environment (JRE)をインストールする必要があります。

シン・クライアントの場合、CPATは環境変数JAVA_HOMEおよびORACLE_HOMEを使用してJava Runtime Environment (JRE)を検索します。JREは、これらのパスのいずれかである必要があります。

シック・クライアントの場合、CPATはOracle Call Interface (OCI)ベースのJDBC接続文字列を使用します。このタイプの接続文字列では、CPATは通常、環境変数ORACLE_SID、ORACLE_HOMEおよびLD_LIBRARY_PATHを使用してデータベースに接続します。

ノート:

CPAT を実行しているユーザーに対してオペレーティング・システム認証を使用する場合のみ、ORACLE_SID を設定する必要があります。必要であれば CPAT スクリプト自体で LD_LIBRARY_PATH を設定できるため、ほとんどの場合、必要な設定は ORACLE_HOME 環境変数のみです。

構文

移行前アドバイザ・ツールのコマンド構文では、大文字と小文字が区別されます。プロパティを渡すには、各コマンド・プロパティについて記載されているとおりに、一連の文字列またはテキスト文字列として指定します。

構文では次の書式が使用されます。characterは大文字と小文字が区別される単一の文字であり、command-string

は大文字と小文字が区別される文字列であり、valueはコマンド・プロパティで指定される入力オプションまたは値です。

Java

```
java -jar premigration.jar [-character [value] | --command-string [value]]
```

シェル・コマンド:

```
./premigration.sh [-character [value] | --command-string value[]
```

文字フラグまたはプロパティのフルネームを使用して、コマンド構文で複数のプロパティを連結できます。

親トピック: コマンド行構文およびプロパティ

20.9.2 移行前アドバイザ・ツールのコマンドライン・プロパティ

移行前アドバイザ・ツールのプロパティを確認して、Oracle Database移行シナリオのコマンド・ツリーおよびオプションを作成します。

- analysisprops
  移行前アドバイザ・ツールのプロパティanalysispropsでは、ソース・データベースのプロパティ・ファイルのパスおよび名前を指定します。
- connectstring
  移行前アドバイザ・ツールのプロパティconnectstringでは、ソース・データベース用のJDBC接続文字列を指定します。
- excludeschemas
  移行前アドバイザ・ツールのプロパティexcludeschemasでは、移行についての分析から除外するスキーマのリストを指定します。
- full
  移行前アドバイザ・ツール(CPAT)のプロパティfullでは、--schemasが使用されている場合でもすべてのチェックを実行することを指定します。
- gettargetprops
  移行前アドバイザ・ツールのプロパティgettargetpropsでは、移行ターゲット・データベース・インスタンスの接続プロパティを読み取り、ソース・データベース・インスタンスに対して分析します。
- help
  移行前アドバイザ・ツールのプロパティhelpでは、コマンドライン・ヘルプ情報が出力され、終了されます。
- logginglevel
  移行前アドバイザ・ツールのプロパティlogginglevelでは、ロギング・ファイルに記録される問題のレベルを指定します。
- maxrelevantobjects
  移行前アドバイザ・ツールのプロパティmaxrelevantobjectsでは、すべてのレポートに含まれる関連オブジェクトの最大数を指定します。
- maxtextdatarows
  移行前アドバイザ・ツールのプロパティmaxtextdatarowsでは、テキスト・レポートに表示される関連オブジェクトの行数制限を指定します(JSONレポートには適用されません)。
- migrationmethod
  移行前アドバイザ・ツールのプロパティmigrationmethodでは、Oracle Cloudへの移行に使用する方法またはツールのタイプを指定します。

- outdir
  移行前アドバイザ・ツールのプロパティoutdirでは、移行前分析ログ・ファイルおよびレポート・ファイルを生成するディレクトリ・パスを指定します。
- outfileprefix
  移行前アドバイザ・ツールのプロパティoutfileprefixでは、移行前アドバイザ・ツールのレポートの接頭辞を指定します。
- pdbname
  移行前アドバイザ・ツールのプロパティpdbnameでは、CPATでレポートを生成するCDBのソースPDBの名前を指定します。
- reportformat
  移行前アドバイザリ・ツール(CPAT)のプロパティreportformatでは、CPATレポート出力の形式を指定します。
- schemas
  移行前アドバイザ・ツールのプロパティschemasでは、移行について分析するスキーマのリストを指定します。
- sqltext
  移行前アドバイザ・ツールのプロパティsqltextでは、TEXTレポートのCPATチェックに使用するSQLを表示するように指定します
- sysdba
  移行前アドバイザ・ツールのプロパティsysdbaは、データベースへの接続時にAS SYSDBAを強制適用するために使用します。
- targetcloud
  移行前アドバイザ・ツールのプロパティtargetcloudでは、移行先のOracle Cloudデータベースのタイプを指定します。
- username
  移行前アドバイザ・ツールのプロパティusernameでは、ソース・データベースへの接続時に使用するユーザー名を指定します。
- version
  移行前アドバイザ・ツールのプロパティversionでは、CPATの現在のバージョンが出力され、終了されます。
- updatecheck
  移行前アドバイザ・ツールのプロパティupdatecheckでは、CPATの現在のバージョンが出力され、それより最新のバージョンがないかが確認されてから、終了されます。

親トピック: コマンド行構文およびプロパティ

20.9.2.1 analysisprops

移行前アドバイザ・ツールのプロパティanalysispropsでは、ソース・データベースのプロパティ・ファイルのパスおよび名前を指定します。

プロパティ	説明
プロパティ・タイプ	文字、文字列
構文	-a|--analysisprops --property-file-name

説明

移行前アドバイザ・ツールのanalysispropsプロパティでは、以前に移行前アドバイザ・ツールのコマンドライン・プロパティ--gettargetpropsの使用によってソース・データベース用に生成したプロパティ・ファイルのパスおよび名前を指定します。このプロパティ・ファイルを移行前アドバイザ・ツールで使用してデータベースのプロパティを分析します。

使用上のノート

コマンド文字列では、ソース・データベースに対してもオプション--connectString (-c)を指定し、移行するクラウド・データベースのタイプを指定するために--targetcloud (-t)を指定する必要があります。

例

この例では、ターゲット・インスタンスからプロパティ・ファイルpremigration_advisor_analysis.propertiesを取得し、analysispropsで使用するためそのファイルを特定します:

```
./premigration.sh --connectstring jdbc:oracle:oci:@ --targetcloud ATPD --sysdba ¥
--analysisprops premigration_advisor_analysis.properties
```

親トピック: 移行前アドバイザ・ツールのコマンドライン・プロパティ

20.9.2.2 connectstring

移行前アドバイザ・ツールのプロパティconnectstringでは、ソース・データベース用のJDBC接続文字列を指定します。

プロパティ	説明
プロパティ・タイプ	文字、文字列
構文	-c, --connectstring connect-string [--pdbname pdb-name]
デフォルト値	なし

説明

connectstringプロパティでは、ソース・データベース用のJDBC接続文字列を指定します。接続文字列がCDB用である場合、--pdbnameスイッチ、--pdbname pdb-nameを使用してPDB名も指定する必要があります。ここで、pdb-nameはソース・データベースを含むPDBの名前です。

CPAT接続では、次のステップを実行します:

1. primigration.shを使用してターゲット・インスタンスに接続し、そこからプロパティを取得します。この接続にはターゲット・インスタンスの接続情報が必要ですが、--targetcloudは必要ありません。このステップでは、premigration_advisor_analysisプロパティ・ファイルを作成します。connectstringが必要です。
2. 必要に応じて、ソース・インスタンスを分析するコンピュータに接続し、そのコンピュータにpremigration_advisor_analysis.propertiesファイルをコピーします。
3. ソース・インスタンスの接続情報を指定してpremigration.shを実行し、CPATレポートを生成します。

ターゲットに関するクラウド・サービス/ロックダウン情報を含むプロパティ・ファイルがある場合、--targetcloudは必要ありません。プロパティ・ファイルを指定しない場合、またはプロパティ・ファイルにクラウド・サービスが指定されていない場合は、最も関連性の高い情報を取得するために、--targetcloudまたは-tを使用してターゲット・クラウドを指定する必要があります。--targetcloudまたは-tを使用してターゲット・クラウドを指定しない場合、デフォルトは、PDBターゲットに既知のクラウド・サービス/ロックダウン・プロファイルが設定されていないクラウド・ターゲットです。

ノート:

ロックダウン・プロファイルによって適用される制限は PDB 全体を対象とし、その PDB のすべてのユーザー(SYS および SYSTEM など)に影響します。

例

次の例では、PDB名はsales1で、connect-stringは接続文字列が配置される場所を示します。

```
premigration.sh -c connect-string --pdbname sales1
```

親トピック: 移行前アドバイザ・ツールのコマンドライン・プロパティ

20.9.2.3 excludeschemas

移行前アドバイザ・ツールのプロパティexcludeschemasでは、移行についての分析から除外するスキーマのリストを指定します。

プロパティ	説明
プロパティ・タイプ	文字列

構文	`--excludeschemas --schemaname ['schemaname' 'schemaname' ...]`
	
	ここでの schemaname は 1 つ以上のスキーマ名であり、スペースで区切ります。
	
	スキーマ名では、大文字と小文字が区別されていると見なされます。たとえば、system ではなく SYSTEM を使用します。スキーマ名が小文字であるか、大文字と小文字であるか、特殊文字を含んでいる場合は、二重引用符と一重引用符を使用してスキーマ名を指定します。次に例を示します。
	
	`--excludeschemas '"MixedCase"' '"Special.Char$"'`

説明

移行前アドバイザ・ツールのexcludeschemasプロパティでは、クラウドへの移行の準備状況についての分析から除外するスキーマを指定します。

使用上のノート

移行前チェックを実行しないスキーマを指定するために使用します。excludeschemasを省略しており、schemasを使用していない場合は、データベース内のすべてのスキーマが分析されます。excludeschemasプロパティは、schemasとの併用はできません。

コマンド文字列では、ソース・データベースに対してもオプション--connectString (-c)を指定し、移行するクラウド・データベースのタイプを指定するために--targetcloud (-t)を指定する必要があります。

親トピック: 移行前アドバイザ・ツールのコマンドライン・プロパティ

20.9.2.4 full

移行前アドバイザ・ツール(CPAT)のプロパティfullでは、--schemasが使用されている場合でもすべてのチェックを実行することを指定します。

プロパティ	説明
プロパティ・タイプ	文字、文字列
構文	-f|--full

説明

各CPATチェックにはスコープが定義されています。チェックのスコープがINSTANCEの場合、FULLを選択して定義されたスコープをオーバーライドしないかぎり、そのチェックは実行されません。CPATのfullプロパティを指定すると、コマンド文字列にチェックのスコープを制限する--schemasが指定されている場合でも、ソース・データベースで強制的にすべてのチェックが実行されます。

使用上のノート

CPATで使用するオプションは、Oracle Data Pumpでも使用する必要があります。Oracle Data PumpをFULLモードで使用する場合は、fullプロパティを使用してCPATを実行する必要があります。Oracle Data PumpをSCHEMAモードで使用する場合は、CPATをschemaモードで実行します。

例

ソース・データベース・インスタンスに100のスキーマがあるが、s1、s2およびs3の3つのスキーマのみをAutonomous Transaction Processing Dedicated (ATP-D)に移行するとします。

この場合、すべてのスキーマを分析する必要はありませんが、3つのスキーマすべてに対してINSTANCE SCOPEDチェックを実行します。これを行うには、CPATを--schemas s1 s2 s3 --fullで実行します

親トピック: 移行前アドバイザ・ツールのコマンドライン・プロパティ

20.9.2.5 gettargetprops

移行前アドバイザ・ツールのプロパティgettargetpropsでは、移行ターゲット・データベース・インスタンスの接続プロパティを読み取り、ソース・データベース・インスタンスに対して分析します。

プロパティ	説明
プロパティ・タイプ	文字列
構文	-g|--gettargetprops プロパティ

説明

移行前アドバイザ・ツールのgettargetpropsプロパティでは、移行ターゲット・インスタンス用の接続パラメータをCPATで収集することを指定します。CPATで移行ターゲット・インスタンスのプロパティが収集されます。これにより、後でソース・データベース・インスタンスでそれらのプロパティを分析できるようになります。

使用上のノート

これらのプロパティは通常、移行フローでCPATを使用するツールによって設定され、これらのプロパティを使用して、移行中に特定の移行操作が実行中、または実行予定であることをCPATに指定します。`--gettargetprops`スイッチおよび`targetconnection`パラメータを使用してプロパティ・ファイルを生成します

詳細は、Microsoft Windowsシステムで`premigration.sh --help`または`premigration.com --help`を実行します。

例

```
./premigration.sh --gettargetprops --connectstring
jdbc:oracle:thin:@atpd_high?TNS_ADMIN=/path/wallet . . .
```

親トピック: 移行前アドバイザ・ツールのコマンドライン・プロパティ

20.9.2.6 help

移行前アドバイザ・ツールのプロパティ`help`では、コマンドライン・ヘルプ情報が出力され、終了されます。

プロパティ	説明
プロパティ・タイプ	文字列
構文	`-h|--help`

説明

移行前アドバイザ・ツールの`help`プロパティでは、コマンドライン・ヘルプの説明が出力され、このアドバイザが終了されます。

使用上のノート

このオプションを使用して、実行中の移行前アドバイザ・ツールのバージョンについてヘルプ情報を取得します。

例

```
premigration.sh --help
```

親トピック: 移行前アドバイザ・ツールのコマンドライン・プロパティ

20.9.2.7 logginglevel

移行前アドバイザ・ツールのプロパティ`logginglevel`では、ロギング・ファイルに記録される問題のレベルを指定します。

プロパティ	説明
プロパティ・タイプ	文字列
構文	`-l|--logginglevel -[severe|warning|info|config|fine|finer|finest]`
デフォルト	このプロパティをコマンド文字列に指定しない場合、デフォルトは

プロパティ	説明
	fine です。

説明

移行前アドバイザ・ツールのloggingleveルプロパティでは、移行前アドバイザ・ツールのレポート内に記録する必要がある問題の重大度を指定します

使用上のノート

ターゲット・データベースまたはデータベースで実行するチェックのタイプを指定するために使用します。ログのプロパティ:

- severe
- warning
- info
- config
- fine
- finer
- finest

親トピック: 移行前アドバイザ・ツールのコマンドライン・プロパティ

20.9.2.8 maxrelevantobjects

移行前アドバイザ・ツールのプロパティmaxrelevantobjectsでは、すべてのレポートに含まれる関連オブジェクトの最大数を指定します。

プロパティ	説明
プロパティ・タイプ	文字列
構文	-M|--maxrelevantobjects maximum-relevant-objects

説明

移行前アドバイザ・ツールのmaxrelevantobjectsプロパティでは、移行前アドバイザ・レポートに表示される関連オブジェクトの最大数を指定します。TEXTレポートの場合、このプロパティはmaxtextdatarowsプロパティより優先されます。

ノート:

レポートされるオブジェクト数に制限を指定すると、レポートで公開されていなくても移行に影響するオブジェクトが存在する可能性があります。

使用上のノート

このプロパティの目的は、CPATによって生成されるレポートに制限を指定することです:

- CPATレポートのサイズを制限します
- CPATが使用するメモリーを制限します
- 独自の、または機密の表、列またはその他の情報をレポートに含む可能性のあるオブジェクトを除外します。

例

```
premigration.sh  -maxrelevantobjects 5 -outfileprefix limit -targettype adws -
analysisprops /usr/example/CPAT/ cloud_premigration_advisor_analysis.properties
```

親トピック: 移行前アドバイザ・ツールのコマンドライン・プロパティ

20.9.2.9 maxtextdatarows

移行前アドバイザ・ツールのプロパティmaxtextdatarowsでは、テキスト・レポートに表示される関連オブジェクトの行数制限を指定します(JSONレポートには適用されません)。

プロパティ	説明
プロパティ・タイプ	文字列
構文	`-n|--maxtextdatarows maximum-number-of-data-rows`
デフォルト	データ表内のすべての行(上限なし)。

説明

移行前アドバイザ・ツールのmaxtextdatarowsプロパティでは、TEXTレポートに含まれる関連オブジェクト行の最大数を指定し、最大行数に達した後の行が表示されないことを示すメッセージが表示されます。このプロパティを設定しない場合は、すべての関連オブジェクトが含まれます(上限なし)。このプロパティは、JSONレポートには適用されません

使用上のノート

移行前アドバイザTEXTレポート・ファイルについては、プロパティ設定に競合がある場合、maxrelevantobjectsがmaxtextdatarowsの設定より優先されます。

例

親トピック: 移行前アドバイザ・ツールのコマンドライン・プロパティ

20.9.2.10 migrationmethod

移行前アドバイザ・ツールのプロパティmigrationmethodでは、Oracle Cloudへの移行に使用する方法またはツールのタイプを指定します。

プロパティ	説明
プロパティ・タイプ	文字列
構文	`-m|--migrationmethod -['datapump'|'goldengate']`
デフォルト	値が指定されていない場合のデフォルトは datapump です。

説明

移行前アドバイザ・ツールのmigrationmethodプロパティでは、クラウドへのデータベースの移行に使用する方法またはツー

ルのタイプを指定します。その移行方法によって、ソース・データベースでどのチェックが実行されるかが変わります。ソース・データベースで検出された内容は、その移行方法と互換性がない場合は、生成されたレポートで示されます。

使用上のノート

ターゲット・データベースまたはデータベースで実行するチェックのタイプを指定するために使用します。

オプション	説明
`datapump`	選択した Oracle Cloud デプロイメントへの移行の実行に Oracle Data Pump を使用する場合のチェックをアップグレード前アドバイザ・ツールで実行することを指定します。
`goldengate`	選択した Oracle Cloud デプロイメントへの移行の実行に Oracle GoldenGate を使用する場合のチェックをアップグレード前アドバイザ・ツールで実行することを指定します。

例

次の例では、`connect-string`は接続文字列が配置される場所を示します。ターゲットのOracle Cloudデータベースは Autonomous Transaction Processing Sharedで、選択した移行方法はOracle GoldenGateです。

```
premigration.cmd --connectstring some-string --targetcloud atps --username SYSTEM -
migrationmethod 'goldengate'
```

親トピック: 移行前アドバイザ・ツールのコマンドライン・プロパティ

20.9.2.11 outdir

移行前アドバイザ・ツールのプロパティ`outdir`では、移行前分析ログ・ファイルおよびレポート・ファイルを生成するディレクトリ・パスを指定します。

プロパティ	説明
プロパティ・タイプ	文字列
構文	`-o|--outdir directory-path` ここでの `directory-path` は、ログ・ファイルおよびレポートのディレクトリのパスです。

説明

移行前アドバイザ・ツールの`outdir`プロパティでは、ログ・ファイルおよびレポート・ファイルを作成する場所を指定します。

使用上のノート

指定したパスが絶対パスでない場合、移行前アドバイザ・ツールでは、CPAT起動元のファイル・パスの場所を基準としてそのディレクトリが指定されます。出力ファイル名を指定しなかった場合の、デフォルトのファイル名は`premigration`です。そのファイル名が存在しない場合は、CPATによって作成されます。

例

次の例では、connect-stringは接続文字列が配置される場所を示します。ターゲットPDBはtrend1、Oracle Cloudデータベースは Autonomous Data Warehouse Dedicated、出力ディレクトリ・パスは/users/analytic/adwd-migrです。

```
premigration.cmd --connectstring connect-string --targetcloud adwd --username SYSTEM
--pdbname trend1 -outdir /users/analytic/adwd-migr
```

親トピック: 移行前アドバイザ・ツールのコマンドライン・プロパティ

20.9.2.12 outfileprefix

移行前アドバイザ・ツールのプロパティoutfileprefixでは、移行前アドバイザ・ツールのレポートの接頭辞を指定します。

プロパティ	説明
プロパティ・タイプ	文字列
構文	-P|--outfileprefix prefix-string

説明

移行前アドバイザ・ツールのoutfileprefixプロパティでは、ソース・データベースについて生成された出力レポートに付ける接頭辞を指定します。接頭辞がない場合の、移行前アドバイザ・ツールのレポートまたはログの標準名はpremigration_advisorです。

使用上のノート

様々なレポート出力を区別するには、接頭辞を使用します。たとえば、あるデータベースに関して、一方でOracle GoldenGateによる移行についてレポートを生成し、他方でOracle Data Pumpによる移行についてレポートを生成する場合、あるいはCDBでPDBごとに個別のレポートを生成する場合に、接頭辞の使用によってそれらのレポートを区別できます。

例

次の例では、接頭辞文字列はcdb4、connect-stringは接続文字列が配置される場所を示し、移行ターゲットのOracle Cloudデータベースは Autonomous Transaction Processing Sharedです。このコマンドのレポートは、cdb4_premigration_advisor_report.txt およびcdb4_premigration_advisor.logです。

```
java -jar premigration.jar -c connect-string --targetcloud atps -P cdb4
```

親トピック: 移行前アドバイザ・ツールのコマンドライン・プロパティ

20.9.2.13 pdbname

移行前アドバイザ・ツールのプロパティpdbnameでは、CPATでレポートを生成するCDBのソースPDBの名前を指定します。

プロパティ	説明
プロパティ・タイプ	文字列
構文	-p|--pdbname pdbname

説明

接続先のPDB名です。ソース・データベースの接続文字列がCDBに対するものである場合のみ適用されます。

使用上のノート

このプロパティは、ソース・データベースの接続文字列がCDBに対するものである場合のみ使用する必要があります。

例

次の例では、connect-stringはソースCDB用に接続文字列が配置される場所を示します。ソースPDBは`trend4`、ターゲットはOracle Cloud Autonomous Data Warehouse Dedicatedデータベースです。

```
premigration.cmd --connectstring connect-string --targetcloud adwd --username SYSTEM
--pdbname trend4
```

親トピック: 移行前アドバイザ・ツールのコマンドライン・プロパティ

20.9.2.14 reportformat

移行前アドバイザリ・ツール(CPAT)のプロパティ`reportformat`では、CPATレポート出力の形式を指定します。

プロパティ	説明
プロパティ・タイプ	文字列
構文	`-r|--reportformat -format [format format]` ここでの `format` はレポート形式です。CPAT では、JSON または TEXT 形式がサポートされます。形式が複数ある場合はスペースで区切ります。このオプションを使用しない場合、レポートのデフォルトの形式は `txt` です。

説明

このリリースの時点では、移行前アドバイザ・ツールで、JSON形式またはテキスト形式でレポートを生成できます。`reportformat`プロパティを使用して、必要なレポート出力を指定します。

使用上のノート

生成するレポート出力のタイプを指定するために使用します。このプロパティが指定されていない場合、デフォルトはTEXTです。

> ノート:
>
> Oracle では、テキスト・レポートと JSON レポートの両方を指定し、常にレポートとログ・ファイルを保存することをお薦めします。移行中に問題が発生した場合は、ログ・ファイル、およびテキストと JSON レポートの両方を含む問題の解決に役立つ可能性のあるすべての情報を含めることが重要です。

オプション	説明
`json`	アップグレード前アドバイザ・ツールで JSON 形式でレポート

オプション	説明
	を生成することを指定します。
text	アップグレード前アドバイザ・ツールでテキスト・ファイル形式でレポートを生成することを指定します。

例

次の例では、JSONおよびテキスト形式のレポート出力が、Oracle Cloud Autonomous Data Warehouse Dedicatedデータベースをターゲットとするレポートに指定されています。生成されるレポートは、premigration_advisor_report.json、premigration_advisor_report.txtです。

```
premigration.cmd --connectstring connect-string --targetcloud adwd --username SYSTEM
--sqltext
```

親トピック: 移行前アドバイザ・ツールのコマンドライン・プロパティ

20.9.2.15 schemas

移行前アドバイザ・ツールのプロパティschemasでは、移行について分析するスキーマのリストを指定します。

プロパティ	説明
プロパティ・タイプ	文字列
構文	-s|--schemas 'schemaname' ['schemaname' 'schemaname' ...] ここでの schemaname は 1 つ以上のスキーマ名であり、スペースで区切ります。

説明

移行前アドバイザ・ツールのschemasプロパティでは、クラウドへの移行の準備状況をチェックするスキーマを指定します。その移行方法によって、ソース・データベースでどのチェックが実行されるかが変わります。ソース・データベースで検出された内容は、その移行方法と互換性がない場合は、生成されたレポートで示されます。

使用上のノート

レポートを、チェック実行対象のスキーマの特定のリストに制限するために使用します。スキーマ・モードでは、SCHEMAおよびUNIVERSALスコープ・チェックが実行されます。INSTANCEスコープ・チェックは実行されません。schemasを指定せず、excludeschemasを使用しない場合、デフォルトはfullプロパティを使用して実行されます。Oracleによって管理されるスキーマを除き、データベース内のすべてのスキーマが分析されます。これにより、クラウド・ターゲットに移行しないスキーマの問題がリストされているレポートを受信する可能性があります。

ノート:

> CPAT で使用するオプションは、Oracle Data Pump でも使用する必要があります。Oracle Data Pump を FULL モードで使用する場合は、full プロパティを使用して CPAT を実行する必要があります。Oracle Data Pump を SCHEMA モードで使用する場合は、CPAT を schema モードで実行します。

schemasプロパティは、excludeschemas.との併用はできません。チェックするスキーマのスコープの制限は、ソース・インスタンスが複数のアプリケーションをホストしており、それぞれ異なるOracle Autonomous Databaseインスタンスに移行する場合に特に役立ちます。

> ノート:
>
> --full プロパティを指定すると、ソース・データベースでチェックの完全セットを強制的に実行し、--schemas でチェックのスコープを制限する場合に有効になっている制限事項をオーバーライドします。

スキーマ名では、大文字と小文字が区別されていると見なされます。たとえば、systemではなくSYSTEMを使用します。スキーマ名が小文字であるか、大文字と小文字であるか、特殊文字を含んでいる場合は、二重引用符と一重引用符を使用してスキーマ名を指定します。次に例を示します。

```
--schemas '"MixedCase"' '"Special.Char$"'
```

例

次の例では、スキーマADMINおよびMixedCase (ターゲットがOracle Cloud Autonomous Data Warehouse Dedicatedデータベース、connect-stringがソース・データベースへの接続文字列)に対してレポートが生成されます。

```
premigration.cmd --connectstring connect-string --targetcloud atps --username ADMIN -
s 'SYSTEM' '"MixedCase"'
```

親トピック: [移行前アドバイザ・ツールのコマンドライン・プロパティ]

20.9.2.16 sqltext

移行前アドバイザ・ツールのプロパティsqltextでは、TEXTレポートのCPATチェックに使用するSQLを表示するように指定します

プロパティ	説明
プロパティ・タイプ	文字列
構文	-S|--sqltext

説明

移行前アドバイザ・ツールのsqltextプロパティは、デフォルトをオーバーライドし、TEXTレポートのCPATチェックに対して実行されたSQLを非表示にします。このプロパティは、JSONレポートには適用されませんオプションは使用しません。

使用上のノート

CPATは、SQL文を使用してデータベースのチェックを実行します。CPATレポートは、TEXT形式とJSON形式の両方で生成できます。デフォルトでは、各チェックに対して実行されたSQLはTEXTレポートに含まれません。TEXTレポートにSQLを表示するには、このパラメータを使用します。

例

```
premigration.cmd --connectstring connect-string --targetcloud adwd --username SYSTEM
--sqltext ImpModes TABLES MySchema MyTable
```

親トピック: 移行前アドバイザ・ツールのコマンドライン・プロパティ

20.9.2.17 sysdba

移行前アドバイザ・ツールのプロパティsysdbaは、データベースへの接続時にAS SYSDBAを強制適用するために使用します。

プロパティ	説明
プロパティ・タイプ	文字、文字列
構文	-d,--sysdba

説明

移行前アドバイザ・ツールのsysdbaプロパティでは、移行前アドバイザ・ツールがソース・データベースにAS SYSDBAで接続することを指定します。

使用上のノート

オペレーティング・システム認証またはSYSユーザーを使用している場合は、--sysdbaを使用する必要があります。また、--sysdbaを使用して、SYSDBAが付与されているユーザーとして接続する必要がありますが、CPATでチェックを実行するために必要な他の権限は接続しないでください。

例

```
./premigration.sh --connectstring jdbc:oracle:oci:@ --targetcloud ATPD --sysdba --
analysisprops premigration_advisor_analysis.properties
```

親トピック: 移行前アドバイザ・ツールのコマンドライン・プロパティ

20.9.2.18 targetcloud

移行前アドバイザ・ツールのプロパティtargetcloudでは、移行先のOracle Cloudデータベースのタイプを指定します。

プロパティ	説明
プロパティ・タイプ	文字列
構文	-t | --targetcloud cloudtype
デフォルト	DEFAULT は、ロックダウン・プロファイルが不明なターゲットを示します。

説明

このオプションは、移行前アドバイザ・ツールのtargetcloudプロパティで、移行先のクラウド・データベースのタイプを指定するために使用されます。構成ファイルにおいて、この値を、チェックするデータベースごとに異なる値に設定できます。

使用上のノート

移行するクラウドのタイプを識別するために使用します。これは、ソース・データベースで実行されるチェックの種類に影響します。

オプション	説明
'ATPD'	Oracle Autonomous Database Transaction Processing 専用
'ATPS'	Oracle Autonomous Database Serverless
'ADWD'	Oracle Autonomous Data Warehouse 専用
'ADWS'	Oracle Autonomous Data Warehouse Serverless。
'DEFAULT'	Oracle Autonomous Database on Exadata Cloud@Customer や Oracle Autonomous Database Cloud Service などのターゲットに使用します。通常、ロックダウン・プロファイルは事前定義されていません

例

```
./premigration.sh --targetcloud atps --outfileprefix ATPS_RUN_01 --outdir
/path/CPAT_output --reportformat TEXT JSON ...
```

親トピック: 移行前アドバイザ・ツールのコマンドライン・プロパティ

20.9.2.19 username

移行前アドバイザ・ツールのプロパティusernameでは、ソース・データベースへの接続時に使用するユーザー名を指定します。

プロパティ	説明
プロパティ・タイプ	文字列
構文	`-u|--username user-name`

説明

`--username`スイッチは、ソース・データベースに接続するユーザーにCPATを提供します。

使用上のノート

指定するユーザー名は`SELECT ANY DICTIONARY`権限を持ち、`SYSTEM.DUM$COLUMNS`および`SYSTEM.DUM$DATABASE`で`SELECT`が付与されている必要があります。ターゲット・データベースに接続する場合は、`ADMIN`ユーザーまたは`PDB_DBA`ロールを持つ別のユーザーを使用します。

例

```
premigration --connectstring jdbc:oracle:thin:@example.oracle.com:1521/ORCLPDB1 --
username ADMIN -t atps
```

親トピック: 移行前アドバイザ・ツールのコマンドライン・プロパティ

20.9.2.20 version

移行前アドバイザ・ツールのプロパティversionでは、CPATの現在のバージョンが出力され、終了されます。

プロパティ	説明
プロパティ・タイプ	文字列
構文	`-v|--version`

説明

移行前アドバイザ・ツールのversionプロパティでは、移行前アドバイザ・ツールのバージョン番号およびリリースされた日付を出力できます。

使用上のノート

このオプションを使用して、実行中のアップグレード前アドバイザ・ツールのバージョンについて情報を取得します。

例

```
premigration.sh -v
Premigration Advisor Application Version: 22.10.0 (production)
Build date: 2022/10/18 10:55:43
Build hash: 53950fd

premigration.com --version
Premigration Advisor Application Version: 22.10.0 (production)
Build date: 2022/10/18 10:55:43
Build hash: 53950fd
```

親トピック: 移行前アドバイザ・ツールのコマンドライン・プロパティ

20.9.2.21 updatecheck

移行前アドバイザ・ツールのプロパティupdatecheckでは、CPATの現在のバージョンが出力され、それより最新のバージョンがないかが確認されてから、終了されます。

プロパティ	説明
プロパティ・タイプ	文字列
構文	`-U | --updatecheck`
デフォルト値	なし

説明

クラウド移行前アドバイザ・ツール(CPAT)の更新されたバージョンが使用可能かどうかを確認します。新しいバージョンの場合は、yesが出力されます。新しいバージョンがない場合はnoが出力されます。チェックを完了すると、CPATが終了します。正常にチェックを実行するにはネットワーク・アクセスが必要です。

移行前アドバイザ・ツールのupdatecheckプロパティでは、Oracle Supportをチェックして、クラウド移行前アドバイザ・ツール(CPAT)の更新されたバージョンが使用可能かどうかを確認します。

使用上のノート

このプロパティを使用するには、ネットワーク接続が必要です。ネットワーク接続がない場合は、エラーCPAT-4001: Error checking for latest available version of the Cloud Premigration Advisor Toolが表示されます。ネットワークがファイアウォールの内側にある場合は、適切なHTTPSプロキシを定義してこのスイッチを使用する必要があります。

例

```
export _JAVA_OPTIONS='-Dhttps.proxyHost=www-proxy.us.oracle.com -Dhttps.proxyPort=80'
./premigration.sh --updatecheck
```

CPATの最新バージョンがすでにある場合は、次の出力が表示されます:

```
Picked up _JAVA_OPTIONS: -Dhttps.proxyHost=www-proxy.us.oracle.com -
Dhttps.proxyPort=80There is no newer version available of the Cloud Premigration
Advisor Tool
```

親トピック: 移行前アドバイザ・ツールのコマンドライン・プロパティ

20.10 移行前アドバイザ・ツールのログ・ファイル構造

移行前アドバイザ・ツールでは、ジョブ・ステータスや構成ファイルを含む、ログ・ファイル構造が生成されます。

移行前ログ・ファイルのベース・パス

移行前アドバイザ・ツールのログ・ファイル・パスは、グローバル・パラメータpremigration_log_dirを使用して設定します。

/cfgtoollogs/premigrate/auto

自動構成ツールのログ・ディレクトリ(/cfgtoollogs/premigration/auto)には、移行前ジョブ・マネージャで処理される各ジョブについて特定の情報を提供する次の3つのトレース・ログ・ファイルが含まれています。

- premigration.log: ジョブの実行中に発生した問題を識別する、ジョブの詳細ログを提供します。
- premigration_usr.log: 読みやすいように書式設定されたジョブ情報。
- premigration_err.log: ジョブの実行時に発生した想定されていない例外のレポート。

ジョブの開始または停止時に問題が発生した場合、これらのログ・ファイルの情報を使用して、問題の原因を特定できます。

/config_files

config_filesディレクトリには、移行前アドバイザ・ツールの内部ランタイム構成およびグローバル一時ファイルが含まれています。

/status

/statusディレクトリには、JSONジョブ・ステータス・ファイルが含まれています。これには2つのディレクトリがあります。

- status.json: このディレクトリには、JSONファイル形式で完了したすべてのジョブの最終ジョブ・ステータスが含まれています。
- progress.json: このディレクトリには、JSONファイル形式で現在実行中のすべてのジョブの進行状況が含まれています。

ディレクトリ内の各モジュールには、実行された操作のステータス・ファイルが含まれています。モジュールの形式は次のようになります。接頭辞dbnameはデータベース名、operationは実行されたアップグレード操作、接尾辞statusはその操作の完了ステータスです。

```
dbname_operation-name.status
```

その操作の成功または失敗は、接尾辞によって示されます。その操作の実行が成功したことを示す.successか、その操作が失敗したことを示す.failureのいずれかです。たとえば、次のモジュール名は、データベースsalesでの事前チェック操作が正常に実行されたことを示します。

```
sales_prechecks.success
```

操作モジュール名には、次のいずれかを指定できます。

- preupgrade: カスタム・スクリプトを実行できるアップグレード前のステージ。
- prechecks: アップグレードの開始前に完了したアップグレード・チェック。
- grp: 保証付きリストア・ポイント(Oracle Flashback Technologyを使用)
- prefixups: アップグレードを開始する前にアップグレード前の修正を実行します
- drain: アップグレードを開始する前に既存のジョブが完了または移行されるステージ。
- dbupgrade: アップグレードが実行されるステージ。

- postchecks: アップグレードの完了後にアップグレード後のチェックが実行されるステージ。
- postfixups: アップグレード後の修正が実行されるステージ。
- postupgrade: カスタムのアップグレード後スクリプトを実行できるステージ。

個々のジョブおよびデータベース・ログ・ファイル・ディレクトリ

移行前ディスパッチャによって起動される各ジョブには、そのジョブ識別子の接頭辞が付いたディレクトリが与えられます。ジョブ・ディレクトリの内部では、ジョブ内の各データベースにパス/database/logs/sidにログ・ディレクトリが提供されます。ここで、sidはデータベースのシステム識別子です。たとえば、構成ファイルで識別されたジョブがsales1で、データベース・システム識別子がsalesの場合、データベースsalesのログ・ファイルのパスは次のようになります。

```
sales.log_dir=/database/logs/sales1
```

ログ・ディレクトリには、そのデータベースに対して実行されたすべてのタスクに関するすべての関連ログ・ファイルが含まれています。デフォルトでは、SIDで識別されるディレクトリが/database/logsディレクトリの下に作成されます。この方法で構成ファイルを設定することを選択した場合、各データベース・ジョブに個別のログ・ディレクトリを設定できます。

/#### (ジョブ番号)

個々のジョブ実行は、/####という形式の実行番号で識別されるサブディレクトリに配置されます。ここで、####は実行番号を表します。たとえば、0004となります。ジョブ実行番号ディレクトリには、次のログ・ファイルが含まれます。

- premigration_err.log: ジョブの実行中に発生した想定されていない例外をレポートします。
- premigrationYYYYMMDD.log: 移行前トレース・ログ・ファイル。ジョブの実行中に発生した問題を識別するジョブの詳細なログを提供します。変数YYYYMMDDは、ジョブの年、月および日を表します。
- premigration_YYYYMMDD_user.log: 移行前ジョブのステータス・ファイルであり、読みやすいように書式設定されています。変数YYYYYYMMDDは、ジョブの年、月および日を表します。

/premigration

移行前ディレクトリ(/premigration)には、次のファイルとログ・ファイルが含まれています。

- prechecks_databasename.log: トレース・ログ・ファイル。このファイルは、アップグレード前のジョブ・ステージ中に発生した問題の識別に役立つ詳細なログを提供します。変数databasenameは、チェックされたデータベースの名前です。
- databasename_preupgrade.html: データベースのステータスに関するHTMLレポート。変数databasenameは、チェックされたデータベースの名前です。
- databasename_preupgrade.html: データベースのステータスに関するテキスト・レポート。変数databasenameは、チェックされたデータベースの名前です。

/premigration

データベース移行前ディレクトリ(/premigration)には、データベース・アップグレードに関連付けられているすべてのログ・ファイルが含まれています。

- premigrationYYYYMMDDHHMISCdbname.log: 並列性を示す、アップグレードが実行された日付やデータベース名によって識別される、ソース・データベースのログ・ファイル。書式:
  - YYYY: 年
  - MM: 月
  - DD: 日
  - HH: 時
  - MI: 分

    - SC: 秒
    - dbname: データベース名。dbnameはデータベース名です。
- premigrationYYYYMMDDHHMISCdbnameN.log: 移行前チェックが実行された日付で識別される、ソース・データベースのログ・ファイル。

  書式:

    - YYYY: 年
    - MM: 月
    - DD: 日
    - HH: 時
    - MI: 分
    - SC: 秒
    - dbnameN: データベース名。dbnameはデータベース名で、Nは並列度を示します。CDB ROOTおよび非CDBデータベースの場合は0...3で、PDBの場合は0...1です。

/temp

移行前一時ファイル(/temp)。このディレクトリには、アップグレード時に使用されるPFILEなどのファイルを含めることができます。

親トピック: クラウド移行前アドバイザ・ツール

20.11 移行前アドバイザ・ツールによって実行されるチェックのリスト

移行前アドバイザ・ツールのレポートに存在するチェックについて情報をご確認ください。

> ノート:
>
> ソース・データベースと移行ターゲットを指定すると、移行前アドバイザ・ツールによって、その移行シナリオに必要なチェックが実行されます。そのシナリオに必要なチェックのみが実行されます。レポートでは、CPAT の起動時に指定した移行シナリオに対する応答が示されます。

- dp_has_low_streams_pool_size
  移行前アドバイザ・ツールのチェックdp_has_low_streams_pool_sizeでは、STREAMS_POOL_SIZEが、データ・ポンプ移行を開始して効率的に機能するのに十分な大きさであることが確認されます。
- gg_enabled_replication
  移行前アドバイザ・ツールのチェックgg_enabled_replicationでは、初期化パラメータENABLE_GOLDENGATE_REPLICATIONがソース・データベースで設定されていないことが通知されます。
- gg_force_logging
  移行前アドバイザ・ツールのチェックgg_force_loggingでは、移行中のすべてのトランザクションおよびロードの強制ロギングが設定されていないことが示されます。
- gg_has_low_streams_pool_size
  移行前アドバイザ・ツールのチェックgg_has_low_streams_pool_sizeでは、STREAMS_POOL_SIZEがOracle GoldenGateに十分な大きさであることが確認されます。
- gg_not_unique_bad_col_no
  移行前アドバイザ・ツールのチェックgg_not_unique_bad_col_noでは、主キーがなく、NULL値が許可されない一意索引もない表が検索されます。
- gg_not_unique_bad_col_yes
  移行前アドバイザ・ツールのチェックgg_not_unique_bad_col_yesでは、主キー、一意索引またはキー列がない表(バインドされていないデータ型で定義された表の列を含む)が検索されます。
- gg_objects_not_supported
  移行前アドバイザ・ツールのチェックgg_objects_not_supportedでは、ソース・データベース上にサポートされていないオブジェクトがあることが示されます。
- gg_supplemental_log_data_min
  移行前アドバイザ・ツールのチェックgg_supplemental_log_data_minでは、ソース・データベースで最小サプリメンタル・ロギングが有効になっていないことが示されます。
- gg_tables_not_supported
  移行前アドバイザ・ツールのチェックgg_tables_not_supported_adbでは、データベース内の一部のオブジェクトをOracle GoldenGateを使用してレプリケートできないことが示されます。
- gg_tables_not_supported
  移行前アドバイザ・ツールのチェックgg_tables_not_supportedでは、ADB以外のデータベース内の一部のオブジェクトをOracle GoldenGateを使用してレプリケートできないことが示されます。
- gg_user_objects_in_ggadmin_schemas

移行前アドバイザ・ツールのチェックgg_user_objects_in_ggadmin_schemasでは、Oracle GoldenGate管理者権限があるスキーマにユーザー・オブジェクトが存在することが示されます。

- has_absent_default_tablespace
  移行前アドバイザ・ツールのチェックhas_absent_default_tablespaceでは、スキーマ所有者のデフォルト表領域が不足していることが示されます。
- has_absent_temp_tablespace
  移行前アドバイザ・ツールのチェックhas_absent_temp_tablespaceでは、スキーマ所有者の一時表領域が不足していることが示されます。
- has_active_data_guard_dedicated
  移行前アドバイザ・ツールのチェックhas_active_data_guard_dedicatedでは、ソース・インスタンスでActive Data Guardが使用されているかどうかが検出されます。
- has_active_data_guard_serverless
  移行前アドバイザ・ツールのチェックhas_active_data_guard_serverlessでは、ソース・インスタンスでActive Data Guardが使用されているかどうかが検出されます。
- has_basic_file_lobs
  移行前アドバイザ・ツールのチェックhas_basic_file_lobsでは、BASICFILE LOB (Oracle Autonomous Databaseでサポートされていない)がスキーマに存在することが示されます。
- has_clustered_tables
  移行前アドバイザ・ツールのチェックhas_clustered_tablesでは、表クラスタ(Oracle Autonomous Databaseでサポートされていない)がスキーマに存在することが示されます。
- has_columns_of_rowid_type
  移行前アドバイザ・ツールのチェックhas_columns_of_rowid_typeでは、移行できない、ROWIDデータ型の列がある表が示されます。
- has_columns_with_media_data_types_adb
  移行前アドバイザ・ツールのチェックhas_columns_with_media_data_types_adbでは、移行できない、マルチメディア・データ型がある表が示されます。
- has_columns_with_media_data_types_default
  移行前アドバイザ・ツールのチェックhas_columns_with_media_data_types_defaultでは、マルチメディア列がある表が示されます。
- has_columns_with_spatial_data_types
  移行前アドバイザ・ツールのチェックhas_columns_with_spatial_data_typesでは、十分サポートされていない空間オブジェクトがあることが示されます。
- has_common_objects
  移行前アドバイザ・ツールのチェックhas_common_objectsでは、データベース・インスタンスに共通オブジェクトがあることが示されます。
- has_compression_disabled_for_objects
  移行前アドバイザ・ツールのチェックhas_compression_disabled_for_objectsでは、COMPRESSION句がない表またはパーティションがあることが示されます。
- has_csmig_schema
  移行前アドバイザ・ツールのチェックhas_csmig_schemaでは、CSSCANユーティリティがソース・データベースにインストールおよび構成されていることが示されます。
- has_data_in_other_tablespaces_dedicated
  移行前アドバイザ・ツールのチェックhas_data_in_other_tablespaces_dedicatedでは、専用インフラス

トラクチャ上のOracle Autonomous Databaseに移行する際の表領域制限の対象となるデータが特定されます。

- has_data_in_other_tablespaces_serverless
  移行前アドバイザ・ツールのチェックhas_data_in_other_tablespaces_serverlessでは、共有インフラストラクチャ上のOracle Autonomous Databaseに移行する際の表領域制限の対象となるデータが特定されます。
- has_db_link_synonyms
  移行前アドバイザ・ツールのチェックhas_db_link_synonymsでは、データベース・リンクを含むシノニムがスキーマに含まれていることが示されます。
- has_db_links
  移行前アドバイザ・ツールのチェックhas_db_linksでは、データベース・リンクを含むシノニムがスキーマに含まれていることが示されます。
- has_dbms_credentials
  移行前アドバイザ・ツールのチェックhas_dbms_credentialsでは、DBMS_CLOUD.CREATE_CREDENTIALで作成されていない資格証明がスキーマに含まれていることが示されます。
- has_dbms_credentials
  移行前アドバイザ・ツールのチェックhas_dbms_credentialsでは、DBMS_CLOUD.CREATE_CREDENTIALで作成されていない資格証明がスキーマに含まれていることが示されます。
- has_directories
  移行前アドバイザ・ツールのチェックhas_directoriesでは、ソース・データベースにディレクトリ・オブジェクトがあることが示されます。
- has_enabled_scheduler_jobs
  移行前アドバイザ・ツールのチェックhas_enabled_scheduler_jobs では、Oracle Data Pumpエクスポートを妨げる可能性があるリスト・スケジューラ・ジョブがあることが示されます。
- has_external_tables_dedicated
  移行前アドバイザ・ツールのチェックhas_external_tables_dedicatedでは、ソース・データベースに非クラウド・オブジェクト・ストレージの外部表が存在することが示されます。
- has_external_tables_default
  移行前アドバイザ・ツールのチェックhas_external_tables_defaultでは、外部表が依存するDIRECTORYオブジェクトが作成されていないかぎり、外部表を移行できないことが示されます。
- has_external_tables_serverless
  移行前アドバイザ・ツールのチェックhas_external_tables_serverlessでは、ソース・データベースに非クラウド・オブジェクト・ストレージの外部表が存在することが示されます。
- has_fmw_registry_in_system
  移行前アドバイザ・ツールのチェックhas_fmw_registry_in_systemでは、移行前にSYSTEMスキーマからFusion Middlewareスキーマ・バージョン・レジストリを移動する必要があることが示されます。
- has_illegal_characters_in_comments
  移行前アドバイザ・ツールのチェックhas_illegal_characters_in_commentsでは、データベース文字セットで有効でない文字が表のコメント内に存在することが示されます。
- has_ilm_ado_policies
  移行前アドバイザ・ツールのチェックhas_ilm_ado_policiesでは、情報ライフサイクル管理(ILM)自動データ最適化(ADO)ポリシーが移行されないことが示されます。
- has_incompatible_jobs
  移行前アドバイザ・ツールのチェックhas_incompatible_jobsでは、情報ライフサイクル管理(ILM)自動データ最適化(ADO)ポリシーが移行されないことが示されます。

- has_index_organized_tables
  移行前アドバイザ・ツールのチェックhas_index_organized_tablesでは、ソース・データベースに索引構成表が存在することが示されます。
- has_java_objects
  移行前アドバイザ・ツールのチェックhas_java_objectsでは、ソース・データベースにJavaオブジェクトが存在することが示されます。
- has_java_source
  移行前アドバイザ・ツールのチェックhas_java_sourceでは、ソース・データベースにJavaソースが存在することが示されます。
- has_libraries
  移行前アドバイザ・ツールのチェックhas_librariesでは、CREATE LIBRARY文を必要とするアプリケーションが存在することが示されます。
- has_logging_off_for_partitions
  移行前アドバイザ・ツールのチェックhas_logging_off_for_partitionsでは、NOLOGGINGストレージ属性を使用したパーティションがあることが示されます。
- has_logging_off_for_subpartitions
  移行前アドバイザ・ツールのチェックhas_logging_off_for_subpartitionsでは、NOLOGGINGストレージ属性を使用したパーティションがあることが示されます。
- has_logging_off_for_tables
  移行前アドバイザ・ツールのチェックhas_logging_off_for_tablesでは、NOLOGGINGストレージ属性を使用する表があることが示されます。
- has_low_streams_pool_size
  移行前アドバイザ・ツールのチェックhas_low_streams_pool_sizeでは、予期しない、あるいは正しくない属性のマイニング・モデルが検出されたことが示されます。
- has_noexport_object_grants
  移行前アドバイザ・ツールのチェックhas_noexport_object_grantsでは、Oracle Data Pumpがすべてのオブジェクト権限をエクスポートできないことが示されます。
- has_parallel_indexes_enabled
  移行前アドバイザ・ツールのチェックhas_parallel_indexes_enabledでは、PARALLEL句索引が存在することが示されます。
- has_profile_not_default
  移行前アドバイザ・ツールのチェックhas_profile_not_defaultでは、PROFILEを使用できないスキーマがターゲット・システムに存在することが示されます。
- has_public_synonyms
  移行前アドバイザ・ツールのチェックhas_public_synonymsでは、パブリック・シノニムがソース・システム・スキーマに存在することが示されます。
- has_refs_to_restricted_packages_dedicated
  移行前アドバイザ・ツールのチェックhas_refs_to_restricted_packages_dedicatedでは、部分的または完全にサポートされていないパッケージへの参照があることが示されます。
- has_refs_to_restricted_packages_serverless
  移行前アドバイザ・ツールのチェックhas_refs_to_restricted_packages_serverlessでは、部分的または完全にサポートされていないパッケージへの参照があることが示されます。
- has_refs_to_user_objects_in_sys

移行前アドバイザ・ツールのチェックhas_refs_to_user_objects_in_sysでは、SYSまたはSYSTEMに依存するユーザー・スキーマ・オブジェクトがあることが示されます。

- has_role_privileges
  移行前アドバイザ・ツールのチェックhas_role_privilegesでは、ソース・データベースで使用される一部のロール権限がターゲット・データベースでは提供されないことが示されます
- has_sqlt_objects_adb
  移行前アドバイザ・ツールのチェックhas_sqlt_objects_adbでは、SQLTXPLAINオブジェクトが検出されることが示されます。
- has_sqlt_objects_default
  移行前アドバイザ・ツールのチェックhas_sqlt_objects_defaultでは、Oracle Data PumpがエクスポートしないSQLTXPLAINオブジェクトが検出されることが示されます。
- has_sys_privileges
  移行前アドバイザ・ツールのチェックhas_sys_privilegesでは、ソース・データベースの一部のシステム権限がターゲット・データベースでは提供されないことが示されます。
- has_tables_that_fail_with_dblink
  移行前アドバイザ・ツールのチェックhas_tables_that_fail_with_dblinkでは、LONGまたはLONG RAWデータ型の表があることが示されます
- has_tables_with_long_raw_datatype
  移行前アドバイザ・ツールのチェックhas_tables_with_long_raw_datatypeでは、LONGまたはLONG RAWデータ型の表があることが示されます
- has_tables_with_xmltype_column
  移行前アドバイザ・ツールのチェックhas_tables_with_xmltype_columnでは、XMLTYPE列を含む表があることが示されます。
- has_trusted_server_entries
  移行前アドバイザ・ツールのチェックhas_trusted_server_entriesでは、再作成できないTRUSTED_SERVERエントリがOracle Autonomous Databaseに存在することが示されます。
- has_user_defined_objects_in_sys
  移行前アドバイザ・ツールのチェックhas_user_defined_objects_in_sysでは、SYSまたはSYSTEMスキーマのユーザー定義オブジェクトがあることが示されます。
- has_users_with_10g_password_version
  移行前アドバイザ・ツールのチェックhas_users_with_10g_password_versionでは、10Gパスワード・バージョンを使用するユーザー・アカウントがあることが示されます。
- has_sys_privileges
  移行前アドバイザ・ツールのチェックhas_sys_privilegesでは、ソース・データベースの一部のシステム権限がターゲット・データベースでは提供されないことが示されます。
- has_tables_that_fail_with_dblink
  移行前アドバイザ・ツールのチェックhas_tables_that_fail_with_dblinkでは、LONGまたはLONG RAWデータ型の表があることが示されます
- has_tables_with_long_raw_datatype
  移行前アドバイザ・ツールのチェックhas_tables_with_long_raw_datatypeでは、LONGまたはLONG RAWデータ型の表があることが示されます
- has_tables_with_xmltype_column
  移行前アドバイザ・ツールのチェックhas_tables_with_xmltype_columnでは、XMLTYPE列を含む表がある

ことが示されます。

- has_trusted_server_entries
  移行前アドバイザ・ツールのチェックhas_trusted_server_entriesでは、再作成できないTRUSTED_SERVERエントリがOracle Autonomous Databaseに存在することが示されます。
- has_user_defined_objects_in_sys
  移行前アドバイザ・ツールのチェックhas_user_defined_objects_in_sysでは、SYSまたはSYSTEMスキーマのユーザー定義オブジェクトがあることが示されます。
- has_users_with_10g_password_version
  移行前アドバイザ・ツールのチェックhas_users_with_10g_password_versionでは、10Gパスワード・バージョンを使用するユーザー・アカウントがあることが示されます。
- has_xmlschema_objects
  移行前アドバイザ・ツールのチェックhas_xmlschema_objectsでは、ソース・データベースにXMLスキーマ・オブジェクトが存在することが示されます。
- has_xmltype_tables
  移行前アドバイザ・ツールのチェックhas_xmltype_tablesでは、ソース・データベースにXMLType表が存在することが示されます。
- modified_db_parameters_dedicated
  移行前アドバイザ・ツールのチェックmodified_db_parameters_dedicatedでは、制限付きの初期化パラメータが変更されることが示されます。
- modified_db_parameters_serverless
  移行前アドバイザ・ツールのチェックmodified_db_parameters_serverlessでは、制限付きの初期化パラメータが変更されることが示されます。
- nls_character_set_conversion
  移行前アドバイザ・ツールのチェックnls_character_set_conversionでは、ソース・データベースにOracle Autonomous Databaseでは無効な文字コードが存在することが示されます。
- nls_national_character_set
  移行前アドバイザ・ツールのチェックnls_national_character_setでは、NCHARとNVARCHAR2の長さがソース・データベースとターゲット・データベースで異なることが示されます。
- nls_nchar_ora_910
  移行前アドバイザ・ツールのチェックnls_nchar_ora_910では、NCHARとNVARCHAR2の長さがターゲット・データベースでの最大長を超えていることが示されます。
- options_in_use_not_available_dedicated
  移行前アドバイザ・ツールのチェックoptions_in_use_not_available_dedicatedでは、ソース・データベースに使用できないデータベース・オプションが存在することが示されます。
- options_in_use_not_available_serverless
  移行前アドバイザ・ツールのチェックoptions_in_use_not_available_serverlessでは、ソース・データベースに使用できないデータベース・オプションが存在することが示されます。
- standard_traditional_audit_adb
  移行前アドバイザ・ツールのチェックstandard_traditional_audit_adbでは、従来の監査構成がデータベースで検出されたことが示されます。
- standard_traditional_audit_default
  移行前アドバイザ・ツールのチェックstandard_traditional_audit_defaultでは、従来の監査構成がデータベースで検出されたことが示されます。

- timezone_table_compatibility_higher_dedicated
  移行前アドバイザ・ツールのチェックtimezone_table_compatibility_higher_dedicatedでは、ソースのタイムゾーン設定がターゲット・データベースより最新のバージョンであることが示されます。
- timezone_table_compatibility_higher_default
  移行前アドバイザ・ツールのチェックtimezone_table_compatibility_higher_defaultでは、ソースのタイムゾーン設定がターゲット・データベースより最新のバージョンであることが示されます。
- timezone_table_compatibility_higher_serverless
  移行前アドバイザ・ツールのチェックtimezone_table_compatibility_higher_serverlessでは、ソースのタイムゾーン設定がターゲット・データベースより最新のバージョンであることが示されます。
- unified_and_standard_traditional_audit_adb
  移行前アドバイザ・ツールのチェックunified_and_standard_traditional_audit_adbでは、従来の監査構成がデータベースで検出されたことが示されます。
- unified_and_standard_traditional_audit_default
  移行前アドバイザ・ツールのチェックunified_and_standard_traditional_audit_defaultでは、従来の監査構成がデータベースで検出されたことが示されます。
- xdb_resource_view_has_entriesチェック
  移行前アドバイザ・ツールのチェックxdb_resource_view_has_entriesチェックでは、Oracle Autonomous DatabaseでサポートされていないXDBリポジトリがあることが示されます。RESOURCE_VIEW内のエントリは移行されません。

親トピック: クラウド移行前アドバイザ・ツール

20.11.1 dp_has_low_streams_pool_size

移行前アドバイザ・ツールのチェックdp_has_low_streams_pool_sizeでは、STREAMS_POOL_SIZEが、データ・ポンプ移行を開始して効率的に機能するのに十分な大きさであることが確認されます。

結果の重要度

ランタイム

修正あり

あり

有効範囲

UNIVERSAL

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)
- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

説明

移行前アドバイザ・ツールのチェックdp_has_low_streams_pool_sizeでは、STREAMS_POOL_SIZEが、Oracle Data Pumpの移行を開始して効率的に機能するのに十分な大きさで割り当てられていることが確認されます。

影響

データベース初期化パラメータSTREAMS_POOL_SIZE値は、Streamsプールのサイズの決定に役立ちます。エクスポート用に十分なメモリーをSTREAMS_POOL_SIZEに割り当てる必要があります。そうしないと、Oracle Data Pumpのエクスポート・パフォーマンスが低下したり、エクスポートが失敗する可能性があります。エクスポート前にソース・データベースでSTREAMS_POOL_SIZEの最小値を定義することをお薦めします。

処置

SQLを実行してSTREAMS_POOL_SIZEを設定し、エクスポート用のメモリーを割り当てます。次に例を示します。

```
ALTER SYSTEM SET streams_pool_size=256M SCOPE=BOTH
```

メモリーを割り当てた後、必要であればインスタンスを再起動します。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.2 gg_enabled_replication

移行前アドバイザ・ツールのチェックgg_enabled_replicationでは、初期化パラメータENABLE_GOLDENGATE_REPLICATIONがソース・データベースで設定されていないことが通知されます。

結果の重要度

処置必須

修正あり

あり

有効範囲

UNIVERSAL

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)
- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

説明

移行前アドバイザ・ツールのgg_enabled_replicationチェックでは、移行方法としてOracle GoldenGateを選択したが初期化パラメータENABLE_GOLDENGATE_REPLICATIONがTRUEに設定されていないことが示されます。

影響

Oracle GoldenGateでデータ移行を実行する場合は、ソース・データベースの初期化パラメータENABLE_GOLDENGATE_REPLICATIONをTRUEに設定する必要があります。そのように設定されていない場合は、移行に失敗します。

処置

データベース初期化ファイル内でENABLE_GOLDENGATE_REPLICATIONをTRUEに設定します。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.3 gg_force_logging

移行前アドバイザ・ツールのチェックgg_force_loggingでは、移行中のすべてのトランザクションおよびロードの強制ロギングが設定されていないことが示されます。

結果の重要度

確認必須

修正あり

あり

ターゲット・クラウド

これはデフォルトのチェックです。これは次に適用されます:

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)
- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

有効範囲

UNIVERSAL

説明

ソース・データベースで強制ロギング・モードが設定されていません。強制ロギング・モードが設定されている場合は、すべてのトランザクションおよびロードのロギングが強制適用されて、これらのトランザクションおよびロードをログに記録しないというユーザー設定またはストレージ設定が上書きされます。

影響

強制ロギングが設定されていない場合は、移行中にOracle GoldenGate Extract構成内のソース・データが欠落する可能性があります。

処置

表領域およびデータベース・レベルで強制ロギングを有効にするには、SYSDBAとしてログインし、強制ロギングを有効にします。次に例を示します。

```
SQL> alter database force logging;
Database altered.
```

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.4 gg_has_low_streams_pool_size

移行前アドバイザ・ツールのチェックgg_has_low_streams_pool_sizeでは、STREAMS_POOL_SIZEがOracle GoldenGateに十分な大きさであることが確認されます。

結果の重要度

ランタイム

修正あり

あり

有効範囲

UNIVERSAL

ターゲット・クラウド

これはデフォルトのチェックです。これは次に適用されます:

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)
- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

説明

移行前アドバイザ・ツールのチェックgg_has_low_streams_pool_sizeでは、STREAMS_POOL_SIZEがOracle GoldenGateの移行が効率的に開始および動作するのに十分な大きさに事前に割り当てられていることが確認されます。

Oracle GoldenGate Extractはソース・データベース内の基になるログマイニング・サーバーと対話し、Replicatはターゲット・データベース内のインバウンド・サーバーと対話します。

サーバーが使用する共有メモリーは、データベースのシステム・グローバル領域(SGA)のStreamsプールの部分から取得されます。そのため、統合モードで実行が予想される数のExtractおよびReplicatプロセスが常に十分なメモリーを使用できるようにするために、データベース初期化パラメータSTREAMS_POOL_SIZEを十分に高い値に設定する必要があります。Streamsプールは、データベースの他のコンポーネント(Oracle Streams、Advanced Queuing、Oracle Data Pumpのエクスポート/インポートなど)でも使用されるため、Oracle GoldenGateのStreamsプールのサイズを設定する際には、他のコンポーネントも考慮してください。

デフォルトでは、1つのExtractは、ログマイニング・サーバーに1GBで実行するように要求します。ベスト・プラクティスとしては、Streamsプールの25パーセントが使用可能であるようにします。したがって、1つプロセスについて、最小のSTREAMS_POOL_SIZEは1.25 GBになります。詳細は、Oracle SupportのドキュメントID 2078459.1およびOracle GoldenGateのドキュメントを参照してください。

影響

Oracle GoldenGateプロセス用に十分なメモリーをSTREAMS_POOL_SIZEに割り当ててください。そうしないと、Oracle GoldenGateのパフォーマンスが低下したり、ExtractまたはReplicatが失敗する可能性があります。Oracle GoldenGateを実行する前に、ソース・データベースでSTREAMS_POOL_SIZEの最小値を定義することをお薦めします。

処置

SQLを実行してSTREAMS_POOL_SIZEを設定し、実行されるOracle GoldenGateプロセスの数に応じてExtractおよびReplicatのメモリーを割り当てます。次に例を示します。

```
ALTER SYSTEM SET streams_pool_size=1250M SCOPE=BOTH;
```

メモリーを割り当てた後、必要であればインスタンスを再起動します。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.5 gg_not_unique_bad_col_no

移行前アドバイザ・ツールのチェックgg_not_unique_bad_col_noでは、主キーがなく、NULL値が許可されない一意索引もない表が検索されます。

結果の重要度

確認必須

修正あり

いいえ

有効範囲

SCHEMA

ターゲット・クラウド

これはデフォルトのチェックです。これは次に適用されます:

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)
- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

移行方法

GOLDENGATE

説明

移行前アドバイザ・ツールのチェックgg_not_unique_bad_col_noでは、主キーがなく、NULL値が許可されない一意索引もない表が検索されます。

このチェックで特定される表の変更が大量であれば、GoldenGateのレプリケーションが遅れて、追いつけなくなる可能性があります。すべてのINSERT、UPDATEまたはDELETE操作をレプリケートするには、全表スキャンが必要になります。

影響

Oracle GoldenGateでこれらの表の大幅な変更を処理する必要がある場合、レプリケーションを継続すると徐々に遅れて、回復できない可能性があります。

処置

この問題に対処するには、次のいずれかを実行します:

- リストされた表に主キーを追加する
- 移行中は可能なかぎりデータベースを休止する
- Oracle Data Pumpなどの別の方法を使用して表に対する変更を移行する

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.6 gg_not_unique_bad_col_yes

移行前アドバイザ・ツールのチェックgg_not_unique_bad_col_yesでは、主キー、一意索引またはキー列がない表(バインドされていないデータ型で定義された表の列を含む)が検索されます。

結果の重要度

処置必須

修正あり

いいえ

有効範囲

SCHEMA

ターゲット・クラウド

これはデフォルトのチェックです。これは次に適用されます:

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)
- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

説明

移行前アドバイザ・ツールのチェックgg_not_unique_bad_col_yesでは、主キー、一意索引またはキー列がない表が検索されます。**問題のある列**により、述語(where句)で有用ではない列がその表にあることが示されます。その表列は、LONGまたはBLOBなど、制限のないデータ型を使用して定義されています。

影響

一意性がなく、バインドされていないdata_typesが設定された表がある場合は、表レコードを一意に識別できないため、論理レプリケーションには使用できません。これらの表は、Oracle DatabaseのGoldenGateガイドではサポートされていないため、Oracle GoldenGateでは移行できません。

処置

この問題に対処するために、可能な場合は表に主キーまたは一意キーを追加してください。主キーまたは一意キーを追加できない場合は、Oracle Data Pumpなどの他の方法で表を移行する必要があります。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.7 gg_objects_not_supported

移行前アドバイザ・ツールのチェックgg_objects_not_supportedでは、ソース・データベース上にサポートされていないオブジェクトがあることが示されます。

結果の重要度

処置必須

修正あり

いいえ

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)

- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)
- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

有効範囲

SCHEMA

説明

このチェックは、Oracle GoldenGateでの移行の場合にスキーマに適用されます。ソース・データベースに、Oracle GoldenGateでの移行でサポートされていないオブジェクトが存在します。

影響

通常、このチェックでリストされたオブジェクトは、追加構成なしでは移行において正常にレプリケートされません。

処置

リストされたSUPPORT_MODE値があるオブジェクトを正常にレプリケートする方法を確認するには、Oracle GoldenGateのドキュメントを参照してください。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.8 gg_supplemental_log_data_min

移行前アドバイザ・ツールのチェックgg_supplemental_log_data_minでは、ソース・データベースで最小サプリメンタル・ロギングが有効になっていないことが示されます。

結果の重要度

処置必須

修正あり

あり

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)
- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

有効範囲

UNIVERSAL

説明

このチェックは、Oracle GoldenGateでの移行の場合にスキーマに適用されます。Oracle GoldenGateを使用する場合は、Oracleソース・データベースに対する最小サプリメンタル・ロギング(データベースレベルのオプション)が必要になります。この構成では、行チェーン情報(存在する場合)が更新操作用のREDOログに追加されます。

影響

最小サプリメンタル・ログ・データが有効になっていない場合、Oracle GoldenGateは機能しません。

処置

SYSDBAとしてログインし、ソース・データベースで最小サプリメンタル・ロギングを有効にします。次に例を示します。

```
SQL> ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;
```

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.9 gg_tables_not_supported

移行前アドバイザ・ツールのチェックgg_tables_not_supported_adbでは、データベース内の一部のオブジェクトをOracle GoldenGateを使用してレプリケートできないことが示されます。

結果の重要度

処置必須

修正あり

いいえ

ターゲット・クラウド

これはデフォルトのチェックです。これは次に適用されます:

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

有効範囲

SCHEMA

説明

このチェックは、Oracle GoldenGateでの移行の場合にスキーマに適用されます。ソース・データベース内のオブジェクトをOracle GoldenGateでレプリケートできない場合、レポートでは、これらのオブジェクトのリストが、このチェック・メッセージとともに示されます。

影響

リストされたオブジェクトは、Oracle GoldenGateで移行されません。

処置

スイッチオーバー時に、Oracle Data Pumpなどの別の移行方法を使用して、リストされた関連オブジェクトをターゲット・データベースに移動する必要があります。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.10 gg_tables_not_supported

移行前アドバイザ・ツールのチェックgg_tables_not_supportedでは、ADB以外のデータベース内の一部のオブジェクトはOracle GoldenGateを使用してレプリケートできないことが示されます。

結果の重要度

処置必須

修正あり

いいえ

ターゲット・クラウド

- デフォルト(Oracle Autonomous Database (ADB)以外のOracle Databaseインスタンス)

有効範囲

SCHEMA

説明

このチェックは、Oracle GoldenGateでの移行の場合にスキーマに適用されます。ソース・データベース内のオブジェクトをOracle GoldenGateでレプリケートできない場合、レポートでは、これらのオブジェクトのリストが、このチェック・メッセージとともに示されます。

影響

リストされたオブジェクトは、Oracle GoldenGateで移行されません。

処置

スイッチオーバー時に、Oracle Data Pumpなどの別の移行方法を使用して、リストされた関連オブジェクトをターゲット・データベースに移動する必要があります。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.11 gg_user_objects_in_ggadmin_schemas

移行前アドバイザ・ツールのチェック`gg_user_objects_in_ggadmin_schemas`では、Oracle GoldenGate管理者権限があるスキーマにユーザー・オブジェクトが存在することが示されます。

結果の重要度

処置必須

修正あり

いいえ

ターゲット・クラウド

これはデフォルトのチェックです。これは次に適用されます:

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)
- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

有効範囲

SCHEMA

説明

このチェックは、Oracle GoldenGateでの移行の場合にスキーマに適用されます。スキーマ内のユーザー・オブジェクトにOracle GoldenGate管理者権限がある場合は、それらのスキーマがCPATレポートでリストされます。Oracle GoldenGateではそれらを移行できません。

影響

リストされたオブジェクトは、Oracle GoldenGateで移行されません。

処置

これらのスキーマをOracle GoldenGateでのデータ移行から除外してください。Oracle Data Pumpなどの別の移行方法を使用して、リストされた関連オブジェクトをターゲット・データベースに移動する必要があります。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.12 has_absent_default_tablespace

移行前アドバイザ・ツールのチェック`has_absent_default_tablespace`では、スキーマ所有者のデフォルト表領域が不足していることが示されます。

結果の重要度

確認必須。

修正あり

いいえ

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ATPD Autonomous Transaction Processing (専用)
- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

有効範囲

SCHEMA

説明

このチェックは、Oracle Data PumpおよびOracle GoldenGateでの移行の場合にスキーマに適用されます。スキーマ所有者のデフォルト表領域が1つ以上不足していることがCPATで検出された場合は、それらのスキーマがレポートでリストされます。

影響

有効なDEFAULT TABLESPACEがないスキーマは、ORA-00959エラーが原因で、ターゲット・インスタンス上に作成できません。

処置

それらのスキーマは、もう使用されない場合は削除します。ただし、それらのスキーマが使用されている場合は、そのスキーマ用の有効なデフォルト表領域を作成するか、DBA_TABLESPACEで問合せを実行して有効なすべての表領域の名前をリストしそれらの1つを有効なデフォルト表領域として選択することでデフォルト表領域を定義します。

関連トピック

- DBA_TABLESPACES

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.13 has_absent_temp_tablespace

移行前アドバイザ・ツールのチェックhas_absent_temp_tablespaceでは、スキーマ所有者の一時表領域が不足していることが示されます。

結果の重要度

確認必須。

修正あり

いいえ

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ATPD Autonomous Transaction Processing (専用)
- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

有効範囲

SCHEMA

説明

このチェックは、Oracle Data PumpおよびOracle GoldenGateでの移行の場合にスキーマに適用されます。スキーマ所有者の一時表領域が1つ以上不足していることがCPATで検出された場合は、それらのスキーマがレポートでリストされます。

影響

トランザクション処理用Oracle Autonomous Database専用インフラストラクチャ(ATPD)およびデータ・ウェアハウス用Oracle Autonomous Database専用インフラストラクチャ(ADWD)の場合、必要な一時表領域が移行前にターゲット上に作成されていないと、ORA-00959エラーが原因で、有効なTEMPORARY TABLESPACEがないソース・データベース・スキーマをターゲット・インスタンス上に作成できません。

処置

必要な一時表領域を移行開始前にOracle Autonomous Database専用インフラストラクチャ上に作成するか、移行ツールの起動時に表領域再マッピング・パラメータを使用して他の表領域をTEMP表領域にマップします。Oracleのゼロ・ダウンタイム移行およびデータベース移行サービスでは、移行の一部として表領域の事前作成およびマッピングを自動的に実行できます。

関連トピック

- DBA_TABLESPACES

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.14 has_active_data_guard_dedicated

移行前アドバイザ・ツールのチェックhas_active_data_guard_dedicatedでは、ソース・インスタンスでActive Data Guardが使用されているかどうかが検出されます。

結果の重要度

確認推奨。

修正あり

いいえ

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ATPD Autonomous Transaction Processing (専用)

有効範囲

`INSTANCE`

説明

このチェックでは、ソース・インスタンスでActive Data Guardが使用されているかどうかが検出されます。

影響

移行されるアプリケーションまたはスキーマがActive Data Guardの特定の機能に依存している場合、それらのアプリケーションは移行後に機能しなくなる可能性があります。

処置

ターゲットOracle Autonomous Databaseインスタンスで Autonomous Data Guardを使用することを検討してください。Autonomous Data Guardの機能の評価方法、およびAutonomous Data Guardの詳細は、Oracle Cloud Oracle Autonomous Database on Dedicated Exadata Infrastructureの*Autonomous Data Guard*を使用した障害および災害からのクリティカル・データベースの保護を参照してください。

関連トピック

- Autonomous Data Guardを使用した障害および災害からのクリティカル・データベースの保護

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.15 has_active_data_guard_serverless

移行前アドバイザ・ツールのチェック`has_active_data_guard_serverless`では、ソース・インスタンスでActive Data Guardが使用されているかどうかが検出されます。

結果の重要度

確認推奨。

修正あり

いいえ

ターゲット・クラウド

- ADWS Autonomous Data Warehouse (共有)
- ATPS Autonomous Transaction Processing (共有)

有効範囲

`INSTANCE`

説明

このチェックでは、ソース・インスタンスでActive Data Guardが使用されているかどうかが検出されます。

影響

移行されるアプリケーションまたはスキーマがActive Data Guardの特定の機能に依存している場合、それらのアプリケーションは移行後に機能しなくなる可能性があります。

処置

ターゲットOracle Autonomous DatabaseインスタンスでAutonomous Data Guardを使用することを検討してください。Autonomous Data Guardの機能の評価方法、およびAutonomous Data Guardの詳細は、Oracle CloudでのOracle Autonomous Database on Shared Exadata Infrastructureの使用の障害回復のための*Autonomous Database*とスタンバイ・データベースの併用を参照してください。

関連トピック

- 障害回復のためのAutonomous Databaseとスタンバイ・データベースの併用

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.16 has_basic_file_lobs

移行前アドバイザ・ツールのチェック`has_basic_file_lobs`では、BASICFILE LOB (Oracle Autonomous Databaseでサポートされていない)がスキーマに存在することが示されます。

結果の重要度

確認必須。

修正あり

いいえ

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

有効範囲

`SCHEMA`

説明

このチェックは、Oracle Data PumpおよびOracle GoldenGateでの移行の場合にスキーマに適用されます。スキーマ所有者の一時表領域の1つ以上にBASICFILE LOBが含まれていることがCPATで検出された場合は、それらのスキーマがレポートでリストされます。

影響

移行中、すべての`BASICFILE` LOBが、そのインポート時に自動的に`SECUREFILE` LOBに変換されます。

処置

処置は必要ありません。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.17 has_clustered_tables

移行前アドバイザ・ツールのチェックhas_clustered_tablesでは、表クラスタ(Oracle Autonomous Databaseでサポートされていない)がスキーマに存在することが示されます。

結果の重要度

確認推奨。

修正あり

いいえ

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

有効範囲

SCHEMA

説明

このチェックは、Oracle Data PumpおよびOracle GoldenGateでの移行の場合にスキーマに適用されます。1つ以上のスキーマに表クラスタが含まれていることがCPATで検出された場合は、それらのスキーマがレポートでリストされます。

影響

Oracle Autonomous Database上でCLUSTER句を使用して表を作成した場合、その表は通常の表として作成されます。

処置

処置は必要ありません。悪影響がないことを確認するために、いくらかのパフォーマンス・テストを実行することを検討してください。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.18 has_columns_of_rowid_type

移行前アドバイザ・ツールのチェックhas_columns_of_rowid_typeでは、移行できない、ROWIDデータ型の列がある表が示されます。

結果の重要度

処置必須。

修正あり

あり

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ATPD Autonomous Transaction Processing (専用)

有効範囲

SCHEMA

説明

このチェックは、Oracle Data PumpおよびOracle GoldenGateでの移行の場合にスキーマに適用されます。ROWIDデータ型は、Oracle Autonomous Database on Dedicated Exadata Infrastructureデプロイメントでは、デフォルトで有効になりません。

影響

デフォルトでは、ROWIDデータ型の列は、ATPDまたはADWDに移行できません。

処置

ターゲットADBDインスタンスで初期化パラメータALLOW_ROWID_COLUMN_TYPEをtrueに設定することで、ROWIDデータ型を有効にできます。ただし、それを有効にすると、ROWID列はローリング・アップグレード操作や、行を物理的に移動する他の内部操作との互換性がなくなることに注意してください。少なくとも、アップグレード中には、ROWIDに関連するデータベース・アクティビティを一時停止することをお薦めします。ROWIDの列を使用するアプリケーションでは、正確性を検証して、行が再配置された場合にアプリケーションで論理エラーが発生しないか確認する必要があります。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.19 has_columns_with_media_data_types_adb

移行前アドバイザ・ツールのチェックhas_columns_with_media_data_types_adbでは、移行できない、マルチメディア・データ型がある表が示されます。

結果の重要度

処置必須。

修正あり

いいえ

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

有効範囲

SCHEMA

説明

このチェックは、Oracle Data PumpおよびOracle GoldenGateでの移行の場合にスキーマに適用されます。マルチメディア・オブジェクト型(ORDSYSによるものなど)は、Oracle Autonomous Databaseでは使用できません。

影響

マルチメディア列がある表の移行は失敗します。

処置

Oracle Autonomous Databaseでは、メディア・データ型の列は使用できません。かわりに、メディア・タイプ・ストレージでのSecureFiles LOBの使用を検討することをお薦めします。

`Oracle_home/ord/im/admin/README.txt`のOracle Multimedia README.txtファイル、またはOracle SupportのドキュメントID 2555923.1の指示に従って、Oracle Multimediaのメソッドおよびパッケージが使用されているかどうかを判断します。Oracle Multimediaが使用されている場合は、Oracle Multimediaの置換に関する推奨事項について、Oracle SupportのドキュメントID 2347372.1を参照してください。Oracle Multimediaオブジェクト・タイプに格納されているデータをSecureFiles LOBに移動する方法の詳細は、Oracle SupportのドキュメントID 2375644.1のHow To Migrate Data From Oracle Multimedia Data Types to BLOB columnsを参照してください。

関連トピック

- Desupport of Oracle Multimedia Component in Oracle 19c (ドキュメントID 2555923.1)
- Oracle Multimedia Statement of Direction (ドキュメントID 2347372.1)
- How To Migrate Data From Oracle Multimedia Data Types to BLOB columns (ドキュメントID 2375644.1)

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.20 has_columns_with_media_data_types_default

移行前アドバイザ・ツールのチェック`has_columns_with_media_data_types_default`では、マルチメディア列がある表が示されます。

結果の重要度

処置必須。

修正あり

いいえ

有効範囲

`SCHEMA`

ターゲット・クラウド

- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

説明

このチェックは、Oracle Data PumpおよびOracle GoldenGateでの移行の場合にスキーマに適用されます。マルチメディア・オブジェクト型(`ORDSYS`によるものなど)は、Oracle Database 19c以降のリリースではサポートされなくなりました。

影響

マルチメディア列がある表の移行は失敗する可能性があります。

処置

Oracle Multimediaは、Oracle Database 19cでサポートが終了しました。メディア・タイプ・ストレージでのSecureFiles LOBの使用を検討することをお薦めします。

`Oracle_home/ord/im/admin/README.txt`のOracle Multimedia README.txtファイル、またはOracle SupportのドキュメントID 2555923.1の指示に従って、Oracle Multimediaのメソッドおよびパッケージが使用されているかどうかを判断します。Oracle Multimediaが使用されている場合は、Oracle Multimediaの置換に関する推奨事項について、Oracle SupportのドキュメントID 2347372.1を参照してください。Oracle Multimediaオブジェクト・タイプに格納されているデータをSecureFiles LOBに移動する方法の詳細は、Oracle SupportのドキュメントID 2375644.1のHow

To Migrate Data From Oracle Multimedia Data Types to BLOB columnsを参照してください。

関連トピック

- Desupport of Oracle Multimedia Component in Oracle 19c (ドキュメントID 2555923.1)
- Oracle Multimedia Statement of Direction (ドキュメントID 2347372.1)
- How To Migrate Data From Oracle Multimedia Data Types to BLOB columns (ドキュメントID 2375644.1)

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.21 has_columns_with_spatial_data_types

移行前アドバイザ・ツールのチェック`has_columns_with_spatial_data_types`では、十分サポートされていない空間オブジェクトがあることが示されます。

結果の重要度

確認必須。

修正あり

あり

ターゲット・クラウド

- ADWS Autonomous Data Warehouse (共有)
- ATPS Autonomous Transaction Processing (共有)

有効範囲

SCHEMA

説明

このチェックは、Oracle Data PumpおよびOracle GoldenGateでの移行の場合にスキーマに適用されます。これにより、空間データ型オブジェクトが存在することが示されます。

影響

空間オブジェクトの一部の機能はOracle Java (`JAVAVM`)の機能に依存しているため、`JAVAVM`が有効になるまでは、共有インフラストラクチャ上のOracle Autonomous Databaseで完全にはサポートされていないオブジェクトが存在する可能性があります。

処置

このSQLを実行してターゲット・システムでJAVAVM機能を有効にしてから、インスタンスを再起動してください:

```
BEGIN
   DBMS_CLOUD_ADMIN.ENABLE_FEATURE(
       feature_name => 'JAVAVM' );
END;
/
```

JAVAVM機能の有効化の詳細は、Oracle Cloud Oracle Autonomous Database Serverlessの使用の*Autonomous Database*での*Oracle Java*の使用を参照してください。ADBでのSpatialの使用の詳細は、Oracle Cloud Oracle Autonomous Database Serverlessの使用の*Autonomous Database*での*Oracle Spatial*の使用を参照してください。

関連トピック

- Autonomous DatabaseでのOracle Javaの使用
- Autonomous DatabaseとのOracle Spatialの使用

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.22 has_common_objects

移行前アドバイザ・ツールのチェックhas_common_objectsでは、データベース・インスタンスに共通オブジェクトがあることが示されます。

結果の重要度

処置必須。

修正あり

あり

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)
- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

有効範囲

INSTANCE

説明

これはデフォルトのチェックです。このチェックは、Oracle Data PumpおよびOracle GoldenGateでの移行の場合にソース・インスタンスに適用されます。これにより、共通オブジェクトが存在することが示されます。

影響

Oracle Data Pumpでは、共通オブジェクトは、Oracle Cloud内のOracle Autonomous Databaseに移行されず、これらのオブジェクトは、Oracle Autonomous Database (ADB)でサポートされません。共通オブジェクトに依存しているものはすべて、正しく移行できません。

処置

アプリケーションに必要なこれらの共通オブジェクトを、移行開始前にターゲット・システム上で再作成する必要があります。ADBをターゲットにする場合は、必要な共通オブジェクトをローカル・オブジェクトとして再作成する必要があります。これは、Oracle SupportのドキュメントID 2739952.1に示されているように、DBMS_METADATA.GET_DDLを使用して実行できます。

関連トピック

- How to Extract DDL for User including Privileges and Roles Using dbms_metadata.get_ddl (ドキュメントID 2739952.1)

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.23 has_compression_disabled_for_objects

移行前アドバイザ・ツールのチェックhas_compression_disabled_for_objectsでは、COMPRESSION句がない表またはパーティションがあることが示されます。

結果の重要度

確認推奨。

修正あり

いいえ

ターゲット・クラウド

- ADWS Autonomous Data Warehouse (共有)
- ATPS Autonomous Transaction Processing (共有)

有効範囲

SCHEMA

説明

このチェックは、Oracle Data PumpおよびOracle GoldenGateでの移行の場合にソース・スキーマに適用されます。これにより、COMPRESSION句がない表またはパーティションが存在することが示されます。表およびパーティションは、Oracle Autonomous Data Warehouse (ADW)でQUERY HIGHに圧縮する必要があります。

影響

ADWに移行する際に、表またはパーティションのSQLデータ定義言語(DDL)文にCOMPRESSION句が含まれていない場合は、デフォルト圧縮のQUERY HIGHを使用して移行中に作成されます。

処置

処置不要。この動作を変更するには、適切な圧縮句(または、NOCOMPRESSでも可)をエクスポート開始前に追加するか、圧縮句をインポート後に変更します。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.24 has_csmig_schema

移行前アドバイザ・ツールのチェックhas_csmig_schemaでは、CSSCANユーティリティがソース・データベースにインストールおよび構成されていることが示されます。

結果の重要度

確認推奨。

修正あり

あり

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)

- ATPS Autonomous Transaction Processing (共有)
- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

有効範囲

UNIVERSAL

説明

これはデフォルトのチェックです。CSSCANユーティリティはサポートされなくなり、Database Migration Assistant for Unicode (DMU)ツールに置き換えられました。

影響

CSSCANユーティリティに関連するオブジェクト、ユーザーまたはロールは、移行ツールで無視される場合があります。

処置

CSMIGユーザーおよびCSSCANユーティリティで作成されたオブジェクトを削除してください。次に例を示します:

```
BEGIN FOR REC IN (SELECT SYNONYM_NAME FROM DBA_SYNONYMS WHERE TABLE_OWNER = 'CSMIG')
LOOP
      EXECUTE IMMEDIATE 'DROP PUBLIC SYNONYM ' || REC.SYNONYM_NAME; END LOOP; END; /
DROP VIEW
      SYS.CSMV$KTFBUE; DROP USER CSMIG CASCADE;
```

Database Migration Assistant for Unicode (DMU)ツールを使用して、文字セット移行の問題をスキャンします。DMUの詳細は、Oracle SupportのドキュメントID 1272374.1を参照してください。

関連トピック

- The Database Migration Assistant for Unicode (DMU) Tool (ドキュメントID 1272374.1)

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.25 has_data_in_other_tablespaces_dedicated

移行前アドバイザ・ツールのチェックhas_data_in_other_tablespaces_dedicatedでは、専用インフラストラクチャ上のOracle Autonomous Databaseに移行する際の表領域制限の対象となるデータが特定されます。

結果の重要度

処置必須。

修正あり

いいえ

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ATPD Autonomous Transaction Processing (専用)

有効範囲

SCHEMA

説明

このチェックは、Oracle Data PumpおよびOracle GoldenGateでの移行の場合にソース・スキーマに適用されます。専用インフラストラクチャ上のAutonomous Databaseに移行する際の表領域制限の対象となるデータが存在することが示されま

す。

影響

ATPDおよびADWD (専用インフラストラクチャ)の場合は、ターゲット上に事前作成されていない表領域についてエラーが通知されます。表領域マッピングが使用されていない場合は、移行中にエラーが発生する可能性があります。

処置

Zero Downtime Migration (ZDM)またはDatabase Migration Service (DMS)を使用してデータベースを移行する場合は、表領域が自動的に事前作成およびマップされるので、この問題は発生しません。

Oracle Data Pumpを使用して手動で移行する場合は、Data Pumpの`impdp`パラメータ・ファイルに`IGNORE=TABLESPACE`および`REMAP_TABLESPACE='%:DATA'`を指定して、移行ツールの起動時に他の表領域が`DATA`表領域にマップされるようにします。

いずれの場合も、特定の表領域名に依存していないかどうかアプリケーションを評価する必要があります。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.26 has_data_in_other_tablespaces_serverless

移行前アドバイザ・ツールのチェック`has_data_in_other_tablespaces_serverless`では、共有インフラストラクチャ上のOracle Autonomous Databaseに移行する際の表領域制限の対象となるデータが特定されます。

結果の重要度

処置必須。

修正あり

いいえ

ターゲット・クラウド

- ADWS Autonomous Data Warehouse (共有)
- ATPS Autonomous Transaction Processing (共有)

有効範囲

`SCHEMA`

説明

このチェックは、Oracle Data PumpおよびOracle GoldenGateでの移行の場合にソース・スキーマに適用されます。これにより、他の表領域がマップされている表の存在が示されます。

影響

ユーザー定義表領域は、ATPSおよびADWS(サーバーレス・インフラストラクチャ)では許可されていません。このクラウド環境の各データベースには、単一のDATA表領域があります。表領域マッピングが使用されておらず、移行を実行するユーザーに`DATA`表領域に対する権限がない場合は、移行中にエラーが発生する可能性があります。

処置

Zero Downtime Migration (ZDM)またはDatabase Migration Service (DMS)を使用してデータベースを移行する場合は、表領域が自動的に事前作成およびマップされるので、この問題は発生しません。

Oracle Data Pumpを使用して手動で移行する場合は、Data Pumpの`impdp`パラメータ・ファイルに`IGNORE=TABLESPACE`および`REMAP_TABLESPACE='%:DATA'`を指定して、移行ツールの起動時に他の表領域が`DATA`表領域にマップされるようにします。

いずれの場合も、特定の表領域名に依存していないかどうかアプリケーションを評価する必要があります。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.27 has_db_link_synonyms

移行前アドバイザ・ツールのチェック`has_db_link_synonyms`では、データベース・リンクを含むシノニムがスキーマに含まれていることが示されます。

結果の重要度

確認推奨。

修正あり

あり

ターゲット・クラウド

- ADWS Autonomous Data Warehouse (共有)
- ATPS Autonomous Transaction Processing (共有)

有効範囲

`SCHEMA`

説明

このチェックは、Oracle Data PumpおよびOracle GoldenGateでの移行の場合にソース・スキーマに適用されます。データベース・リンクは移行できません。

影響

そのシノニムに依存するアプリケーションは、移行後は、それらのデータベース・リンクが再作成されるまでエラーになります。

処置

移行が完了したら、`DBMS_CLOUD_ADMIN.CREATE_DATABASE_LINK`を使用して、ターゲットOracle Autonomous Database内にデータベース・リンクを作成し、シノニムを再作成してください。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.28 has_db_links

移行前アドバイザ・ツールのチェック`has_db_links`では、データベース・リンクを含むシノニムがスキーマに含まれていることが示されます。

結果の重要度

確認必須

修正あり

いいえ

ターゲット・クラウド

- ADWS Autonomous Data Warehouse (共有)
- ATPS Autonomous Transaction Processing (共有)

有効範囲

SCHEMA

説明

このチェックは、Oracle Data PumpおよびOracle GoldenGateでの移行の場合にソース・スキーマに適用されます。データベース・リンクは移行できません。

影響

データベース・リンクに依存するアプリケーションは、移行後は、それらのデータベース・リンクが再作成されるまでエラーになります。

処置

移行する前に、それぞれのデータベース・スキーマでDBMS_CLOUD_ADMIN.CREATE_DATABASE_LINKを使用して、ADBにデータベース・リンクを手動で事前に作成してください。文の正しい順序は次のとおりです:

1. リンクを所有するスキーマを作成します。
2. DBMS_CLOUD_ADMIN.CREATE_DATABASE_LINKを使用してリンクを作成します。
3. 移行するスキーマをインポートします。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.29 has_dbms_credentials

移行前アドバイザ・ツールのチェックhas_dbms_credentialsでは、DBMS_CLOUD.CREATE_CREDENTIALで作成されていない資格証明がスキーマに含まれていることが示されます。

結果の重要度

確認必須

修正あり

いいえ

ターゲット・クラウド

- ADWS Autonomous Data Warehouse (共有)
- ATPS Autonomous Transaction Processing (共有)

有効範囲

SCHEMA

説明

このチェックは、Oracle Data PumpおよびOracle GoldenGateでの移行の場合にソース・スキーマに適用されます。DBMS_CREDENTIALまたはDBMS_SCHEDULERパッケージで最初に作成された資格証明は、自動的にはOracle Autonomous Databaseに移行できません。

影響

移行すると、DBMS_CREDENTIALまたはDBMS_SCHEDULERパッケージで最初に作成された資格証明を持つユーザーには、ORA-27486「権限が不足しています」というエラーが表示されますこれらの資格証明はADBSに自動的に移行できません。

処置

移行の完了後に、リストされた資格証明がターゲットOracle Autonomous Databaseインスタンスでまだ必要とされていることを確認します。これらの資格証明が必要な場合は、DBMS_CLOUD.CREATE_CREDENTIALを使用して資格証明を再作成します。詳細は、My Oracle SupportのドキュメントID 2746284.1を参照してください。

関連トピック

- Autonomous Database (共有) - ORA-27486でdbms_credential.create_credentialが失敗する(ドキュメントID 2746284.1)

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.30 has_dbms_credentials

移行前アドバイザ・ツールのチェックhas_dbms_credentialsでは、DBMS_CLOUD.CREATE_CREDENTIALで作成されていない資格証明がスキーマに含まれていることが示されます。

結果の重要度

確認必須

修正あり

いいえ

ターゲット・クラウド

- ADWS Autonomous Data Warehouse (共有)
- ATPS Autonomous Transaction Processing (共有)

有効範囲

SCHEMA

説明

このチェックは、Oracle Data PumpおよびOracle GoldenGateでの移行の場合にソース・スキーマに適用されます。DBMS_CREDENTIALまたはDBMS_SCHEDULERパッケージで最初に作成された資格証明は、自動的にはOracle Autonomous Databaseに移行できません。

影響

移行すると、DBMS_CREDENTIALまたはDBMS_SCHEDULERパッケージで最初に作成された資格証明を持つユーザーには、ORA-27486「権限が不足しています」というエラーが表示されますスキーマ所有者のデフォルトの表領域は'DATA'にする必要があります。

処置

スキーマ所有者のDEFAULT TABLESPACEが、ADBでは'DATA'に変更されます。複数の表領域でユーザーに割当て制限がある場合は、移行の完了後に、適切な割当て制限が設定されていることを確認してください。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.31 has_directories

移行前アドバイザ・ツールのチェックhas_directoriesでは、ソース・データベースにディレクトリ・オブジェクトがあることが示されます。

結果の重要度

確認必須

修正あり

いいえ

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

有効範囲

INSTANCE

説明

このチェックでは、ソース・データベースにディレクトリ・オブジェクトがあることが示されます。

影響

それらのディレクトリに依存するアプリケーションは、移行後、ソース・データベース上のそれらのディレクトリがターゲット・データベース上で再作成されるまで機能しません。

処置

移行の完了後に、Oracle Autonomous Databaseインスタンス上でそれらのディレクトリを再作成します。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.32 has_enabled_scheduler_jobs

移行前アドバイザ・ツールのチェックhas_enabled_scheduler_jobs では、Oracle Data Pumpエクスポートを妨げる可能性があるリスト・スケジューラ・ジョブがあることが示されます。

結果の重要度

確認推奨

修正あり

いいえ

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

有効範囲

INSTANCE

説明

これはデフォルトのチェックです。このチェックでは、Oracle Data Pumpエクスポートを妨げる可能性があるリスト・スケジューラ・ジョブがあることが示されます。

影響

スケジューラ・ジョブをFULLエクスポートと同時に実行すると、Oracle Data PumpエクスポートがORA-39127エラーで失敗する可能性があります。

処置

クリティカルでないスケジューラ・ジョブを無効にするか、スケジューラ・ジョブが実行されていないことが確実な時間帯にエクスポートを計画します。移行前にスケジューラ・ジョブを停止するか、スケジューラ・ジョブが実行されていないことが確実な時間帯にエクスポートを計画してください。

次のSQL文を実行すれば、移行中にスケジューラ・ジョブが実行されません:

```
ALTER SYSTEM SET JOB_QUEUE_PROCESSES=0;
```

この文の実行後に再起動する必要はありません。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.33 has_external_tables_dedicated

移行前アドバイザ・ツールのチェックhas_external_tables_dedicatedでは、ソース・データベースに非クラウド・オブジェクト・ストレージの外部表が存在することが示されます。

結果の重要度

確認必須

修正あり

いいえ

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ATPD Autonomous Transaction Processing (専用)

有効範囲

SCHEMA

説明

このチェックでは、ソース・データベースに非クラウド・オブジェクト・ストレージの外部表が存在することが示されます。これらの表は、Oracle Autonomous Databaseでは使用できません。

影響

ユーザーが作成した外部表に依存するアプリケーションは、想定されたとおりに機能しません。

処置

クラウド・オブジェクト・ストレージを使用する外部表の作成には、DBMS_CLOUDパッケージの使用を検討してください。

関連トピック

- 専用Exadataインフラストラクチャ上のAutonomous Databaseへのネットワーク・ファイル・ストレージのアタッチ

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.34 has_external_tables_default

移行前アドバイザ・ツールのチェックhas_external_tables_defaultでは、外部表が依存するDIRECTORYオブジェクトが作成されていないかぎり、外部表を移行できないことが示されます。

結果の重要度

処置必須

修正あり

いいえ

ターゲット・クラウド

これはデフォルトのチェックです。これは次に適用されます:

- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

有効範囲

SCHEMA

説明

このチェックでは、外部表が依存するDIRECTORYオブジェクトがターゲット・データベースにすでに作成されていないかぎり、外部表を移行できないことが示されます。

影響

外部表がまだ存在しないDIRECTORYオブジェクトに依存している場合、外部表のスキーマ・モードの移行は失敗します。

処置

移行の前に、ターゲット・データベースで必要なDIRECTORYオブジェクトを作成するか、全体モードを使用してターゲット・データベースに移行します。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.35 has_external_tables_serverless

移行前アドバイザ・ツールのチェックhas_external_tables_serverlessでは、ソース・データベースに非クラウド・オブジェクト・ストレージの外部表が存在することが示されます。

結果の重要度

確認必須

修正あり

いいえ

ターゲット・クラウド

- ADWS Autonomous Data Warehouse (共有)
- ATPS Autonomous Transaction Processing (共有)

有効範囲

SCHEMA

説明

非クラウド・オブジェクト・ストレージ外部表が見つかりました。これらのオブジェクトは、Oracle Autonomous Databaseでは使用できません。

影響

ユーザーが作成した外部表に依存するアプリケーションは、想定されたとおりに機能しません。Oracle Autonomous Database (ADB)の外部表は、Object Storage ServiceまたはFile Storage Serviceを使用して再作成する必要があります。

非クラウド・オブジェクト・ストレージ外部表を移行の一部として作成しようとすると、それらの表は外部表以外の表として作成されます。

処置

インポートされた空の表を削除します。DBMS_CLOUDパッケージでCloud Object Storage Serviceを使用して外部表を作成するか、File Storage Serviceを使用します。詳細は、次を参照してください

関連トピック

- Autonomous Databaseからのネットワーク・ファイル・システムへのアクセス

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.36 has_fmw_registry_in_system

移行前アドバイザ・ツールのチェックhas_fmw_registry_in_systemでは、移行前にSYSTEMスキーマからFusion Middlewareスキーマ・バージョン・レジストリを移動する必要があることが示されます。

結果の重要度

処置必須

修正あり

いいえ

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

有効範囲

INSTANCE

説明

Fusion Middlewareスキーマ・バージョン・レジストリはSYSTEMスキーマにあります。移行を開始する前に、SYSTEMスキーマの外に移動する必要があります。

影響

Fusion Middlewareバージョン・レジストリが移動されない場合、アップグレード後に、インストールされたFusion Middlewareアプリケーションに関する重要な情報が失われます。

処置

移行する前に、Fusion Middleware Upgrade Assistantコマンド`ua -moveRegistry`を実行します。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.37 has_illegal_characters_in_comments

移行前アドバイザ・ツールのチェック`has_illegal_characters_in_comments`では、データベース文字セットで有効でない文字が表のコメント内に存在することが示されます。

結果の重要度

確認必須

修正あり

いいえ

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)
- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

有効範囲

SCHEMA

説明

これは、データベースの文字セットで有効でない文字に関する、TABLEおよびCOLUMNのコメントの文字およびPL/SQLソースに対する、デフォルトのチェックです。

影響

有効でない文字があると、インポート中にORA-39346: オブジェクトの文字セット変換でデータが失われましたのエラーが発生する可能性があります。有効でない文字はデフォルトの置換文字に置き換えられます。

処置

移行の前に、有効でない文字を削除するか、有効な文字に置き換えてください。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.38 has_ilm_ado_policies

移行前アドバイザ・ツールのチェックhas_ilm_ado_policiesでは、情報ライフサイクル管理(ILM)自動データ最適化(ADO)ポリシーが移行されないことが示されます。

結果の重要度

確認必須

修正あり

いいえ

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

有効範囲

SCHEMA

説明

ILM自動データ最適化ポリシーを持つ表が存在します。これらのポリシーは、Oracle Autonomous Databaseに移行されません。

影響

ILM ADOポリシーを含む表(リリース12c以降)は、Oracle Autonomous Transaction Processing (ATP)およびOracle Autonomous Data Warehouse (ADW)でILM ADOポリシーなしで作成されます。

処置

処置は必要ありません。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.39 has_incompatible_jobs

移行前アドバイザ・ツールのチェックhas_incompatible_jobsでは、情報ライフサイクル管理(ILM)自動データ最適化(ADO)ポリシーが移行されないことが示されます。

結果の重要度

確認必須

修正あり

いいえ

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)

- ATPS Autonomous Transaction Processing (共有)

有効範囲

SCHEMA

説明

PLSQL_BLOCKまたはSTORED_PROCEDURE以外のスケジューラのジョブおよびプログラムはソースに存在しますが、Oracle Autonomous Database (ADB)ではサポートされていません。

影響

EXECUTABLEやEXTERNAL_SCRIPTなどのスケジューラ・ジョブおよびプログラムのタイプは、Oracle Autonomous Databaseでは実行されません。

処置

サポートされていないジョブまたはプログラムのタイプを使用するデータベースは、Oracle Autonomous Databaseに移行する前に変更する必要があります。ADBで許可されるタイプを使用して必要なジョブまたはプログラムを再作成してください

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.40 has_index_organized_tables

移行前アドバイザ・ツールのチェックhas_index_organized_tablesでは、ソース・データベースに索引構成表が存在することが示されます。

結果の重要度

確認推奨

修正あり

いいえ

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

有効範囲

SCHEMA

説明

索引構成表は、Oracle Autonomous Database (ADB)では使用できません。ただし、作成しようとしてもエラーは生成されません。かわりに、主キー索引が設定されたヒープ構成表が作成されます。

影響

再作成された表の実行方法は異なるため、確認する必要があります。

処置

ターゲット・データベース内の表は、非索引構成表(通常の表)として作成されます。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.41 has_java_objects

移行前アドバイザ・ツールのチェックhas_java_objectsでは、ソース・データベースにJavaオブジェクトが存在することが示されます。

結果の重要度

処置必須

修正あり

あり

ターゲット・クラウド

- ADWS Autonomous Data Warehouse (共有)
- ATPS Autonomous Transaction Processing (共有)

有効範囲

SCHEMA

説明

Javaオブジェクトはデフォルトでは移行されません。

影響

Java仮想マシン(JAVAVM)機能がターゲット・システムで有効になっていない場合、Javaオブジェクトに依存するアプリケーションは移行後に失敗します。

処置

必須でないJavaオブジェクトは移行プロセスから除外する必要があります。*Oracle Autonomous Database*での*Oracle Autonomous Database on Shared Exadata Infrastructure*の使用の「Autonomous DatabaseでのOracle Javaの使用」の説明に従って、ターゲット・システムでJAVAVM機能を有効にします

関連トピック

- Autonomous DatabaseでのOracle Javaの使用

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.42 has_java_source

移行前アドバイザ・ツールのチェックhas_java_sourceでは、ソース・データベースにJavaソースが存在することが示されます。

結果の重要度

処置必須

修正あり

あり

有効範囲

SCHEMA

ターゲット・クラウド

- ADWS Autonomous Data Warehouse (共有)
- ATPS Autonomous Transaction Processing (共有)

説明

Javaソースはデフォルトでは移行されません。

影響

Java仮想マシン(JAVAVM)機能がターゲット・システムで有効になっていない場合、Javaオブジェクトに依存するアプリケーションは移行後に失敗します。

処置

必須でないJavaオブジェクトは移行プロセスから除外する必要があります。*Oracle Autonomous Database*での*Oracle Autonomous Database on Shared Exadata Infrastructure*の使用の「Autonomous DatabaseでのOracle Javaの使用」の説明に従って、ターゲット・システムでJAVAVM機能を有効にします

関連トピック

- Autonomous DatabaseでのOracle Javaの使用

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.43 has_libraries

移行前アドバイザ・ツールのチェックhas_librariesでは、CREATE LIBRARY文を必要とするアプリケーションが存在することが示されます。

結果の重要度

処置必須

修正あり

いいえ

有効範囲

SCHEMA

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

説明

CREATE LIBRARY文は、Oracle Autonomous Databaseでは使用できません。

影響

これらのライブラリに依存するアプリケーションは、ライブラリがターゲット・インスタンスで作成されないため、失敗します。

処置

リストされているライブラリの依存性を削除するには、アプリケーションを更新する必要があります。

外部ライブラリで以前に実装されたビジネス・ロジック用の関数を使用することを検討してください。

関連トピック

- 関数

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.44 has_logging_off_for_partitions

移行前アドバイザ・ツールのチェックhas_logging_off_for_partitionsでは、NOLOGGINGストレージ属性を使用するパーティションがあることが示されます。

結果の重要度

確認推奨

修正あり

いいえ

有効範囲

SCHEMA

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

説明

NOLOGGINGストレージ属性を持つパーティションは、移行中にLOGGINGに変更されます。

影響

NOLOGGINGで作成されたパーティションは、LOGGINGのパーティションとしてOracle Autonomous Databaseで自動的に作成されます。DBA_TAB_PARTITIONSのLOGGING属性を確認します。

処置

処置不要。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.45 has_logging_off_for_subpartitions

移行前アドバイザ・ツールのチェックhas_logging_off_for_subpartitionsでは、NOLOGGINGストレージ属性を使用するパーティションがあることが示されます。

結果の重要度

確認推奨

修正あり

いいえ

有効範囲

SCHEMA

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

説明

NOLOGGINGストレージ属性を持つサブパーティションは、移行中にLOGGINGに変更されます。

影響

NOLOGGINGで作成されたサブパーティションは、LOGGINGのサブパーティションとしてOracle Autonomous Databaseで自動的に作成されます。DBA_TAB_SUBPARTITIONSのLOGGING属性を確認します。

処置

処置不要。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.46 has_logging_off_for_tables

移行前アドバイザ・ツールのチェックhas_logging_off_for_tablesでは、NOLOGGINGストレージ属性を使用する表があることが示されます。

結果の重要度

確認推奨

修正あり

いいえ

有効範囲

SCHEMA

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

説明

NOLOGGINGストレージ属性を持つ表は、移行中にLOGGINGに変更されます。

影響

NOLOGGINGで作成された表は、LOGGINGの表としてOracle Autonomous Databaseで自動的に作成されます。

DBA_TABLESのLOGGING属性を確認します。

処置

処置不要。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.47 has_low_streams_pool_size

移行前アドバイザ・ツールのチェックhas_low_streams_pool_sizeでは、予期しない、あるいは正しくない属性のマイニング・モデルが検出されたことが示されます。

結果の重要度

処置必須

修正あり

いいえ

有効範囲

SCHEMA

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)
- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

説明

マイニング・モデルは、データ・マイニングを実行するデータベース・スキーマ・オブジェクトです。予期しない属性または正しくない属性を持つマイニング・モデルが検出されました。これらのマイニング・モデルは移行されません。

影響

問題のあるマイニング・モデルは適切にエクスポートされず、インポート時のORA-39083エラーの原因となります。

処置

パッチID 33270686のダウンロードおよび適用

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.48 has_noexport_object_grants

移行前アドバイザ・ツールのチェックhas_noexport_object_grantsでは、Oracle Data Pumpがすべてのオブジェクト権限をエクスポートできないことが示されます。

結果の重要度

確認必須

修正あり

あり

有効範囲

SCHEMA

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)
- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

説明

Oracle Data Pumpでは、すべてのオブジェクト権限をエクスポートできません。

影響

アプリケーションに必要なオブジェクト権限がターゲット・インスタンスにないため、Oracle Data Pumpがこれをターゲット・インスタンスにエクスポートできない可能性があります。

処置

ターゲット・インスタンスで必要な権限を再作成します。詳細は、My Oracle SupportのドキュメントID 1911151.1を参照してください。SELECTはシステム・オブジェクト上で使用できなくなったため、システム・オブジェクトのSELECT権限は、READ権限に置換する必要があることに注意してください。

関連トピック

- Data Pump: SYS所有オブジェクトの権限がData Pumpで転送されず、ターゲット・データベースで見つからない(ドキュメントID 1911151.1)

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.49 has_parallel_indexes_enabled

移行前アドバイザ・ツールのチェックhas_parallel_indexes_enabledでは、PARALLEL句索引が存在することが示されます。

結果の重要度

確認推奨

修正あり

いいえ

有効範囲

SCHEMA

ターゲット・クラウド

- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

説明

INDEXで並列DEGREEが1より大きい場合、この設定により、移行後に予期しない動作が発生する可能性があります。

影響

Oracle Autonomous Database Transaction Processing (ATP)に移行するとき、ソース・データベースの索引にPARALLEL句が指定されている場合、Oracle Data Pumpを使用、または手動でターゲット・データベースに作成された句が索引とともに残ります。PARALLEL度が1より大きい場合、この構成によってSQL文が並列実行され、エンド・ユーザーには認識されないことがあります。

処置

シリアル処理を指定するには、INDEX並列句をNOPARALLELに変更するか、PARALLEL度を移行の前後に1に変更します。

関連トピック

- Data Pump: SYS所有オブジェクトの権限がData Pumpで転送されず、ターゲット・データベースで見つからない(ドキュメントID 1911151.1)

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.50 has_profile_not_default

移行前アドバイザ・ツールのチェックhas_profile_not_defaultでは、PROFILEを使用できないスキーマがターゲット・システムに存在することが示されます。

結果の重要度

ランタイム

修正あり

あり

有効範囲

SCHEMA

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)
- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

説明

PROFILEをターゲット・システムで使用できないスキーマを識別します。

影響

PROFILEがないため、ターゲット・システムのスキーマの作成が失敗します。

処置

Oracle Data PumpをFULLモードで使用するか、ターゲット・システムで移行する前に必要なプロファイルを作成してから、CPATを--gettargetpropsオプションとともに使用して作成されたプロパティ・ファイルで--analysispropsオプションを使用します。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.51 has_public_synonyms

移行前アドバイザ・ツールのチェックhas_public_synonymsでは、パブリック・シノニムがソース・システム・スキーマに存在することが示されます。

結果の重要度

確認必須

修正あり

いいえ

有効範囲

SCHEMA_ONLY

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)
- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

説明

パブリック・シノニムを含むスキーマを識別します。Oracle Data Pumpでは、パブリック・シノニムはSCHEMAモードで移行されません。

影響

パブリック・シノニムに依存するアプリケーションは、パブリック・シノニムがターゲット・インスタンスで再作成されるまで正しく機能しません。

処置

Oracle Data PumpをFULLモードで使用するか、リストされた関連オブジェクトをターゲット・インスタンスで再作成します。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.52 has_refs_to_restricted_packages_dedicated

移行前アドバイザ・ツールのチェックhas_refs_to_restricted_packages_dedicatedでは、部分的または完全にサポートされていないパッケージへの参照があることが示されます。

結果の重要度

確認必須

修正あり

いいえ

有効範囲

SCHEMA

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ATPD Autonomous Transaction Processing (専用)

説明

サポートされていないか、サポートが一部だけのパッケージへの参照を確認します。

影響

サポートされていない、または使用が制限されているパッケージを参照するアプリケーションは、失敗する可能性があります。

処置

サポートされていないパッケージを参照するアプリケーションは、Oracle Autonomous Database Dedicatedに移行する前に変更する必要があります。部分的にサポートされているパッケージを参照するアプリケーションでは、無制限のファンクションおよびプロシージャのみを使用することを確認するテストと検証が必要です。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.53 has_refs_to_restricted_packages_serverless

移行前アドバイザ・ツールのチェック`has_refs_to_restricted_packages_serverless`では、部分的または完全にサポートされていないパッケージへの参照があることが示されます。

結果の重要度

確認必須

修正あり

いいえ

有効範囲

SCHEMA

ターゲット・クラウド

- ADWS Autonomous Data Warehouse (共有)
- ATPS Autonomous Transaction Processing (共有)

説明

サポートされていないか、サポートが一部だけのパッケージへの参照を確認します。

影響

サポートされていない、または使用が制限されているパッケージを参照するアプリケーションは、失敗する可能性があります。

処置

サポートされていないパッケージを参照するアプリケーションは、Oracle Autonomous Database Serverlessに移行する前に変更する必要があります。部分的にサポートされているパッケージを参照するアプリケーションでは、無制限のファンクションおよびプロシージャのみを使用することを確認するテストと検証が必要です。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.54 has_refs_to_user_objects_in_sys

移行前アドバイザ・ツールのチェックhas_refs_to_user_objects_in_sysでは、SYSまたはSYSTEMに応じたユーザー・スキーマ・オブジェクトがあることが示されます。

結果の重要度

処置必須

修正あり

いいえ

有効範囲

SCHEMA

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

説明

ユーザー・スキーマ内のオブジェクトが、SYSまたはSYSTEMスキーマ内のユーザー定義オブジェクトに依存するかどうかが検出されます。

影響

SYSまたはSYSTEM内のユーザー定義オブジェクトに依存するスキーマの移行は失敗します。

処置

移行前にSYSおよびSYSTEMスキーマ内のユーザー定義オブジェクトを移動し、参照を更新することをお薦めします。不要になったユーザー定義オブジェクトは削除することを検討してください。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.55 has_role_privileges

移行前アドバイザ・ツールのチェックhas_role_privilegesでは、ソース・データベースで使用される一部のロール権限がターゲット・データベースでは提供されないことが示されます

結果の重要度

処置必須

修正あり

いいえ

有効範囲

SCHEMA

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)

- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

説明

ターゲットOracle Autonomous Databaseで使用できないソース・データベースで使用されるロール権限の存在を検出します。

影響

移行後、ロール権限の問題により、アプリケーションで操作が失敗する可能性があります。

処置

ターゲットのOracle Autonomous Databaseインスタンスで使用できないソース・データベースで付与されたロールの代替方法を見つけます。たとえば、ソース・インスタンスでDBAロールを付与されたスキーマについては、PDB_DBAロールで代替できる場合があります。同様に、ソース・インスタンスでDATAPUMP_IMP_FULL_DATABASEまたはIMP_FULL_DATABASEが付与されたスキーマについては、ターゲット・インスタンスのDATAPUMP_CLOUD_IMPロールで代替できる場合があります。このような代替方法が適切かどうかは、テストでのみ、またこれらのロール権限が発生するアプリケーションをよく理解している専門家が判断できます。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.56 has_sqlt_objects_adb

移行前アドバイザ・ツールのチェックhas_sqlt_objects_adbでは、SQLTXPLAINオブジェクトが検出されることが示されます。

結果の重要度

確認推奨

修正あり

いいえ

有効範囲

UNIVERSAL

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

説明

Oracle Autonomous DatabaseでサポートされていないSQLTXPLAIN (SQLT)オブジェクトの存在を検出します。

影響

SQLTXPLAINに関連するオブジェクトは、Oracle Autonomous Database (ADB)へのインポートで失敗し、インポート・エラーが発生する可能性があります。

処置

ソース・データベースをOracle Autonomous Databaseに移行する管理者がSQLTXPLAINのインストール・ディレクトリに`sqdrop.sql`を適用して、すべてのSQLTXPLAINおよびSQLTXADMINオブジェクトを削除することをお薦めします。詳細は、My Oracle SupportのドキュメントID 1614107.1を参照してください。

関連トピック

- SQLT使用手順(ドキュメントID 1614107.1)

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.57 has_sqlt_objects_default

移行前アドバイザ・ツールのチェック`has_sqlt_objects_default`では、Oracle Data PumpでエクスポートされないSQLTXPLAINオブジェクトが検出されることが示されます。

結果の重要度

確認推奨

修正あり

いいえ

有効範囲

UNIVERSAL

ターゲット・クラウド

- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

説明

Oracle Data PumpでエクスポートされないSQLTXPLAIN (SQLT)オブジェクトの存在が検出されます。

影響

SQLTXPLAINに関連する一部のオブジェクトがターゲット・インスタンスにインポートされないため、インポート・エラーが発生する可能性があります。

処置

インポートの完了後は、SQLTXPLAINユーザーがターゲット環境で`sqcreate.sql`を実行することをお薦めします。`sqcreate.sql`スクリプトは`sqdrop.sql`を実行し、必要なすべてのオブジェクトを再インストールします。詳細は、My Oracle SupportのドキュメントID 1614107.1を参照してください。

関連トピック

- SQLT使用手順(ドキュメントID 1614107.1)

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.58 has_sys_privileges

移行前アドバイザ・ツールのチェック`has_sys_privileges`では、ソース・データベースの一部のシステム権限がターゲット・データベースでは提供されないことが示されます。

結果の重要度

処置必須

修正あり

いいえ

有効範囲

SCHEMA

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

説明

Oracle Autonomous Databaseで使用できないソース・データベースが使用されているシステム権限を検出します。

影響

システム権限の問題のため、Oracle Autonomous Databaseで操作が失敗する可能性があります。

処置

Oracle Autonomous Databaseですべてのシステム権限が必要かどうかを確認し、不要になった権限の付与を解除します。ターゲットのOracle Autonomous Database (ADB)で使用できない、付与済のシステム権限に対する代替方法を見つけます。たとえば、ADBインスタンスのスキーマでは、GRANT CREATE JOBをUSER-WHO-HAD-CREATE-ANY-JOBで置換します。このような代替が適切であるかどうかは、当該アプリケーションおよびテストをよく理解している専門家のみが判断できます。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.59 has_tables_that_fail_with_dblink

移行前アドバイザ・ツールのチェックhas_tables_that_fail_with_dblinkでは、LONGまたはLONG RAWデータ型の表があることが示されます

結果の重要度

処置必須

修正あり

いいえ

有効範囲

SCHEMA

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)

- ATPS Autonomous Transaction Processing (共有)
- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

説明

LONGまたはLONG RAWデータ型を含む表は、Oracle Data PumpではDBLINKで移行されません。

LONGデータ型(LONG、LONG RAW、LONG VARCHAR、LONG VARRAW)のすべての形式は、Oracle8iリリース8.1.6で非推奨になりました。以降のリリースでは、LONGデータ型は既存のアプリケーションとの下位互換性のために提供されていました。以降のリリースで開発された新しいアプリケーションでは、大量の文字データにCLOBおよびNCLOBデータ型を使用することをお薦めします。

影響

LONGまたはLONG RAWデータ型を含む表に依存しているアプリケーションは失敗します。

処置

DBLINKを指定せずにOracle Data Pumpを使用するか、LONGまたはLONG RAWデータ型の列を含むスキーマおよび表を除外します。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.60 has_tables_with_long_raw_datatype

移行前アドバイザ・ツールのチェックhas_tables_with_long_raw_datatypeでは、LONGまたはLONG RAWデータ型の表があることが示されます

結果の重要度

処置必須

修正あり

いいえ

有効範囲

SCHEMA

ターゲット・クラウド

- ADWS Autonomous Data Warehouse (共有)

説明

Oracle Hybrid Columnar Compression (HCC)の圧縮句が表に含まれる場合、または圧縮がDISABLEDの場合は、LONGまたはLONGRAWのデータを含む表がADWSでサポートされません。

影響

LONGまたはLONG RAWデータ型を含む表は移行されません。

Oracle Autonomous Data Warehouse (ADW)では、表にHCC圧縮句がある場合、または圧縮がDISABLEDの場合、LONGまたはLONG RAWデータ型の表は作成されません。この場合、デフォルトでHCCで圧縮されている表がADWでデフォルトで圧縮されます。

LONGデータ型(LONG、LONG RAW、LONG VARCHAR、LONG VARRAW)のすべての形式は、Oracle8iリリース8.1.6で非

推奨になりました。以降のリリースでは、LONGデータ型は既存のアプリケーションとの下位互換性のために提供されていました。以降のリリースで開発された新しいアプリケーションでは、大量の文字データにCLOBおよびNCLOBデータ型を使用することをお薦めします。

処置

圧縮を有効にして、ADWで表を手動で作成することをお薦めします。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.61 has_tables_with_xmltype_column

移行前アドバイザ・ツールのチェックhas_tables_with_xmltype_columnでは、XMLTYPE列を含む表があることが示されます。

結果の重要度

処置必須

修正あり

いいえ

有効範囲

SCHEMA

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

説明

XMLType列を含む表は、STORAGE_TYPE設定がBINARYでなければ移行されません。

影響

BINARYとして格納されていないXMLType列に依存するアプリケーションは失敗します。

処置

CLOBまたはオブジェクト・リレーショナル・ストレージで定義されているXMLType列がある表は、Oracle Autonomous Databaseではサポートされていません。関連するオブジェクトの列XMLSCHEMAが空でない場合は、アプリケーションでXMLスキーマ・オブジェクトが使用されているため、追加作業が必要になる可能性があります。非スキーマ・タイプについては、BINARY記憶域オプションを使用する必要があります。CLOB列をBINARYに変換する方法の詳細は、Oracle SupportのドキュメントID 1581065.1を参照してください。

関連トピック

- How to Convert Basicfile CLOB to Securfile Binary XML (ドキュメントID 1581065.1)

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.62 has_trusted_server_entries

移行前アドバイザ・ツールのチェックhas_trusted_server_entriesでは、再作成できないTRUSTED_SERVERエントリがOracle Autonomous Databaseに存在することが示されます。

結果の重要度

ランタイム

修正あり

いいえ

有効範囲

INSTANCE

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

説明

TRUSTED_SERVERエントリがあるかを確認してください。これらのエントリは、Oracle Autonomous Database (ADB)で再作成できません。

影響

DBMS_DISTRIBUTED_TRUST_ADMINパッケージは、Oracle Autonomous Database (ADB)では使用できません。その結果、デフォルト(Trusted:All)以外のTRUSTED_SERVERエントリは、ターゲットADBインスタンスで再作成されません。

処置

ソース・データベースからターゲット・データベースへの移行中にOracle Data Pumpによって報告される例外を回避するには、exclude=trusted_db_linkを指定します。ADBインスタンスへのアクセスを制御する場合は、Oracle Cloud Infrastructureのファイアウォール機能を使用してADBインスタンスへのアクセスを制御してください。

関連トピック

- 仮想ファイアウォールを使用したクラウド・リソースの保護

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.63 has_user_defined_objects_in_sys

移行前アドバイザ・ツールのチェックhas_user_defined_objects_in_sysでは、SYSまたはSYSTEMのユーザー定義オブジェクトがあることが示されます。

結果の重要度

処置必須

修正あり

いいえ

有効範囲

INSTANCE

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

説明

このチェックは、ユーザー定義オブジェクトがSYSまたはSYSTEMスキーマに存在することを示します

影響

SYSまたはSYSTEMスキーマ内のユーザー定義オブジェクトは移行されません。SYSおよびSYSTEM内のユーザー定義オブジェクトに依存しているアプリケーションは失敗します。

処置

¥

移行の前に、SYSおよびSYSTEMからユーザー定義オブジェクトを移動することをお薦めします。これらのオブジェクトへの参照がハードコードされている場合は更新してください。不要になったユーザー定義オブジェクトは削除することを検討してください。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.64 has_users_with_10g_password_version

移行前アドバイザ・ツールのチェックhas_users_with_10g_password_versionでは、10Gパスワード・バージョンを使用するユーザー・アカウントがあることが示されます。

結果の重要度

確認必須。

修正あり

いいえ

有効範囲

SCHEMA

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)
- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

説明

このチェックは、10Gパスワード・バージョンを使用しているユーザーがソース・データベースに存在することを示します。このパスワード・バージョンはサポートされなくなりました。移行後、10Gパスワード・バージョンで検証されていたユーザーはログインできなくなり

ます。

影響

移行後、10Gパスワード・バージョンで識別されていたユーザーはデータベースへの接続に失敗し、ORA-1017エラーを受け取ります。Oracle Data Pumpの移行中に、ORA-39384アラートが生成されます。

処置

Oracle Data Pump移行のアラートを回避するために、移行前に10Gパスワード・バージョンを使用するユーザーのパスワードを変更することをお薦めします。または、移行後にこれらのユーザーのパスワードを変更して、ログインの失敗を回避することもできます。詳細は、Oracle SupportのドキュメントID 2289453.1を参照してください。

関連トピック

- ORA-39384: 警告: ユーザー<USERNAME>がロックされ、インポート中にパスワードが失効しました(ドキュメントID 2289453.1)

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.65 has_sys_privileges

移行前アドバイザ・ツールのチェックhas_sys_privilegesでは、ソース・データベースの一部のシステム権限がターゲット・データベースでは提供されないことが示されます。

結果の重要度

処置必須

修正あり

いいえ

有効範囲

SCHEMA

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

説明

Oracle Autonomous Databaseで使用できないソース・データベースが使用されているシステム権限を検出します。

影響

システム権限の問題のため、Oracle Autonomous Databaseで操作が失敗する可能性があります。

処置

Oracle Autonomous Databaseですべてのシステム権限が必要かどうかを確認し、不要になった権限の付与を解除します。ターゲットのOracle Autonomous Database (ADB)で使用できない、付与済のシステム権限に対する代替方法を見つけます。たとえば、ADBインスタンスのスキーマでは、GRANT CREATE JOBをUSER-WHO-HAD-CREATE-ANY-JOBで置換します。このような代替が適切であるかどうかは、当該アプリケーションおよびテストをよく理解している専門家のみが判断でき

ます。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.66 has_tables_that_fail_with_dblink

移行前アドバイザ・ツールのチェックhas_tables_that_fail_with_dblinkでは、LONGまたはLONG RAWデータ型の表があることが示されます

結果の重要度

処置必須

修正あり

いいえ

有効範囲

SCHEMA

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)
- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

説明

LONGまたはLONG RAWデータ型を含む表は、Oracle Data PumpではDBLINKで移行されません。

LONGデータ型(LONG、LONG RAW、LONG VARCHAR、LONG VARRAW)のすべての形式は、Oracle8iリリース8.1.6で非推奨になりました。以降のリリースでは、LONGデータ型は既存のアプリケーションとの下位互換性のために提供されていました。以降のリリースで開発された新しいアプリケーションでは、大量の文字データにCLOBおよびNCLOBデータ型を使用することをお薦めします。

影響

LONGまたはLONG RAWデータ型を含む表に依存しているアプリケーションは失敗します。

処置

DBLINKを指定せずにOracle Data Pumpを使用するか、LONGまたはLONG RAWデータ型の列を含むスキーマおよび表を除外します。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.67 has_tables_with_long_raw_datatype

移行前アドバイザ・ツールのチェックhas_tables_with_long_raw_datatypeでは、LONGまたはLONG RAWデータ型の表があることが示されます

結果の重要度

処置必須

修正あり

いいえ

有効範囲

SCHEMA

ターゲット・クラウド

- ADWS Autonomous Data Warehouse (共有)

説明

Oracle Hybrid Columnar Compression (HCC)の圧縮句が表に含まれる場合、または圧縮がDISABLEDの場合は、LONGまたはLONGRAWのデータを含む表がADWSでサポートされません。

影響

LONGまたはLONG RAWデータ型を含む表は移行されません。

Oracle Autonomous Data Warehouse (ADW)では、表にHCC圧縮句がある場合、または圧縮がDISABLEDの場合、LONGまたはLONG RAWデータ型の表は作成されません。この場合、デフォルトでHCCで圧縮されている表がADWでデフォルトで圧縮されます。

LONGデータ型(LONG、LONG RAW、LONG VARCHAR、LONG VARRAW)のすべての形式は、Oracle8iリリース8.1.6で非推奨になりました。以降のリリースでは、LONGデータ型は既存のアプリケーションとの下位互換性のために提供されていました。以降のリリースで開発された新しいアプリケーションでは、大量の文字データにCLOBおよびNCLOBデータ型を使用することをお薦めします。

処置

圧縮を有効にして、ADWで表を手動で作成することをお薦めします。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.68 has_tables_with_xmltype_column

移行前アドバイザ・ツールのチェックhas_tables_with_xmltype_columnでは、XMLTYPE列を含む表があることが示されます。

結果の重要度

処置必須

修正あり

いいえ

有効範囲

SCHEMA

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

説明

XMLType列を含む表は、STORAGE_TYPE設定がBINARYでなければ移行されません。

影響

BINARYとして格納されていないXMLType列に依存するアプリケーションは失敗します。

処置

CLOBまたはオブジェクト・リレーショナル・ストレージで定義されているXMLType列がある表は、Oracle Autonomous Databaseではサポートされていません。関連するオブジェクトの列XMLSCHEMAが空でない場合は、アプリケーションでXMLスキーマ・オブジェクトが使用されているため、追加作業が必要になる可能性があります。非スキーマ・タイプについては、BINARY記憶域オプションを使用する必要があります。CLOB列をBINARYに変換する方法の詳細は、Oracle SupportのドキュメントID 1581065.1を参照してください。

関連トピック

- How to Convert Basicfile CLOB to Securfile Binary XML (ドキュメントID 1581065.1)

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.69 has_trusted_server_entries

移行前アドバイザ・ツールのチェックhas_trusted_server_entriesでは、再作成できないTRUSTED_SERVERエントリがOracle Autonomous Databaseに存在することが示されます。

結果の重要度

ランタイム

修正あり

いいえ

有効範囲

INSTANCE

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

説明

TRUSTED_SERVERエントリがあるかを確認してください。これらのエントリは、Oracle Autonomous Database (ADB)で再作成できません。

影響

DBMS_DISTRIBUTED_TRUST_ADMINパッケージは、Oracle Autonomous Database (ADB)では使用できません。その結果、デフォルト(Trusted:All)以外のTRUSTED_SERVERエントリは、ターゲットADBインスタンスで再作成されません。

処置

ソース・データベースからターゲット・データベースへの移行中にOracle Data Pumpによって報告される例外を回避するには、`exclude=trusted_db_link`を指定します。ADBインスタンスへのアクセスを制御する場合は、Oracle Cloud Infrastructureのファイアウォール機能を使用してADBインスタンスへのアクセスを制御してください。

関連トピック

- 仮想ファイアウォールを使用したクラウド・リソースの保護

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.70 has_user_defined_objects_in_sys

移行前アドバイザ・ツールのチェック`has_user_defined_objects_in_sys`では、SYSまたはSYSTEMのユーザー定義オブジェクトがあることが示されます。

結果の重要度

処置必須

修正あり

いいえ

有効範囲

`INSTANCE`

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

説明

このチェックは、ユーザー定義オブジェクトがSYSまたはSYSTEMスキーマに存在することを示します

影響

SYSまたはSYSTEMスキーマ内のユーザー定義オブジェクトは移行されません。SYSおよびSYSTEM内のユーザー定義オブジェクトに依存しているアプリケーションは失敗します。

処置

¥

移行の前に、SYSおよびSYSTEMからユーザー定義オブジェクトを移動することをお薦めします。これらのオブジェクトへの参照がハードコードされている場合は更新してください。不要になったユーザー定義オブジェクトは削除することを検討してください。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.71 has_users_with_10g_password_version

移行前アドバイザ・ツールのチェック`has_users_with_10g_password_version`では、10Gパスワード・バージョンを使用するユーザー・アカウントがあることが示されます。

結果の重要度

確認必須。

修正あり

いいえ

有効範囲

SCHEMA

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)
- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

説明

このチェックは、10Gパスワード・バージョンを使用しているユーザーがソース・データベースに存在することを示します。このパスワード・バージョンはサポートされなくなりました。移行後、10Gパスワード・バージョンで検証されていたユーザーはログインできなくなります。

影響

移行後、10Gパスワード・バージョンで識別されていたユーザーはデータベースへの接続に失敗し、ORA-1017エラーを受け取ります。Oracle Data Pumpの移行中に、ORA-39384アラートが生成されます。

処置

Oracle Data Pump移行のアラートを回避するために、移行前に10Gパスワード・バージョンを使用するユーザーのパスワードを変更することをお薦めします。または、移行後にこれらのユーザーのパスワードを変更して、ログインの失敗を回避することもできます。詳細は、Oracle SupportのドキュメントID 2289453.1を参照してください。

関連トピック

- ORA-39384: 警告: ユーザー<USERNAME>がロックされ、インポート中にパスワードが失効しました(ドキュメントID 2289453.1)

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.72 has_xmlschema_objects

移行前アドバイザ・ツールのチェックhas_xmlschema_objectsでは、ソース・データベースにXMLスキーマ・オブジェクトが存在することが示されます。

結果の重要度

処置必須

修正あり

いいえ

有効範囲

UNIVERSAL

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

説明

このチェックでは、XMLスキーマ・オブジェクトがソース・データベースにあることが示されます。これらのオブジェクトは移行されません。

影響

XMLスキーマは、Oracle Autonomous Databaseではサポートされていません。

処置

XMLスキーマ・オブジェクトを使用しないようにアプリケーションを変更します。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.73 has_xmltype_tables

移行前アドバイザ・ツールのチェック`has_xmltype_tables`では、ソース・データベースに`XMLType`表が存在することが示されます。

結果の重要度

処置必須

修正あり

いいえ

有効範囲

`SCHEMA`

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

説明

このチェックでは、ソース・データベースにXMLType表があることが示されます。これらの表は、`STORAGE_TYPE`が`BINARY`でないかぎり移行されません。

影響

`BINARY`として格納されていない`XMLType`表に依存するアプリケーションは失敗します。

処置

`CLOB`またはオブジェクト・リレーショナル・ストレージのある`XMLType`表は、Oracle Autonomous Databaseではサポートされていません。`XMLType`ストレージ・オプションを`BINARY`に変更します。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.74 modified_db_parameters_dedicated

移行前アドバイザ・ツールのチェック`modified_db_parameters_dedicated`では、制限付きの初期化パラメータが変更されることが示されます。

結果の重要度

確認推奨

修正あり

いいえ

有効範囲

`INSTANCE`

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ATPD Autonomous Transaction Processing (専用)

説明

このチェックでは、Oracle Autonomous Database (専用インフラストラクチャ)で変更できないOracle Databaseパラメータがソース・データベース・インスタンスにあることが示されます。

影響

データベースでは変更されているが、Oracle Autonomous Databaseでは変更できない初期化パラメータのリストが提供されます。

処置

どのパラメータが変更可能かを理解するには、Oracle Autonomous Databaseのドキュメントを参照してください。

関連トピック

- 変更可能な初期化パラメータのリスト

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.75 modified_db_parameters_serverless

移行前アドバイザ・ツールのチェック`modified_db_parameters_serverless`では、制限付きの初期化パラメータが変更されることが示されます。

結果の重要度

確認推奨

修正あり

いいえ

有効範囲

`INSTANCE`

ターゲット・クラウド

これはデフォルトのチェックです。これは次に適用されます:

- ADWS Autonomous Data Warehouse (共有)
- ATPS Autonomous Transaction Processing (共有)

説明

このチェックでは、Oracle Autonomous Database (共有インフラストラクチャ)で変更できないOracle Databaseパラメータがソース・データベース・インスタンスにあることが示されます。

影響

データベースでは変更されているが、Oracle Autonomous Databaseでは変更できない初期化パラメータのリストが提供されます。

処置

どのパラメータが変更可能かを理解するには、Oracle Autonomous Databaseのドキュメントを参照してください。

関連トピック

- 変更可能な初期化パラメータのリスト

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.76 nls_character_set_conversion

移行前アドバイザ・ツールのチェック`nls_character_set_conversion`では、ソース・データベースにOracle Autonomous Databaseでは無効な文字コードが存在することが示されます。

結果の重要度

ランタイム

修正あり

いいえ

有効範囲

`SCHEMA`

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)
- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

説明

このチェックでは、ソースからターゲットへのデータベース文字セットの文字データ変換によって発生した問題(列の長さを超える文字値の拡張、無効な文字コードの損失など)が警告されます。

影響

移行中、ORA-1401、またはソースからターゲットへのデータベース文字セット変換による無効な文字コードの損失が発生する可能性があります。

処置

必要に応じて問題を修正します。考えられる解決策は次のとおりです:

- Database Migration Assistant for Unicode (DMU)を使用して、移行するスキーマをスキャンし、考えられる変換可能性の問題をすべて分析します
- ソース・インスタンスと同じ文字セットを使用して、新しいターゲット・インスタンスを作成します。データベース・インスタンスの作成時に文字セットを選択する方法の詳細は、Oracle Cloud Infrastructureのドキュメントを参照してください。

データベース・インスタンスの作成時に文字セットを選択する方法の詳細は、Oracle Cloud Infrastructureのドキュメントを参照してください。

ノート:

デフォルトのデータベース文字セットには、AL32UTF8を使用することをお薦めします

関連トピック

- The Database Migration Assistant for Unicode (DMU) Tool (ドキュメントID 1272374.1)

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.77 nls_national_character_set

移行前アドバイザ・ツールのチェックnls_national_character_setでは、NCHARとNVARCHAR2の長さがソース・データベースとターゲット・データベースで異なることが示されます。

結果の重要度

確認必須

修正あり

いいえ

有効範囲

UNIVERSAL

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)
- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

説明

このチェックでは、ソース・データベースとターゲット・データベースでNCHARとNVARCHAR2の長さが異なることが示されます。

ソースからターゲットへの各国語文字セットの文字データ変換によって発生した問題(データ型の制限を超える文字値の拡張、無効な文字コードの損失など)がないかどうかを確認してください。

影響

移行中、ORA-01401、またはソースからターゲットへの各国語文字セット変換による無効な文字コードの損失が発生する可能性があります。

処置

可能であれば、ソース・データベースと同じ各国語文字セットでOracle Cloud Infrastructureのターゲット・データベースをプロビジョニングし、ターゲット・クラウド・データベースで拡張データ型を有効にします。データベース・インスタンスの作成時に各国語文字セットを選択する方法の詳細は、Oracle Cloud Infrastructureのドキュメントを参照してください。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.78 nls_nchar_ora_910

移行前アドバイザ・ツールのチェックnls_nchar_ora_910では、NCHARとNVARCHAR2の長さがターゲット・データベースでの最大長を超えていることが示されます。

結果の重要度

処置必須

修正あり

いいえ

有効範囲

SCHEMA

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)
- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

説明

このチェックでは、NCHARとNVARCHAR2の長さがターゲット・データベースでの最大長を超えていることが示されます。

ターゲット・データベースの各国語データベース文字セット列の最大長を決定し、文字長がターゲット・データベースの制限を超えるNCHARおよびNVARCHAR2列をソース・データベースで確認します。

影響

移行中、ソース・データベースとターゲット・データベース間でNCHARおよびNVARCHAR2列の最大文字長が異なるため、ORA-00910エラーが発生する可能性があります。

処置

可能であれば、ソース・データベースと同じ各国語文字セットでOracle Cloud Infrastructureのターゲット・データベースをプロビジョニングし、ターゲット・クラウド・データベースで拡張データ型を有効にします。データベース・インスタンスの作成時に各国語文字セットを選択する方法の詳細は、Oracle Cloud Infrastructureのドキュメントを参照してください。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.79 options_in_use_not_available_dedicated

移行前アドバイザ・ツールのチェック`options_in_use_not_available_dedicated`では、ソース・データベースに使用できないデータベース・オプションが存在することが示されます。

結果の重要度

確認推奨

修正あり

いいえ

有効範囲

`INSTANCE`

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ATPD Autonomous Transaction Processing (専用)

説明

ソースで使用されているが、Oracle Autonomous Database (専用インフラストラクチャ)では使用できないデータベース・オプションのリストを生成します。

影響

移行するデータベースに、Oracle Autonomous Databaseで使用できないオプションを使用するアプリケーションまたはスキーマがある場合は、移行後にこれらのアプリケーションが動作しない可能性があります。

処置

ソース・データベースのアプリケーションまたはスキーマが、Oracle Autonomous Database (専用インフラストラクチャ)でサポートされていないオプションに依存していることを確認し、それに応じて計画します。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.80 options_in_use_not_available_serverless

移行前アドバイザ・ツールのチェック`options_in_use_not_available_serverless`では、ソース・データベースに使用できないデータベース・オプションが存在することが示されます。

結果の重要度

確認推奨

修正あり

いいえ

有効範囲

`INSTANCE`

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ATPD Autonomous Transaction Processing (専用)

説明

ソースで使用されているが、Oracle Autonomous Database (共有インフラストラクチャ)では使用できないデータベース・オプションのリストを生成します。

影響

移行するデータベースに、Oracle Autonomous Databaseで使用できないオプションを使用するアプリケーションまたはスキーマがある場合は、移行後にこれらのアプリケーションが動作しない可能性があります。

処置

ソース・データベースのアプリケーションまたはスキーマが、Oracle Autonomous Database (共有インフラストラクチャ)でサポートされていないオプションに依存していることを確認し、それに応じて計画します。

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.81 standard_traditional_audit_adb

移行前アドバイザ・ツールのチェック`standard_traditional_audit_adb`では、従来の監査構成がデータベースで検出されたことが示されます。

結果の重要度

確認推奨

修正あり

いいえ

有効範囲

`INSTANCE`

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

説明

Oracle Database 21cで非推奨となった従来の監査は、Oracle Database 23c以降ではサポートされなくなりました。従来の監査構成がこのデータベースで検出されました。

影響

Oracle Database 23cでは、従来の監査はサポートされなくなりました。統合監査の使用を開始することをお薦めします。

処置

従来の監査構成を削除してください。支援が必要な場合は、Oracle SupportのドキュメントID 2909718.1の手順を使用してください。

関連トピック

- Traditional to Unified Audit Syntax Converter - Generate Unified Audit Policies from Current

Traditional Audit Configuration (ドキュメントID 2909718.1)

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.82 standard_traditional_audit_default

移行前アドバイザ・ツールのチェック`standard_traditional_audit_default`では、従来の監査構成がデータベースで検出されたことが示されます。

結果の重要度

確認推奨

修正あり

いいえ

有効範囲

`INSTANCE`

ターゲット・クラウド

- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

説明

Oracle Database 21cで非推奨となった従来の監査は、Oracle Database 23c以降ではサポートされなくなりました。従来の監査構成がこのデータベースで検出されました。

影響

Oracle Database 23cでは、従来の監査はサポートされなくなりました。統合監査の使用を開始することをお薦めします。

処置

Oracle SupportドキュメントID 2909718.1に記載されている手順を使用して、従来の監査構成を削除してください。次の`init.ora`のパラメータ値が`CDB$ROOT`に設定されていることを確認し、データベースを再起動します:

```
AUDIT_TRAIL=none
AUDIT_SYS_OPERATIONS=false
```

関連トピック

- Traditional to Unified Audit Syntax Converter - Generate Unified Audit Policies from Current Traditional Audit Configuration (ドキュメントID 2909718.1)

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.83 timezone_table_compatibility_higher_dedicated

移行前アドバイザ・ツールのチェック`timezone_table_compatibility_higher_dedicated`では、ソースのタイムゾーン設定がターゲット・データベースより最新のバージョンであることが示されます。

結果の重要度

ランタイム

修正あり

いいえ

有効範囲

UNIVERSAL

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ATPD Autonomous Transaction Processing (専用)

説明

ソース・データベースのTZ_VERSIONは、ターゲットのTZ_VERSIONより高くできません。

影響

ターゲットのTZ_VERSIONがソース・データベースのTZ_VERSION以上になるまで、移行はできません。

処置

四半期メンテナンス更新の「メンテナンスのスケジュール」ダイアログのタイムゾーン更新の有効化オプションを使用して、ターゲット・インスタンスのタイムゾーン・バージョンを更新します。

関連トピック

- 四半期メンテナンス更新のスケジュール

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.84 timezone_table_compatibility_higher_default

移行前アドバイザ・ツールのチェックtimezone_table_compatibility_higher_defaultでは、ソースのタイムゾーン設定がターゲット・データベースより最新のバージョンであることが示されます。

結果の重要度

ランタイム

修正あり

いいえ

有効範囲

UNIVERSAL

ターゲット・クラウド

- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

説明

ソース・データベースのTZ_VERSIONは、ターゲットのTZ_VERSIONより高くできません。

影響

ターゲットのTZ_VERSIONがソース・データベースのTZ_VERSION以上になるまで、移行はできません。

処置

OracleサポートのドキュメントID 412160.1から適切なパッチをダウンロードしてインストールし、ターゲット・インスタンスのタイム

ゾーン・バージョンがソース・インスタンス以上であることを確認してください

関連トピック

- Primary Note DST FAQ : Updated DST Transitions and New Time Zones in Oracle RDBMS and OJVM Time Zone File Patches (ドキュメントID 412160.1)

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.85 timezone_table_compatibility_higher_serverless

移行前アドバイザ・ツールのチェック`timezone_table_compatibility_higher_serverless`では、ソースのタイムゾーン設定がターゲット・データベースより最新のバージョンであることが示されます。

結果の重要度

ランタイム

修正あり

いいえ

有効範囲

`UNIVERSAL`

ターゲット・クラウド

- ADWS Autonomous Data Warehouse (共有)
- ATPS Autonomous Transaction Processing (共有)

説明

ソース・データベースの`TZ_VERSION`は、ターゲットの`TZ_VERSION`より高くできません。

影響

ターゲットの`TZ_VERSION`がソース・データベースの`TZ_VERSION`以上になるまで、移行はできません。

処置

タイムゾーン・ファイルのバージョンを更新します。Autonomous Databaseでのタイムゾーン・ファイルのバージョンの管理を参照してください。

関連トピック

- Autonomous Databaseでのタイムゾーン・ファイルのバージョンの管理

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.86 unified_and_standard_traditional_audit_adb

移行前アドバイザ・ツールのチェック`unified_and_standard_traditional_audit_adb`では、従来の監査構成がデータベースで検出されたことが示されます。

結果の重要度

ランタイム

修正あり

いいえ

有効範囲

INSTANCE

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)

説明

Oracle Database 21cで非推奨となった従来の監査は、Oracle Database 23c以降ではサポートされなくなりました。このデータベースは、統合監査のみを使用するように構成されていますが、従来の監査構成が検出されました。

影響

データベース内の従来の監査構成を削除しないと、パフォーマンスが低下する可能性があります。

処置

従来の監査構成を削除することをお薦めします

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.87 unified_and_standard_traditional_audit_default

移行前アドバイザ・ツールのチェックunified_and_standard_traditional_audit_defaultでは、従来の監査構成がデータベースで検出されたことが示されます。

結果の重要度

ランタイム

修正あり

いいえ

有効範囲

INSTANCE

ターゲット・クラウド

- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

説明

Oracle Database 21cで非推奨となった従来の監査は、Oracle Database 23c以降ではサポートされなくなりました。このデータベースは、統合監査のみを使用するように構成されていますが、従来の監査構成が検出されました。

影響

データベース内の従来の監査構成を削除しないと、パフォーマンスが低下する可能性があります。

処置

Oracle SupportドキュメントID 2909718.1に記載されている手順を使用して、従来の監査構成を削除してください。次の

init.oraのパラメータ値がCDB$ROOTに設定されていることを確認し、データベースを再起動します:

```
AUDIT_TRAIL=none
AUDIT_SYS_OPERATIONS=false
```

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

20.11.88 xdb_resource_view_has_entriesチェック

移行前アドバイザ・ツールのチェックxdb_resource_view_has_entriesチェックでは、Oracle Autonomous DatabaseでサポートされていないXDBリポジトリがあることが示されます。RESOURCE_VIEW内のエントリは移行されません。

結果の重要度

確認必須

修正あり

いいえ

ターゲット・クラウド

- ADWD Autonomous Data Warehouse (専用)
- ADWS Autonomous Data Warehouse (共有)
- ATPD Autonomous Transaction Processing (専用)
- ATPS Autonomous Transaction Processing (共有)
- デフォルト(Oracle Autonomous Database以外のOracle Databaseインスタンス)

説明

このチェックは、Oracle Data PumpおよびOracle GoldenGateでの移行、およびOracle Data Pumpデータベース・リンクの場合にソース・スキーマに適用されます。Oracle Autonomous Database (ADB)でサポートされていないOracle XML DBリポジトリ(XDBリポジトリ)がある場合、RESOURCE_VIEW内のエントリは移行されません。

影響

XDBリポジトリRESOURCE_VIEW内のエントリに依存するアプリケーションは、想定どおりに機能しない可能性があります。

処置

XDBリポジトリの依存性を削除するには、アプリケーションを更新する必要があります。データベースでXDBが使用されているかどうかの判断の詳細は、Oracle SupportのドキュメントID 733667.1を参照してください

関連トピック

- How to Determine if XDB is Being Used in the Database? (ドキュメントID 733667.1)

親トピック: 移行前アドバイザ・ツールによって実行されるチェックのリスト

21 ADRCI: ADRコマンド・インタプリタ

自動診断リポジトリ・コマンド・インタプリタ(ADRCI)ユーティリティは、Oracle Database診断データの管理に使用するコマンドライン・ツールです。

> ノート:
>
> `UIDRVCI.exe` ファイルは、診断データへのアクセスに使用されるため、使用しないでください。

- ADRコマンド・インタプリタ(ADRCI)ユーティリティについて
  自動診断リポジトリ・コマンド・インタプリタ(ADRCI)は、Oracle Database障害診断インフラストラクチャの一部であるコマンドライン・ツールです。
- Oracle Database ADRCの定義
  Oracle Databaseの問題を診断する方法を理解するために、OracleがADRCIとOracle Databaseの障害診断機能インフラストラクチャに使用する用語の定義を学習します。
- ADRCIの起動とヘルプの利用
  対話方式モードまたはバッチ・モードでADRCIを使用できます。
- ADRCIコマンドを使用する前のADRCIホームパスの設定
  問題を診断するときは、複数のデータベース・インスタンスまたはコンポーネントからの診断データを使用する場合や、1つのインスタンスまたはコンポーネントからの診断データを対象とする場合があります。
- アラート・ログの表示
  ACRコマンド・インタプリタのアラート・ログ(ADRCI)を表示するには、この手順を使用してアラート・ログをデフォルト・エディタで表示します。
- トレース・ファイルの検索
  ADRCIにより、現在自動診断リポジトリ(ADR)に存在するトレース・ファイルの名前が表示できます。
- インシデントの表示
  ADRCIの`SHOW INCIDENT`コマンドでは、未解決のOracle Databaseインシデントに関する情報が表示されます。
- インシデントのパッケージ化
  Oracleサポート・サービスに送信して分析を依頼するために、ADRCIコマンドを使用して1つ以上のインシデントをパッケージ化できます。
- ADRCIコマンド・リファレンス
  自動診断リポジトリ・コマンド・インタプリタ(ADRCI)で使用できるコマンドについて説明します。
- ADRCIのトラブルシューティング
  トラブルシューティングを支援するため、一般的なADRCIエラー・メッセージの一部と、考えられる原因および処置を確認します。

親トピック: その他のユーティリティ

21.1 ADRコマンド・インタプリタ(ADRCI)ユーティリティ

自動診断リポジトリ・コマンド・インタプリタ(ADRCI)は、Oracle Databaseの障害診断インフラストラクチャの一部であるコマンドライン・ツールです。

ADRCIユーティリティを使用すると、データベース内の問題(インシデント)の原因を診断できます。これは、問題の根本原因に対処するためにOracleサポートが必要とすることがあるインシデント・パッケージのデータ収集に役立つ場合があります。

ADRCIでは、次のことができます。

- 自動診断リポジトリ(ADR)内の診断データを表示します。
- 状態モニターのレポートを表示します。
- Oracleサポート・サービスへ送信するために、インシデントや問題の情報をZIPファイルにパッケージ化します。

診断データには、インシデントおよび問題についての説明、トレース・ファイル、ダンプ、状態モニターのレポート、アラート・ログ・エントリなどが含まれます。

ADRデータは、ADRディレクトリのオペレーティング・システム権限により保護されているため、ADRCIにログインする必要はありません。

ADRCIには豊富なコマンド・セットがあります。これらのコマンドは、対話型モードまたはスクリプト内で使用できます。

> ノート:
>
> 診断データを管理する場合の簡単で推奨される方法は、Oracle Enterprise Manager サポート・ワークベンチ(サポート・ワークベンチ)を使用する方法です。ADRCI では、サポート・ワークベンチの機能の大部分にかわるコマンドライン機能が備わり、トレース・ファイルのリスト表示や問合せなどの機能が追加されています。
>
> Oracle Database の障害診断インフラストラクチャの詳細は、*Oracle Database* 管理者ガイドを参照してください。

関連トピック

- Oracle Database管理者ガイド 問題の診断と解決

親トピック: ADRCI: ADRコマンド・インタプリタ

21.2 Oracle Database ADRCの定義

Oracle Databaseの問題を診断する方法を理解するために、OracleがADRCIとOracle Databaseの障害診断機能インフラストラクチャに使用する用語の定義を学習します。

次の用語は、Oracle Databaseの自動診断リポジトリ・インシデント障害診断インフラストラクチャ(ADRCI)とOracle Database障害診断インフラストラクチャに関連付けられています。

自動診断リポジトリ(ADR)

自動診断リポジトリ(ADR)は、データベース診断データ(トレース、ダンプ、アラート・ログ、状態モニターのレポートなど)のファイルベース・リポジトリです。複数のインスタンスや製品にまたがる一元化されたディレクトリ構造を持っています。リリースOracle Database 11g以降では、データベース、Oracle Automatic Storage Management (Oracle ASM)およびその他のOracle Database製品やコンポーネントは、すべての診断データをADRに格納します。各製品のインスタンスはそれぞれ、診断データを独自のADRホーム・ディレクトリの下に配置します。たとえば、共有記憶域とASMを使用するOracle Real Application Clusters(Oracle RAC)環境では、各データベース・インスタンスおよび各Oracle ASMインスタンスがADR内にホーム・ディレクトリを持ちます。ADRの統一されたディレクトリ構造を使用することで、ユーザーおよびOracleサポート・サービスによる、複数のインスタンスおよび複数の製品間での診断データを関連付けた分析が可能になります。

問題

問題とは、データベースで発生するクリティカル・エラーのことです。クリティカル・エラーには、ORA-00600などの内部エラーや、

ORA-07445(オペレーティング・システムの例外)、ORA-04031(共有プールのメモリー不足)などのその他の重大なエラーが含まれます。問題はADRで追跡されます。各問題には、**問題キー**と一意の**問題ID**が割り当てられます。

インシデント

インシデントは、問題が1回発生したことを表します。問題が複数回発生した場合は、発生ごとにインシデントが作成されます。インシデントはADR内で追跡されます。各インシデントは、数値型のインシデントIDによって識別されます。このIDはADR内で一意です。インシデントが発生すると、データベースによりアラート・ログ内にエントリが作成され、Oracle Enterprise Managerに**インシデント・アラート**が送信されます。次に、インシデントに関する診断データがダンプ・ファイルの形式(インシデント・ダンプ)で収集され、**インシデントID**を使用してインシデント・ダンプにタグが付けられ、インシデント・ダンプがそのインシデント用に作成されたADRサブディレクトリに格納されます。

通常、クリティカル・エラーの診断と解決は、インシデント・アラートから開始されます。ADR内のすべてのインシデントのリストは、ADRCIコマンドを使用して取得できます。各インシデントは1つの問題にのみマップされます。

インシデントは**フラッド制御**,されているため、1つの問題が生成するインシデントやインシデント・ダンプが多すぎることはありません。

問題キー

すべての問題には**問題キー**が割り当てられています。問題キーはエラー・コード(ORA-600など)を含むテキスト文字列で、1つ以上のエラー・パラメータを含むことがあります。2つのインシデントは、その2つの問題キーが一致した場合に発生原因が同じとみなされます。

インシデント・パッケージ

インシデント・パッケージ(パッケージ)は、1つ以上の問題に対するインシデント・データの集合です。Oracleサポート・サービスにインシデント・データを送信する前に、**インシデント・パッケージング・サービス(IPS)**を使用して、パッケージにデータを収集する必要があります。パッケージを作成した後は、パッケージに外部ファイルを追加したり、パッケージ内からファイルを選択して削除することや、パッケージ内で選択したファイルを**スクラブ**(編集)して機密データを削除することも可能です。

パッケージは、パッケージの内容から物理ファイルを作成するまでは論理構成のみです。つまり、インシデント・パッケージは、ADR内のメタデータの集合として開始されます。パッケージの内容を追加および削除する場合は、メタデータのみが変更されます。Oracleサポート・サービスへデータをアップロードする準備ができたら、データをZIPファイルに保存するADRCIを使用することで物理パッケージを作成します。次に、ZIPファイルをOracleサポートにアップロードできます。

ファイナライズ

ADRCIで論理パッケージから物理パッケージを生成する前に、パッケージをファイナライズする必要があります。これにより、他のコンポーネントが呼び出され、相関診断データ・ファイルがこのパッケージにすでに存在するインシデントに追加されます。また、ファイナライズすることで、最新のトレース・ファイル、アラート・ログ・エントリ、状態モニター・レポート、SQLテスト・ケースおよび構成情報も追加されます。このステップは、物理パッケージが生成されると自動的に実行されますが、ADRCIユーティリティを使用して手動で実行することもできます。パッケージを手動でファイナライズした後、追加されたファイルを確認し、機密情報を含むファイルを削除または編集することができます。

ADRホーム

ADRホームは、特定のオラクル社の製品またはコンポーネントの特定のインスタンスに対するすべての診断データ(トレース、ダンプ、アラート・ログなど)のルート・ディレクトリです。たとえば、Oracle ASMを使用するOracle RAC環境では、各データベース・インスタンスと各Oracle ASMインスタンスにADRホームがあります。すべてのADRホームは、同じ階層ディレクトリ構造を共有します。各ADRホームの標準的なサブディレクトリには、アラート(アラート・ログ)、トレース(トレース・ファイル)およびインシデント(インシデント情報)を含むものもあります。すべてのADRホームはADRベース・ディレクトリ内に配置されます。

一部のADRCIコマンドは、同時に複数のADRホームでの動作が可能です。カレントADRCI**ホームパス**により、ADRCIコマン

ドの発行時に診断データを検索するADRホームが決定されます。

ADRベース

複数のADRホーム間で診断データの相関付けを行うことができるようにするために、ADRホームは**ADRベース**と呼ばれる同じルート・ディレクトリ下でグループ化されます。たとえば、Oracle RAC環境では、ADRベースが共有ディスク上に存在し、各Oracle RACインスタンスのADRホームがこのADRベース下に配置される場合があります。

データベース・インスタンスのADRベースの場所は、DIAGNOSTIC_DEST初期化パラメータによって設定されます。このパラメータが指定されない場合またはNULLである場合は、データベースではこのパラメータがデフォルト値に設定されます。

複数のデータベース・インスタンスがOracleホームを共有しているときは、これらのインスタンスが複数の単一インスタンスであるかOracle RACデータベースの複数のインスタンスであるかどうかに関係なく、1つ以上のこれらのインスタンスがADRベースを異なる場所に設定している場合、最後に起動するインスタンスがADRCIのデフォルトADRベースを決定します。

ホームパス

すべてのADRCIコマンドは、**カレント**ADRホーム内の診断データに対して動作します。常に複数のADRホームをカレントにできます。一部のADRCIコマンド(SHOW INCIDENTなど)は、すべてのカレントADRホームから診断データを検索して表示します。その他のコマンドは、1つのADRホームをとカレントとし、カレントADRホームが複数存在している場合はエラー・メッセージを表示します。

ADRCI**ホームパス**により、カレントであるADRホームが決定されます。これは、ADRベース階層内のディレクトリを指定することによって決定されます。1つのADRホーム・ディレクトリが指定されている場合、そのADRホームが唯一のカレントADRホームとなります。ホームパスが階層内のADRホーム・ディレクトリ・レベルより上位のディレクトリを示している場合は、そのディレクトリより下位のすべてのADRホームがカレントになります。

デフォルトでは、ADRCI起動時のホームパスはNULLです。つまり、ADRベース下のADRホームはすべてカレントになります。

SHOW HOMEおよびSHOW HOMEPATHの各コマンドは、カレントであるADRホームのリストを表示し、SET HOMEPATHコマンドはホームパスを設定します。

関連トピック

- 『Oracle Database管理者ガイド』インシデントと問題についてに関する項
- 『Oracle Database管理者ガイド』インシデント・パッケージの相関診断データについてに関する項

親トピック: ADRCI: ADRコマンド・インタプリタ

21.3 ADRCIの起動とヘルプの利用

ADRCIは対話方式モードまたはバッチ・モードで使用できます。

詳細は次の項を参照してください。

- 対話方式モードでのADRCIの使用方法
  ADRCIを対話方式モードで使用してOracle Databaseのインシデントを診断する場合、一度に1つのコマンドを個別に入力するようにプロンプトが表示されます。
- ヘルプの表示
  ADRコマンド・インタプリタ(ADRCI)ユーティリティの使用時にヘルプを取得する方法について説明します。
- バッチ・モードでのADRCIの使用方法
  バッチ・モードを使用すると、入力を求めるプロンプトが表示されることなく、スクリプトまたはバッチ・ファイルを使用して一連のADRCIコマンドを一度に実行できます。

親トピック: ADRCI: ADRコマンド・インタプリタ

21.3.1 対話方式モードでのADRCIの使用方法

ADRCIを対話方式モードで使用してOracle Databaseのインシデントを診断する場合、一度に1つのコマンドを個別に入力するようにプロンプトが表示されます。

1. ORACLE_HOMEおよびPATH環境変数が適切に設定されていることを確認します。

   Microsoft Windowsプラットフォームの場合、これらの環境変数はインストール時に自動的にWindowsレジストリに設定されます。その他のプラットフォームの場合は、オペレーティング・システムのコマンドを使用して環境変数を設定および確認する必要があります。

   PATH環境変数には、Oracle_home/binが含まれている必要があります

   。

2. オペレーティング・システムのコマンド・プロンプトで、次のコマンドを入力します。

   ```
   ADRCI
   ```

   ユーティリティが起動し、次のプロンプトが表示されます。

   ```
   adrci>
   ```

3. ADRCIコマンドを入力します。各コマンドの入力後に[Enter]キーを押します。
4. ADRCIを終了するには、次のいずれかのコマンドを入力します。

   ```
   EXIT
   QUIT
   ```

親トピック: ADRCIの起動とヘルプの利用

21.3.2 ヘルプの表示

ADRコマンド・インタプリタ(ADRCI)ユーティリティの使用時にヘルプを取得する方法について説明します。

ADRCIヘルプ・システムを使用すると、次の項目を実行できます。

- ADRコマンドのリストの表示
- 個々のコマンドのヘルプの表示
- ADRCIコマンドライン・オプションのリストの表示

ADRCIコマンドのリストを表示するには

1. 対話方式モードでADRCIを起動します。
2. ADRCIプロンプトで、次のコマンドを入力します。

   ```
   HELP
   ```

特定のADRCIコマンドのヘルプを利用するには

1. 対話方式モードでADRCIを起動します。
2. ADRCIプロンプトで、次のコマンドを入力します。commandは、詳細情報が必要なADRCIコマンドです。

   ```
   HELP command
   ```

たとえば、SHOW TRACEFILEコマンドに関するヘルプを利用するには、次のように入力します。

```
HELP SHOW TRACEFILE
```

コマンドライン・オプションのリストを表示するには

- オペレーティング・システムのコマンド・プロンプトで、次のコマンドを入力します。

```
ADRCI -HELP
```

ユーティリティによって、次のような出力が示されます。

```
Syntax:
   adrci [-help] [script=script_filename] [exec="command [;command;...]"]

Options      Description                     (Default)
----------------------------------------------------------------
script       script file name                (None)
help         help on the command options     (None)
exec         exec a set of commands          (None)
----------------------------------------------------------------
```

関連トピック

- 対話方式モードでのADRCIの使用方法

親トピック: ADRCIの起動とヘルプの利用

21.3.3 バッチ・モードでのADRCIの使用方法

バッチ・モードを使用すると、入力を求めるプロンプトが表示されることなく、スクリプトまたはバッチ・ファイルを使用して一連のADRCIコマンドを実行できます。

バッチ・モードを使用するには、ADRCIの起動時にADRCIコマンドにコマンドライン・パラメータを追加します。バッチ・モードでは、シェル・スクリプトまたはMicrosoft Windowsバッチ・ファイルにADRCIコマンドを含めることができます。対話方式モードと同様、ADRCIを起動する前に、ORACLE_HOMEおよびPATH環境変数を設定する必要があります。

バッチ操作用のADRCIのコマンドライン・パラメータ

次のコマンドライン・パラメータは、バッチ操作に使用できます。

表21-1 ADRCIバッチ操作パラメータ

パラメータ	説明
EXEC	ADRCI を起動するオペレーティング・システム・コマンドラインで、1 つ以上の ADRCI コマンドを発行できます。コマンドが複数ある場合は、セミコロン(;)で区切ります。
SCRIPT	ADRCI コマンドを含むスクリプトを実行できます。

コマンドラインでADRCIコマンドを発行する方法

- オペレーティング・システムのコマンド・プロンプトで、次のコマンドを入力します。

```
ADRCI EXEC="COMMAND[; COMMAND]..."
```

たとえば、SHOW HOMESコマンドをバッチ・モードで実行するには、オペレーティング・システムのコマンド・プロンプトで次

のコマンドを入力します。

```
ADRCI EXEC="SHOW HOMES"
```

SHOW HOMESコマンドを実行してから、SHOW INCIDENTコマンドを実行するには、次のように入力します。

```
ADRCI EXEC="SHOW HOMES; SHOW INCIDENT"
```

ADRCIスクリプトの実行方法:

オペレーティング・システムのコマンド・プロンプトで、次のコマンドを入力します。

```
ADRCI SCRIPT=SCRIPT_FILE_NAME
```

たとえば、adrci_script.txtというスクリプト・ファイルを実行するには、オペレーティング・システムのコマンド・プロンプトで次のコマンドを入力します。

```
ADRCI SCRIPT=adrci_script.txt
```

スクリプト・ファイルには、セミコロン(;)または改行で区切られた一連のコマンドが含まれています。たとえば:

```
SET HOMEPATH diag/rdbms/orcl/orcl; SHOW ALERT -term
```

親トピック: ADRCIの起動とヘルプの利用

21.4 ADRCIコマンドを使用する前のADRCIホームパスの設定

問題を診断するときは、複数のデータベース・インスタンスまたはコンポーネントからの診断データを使用する場合や、1つのインスタンスまたはコンポーネントからの診断データを対象とする場合があります。

複数のインスタンスまたはコンポーネントからの診断データを使用するには、これらのインスタンスまたはコンポーネントすべてのADRホームがカレントであることを確認する必要があります。1つのインスタンスまたはコンポーネントからの診断データを使用するには、そのインスタンスまたはコンポーネントのADRホームのみがカレントであることを確認する必要があります。ADRCIホームパスを設定してカレントのADRホームを制御します。

複数のホームがカレントになっている場合は、ADRディレクトリ構造におけるホームパスは、直下に複数のADRホーム・ディレクトリを含むディレクトリを示しています。1つのADRホームを対象とするには、ホームパスをディレクトリ階層の下部にある1つのADRホーム・ディレクトリを示すように設定する必要があります。

たとえば、orclbiという名前のOracle RACデータベースが、SID orclbi1とorclbi2が指定された2つのインスタンスを持ち、Oracle RACが共有Oracleホームを使用している場合は、次の2つのADRホームが存在します。

```
/diag/rdbms/orclbi/orclbi1/
/diag/rdbms/orclbi/orclbi2/
```

すべてのADRCIコマンドおよび出力では、ADRホーム・ディレクトリ・パス(ADRホーム)は常にADRベースに対して相対的に示されます。したがって、ADRベースが現在/u01/app/oracleである場合、これら2つのADRホームの絶対パスは次のようになります。

```
/u01/app/oracle/diag/rdbms/orclbi/orclbi1/
/u01/app/oracle/diag/rdbms/orclbi/orclbi2/
```

SET HOMEPATHコマンドを使用して、1つ以上のADRホームをカレントに設定します。ADRベースが/u01/app/oracleである場合に、ホームパスを/u01/app/oracle/diag/rdbms/orclbi/orclbi2/に設定するには、このコマンドを次のように使用します。

```
adrci> set homepath diag/rdbms/orclbi/orclbi2
```

ADRCIの起動時には、ホームパスはデフォルトでNULLになります。つまり、ADRベース下のすべてのADRホームがカレントとなります。前述の例では、両方のOracle RACインスタンスのADRホームがカレントになります。

```
adrci> show homes
ADR Homes:
diag/rdbms/orclbi/orclbi1
diag/rdbms/orclbi/orclbi2
```

この場合、実行するADRCIコマンドはいずれも、複数のカレントADRホームをコマンドがサポートしていると想定して、両方のADRホームからの診断データを処理します。ホームパスを/diag/rdbms/orclbi/orclbi2に設定すると、orclbi2のSIDを持つインスタンスのADRホームのみがカレントになります。

```
adrci> set homepath diag/rdbms/orclbi/orclbi2
adrci> show homes
ADR Homes:
diag/rdbms/orclbi/orclbi2
```

この場合、実行するADRCIコマンドはいずれも、この1つのADRホームからの診断データのみを使用します。

関連項目:

- ADRホームの構造の詳細は、『Oracle Database管理者ガイド』を参照してください。
- ADRベース
- ADRホーム
- ホームパス
- SET HOMEPATH
- SHOW HOMES

親トピック: ADRCI: ADRコマンド・インタプリタ

21.5 アラート・ログの表示

ACRコマンド・インタプリタのアラート・ログ(ADRCI)を表示するには、この手順を使用してアラート・ログをデフォルト・エディタで表示します。

アラート・ログは、XML形式ファイルとテキスト・ファイルの両方として書き出されます。任意のテキスト・エディタでいずれかの形式のファイルを表示するか、ADRCIコマンドを実行してXMLタグを省略したXML形式のアラート・ログを表示することができます。

デフォルトでは、ADRCIはデフォルトのエディタにアラート・ログを表示します。SET EDITERコマンドを使用すると、デフォルトのエディタを変更できます。

ADRCIでアラート・ログを表示するには:

1. 対話方式モードでADRCIを起動します。
2. (オプション)SET HOMEPATHコマンドを使用して、ADRホームを1つ選択します(カレントにします)。

   最初にSHOW HOMESコマンドを使用すると、カレントのADRホームのリストを表示できます。詳細は、「ホームパス」および「ADRCIコマンドを使用する前のADRCIホームパスの設定」を参照してください。

3. ADRCIプロンプトで、次のコマンドを入力します。

```
SHOW ALERT
```

複数のADRホームがカレントの場合は、リストから1つのADRホームを選択するように要求されます。デフォルトのエディタに、XMLタグを省略したアラート・ログが表示されます。

4. エディタを終了し、ADRCIコマンド・プロンプトに戻ります。

SHOW ALERTコマンドのバリエーションを次に示します。

```
SHOW ALERT -TAIL
```

これは、ターミナル・セッションでのアラート・ログの最後の部分(最後の10エントリ)を表示します。

```
SHOW ALERT -TAIL 50
```

これは、ターミナル・セッションでのアラート・ログの最後の50エントリを表示します。

```
SHOW ALERT -TAIL -F
```

これは、アラート・ログの最後の10エントリを表示した後、アラート・ログに到着する追加メッセージを待機します。各メッセージは、到着時に表示に追加されます。このコマンドにより、アラート・ログのライブ監視を実行できます。待機を停止してADRCIプロンプトに戻るには、[Ctrl]を押しながら[C]を押します。

```
SPOOL /home/steve/MYALERT.LOG
SHOW ALERT -TERM
SPOOL OFF
```

これは、XMLタグのないアラート・ログを/home/steve/MYALERT.LOGファイルに出力します。

```
SHOW ALERT -P "MESSAGE_TEXT LIKE '%ORA-600%'"
```

これは、文字列ORA-600を含むアラート・ログ・メッセージのみを表示します。出力は次のようになります。

```
ADR Home = /u01/app/oracle/product/11.1.0/db_1/log/diag/rdbms/orclbi/orclbi:
*************************************************************************
01-SEP-06 09.17.44.849000000 PM -07:00
AlertMsg1: ORA-600 dbgris01, addr=0xa9876541
```

関連トピック

- SHOW ALERT

関連項目:

- SHOW ALERT
- SET EDITOR
- Oracle Enterprise Managerまたはテキスト・エディタを使用してアラート・ログを表示する方法については、『Oracle Database管理者ガイド』を参照してください。

親トピック: ADRCI: ADRコマンド・インタプリタ

21.6 トレース・ファイルの検索

ADRCIにより、現在自動診断リポジトリ(ADR)に存在するトレース・ファイルの名前が表示できます。

ADR内に存在するすべてのトレース・ファイルの名前を表示することも、フィルタを適用して名前のサブセットを表示することもでき

ます。たとえば、ADRCIのコマンドで、次の操作を行うことができます。

- ファイル名が検索文字列と一致するトレース・ファイルのリストの取得
- 特定のディレクトリ内に存在するトレース・ファイルのリストの取得
- 特定のインシデントに関連するトレース・ファイルのリストの取得

適切なコマンドライン・パラメータを使用して、フィルタ処理機能を組み合せることができます。

SHOW TRACEFILEコマンドは、カレントADRホーム下のトレース・ディレクトリおよびインシデント・ディレクトリに存在するトレース・ファイルのリストを表示します。複数のADRホームがカレントの場合、すべてのADRホームからのトレース・ファイル・リストが順次出力されます。

次の文は、フィルタ処理を行わずに、カレントADRホームに存在するすべてのトレース・ファイルの名前を表示します。

```
SHOW TRACEFILE
```

次の文は、ファイル名に文字列mmonが含まれるすべてのトレース・ファイルの名前を表示します。パーセント符号(%)はワイルドカード文字として使用されます。また、検索文字列では大/小文字が区別されます。

```
SHOW TRACEFILE %mmon%
```

この文は、/home/steve/tempディレクトリ内でファイル名に文字列mmonを含むすべてのトレース・ファイルの名前を表示します。

```
SHOW TRACEFILE %mmon% -PATH /home/steve/temp
```

この文は、トレース・ファイルの名前を、最終変更時間を逆にたどる順序でリストします。つまり、最近変更されたトレース・ファイルがリストの最初に示されます。

```
SHOW TRACEFILE -RT
```

この文は、インシデント番号1681に関連付けられているすべてのトレース・ファイルの名前を表示します。

```
SHOW TRACEFILE -I 1681
```

関連項目:

- SHOW TRACEFILE
- ADRのディレクトリ構造の詳細は、『Oracle Database管理者ガイド』を参照してください。

親トピック: ADRCI: ADRコマンド・インタプリタ

21.7 インシデントの表示

ADRCIのSHOW INCIDENTコマンドでは、未解決のOracle Databaseインシデントに関する情報が表示されます。

SHOW INCIDENTコマンドを発行すると、ADRCIレポートには、インシデントごとに、インシデントID、問題キーおよびインシデント作成時間が表示されます。階層の場所に複数のカレントADRホームが存在するように**ホームパス**(ADRベース階層内のディレクトリ)を設定した場合、レポートにはすべてのADRホームからのインシデントが含まれます。ホームパスおよびその他のADRCI用語の詳細は、「Oracle Database ADRCの定義」を参照してください。

1. ADRCIを対話方式モードで開始し、ホームパスがADRベース・ディレクトリ階層内で正確なディレクトリを示していることを確認します。

2. ADRCIプロンプトで、次のコマンドを入力します。

```
SHOW INCIDENT
```

ADRCIによって、次のような出力が生成されます。

```
ADR Home = /u01/app/oracle/product/11.1.0/db_1/log/diag/rdbms/orclbi/orclbi:
*************************************************************************
INCIDENT_ID       PROBLEM_KEY                CREATE_TIME
----------------- -------------------------- ---------------------------------
3808              ORA 603                    2010-06-18 21:35:49.322161 -07:00
3807              ORA 600 [4137]             2010-06-18 21:35:47.862114 -07:00
3805              ORA 600 [4136]             2010-06-18 21:35:25.012579 -07:00
3804              ORA 1578                   2010-06-18 21:35:08.483156 -07:00
4 rows fetched
```

SHOW INCIDENTコマンドのバリエーションを次に示します。

```
SHOW INCIDENT -MODE BRIEF
SHOW INCIDENT -MODE DETAIL
```

これらのコマンドは、インシデント・レポートの詳細バージョンを生成します。たとえば、インシデント1681の詳細なインシデント・レポートを表示するには、次のコマンドを入力します。

```
SHOW INCIDENT -MODE DETAIL -P "INCIDENT_ID=1681"
```

関連トピック

- ADRCIコマンド・リファレンス
- Oracle Database ADRCの定義

親トピック: ADRCI: ADRコマンド・インタプリタ

21.8 インシデントのパッケージ化

Oracleサポート・サービスに送信して分析を依頼するために、ADRCIコマンドを使用して1つ以上のインシデントをパッケージ化できます。

詳細は、次の項目を参照してください。

- インシデントのパッケージ化
  ADRコマンド・インタプリタ(ADRCI)インシデントのパッケージ化は、3つのステップからなるプロセスです。
- インシデント・パッケージの作成
  次のトピックでは、インシデント・パッケージの作成方法を説明します。

親トピック: ADRCI: ADRコマンド・インタプリタ

21.8.1 インシデントのパッケージ化

ADRコマンド・インタプリタ(ADRCI)インシデントのパッケージ化は、3つのステップからなるプロセスです。

ステップ1: 論理インシデント・パッケージの作成

インシデント・パッケージ(パッケージ)は、自動診断リポジトリ(ADR)内のメタデータとしてのみ存在するため、論理パッケージとして示されます。インシデント・パッケージは、論理パッケージから物理パッケージを生成するまではコンテンツが含まれません。論理パッケージにはパッケージ番号が割り当てられますので、以降のコマンドではこの番号を使用してパッケージを参照します。

論理パッケージは空のパッケージとして作成することも、インシデント番号、問題番号、問題キーまたは時間間隔に基づいたパッケージとして作成することもできます。パッケージを空のパッケージとして作成する場合は、ステップ2でパッケージに診断情報を追加できます。

インシデントに基づいてパッケージを作成する場合は、そのインシデントの診断データ(ダンプ、状態モニターのレポートなど)が含まれます。問題番号または問題キーに基づいてパッケージを作成する場合は、パッケージにその問題番号または問題キーを参照するインシデントの診断データが含まれます。時間間隔に基づいてパッケージを作成する場合は、その時間間隔内で発生したインシデントに関する診断データが含まれます。

ステップ2: インシデント・パッケージへの診断情報の追加

インシデント番号、問題番号、問題キーまたは時間間隔に基づいて論理パッケージを作成した場合、このステップはオプションとなります。パッケージにインシデントを追加したり、ADR内のファイルをパッケージに追加することができます。空のパッケージを作成した場合は、ADRCIコマンドを使用してパッケージにインシデントまたはファイルを追加する必要があります。

ステップ3: 物理インシデント・パッケージの生成

コマンドを送信して物理パッケージを生成するときに、ADRCIは必要なすべての診断ファイルを収集して、指定したディレクトリ内のZIPファイルに追加します。完全なZIPファイルまたは増分ZIPファイルを生成できます。増分ファイルには、同じ論理パッケージに対してZIPファイルが最後に作成されて以降に追加または変更された診断ファイルすべてが含まれます。増分ファイルは完全ファイルを作成した後にのみ作成でき、必要な数だけ作成できます。各ZIPファイルには順序番号が割り当てられるため、ファイルを正しい順序で分析できます。

ZIPファイルの名前は次の形式に従って指定されます。

```
packageName_mode_sequence.zip
```

説明:

- `packageName`は、問題キーの一部とその後のタイムスタンプで構成されます。
- `mode`は、`COM`(完全)または`INC`(増分)のいずれかです
- `sequence`は、整数です。

たとえば、2006年9月6日午後4時53分に作成した論理パッケージの完全なZIPファイルを生成してから、同じ論理パッケージに対して増分ZIPファイルを生成する場合、次のような名前のファイルを作成します。

```
ORA603_20060906165316_COM_1.zip
ORA603_20060906165316_INC_2.zip
```

親トピック: インシデントのパッケージ化

21.8.2 インシデント・パッケージの作成

次のトピックでは、インシデント・パッケージの作成方法を説明します。

論理インシデント・パッケージ(パッケージ)の作成および物理パッケージの生成に使用するADRCIコマンドについて次に説明します。

- 論理インシデント・パッケージの作成
  `IPS CREATE PACKAGE`コマンドの変形を使用して、論理パッケージ(パッケージ)を作成します。
- 論理インシデント・パッケージへの診断情報の追加
  インシデントをパッケージ化するために既存の論理パッケージ(**パッケージ**)を構成した後、そのパッケージに診断情報を追加できます。

- 物理インシデント・パッケージの生成
  パッケージを生成するときは、既存の論理パッケージに対して物理パッケージ(ZIPファイル)を作成します。

関連項目:

インシデントのパッケージ化

親トピック: インシデントのパッケージ化

21.8.2.1 論理インシデント・パッケージの作成

`IPS CREATE PACKAGE`コマンドの変形を使用して、論理パッケージ(パッケージ)を作成します。

1. ADRCIを対話方式モードで開始し、**ホームパス**(ADRベース階層内のディレクトリ)が、論理パッケージを作成するデータベースのADRベース・ディレクトリ階層内で正確なディレクトリを示していることを確認します。
   ホームパスおよびその他のADRCI用語の詳細は、「Oracle Database ADRCの定義」を参照してください。
2. ADRCIプロンプトで、次のコマンドを入力します。

   ```
   IPS CREATE PACKAGE INCIDENT incident_number
   ```

   たとえば、次のコマンドはインシデント3に基づいてパッケージを作成します。

   ```
   IPS CREATE PACKAGE INCIDENT 3
   ```

   ADRCIによって、次のような出力が生成されます。

   ```
   Created package 10 based on incident id 3, correlation level typical
   ```

   この論理パッケージに割り当てられたパッケージ番号は10です。

`IPS CREATE PACKAGE`コマンドのバリエーションを次に示します。

```
IPS CREATE PACKAGE
```

指定なしでコマンドを入力すると、空のパッケージが作成されます。生成する前にパッケージに診断データを追加するには、`IPS ADD INCIDENT`または`IPS ADD FILE`コマンドを使用する必要があります。

```
IPS CREATE PACKAGE PROBLEM problem_ID
```

このコマンドは、パッケージを作成して、指定した問題IDを参照するインシデントの診断情報を含めます。(問題IDは整数です。)表示されているレポートからインシデントの問題IDを取得するには、`SHOW INCIDENT -MODE BRIEF`コマンドを使用します。同じ問題IDを割り当てられたインシデントが数多く存在する場合もあるため、ADRCIは、90日を超えるインシデントを除き、この問題IDを持つインシデントで最初に発生した3つのインシデント(**早期インシデント**)と最後に発生した3つのインシデント(**最新インシデント**)の診断情報をパッケージに追加します。

> ノート:
>
> 早期インシデントと最新インシデントの数と 90 日の経過日数制限はデフォルト値であり、変更が可能です。IPS SET CONFIGURATION を参照してください。

また、ADRCIによって、すでに追加されているインシデントに時間その他の基準で密接に相関付けられている他のインシデントが追加される場合もあります。

```
IPS CREATE PACKAGE PROBLEMKEY "problem_key"
```

このコマンドは、パッケージを作成して、指定した問題キーを参照するインシデントの診断情報を含めます。表示されているレポートから問題キーを取得するには、SHOW INCIDENTコマンドを使用します。同じ問題キーを割り当てられたインシデントが数多く存在する場合もあるため、ADRCIは、90日を超えるインシデントを除き、この問題キーを持つインシデントで最初に発生した3つの早期インシデントと最後に発生した3つの最新インシデントの診断情報のみをパッケージに追加します。

> ノート:
>
> 早期インシデントと最新インシデントの数と 90 日の経過日数制限はデフォルト値であり、変更が可能です。IPS SET CONFIGURATION を参照してください。

また、ADRCIによって、すでに追加されているインシデントに時間その他の基準で密接に相関付けられている他のインシデントが追加される場合もあります。

問題キーに空白や引用符が含まれている場合、問題キーを一重引用符(')または二重引用符(")で囲む必要があります。

```
IPS CREATE PACKAGE SECONDS sec
```

これは、パッケージを作成し、sec秒前から現在までに発生したすべてのインシデントの診断情報を含みます。secは整数である必要があります。

```
IPS CREATE PACKAGE TIME 'start_time' TO 'end_time'
```

このコマンドは、パッケージを作成し、指定した時間範囲内で発生したすべてのインシデントの診断情報を含めます。start_timeおよびend_timeは、YYYY-MM-DD HH24:MI:SS.FF TZR書式で指定する必要があります。この文字列は、NLS_TIMESTAMP_TZ_FORMAT初期化パラメータに対して有効なフォーマット文字列です。時間の小数(FF)部分はオプションで、HH24:MI:SSのデリミタにはコロンまたはピリオドを使用できます。

たとえば、次のコマンドは、2010年7月24日から7月30日に発生したインシデントを含むパッケージを作成します。

```
IPS CREATE PACKAGE TIME '2010-07-24 00:00:00 -07:00' to '2010-07-30 23.59.59 -07:00'
```

関連トピック

- ADRCIコマンド・リファレンス
- Oracle Database ADRCの定義
- IPS CREATE PACKAGE

親トピック: インシデント・パッケージの作成

21.8.2.2 論理インシデント・パッケージへの診断情報の追加

インシデントをパッケージ化するために既存の論理パッケージ(**パッケージ**)を構成した後、そのパッケージに診断情報を追加できます。

論理パッケージに診断情報を追加すると、パッケージの作成後に次のようなインシデント情報を追加できます。

- 特定のインシデントに関するすべての診断情報
- 自動診断リポジトリ(ADR)内の名前付きファイル

1. ADRCIを対話方式モードで開始し、**ホームパス**(ADRベース階層内のディレクトリ)が、追加する診断情報のADRベース・ディレクトリ階層内で正確なディレクトリを示していることを確認します。

ホームパスおよびその他のADRCI用語の詳細は、「Oracle Database ADRCの定義」を参照してください。

2. ADRCIプロンプトで、追加する診断情報のコマンドを入力します。
   すべての診断情報を追加するには:

```
IPS ADD INCIDENT incident_number PACKAGE package_number
```

   既存のパッケージにADR内のファイルを追加するには:

   - ADRCIプロンプトで、次のコマンドを入力します。

```
IPS ADD FILE filespec PACKAGE package_number
```

     filespecは、(パスを含む)完全修飾ファイル名である必要があります。ADRベース・ディレクトリ階層内に存在するファイルのみを追加できます。

関連トピック

- ADRCIコマンド・リファレンス
- Oracle Database ADRCの定義

親トピック: インシデント・パッケージの作成

21.8.2.3 物理インシデント・パッケージの生成

パッケージを生成するときは、既存の論理パッケージに対して物理パッケージ(ZIPファイル)を作成します。

1. ADRCIを対話方式モードで開始し、ホームパス(ADRベース階層内のディレクトリ)がADRベース・ディレクトリ階層内で正確なディレクトリを示していることを確認します。
   ホームパスおよびその他のADRCI用語の詳細は、「Oracle Database ADRCの定義」を参照してください。
2. ADRCIプロンプトで、生成するパッケージ情報(完全または増分)のコマンドを入力します。
   完全な物理パッケージを生成するには:

   次のコマンドは、指定したパスに完全な物理パッケージ(zipファイル)を生成します。

```
IPS GENERATE PACKAGE package_number IN path
```

   たとえば、次のコマンドは、ディレクトリ/home/steve/diagnosticsに論理パッケージ番号2から完全な物理パッケージを作成します。

```
IPS GENERATE PACKAGE 2 IN /home/steve/diagnostics
```

   増分物理パッケージを生成するには:

   また、最後にパッケージを生成して以降に発生したインシデントのみを含む増分パッケージも生成できます。ADRCIプロンプトで、次のコマンドを入力します。

```
IPS GENERATE PACKAGE package_number IN path INCREMENTAL
```

関連トピック

- インシデントのパッケージ化
- ADRCIコマンド・リファレンス
- Oracle Database ADRCの定義

親トピック: インシデント・パッケージの作成

21.9 ADRCIコマンド・リファレンス

自動診断リポジトリ・コマンド・インタプリタ(ADRCI)で使用できるコマンドについて学習します。

ADRCIには次の4種類のコマンドが用意されています。

- 1つ以上のカレントADRホームで動作するコマンド
- 1つのカレントADRホームのみで動作し、複数のカレントADRホームが存在する場合にはエラー・メッセージを発行するコマンド
- 複数のカレントADRホームが存在するときに、1つのADRホームを選択するように要求するコマンド
- カレントADRホームを必要としないコマンド

すべてのADRCIコマンドは、1つのカレントADRホームが存在する場合をサポートしています。

ノート:

特に指定されないかぎり、すべてのコマンドは複数のカレント ADR ホームで動作します。

- CREATE REPORT
- ECHO
  ECHO ADRCIユーティリティ・コマンドは、入力文字列を出力します。
- EXIT
  EXITコマンドは、ADRCIユーティリティを終了します。
- HOST
  ADRCIユーティリティのHOSTコマンドは、ADRCIを終了しないでオペレーティング・システム・コマンドを実行します。
- IPS
- PURGE
- QUIT
  QUITは、EXITコマンドのシノニムです。
- RUN
  ADRコマンド・インタプリタ(ADRCI)スクリプトを実行します。
- SELECT
- SET BASE
  カレントADRCIセッションで使用するADRベースを設定します。
- SET BROWSER
  レポートを表示するデフォルトのブラウザを設定します。
- SET CONTROL
  ADRの内容の削除ポリシーを設定します。
- SET ECHO
  コマンド出力をオンまたはオフにします。このコマンドは、スクリプトに表示される出力またはスプール・モードを使用する出力のみに影響します。
- SET EDITOR
  アラート・ログおよびトレース・ファイルの内容を表示するためのエディタを設定します。
- SET HOMEPATH
  1つ以上のADRホームをカレントにします。多くのADRコマンドが、カレントADRホームでのみ動作します。

- SET TERMOUT
  端末への出力をオンまたはオフにします。
- SHOW ALERT
- SHOW BASE
  カレントADRベースを示します。
- SHOW CONTROL
  削除ポリシーを含む、ADRについての情報を示します。
- SHOW HM_RUN
- SHOW HOMEPATH
  `SHOW HOMEPATH`コマンドは、`SHOW HOMES`コマンドと同じです。
- SHOW HOMES
  カレントADRCIセッションのADRホームを表示します。
- SHOW INCDIR
- SHOW INCIDENT
- SHOW LOG
  診断ログ・メッセージを表示します。
- SHOW PROBLEM
- SHOW REPORT
  指定したレポート・タイプおよび実行名のレポートを表示します。
- SHOW TRACEFILE
- SPOOL
  ADRCI出力をファイルへ送信します。

親トピック: ADRCI: ADRコマンド・インタプリタ

21.9.1 CREATE REPORT

用途

指定したレポート・タイプと実行IDのレポートを作成し、そのレポートをADRに格納します。現在は、`hm_run`(状態モニター)レポート・タイプのみがサポートされています。

> ノート:
>
> 状態モニターの実行の結果は、内部形式で ADR に格納されます。これらの結果を表示するには、結果から状態モニター・レポートを作成した後に、このレポートを表示する必要があります。レポートは 1 回のみ作成する必要があります。その後は、このレポートを複数回表示できます。

構文および説明

```
create report report_type run_name
```

`report_type`は`hm_run`である必要があります。`run_name`は状態モニターの実行名です。SHOW HM_RUNコマンドを使用して実行名を取得します。

すでにレポートが存在する場合は、そのレポートが上書きされます。レポートを表示するにはSHOW REPORTコマンドを使用します。

このコマンドは、複数のADRホームをサポートしていません。

例

この例では、実行名hm_run_1421を使用した状態モニターの実行に対するレポートを作成します。

```
create report hm_run hm_run_1421
```

ノート:

CREATE REPORT は、複数の ADR ホームが設定されている場合には機能しません。1 つの ADR ホームの設定については、「ADRCI コマンドを使用する前の ADRCI ホームパスの設定」を参照してください。

親トピック: ADRCIコマンド・リファレンス

21.9.2 ECHO

ECHO ADRCIユーティリティ・コマンドは、入力文字列を出力します。

用途

入力文字列を出力します。このコマンドを使用すると、ADRCIスクリプトからカスタム・テキストを出力できます。

構文および説明

```
ECHO quoted_string
```

文字列は、一重引用符または二重引用符で囲んで記述します。

このコマンドを使用する前に、ADRホームを設定しておく必要はありません。

例

次の例は、文字列Hello, world!を出力します。

```
ECHO "Hello, world!"

ECHO 'Hello, world!'
```

親トピック: ADRCIコマンド・リファレンス

21.9.3 EXIT

EXITコマンドは、ADRCIユーティリティを終了します。

用途

ADRCIユーティリティを終了します。

構文および説明

```
exit
```

EXITは、QUITコマンドのシノニムです。

このコマンドを使用する前に、ADRホームを設定しておく必要はありません。

親トピック: ADRCIコマンド・リファレンス

21.9.4 HOST

ADRCIユーティリティのHOSTコマンドは、ADRCIを終了しないでオペレーティング・システム・コマンドを実行します。

用途

ADRCIを終了しないでオペレーティング・システムのコマンドを実行します。

構文および説明

```
host ["host_command_string"]
```

hostのみを使用すると、オペレーティング・システムのシェルに入り、複数のオペレーティング・システムのコマンドを入力できるようになります。シェルを終了してADRCIに戻るにはEXITを入力します。

同じ行でコマンド(host_command_string)を二重引用符で囲んで指定することもできます。

このコマンドを使用する前に、ADRホームを設定しておく必要はありません。

例

```
host

host "ls -l *.pl"
```

親トピック: ADRCIコマンド・リファレンス

21.9.5 IPS

用途

インシデント・パッケージング・サービス(IPS)を呼び出します。IPSコマンドは、論理インシデント・パッケージ(パッケージ)の作成、パッケージへの診断データの追加、およびOracleサポート・サービスへ送信する物理パッケージの生成を行うためのオプションを提供します。

> ノート:
>
> IPS コマンドは、複数の ADR ホームが設定されている場合には機能しません。1 つの ADR ホームの設定については、「ADRCI コマンドを使用する前の ADRCI ホームパスの設定」を参照してください。

- IPSコマンドでの<ADR_HOME>および<ADR_BASE>変数の使用
  IPSコマンド・セットには、カレントADRホームとADRベース・ディレクトリを参照するためのショートカットがあります。
- IPS ADD
- IPS ADD FILE
  ファイルを既存のパッケージに追加します。
- IPS ADD NEW INCIDENTS
  指定したパッケージ内の問題すべてに対して新しいインシデントを検索して追加します。
- IPS COPY IN FILE
  外部ファイル・システムからADRにファイルをコピーします。
- IPS COPY OUT FILE
  ADRから外部ファイル・システムにファイルをコピーします。
- IPS CREATE PACKAGE

- IPS DELETE PACKAGE
  ADRからパッケージおよびその内容を削除します。
- IPS FINALIZE
  アップロードする前にパッケージをファイナライズします。
- IPS GENERATE PACKAGE
  ターゲット・ディレクトリに物理パッケージ(ZIPファイル)を作成します。
- IPS GET MANIFEST
  パッケージのZIPファイルからマニフェストを抽出して表示します。
- IPS GET METADATA
  パッケージ・ファイルからADR関連のメタデータを抽出して表示します。
- IPS PACK
- IPS REMOVE
- IPS REMOVE FILE
  既存のパッケージからファイルを削除します。
- IPS SET CONFIGURATION
  IPS構成パラメータの値を変更します。
- IPS SHOW CONFIGURATION
  IPS構成パラメータおよびその値のリストを表示します。
- IPS SHOW FILES
  指定したパッケージに含まれているファイルを表示します。
- IPS SHOW INCIDENTS
  指定したパッケージに含まれているインシデントを表示します。
- IPS SHOW PACKAGE
  指定したパッケージに関する情報を表示します。
- IPS UNPACK FILE
  物理パッケージ・ファイルを指定したパスに解凍します。

関連項目:

パッケージ化の詳細は、「インシデントのパッケージ化」を参照してください

親トピック: ADRCIコマンド・リファレンス

21.9.5.1 IPSコマンドでの<ADR_HOME>および<ADR_BASE>変数の使用

IPSのコマンド・セットは、カレントADRホームおよびADRベースのディレクトリを参照するためのショートカットを提供します。

カレントADRホームのディレクトリにアクセスするには、<ADR_HOME>変数を次のように使用します。

```
ips add file <ADR_HOME>/trace/orcl_ora_13579.trc package 12
```

ADRベース・ディレクトリにアクセスするには、<ADR_BASE>変数を次のように使用します。

```
ips add file <ADR_BASE>/diag/rdbms/orcl/orcl/trace/orcl_ora_13579.trc package 12
```

ノート:

山カッコ(< >)は、ここに示すとおりに入力します。

親トピック: IPS

21.9.5.2 IPS ADD

用途

インシデントをパッケージに追加します。

構文および説明

```
ips add {incident first [n] | incident inc_id | incident last [n] |
     problem first [n] | problem prob_id | problem last [n] |
     problemkey pr_key | seconds secs | time start_time to end_time}
     package package_id
```

表21-2に、IPS ADDの引数を示します。

表21-2 IPS ADDコマンドの引数

引数	説明
`incident first [n]`	最初に発生した n つのインシデントをパッケージに追加します(n は正の整数)。たとえば n に 5 を設定すると、最初に発生した 5 つのインシデントが追加されます。n を省略すると、デフォルトの 1 となり、最初に発生したインシデントが追加されます。
`incident inc_id`	ID が `inc_id` のインシデントをパッケージに追加します。
`incident last [n]`	最後に発生した n つのインシデントをパッケージに追加します(n は正の整数)。たとえば n に 5 を設定すると、最後に発生した 5 つのインシデントが追加されます。n を省略すると、デフォルトの 1 となり、最後に発生したインシデントが追加されます。
`problem first [n]`	最初に発生した n つの問題のインシデントをパッケージに追加します(n は正の整数)。たとえば n に 5 を設定すると、最初に発生した 5 つの問題のインシデントが追加されます。n を省略すると、デフォルトの 1 となり、最初に発生した問題のインシデントが追加されます。 90 日を超えるインシデントを除き、各問題について、最初に発生した 3 つの早期インシデントと最後に発生した 3 つの最新インシデントのみを追加します。(ノート: これらの制限はデフォルト値であり、変更可能です。詳細は、「IPS SET CONFIGURATION」を参照してください。
`problem prob_id`	問題 ID が `prob_id` のインシデントをすべてパッケージに追加します。90 日を超えるインシデントを除き、その問題について、最初に発生した 3 つの早期インシデントと最後に発生した 3 つの最新インシデントのみを追加します。(ノート: これらの制限はデフォルト値であり、変更可能です。詳細は、

引数	説明
	「IPS SET CONFIGURATION」を参照してください。
`problem last [n]`	最後に発生した n つの問題のインシデントをパッケージに追加します(n は正の整数)。たとえば n に 5 を設定すると、最後に発生した 5 つの問題のインシデントが追加されます。n が省略されると、デフォルトの 1 となり、最後に発生した問題のインシデントが追加されます。 90 日を超えるインシデントを除き、各問題について、最初に発生した 3 つの早期インシデントと最後に発生した 3 つの最新インシデントのみを追加します。(ノート: これらの制限はデフォルト値であり、変更可能です。詳細は、「IPS SET CONFIGURATION」を参照してください。
`problemkey pr_key`	問題キーが `pr_key` のインシデントをパッケージに追加します。90 日を超えるインシデントを除き、その問題キーについて、最初に発生した 3 つの早期インシデントと最後に発生した 3 つの最新インシデントのみを追加します。(ノート: これらの制限はデフォルト値であり、変更可能です。)
`seconds secs`	現時点で過去 `secs` 秒以内に発生したすべてのインシデントを追加します。
`time start_time to end_time`	`start_time` と `end_time` の間のすべてのインシデントをパッケージに追加します。時刻の書式は `YYYY-MM-YY HH24:MI:SS.FF TZR` です。小数部分(`FF`)はオプションです。
`package package_id`	インシデントの追加先となるパッケージを指定します。

例

この例では、インシデント22をパッケージ12に追加します。

```
ips add incident 22 package 12
```

この例では、90日を超えるインシデントを除き、問題IDが6の最初に発生した3つの早期インシデントおよび最後に発生した3つの最新インシデントをパッケージ2に追加します。

```
ips add problem 6 package 2
```

この例では、過去1分間に発生したすべてのインシデントをパッケージ5に追加します。

```
ips add seconds 60 package 5
```

この例では、2010年5月1日の午前10時から午後11時の間に発生したすべてのインシデントを追加します。

```
ips add  time '2010-05-01 10:00:00.00 -07:00' to '2010-05-01 23:00:00.00 -07:00'
```

親トピック: IPS

21.9.5.3 IPS ADD FILE

ファイルを既存のパッケージに追加します。

構文および説明

```
ips add file file_name package package_id
```

file_nameは、ファイルのフルパス名です。必要に応じて、<ADR_HOME>および<ADR_BASE>変数を使用できます。ファイルは、パッケージと同じADRベース下に存在する必要があります。

package_idはパッケージのIDです。

例

この例では、トレース・ファイルをパッケージ12に追加します。

```
ips add file <ADR_HOME>/trace/orcl_ora_13579.trc package 12
```

関連項目:

<ADR_HOME>ディレクトリ構文の詳細は、「IPSコマンドでの<ADR_HOME>および<ADR_BASE>変数の使用」を参照してください

親トピック: IPS

21.9.5.4 IPS ADD NEW INCIDENTS

指定したパッケージ内の問題すべてに対して新しいインシデントを検索して追加します。

構文および説明

```
ips add new incidents package package_id
```

package_idは更新するパッケージのIDです。パッケージ内の問題の新しいインシデントのみが追加されます。

例

この例では、パッケージ12内の問題について新しい最新インシデントを3つまで追加します。

```
ips add new incidents package 12
```

ノート:

追加される最新インシデントの数はデフォルト値であり、変更が可能です。IPS SET CONFIGURATION を参照してください。

親トピック: IPS

21.9.5.5 IPS COPY IN FILE

外部ファイル・システムからADRへファイルをコピーします。

用途

パッケージ内のファイルを編集するには、ファイルを指定したディレクトリにコピーし、そのファイルを編集してから、コピーして元のパッケージに戻す必要があります。この作業は、パッケージをOracleサポート・サービスに送信する前にファイル内の機密データを削

除するために行うことがあります。

構文および説明

```
ips copy in file filename [to new_name][overwrite] package package_id
     [incident incid]
```

外部ファイルfilename(フルパス名で指定)をADRにコピーし、その際に既存のパッケージpackage_idおよびオプションでインシデントincidと関連付けます。ADR内でコピーされたファイルに新しいファイル名を付ける場合は、to new_nameオプションを使用します。すでに存在するファイルを上書きする場合は、overwriteオプションを使用します。

例

この例では、トレース・ファイルをファイル・システムからADRにコピーします。ファイルはパッケージ2およびインシデント4に関連付けます。

```
ips copy in file /home/nick/trace/orcl_ora_13579.trc to
<ADR_HOME>/trace/orcl_ora_13579.trc package 2 incident 4
```

関連項目:

- <ADR_HOME>変数の詳細は、「IPSコマンドでの<ADR_HOME>および<ADR_BASE>変数の使用」を参照してください
- パッケージ内のファイルのリスト表示については、「IPS SHOW FILES」を参照してください

親トピック: IPS

21.9.5.6 IPS COPY OUT FILE

ADRから外部ファイル・システムへファイルをコピーします。

用途

パッケージ内のファイルを編集するには、ファイルを指定したディレクトリにコピーし、そのファイルを編集してから、コピーして元のパッケージに戻す必要があります。この作業は、パッケージをOracleサポート・サービスに送信する前にファイル内の機密データを削除するために行うことがあります。

構文および説明

```
ips copy out file source to target [overwrite]
```

ファイルsourceをADRの外の場所target(フルパス名で指定)にコピーします。すでに存在するファイルを上書きする場合は、overwriteオプションを使用します。

例

この例では、カレントのADRホームのトレース・サブディレクトリ内のファイルorcl_ora_13579.trcをローカル・フォルダにコピーします。

```
ips copy out file <ADR_HOME>/trace/orcl_ora_13579.trc to
/home/nick/trace/orcl_ora_13579.trc
```

関連項目:

- <ADR_HOME>ディレクトリ構文の詳細は、「IPSコマンドでの<ADR_HOME>および<ADR_BASE>変数の使用」を参照
- パッケージ内のファイルのリスト表示については、「IPS SHOW FILES」を参照してください

親トピック: IPS

21.9.5.7 IPS CREATE PACKAGE

用途

新しいパッケージを作成します。ADRCIにより、新しいパッケージに対してパッケージ番号が自動的に割り当てられます。

構文および説明

```
ips create package {incident first [n] | incident inc_id |
     incident last [n] | problem first [n] | problem prob_id |
     problem last [n] | problemkey prob_key | seconds secs |
     time start_time to end_time} [correlate {basic |typical | all}]
```

また、指定されたオプションを使用しても、インシデントを新しいパッケージに追加できます。

表21-3に、IPS CREATE PACKAGEの引数を示します。

表21-3 IPS CREATE PACKAGEコマンドの引数

引数	説明
incident first [n]	最初に発生した n つのインシデントをパッケージに追加します(n は正の整数)。たとえば n に 5 を設定すると、最初に発生した 5 つのインシデントが追加されます。n を省略すると、デフォルトの 1 となり、最初に発生したインシデントが追加されます。
incident inc_id	ID が inc_id のインシデントをパッケージに追加します。
incident last [n]	最後に発生した n つのインシデントをパッケージに追加します(n は正の整数)。たとえば n に 5 を設定すると、最後に発生した 5 つのインシデントが追加されます。n を省略すると、デフォルトの 1 となり、最後に発生したインシデントが追加されます。
problem first [n]	最初に発生した n つの問題のインシデントをパッケージに追加します(n は正の整数)。たとえば n に 5 を設定すると、最初に発生した 5 つの問題のインシデントが追加されます。n を省略すると、デフォルトの 1 となり、最初に発生した問題のインシデントが追加されます。 90 日を超えるインシデントを除き、各問題について、最初に発生した 3 つの早期インシデントと最後に発生した 3 つの最新インシデントのみを追加します。(ノート: これらの制限はデフォルト値であり、変更可能です。詳細は、「IPS SET CONFIGURATION」を参照してください。

引数	説明
problem prob_id	問題 ID が prob_id のインシデントをすべてパッケージに追加します。90 日を超えるインシデントを除き、その問題について、最初に発生した 3 つの早期インシデントと最後に発生した 3 つの最新インシデントのみを追加します。(ノート: これらの制限はデフォルト値であり、変更可能です。詳細は、「IPS SET CONFIGURATION」を参照してください。
problem last [n]	最後に発生した n つの問題のインシデントをパッケージに追加します(n は正の整数)。たとえば n に 5 を設定すると、最後に発生した 5 つの問題のインシデントが追加されます。n が省略されると、デフォルトの 1 となり、最後に発生した問題のインシデントが追加されます。 90 日を超えるインシデントを除き、各問題について、最初に発生した 3 つの早期インシデントと最後に発生した 3 つの最新インシデントのみを追加します。(ノート: これらの制限はデフォルト値であり、変更可能です。詳細は、「IPS SET CONFIGURATION」を参照してください。
problemkey pr_key	問題キーが pr_key のインシデントをすべてパッケージに追加します。90 日を超えるインシデントを除き、その問題キーについて、最初に発生した 3 つの早期インシデントと最後に発生した 3 つの最新インシデントのみを追加します。(ノート: これらの制限はデフォルト値であり、変更可能です。)
seconds secs	現時点で過去 secs 秒以内に発生したすべてのインシデントを追加します。
time start_time to end_time	start_time と end_time の間に発生したすべてのインシデントをパッケージに追加します。時刻の書式は YYYY-MM-YY HH24:MI:SS.FF TZR です。小数部分(FF)はオプションです。
correlate {basic |typical | all}	パッケージに相関インシデントを指定する方法を選択します。この引数には 3 つのオプションがあります。 ● correlate basic を選択すると、インシデント・ダンプおよびインシデント・プロセスのトレース・ファイルが指定されます。 ● correlate typical を選択すると、各インシデントのインシデント・ダンプ、および 5 分以内に変更されたトレース・ファイルが指定されます。時間間隔を変更するには、INCIDENT_TIME_WINDOW 構成パラメータを変更します。 ● correlate all を選択すると、インシデント・ダンプ、およびインシデントが最初に選択された時刻と最後に選択された時刻の間に

引数	説明
	変更されたすべてのトレース・ファイルが指定されます。 デフォルト値は `correlate typical` です。

例

この例では、インシデントを含まないパッケージを作成します。

```
ips create package
```

出力:

```
Created package 5 without any contents, correlation level typical
```

この例では、指定された日の午前10時から午後11時の間に発生したすべてのインシデントを含むパッケージを作成します。

```
ips create package time '2010-05-01 10:00:00.00 -07:00' to '2010-05-01 23:00:00.00 -
07:00'
```

出力:

```
Created package 6 based on time range 2010-05-01 10:00:00.00 -07:00 to 2010-05-01
23:00:00.00 -07:00, correlation level typical
```

この例では、パッケージを作成し、90日を超えるインシデントを除いて、問題IDが3の最初に発生した3つの早期インシデントおよび最後に発生した3つの最新インシデントを追加します。

```
ips create package problem 3
```

出力:

```
Created package 7 based on problem id 3, correlation level typical
```

ノート:

追加される早期インシデントと最新インシデントの数と 90 日の経過日数制限はデフォルト値であり、変更が可能です。詳細は、「IPS SET CONFIGURATION」を参照してください。

関連項目:

「インシデント・パッケージの作成」

親トピック: IPS

21.9.5.8 IPS DELETE PACKAGE

ADRからパッケージおよびその内容を削除します。

構文および説明

```
ips delete package package_id
```

package_idは削除するパッケージです。

例

```
ips delete package 12
```

親トピック: IPS

21.9.5.9 IPS FINALIZE

アップロードする前にパッケージをファイナライズします。

構文および説明

```
ips finalize package package_id
```

package_idはファイナライズするパッケージのIDです。

例

```
ips finalize package 12
```

関連項目:

パッケージのファイナライズの詳細は、『Oracle Database管理者ガイド』を参照してください。

親トピック: IPS

21.9.5.10 IPS GENERATE PACKAGE

ターゲット・ディレクトリに物理パッケージ(ZIPファイル)を作成します。

構文および説明

```
ips generate package package_id [in path] [complete | incremental]
```

package_idは生成するパッケージのIDです。オプションで、ディレクトリpathにファイルを保存できます。このオプションを指定しない場合、パッケージは現在の作業ディレクトリに生成されます。

completeオプションを指定すると、ADRCIにより、すべてのパッケージ・ファイルがパッケージに強制的に含まれるようになります。これはデフォルトの動作です。

incrementalオプションを選択すると、このパッケージが最後に生成された後に追加または変更されたファイルのみが指定されます。したがって、incrementalオプションでのコマンドの動作はより短い時間で終了します。

例

この例では、パス/home/steveに物理パッケージ・ファイルを生成します。

```
ips generate package 12 in /home/steve
```

この例では、物理パッケージが最後に生成された後に追加または変更されたファイルからパッケージを生成します。

```
ips generate package 14 incremental
```

関連項目:

物理インシデント・パッケージの生成

親トピック: IPS

21.9.5.11 IPS GET MANIFEST

パッケージのZIPファイルからマニフェストを抽出して表示します。

構文および説明

```
ips get manifest from file filename
```

filenameは、パッケージのZIPファイルです。マニフェストは、パッケージ・ファイルのXML形式のメタデータのセットで、ADR構成、相関ファイル、インシデントおよびパッケージが生成された方法に関する情報を含みます。

このコマンドを使用する前に、ADRホームを設定しておく必要はありません。

例

```
ips get manifest from file /home/steve/ORA603_20060906165316_COM_1.zip
```

親トピック: IPS

21.9.5.12 IPS GET METADATA

パッケージ・ファイルからADR関連のメタデータを抽出して表示します。

構文および説明

```
ips get metadata {from file filename | from adr}
```

filenameは、パッケージのZIPファイルです。(metadata.xmlに格納されている)パッケージ・ファイルのメタデータには、ADRホーム、ADRベースおよび製品に関する情報が含まれています。

IPS UNPACKを使用してADRホームに解凍されたパッケージZIPファイルからメタデータを取得するには、from adrオプションを使用します。

from adrオプションを使用する場合は、ADRホームを設定する必要があります。

例

この例では、パッケージ・ファイルからメタデータを表示します。

```
ips get metadata from file /home/steve/ORA603_20060906165316_COM_1.zip
```

次の例では、ディレクトリ/scratch/oracle/package1に解凍したパッケージ・ファイルからメタデータを表示します。

```
set base /scratch/oracle/package1
ips get metadata from adr
```

前述の例でADRCIは、SET BASEコマンドを受け取ると、IPS UNPACK FILEコマンドで/scratch/oracle/package1に作成されたADRホームをホームパスに自動的に追加します。

関連項目:

パッケージ・ファイルの解凍の詳細は、「IPS UNPACK FILE」を参照してください

親トピック: IPS

21.9.5.13 IPS PACK

用途

すぐにパッケージを作成して、物理パッケージを生成します。

構文および説明

```
ips pack [incident first [n] | incident inc_id | incident last [n] |
     problem first [n] | problem prob_id | problem last [n] |
     problemkey prob_key | seconds secs | time start_time to end_time]
     [correlate {basic |typical | all}] [in path]
```

ADRCIにより、自動的に新しいパッケージに対してパッケージ番号が生成されます。パッケージの内容が指定されない場合、IPS PACKは空のパッケージを作成します。

表21-4に、IPS PACKの引数を示します。

表21-4 IPS PACKコマンドの引数

引数	説明
incident first [n]	最初に発生した n つのインシデントをパッケージに追加します(n は正の整数)。たとえば n に 5 を設定すると、最初に発生した 5 つのインシデントが追加されます。n を省略すると、デフォルトの 1 となり、最初に発生したインシデントが追加されます。
incident inc_id	ID が inc_id のインシデントをパッケージに追加します。
incident last [n]	最後に発生した n つのインシデントをパッケージに追加します(n は正の整数)。たとえば n に 5 を設定すると、最後に発生した 5 つのインシデントが追加されます。n を省略すると、デフォルトの 1 となり、最後に発生したインシデントが追加されます。
problem first [n]	最初に発生した n つの問題のインシデントをパッケージに追加します(n は正の整数)。たとえば n に 5 を設定すると、最初に発生した 5 つの問題のインシデントが追加されます。n を省略すると、デフォルトの 1 となり、最初に発生した問題のインシデントが追加されます。 90 日を超えるインシデントを除き、各問題について、最初に発生した 3 つの早期インシデントと最後に発生した 3 つの最新インシデントのみを追加します。(ノート: これらの制限はデフォルト値であり、変更可能です。詳細は、「IPS SET CONFIGURATION」を参照してください。
problem prob_id	問題 ID が prob_id のインシデントをすべてパッケージに追加します。90 日を超えるインシデントを除き、その問題について、最初に発生した 3 つの早期インシデントと最後に発生した 3 つの最新インシデントのみを追加します。(ノート: これらの制限はデフォルト値であり、変更可能です。詳細は、

引数	説明
	「IPS SET CONFIGURATION」を参照してください。
problem last [n]	最後に発生した n つの問題のインシデントをパッケージに追加します(n は正の整数)。たとえば n に 5 を設定すると、最後に発生した 5 つの問題のインシデントが追加されます。n が省略されると、デフォルトの 1 となり、最後に発生した問題のインシデントが追加されます。 90 日を超えるインシデントを除き、各問題について、最初に発生した 3 つの早期インシデントと最後に発生した 3 つの最新インシデントのみを追加します。(ノート: これらの制限はデフォルト値であり、変更可能です。詳細は、「IPS SET CONFIGURATION」を参照してください。
problemkey pr_key	問題キーが pr_key のインシデントをパッケージに追加します。90 日を超えるインシデントを除き、その問題キーについて、最初に発生した 3 つの早期インシデントと最後に発生した 3 つの最新インシデントのみを追加します。(ノート: これらの制限はデフォルト値であり、変更可能です。)
seconds secs	現時点で過去 secs 秒以内に発生したすべてのインシデントを追加します。
time start_time to end_time	start_time と end_time の間に発生したすべてのインシデントをパッケージに追加します。時刻の書式は YYYY-MM-YY HH24:MI:SS.FF TZR です。小数部分(FF)はオプションです。
correlate {basic |typical | all}	パッケージに相関インシデントを指定する方法を選択します。この引数には 3 つのオプションがあります。 ● correlate basic を選択すると、インシデント・ダンプおよびインシデント・プロセスのトレース・ファイルが指定されます。 ● correlate typical を選択すると、各インシデントのインシデント・ダンプ、および 5 分以内に変更されたトレース・ファイルが指定されます。時間間隔を変更するには、INCIDENT_TIME_WINDOW 構成パラメータを変更します。 ● correlate all を選択すると、インシデント・ダンプ、およびインシデントが最初に選択された時刻と最後に選択された時刻の間に変更されたすべてのトレース・ファイルが指定されます。 デフォルト値は correlate typical です。

引数	説明
in path	物理パッケージをディレクトリ path へ保存します。

例

この例では、空のパッケージを作成します。

```
ips pack
```

この例では、インシデント861に関するすべての情報を含む物理パッケージを作成します。

```
ips pack incident 861
```

次の例では、過去1分間のすべてのインシデントを完全な相関状態で含む物理パッケージを作成します。

```
ips pack seconds 60 correlate all
```

関連項目:

構成パラメータの設定の詳細は、「IPS SET CONFIGURATION」を参照してください。

親トピック: IPS

21.9.5.14 IPS REMOVE

用途

既存のパッケージからインシデントを削除します。

構文および説明

```
ips remove {incident inc_id | problem prob_id | problemkey prob_key}
     package package_id
```

パッケージからインシデントを削除した後も、これらのインシデントは引き続きパッケージのメタデータ内で追跡され、(ADD NEW INCIDENTSなどを使用したときに)ADRCIが後から自動的にこれらのインシデントを追加してしまうのを防ぎます。

表21-5に、IPS REMOVEの引数を示します。

表21-5 IPS REMOVEコマンドの引数

引数	説明
incident inc_id	ID が inc_id のインシデントをパッケージから削除します。
problem prob_id	問題 ID が prob_id のインシデントをすべてパッケージから削除します。
problemkey pr_key	問題キーが pr_key のインシデントをすべてパッケージから削除します。
package package_id	ID が package_id のパッケージからインシデントを削除します。

例

この例では、パッケージ12からインシデント22を削除します。

```
ips remove incident 22 package 12
```

関連項目:

パッケージのメタデータの詳細は、「IPS GET MANIFEST」を参照してください

親トピック: IPS

21.9.5.15 IPS REMOVE FILE

既存のパッケージからファイルを削除します。

構文および説明

```
ips remove file file_name package package_id
```

file_nameはパッケージpackage_idから削除するファイルです。ファイルの完全パスを指定する必要があります。(必要に応じて、<ADR_HOME>および<ADR_BASE>変数を使用できます。)

削除後も、このファイルは引き続きパッケージのメタデータ内で追跡され、(ADD NEW INCIDENTSなどを使用したときに)ADRCIが後から自動的にこのファイルを追加してしまうのを防ぎます。したがって、ファイルを削除しても、そのファイルのEXCLUDEフラグがExplicitly excludedに設定されるだけです。

例

この例では、パッケージ12からトレース・ファイルを削除します。

```
ips remove file <ADR_HOME>/trace/orcl_ora_13579.trc package 12
Removed file <ADR_HOME>/trace/orcl_ora_13579.trc from package 12
ips show files package 12
.
.
.
FILE_ID                 4
FILE_LOCATION           <ADR_HOME>/trace
FILE_NAME               orcl_ora_13579.trc
LAST_SEQUENCE           0
EXCLUDE                 Explicitly excluded
.
.
.
```

関連項目:

- パッケージのメタデータの詳細は、「IPS GET MANIFEST」を参照してください
- <ADR_BASE>ディレクトリ構文の詳細は、「IPSコマンドでの<ADR_HOME>および<ADR_BASE>変数の使用」を参照してください
- IPS SHOW FILES

親トピック: IPS

21.9.5.16 IPS SET CONFIGURATION

IPS構成パラメータの値を変更します。

構文および説明

```
ips set configuration {parameter_id | parameter_name} value
```

parameter_idは変更するパラメータのIDでparameter_nameは、変更するパラメータの名前です。valueは新規の値です。構成パラメータおよびそのIDのリストについては、IPS SHOW CONFIGURATIONを使用します。

例

```
ips set configuration 3 10
```

親トピック: IPS

21.9.5.17 IPS SHOW CONFIGURATION

IPS構成パラメータおよびその値のリストを表示します。

用途

これらのパラメータは、タイムアウトおよびインシデント追加間隔など、IPSデータの各種しきい値を制御します。

構文および説明

```
ips show configuration {parameter_id | parameter_name}]
```

IPS SHOW CONFIGURATIONは、構成パラメータごとに次の情報を表示します。

- パラメータID
- 名前
- 説明
- パラメータを使用する単位(日数や時間数など)
- 値
- デフォルト値
- 最小値
- 最大値
- フラグ

オプションで、parameter_idまたはparameter_nameを指定して特定のパラメータに関する情報を取得できます。

例

次のコマンドは、すべてのIPS構成パラメータを示します。

```
ips show configuration
```

出力:

```
PARAMETER INFORMATION:
   PARAMETER_ID            1
   NAME                    CUTOFF_TIME
   DESCRIPTION             Maximum age for an incident to be considered for
```

```
                            inclusion
   UNIT                     Days
   VALUE                    90
   DEFAULT_VALUE            90
   MINIMUM                  1
   MAXIMUM                  4294967295
   FLAGS                    0

PARAMETER INFORMATION:
   PARAMETER_ID             2
   NAME                     NUM_EARLY_INCIDENTS
   DESCRIPTION              How many incidents to get in the early part of the range
   UNIT                     Number
   VALUE                    3
   DEFAULT_VALUE            3
   MINIMUM                  1
   MAXIMUM                  4294967295
   FLAGS                    0

PARAMETER INFORMATION:
   PARAMETER_ID             3
   NAME                     NUM_LATE_INCIDENTS
   DESCRIPTION              How many incidents to get in the late part of the range
   UNIT                     Number
   VALUE                    3
   DEFAULT_VALUE            3
   MINIMUM                  1
   MAXIMUM                  4294967295
   FLAGS                    0

PARAMETER INFORMATION:
   PARAMETER_ID             4
   NAME                     INCIDENT_TIME_WINDOW
   DESCRIPTION              Incidents this close to each other are considered
                            correlated
   UNIT                     Minutes
   VALUE                    5
   DEFAULT_VALUE            5
   MINIMUM                  1
   MAXIMUM                  4294967295
   FLAGS                    0

PARAMETER INFORMATION:
   PARAMETER_ID             5
   NAME                     PACKAGE_TIME_WINDOW
   DESCRIPTION              Time window for content inclusion is from x hours
                            before first included incident to x hours after last
                            incident
   UNIT                     Hours
   VALUE                    24
   DEFAULT_VALUE            24
   MINIMUM                  1
   MAXIMUM                  4294967295
   FLAGS                    0

PARAMETER INFORMATION:
   PARAMETER_ID             6
   NAME                     DEFAULT_CORRELATION_LEVEL
   DESCRIPTION              Default correlation level for packages
   UNIT                     Number
   VALUE                    2
   DEFAULT_VALUE            2
   MINIMUM                  1
   MAXIMUM                  4
   FLAGS                    0
```

例

このコマンドは、構成パラメータNUM_EARLY_INCIDENTSを示します。

```
ips show configuration num_early_incidents
```

このコマンドは、構成パラメータ3を示します。

```
ips show configuration 3
```

構成パラメータの説明

表21-6に、IPS構成パラメータの詳細を示します。

表21-6 IPS構成パラメータ

パラメータ	ID	説明
CUTOFF_TIME	1	インシデントが追加対象とみなされる最長期間(日単位)。
NUM_EARLY_INCIDENTS	2	問題に基づいてパッケージを作成するときに、前半部分に含めるインシデントの数。デフォルトでは、ADRCI は最初の 3 つのインシデントと最新の 3 つのインシデントをパッケージに追加します。
NUM_LATE_INCIDENTS	3	問題に基づいてパッケージを作成するときに、後半部分に含めるインシデントの数。デフォルトでは、ADRCI は最初の 3 つのインシデントと最新の 3 つのインシデントをパッケージに追加します。
INCIDENT_TIME_WINDOW	4	2 つのインシデントが相関とみなされるためのインシデント発生の間隔(分数)。
PACKAGE_TIME_WINDOW	5	インシデントをパッケージに追加するための時間ウィンドウとして使用時間数。たとえば、値が 5 の場合、パッケージ内でインシデントが最初に発生した 5 時間前、およびパッケージ内の最新のインシデントの発生から 5 時間後のインシデントを含めます。
DEFAULT_CORRELATION_LEVEL	6	パッケージ内のインシデントを相関付けるために使用するデフォルトの相関レベル。相関レベルは次のとおりです。 ● 1(basic): インシデント・ダンプおよびインシデント・プロセスのトレース・ファイルが含まれます。 ● 2(typical): インシデント・ダンプおよび INCIDENT_TIME_WINDOW(前述の内容を参照)で指定された時間ウィンドウ内で変更されたトレース・ファイルが含まれます。 ● 4(all): インシデント・ダンプ、および最初に選択されたインシ

パラメータ	ID	説明
		デントと最後に選択されたインシデント間で変更されたすべてのトレース・ファイルが含まれます。同じ時間範囲内で追加インシデントが発生した場合は、これらのインシデントを自動的に含めることができます。

関連項目:

IPS SET CONFIGURATION

親トピック: IPS

21.9.5.18 IPS SHOW FILES

指定したパッケージ内に含まれているファイルを表示します。

用途

指定したパッケージ内に含まれているファイルを表示します。

構文および説明

```
ips show files package package_id
```

`package_id`は表示するパッケージのIDです。

例

この例では、パッケージ1に関連付けられているすべてのファイルを示します。

```
ips show files package 1
```

出力:

```
   FILE_ID                 1
   FILE_LOCATION           <ADR_HOME>/alert
   FILE_NAME               log.xml
   LAST_SEQUENCE           1
   EXCLUDE                 Included

   FILE_ID                 2
   FILE_LOCATION           <ADR_HOME>/trace
   FILE_NAME               alert_adcdb.log
   LAST_SEQUENCE           1
   EXCLUDE                 Included

   FILE_ID                 27
   FILE_LOCATION           <ADR_HOME>/incident/incdir_4937
   FILE_NAME               adcdb_ora_692_i4937.trm
   LAST_SEQUENCE           1
   EXCLUDE                 Included

   FILE_ID                 28
   FILE_LOCATION           <ADR_HOME>/incident/incdir_4937
   FILE_NAME               adcdb_ora_692_i4937.trc
   LAST_SEQUENCE           1
   EXCLUDE                 Included
```

```
   FILE_ID                 29
   FILE_LOCATION           <ADR_HOME>/trace
   FILE_NAME               adcdb_ora_692.trc
   LAST_SEQUENCE           1
   EXCLUDE                 Included

   FILE_ID                 30
   FILE_LOCATION           <ADR_HOME>/trace
   FILE_NAME               adcdb_ora_692.trm
   LAST_SEQUENCE           1
   EXCLUDE                 Included
.
.
.
```

親トピック: IPS

21.9.5.19 IPS SHOW INCIDENTS

指定したパッケージ内に含まれているインシデントを表示します。

構文および説明

```
ips show incidents package package_id
```

package_idは表示するパッケージのIDです。

例

この例では、パッケージ1のインシデントを表示します。

```
ips show incidents package 1
```

出力:

```
MAIN INCIDENTS FOR PACKAGE 1:
   INCIDENT_ID             4985
   PROBLEM_ID              1
   EXCLUDE                 Included

CORRELATED INCIDENTS FOR PACKAGE 1:
```

親トピック: IPS

21.9.5.20 IPS SHOW PACKAGE

指定したパッケージに関する情報を表示します。

構文および説明

```
ips show package package_id {basic | brief | detail}
```

package_idは表示するパッケージのIDです。

basicオプションを使用して、最小限の情報を表示します。これは、package_idが指定されていない場合のデフォルトです。

briefオプションを使用して、basicオプションよりも詳細なパッケージ情報を表示します。これは、package_idが指定されている場合のデフォルトです。

detailオプションを使用して、briefオプションで表示される情報に加えて、パッケージ履歴および含まれているインシデントやファイルの情報を表示します。

例

```
ips show package 12

ips show package 12 brief
```

親トピック: IPS

21.9.5.21 IPS UNPACK FILE

物理パッケージ・ファイルを指定したパスに解凍します。

構文および説明

```
ips unpack file file_name [into path]
```

file_nameは、解凍する物理パッケージ(ZIPファイル)のフルパス名です。オプションで、ファイルをディレクトリpathに解凍することもできます。このディレクトリは、すでに存在していて書込み可能である必要があります。パスを省略すると、現在の作業ディレクトリが使用されます。解凍先のディレクトリはADRベースとして処理され、有効なADRホームが含まれるADRベースのディレクトリ階層が完全に作成されます。

このコマンドを使用する前に、ADRホームを設定しておく必要はありません。

例

```
ips unpack file /tmp/ORA603_20060906165316_COM_1.zip into /tmp/newadr
```

親トピック: IPS

21.9.6 PURGE

用途

現在の削除ポリシーに従って、カレントADRホーム内の診断データを削除します。削除が予定されているADRの内容のみが削除されます。

ADR内の診断データにはデフォルトのライフサイクルが設定されています。たとえば、インシデントおよび問題に関する情報は1年後に削除されますが、関連するダンプ・ファイル(ダンプ)はわずか30日後に削除されます。

Oracle Databaseなどの一部のオラクル社の製品では、ライフ・サイクルの終了時に診断データが自動的に削除されます。その他の製品やコンポーネントでは、このコマンドを使用して診断データを手動で削除する必要があります。また、このコマンドで、自動削除される予定のデータも削除できます。

SHOW CONTROLコマンドでは、存続期間が短いADRの内容と存続期間が長いADRの内容に対して、デフォルトの削除ポリシーが表示されます。

構文および説明

```
purge [-i {id | start_id end_id} |
  -age mins [-type {ALERT|INCIDENT|TRACE|CDUMP|HM|UTSCDMP}]]
```

表21-7に、PURGEのフラグを示します。

表21-7 PURGEコマンドのフラグ

フラグ	説明

フラグ	説明
-i {id1 | start_id end_id}	特定のインシデント ID(id)またはインシデント ID の範囲(start_id および end_id)を削除します。
-age mins	mins 分以上経過したデータのみを削除します。
-type {ALERT|INCIDENT|TRACE|CDUMP|HM|UTSCDMP}	削除する診断データのタイプを指定します。-age 句とともに使用します。 次のタイプを指定できます。 ● ALERT - アラート・ログ ● INCIDENT - インシデント・データ ● TRACE - トレース・ファイル(ダンプを含む) ● CDUMP - コア・ダンプ・ファイル ● HM - 状態モニターの実行データおよびレポート ● UTSCDMP - 各セッションのインメモリー・トレースのダンプ UTSCDMP データは、トレース・ディレクトリの下のディレクトリに格納されます。これらのディレクトリの名前は、cdmp_timestamp です。バックグラウンド・プロセスは、クリティカル・エラー(ORA-600 または ORA-7445 エラーなど)が発生すると、このようなディレクトリを作成し、各セッションのインメモリー・トレース・データをトレース・ファイルに書き込みます。このデータは、エラーが発生する前に実行されていたインスタンスを特定するのに役立ちます。

例

この例では、デフォルトの削除ポリシーに基づいて、カレントADRホーム内のすべての診断データを削除します。

```
purge
```

この例では、123から456の間のすべてのインシデントについて、診断データをすべて削除します。

```
purge -i 123 456
```

この例では、過去1時間より前のすべてのインシデント・データを削除します。

```
purge -age 60 -type incident
```

ノート:

PURGE は、複数の ADR ホームが設定されている場合には機能しません。1 つの ADR ホームの設定については、「ADRCI コマンドを使用する前の ADRCI ホームパスの設定」を参照してください。

親トピック: ADRCIコマンド・リファレンス

21.9.7 QUIT

QUITは、EXITコマンドのシノニムです。

関連トピック

- EXIT

親トピック: ADRCIコマンド・リファレンス

21.9.8 RUN

ADRコマンド・インタプリタ(ADRCI)スクリプトを実行します。

構文および説明

```
run script_name
@ script_name
@@ script_name
```

変数script_nameは、実行するADRCIコマンドが含まれているファイルです。フルパス名が指定されていないかぎり、ADRCIはカレント・ディレクトリ内のスクリプトを検索します。ファイル拡張子のないファイル名を指定すると、ADRCIはデフォルトの拡張子.adiを使用します。

runおよび@コマンドはシノニムです。@@コマンドは、runおよび@と同じです。ただし、スクリプト内で使用すると、@@はカレント・ディレクトリではなく、コール元スクリプトのパスを使用してscript_nameを検索します。

runコマンドの使用前に、ADRホームを設定しておく必要はありません。

例

```
run my_script
@my_script
```

親トピック: ADRCIコマンド・リファレンス

21.9.9 SELECT

用途

指定されたインシデントまたは問題に関して、修飾されたレコードを取得します。

構文および説明

```
select {*|[field1, [field2, ...]} FROM {incident|problem}
  [WHERE predicate_string]
  [ORDER BY field1 [, field2, ...] [ASC|DSC|DESC]]
  [GROUP BY field1 [, field2, ...]]
  [HAVING having_predicate_string]
```

表21-8 SELECTコマンドのフラグ

フラグ	説明
field1, field2, ...	取得するフィールドをリストします。*を指定すると、すべてのフィールドが取得されます。
incident|problem	インシデントまたは問題を問い合せるかどうかを示します。
WHERE "predicate_string"	SQL に類似した述語文字列を使用して、述語が true であるインシデントまたは問題のみを表示します。述語文字列は、二重引用符で囲む必要があります。 表 21-13 に、インシデントの述語文字列で使用できるフィールドを表示します。 表 21-17 に、問題の述語文字列で使用できるフィールドを表示します。
ORDER BY field1, field2, ... [ASC|DSC|DESC]	フィールドを指定順序、昇順(ASC)および降順(DSC または DESC)でソートして、結果を表示します。ORDER BY 句を指定すると、結果はデフォルトの昇順で表示されます。
GROUP BY field1, field2, ...	指定したフィールドごとにグループ化された結果が表示されます。 GROUP BY フラグは行をグループ化しますが、結果セットの順序は保証しません。グループ化の順序付けを行うには、ORDER BY フラグを使用します。
HAVING "having_predicate_string"	返される行のグループを、HAVING の述語が true であるグループに制限します。HAVING フラグは、GROUP BY フラグと一緒に使用する必要があります。

ノート:

WHERE、ORDER BY、GROUP BY および HAVING フラグは、SELECT SQL 文の同名の句に似ています。SELECT SQL 文における句の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

制限事項

SELECTコマンドを使用する際は、次の制限事項があります。

- 結合できる表は、2つまでです。
- 表の別名は使用できません。
- 使用できる関数は、この項に示す一部の関数に限定されます。
- 表を結合するとき、またはGROUP BY句を使用するときに、ワイルドカード(*)は使用できません。
- 文を1つの行に収める必要があります。
- 文に副問合せを含めることはできません。

- 文にWITH句を含めることはできません。
- 一部の疑似列のみ、使用可能です。たとえば、ROWNUMは使用できますが、ROWIDは使用できません。

例

この例では、1を超えるincident_idを持つインシデントの、incident_idおよびcreate_timeを取得します。

```
select incident_id, create_time from incident where incident_id > 1
```

この問合せの出力例を次に示します。

```
INCIDENT_ID          CREATE_TIME
-------------------- ----------------------------------------
4801                 2011-05-27 10:10:26.541656 -07:00
4802                 2011-05-27 10:11:02.456066 -07:00
4803                 2011-05-27 10:11:04.759654 -07:00
```

この例では、problem_keyに600が含まれる各問題の、problem_idおよびfirst_incidentを取得します。

```
select problem_id, first_incident from problem where problem_key like '%600%'
```

この問合せの出力例を次に示します。

```
PROBLEM_ID           FIRST_INCIDENT
-------------------- --------------------
1                    4801
2                    4802
3                    4803
```

関数

このセクションでは、SELECTコマンドと一緒に使用できる関数について説明します。

これらの関数の目的および構文は、対応するSQL関数に似ていますが、一部において異なっています。この項では、ADRCIユーティリティで使用される関数とおよびSQL関数の間の相違点について説明します。

すべての関数に次の制限事項が適用されます。

- 式は、単純式である必要があります。単純式については、『Oracle Database SQL言語リファレンス』を参照してください。
- 複数の関数コールを組み合せることはできません。たとえば、次の関数コールの組合せはサポートされません。

  ```
  sum(length(column_name))
  ```

- 関数はオーバーロードされません。
- すべての関数の引数は必須です。
- 関数は、他のADRCIユーティリティ・コマンドと一緒に使用できません。

- AVG
- CONCAT
- COUNT
- DECODE

  DECODE関数は、1つの式を各検索値と1つずつ比較します。式が検索値に等しい場合、Oracle Databaseは対応する結果を返します。一致するものがない場合、Oracleは指定されたデフォルト値を返します。
- LENGTH

入力文字セットによる定義を使用して、文字列の長さを返します。

- MAX
  式の最大値を返します。
- MIN
  式の最小値を返します。
- NVL
  問合せの結果で、null (空白として返される)を文字データに置換します。
- REGEXP_LIKE
  指定の正規表現で指定したパターンに一致する行を返します。
- SUBSTR
  文字データの一部を返します。
- SUM
  式の値の合計を返します。
- TIMESTAMP_TO_CHAR
  TIMESTAMPデータ型の値を、指定した書式でVARCHAR2データ型の値に変換します。
- TOLOWER
  すべての文字を小文字にして、文字データを返します。
- TOUPPER
  すべての文字を大文字にして、文字データを返します。

親トピック: ADRCIコマンド・リファレンス

21.9.9.1 AVG

式の平均値を返します。

構文

『Oracle Database SQL言語リファレンス』を参照してください。

制限事項

SELECTコマンドでAVG関数を使用する場合は、次の制限事項が適用されます。

- 式は、数値型の列または正の数値定数である必要があります。
- この関数は、DISTINCTまたはALLキーワードをサポートしていません。
- この関数は、OVER句をサポートしていません。

親トピック: SELECT

21.9.9.2 CONCAT

2つの文字列を連結して返します。データ型がCHARおよびVARCHAR2の文字データを使用できます。戻り値は、文字データと同じデータ型です。

構文

『Oracle Database SQL言語リファレンス』を参照してください。

制限事項

SELECTコマンドでCONCAT関数を使用する場合は、次の制限事項が適用されます。

- この関数は、BLOB、CLOB、NCLOB、BFILEデータ型などのLOBデータ型をサポートしていません。
- この関数は、NCHAR、NVARCHAR2、NCLOBデータ型などの各国語文字セットのデータ型をサポートしていません。

親トピック: SELECT

21.9.9.3 COUNT

問合せによって返された行数を返します。

構文

『Oracle Database SQL言語リファレンス』を参照してください。

制限事項

SELECTコマンドでCOUNT関数を使用する場合は、次の制限事項が適用されます。

- この式は、列、数値定数または文字列定数である必要があります。
- この関数は、DISTINCTまたはALLキーワードをサポートしていません。
- この関数は、OVER句をサポートしていません。
- この関数は、通常、重複およびNULLを含めて問合せのすべての行をカウントします。

例

この例では、flood_controlledが0(ゼロ)のインシデントの数を返します。

```
select count(*) from incident where flood_controlled = 0;
```

この例では、problem_keyにORA-600が含まれる問題の数を返します。

```
select count(*) from problem where problem_key like '%ORA-600%';
```

親トピック: SELECT

21.9.9.4 DECODE

DECODE関数は、1つの式を各検索値と1つずつ比較します。式が検索値に等しい場合、Oracle Databaseは対応する結果を返します。一致するものがない場合、Oracleは指定されたデフォルト値を返します。

構文

『Oracle Database SQL言語リファレンス』を参照してください。

制限事項

SELECTコマンドでDECODE関数を使用する場合は、次の制限事項が適用されます。

- 検索引数は、文字データである必要があります。
- デフォルト値を指定する必要があります。

例

次の例では、各incident_idと、インシデントがフラッド制御されているかどうかを示します。この例では、DECODE関数を使用して、flood_controlledフィールドの数値のかわりにテキストを表示します。

```
select incident_id, decode(flood_controlled, 0, ¥
  "Not flood-controlled", "Flood-controlled") from incident;
```

親トピック: SELECT

21.9.9.5 LENGTH

入力文字セットによる定義を使用して、文字列の長さを返します。

用途

この文字列のデータ型には、CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB、NCLOBのいずれかを使用できます。戻り値のデータ型はNUMBERです。文字列のデータ型がCHARの場合、長さにはすべての後続の空白が含まれます。文字列がNULLの場合、この関数は0(ゼロ)を返します。

ノート:

文字列が NULL の場合、SQL 関数は NULL を戻します。

構文

『Oracle Database SQL言語リファレンス』を参照してください。

制限事項

SELECTコマンドは、関数LENGTHB、LENGTHC、LENGTH2およびLENGTH4をサポートしていません。

例

この例では、各問題のproblem_idおよびproblem_keyの長さを示します。

```
select problem_id, length(problem_key) from problem;
```

親トピック: SELECT

21.9.9.6 MAX

式の最大値を返します。

構文

『Oracle Database SQL言語リファレンス』を参照してください。

制限事項

SELECTコマンドでMAX関数を使用する場合は、次の制限事項が適用されます。

- この関数は、DISTINCTまたはALLキーワードをサポートしていません。
- この関数は、OVER句をサポートしていません。

例

この例では、すべての記録された問題の最大のlast_incident値を示します。

```
select max(last_incident) from problem;
```

親トピック: SELECT

21.9.9.7 MIN

式の最小値を返します。

構文

『Oracle Database SQL言語リファレンス』を参照してください。

制限事項

SELECTコマンドでMIN関数を使用する場合は、次の制限事項が適用されます。

- この関数は、DISTINCTまたはALLキーワードをサポートしていません。
- この関数は、OVER句をサポートしていません。

例

この例では、すべての記録された問題に関する最小のfirst_incident値を示します。

```
select min(first_incident) from problem;
```

親トピック: SELECT

21.9.9.8 NVL

問合せの結果において、null(空白として返される)を文字データに置換します。

用途

指定した最初の式がNULLの場合、NVLは指定した2番目の式を返します。指定した最初の式がNULLでない場合、NVLは最初の式の値を返します。

構文

『Oracle Database SQL言語リファレンス』を参照してください。

制限事項

SELECTコマンドでNVL関数を使用する場合は、次の制限事項が適用されます。

- 置換する値(2番目の式)は、文字データとして指定する必要があります。
- この関数は、データ変換をサポートしません。

例

この例では、signalling_componentの出力のNULLをテキスト"No component"に置換します。

```
select nvl(signalling_component, 'No component') from incident;
```

親トピック: SELECT

21.9.9.9 REGEXP_LIKE

指定した正規表現の指定パターンに一致する行を返します。

用途

SQLでは、REGEXP_LIKEは関数ではなく条件です。

構文

『Oracle Database SQL言語リファレンス』を参照してください。

制限事項

SELECTコマンドでREGEXP_LIKE関数を使用する場合は、次の制限事項が適用されます。

- パターン一致では、常に大/小文字が区別されます。
- この関数は、match_param引数をサポートしていません。

例

この例では、problem_keyが数値で終了するすべての問題のproblem_idおよびproblem_keyを表示します。

```
select problem_id, problem_key from problem ¥
  where regexp_like(problem_key, '[0-9]$') = true
```

親トピック: SELECT

21.9.9.10 SUBSTR

文字データの一部を返します。

用途

返されるデータ部分は、指定した部分から始まる、指定した文字長の部分文字列となります。SUBSTRは、入力文字セットによって定義された文字を使用して、長さを計算します。

構文

『Oracle Database SQL言語リファレンス』を参照してください。

制限事項

SELECTコマンドでSUBSTR関数を使用する場合は、次の制限事項が適用されます。

- この関数は、正の整数のみをサポートします。負の値または浮動小数点数については、サポートしていません。
- SELECTコマンドは、SUBSTRB、SUBSTRC、SUBSTR2およびSUBSTR4関数をサポートしていません。

例

この例では、各problem_keyの5文字目以降を表示します。

```
select substr(problem_key, 5) from problem;
```

親トピック: SELECT

21.9.9.11 SUM

式の値の合計を返します。

構文

『Oracle Database SQL言語リファレンス』を参照してください。

制限事項

SELECTコマンドでSUM関数を使用する場合は、次の制限事項が適用されます。

- この式は、数値型の列または数値定数である必要があります。
- この関数は、DISTINCTまたはALLキーワードをサポートしていません。
- この関数は、OVER句をサポートしていません。

親トピック: SELECT

21.9.9.12 TIMESTAMP_TO_CHAR

TIMESTAMPデータ型の値を、VARCHAR2データ型の値に、指定した書式で変換します。

用途

形式を指定しない場合、値が関数によってデフォルトのタイムスタンプ形式に変換されます。

構文

『Oracle Database SQL言語リファレンス』でTO_CHAR関数の構文を参照してください。

制限事項

SELECTコマンドでTIMESTAMP_TO_CHAR関数を使用する場合は、次の制限事項が適用されます。

- この関数は、TIMESTAMPデータ型のみを変換します。TIMESTAMP WITH TIME ZONE、TIMESTAMP WITH LOCAL TIME ZONEなどのデータ型はサポートされていません。
- この関数は、nlsparm引数をサポートしていません。この関数は、セッションのデフォルト言語を使用します。

例

この例では、各インシデントのcreate_timeを、TIMESTAMPデータ型からVARCHAR2データ型にDD-MON-YYYY形式で変換します。

```
select timestamp_to_char(create_time, 'DD-MON-YYYY') from incident;
```

親トピック: SELECT

21.9.9.13 TOLOWER

すべての文字を小文字にして、文字データを返します。

用途

データ型がCHARおよびVARCHAR2の文字データを使用できます。戻り値は、文字データと同じデータ型です。データベースは、基礎となる文字セットに対して定義したバイナリ・マッピングに基づいて文字の形式を設定します。

構文

『Oracle Database SQL言語リファレンス』でLOWER関数の構文を参照してください。

制限事項

SELECTコマンドでTOLOWER関数を使用する場合は、次の制限事項が適用されます。

- この関数は、BLOB、CLOB、NCLOB、BFILEデータ型などのLOBデータ型をサポートしていません。
- この関数は、NCHAR、NVARCHAR2、NCLOBデータ型などの各国語文字セットのデータ型をサポートしていません。

例

この例では、各problem_keyをすべて小文字で表示します。

```
select tolower(problem_key) from problem;
```

親トピック: SELECT

21.9.9.14 TOUPPER

すべての文字を大文字にして、文字データを返します。

用途

データ型がCHARおよびVARCHAR2の文字データを使用できます。戻り値は、文字データと同じデータ型です。データベースは、基礎となる文字セットに対して定義したバイナリ・マッピングに基づいて文字の形式を設定します。

構文

『Oracle Database SQL言語リファレンス』でUPPER関数の構文を参照してください。

制限事項

SELECTコマンドでTOUPPER関数を使用する場合は、次の制限事項が適用されます。

- この関数は、BLOB、CLOB、NCLOB、BFILEデータ型などのLOBデータ型をサポートしていません。
- この関数は、NCHAR、NVARCHAR2、NCLOBデータ型などの各国語文字セットのデータ型をサポートしていません。

例

この例では、各problem_keyをすべて大文字で表示します。

```
select toupper(problem_key) from problem;
```

親トピック: SELECT

21.9.10 SET BASE

カレントADRCIセッションで使用するADRベースを設定します。

構文および説明

set base base_str

base_strは、ディレクトリへのフルパスです。base_strの形式は、オペレーティング・システムによって異なります。ベース・ディレクトリ下に有効なADRホームが存在する場合、これらのホームはカレントADRCIセッションのホームパスに追加されます。

このコマンドを使用する前に、ADRホームを設定しておく必要はありません。

例

```
set base /u01/app/oracle
```

関連項目:

ADRベース

親トピック: ADRCIコマンド・リファレンス

21.9.11 SET BROWSER

レポートを表示するデフォルトのブラウザを設定します。

ノート:

このコマンドは今後使用する目的で確保されています。現時点では、ADRCI はブラウザで HTML 形式のレポートをサポートしていません。

構文および説明

`set browser browser_program`

`browser_program`は、ブラウザのプログラム名です(ブラウザはカレントのADR作業ディレクトリから起動可能であると想定されています)。ブラウザが設定されていない場合、ADRCIはレポートを端末またはスプール・ファイルに表示します。

このコマンドを使用する前に、ADRホームを設定しておく必要はありません。

例

```
set browser mozilla
```

関連項目:

- レポート表示の詳細は、「SHOW REPORT」を参照してください
- スプールの詳細は、「SPOOL」を参照してください

親トピック: ADRCIコマンド・リファレンス

21.9.12 SET CONTROL

ADRの内容の削除ポリシーを設定します。

構文および説明

`set control (purge_policy = value, ...)`

`purge_policy`は、`SHORTP_POLICY`または`LONGP_POLICY`のいずれかです。

関連項目:

詳細は、「SHOW CONTROL」を参照してください。

`value`は、ADRの内容が削除可能になるまでの時間数です。

`SHORTP_POLICY`および`LONGP_POLICY`は相互に排他的ではありません。各ポリシーは、異なるタイプの内容を制御します。

このコマンドは、1つのADRホームでのみ機能します。

例

```
set control (SHORTP_POLICY = 360)
```

親トピック: ADRCIコマンド・リファレンス

21.9.13 SET ECHO

コマンド出力をオンまたはオフにします。このコマンドは、スクリプトに表示される出力またはスプール・モードを使用する出力のみに影響します。

構文および説明

SET ECHO ON | OFF

このコマンドを使用する前に、ADRホームを設定しておく必要はありません。

例

```
SET ECHO OFF
```

関連項目:

スプールの詳細は、「SPOOL」を参照してください

親トピック: ADRCIコマンド・リファレンス

21.9.14 SET EDITOR

アラート・ログおよびトレース・ファイルの内容を表示するためのエディタを設定します。

構文および説明

```
SET EDITOR editor_program
```

editor_programは、エディタのプログラム名です。エディタが設定されていない場合、ADRCIはオペレーティング・システムの環境変数EDITORで指定されているエディタを使用します。EDITORが設定されていない場合、ADRCIはデフォルトのエディタとしてviを使用します。

このコマンドを使用する前に、ADRホームを設定しておく必要はありません。

例

```
SET EDITOR xemacs
```

親トピック: ADRCIコマンド・リファレンス

21.9.15 SET HOMEPATH

1つ以上のADRホームをカレントにします。多くのADRコマンドが、カレントADRホームでのみ動作します。

構文および説明

```
SET HOMEPAGH homepagh_str1 homepath_str2 ...
```

homepath_strn文字列は、カレントADRベースに対して相対的なADRホームのパスです。ディレクトリ名diagはパスから省略可能です。指定したパスに複数のADRホームが含まれている場合は、すべてのホームがホームパスに追加されます。

目的とする新しいADRホームがカレントADRベース内に存在しない場合は、SET BASEを使用して新しいADRベースを設定してから、SET HOMEPATHを使用します。

このコマンドを使用する前に、ADRホームを設定しておく必要はありません。

例

```
SET HOMEPATH diag/rdbms/orcldw/orcldw1 diag/rdbms/orcldw/orcldw2
```

次のコマンドにより、前述の例と同一のホームパスを設定できます。

```
SET HOMEPATH rdbms/orcldw/orcldw1 rdbms/orcldw/orcldw2
```

関連項目:

ホームパス

親トピック: ADRCIコマンド・リファレンス

21.9.16 SET TERMOUT

端末への出力をオンまたはオフにします。

構文および説明

```
SET TERMOUT ON | OFF
```

この設定はスプールとは関係なく使用できます。つまり、出力は端末とファイル両方に同時に送信できます。

このコマンドを使用する前に、ADRホームを設定しておく必要はありません。

関連項目:

スプールの詳細は、「SPOOL」を参照してください

例

```
SET TERMOUT ON
```

親トピック: ADRCIコマンド・リファレンス

21.9.17 SHOW ALERT

用途

デフォルトのエディタにアラート・ログの内容を表示します。

構文および説明

```
show alert [-p "predicate_string"] [-tail [num] [-f]] [-term]
  [-file alert_file_name]
```

`-term`フラグを使用している場合を除き、このコマンドは1つのカレントADRホームでのみ動作します。複数のADRホームが設定されている場合、ADRCIによって、使用するADRホームを選択するように要求されます。

表21-9 SHOW ALERTコマンドのフラグ

フラグ	説明
`-p "predicate_string"`	SQL に類似した述語文字列を使用して、述語が true であるアラート・ログのエントリのみを表示します。述語文字列は、二重引用符で囲む必要があります。 表 21-10 に、述語文字列で使用できるフィールドを表示します。

フラグ	説明
-tail [num] [-f]	アラート・ログの最新のエントリを表示します。 num オプションを使用して、アラート・ログで最新の num 個のエントリを表示します。num が省略されている場合は、最新の 10 個のエントリが表示されます。 -f オプションが指定されている場合は、要求されたメッセージを表示した後に、コマンドが戻りません。かわりに、コマンドがアクティブのまま残り、アラート・ログに新しいエントリが到着すると、引き続きそのエントリを端末に表示します。このコマンドを使用すると、アラート・ログのライブ監視を実行できます。このコマンドを終了するには、[Ctrl]を押しながら[C]を押します。
-term	結果を端末に送信します。すべてのカレント ADR ホームからアラート・ログ全体を順次出力します。このオプションが指定されていない場合、結果はデフォルトのエディタに表示されます。
-file alert_file_name	ADR の外のアラート・ファイルを指定できます。alert_file_name は、フルパス名で指定する必要があります。このオプションは、-tail オプションとともに使用できません。

表21-10 SHOW ALERTのアラート・フィールド

フィールド	型
ORIGINATING_TIMESTAMP	timestamp
NORMALIZED_TIMESTAMP	タイムスタンプ
ORGANIZATION_ID	text(65)
COMPONENT_ID	テキスト(65)
HOST_ID	テキスト(65)
HOST_ADDRESS	text(17)
MESSAGE_TYPE	number
MESSAGE_LEVEL	数値
MESSAGE_ID	テキスト(65)
MESSAGE_GROUP	テキスト(65)

フィールド	型
CLIENT_ID	テキスト(65)
MODULE_ID	テキスト(65)
PROCESS_ID	text(33)
THREAD_ID	テキスト(65)
USER_ID	テキスト(65)
INSTANCE_ID	テキスト(65)
DETAILED_LOCATION	text(161)
UPSTREAM_COMP_ID	text(101)
DOWNSTREAM_COMP_ID	テキスト(101)
EXECUTION_CONTEXT_ID	テキスト(101)
EXECUTION_CONTEXT_SEQUENCE	数値
ERROR_INSTANCE_ID	数値
ERROR_INSTANCE_SEQUENCE	数値
MESSAGE_TEXT	text(2049)
MESSAGE_ARGUMENTS	text(129)
SUPPLEMENTAL_ATTRIBUTES	テキスト(129)
SUPPLEMENTAL_DETAILS	text(4000)
PROBLEM_KEY	テキスト(65)

例

この例では、カレントADRホームのすべてのアラート・メッセージをデフォルトのエディタに表示します。

```
show alert
```

この例では、カレントADRホームのすべてのアラート・メッセージを表示し、デフォルトのエディタではなく端末に出力を送信します。

```
show alert -term
```

この例では、カレントADRホームのすべてのアラート・メッセージを、インシデントを示すメッセージ・テキストとともに表示します。

```
show alert -p "message_text like '%incident%'"
```

この例では、最新の20個のアラート・メッセージを表示してから、アラート・ログをオープンしたままにし、新しいアラート・ログのエントリが到着すると、それらのエントリを表示します。

```
show alert -tail 20 -f
```

この例では、複数のADRホームが設定されているときに、1つのカレントADRホームのすべてのアラート・メッセージをデフォルトのエディタに表示します。

```
show alert
Choose the alert log from the following homes to view:
1: diag/tnslsnr/dbhost1/listener
2: diag/asm/+asm/+ASM
3: diag/rdbms/orcl/orcl
4: diag/clients/user_oracle/host_9999999999_11
Q: to quit
Please select option:
3
```

関連項目:

「SET EDITOR」

親トピック: ADRCIコマンド・リファレンス

21.9.18 SHOW BASE

カレントADRベースを示します。

構文および説明

`SHOW BASE [-product product_name]`

特定の製品では、オプションで、製品のADRベースの場所を表示できます。現在サポートされている製品は、`CLIENT`と`ADRCI`です。

このコマンドを使用する前に、ADRホームを設定しておく必要はありません。

例

次の例では、カレントADRベースを示します。

```
SHOW BASE
```

出力:

```
ADR base is "/u01/app/oracle"
```

次の例では、Oracle Databaseクライアントのカレント ADRベースを示します。

```
SHOW BASE -product client
```

親トピック: ADRCIコマンド・リファレンス

21.9.19 SHOW CONTROL

削除ポリシー属性を含む、ADRについての情報を示します。

構文および説明

```
SHOW CONTROL
```

次の削除ポリシー属性を含む、ADRの各種属性を示します。

属性名	説明
SHORTP_POLICY	存続期間の短い ADR の内容が削除可能になるまでの時間数。デフォルト値は 720 (30 日)です。 0(ゼロ)を設定すると、存続期間の短い内容をすべて削除できます。設定の最大値は、35791394 です。35791394 を超える値を指定すると、この属性は 0(ゼロ)に設定されます。 存続期間の短い ADR の内容を次に示します。 ● cdmp_timestamp サブディレクトリに格納済のこれらのファイルを含む、トレース・ファイル ● コア・ダンプ・ファイル ● パッケージ情報
LONGP_POLICY	存続期間の長い ADR の内容が削除可能になるまでの時間数。デフォルト値は 8760 (365 日)です。 0(ゼロ)を設定すると、存続期間の長い内容をすべて削除できます。設定の最大値は、35791394 です。35791394 を超える値を指定すると、この属性は 0(ゼロ)に設定されます。 存続期間の長い ADR の内容を次に示します。 ● インシデント情報 ● インシデント・ダンプ ● アラート・ログ

ノート:

SHORTP_POLICY および LONGP_POLICY 属性は相互に排他的ではありません。各ポリシーは、異なるタイプの内容を制御します。

21.9.20 SHOW HM_RUN

用途

状態モニターの実行に関するすべての情報を表示します。

構文および説明

```
show hm_run [-p "predicate_string"]
```

`predicate_string`は、選択するフィールド名を指定するSQLに類似した述語です。表21-11に、使用可能なフィールド名のリストを示します。

表21-11 状態モニターの実行に使用するフィールド

フィールド	タイプ
RUN_ID	数値
RUN_NAME	text(31)
CHECK_NAME	テキスト(31)
NAME_ID	数値
MODE	数値
START_TIME	タイムスタンプ
RESUME_TIME	タイムスタンプ
END_TIME	タイムスタンプ
MODIFIED_TIME	タイムスタンプ
TIMEOUT	数値
FLAGS	数値
STATUS	数値
SRC_INCIDENT_ID	数値
NUM_INCIDENTS	数値
ERR_NUMBER	数値

フィールド	タイプ
REPORT_FILE	bfile

例

次の例では、すべての状態モニターの実行に関するデータを示します。

```
show hm_run
```

次の例では、123のIDを持つ状態モニターの実行に関するデータを示します。

```
show hm_run -p "run_id=123"
```

関連項目:

状態モニターの詳細は、『Oracle Database管理者ガイド』を参照してください。

親トピック: ADRCIコマンド・リファレンス

21.9.21 SHOW HOMEPATH

SHOW HOMEPATHコマンドは、SHOW HOMESコマンドと同じです。

構文および説明

SHOW HOMEPATH | SHOW HOMES | SHOW HOME

このコマンドを使用する前に、ADRホームを設定しておく必要はありません。

例

```
SHOW HOMEPATH
```

出力:

```
ADR Homes:
diag/tnslsnr/dbhost1/listener
diag/asm/+asm/+ASM
diag/rdbms/orcl/orcl
diag/clients/user_oracle/host_9999999999_11
```

関連項目:

ホームパスの設定方法の詳細は、「SET HOMEPATH」を参照してください

親トピック: ADRCIコマンド・リファレンス

21.9.22 SHOW HOMES

カレントADRCIセッションのADRホームを表示します。

構文および説明

SHOW HOMES | SHOW HOME | SHOW HOMEPATH

このコマンドを使用する前に、ADRホームを設定しておく必要はありません。

例

```
SHOW HOMES
```

出力:

```
ADR Homes:
diag/tnslsnr/dbhost1/listener
diag/asm/+asm/+ASM
diag/rdbms/orcl/orcl
diag/clients/user_oracle/host_9999999999_11
```

親トピック: ADRCIコマンド・リファレンス

21.9.23 SHOW INCDIR

用途

指定したインシデントのトレース・ファイルを表示します。

構文および説明

`show incdir [id | id_low id_high]`

1つのインシデントID(`id`)またはインシデントの範囲(`id_low`から`id_high`)を指定できます。インシデントIDが指定されていない場合は、すべてのインシデントのトレース・ファイルが表示されます。

例

この例では、すべてのインシデントのすべてのトレース・ファイルを表示します。

```
show incdir
```

出力:

```
ADR Home = /u01/app/oracle/log/diag/rdbms/emdb/emdb:
*********************************************************************
diag/rdbms/emdb/emdb/incident/incdir_3801/emdb_ora_23604_i3801.trc
diag/rdbms/emdb/emdb/incident/incdir_3801/emdb_m000_23649_i3801_a.trc
diag/rdbms/emdb/emdb/incident/incdir_3802/emdb_ora_23604_i3802.trc
diag/rdbms/emdb/emdb/incident/incdir_3803/emdb_ora_23604_i3803.trc
diag/rdbms/emdb/emdb/incident/incdir_3804/emdb_ora_23604_i3804.trc
diag/rdbms/emdb/emdb/incident/incdir_3805/emdb_ora_23716_i3805.trc
diag/rdbms/emdb/emdb/incident/incdir_3805/emdb_m000_23767_i3805_a.trc
diag/rdbms/emdb/emdb/incident/incdir_3806/emdb_ora_23716_i3806.trc
diag/rdbms/emdb/emdb/incident/incdir_3633/emdb_pmon_28970_i3633.trc
diag/rdbms/emdb/emdb/incident/incdir_3633/emdb_m000_23778_i3633_a.trc
diag/rdbms/emdb/emdb/incident/incdir_3713/emdb_smon_28994_i3713.trc
diag/rdbms/emdb/emdb/incident/incdir_3713/emdb_m000_23797_i3713_a.trc
diag/rdbms/emdb/emdb/incident/incdir_3807/emdb_ora_23783_i3807.trc
diag/rdbms/emdb/emdb/incident/incdir_3807/emdb_m000_23803_i3807_a.trc
diag/rdbms/emdb/emdb/incident/incdir_3808/emdb_ora_23783_i3808.trc
```

この例では、インシデント3713のすべてのトレース・ファイルを表示します。

```
show incdir 3713
```

出力:

```
ADR Home = /u01/app/oracle/log/diag/rdbms/emdb/emdb:
*********************************************************************
diag/rdbms/emdb/emdb/incident/incdir_3713/emdb_smon_28994_i3713.trc
diag/rdbms/emdb/emdb/incident/incdir_3713/emdb_m000_23797_i3713_a.trc
```

次の例では、3801から3804の間のインシデントのすべてのトレース・ファイルを表示します。

```
show incdir 3801 3804
```

出力:

```
ADR Home = /u01/app/oracle/log/diag/rdbms/emdb/emdb:
*************************************************************************
diag/rdbms/emdb/emdb/incident/incdir_3801/emdb_ora_23604_i3801.trc
diag/rdbms/emdb/emdb/incident/incdir_3801/emdb_m000_23649_i3801_a.trc
diag/rdbms/emdb/emdb/incident/incdir_3802/emdb_ora_23604_i3802.trc
diag/rdbms/emdb/emdb/incident/incdir_3803/emdb_ora_23604_i3803.trc
diag/rdbms/emdb/emdb/incident/incdir_3804/emdb_ora_23604_i3804.trc
```

親トピック: ADRCIコマンド・リファレンス

21.9.24 SHOW INCIDENT

用途

カレントADRホームに関連付けられているすべてのインシデントを表示します。未解決のインシデントとクローズしたインシデントの両方を含めます。

構文および説明

```
show incident [-p "predicate_string"] [-mode {BASIC|BRIEF|DETAIL}]          [-orderby
field1, field2, ... [ASC|DSC]]
```

表21-12に、SHOW INCIDENTのフラグを示します。

表21-12 SHOW INCIDENTコマンドのフラグ

フラグ	説明
-p "predicate_string"	述語文字列を使用して、述語が true であるインシデントのみを示します。述語文字列は、二重引用符で囲む必要があります。 表 21-13 に、述語文字列で使用できるフィールドを表示します。
-mode {BASIC|BRIEF|DETAIL}	インシデントの出力モードを選択します。BASIC はデフォルトです。 ● BASIC は、基本的なインシデント情報(INCIDENT_ID、PROBLEM_ID および CREATE_TIME の各フィールド)のみを表示します。フラッド制御されているインシデントは表示されません。 ● BRIEF は、表 21-13 で説明されているフィールドの指定に従って、インシデントに関連するすべての情報を表示します。フラッド制御されているインシデントも含みます。 ● DETAIL は、(BRIEF モードを使用した場合と同じ)インシデントに関連するすべての情報と、インシデント・ダンプに関する情報を表示します。フラッド制御されているインシデントも含

フラグ	説明
	みます。
-orderby field1, field2, ... [ASC|DSC]	結果を指定した順序のフィールドでソートするだけでなく、昇順(ASC)および降順(DSC)でソートして表示します。デフォルトでは、結果は昇順で表示されます。

表21-13 SHOW INCIDENTのインシデント・フィールド

フィールド	タイプ	説明
INCIDENT_ID	数値	インシデントの ID
PROBLEM_ID	数値	インシデントが属する問題の ID
CREATE_TIME	タイムスタンプ	インシデントが作成された時間
CLOSE_TIME	タイムスタンプ	インシデントがクローズされた時間
STATUS	数値	このインシデントの状態
FLAGS	数値	内部使用のフラグ
FLOOD_CONTROLLED	number(ADRCI によりテキスト・ステータスにデコードされる)	インシデントのフラッド制御の状態をエンコード
ERROR_FACILITY	text(10)	インシデントの原因となったエラーのエラー機能
ERROR_NUMBER	数値	インシデントの原因となったエラーのエラー番号
ERROR_ARG1	text(64)	インシデントの原因となったエラーに関する最初の引数 エラーの引数では、エラーを発行したコードの位置など、エラーの追加情報が提供されます。
ERROR_ARG2	テキスト(64)	インシデントの原因となったエラーに関する 2 番目の引数
ERROR_ARG3	テキスト(64)	インシデントの原因となったエラーに関する 3 番目

フィールド	タイプ	説明
		の引数
ERROR_ARG4	テキスト(64)	インシデントの原因となったエラーに関する 4 番目の引数
ERROR_ARG5	テキスト(64)	インシデントの原因となったエラーに関する 5 番目の引数
ERROR_ARG6	テキスト(64)	インシデントの原因となったエラーに関する 6 番目の引数
ERROR_ARG7	テキスト(64)	インシデントの原因となったエラーに関する 7 番目の引数
ERROR_ARG8	テキスト(64)	インシデントの原因となったエラーに関する 8 番目の引数
SIGNALLING_COMPONENT	テキスト(64)	インシデントの原因となったエラー信号を出したコンポーネント
SIGNALLING_SUBCOMPONENT	テキスト(64)	インシデントの原因となったエラー信号を出したサブコンポーネント
SUSPECT_COMPONENT	テキスト(64)	インシデントの原因の可能性があると自動的に識別されたコンポーネント
SUSPECT_SUBCOMPONENT	テキスト(64)	インシデントの原因の可能性があると自動的に識別されたサブコンポーネント
ECID	テキスト(64)	実行コンテキスト ID
IMPACT	数値	インシデントの影響のエンコード化
ERROR_ARG9	テキスト(64)	インシデントの原因となったエラーに関する 9 番目の引数
ERROR_ARG10	テキスト(64)	インシデントの原因となったエラーに関する 10 番目の引数

フィールド	タイプ	説明
ERROR_ARG11	テキスト(64)	インシデントの原因となったエラーに関する 11 番目の引数
ERROR_ARG12	テキスト(64)	インシデントの原因となったエラーに関する 12 番目の引数

例

次の例では、このADRホームのすべてのインシデントを示します。

```
show incident
```

出力:

```
ADR Home = /u01/app/oracle/log/diag/rdbms/emdb/emdb:
*********************************************************************
INCIDENT_ID          PROBLEM_KEY                                       CREATE_TIME
-------------------- ------------------------------------------- -------------------
---------
3808                 ORA 603                                           2010-06-18
21:35:49.322161 -07:00
3807                 ORA 600 [4137]                                    2010-06-18
21:35:47.862114 -07:00
3806                 ORA 603                                           2010-06-18
21:35:26.666485 -07:00
3805                 ORA 600 [4136]                                    2010-06-18
21:35:25.012579 -07:00
3804                 ORA 1578                                          2010-06-18
21:35:08.483156 -07:00
3713                 ORA 600 [4136]                                    2010-06-18
21:35:44.754442 -07:00
3633                 ORA 600 [4136]                                    2010-06-18
21:35:35.776151 -07:00
7 rows fetched
```

次の例では、インシデント3805の詳細を示します。

```
adrci> show incident -mode DETAIL -p "incident_id=3805"
```

出力:

```
ADR Home = /u01/app/oracle/log/diag/rdbms/emdb/emdb:
*********************************************************************

*************************************************************
INCIDENT INFO RECORD 1
*************************************************************
   INCIDENT_ID                   3805
   STATUS                        closed
   CREATE_TIME                   2010-06-18 21:35:25.012579 -07:00
   PROBLEM_ID                    2
   CLOSE_TIME                    2010-06-18 22:26:54.143537 -07:00
   FLOOD_CONTROLLED              none
   ERROR_FACILITY                ORA
   ERROR_NUMBER                  600
   ERROR_ARG1                    4136
   ERROR_ARG2                    2
   ERROR_ARG3                    18.0.628
   ERROR_ARG4                    <NULL>
```

```
   ERROR_ARG5                      <NULL>
   ERROR_ARG6                      <NULL>
   ERROR_ARG7                      <NULL>
   ERROR_ARG8                      <NULL>
   SIGNALLING_COMPONENT            <NULL>
   SIGNALLING_SUBCOMPONENT         <NULL>
   SUSPECT_COMPONENT               <NULL>
   SUSPECT_SUBCOMPONENT            <NULL>
   ECID                            <NULL>
   IMPACTS                         0
   PROBLEM_KEY                     ORA 600 [4136]
   FIRST_INCIDENT                  3805
   FIRSTINC_TIME                   2010-06-18 21:35:25.012579 -07:00
   LAST_INCIDENT                   3713
   LASTINC_TIME                    2010-06-18 21:35:44.754442 -07:00
   IMPACT1                         0
   IMPACT2                         0
   IMPACT3                         0
   IMPACT4                         0
   KEY_NAME                        Client ProcId
   KEY_VALUE                       oracle@dbhost1 (TNS V1-V3).23716_3083142848
   KEY_NAME                        SID
   KEY_VALUE                       127.52237
   KEY_NAME                        ProcId
   KEY_VALUE                       23.90
   KEY_NAME                        PQ
   KEY_VALUE                       (0, 1182227717)
   OWNER_ID                        1
   INCIDENT_FILE
/.../emdb/emdb/incident/incdir_3805/emdb_ora_23716_i3805.trc
   OWNER_ID                        1
   INCIDENT_FILE                   /.../emdb/emdb/trace/emdb_ora_23716.trc
   OWNER_ID                        1
   INCIDENT_FILE
/.../emdb/emdb/incident/incdir_3805/emdb_m000_23767_i3805_a.trc
1 rows fetched
```

親トピック: ADRCIコマンド・リファレンス

21.9.25 SHOW LOG

診断ログ・メッセージを表示します。

構文および説明

```
SHOW LOG [-l log_name] [-p "predicate_string"] [-term] [ [-tail [num] [-f]] ]
```

表21-14に、SHOW LOGのフラグを示します。

表21-14 SHOW LOGコマンドのフラグ

フラグ	説明
-l log_name	表示するログの名前。 ログ名を指定しない場合、このコマンドによって、カレント ADR ホーム以下のすべての診断ログに含まれるすべてのメッセージが表示されます。
-p "predicate_string"	SQL に類似した述語文字列を使用して、述語が true であるログのエントリのみを表

フラグ	説明
	示します。述語文字列は、二重引用符で囲む必要があります。 表 21-15 に、述語文字列で使用できるフィールドを表示します。
-term	結果を端末に送信します。 このオプションを指定しない場合、このコマンドによってエディタに結果が表示されます。デフォルトでは、結果は Emacs に表示されますが、SET EDITOR コマンドを使用して別のエディタに結果を表示することもできます。
-tail [num] [-f]	ログの最新のエントリを表示します。 num オプションを使用して、ログで最新の num 個のエントリを表示します。num が省略されている場合は、最新の 10 個のエントリが表示されます。 -f オプションが指定されている場合は、要求されたメッセージを表示した後に、コマンドが戻りません。かわりに、コマンドがアクティブのまま残り、ログに新しいエントリが到着すると、引き続きそのエントリを端末に表示します。このコマンドを使用すると、ログのライブ監視を実行できます。このコマンドを終了するには、[Ctrl]を押しながら[C]を押します。

表21-15 SHOW LOGのログ・フィールド

フィールド	タイプ
ORIGINATING_TIMESTAMP	タイムスタンプ
NORMALIZED_TIMESTAMP	タイムスタンプ
ORGANIZATION_ID	テキスト(65)
COMPONENT_ID	テキスト(65)
HOST_ID	テキスト(65)
HOST_ADDRESS	テキスト(17)
MESSAGE_TYPE	数値
MESSAGE_LEVEL	数値
MESSAGE_ID	テキスト(65)

フィールド	タイプ
MESSAGE_GROUP	テキスト(65)
CLIENT_ID	テキスト(65)
MODULE_ID	テキスト(65)
PROCESS_ID	テキスト(33)
THREAD_ID	テキスト(65)
USER_ID	テキスト(65)
INSTANCE_ID	テキスト(65)
DETAILED_LOCATION	テキスト(161)
UPSTREAM_COMP_ID	テキスト(101)
DOWNSTREAM_COMP_ID	テキスト(101)
EXECUTION_CONTEXT_ID	テキスト(101)
EXECUTION_CONTEXT_SEQUENCE	数値
ERROR_INSTANCE_ID	数値
ERROR_INSTANCE_SEQUENCE	数値
MESSAGE_TEXT	テキスト(2049)
MESSAGE_ARGUMENTS	テキスト(129)
SUPPLEMENTAL_ATTRIBUTES	テキスト(129)
SUPPLEMENTAL_DETAILS	テキスト(4000)
PROBLEM_KEY	テキスト(65)

親トピック: ADRCIコマンド・リファレンス

21.9.26 SHOW PROBLEM

用途

カレントADRホームの問題情報を示します。

構文および説明

```
show problem [-p "predicate_string"] [-last num | -all]
    [-orderby field1, field2, ... [ASC|DSC]]
```

次の表に、SHOW PROBLEMのフラグを示します。

表21-16 SHOW PROBLEMコマンドのフラグ

フラグ	説明
-p "predicate_string"	SQL に類似した述語文字列を使用して、述語が true であるインシデントのみを示します。述語文字列は、二重引用符で囲む必要があります。 表「SHOW PROBLEM の問題フィールド」に、述語文字列で使用できるフィールドを示します。
-last num | -all	最新の num 個の問題を示すか、またはすべての問題を示します。デフォルトでは、SHOW PROBLEM は最新の 50 個の問題を示します。
-orderby field1, field2, ... [ASC|DSC]	結果を指定した順序のフィールド(field1、field2、...)でソートするだけでなく、昇順(ASC)および降順(DSC)でソートして表示します。デフォルトでは、結果は昇順で表示されます。

表21-17 SHOW PROBLEMの問題フィールド

フィールド	タイプ	説明
PROBLEM_ID	数値	問題の ID
PROBLEM_KEY	text(550)	問題の問題キー
FIRST_INCIDENT	数値	問題の最初のインシデントのインシデント ID
FIRSTINC_TIME	タイムスタンプ	問題の最初のインシデントの作成時間
LAST_INCIDENT	数値	問題の最後のインシデントのインシデント ID
LASTINC_TIME	タイムスタンプ	問題の最後のインシデントの作成時間
IMPACT1	数値	この問題の影響のエンコード化
IMPACT2	数値	この問題の影響のエンコード化

フィールド	タイプ	説明
IMPACT3	数値	この問題の影響のエンコード化
IMPACT4	数値	この問題の影響のエンコード化
SERVICE_REQUEST	テキスト(64)	問題のサービス・リクエスト(サポート・ワークベンチで入力)
BUG_NUMBER	テキスト(64)	問題のバグ番号(サポート・ワークベンチで入力)

例

次の例では、カレントADRホームのすべての問題を表示します。

```
show problem -all
```

次の例では、IDが4の問題を示します。

```
show problem -p "problem_id=4"
```

親トピック: ADRCIコマンド・リファレンス

21.9.27 SHOW REPORT

指定したレポート・タイプおよび実行名のレポートを表示します。

用途

現在は、hm_run(状態モニター)レポート・タイプのみが、XML形式のみでサポートされています。HTML形式の状態モニター・レポートを表示するには、Oracle Enterprise ManagerまたはDBMS_HM PL/SQLパッケージを使用します。詳細は、『Oracle Database管理者ガイド』を参照してください。

構文および説明

```
SHOW REPORT report_type run_name
```

report_typeはhm_runである必要があります。run_nameは、レポートの作成元となる状態モニターの実行名です。CREATE REPORTコマンドを使用して、最初にレポートを作成する必要があります。

このコマンドを使用する前に、ADRホームを設定しておく必要はありません。

例

```
SHOW REPORT hm_run hm_run_1421
```

関連項目:

- CREATE REPORT
- SHOW HM_RUN

親トピック: ADRCIコマンド・リファレンス

21.9.28 SHOW TRACEFILE

用途

トレース・ファイルを表示します。

構文および説明

```
show tracefile [file1 file2 ...] [-rt | -t]
  [-i inc1 inc2 ...] [-path path1 path2 ...]
```

このコマンドは、-iまたは-pathフラグが指定されていないかぎり、カレントADRホームのトレース・ディレクトリおよびすべてのインシデント・ディレクトリ下で1つ以上のファイルを検索します。

このコマンドの使用時に-iオプションを指定しないかぎり、ADRホームを設定しておく必要はありません。

表21-18に、SHOW TRACEFILEの引数を示します。

表21-18 SHOW TRACEFILEコマンドの引数

引数	説明
file1 file2 ...	ファイル名で結果をフィルタ処理します。%記号はワイルドカード文字です。

表21-19 SHOW TRACEFILEコマンドのフラグ

フラグ	説明
-rt | -t	トレース・ファイル名をタイムスタンプに従って並べ替えます。-t を指定すると、ファイル名がタイムスタンプの昇順でソートされ、-rt を指定すると、ファイル名がその逆の順序でソートされます。ファイル名の並べ替えは、ファイルのディレクトリを基準としてのみ行われます。トレース・ファイルの複数のディレクトリを表示すると、各ディレクトリに個別に順序付けが行われます。 このオプションを使用すると、タイムスタンプが各ファイル名の横に表示されます。
-i inc1 inc2 ...	指定したインシデント ID に対して作成されたトレース・ファイルのみを選択します。
-path path1 path2 ...	指定したパス名下のトレース・ファイルのみを問い合せます。

例

次の例では、カレントADRホーム下のすべてのトレース・ファイルを示します。

```
show tracefile
```

次の例では、すべてのmmonトレース・ファイルを、タイムスタンプに従って降順で表示します。

```
show tracefile %mmon% -rt
```

次の例では、/home/steve/temp下にある、インシデント1および4に対するすべてのトレース・ファイルを示します。

```
show tracefile -i 1 4 -path /home/steve/temp
```

親トピック: ADRCIコマンド・リファレンス

21.9.29 SPOOL

ADRCI出力をファイルへ送信します。

構文および説明

```
SPOOL filename [[APPEND] | [OFF]]
```

filenameは、出力が送信されるファイル名です。フルパス名が指定されていない場合、ファイルはカレントADRCI作業ディレクトリに作成されます。ファイル拡張子が指定されていない場合、デフォルトの拡張子.adoが使用されます。APPENDを指定すると、出力がファイルの末尾に追加されます。指定されていない場合は、ファイルが上書きされます。スプールをオフにするにはOFFを使用します。

このコマンドを使用する前に、ADRホームを設定しておく必要はありません。

例

```
SPOOL myfile

SPOOL myfile.ado APPEND

SPOOL OFF

SPOOL
```

親トピック: ADRCIコマンド・リファレンス

21.10 ADRCIのトラブルシューティング

トラブルシューティングを支援するため、一般的なADRCIエラー・メッセージの一部と、考えられる原因および処置を確認します。

ADR ベースが設定されていません

原因: ORACLE_HOME 環境変数に NULL または無効な変数を指定して ADRCI を起動した可能性があります。

処理: ADRCI を終了し、ORACLE_HOME 環境変数を設定してから ADRCI を再起動します。詳細は、Oracle Database ADRC の定義の ADR ベースを参照してください。

DIA-48323: 指定されたパス名[string]は現行 ADR ホームの内部に存在する必要があります

原因: ADR ホームの外のファイルは、このコマンドではインシデント・ファイルとして使用できません。

処理: ADR ホーム内のインシデント・ファイルを使用して再試行します。

DIA-48400: ADRCI の初期化に失敗しました

原因: ADR ベース・ディレクトリが存在しません。

処理: DIAGNOSTIC_DEST 初期化パラメータの値をチェックし、1 つ以上の ADR ホームを含む ADR ベース・ディレクトリを示していることを確認します。DIAGNOSTIC_DEST が欠落している場合

や NULL である場合は、ORACLE_HOME/log で有効な ADR ベース・ディレクトリ階層を確認します。

DIA-48431: ADR ホーム・パスを 1 つ以上指定する必要があります

原因: そのコマンドで、1 つ以上の ADR ホームをカレントにする必要があります。

処理: SET HOMEPATH コマンドを使用して、1 つ以上の ADR ホームをカレントにします。

DIA-48432: ADR ホーム・パス[string]が無効です

原因: 指定された ADR ホームが有効ではありません。パスが存在していない可能性があります。

処理: 指定された ADR ホームパスが存在していることを確認します。

DIA-48447: 入力パス[path]に ADR ホームが含まれていません

原因: SET HOMEPATH を使用して ADR ホームを設定する場合、カレント ADR ベースに対して相対的なパスを指定する必要があります。

処理: 目的とする新しい ADR ホームがカレント ADR ベースに存在しない場合は、まず SET BASE を使用して ADR ベースを設定し、その後で SHOW HOMES を使用して新しい ADR ベースの下で ADR ホームを確認します。次に、必要に応じて、SET HOMEPATH を使用して新しい ADR ホームを設定します。

DIA-48448: このコマンドは、複数の ADR ホームをサポートしていません

原因: カレント ADRCI セッションに複数のカレント ADR ホームが存在します。

処理: SET HOMEPATH コマンドを使用して、1 つの ADR ホームをカレントにします。

親トピック: ADRCI: ADRコマンド・インタプリタ

22 DBVERIFY: オフライン・データベース検査ユーティリティ

DBVERIFYは外部コマンドライン・ユーティリティで、物理データ構造に対する整合性チェックを実行します。

DBVERIFYは、オフラインまたはオンライン・データベースおよびバックアップ・ファイル上で使用できます。DBVERIFYを使用するのは、バックアップ・データベース(またはデータ・ファイル)をリストアする前にそれが有効であることを確認する場合です。また、データ破損の問題が発生した場合に診断機能としても使用します。DBVERIFYはオフライン・データベースに対して実行できるため、整合性チェックが非常に高速に行えます。

DBVERIFYによるチェックは、キャッシュ管理(データ・ブロック)ブロックのみに制限されています。DBVERIFYは、データ・ファイルでのみ使用します。制御ファイルまたはREDOログに対しては機能しません。

DBVERIFYには、2つのコマンドライン・インタフェースがあります。最初のインタフェースでは、確認の対象として単一データ・ファイルのディスク・ブロックを指定します。次のインタフェースでは、確認の対象としてセグメントを指定します。いずれのインタフェースでも、最初にdbvコマンドを記述します。次の項では、これらのインタフェースの記述について説明します。

- DBVERIFYを使用した単一データ・ファイルのディスク・ブロックの検査
  このモードでは、DBVERIFYによって、単一データ・ファイルの1つ以上のディスク・ブロックがスキャンされ、ページのチェックが実行されます。
- DBVERIFYを使用したセグメントの検査
  このモードでは、表セグメントまたは索引セグメントをDBVERIFYの検査対象に指定できます。

親トピック: その他のユーティリティ

22.1 DBVERIFYを使用した単一データ・ファイルのディスク・ブロックの検査

このモードでは、DBVERIFYによって、単一データ・ファイルの1つ以上のディスク・ブロックがスキャンされ、ページのチェックが実行されます。

検査するファイルが、Oracle Automatic Storage Management (Oracle ASM)ファイルである場合、USERIDを指定する必要があります。DBVERIFYでOracle ASMファイルにアクセスするには、Oracleインスタンスに接続する必要があるためです。

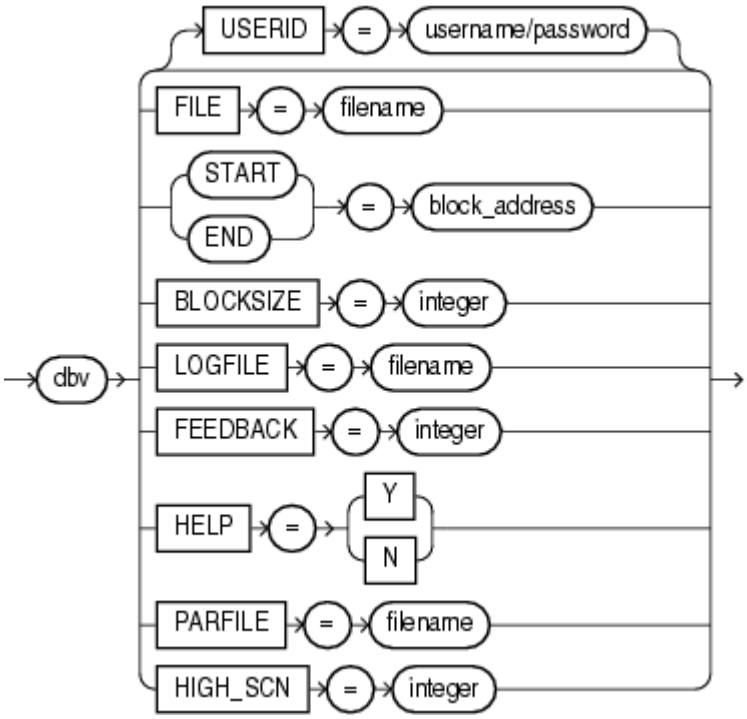
- 単一ファイルのブロックを検査する際のDBVERIFY構文
  DBVERIFYを使用して単一ファイルのブロックを検証するための構文を参照してください。
- 単一ファイルのブロックを検査する際のDBVERIFYパラメータ
  単一ファイルのブロックを検査するために使用できるDBVERIFYのパラメータを参照してください。
- 単一データ・ファイルのDBVERIFY出力例
  単一データ・ファイルの検証例およびその説明を参照してください。

親トピック: DBVERIFY: オフライン・データベース検査ユーティリティ

22.1.1 単一ファイルのブロックを検査する際のDBVERIFY構文

DBVERIFYを使用して単一ファイルのブロックを検証するための構文を参照してください。

単一データ・ファイルのディスク・ブロックを検査するときに使用するDBVERIFYの構文は次のとおりです。



親トピック: [DBVERIFYを使用した単一データ・ファイルのディスク・ブロックの検査]

22.1.2 単一ファイルのブロックを検査する際のDBVERIFYパラメータ

単一ファイルのブロックの検証に使用できるDBVERIFYパラメータを参照してください。

パラメータ	説明
USERID	ユーザー名およびパスワードを指定します。 このパラメータは、検査するファイルが Oracle ASM ファイルである場合にのみ必要です。 このパラメータを指定する場合は、ユーザー名とパスワードの両方を入力する必要があります。入力しない場合、DBV-00112: USERID incorrectly specified エラーが返されます。
FILE	検証するデータベース・ファイルの名前。
START	検証する開始ブロック・アドレス。ブロック・アドレスは、Oracle ブロックで指定します(オペレーティング・システム・ブロックではありません)。START を指定しないと、DBVERIFY はファイル内の最初のブロックにデフォルト設定されます。
END	検証する終了ブロック・アドレス。END を指定しないと、DBVERIFY はファイル内の最後のブロックにデフォルト設定されます。
BLOCKSIZE	BLOCKSIZE は、検査するファイルのブロック・サイズが 2KB でない場合にのみ指定します。ファイルのブロック・サイズが 2KB でない場合、

パラメータ	説明
	BLOCKSIZE を指定しないとエラーDBV-00103 が表示されます。
HIGH_SCN	HIGH_SCN の値が指定されている場合、DBVERIFY は、ブロック・レベルのシステム変更番号(SCN)が指定された値を超えているブロックごとに診断メッセージを書き込みます。 このパラメータは省略可能です。デフォルトはありません。
LOGFILE	ログ情報を書き込むログ・ファイルの名前とパスを指定します。デフォルトでは、端末画面への出力となります。
FEEDBACK	DBVERIFY により、DBVERIFY の実行中に検証されたページ数 n につき 1 つのピリオド(.)の形式で、進捗表示が端末に表示されます。n = 0 と設定すると、進捗画面は表示されません。
HELP	オンライン・ヘルプを表示します。特定のバージョンの DBVERIFY のコマンドライン・パラメータのヘルプを表示するには、コマンドラインで dbv help=y と入力します。
PARFILE	使用するパラメータ・ファイルの名前を指定します。DBVERIFY パラメータの様々な値をフラット・ファイルに保存できます。これにより、パラメータ・ファイルをカスタマイズして様々なタイプのデータ・ファイルを処理したり、特定のタイプの整合性チェックをデータ・ファイルに対して実行できます。

関連トピック

- DBVERIFY - Database file Verification Utility (ドキュメントID 35512.1)

親トピック: DBVERIFYを使用した単一データ・ファイルのディスク・ブロックの検査

22.1.3 単一データ・ファイルに対するDBVERIFYの出力例

単一のデータファイルの検証例と、その説明を参照してください。

次に、ファイルt_db1.dbfの検証例を示します。フィードバック・パラメータには値100が指定され、処理された100ページごとに1つのピリオド(.)が表示されます。

```
% dbv FILE=t_db1.dbf FEEDBACK=100
```

出力例

このコマンドの出力は次のようになります。

```
.
.
.
DBVERIFY - Verification starting : FILE = t_db1.f
```

```
DBVERIFY - Verification complete
Total Pages Examined : 120424
Total Pages Processed (Data) : 79507
Total Pages Failing (Data) : 0
Total Pages Processed (Index): 15236
Total Pages Failing (Index): 0
Total Pages Processed (Other): 5626
Total Pages Processed (Seg) : 1
Total Pages Failing (Seg) : 0
Total Pages Empty : 20055
Total Pages Marked Corrupt : 0
Total Pages Influx : 0
Total Pages Encrypted : 0
Highest block SCN : 25565681 (0.25565681)
```

ノート

- Pages = ブロック
- Total Pages Examined =ファイルのブロック数。
- Total Pages Processed (Data) = 検査されたブロック数(フォーマットされたブロック)。
- Total Pages Processed (Other) = メタデータ・ブロック。これらのブロックは検証されていないため、Total Pages Failing (Other)の出力はありません。
- Total Pages Processed (Seg) = セグメント・ヘッダーのブロック数。
- Total Pages Failing (Data) = データ・ブロック・チェック・ルーチンで違反したブロック数。
- Total Pages Failing (Index) = 索引ブロック・チェック・ルーチンで違反したブロック数。
- Total Pages Marked Corrupt = キャッシュ・ヘッダーが無効で、DBVERIFYがブロック・タイプを識別できないブロック数。
- Total Pages Influx = 読取りおよび書込みが同時に行われるブロック数。DBVERIFYの実行時にデータベースがオープンされている場合は、一貫性を維持したイメージの取得のためにDBVERIFYによってブロックが複数回読み込まれます。ただし、データベースがオープンされているため、読取りおよび書込みが同時に行われるブロックがある可能性があります(INFLUX)。その場合、DBVERIFYでは、絶え間なく変化するページの一貫性のあるイメージは取得できません。
- Total Pages Encrypted = データまたは索引のみでなく、すべてのブロック(データ、索引、その他、セグメント)。Total Pages Encryptedがゼロ以外の場合は、DBVERIFYが「DBVerifyは暗号化ブロックに対して論理チェックを実行できません。RMANを使用する必要があります。」というメッセージを出力します。

親トピック: DBVERIFYを使用した単一データ・ファイルのディスク・ブロックの検査

22.2 DBVERIFYを使用したセグメントの検査

このモードでは、DBVERIFYを使用して表セグメントまたは索引セグメントの検査を指定できます。

確認中のセグメントに行連鎖ポインタがあることを確認します。

このモードでは、検査するセグメント(データ・セグメントまたは索引セグメント)を指定する必要があります。また、セグメントについての情報をデータベースから取得するため、SYSDBA権限でデータベースにログインする必要があります。

このモードでは、セグメントはロックされます。1つの索引セグメントを指定する場合、親表はロックされます。IOTなど、一部の索引には親表はありません。

- セグメント検証時のDBVERIFY構文
  DBVERIFYを使用してセグメントを検証するための構文を参照してください。
- 単一のセグメントを検査する際のDBVERIFYパラメータ

単一のセグメントを検査するために使用できるDBVERIFYパラメータを参照してください。

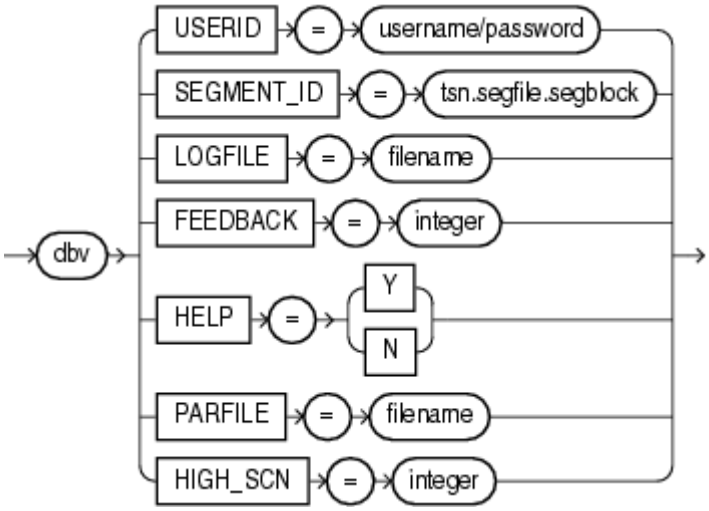
- 検査対象のセグメントに対するDBVERIFYの出力例
  検査対象のセグメントの検証の例を参照してください。

親トピック: DBVERIFY: オフライン・データベース検査ユーティリティ

22.2.1 セグメントを検査する際のDBVERIFY構文

DBVERIFYを使用してセグメントを検証するための構文を参照してください。

セグメントを検査するときのDBVERIFYの構文は次のとおりです。



親トピック: DBVERIFYを使用したセグメントの検査

22.2.2 単一のセグメントを検査する際のDBVERIFYパラメータ

単一セグメントの検証に使用できるDBVERIFYパラメータを参照してください。

パラメータ	説明
USERID	ユーザー名およびパスワードを指定します。ユーザー名とパスワードのどちらも入力しない場合、エラーDBV-00112: USERID incorrectly specified が返されます。 コンテナ・データベース(CDB)に接続する場合は、username@cdbname/password と入力します。
SEGMENT_ID	検証するセグメントを指定します。セグメント識別子は、表領域 ID 番号(tsn)、セグメント・ヘッダー・ファイル番号(segfile)およびセグメント・ヘッダー・ブロック番号(segblock)で構成されます。この情報は SYS_USER_SEGS から取得できます。関連する列は、TABLESPACE_ID、HEADER_FILE および HEADER_BLOCK です。SYS_USER_SEGS を問い合せるには、SYSDBA 権限が必要です。 たとえば、表領域番号(TS#)が 2、セグメント・ヘッダー・ファイル番号(HEADER_FILE)が 5、セグメント・ヘッダー・ブロック番号(HEADER_BLOCK)が 37767 の場合、SEGMENT_ID=2.5.37767 を使用してそのセグメントを確認してください。

パラメータ	説明
HIGH_SCN	HIGH_SCN の値が指定されている場合、DBVERIFY は、ブロック・レベルの SCN が指定された値を超えているブロックごとに診断メッセージを書き込みます。 このパラメータは省略可能です。デフォルトはありません。
LOGFILE	ログ情報を書き込むファイルを指定します。デフォルトでは、端末画面への出力となります。
FEEDBACK	DBVERIFY により、DBVERIFY の実行中に検証されたページ数 n につき 1 つのピリオド(.)の形式で、進捗表示が端末に表示されます。n = 0 と設定すると、進捗画面は表示されません。
HELP	オンライン・ヘルプを表示します。
PARFILE	使用するパラメータ・ファイルの名前を指定します。DBVERIFY パラメータの様々な値をフラット・ファイルに保存できます。これにより、パラメータ・ファイルをカスタマイズして様々なタイプのデータ・ファイルを処理したり、特定のタイプの整合性チェックをデータ・ファイルに対して実行できます。

親トピック: DBVERIFYを使用したセグメントの検査

22.2.3 検査対象のセグメントに対するDBVERIFYの出力例

検査対象のセグメントに対する検証の例を参照してください。

次に、表領域セグメントを指定してDBVERIFYコマンドを使用する例、およびDBVERIFY操作によって生成される出力を示します。

```
% dbv userid=system/ SEGMENT_ID=2.5.37767
```

このコマンドの出力は次のようになります。

```
DBVERIFY - Verification starting : SEGMENT_ID = 2.5.37767
DBVERIFY - Verification complete
Total Pages Examined : 640
Total Pages Processed (Data) : 0
Total Pages Failing (Data) : 0
Total Pages Processed (Index): 0
Total Pages Failing (Index): 0
Total Pages Processed (Other): 591
Total Pages Processed (Seg) : 8
Total Pages Failing (Seg) : 0
Total Pages Empty : 13
Total Pages Marked Corrupt : 0
Total Pages Influx : 0
Total Pages Encrypted : 28
Highest block SCN : 7877587 (0.7877587)
```

関連トピック

- DBVERIFY enhancement - How to scan an object/segment (ドキュメントID 139962.1)

親トピック: DBVERIFYを使用したセグメントの検査

23 DBNEWIDユーティリティ

DBNEWIDは、オペレーショナル・データベースのDBIDおよびDBNAMEを変更できるデータベース・ユーティリティです。

次のトピックを参照してください。

- DBNEWIDユーティリティとは
  DBNEWIDユーティリティを使用すると、DBID、DBNAME、またはデータベースのDBIDとDBNAMEの両方のみを変更できます。
- DBIDおよびDBNAMEの変更による影響
  DBNEWIDユーティリティを使用してデータベースのDBIDおよびDBNAMEを変更する前に、次のガイドラインを確認してください。
- グローバル・データベース名に関する考慮点
  分散データベース・システム内のデータベースを使用している場合、各データベースには一意のグローバル・データベース名が必要です。
- DBNEWIDを使用したCDB DBIDとPDB DBIDの両方の変更
  DBNEWIDパラメータPDBを使用すると、プラガブル・データベース(PDB)のDBIDを変更できます。
- データベースのDBIDおよびDBNAMEの変更
  DBIDまたはDBNAME、あるいはデータベースのDBIDとDBNAMEの両方を変更するには、必要なDBNEWIDプロシージャを選択します。
- DBNEWIDの構文
  DBNEWIDの構文について説明します。

親トピック: その他のユーティリティ

23.1 DBNEWIDユーティリティとは

DBNEWIDユーティリティを使用すると、DBID、DBNAMEのみ、またはデータベースのDBIDとDBNAMEの両方を変更できます。

DBNEWIDユーティリティの導入以前は、データベースのコピーを手動で作成し、制御ファイルを再作成してデータベースに新しいデータベース名(DBNAME)を指定できました。ただし、データベースに新しい識別子(DBID)は指定できませんでした。DBIDは、データベース用の内部的な一意の識別子です。Recovery Manager(RMAN)ではDBIDによってデータベースが区別されるため、シード・データベースおよび手動でコピーされたデータベースは、同じRMANリポジトリに登録できません。DBNEWIDユーティリティを使用すると、次のいずれかを変更することによって、この問題を解決できます。

- データベースのDBIDのみ
- データベースのDBNAMEのみ
- データベースのDBNAMEおよびDBIDの両方

親トピック: DBNEWIDユーティリティ

23.2 DBIDおよびDBNAMEの変更による影響

DBNEWIDユーティリティを使用してデータベースのDBIDおよびDBNAMEを変更する前に、次のガイドラインを確認してください。

DBIDを変更する場合は、データベース全体のバックアップをすぐに作成する必要があります。

データベースのDBIDの変更は、重要な処理です。データベースのDBIDを変更すると、そのデータベースのすべての古いバックアップおよびアーカイブ・ログが使用できなくなります。DBIDの変更は、データがすでにデータファイルに存在する点を除いて、デー

タベースの作成と同様です。DBIDを変更すると、DBIDの変更前に作成されたバックアップおよびアーカイブ・ログは、現行のDBIDに一致しない元のDBIDを保持しているため、使用できなくなります。オンラインREDOログを再作成し、REDOログ順序を1にリセットするRESETLOGSオプションを使用して、データベースをオープンする必要があります。したがって、次のことが該当します。

DBNAMEを変更し、DBIDを変更しない場合は、DBNAME初期化パラメータを変更し、追加のガイドラインに従う必要があります。

DBIDを変更せずにDBNAMEを変更する場合は、RESETLOGSオプションを指定してデータベースをオープンする必要がないため、データベースのバックアップおよびアーカイブ・ログは無効になりません。ただし、DBNAMEを変更すると、これらが無効になります。データベース名の変更後、新しい名前を反映させるにはDB_NAME初期化パラメータを変更する必要があります。また、Oracleパスワード・ファイルの再作成が必要な場合もあります。制御ファイルの古い(データベース名を変更する前の)バックアップをリストアする場合、データベース名を変更する前の初期化パラメータ・ファイルおよびパスワード・ファイルを使用する必要があります。

> 注意:
>
> 取得プロセスを使用してデータベースに対する変更を取得する場合は、データベースの DBID または DBNAME を変更しないでください。

Oracle RAC環境の場合のみ、DBNEWIDユーティリティを実行する前に、まずクラスタからデータベースをデタッチする必要があります。SQL*Plusを使用して次のコマンドを入力し、CLUSTER_DATABASEの初期化パラメータ値をFALSEに設定します

1. ALTER SYSTEM SET CLUSTER_DATABASE=FALSE SCOPE=SPFILE;

   CLUSTER_DATABASEパラメータを変更した後にデータベースを再起動します。

2. データベースを停止します。

   SHUTDOWN IMMEDIATE

その後、STARTUP MOUNT EXCLUSIVEを実行し、グローバル・データベース名を変更できます。CLUSTER_DATABASE=TRUE中にDBNEWIDユーティリティを使用しようとすると、NID-00120: Database should be mounted exclusivelyでコマンドが失敗します。

関連トピック

- NIDユーティリティを使用してDBID、DBNAMEを変更する方法(ドキュメントID 863800.1)

親トピック: DBNEWIDユーティリティ

23.3 グローバル・データベース名に関する考慮点

分散データベース・システム内のデータベースを使用している場合、各データベースには一意のグローバル・データベース名が必要です。

DBNEWIDユーティリティは、グローバル・データベース名を変更しません。

グローバル・データベース名を変更するには、SQLのALTER DATABASE文を使用する必要があります。構文は次のとおりです。

```
ALTER DATABASE RENAME GLOBAL_NAME TO newname.domain;
```

グローバル・データベース名は、データベース名およびドメインで構成されます。これらはデータベースが最初に作成されたときにDB_NAMEおよびDB_DOMAIN初期化パラメータによって指定されます。

たとえば、DBNEWIDを使用してデータベース名をsalesに変更するとします。ドメインexample.comでグローバル・データベース名をsalesに変更するには、次のようにALTER DATABASE RENAMEを使用する必要があります。

```
ALTER DATABASE RENAME GLOBAL_NAME TO sales.example.com
```

関連トピック

- グローバル・データベース名の変更: Oracle Database管理者ガイドのシナリオ

関連項目:

グローバル・データベース名の詳細は*Oracle Database*管理者ガイド、およびMy Oracle SupportのNIDユーティリティを使用したDBID、DBNAMEの変更方法(ドキュメントID 863800.1)を参照

親トピック: DBNEWIDユーティリティ

23.4 DBNEWIDを使用したCDBおよびPDB DBIDの両方の変更

DBNEWIDパラメータであるPDBを使用すると、プラガブル・データベース(PDB)のDBIDを変更できます。

デフォルトでは、DBNEWIDユーティリティをコンテナ・データベース(CDB)で実行すると、このユーティリティはCDBのDBIDのみを変更します。CDBに接続されている各プラガブル・データベース(PDB)のDBID値は同じままです。場合によっては、このデフォルトの動作によってPDBのDBID値が重複して問題が発生することがあります。たとえば、CDBのクローニング時にこの問題が発生する可能性があります。

Oracle Database 12cリリース2 (12.2)以降では、マルチテナント・データベースでDBNEWIDユーティリティPDBパラメータを使用して、PDBのDBID値を変更できます。特定のPDBを指定することはできません。すべてのPDBに新しいDBID値が割り当てられているか、どのPDBにも割り当てられていません。PDBパラメータの形式は次のとおりです。

```
PDB=[ALL | NONE]
```

- ALLを指定すると、CDB変更のDBIDに加えて、CDBに接続されているすべてのPDBのDBID値も変更されます。
- NONE(デフォルト)を指定すると、CDBのDBIDが変更されてもPDBのDBIDは変更されません。

PDB=ALLを使用することをお薦めします。下位互換性を保つために、デフォルトはPDB=NONEです。

親トピック: DBNEWIDユーティリティ

23.5 データベースのDBIDおよびDBNAMEの変更

DBIDまたはDBNAME、あるいはデータベースのDBIDとDBNAMEの両方を変更するには、必要なDBNEWIDプロシージャを選択します。

- DBIDおよびデータベース名の変更
  DBNEWIDを持つデータベースのDBIDまたはDBIDとDBNAMEの両方を変更するには、このプロシージャを使用します。
- データベースIDのみの変更
  データベース名を変更せずにデータベースID (DBID)を変更するには、このDBNEWIDプロシージャを使用します。
- データベース名のみの変更

DBIDを変更せずにデータベース名(DBNAME)を変更するには、このDBNEWIDプロシージャを使用します。

- DBNEWIDのトラブルシューティング
  DBNEWIDを使用してデータベースIDを変更する際にエラーが発生する場合は、次のトラブルシューティング・ヒントを参照してください。

親トピック: DBNEWIDユーティリティ

23.5.1 DBIDおよびデータベース名の変更

DBNEWIDを持つデータベースのDBIDまたはDBIDとDBNAMEの両方を変更するには、このプロシージャを使用します。

次のステップでは、データベースのDBIDを変更する方法について説明します。データベース名も変更できます。

1. データベース全体のリカバリ可能なバックアップがあることを確認します。
2. ターゲット・データベースがマウント状態であること、またそのデータベースがマウント前に一貫性を維持した状態で停止されていることを確認します。たとえば:

```
SHUTDOWN IMMEDIATE
STARTUP MOUNT
```

3. コマンドラインで、SYSDBA権限を持つ有効なユーザー(TARGET)を指定してDBNEWIDユーティリティを起動します(パスワードの入力を求められます)。

```
% nid TARGET=SYS
```

DBIDの他にデータベース名を変更するには、コマンドラインでDBNAMEパラメータも指定します(パスワードの入力を求められます)。次の例は、データベース名をtest_dbに変更します。

```
% nid TARGET=SYS DBNAME=test_db
```

DBNEWIDユーティリティは、ファイルを変更する前に、データファイルおよび制御ファイルのヘッダーで検証を実行します。妥当性チェックが正常に行われると、DBNEWIDは、操作の確認を求め(ログ・ファイルを指定した場合は確認を求めません)、オフラインの通常のデータ・ファイルおよび読取り専用データ・ファイルを含む各データ・ファイルのDBIDを変更し(この例のように、DBNAMEが指定された場合は、この例のように変更します)、データベースを停止し、終了します。次に、この出力例を示します。

```
.
.
.
Connected to database PROD (DBID=86997811)
.
.
.
Control Files in database:
    /oracle/TEST_DB/data/cf1.dbf
    /oracle/TEST_DB/data/cf2.dbf
The following datafiles are offline clean:
    /oracle/TEST_DB/data/tbs_61.dbf (23)
    /oracle/TEST_DB/data/tbs_62.dbf (24)
    /oracle/TEST_DB/data/temp3.dbf (3)
These files must be writable by this utility.
The following datafiles are read-only:
    /oracle/TEST_DB/data/tbs_51.dbf (15)
    /oracle/TEST_DB/data/tbs_52.dbf (16)
    /oracle/TEST_DB/data/tbs_53.dbf (22)
These files must be writable by this utility.
Changing database ID from 86997811 to 1250654267
Changing database name from PROD to TEST_DB
    Control File /oracle/TEST_DB/data/cf1.dbf - modified
```

```
    Control File /oracle/TEST_DB/data/cf2.dbf - modified
    Datafile /oracle/TEST_DB/data/tbs_01.dbf - dbid changed, wrote new name
    Datafile /oracle/TEST_DB/data/tbs_ax1.dbf - dbid changed, wrote new name
    Datafile /oracle/TEST_DB/data/tbs_02.dbf - dbid changed, wrote new name
    Datafile /oracle/TEST_DB/data/tbs_11.dbf - dbid changed, wrote new name
    Datafile /oracle/TEST_DB/data/tbs_12.dbf - dbid changed, wrote new name
    Datafile /oracle/TEST_DB/data/temp1.dbf - dbid changed, wrote new name
    Control File /oracle/TEST_DB/data/cf1.dbf - dbid changed, wrote new name
    Control File /oracle/TEST_DB/data/cf2.dbf - dbid changed, wrote new name
    Instance shut down
Database name changed to TEST_DB.
Modify parameter file and generate a new password file before restarting.
Database ID for database TEST_DB changed to 1250654267.
All previous backups and archived redo logs for this database are unusable.
Database has been shutdown, open database with RESETLOGS option.
Successfully changed database name and ID.
DBNEWID - Completed successfully.
```

妥当性チェックが正常に行われなかった場合、次に示す出力例のようにDBNEWIDユーティリティが終了し、ターゲット・データベースは変更されません。データベースをオープンし、エラーを修正した後、DBNEWIDの操作を再開するか、またはDBIDを変更せずに継続してデータベースを使用できます。

```
.
.
.
Connected to database PROD (DBID=86997811)
.
.
.
Control Files in database:
    /oracle/TEST_DB/data/cf1.dbf
    /oracle/TEST_DB/data/cf2.dbf

The following datafiles are offline clean:
    /oracle/TEST_DB/data/tbs_61.dbf (23)
    /oracle/TEST_DB/data/tbs_62.dbf (24)
    /oracle/TEST_DB/data/temp3.dbf (3)
These files must be writable by this utility.

The following datafiles are read-only:
    /oracle/TEST_DB/data/tbs_51.dbf (15)
    /oracle/TEST_DB/data/tbs_52.dbf (16)
    /oracle/TEST_DB/data/tbs_53.dbf (22)
These files must be writable by this utility.

The following datafiles are offline immediate:
    /oracle/TEST_DB/data/tbs_71.dbf (25)
    /oracle/TEST_DB/data/tbs_72.dbf (26)

NID-00122: Database should have no offline immediate datafiles

Change of database name failed during validation - database is intact.
DBNEWID - Completed with validation errors.
```

4. データベースをマウントします。たとえば:

```
STARTUP MOUNT
```

5. RESETLOGSモードでデータベースをオープンし、通常の使用を再開します。たとえば:

```
ALTER DATABASE OPEN RESETLOGS;
```

ログをリセットした後、新しいデータベース・バックアップを作成します。オンラインREDOログをリセットしたため、現行のデータベース

では古いバックアップとアーカイブ・ログは使用できません。

親トピック: データベースのDBIDおよびDBNAMEの変更

23.5.2 データベースIDのみの変更

データベース名を変更せずにデータベースID (DBID)を変更するには、このDBNEWIDプロシージャを使用します。

DBIDおよびデータベース名の変更のステップに従ってください。ただしステップ3では、オプションのデータベース名(DBNAME)を指定しないでください。次に、データベースIDのみを変更した場合に生成される出力例を示します。

```
.
.
.
Connected to database PROD (DBID=86997811)
.
.
.
Control Files in database:
    /oracle/TEST_DB/data/cf1.dbf
    /oracle/TEST_DB/data/cf2.dbf

The following datafiles are offline clean:
    /oracle/TEST_DB/data/tbs_61.dbf (23)
    /oracle/TEST_DB/data/tbs_62.dbf (24)
    /oracle/TEST_DB/data/temp3.dbf (3)
These files must be writable by this utility.

The following datafiles are read-only:
    /oracle/TEST_DB/data/tbs_51.dbf (15)
    /oracle/TEST_DB/data/tbs_52.dbf (16)
    /oracle/TEST_DB/data/tbs_53.dbf (22)
These files must be writable by this utility.

Changing database ID from 86997811 to 4004383693
    Control File /oracle/TEST_DB/data/cf1.dbf - modified
    Control File /oracle/TEST_DB/data/cf2.dbf - modified
    Datafile /oracle/TEST_DB/data/tbs_01.dbf - dbid changed
    Datafile /oracle/TEST_DB/data/tbs_ax1.dbf - dbid changed
    Datafile /oracle/TEST_DB/data/tbs_02.dbf - dbid changed
    Datafile /oracle/TEST_DB/data/tbs_11.dbf - dbid changed
    Datafile /oracle/TEST_DB/data/tbs_12.dbf - dbid changed
    Datafile /oracle/TEST_DB/data/temp1.dbf - dbid changed
    Control File /oracle/TEST_DB/data/cf1.dbf - dbid changed
    Control File /oracle/TEST_DB/data/cf2.dbf - dbid changed
    Instance shut down

Database ID for database TEST_DB changed to 4004383693.
All previous backups and archived redo logs for this database are unusable.
Database has been shutdown, open database with RESETLOGS option.
Succesfully changed database ID.
DBNEWID - Completed succesfully.
```

親トピック: データベースのDBIDおよびDBNAMEの変更

23.5.3 データベース名のみの変更

DBIDを変更せずにデータベース名(DBNAME)を変更するには、このDBNEWIDプロシージャを使用します。

次のステップを実行します。

1. データベース全体のリカバリ可能なバックアップがあることを確認します。

2. ターゲット・データベースがマウント状態であること、またそのデータベースがマウント前に一貫性を維持した状態で停止されていることを確認します。たとえば:

```
SHUTDOWN IMMEDIATE
STARTUP MOUNT
```

3. コマンドラインでユーティリティを起動し、SYSDBA権限を持つ有効なユーザーを指定します(パスワードを入力するように要求されます)。DBNAMEパラメータとSETNAMEパラメータの両方を指定する必要があります。この例では、データベース名をtest_dbに変更します。

```
% nid TARGET=SYS DBNAME=test_db SETNAME=YES
```

DBNEWIDユーティリティを使用して、(データ・ファイルではなく)制御ファイルへの入出力を試行する前に、制御ファイルのヘッダーで妥当性チェックを実行します。妥当性チェックが正常に行われると、操作の確認が求められ、制御ファイル内のデータベース名が変更されます。これらの処理の完了後、データベースが停止され、DBNEWIDユーティリティが終了します。次に、この出力例を示します。

```
.
.
.
Control Files in database:
    /oracle/TEST_DB/data/cf1.dbf
    /oracle/TEST_DB/data/cf2.dbf
The following datafiles are offline clean:
    /oracle/TEST_DB/data/tbs_61.dbf (23)
    /oracle/TEST_DB/data/tbs_62.dbf (24)
    /oracle/TEST_DB/data/temp3.dbf (3)
These files must be writable by this utility.
The following datafiles are read-only:
    /oracle/TEST_DB/data/tbs_51.dbf (15)
    /oracle/TEST_DB/data/tbs_52.dbf (16)
    /oracle/TEST_DB/data/tbs_53.dbf (22)
These files must be writable by this utility.
Changing database name from PROD to TEST_DB
    Control File /oracle/TEST_DB/data/cf1.dbf - modified
    Control File /oracle/TEST_DB/data/cf2.dbf - modified
    Datafile /oracle/TEST_DB/data/tbs_01.dbf - wrote new name
    Datafile /oracle/TEST_DB/data/tbs_ax1.dbf - wrote new name
    Datafile /oracle/TEST_DB/data/tbs_02.dbf - wrote new name
    Datafile /oracle/TEST_DB/data/tbs_11.dbf - wrote new name
    Datafile /oracle/TEST_DB/data/tbs_12.dbf - wrote new name
    Datafile /oracle/TEST_DB/data/temp1.dbf - wrote new name
    Control File /oracle/TEST_DB/data/cf1.dbf - wrote new name
    Control File /oracle/TEST_DB/data/cf2.dbf - wrote new name
    Instance shut down
Database name changed to TEST_DB.
Modify parameter file and generate a new password file before restarting.
Successfully changed database name.
DBNEWID - Completed successfully.
```

妥当性チェックが正常に行われなかった場合、DBNEWIDユーティリティは終了し、ターゲット・データベースは変更されません。データベースをオープンし、エラーを修正した後で、DBNEWIDユーティリティの操作を再開するか、またはデータベース名を変更せずに継続してデータベースを使用することができます。妥当性チェックが正常に行われなかった場合の出力例は、「DBIDおよびデータベース名の変更」のステップ3を参照してください。

4. 初期化パラメータ・ファイル(PFILE)のDB_NAME初期化パラメータを新しいデータベース名に設定します。

ノート:

DBNEWID ユーティリティは、サーバー・パラメータ・ファイル(SPFILE)を変更しません。したがって、SPFILE を使用して、Oracle Database を起動した場合は、サーバー・パラメータ・ファイルから初期化パラメータ・ファイルを再作成し、サーバー・パラメータ・ファイルを削除して、初期化パラメータ・ファイルの DB_NAME を変更した後で、サーバー・パラメータ・ファイルを再作成する必要があります。

5. 新しいパスワード・ファイルを作成します。
6. データベースを起動し、通常の使用を再開します。たとえば:

```
STARTUP
```

データベースIDは変更せずに、データベース名のみを変更したため、データベースのオープン時にRESETLOGSオプションを使用する必要はありません。以前のすべてのバックアップは引き続き使用可能です。

親トピック: データベースのDBIDおよびDBNAMEの変更

23.5.4 DBNEWIDのトラブルシューティング

DBNEWIDを使用してデータベースIDを変更するときにエラーが発生した場合は、次のトラブルシューティング・ヒントを参照してください。

DBNEWIDユーティリティによって妥当性チェックが正常に行われたが、要求された変更の実行中にエラーが検出された場合、DBNEWIDユーティリティは停止し、データベースは変更中のままになります。この場合、DBNEWIDユーティリティの操作を完了するか、または元に戻さないかぎり、データベースをオープンできません。DBNEWIDユーティリティによって操作の状態を示すメッセージが表示されます。

操作を続行するか元に戻す前に、エラーの原因を解決します。データベース全体を最新のバックアップからリストアして、DBNEWIDユーティリティの起動前の時点までリカバリを実行する以外に解決策がない場合もあります。このシナリオでは、DBNEWIDの前に最近のバックアップを使用可能にすることの重要性を理解しています。

変更を続行する場合は、元のコマンドを再実行します。DBNEWIDユーティリティが再開され、すべてのデータ・ファイルおよび制御ファイルに新しい値が指定されるまで変更操作が続行されます。この時点では、データベースは停止されます。RESETLOGSオプションを使用してデータベースをオープンする前に、マウントする必要があります。

DBNEWIDユーティリティの操作を元に戻した場合、DBNEWIDユーティリティによって実行されたすべての変更が元に戻され、データベースはマウントされた状態のままになります。

DBNEWIDがOracle Database 10gリリース1 (10.1)以降のリリースのOracle Databaseに対して実行されている場合、操作のサマリーがアラート・ファイルに書き込まれます。

例23-1 データベース名とデータベースIDの変更に関するアラート・ファイル

データベース名とデータベースIDを変更したとします。アラート・ファイルには、次のような結果が表示されます。

```
*** DBNEWID utility started ***
DBID will be changed from 86997811 to new DBID of 1250452230 for
database PROD
DBNAME will be changed from PROD to new DBNAME of TEST_DB
Starting datafile conversion
Setting recovery target incarnation to 1
Datafile conversion complete
Database name changed to TEST_DB.
Modify parameter file and generate a new password file before restarting.
Database ID for database TEST_DB changed to 1250452230.
All previous backups and archived redo logs for this database are unusable.
Database has been shutdown, open with RESETLOGS option.
Successfully changed database name and ID.
```

```
*** DBNEWID utility finished successfully ***
```

データベース名だけの変更では、次のような内容がアラート・ファイルに記録されます。

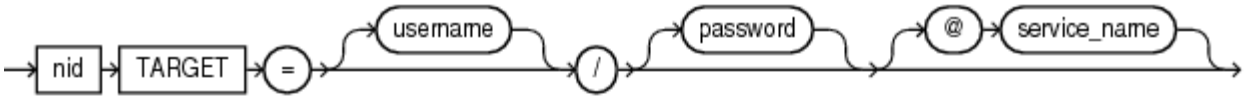
```
*** DBNEWID utility started ***
DBNAME will be changed from PROD to new DBNAME of TEST_DB
Starting datafile conversion
Datafile conversion complete
Database name changed to TEST_DB.
Modify parameter file and generate a new password file before restarting.
Successfully changed database name.
*** DBNEWID utility finished successfully ***

In case of failure during DBNEWID the alert will also log the failure:
*** DBNEWID utility started ***
DBID will be changed from 86997811 to new DBID of 86966847 for database
AV3
Change of database ID failed.
Must finish change or REVERT changes before attempting any database
operation.
*** DBNEWID utility finished with errors ***
```

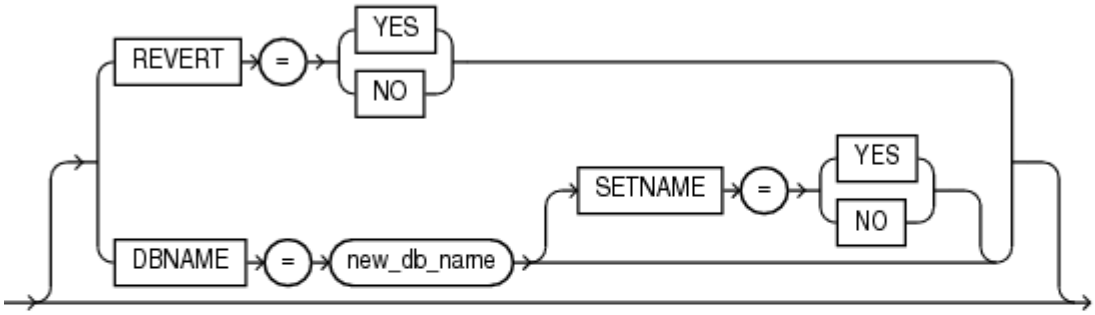
親トピック: データベースのDBIDおよびDBNAMEの変更

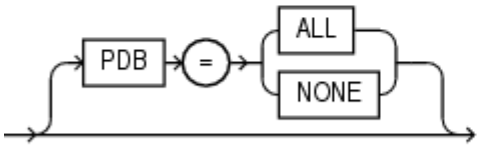
23.6 DBNEWIDの構文

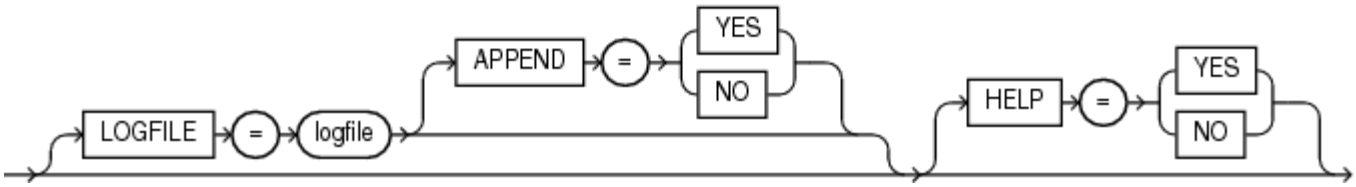
DBNEWIDの構文について説明します。

次の図に、DBNEWIDユーティリティの構文を示します。









- DBNEWIDパラメータ
  DBNEWIDのパラメータについて説明します。
- 制限と使用方法

DBNEWIDユーティリティの制限について説明します。

- Oracle Database 10gより前のリリースに対する追加の制限事項
  リリース10.1より前のOracle Databaseに対してDBNEWIDユーティリティを実行する場合の追加の制限事項について説明します。

親トピック: DBNEWIDユーティリティ

23.6.1 DBNEWIDパラメータ

DBNEWIDユーティリティのパラメータについて説明します。

次の表に、DBNEWIDの構文のパラメータを示します。

表23-1 DBNEWIDユーティリティのパラメータ

パラメータ	説明
TARGET	データベースへの接続に使用するユーザー名およびパスワードを指定します。ユーザーには、SYSDBA 権限が必要です。オペレーティング・システム認証を使用している場合、スラッシュ(/)を使用して続けることができます。環境変数$ORACLE_HOME および$ORACLE_SID が正しく設定されていない場合、ターゲット・データベースへの接続にセキュリティで保護されたサービス(IPC または BEQ)を指定できます。DBNEWID ユーティリティを起動する場合は、常にターゲット・データベースを指定する必要があります。
REVERT	YES(デフォルトは NO)を指定すると、失敗した DBID の変更が元に戻ります。ターゲット・データベース上で DBID の変更操作が進行していない場合は、ユーティリティによってエラーが出力されます。DBID の変更が正常に完了した場合は、元に戻せません。REVERT=YES は、DBID の変更が正常に行われなかった場合にのみ有効です。
DBNAME=new_db_name	データベース名を変更します。データベースの DBID および DBNAME は同時に変更できます。DBNAME のみを変更するには、SETNAME パラメータも指定します。
SETNAME	YES(デフォルトは NO)を指定すると、DBNEWID ユーティリティによってデータベースの名前のみが変更され、DBID は変更されません。SETNAME=YES を指定すると、ユーティリティによってターゲット・データベースの制御ファイルのみに書き込まれます。
PDB	マルチテナント・コンテナ・データベース(CDB)内のすべてのプラガブル・データベース(PDB)の DBID を変更するか、どの PDB の DBID も変更しません。(デフォルトでは、コンテナ・データベース(CDB)に対して DBNEWID ユーティリティを実行すると、その CDB のみの DBID が変更され、その CDB に含まれるプラガブル・データベース(PDB)の DBID は変更されません。)PDB パラメータは、CDB 環境にのみ適用可能です。
LOGFILE=logfile	DBNEWID ユーティリティによってメッセージが書き込まれるファイルを指定します。デフォルトでは、古いログに上書きされます。ログ・ファイルを指定すると、DBNEWID ユーティリティによって確認が求められません。

パラメータ	説明
APPEND	YES(デフォルトは NO)を指定すると、既存のログ・ファイルにログの出力が追加されます。
HELP	YES(デフォルトは NO)を指定すると、DBNEWID ユーティリティの構文オプションのリストが出力されます。

親トピック: DBNEWIDの構文

23.6.2 制限と使用上のノート

DBNEWIDユーティリティの制限事項について説明します。

たとえば:

- データベースのDBIDを変更するには、データベースがマウントされており、マウント前に一貫性を維持した状態で停止されている必要があります。Oracle Real Application Clustersデータベースの場合は、データベースをNOPARALLELモードでマウントする必要があります。
- DBIDの変更後は、RESETLOGSオプションを使用してデータベースをオープンする必要があります。データベース名のみの変更後は、RESETLOGSオプションを使用してオープンする必要はありません。
- DBNEWIDユーティリティの実行中に、データベースに対して他のプロセスを実行することはできません。他のセッションがデータベースを停止して起動した場合、DBNEWIDユーティリティは異常終了します。
- すべてのオンライン・データ・ファイルは、リカバリが不要な、一貫性を維持した状態である必要があります。
- NORMALモードでオフラインにされたデータ・ファイルは、アクセスおよび書込みが可能である必要があります。そうでない場合は、DBNEWIDユーティリティを起動する前に、これらのファイルを削除する必要があります。
- すべての読取り専用表領域は、DBNEWIDユーティリティを起動する前に、オペレーティング・システム・レベルでアクセスおよび書込みが可能である必要があります。これらの表領域を書込み可能にできない場合(たとえば、表領域がCD-ROMにある場合)は、トランスポータブル表領域機能を使用して、DBNEWIDユーティリティを起動する前に、表領域を書込み可能な領域にトランスポートする必要があります。
- DBNEWIDユーティリティは、グローバル・データベース名を変更しません。詳細は、「グローバル・データベース名に関する考慮点」を参照してください。

親トピック: DBNEWIDの構文

23.6.3 Oracle Database 10gより前のリリースに対する制限事項

DBNEWIDユーティリティをリリース10.1より前のOracle Databaseに対し実行する場合の追加の制限事項について説明します。

たとえば:

- nid実行可能ファイルはデータ・ファイルおよび制御ファイルに直接アクセスする必要があるため、この実行可能ファイルはOracle所有者が所有し、実行する必要があります。他のユーザーがこのユーティリティを実行する場合は、そのユーザーIDをデータ・ファイルおよび制御ファイルの所有者に設定します。
- DBNEWIDユーティリティを使用するには、ローカル接続を介してデータベースのデータ・ファイルに直接アクセスする必

要があります。DBNEWIDユーティリティでは、ネット・サービス名は指定できますが、非ローカル・データベースのDBIDは変更できません。

親トピック: DBNEWIDの構文

24 REDOログ・ファイル分析のためのLogMinerの使用

Oracle Databaseの一部であるOracle LogMinerを使用すると、SQLインタフェースを介してオンラインおよびアーカイブのREDOログ・ファイルを問い合せることができます。

REDOログ・ファイルには、データベースでのアクティビティの履歴情報が含まれています。

次のトピックを参照してください。

LogMinerは、コマンドラインから使用するか、またはOracle LogMinerビューアのグラフィカル・ユーザー・インタフェースを介してアクセスできます。Oracle LogMinerビューアは、Oracle Enterprise Managerの一部です。Oracle LogMinerビューアの詳細は、Oracle Enterprise Managerオンライン・ヘルプを参照してください。

> ノート:
>
> `dbms_logmnr.start_logmnr` パッケージの `continuous_mine` オプションは、Oracle Database 19c (19.1)ではサポートされなくなり、使用できなくなりました。

- LogMinerのメリット
  ユーザー・データまたはデータベース・ディクショナリに対して行われたすべての変更は、データベースのリカバリ操作を実行できるように、Oracle REDOログ・ファイルに記録されます。
- LogMinerの概要
  これらのトピックでは、LogMinerの概要について次のトピックを説明します。
- CDBのLogMinerの使用
  LogMinerは、マルチテナント・コンテナ・データベース(CDB)で使用できます。
- LogMinerディクショナリ・ファイルとREDOログ・ファイル
  LogMinerを使用する前に、LogMinerがLogMinerディクショナリ・ファイルおよびREDOログ・ファイルとともにどのように動作するかを理解しておくことが重要です。これは、正確な結果の取得と、システム・リソースの使用計画の作成に役立ちます。
- LogMinerの起動
  LogMinerを起動するには、`DBMS_LOGMNR.START_LOGMNR`プロシージャをコールします。
- 分析するREDOデータについてのV$LOGMNR_CONTENTSの問合せ
  分析するREDOデータにアクセスするには、`V$LOGMNR_CONTENTS`ビューを問い合せます。
- V$LOGMNR_CONTENTSに返されるデータのフィルタ処理および書式設定
  LogMinerは、大量の情報を処理できます。`V$LOGMNR_CONTENTS`ビューに返される情報、および情報が返される速度を制限できます。
- V$LOGMNR_CONTENTSに返されるDDL文の再適用
  ユーザーが発行する一部のDDL文によって、1つ以上の他のDDL文がOracleによって内部的に実行される場合があります。
- DBMS_LOGMNR.START_LOGMNRの複数回のコール
  `DBMS_LOGMNR.START_LOGMNR`のコールに成功し、`V$LOGMNR_CONTENTS`ビューから選択した後でも、現在のLogMinerセッションを終了せずに`DBMS_LOGMNR.START_LOGMNR`を再度コールし、異なるオプションおよび時間範囲またはSCN範囲を指定できます。
- サプリメンタル・ロギング
  サプリメンタル・ロギングについて説明します。

- ビューでのLogMiner操作情報へのアクセス
  LogMiner操作情報(REDOデータではない)は、ビューに含まれています。
- 一般的なLogMinerセッションのステップ
  LogMinerを使用してデータを抽出およびマイニングできる一般的な方法について説明します。
- LogMinerの使用例
  データ・マイニングにLogMinerを使用する方法を確認するには、示されている例を確認してください。
- サポートされるデータ型、記憶域属性、およびデータベースとREDOログ・ファイルのバージョン
  データ型と記憶域属性のサポートに関する情報およびサポートされるデータベースとREDOログ・ファイルのリリースについて説明します。

親トピック: その他のユーティリティ

24.1 LogMinerのメリット

ユーザー・データやデータベース・ディクショナリに対するすべての変更は、Oracle REDOログ・ファイルに記録されます。これによってデータベース・リカバリ操作が実行できます。

LogMinerには、REDOログ・ファイルに対して適切に定義された使いやすい包括的なリレーショナル・インタフェースが用意されているため、強力なデータ監査ツールとしても、高度なデータ分析ツールとしても使用できます。次に、LogMinerの主要機能の一部を示します。

- アプリケーション・レベルで発生したエラーなど、データベースの論理的破損が発生した時期を特定します。WHERE句の値に誤りがあったために不適切な行を削除した、不適切な値で行を更新した、不適切な索引を消去したなどのエラーがこれに含まれます。たとえば、ユーザー・アプリケーションで、すべての社員の給与を10%増額するところを100%増額するように誤ってデータベースを更新してしまったり、データベース管理者(DBA)が誤って重要なシステム表を削除してしまったりすることがあります。エラーが発生した時期を正確に知ることは、時刻ベースまたは変更ベースのリカバリの開始時期の特定に役立ちます。この情報から、破損する直前の状態にデータベースをリストアできます。LogMinerを使用したこのような処理方法の詳細は、「列の値に基づいたV$LOGMNR_CONTENTSの問合せ」を参照してください。
- トランザクション・レベルでファイングレインなリカバリを行うために必要な処置を特定します。既存の依存性についての正しい理解と認識があれば、表固有のundo操作を実行し、表を元の状態に戻すことができます。そのために、LogMinerは、元のSQL文の発行順序を逆にたどる表固有の再構築されたSQL文を適用します。例については、「使用例1: LogMinerを使用した特定のユーザーによる変更の追跡」を参照してください。

  通常は、表を以前の状態にリストアしてから、アーカイブされたREDOログ・ファイルを適用して、表をロールフォワードします。
- 傾向分析により、パフォーマンス・チューニングとキャパシティ・プランニングを実行します。どの表で更新や挿入が最も多いかを判断できます。その情報は、ディスク・アクセス統計に関する履歴的観点を提供するため、チューニングに使用できます。例については、「使用例2: LogMinerを使用した表アクセス統計の計算」を参照してください。
- 事後監査を実行します。LogMinerを使用して、データベースで実行されたデータ操作言語(DML)文およびデータ定義言語(DDL)文、これらの文が実行された順序、および文の実行者を追跡できます。(ただし、LogMinerをそのような目的に使用するには、そのイベントが発生した時期を知り、分析に適したログを指定できる必要があります。そうでない場合、多数のREDOログ・ファイルのマイニングが必要となり、時間がかかる場合があります。LogMinerの使用は、データベース使用の監査のための補助と考えてください。データベースの監査の詳細は、『Oracle Database管理者ガイド』を参照してください。

親トピック: REDOログ・ファイル分析のためのLogMinerの使用

24.2 LogMinerの概要

これらのトピックでは、LogMinerの概要を説明します。

- LogMinerの構成
  LogMinerの分析対象オブジェクトについて説明します。また、構成ファイルの例も示します。
- LogMiner操作の指定および分析するデータの取得
  LogMiner操作の指定にはPL/SQLパッケージDBMS_LOGMNRおよびDBMS_LOGMNR_Dを、分析するデータの取得にはV$LOGMNR_CONTENTSビューを使用します。

親トピック: REDOログ・ファイル分析のためのLogMinerの使用

24.2.1 LogMiner構成

LogMinerの分析対象オブジェクトについて学習します。また、構成ファイルの例も示します。

- LogMiner構成ファイルのオブジェクト
  DataMiner構成ファイルには、ソース・データベース、マイニング・データベース、LogMinerディクショナリおよび注目するデータを格納するREDOログ・ファイルの4つのオブジェクトがあります。
- LogMinerの構成例
  この例では、あるOracle Databaseリリースのある場所にREDOログを生成して、そのログをリリースが異なる別のOracle Databaseの別の場所に送信する方法を示します。
- LogMinerの要件
  LogMinerでマイニングするデータベースのソースおよびマイニング・データベース、データ・ディクショナリ、REDOログ・ファイルおよび表と列の名前制限の要件について説明します。

親トピック: LogMinerの概要

24.2.1.1 LogMiner構成ファイルのオブジェクト

DataMiner構成ファイルには、ソース・データベース、マイニング・データベース、LogMinerディクショナリおよび注目するデータを格納するREDOログ・ファイルの4つのオブジェクトがあります。

- **ソース・データベース**は、LogMinerで分析するすべてのREDOログ・ファイルを生成するデータベースです。
- **マイニング・データベース**は、分析実行時にLogMinerにより使用されるデータベースです。
- **LogMinerディクショナリ**を使用すると、LogMinerがユーザーの要求したREDOログ・データを表すときに、内部オブジェクトIDではなく表名と列名を提供できます。

  LogMinerでディクショナリを使用して、内部オブジェクト識別子およびデータ型を、オブジェクト名および外部データ・フォーマットに変換します。ディクショナリがない場合、LogMinerから内部オブジェクトIDが返され、データはバイナリ・データとして表されます。

  たとえば、次のSQL文について検討します。

```
 INSERT INTO HR.JOBS(JOB_ID, JOB_TITLE, MIN_SALARY, MAX_SALARY)
VALUES('IT_WT','Technical Writer', 4000, 11000);
```

  LogMinerがLogMinerディクショナリなしで結果を配信するときには、LogMinerは次の出力を表示します。

```
insert into "UNKNOWN"."OBJ# 45522"("COL 1","COL 2","COL 3","COL 4") values
(HEXTORAW('45465f4748'),HEXTORAW('546563686e6963616c20577269746572'),
HEXTORAW('c229'),HEXTORAW('c3020b'));
```

- **REDOログ・ファイル**には、データベースまたはデータベース・ディクショナリに対して行われた変更が含まれます。

親トピック: LogMinerの構成

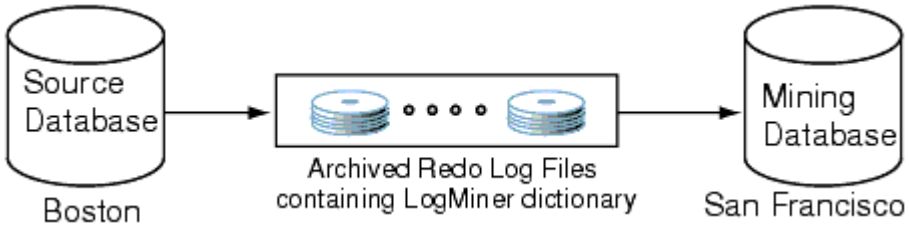
24.2.1.2 LogMinerの構成例

この例では、あるOracle Databaseリリースのある場所にREDOログを生成して、そのログをリリースが異なる別のOracle Databaseの別の場所に送信する方法を示します。

次の図に、LogMinerの構成例を示します。この図では、ソース・データベースがボストンにあり、ターゲット・データベースがサンフランシスコにあります。

ボストンにあるソース・データベースでREDOログ・ファイルが生成され、アーカイブ後にサンフランシスコにあるデータベースに送られます。LogMinerディクショナリは、これらのREDOログ・ファイルに対して抽出されます。LogMinerが実際にREDOログ・ファイルを分析するマイニング・データベースは、サンフランシスコにあります。ボストンのデータベースでOracle Database 12gが実行され、サンフランシスコのデータベースでOracle Database 19cが実行されています。

図24-1 LogMinerデータベースの構成例



この例は、有効なLogMiner構成の1つを示しているにすぎません。その他に有効な構成としては、ソース・データベースとマイニング・データベースの両方に同じデータベースを使用する構成、データ・ディクショナリを別の方法で提供する構成などがあります。

関連トピック

- LogMinerディクショナリ・オプション

親トピック: LogMinerの構成

24.2.1.3 LogMinerの要件

LogMinerでマイニングするデータベースのソースおよびマイニング・データベース、データ・ディクショナリ、REDOログ・ファイルおよび表と列の名前制限の要件について学習します。

LogMinerには、次のオブジェクトが必要です。

- 次の特性を持つソース・データベースおよびマイニング・データベース。
  - ソース・データベースおよびマイニング・データベースは、同一のハードウェア・プラットフォームで稼働している必要があります。
  - マイニング・データベースは、ソース・データベースと同一でも、まったく別であってもかまいません。
  - マイニング・データベースは、ソース・データベースと同位か、上位のリリースのOracle Databaseを使用して実行する必要があります。
  - マイニング・データベースでは、ソース・データベースで使用するものと同じ文字セット(または文字セットのスーパーセット)を使用する必要があります。
- LogMinerディクショナリ
  - ディクショナリは、LogMinerで分析する必要があるREDOログ・ファイルを生成するソース・データベースと同じデータベースで生成されている必要があります。
- すべてのREDOログ・ファイル(次の特性があるもの)。

- REDOログ・ファイルは、同じソース・データベースで生成する必要があります。
- REDOログ・ファイルは、同じデータベースのRESETLOGS SCNに関連付ける必要があります。
- REDOログ・ファイルは、リリース8.0以降のOracle Databaseのものであることが必要です。ただし、リリース9.0.1で導入されたLogMiner機能の一部は、Oracle9i以降のデータベースで生成されたREDOログ・ファイルでのみ機能します。
- マイニング用に選択する表または列の名前は、30文字以内にする必要があります。

ノート:

JSON 形式の拡張 varchar2 列および拡張 varchar 列名など、30 文字を超える拡張列形式を使用する Oracle Database 12c リリース 2 (12.2)の後に追加されたデータ型および機能は、DBMS_ROLLING PL/SQL パッケージ、Oracle GoldenGate および XStream からのみサポートされます。30 文字を超える仮想列名は、v$logmnr_contents (dba_logstdby_unsupported および dba_rolling_unsupported ビュー)の UNSUPPORTED です。

LogMinerでは、異なるデータベースからのREDOログ・ファイルを混在させること、また分析する必要があるREDOログ・ファイルを生成したデータベース以外からのディクショナリを使用することはできません。LogMinerには、30文字以下の表名または列名が必要です。

ノート:

LogMiner で分析するログ・ファイルを生成するには、その前にサプリメンタル・ロギングを有効にする必要があります。

サプリメンタル・ロギングが有効な場合、REDO ログ・ファイルの情報を役立てるために必要な補足情報が REDO ストリームに記録されます。したがって、次の SQL 文で示すように、少なくとも最小サプリメンタル・ロギングは有効にする必要があります。

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;
```

サプリメンタル・ロギングが有効であるかどうかを確認するには、次の SQL 文で示すように、V$DATABASE ビューを問い合せます。

```
SELECT SUPPLEMENTAL_LOG_DATA_MIN FROM V$DATABASE;
```

問合せから YES または IMPLICIT の値が返された場合、最小サプリメンタル・ロギングは有効です。

サプリメンタル・ロギングが有効な場合、LogMinerユーティリティ(DBMS_LOGMNR)では長い表名または列名はサポートされないことに注意してください。オンライン・ディクショナリを使用していて、サプリメンタル・ロギングが有効になっていない場合、v$logmnr_contentsにはすべての名前と、関連オブジェクトのsql_undoまたはsql_redoが表示されます。ただし、LogMinerユーティリティを使用するには、少なくとも最小サプリメンタル・ロギングを有効にする必要があります。表名または列名が30文字を超える表をマイニングすると、v$logmnr_contentsに次のようなエントリが表示されます。

```
select sql_redo , operation, seg_name, info
         from v$logmnr_contents  where seg_name =
        upper('my_table_with_a_very_very_long_name_for_test') or    seg_name =
        upper('table_with_long_col_name') ;
SQL_REDO --- OPERATION -- SEG_NAME ----------------------- INFO  --------------------
------
```

```
Unsupported  UNSUPPORTED  MY_TABLE_W_A_VERY_VERY_LONG_NAME Object or Data type
Unsupported
Unsupported  UNSUPPORTED  TABLE_WITH_LONG_COL_NAME         Object or Data type
Unsupported
```

したがって、30文字以下の名前を持つ表および列でLogMinerを使用します。

関連トピック

- サポートされるデータベースとREDOログ・ファイルのバージョン
- サプリメンタル・ロギング

親トピック: LogMinerの構成

24.2.2 LogMiner操作の指定および分析するデータの取得

LogMiner操作の指定にはPL/SQLパッケージDBMS_LOGMNRおよびDBMS_LOGMNR_Dを、分析するデータの取得にはV$LOGMNR_CONTENTSビューを使用します。

たとえば:

1. LogMinerディクショナリを指定します。

   使用するディクショナリのタイプに応じて、DBMS_LOGMNR_D.BUILDプロシージャを使用するか、LogMiner起動時(ステップ3)にディクショナリを指定するか、その両方を実行します。

2. 分析するREDOログ・ファイルのリストを指定します。

   DBMS_LOGMNR.ADD_LOGFILEプロシージャを使用するか、LogMiner起動時(ステップ3)に自動的に分析するログ・ファイルのリストをLogMinerが作成するよう指定します。

3. LogMinerを起動します。

   DBMS_LOGMNR.START_LOGMNRプロシージャを使用します。

4. 分析するREDOデータを要求します。

   V$LOGMNR_CONTENTSビューを問い合せます。

5. LogMinerセッションを終了します。

   DBMS_LOGMNR.END_LOGMNRプロシージャを使用します。

V$LOGMNR_CONTENTSビューを問い合せて、LogMiner PL/SQLパッケージを使用するには、EXECUTE_CATALOG_ROLEロールおよびLOGMINING権限を持っている必要があります。

> ノート:
>
> Oracle RAC データベースで生成されたアーカイブ・ログ内で、指定された分析する時間範囲または SCN 範囲のマイニングを行う場合、指定の時間範囲または SCN 範囲でアクティブであった、すべての REDO スレッドのすべてのアーカイブ・ログが指定されていることを確認する必要があります。指定されていない場合、V$LOGMNR_CONTENTS という問合せによって、(DBMS_LOGMNR.ADD_LOGFILE プロシージャを使用して LogMiner に指定されたアーカイブ・ログに基づき)部分的な結果のみが返されます。
>
> dbms_logmnr.start_logmnr パッケージの CONTINUOUS_MINE オプションは、Oracle Database 19c

(19.1)ではサポートされなくなり、使用できなくなりました。

関連項目:

LogMinerの使用例は、「一般的なLogMinerセッションのステップ」を参照してください。

親トピック: LogMinerの概要

24.3 CDBのLogMinerの使用

LogMinerは、マルチテナント・コンテナ・データベース(CDB)で使用できます。

次の項では、CDBと非CDBでLogMinerの使用時に注意する必要がある相違点について説明します。

ルート・コンテナにすべてのPDBのスーパーセットの文字セットが含まれている場合、LogMinerでは異なる文字セットのPDBを含むCDBがサポートされます。

マルチテナント環境を管理するには、CDB_DBAロールを持っている必要があります。

- CDBでのLogMiner V$ビューおよびDBAビュー
  CDBでは、システムで実行されているLogMinerセッションの情報を表示するためにLogMinerによって使用されるビューに、CON_IDという追加の列が含まれます。
- CDBでのV$LOGMNR_CONTENTSビュー
  CDBでは、V$LOGMNR_CONTENTSビューとその関連機能はルート・データベースに制限されています。V$LOGMNR_CONTENTSには、CDBを支援するいくつかの新しい列が存在します。
- CDBでのサプリメンタル・ロギングの有効化
  CDBで、データベース全体のサプリメンタル・ロギングを有効化および無効化する構文は、非CDBデータベースと同じです
- CDBでのフラット・ファイル・ディクショナリの使用
  単一のフラット・ファイルにCDB全体のディクショナリ・スナップショットを取得することはできません。個々のPDBに接続する必要があり、そのPDB専用のスナップショットをフラット・ファイルに取得できます。

関連トピック

- CDBでのLogMiner V$ビューおよびDBAビュー
- CDBでのV$LOGMNR_CONTENTSビュー
- CDBでのサプリメンタル・ロギングの有効化
- CDBでのフラット・ファイル・ディクショナリの使用

親トピック: REDOログ・ファイル分析のためのLogMinerの使用

24.3.1 CDBでのLogMiner V$ビューおよびDBAビュー

CDBでは、システムで実行されているLogMinerセッションの情報を表示するためにLogMinerによって使用されるビューに、CON_IDという追加の列が含まれます。

CON_ID列によって、情報が表示されるセッションに関連付けられたコンテナIDを識別します。プラガブル・データベース(PDB)のビューを問い合せると、データベースに関連付けられた情報のみが表示されます。この新しい動作の影響を受けるビューは次のとおりです。

- V$LOGMNR_DICTIONARY_LOAD

- V$LOGMNR_LATCH
- V$LOGMNR_PROCESS
- V$LOGMNR_SESSION
- V$LOGMNR_STATS

> ノート:
>
> CDB をサポートするため、V$LOGMNR_CONTENTS ビューには、CON_ID 以外にもいくつかの新しい列が含まれます。

次のDBAビューには、名前がCDBで始まる類似のCDBビューがあります。

ログ・ビューのタイプ	DBAビュー	CDBビュー
LogMiner ログ・ビュー	DBA_LOGMNR_LOG	CDB_LOGMNR_LOG
LogMiner パージ済ログ・ビュー	DBA_LOGMNR_PURGED_LOG	CDB_LOGMNR_PURGED_LOG
LogMiner セッション・ログ・ビュー	DBA_LOGMNR_SESSION	CDB_LOGMNR_SESSION

DBAビューには、問合せが行われたコンテナで定義されているセッションに関連する情報のみが表示されます。

CDBビューには、特定の行が表すデータを含むコンテナを識別する追加のCON_ID列が含まれます。ルートからCDBビューを問い合せると、それらのビューを使用してすべてのコンテナに関する情報を参照できます。

親トピック: CDBのLogMinerの使用

24.3.2 CDBでのV$LOGMNR_CONTENTSビュー

CDBでは、V$LOGMNR_CONTENTSビューとその関連機能はルート・データベースに制限されています。V$LOGMNR_CONTENTSには、CDBを支援するいくつかの新しい列が存在します。

- CON_ID—問合せの実行元のコンテナに関連付けられたIDが含まれます。V$LOGMNR_CONTENTSはルート・データベースに制限されるため、問合せがCDBで実行されると、この列は値1を返します。
- SRC_CON_NAME—PDBの名前。この情報は、マイニングがLogMinerディクショナリで実行される場合にのみ使用できます。
- SRC_CON_ID—REDOレコードを生成したPDBのコンテナID。この情報は、マイニングがLogMinerディクショナリで実行される場合にのみ使用できます。
- SRC_CON_DBID—PDBの識別子。この情報は、マイニングが現在のLogMinerディクショナリで実行される場合にのみ使用できます。
- SRC_CON_GUID—PDBに関連付けられたGUIDが含まれます。この情報は、マイニングが現在のLogMinerディクショナリで実行される場合にのみ使用できます。

親トピック: CDBのLogMinerの使用

24.3.3 CDBでのサプリメンタル・ロギングの有効化

CDBで、データベース全体のサプリメンタル・ロギングを有効化および無効化する構文は、非CDBデータベースと同じです

たとえば、サプリメンタル・ログ・データを追加または削除する場合は、次の構文を使用します。

```
ALTER DATABASE [ADD|DROP] SUPPLEMENTAL LOG DATA ...
```

ただし、次のことに注意してください。

- CDBでは、CDB$ROOTから有効にしたサプリメンタル・ロギング・レベルは、CDB全体で有効になります。
- 少なくとも最小サプリメンタル・ロギングがCDB$ROOTで有効になっている場合、追加のサプリメンタル・ロギング・レベルをPDBレベルで有効にできます。
- CDB$ROOTからCDBレベルで有効にしたサプリメンタル・ロギング・レベルは、PDBレベルでは無効にできません。
- CDB$ROOTからすべてのサプリメンタル・ロギングを削除した場合、以前のPDBレベルの設定に関係なく、CDB全体ですべてのサプリメンタル・ロギングが無効になります。

CREATE TABLE文およびALTER TABLE文で開始するサプリメンタル・ロギング操作は、CDBルートまたはPDBから実行できます。これらは、適用先の表にのみ影響します。

親トピック: CDBのLogMinerの使用

24.3.4 CDBでのフラット・ファイル・ディクショナリの使用

単一のフラット・ファイルにCDB全体のディクショナリ・スナップショットを取得することはできません。個々のPDBに接続する必要があり、そのPDB専用のスナップショットをフラット・ファイルに取得できます。

したがって、フラット・ファイル・ディクショナリを使用する場合、データ・ディクショナリがフラット・ファイル内に含まれるPDBに関連付けられた変更を対象とするREDOログのみをマイニングできます。

親トピック: CDBのLogMinerの使用

24.4 LogMinerディクショナリ・ファイルとREDOログ・ファイル

LogMinerを使用する前に、LogMinerがLogMinerディクショナリ・ファイルおよびREDOログ・ファイルとともにどのように動作するかを理解しておくことが重要です。これは、正確な結果の取得と、システム・リソースの使用計画の作成に役立ちます。

この項では、次の概念について説明します。

- LogMinerディクショナリ・オプション
  LogMinerは、REDOデータをユーザーに返す際に、オブジェクトIDをオブジェクト名に変換するためにディクショナリを必要とします。
- REDOログ・ファイル・オプション
  REDOログ・ファイルにあるデータをマイニングするには、LogMinerは、マイニングするREDOログ・ファイルの情報を必要とします。

親トピック: REDOログ・ファイル分析のためのLogMinerの使用

24.4.1 LogMinerディクショナリ・オプション

LogMinerは、REDOデータをユーザーに返す際に、オブジェクトIDをオブジェクト名に変換するためにディクショナリを必要としま

す。

LogMinerには、ディクショナリ提供方法として次の3つのオプションがあります。

- オンライン・カタログの使用

  REDOログ・ファイルが作成されたソース・データベースにアクセスでき、分析する表の列定義に変更がないことが見込まれる場合には、このオプションの使用をお薦めします。このオプションは、最も効率的で簡単に使用できます。

- REDOログ・ファイルへのLogMinerディクショナリの抽出

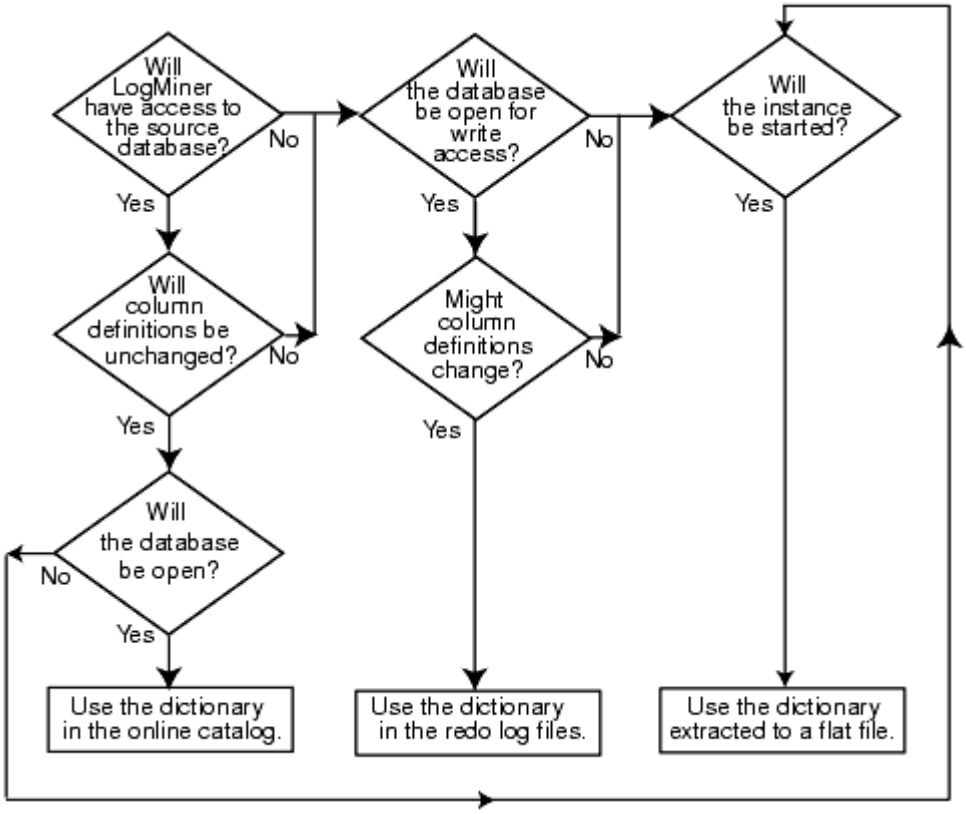
  REDOログ・ファイルが作成されたソース・データベースにアクセスできない場合、または分析する表の列定義が変更される可能性がある場合には、このオプションの使用をお薦めします。

- フラット・ファイルへのLogMinerディクショナリの抽出

  このオプションは、以前のリリースとの下位互換性を保つものです。このオプションは、トランザクションの一貫性を保証しません。オンライン・カタログを使用するか、REDOログ・ファイルにディクショナリを抽出することをお薦めします。

図24-2に、状況に応じたLogMinerディクショナリの選択方法を示します。

図24-2 LogMinerディクショナリの選択方法



次の項では、使用可能な各ディクショナリ・オプションの指定方法を説明します。

- オンライン・カタログの使用
  データベースで現在使用中のディクショナリを使用することをLogMinerに指定するには、LogMiner起動時にディクショナリ・ソースとしてオンライン・カタログを指定します。
- REDOログ・ファイルへのLogMinerディクショナリの抽出
  REDOログ・ファイルにLogMinerディクショナリを抽出するには、データベースがオープンしていて、ARCHIVELOGモードの状態で、アーカイブが有効になっている必要があります。
- フラット・ファイルへのLogMinerディクショナリの抽出
  LogMinerディクショナリがフラット・ファイルに存在する場合、REDOログ・ファイルに存在する場合よりもシステム・リソー

スの消費が少なくなります。

親トピック: LogMinerディクショナリ・ファイルとREDOログ・ファイル

24.4.1.1 オンライン・カタログの使用

データベースで現在使用中のディクショナリを使用することをLogMinerに指定するには、LogMiner起動時にディクショナリ・ソースとしてオンライン・カタログを指定します。

たとえば:

```
EXECUTE DBMS_LOGMNR.START_LOGMNR(-
   OPTIONS => DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG);
```

オンライン・カタログは、オンラインREDOログ・ファイルの分析に使用する他にも、アーカイブREDOログ・ファイルが同じシステム上で生成されている場合には、アーカイブREDOログ・ファイルの分析に使用できます。

オンライン・カタログにはデータベースに関する最新情報が含まれており、分析を開始するには最も速い方法です。重要な表を変更するDDL操作は頻繁に行われないため、通常、オンライン・カタログには分析に必要な情報が提供されています。

ただし、オンライン・カタログが再構成できるのは、表の最新バージョンに対して実行されるSQL文のみであることに注意してください。表が変更されると、オンライン・カタログは以前の表のバージョンを反映しなくなります。つまり、LogMinerは、以前の表のバージョンに対して実行されたSQL文を再構築することはできません。かわりに、LogMinerは、次に示す`V$LOGMNR_CONTENTS`ビューの`SQL_REDO`列で実行不可SQL(バイナリ値のhextoraw書式を含む)を生成します。

```
insert into HR.EMPLOYEES(col#1, col#2) values (hextoraw('4a6f686e20446f65'),
hextoraw('c306'));"
```

オンライン・カタログ・オプションでは、データベースがオープンしている必要があります。

オンライン・カタログ・オプションは、`DBMS_LOGMNR.START_LOGMNR`の`DDL_DICT_TRACKING`オプションに対し有効ではありません。

親トピック: LogMinerディクショナリ・オプション

24.4.1.2 REDOログ・ファイルへのLogMinerディクショナリの抽出

REDOログ・ファイルにLogMinerディクショナリを抽出するには、データベースがオープンしていて、`ARCHIVELOG`モードの状態で、アーカイブが有効になっている必要があります。

ディクショナリをREDOログ・ストリームに抽出している間は、DDL文を実行できません。したがって、REDOログ・ファイルに抽出されるディクショナリは、一貫性が保証されます(一方、フラット・ファイルに抽出されたディクショナリは一貫性が保証されません)。

ディクショナリ情報をREDOログ・ファイルに抽出するには、`STORE_IN_REDO_LOGS`オプションを指定してPL/SQL `DBMS_LOGMNR_D.BUILD`プロシージャを実行します。ファイル名や位置を指定しないでください。

```
EXECUTE DBMS_LOGMNR_D.BUILD( -
   OPTIONS=> DBMS_LOGMNR_D.STORE_IN_REDO_LOGS);
```

関連項目:

- `ARCHIVELOG`モードの詳細は、『Oracle Database管理者ガイド』を参照してください。
- `DBMS_LOGMNR_D.BUILD`の詳細は、『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』を参照してください。

ディクショナリをREDOログ・ファイルに抽出するプロセスはデータベース・リソースを消費しますが、ピーク時間帯を外して抽出を実行することで問題を解決できます。また、フラット・ファイルに抽出するより高速になります。ディクショナリのサイズに応じて、複数のREDOログ・ファイルを含めることもできます。関係するREDOログ・ファイルがアーカイブされている場合、抽出されたディクショナリの先頭と終わりがどのREDOログ・ファイルにあるかを特定することができます。これを行うには、次のとおりV$ARCHIVED_LOGビューを問い合せます。

```
SELECT NAME FROM V$ARCHIVED_LOG WHERE DICTIONARY_BEGIN='YES';
SELECT NAME FROM V$ARCHIVED_LOG WHERE DICTIONARY_END='YES';
```

LogMinerセッションの起動を準備するときに、ADD_LOGFILEプロシージャで、先頭と終わりのREDOログ・ファイル名(およびその間の他のREDOログ)を指定してください。

定期的にREDOログ・ファイルのバックアップを取って情報を保存し、後で利用できるようにしておくことをお薦めします。データベースが適切に管理されている理想的な環境では、アーカイブされたREDOログ・ファイルのバックアップとリストアのためのプロセスがすでに整っているため、そのための特別なステップは必要ありません。バックアップ作業も時間がかかる作業のため、ピークを外した時間帯に実行することをお薦めします。

親トピック: LogMinerディクショナリ・オプション

24.4.1.3 フラット・ファイルへのLogMinerディクショナリの抽出

LogMinerディクショナリがフラット・ファイルに存在する場合、REDOログ・ファイルに存在する場合よりもシステム・リソースの消費が少なくなります。

> ノート:
>
> プラガブル・データベース(PDB)のフラット・ファイル・ディクショナリ・ダンプを作成する機能は、Oracle Database 19c でサポートが終了し、以降のリリースで完全にサポートが終了しました。
>
> 以前のリリースでは、フラット・ファイル・ディクショナリの使用は、データ・ディクショナリがフラット・ファイル内に含まれていた特定の PDB に関連付けられた変更の REDO ログをマイニングする手段でした。Oracle では、DBMS_LOGMNR.START_LOGMNR をコールし、マイニングするシステム変更番号(SCN)または時間範囲を指定することをお薦めします。START_LOGMNR の SCN または時間範囲のオプションは、個々の PDB のマイニングをサポートするように拡張されています。

データベース・ディクショナリ情報をフラット・ファイルに抽出するには、STORE_IN_FLAT_FILEオプションを指定してDBMS_LOGMNR_D.BUILDプロシージャを実行します。古いREDOログ・ファイルを正しく分析できるように、定期的にディクショナリ抽出のバックアップを取ることをお薦めします。

ディクショナリをフラット・ファイルに抽出するステップは、次のとおりです。ステップ1と2は準備のためのステップです。これらは、1回のみの実行で、以後は何回でもディクショナリをフラット・ファイルに抽出できます。

1. DBMS_LOGMNR_D.BUILDプロシージャは、ディクショナリ・ファイルが置かれたディレクトリにアクセスする必要があります。通常、PL/SQLプロシージャは、ユーザー・ディレクトリにアクセスしないため、ディレクトリの場所を指定する必要があります。ディレクトリを指定しない場合、プロシージャの実行は失敗します。ディレクトリの場所はディレクトリ・オブジェクトである必要があります。次に、SQL CREATE DIRECTORY文を使用してパス/oracle/databaseのmy_dictionary_dirというディクショナリ・オブジェクトを作成する例を示します。

   ```
   SQL> CREATE DIRECTORY "my_dictionary_dir" AS '/oracle/database';
   ```

ノート:

Oracle Database 12c リリース 2 (12.2)より前には、UTL_FILE_DIR 初期化パラメータを使用してディレクトリの場所を指定しました。ただし、Oracle Database 18c からは、UTL_FILE_DIR 初期化パラメータはサポート対象外になりました。これは下位互換性のために引き続きサポートされていますが、かわりにディレクトリ・オブジェクトを使用することをお薦めします。

2. データベースがクローズされている場合、SQL*Plusを使用してマウントしてから、分析するREDOログ・ファイルが含まれるデータベースをオープンします。たとえば、SQL STARTUPコマンドを入力すると、データベースがマウントされ、オープンします。

```
SQL> STARTUP
```

3. DBMS_LOGMNR_D.BUILD PL/SQLプロシージャを実行します。次の例では、ステップ1で作成したディレクトリ・オブジェクトmy_dictionary_dir内のdictionary.oraという名前のフラット・ファイルにLogMinerディレクトリ・ファイルを抽出します。

```
SQL> EXECUTE dbms_logmnr_d.build(dictionary_location=>'my_dictionary_dir', -
                                 dictionary_filename=>'dictionary.ora', -
                                 options => dbms_logmnr_d.store_in_flat_file);
```

STORE_IN_FLAT_FILEオプションを指定せずに、ファイル名と位置を指定することもできます。結果は同じになります。

関連トピック

- LogMinerの起動
- SCNによるデータのフィルタ処理

親トピック: LogMinerディクショナリ・オプション

24.4.2 REDOログ・ファイル・オプション

REDOログ・ファイルにあるデータをマイニングするには、LogMinerは、マイニングするREDOログ・ファイルの情報を必要とします。

これらのREDOログ・ファイルで検出されたデータベースに対する変更は、V$LOGMNR_CONTENTSビューを介してユーザーに表示されます。

分析するREDOログ・ファイルのリストを自動的かつ動的に作成するようにLogMinerに対して指定したり、LogMinerが分析するREDOログ・ファイルのリストを明示的に指定することができます。その方法は次のとおりです。

- 自動

  LogMinerをソース・データベースで使用する場合、LogMinerに対して、分析するREDOログ・ファイルを自動的に検出し、リストを作成するように指定できます。この例では、オンライン・カタログからのディクショナリを指定していますが、任意のLogMinerディクショナリを使用できます。

ノート:

dbms_logmnr.start_logmnr パッケージの continuous_mine オプションは、Oracle Database 19c (19.1)ではサポートされなくなり、使用できなくなりました。

LogMinerは、データベース制御ファイルを使用して、指定された時間範囲またはSCN範囲を満たすREDOログ・ファイルを検出し、LogMiner REDOログ・ファイル・リストに追加します。たとえば:

```
ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';
EXECUTE DBMS_LOGMNR.START_LOGMNR( -
   STARTTIME => '01-Jan-2012 08:30:00', -
   ENDTIME => '01-Jan-2012 08:45:00', -
   OPTIONS => DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG + -
   );
```

(この例では、DBMS_LOGMNR.START_LOGMNRプロシージャに対するPL/SQLコールで日付書式を指定する必要がないように、最初にSQL ALTER SESSION SET NLS_DATE_FORMAT文を使用しています。)

DBMS_LOGMNR.ADD_LOGFILEを使用して1つのREDOログ・ファイルのみを指定することにより、分析するREDOログ・ファイルのリストをLogMinerに自動的に作成するよう指示することもできます。ただし、前述した方法がより一般的です。

- 手動

LogMinerを起動する前に手動でREDOログ・ファイルのリストを作成するには、DBMS_LOGMNR.ADD_LOGFILEプロシージャを使用します。最初のREDOログ・ファイルがこのリストに追加された後、それ以降に追加される各REDOログ・ファイルは、同じデータベースからのもので、同じデータベースのRESETLOGS SCNと関連付けられている必要があります。この方法を使用する場合、LogMinerはソース・データベースに接続されている必要はありません。

たとえば、REDOログ・ファイルの新規リストを開始するには、PL/SQLプロシージャDBMS_LOGMNR.ADD_LOGFILEのNEWオプションを指定し、新規リストの開始であることを指定します。たとえば、/oracle/logs/log1.fを指定するには、次のように入力します。

```
EXECUTE DBMS_LOGMNR.ADD_LOGFILE( -
   LOGFILENAME => '/oracle/logs/log1.f', -
   OPTIONS => DBMS_LOGMNR.NEW);
```

必要に応じて、PL/SQL DBMS_LOGMNR.ADD_LOGFILEプロシージャのADDFILEオプションを指定し、さらにREDOログ・ファイルを追加します。たとえば、/oracle/logs/log2.fを追加するには、次のように入力します。

```
EXECUTE DBMS_LOGMNR.ADD_LOGFILE( -
   LOGFILENAME => '/oracle/logs/log2.f', -
   OPTIONS => DBMS_LOGMNR.ADDFILE);
```

現在のLogMinerセッションでどのREDOログ・ファイルが分析されているかを確認するには、V$LOGMNR_LOGSビューを問い合せます。このビューでは、REDOログ・ファイルごとに1行で構成しています。

親トピック: LogMinerディクショナリ・ファイルとREDOログ・ファイル

24.5 LogMinerの起動

DBMS_LOGMNR.START_LOGMNRプロシージャを呼び出して、LogMinerを起動します。

DBMS_LOGMNR.START_LOGMNRプロシージャで使用できるオプションは、V$LOGMNR_CONTENTSビューに対する出力を制御します。そのためV$LOGMNR_CONTENTSビューを問い合せる前に、DBMS_LOGMNR.START_LOGMNRをコールする必要があります。

LogMiner起動時には、次の項目を指定できます。

- LogMinerが返すデータをフィルタ処理する方法(たとえば、開始時刻と終了時刻、またはSCN値によるフィルタ処理)

- LogMinerにより返されるデータを書式設定するためのオプション
- 使用するLogMinerディクショナリ

次のリストに、DBMS_LOGMNR.START_LOGMNRに対してOPTIONSパラメータで指定できるLogMiner設定の要約と、詳細情報の参照先を示します。

- DICT_FROM_ONLINE_CATALOG
- DICT_FROM_REDO_LOGS
- COMMITTED_DATA_ONLY
- SKIP_CORRUPTION
- NO_SQL_DELIMITER
- PRINT_PRETTY_SQL
- NO_ROWID_IN_STMT
- DDL_DICT_TRACKING

DBMS_LOGMNR.START_LOGMNRプロシージャの実行時には、指定されたパラメータとオプションの組合せが有効かどうか、指定されたディクショナリとREDOログ・ファイルが使用可能かどうかがLogMinerによりチェックされます。ただし、V$LOGMNR_CONTENTSビューは、ビューを問い合せるまで移入されません。

パラメータおよびオプションは、次回のDBMS_LOGMNR.START_LOGMNRへのコールでは保持されません。DBMS_LOGMNR.START_LOGMNRをコールするたびに、必要なパラメータおよびオプション(SCN範囲や時間範囲を含む)をすべて指定する必要があります。

親トピック: REDOログ・ファイル分析のためのLogMinerの使用

24.6 分析するREDOデータについてのV$LOGMNR_CONTENTSの問合せ

分析するREDOデータにアクセスするには、V$LOGMNR_CONTENTSビューを問い合せます。

(V$LOGMNR_CONTENTSを問い合せるには、SYSDBAまたはLOGMINING権限が必要なことに注意してください。)このビューは、データベースに対して行われた変更に関する次のような履歴情報を提供します(次に示す以外の情報もあります)。

- データベースに対して行われた変更のタイプ: INSERT、UPDATE、DELETEまたはDDL (OPERATION列)。
- 変更が行われたSCN(SCN列)。
- 変更がコミットされたSCN(COMMIT_SCN列)。
- 変更が属するトランザクション(XIDUSN列、XIDSLT列およびXIDSQN列)。
- 変更されたオブジェクトの表とスキーマの名前(SEG_NAME列およびSEG_OWNER列)。
- 変更を行うDDL文またはDML文を発行したユーザーの名前(USERNAME列)。
- SQL DML文による変更の場合では、REDOレコードを生成するために使用されたSQL DMLと等価のSQL DML(必ずしも同一ではない)を表すように再構築されたSQL文(SQL_REDO列)。
- パスワードがSQL_REDO列にある文に含まれる場合のパスワードの暗号化。DDL文に対応するSQL_REDO列の値は、REDOレコードを生成するために使用されるSQL DDLと常に同一です。
- SQL DML変更による変更の場合では、変更を元に戻すために必要なSQL DML文を示すように再構築されたSQL

文(SQL_UNDO列)。

DDL文に対応するSQL_UNDO列は常にNULLです。一部のデータ型とロールバック操作についても、SQL_UNDO列はNULLです。

ノート:

LogMiner は透過的データ暗号化(TDE),をサポートしているため、対象オブジェクト用のメタデータが LogMiner データ・ディクショナリに含まれ、適切なマスター・キーが Oracle ウォレットにある場合は、暗号化された列(更新中の暗号化された列を含む)が含まれる表で実行された DML 操作が V$LOGMNR_CONTENTS に表示されます。ウォレットはオープンされている必要があります。オープンされていない場合は、V$LOGMNR_CONTENTS では関連付けられた REDO レコードを解釈することができません。データベースがオープンされていない(読取り専用、または読取り/書込み可能のいずれかの)場合、TDE はサポートされません。

関連項目:

TDEの詳細は、『Oracle Database Advanced Securityガイド』を参照してください。

V$LOGMNR_CONTENTSの問合せの例

Ronという名前のユーザーがoe.orders表で行ったすべての削除操作を調べる場合、次のようなSQL問合せを発行します。

```
SELECT OPERATION, SQL_REDO, SQL_UNDO
   FROM V$LOGMNR_CONTENTS
   WHERE SEG_OWNER = 'OE' AND SEG_NAME = 'ORDERS' AND
   OPERATION = 'DELETE' AND USERNAME = 'RON';
```

問合せによって生成される出力は次のとおりです。画面上での書式設定は、ここに示したものと異なる場合があります。

```
OPERATION   SQL_REDO                          SQL_UNDO
DELETE      delete from "OE"."ORDERS"         insert into "OE"."ORDERS"
            where "ORDER_ID" = '2413'         ("ORDER_ID","ORDER_MODE",
            and "ORDER_MODE" = 'direct'        "CUSTOMER_ID","ORDER_STATUS",
            and "CUSTOMER_ID" = '101'          "ORDER_TOTAL","SALES_REP_ID",
            and "ORDER_STATUS" = '5'           "PROMOTION_ID")
            and "ORDER_TOTAL" = '48552'        values ('2413','direct','101',
            and "SALES_REP_ID" = '161'         '5','48552','161',NULL);
            and "PROMOTION_ID" IS NULL
            and ROWID = 'AAAHTCAABAAAZAPAAN';
DELETE      delete from "OE"."ORDERS"          insert into "OE"."ORDERS"
            where "ORDER_ID" = '2430'          ("ORDER_ID","ORDER_MODE",
            and "ORDER_MODE" = 'direct'         "CUSTOMER_ID","ORDER_STATUS",
            and "CUSTOMER_ID" = '101'           "ORDER_TOTAL","SALES_REP_ID",
            and "ORDER_STATUS" = '8'            "PROMOTION_ID")
            and "ORDER_TOTAL" = '29669.9'       values('2430','direct','101',
            and "SALES_REP_ID" = '159'          '8','29669.9','159',NULL);
            and "PROMOTION_ID" IS NULL
            and ROWID = 'AAAHTCAABAAAZAPAAe';
```

この出力は、ユーザーRonがoe.orders表から2行を削除したことを示しています。再構築されたSQL文は、Ronが発行した実際の文と等価ですが、必ずしも同一とはかぎりません。この違いの理由は、元のWHERE句の記録がREDOログ・ファイルになく、そのため、LogMinerは、削除(あるいは更新または挿入)された行のみを個別に表示するためです。

したがって、両方の行の削除に1つのDELETE文のみが関係している場合でも、V$LOGMNR_CONTENTSの出力にはその事実が反映されません。実際のDELETE文は、DELETE FROM OE.ORDERS WHERE CUSTOMER_ID ='101'または

DELETE FROM OE.ORDERS WHERE PROMOTION_ID = NULLのいずれかです。

- V$LOGMNR_CONTENTSビューへの移入方法
  V$LOGMNR_CONTENTS固定ビューは、他のビューと異なり、表に格納されたデータの選択的表示ではありません。ユーザーがREDOログ・ファイルに対して要求したデータのリレーショナルな表示です。
- 列の値に基づいたV$LOGMNR_CONTENTSの問合せ
  LogMinerは、列の値に基づいた問合せを行うことができます。
- XMLType列およびXMLType表に基づいたV$LOGMNR_CONTENTSの問合せ
  LogMinerでは、XMLType列に対して生成されるREDOがサポートされます。CLOBとして格納されるXMLTypeデータは、11.0.0.0以上の互換性設定で生成されたREDOに対してサポートされます。

親トピック: REDOログ・ファイル分析のためのLogMinerの使用

24.6.1 V$LOGMNR_CONTENTSビューへの移入方法

V$LOGMNR_CONTENTS固定ビューは、他のビューと異なり、表に格納されたデータの選択的表示ではありません。ユーザーがREDOログ・ファイルに対して要求したデータのリレーショナルな表示です。

LogMinerがこのビューに移入するのは、このビューに対する問合せに応答する場合のみです。V$LOGMNR_CONTENTSを問い合せる前に、LogMinerを正常に起動しておく必要があります。

SQL選択操作がV$LOGMNR_CONTENTSビューに対して実行された場合、REDOログ・ファイルは順番に読み込まれます。REDOログ・ファイルから変換された情報が、V$LOGMNR_CONTENTSビューの行として返されます。これは、起動時に指定されたフィルタ基準を満たすまで、またはREDOログ・ファイルの終わりに達するまで繰り返されます。

V$LOGMNR_CONTENTSの特定の列が移入されない場合があります。たとえば:

- TABLE_SPACE列は、OPERATION列の値がDDLの行には移入されません。DDLは、複数の表領域で動作する可能性があるためです。たとえば、表は、複数の表領域にわたる複数のパーティションで作成される可能性があるため、列に正しく移入されません。
- LogMinerでは、一時表に対するSQLのREDOまたはSQLのUNDOは生成されません。SQL_REDO列には「/* No SQL_REDO for temporary tables */」、SQL_UNDO列には「/* No SQL_UNDO for temporary tables */」という文字列が表示されます。

LogMinerでは、COMMITTED_DATA_ONLYオプションを使用して、コミット済トランザクションのみを取得するように指定しないかぎり、すべての行がSCN順に返されます。SCN順は、メディア・リカバリで通常適用される順序です。

関連項目:

DBMS_LOGMNR.START_LOGMNRに対するCOMMITTED_DATA_ONLYオプションの詳細は、「コミット済トランザクションのみの表示」を参照してください。

ノート:

LogMiner は、問合せに応答する場合にのみ V$LOGMNR_CONTENTS ビューに移入し、要求したデータをデータベースに格納しないため、次の条件が適用されます。

- V$LOGMNR_CONTENTS を問い合せるたびに、LogMiner は、要求したデータの REDO ログ・ファイル

を分析します。

- 問合せで消費されるメモリー量は、問合せを満たすために返す必要がある行数によっては異なりません。
- 要求したデータを返すためにかかる時間は、そのデータを検出するためにマイニングする必要があるREDO ログ・データの量および型によって異なります。

前述の理由から、さらに分析を行うためにデータを保持する必要がある場合、特に問合せによって返されるデータの量が、そのデータを提供するためにLogMinerによって分析されるREDOデータの量と比較して少ない場合は、V$LOGMNR_CONTENTSという問合せの結果を一時的に格納する表を作成することをお薦めします。

親トピック: 分析するREDOデータについてのV$LOGMNR_CONTENTSの問合せ

24.6.2 列の値に基づいたV$LOGMNR_CONTENTSの問合せ

LogMinerは、列の値に基づいた問合せを行うことができます。

たとえば、hr.employees表に対して行われた、salaryを一定額より多く増額するすべての更新を示す問合せを実行できます。このようなデータは、システム動作の分析や監査タスクの実行に使用できます。

LogMinerによるREDOログ・ファイルからのデータ抽出は、DBMS_LOGMNR.MINE_VALUEおよびDBMS_LOGMNR.COLUMN_PRESENTという2つのマイニング関数を使用して実行されます。これらのマイニング関数に対するサポートは、V$LOGMNR_CONTENTSビューのREDO_VALUE列とUNDO_VALUE列によって提供されます。

次に、MINE_VALUE関数を使用して、hr.employees表に対して行われた、salary列を元の値の2倍より多く増額したすべての更新を選択する例を示します。

```
SELECT SQL_REDO FROM V$LOGMNR_CONTENTS
   WHERE
   SEG_NAME = 'EMPLOYEES' AND
   SEG_OWNER = 'HR' AND
   OPERATION = 'UPDATE' AND
   DBMS_LOGMNR.MINE_VALUE(REDO_VALUE, 'HR.EMPLOYEES.SALARY') >
   2*DBMS_LOGMNR.MINE_VALUE(UNDO_VALUE, 'HR.EMPLOYEES.SALARY');
```

この例で示したとおり、MINE_VALUE関数には2つの引数があります。

- 最初の引数は、データのREDO部分(REDO_VALUE)またはUNDO部分(UNDO_VALUE)のいずれをマイニングするかを指定します。データのREDO部分とは、挿入、更新または削除操作後に列に含まれるデータです。データのUNDO部分とは、挿入、更新または削除操作前に列に存在していたデータです。REDO_VALUEは新しい値、UNDO_VALUEは古い値と考えてください。
- 2番目の引数は、マイニングする列の完全修飾名を指定する文字列(この場合は、hr.employees.salary)です。MINE_VALUE関数からは、常に、元のデータ型に戻すことができる文字列が返されます。

- MINE_VALUE関数によって返されるNULL値の意味
  MINE_VALUE関数によって返されるNULL値の意味を説明します。
- MINE_VALUE関数およびCOLUMN_PRESENT関数の使用規則
  MINE_VALUE関数およびCOLUMN_PRESENT関数に適用される使用規則を説明します。
- MINE_VALUEファンクションを使用してNCHAR値を取得する際の制限事項
  MINE_VALUEファンクションを使用する際の制限事項について説明します。

親トピック: 分析するREDOデータについてのV$LOGMNR_CONTENTSの問合せ

24.6.2.1 MINE_VALUE関数によって返されるNULL値の意味

MINE_VALUE関数によって返されるNULL値の意味を説明します。

MINE_VALUE関数がNULL値を返した場合は、次のいずれかを意味します。

- 指定した列がデータのREDO部分またはUNDO部分に存在していない。
- 指定した列は存在するが、その値がNULL。

この2つの場合を区別するには、DBMS_LOGMNR.COLUMN_PRESENT関数を使用します。この関数では、列がデータのREDO部分またはUNDO部分に存在している場合は、1が返されます。存在していない場合には、0が返されます。たとえば、salary列の値の増額および対応するトランザクション識別子を検索するとします。次のSQL問合せを発行します。

```
SELECT
  (XIDUSN || '.' || XIDSLT || '.' || XIDSQN) AS XID,
  (DBMS_LOGMNR.MINE_VALUE(REDO_VALUE, 'HR.EMPLOYEES.SALARY') -
   DBMS_LOGMNR.MINE_VALUE(UNDO_VALUE, 'HR.EMPLOYEES.SALARY')) AS INCR_SAL
   FROM V$LOGMNR_CONTENTS
   WHERE
   OPERATION = 'UPDATE' AND
   DBMS_LOGMNR.COLUMN_PRESENT(REDO_VALUE, 'HR.EMPLOYEES.SALARY') = 1 AND
   DBMS_LOGMNR.COLUMN_PRESENT(UNDO_VALUE, 'HR.EMPLOYEES.SALARY') = 1;
```

親トピック: 列の値に基づいたV$LOGMNR_CONTENTSの問合せ

24.6.2.2 MINE_VALUE関数およびCOLUMN_PRESENT関数の使用規則

MINE_VALUE関数およびCOLUMN_PRESENT関数に適用される使用規則を説明します。

具体的には次のとおりです。

- LogMinerセッション内でのみ使用できます。
- V$LOGMNR_CONTENTSビューからの選択操作中に起動する必要があります。
- LONG、LONG RAW、CLOB、BLOB、NCLOB、ADTまたはCOLLECTIONデータ型はサポートされません。

親トピック: 列の値に基づいたV$LOGMNR_CONTENTSの問合せ

24.6.2.3 MINE_VALUEファンクションを使用してNCHAR値を取得する際の制限事項

MINE_VALUEファンクションを使用する際の制限事項について説明します。

DBMS_LOGMNR.MINE_VALUEファンクションを使用してデータベース文字セットに存在しない文字を含むNCHAR値を取得する場合、それらの文字は、データベース文字セットの置換文字(逆向きの疑問符など)として返されます。

親トピック: 列の値に基づいたV$LOGMNR_CONTENTSの問合せ

24.6.3 XMLType列およびXMLType表に基づいたV$LOGMNR_CONTENTSの問合せ

LogMinerでは、XMLType列に対して生成されるREDOがサポートされます。CLOBとして格納されるXMLTypeデータは、11.0.0.0以上の互換性設定で生成されたREDOに対してサポートされます。

- XMLType列およびXMLType表に基づくV$LOGMNR_CONTENTSの問合せ方法
  使用するXMLTypeストレージに応じて、LogMinerでは、様々な方法でV$LOGMNR_CONTENTSにSQL_REDOが表示されます。

- XMLTypeデータを使用するLogMinerの使用上の制限
  XMLTypeデータを使用するLogMinerの使用上の制限を説明します。
- XMLTypeデータを作成するためのPL/SQLプロシージャの例
  表外のXMLデータを含む表に対してXML REDOをマイニングおよび作成するのに使用可能なプロシージャについて説明する例。

親トピック: 分析するREDOデータについてのV$LOGMNR_CONTENTSの問合せ

24.6.3.1 XMLType列およびXMLType表に基づいたV$LOGMNR_CONTENTSの問合せ方法

使用するXMLTypeストレージに応じて、LogMinerは様々な方法でV$LOGMNR_CONTENTS内のSQL_REDOを表示します。

オブジェクト・リレーショナルおよびバイナリXMLとして格納されるXMLTypeデータは、11.2.0.3以上の互換性設定で生成されたREDOに対してサポートされます。

LogMinerでは、XMLTypeが何で保存されているかに応じて、V$LOGMNR_CONTENTS内のSQL_REDOを別の方法で示します。いずれの場合においても、STATUS列に加えSQL_REDO列の内容は、注意深く確認し、変更をREDOするためにSQLまたはPL/SQL文が生成される前に、通常は再アセンブリする必要があります。このような変更の構築に、SQL_REDOデータを使用できない場合があります。次の項の例では、XMLTypeは行を完全に変更するための再構築で最も単純なCLOBとして格納されています。

> ノート:
>
> CLOB として格納された XMLType データが、Oracle Database 12c リリース 1 (12.1)で非推奨になり、サポートされなくなる可能性があります。

XMLType列を使用した表に対して行われた変更についてのV$LOGMNR_CONTENTSの問合せ

この項の例では、次の列を使用したXML_CLOB_COL_TABという表について説明します。

- f1 NUMBER
- f2 VARCHAR2(100)
- f3 XMLTYPE
- f4 XMLTYPE
- f5 VARCHAR2(10)

ログおよびCOMMITTED_DATA_ONLYオプションを使用して、LogMinerセッションを起動したとします。
XML_CLOB_COL_TAB表に対して行われた変更について、V$LOGMNR_CONTENTSに対して次の問合せが実行されます。

```
SELECT OPERATION, STATUS, SQL_REDO FROM V$LOGMNR_CONTENTS
  WHERE SEG_OWNER = 'SCOTT' AND TABLE_NAME = 'XML_CLOB_COL_TAB';
```

問合せの出力は、次のようになります。

```
OPERATION          STATUS  SQL_REDO
INSERT             0       insert into "SCOTT"."XML_CLOB_COL_TAB"("F1","F2","F5")
values
                              ('5010','Aho40431','PETER')

XML DOC BEGIN      5       update "SCOTT"."XML_CLOB_COL_TAB" a set a."F3" =
XMLType(:1)
```

```
                                where a."F1" = '5010' and a."F2" = 'Aho40431' and a."F5"
= 'PETER'
XML DOC WRITE     5       XML Data
XML DOC WRITE     5       XML Data
XML DOC WRITE     5       XML Data
XML DOC END       5
```

XML DOC WRITE操作のSQL_REDO列には、XML文書の実際のデータがあります。データは文字列'XML Data'ではありません。

XMLType列を使用した表に挿入するための一般的なモデルを示すこの出力は、次のようになります。

1. すべてのスカラー列を使用した最初の挿入
2. バインド変数を使用して、1つのXMLType列に値を設定する更新文を使用したXML DOC BEGIN操作
3. XML文書のデータを使用した1回以上のXML DOC WRITE操作
4. XML文書のすべてのデータが送信されたことを示すXML DOC END操作
5. 表に複数のXMLType列がある場合は、元のDMLによって変更されるそれぞれのXMLType列に対して、ステップ2から4が繰り返されます。

XML文書が表外の列として格納されていない場合、その列に対してXML DOC BEGIN、XML DOC WRITEまたはXML DOC END操作は行われません。文書は、次のような更新文に含まれます。

```
OPERATION   STATUS          SQL_REDO
UPDATE      0               update "SCOTT"."XML_CLOB_COL_TAB" a
                            set a."F3" = XMLType('<?xml version="1.0"?>
                            <PO pono="1">
                            <PNAME>Po_99</PNAME>
                            <CUSTNAME>Dave Davids</CUSTNAME>
                            </PO>')
                            where a."F1" = '5006' and a."F2" = 'Janosik' and a."F5" =
'MMM'
```

XMLType表に対して行われた変更についてのV$LOGMNR_CONTENTSの問合せ

XMLType表に対するDMLは、XMLType列に対するDMLとは多少異なります。XML文書では、XMLType表の行に対する値を表します。XMLType列の場合とは異なり、最初の挿入に続いてXML文書を含む更新を行うことはできません。データを表に挿入する前に、文書全体を作成する必要があります。

XMLType表のもう1つの相違点は、OBJECT_ID列の存在です。オブジェクト識別子は、オブジェクト表の各オブジェクトを一意に識別するために使用されます。XMLType表については、表に行が挿入されると、Oracle Databaseによってこの値が生成されます。OBJECT_IDの値は、SQLを使用して表に直接挿入することはできません。したがって、LogMinerでは、この値を含む実行可能なSQL_REDOを生成できません。

V$LOGMNR_CONTENTSビューには、XMLType表に対して行われた変更に対して移入される新しいOBJECT_ID列があります。この値は、元の表のオブジェクト識別子です。ただし、この同じXML文書が同じXMLType表に挿入されても、新しいオブジェクト識別子が生成されます。XMLType表における更新や削除などの次のDMLのSQL_REDOには、WHERE句に元の表の行を一意に識別するためのオブジェクト識別子が含まれます。

親トピック: XMLType列およびXMLType表に基づいたV$LOGMNR_CONTENTSの問合せ

24.6.3.2 XMLTypeデータを使用するLogMinerの使用上の制限

XMLTypeデータを使用するLogMinerの使用上の制限を説明します。

XMLTypeデータのマイニングは、DBMS_LOGMNR.COMMITTED_DATA_ONLYオプションを使用している場合にのみ行う必

要があります。そうしないと、一部の行の変更が不足しているために、不完全な変更が表示されたり、XMLとして表示されるはずの変更がCLOBの変更として表示される可能性があります。このために、これらのSQL DML文に対するSQL_REDOが不完全かつ無効になってしまいます。

SQL_UNDO列は、XMLTypeデータへの変更に対して移入されません。

親トピック: XMLType列およびXMLType表に基づいたV$LOGMNR_CONTENTSの問合せ

24.6.3.3 XMLTypeデータを作成するためのPL/SQLプロシージャの例

表外のXMLデータを含む表に対してXML REDOをマイニングおよび作成するのに使用可能なプロシージャについて説明する例。

この例では、一時LOBを使用してXMLデータを作成する方法を示します。XML文書を一度作成すると、その文書を有効に使用できます。この例では、EmployeeName要素に対して作成された文書を問い合せて、EMPLOYEE_XML_DOCS表の元のDMLに対して返された名前、XML文書およびSQL_REDOを格納します。

> ノート:
>
> このプロシージャでは、簡単な例のみを示します。この例は、LogMiner を使用して、XMLType データを含む表に対する DML のマイニングおよび作成が可能であることの説明のみを目的としています。

このプロシージャをコールする前に、関連するすべてのログをLogMinerセッションに追加し、COMMITTED_DATA_ONLYオプションを使用して、DBMS_LOGMNR.START_LOGMNR()をコールする必要があります。その後、マイニング対象のXMLデータを含む表のスキーマ名と表名を使用して、MINE_AND_ASSEMBLE()プロシージャをコールすることができます。

```
-- table to store assembled XML documents
create table employee_xml_docs  (
  employee_name         varchar2(100),
  sql_stmt                     varchar2(4000),
  xml_doc                     SYS.XMLType);

-- procedure to assemble the XML documents
create or replace procedure mine_and_assemble(
  schemaname        in varchar2,
  tablename         in varchar2)
AS
  loc_c      CLOB;
  row_op     VARCHAR2(100);
  row_status NUMBER;
  stmt       VARCHAR2(4000);
  row_redo   VARCHAR2(4000);
  xml_data   VARCHAR2(32767 CHAR);
  data_len   NUMBER;
  xml_lob    clob;
  xml_doc    XMLType;
BEGIN

-- Look for the rows in V$LOGMNR_CONTENTS that are for the appropriate schema
-- and table name but limit it to those that are valid sql or that need assembly
-- because they are XML documents.

 For item in ( SELECT operation, status, sql_redo  FROM v$logmnr_contents
 where seg_owner = schemaname and table_name = tablename
 and status IN (DBMS_LOGMNR.VALID_SQL, DBMS_LOGMNR.ASSEMBLY_REQUIRED_SQL))
 LOOP
    row_op := item.operation;
    row_status := item.status;
```

```
    row_redo := item.sql_redo;

     CASE row_op

          WHEN 'XML DOC BEGIN' THEN
             BEGIN
               -- save statement and begin assembling XML data
               stmt := row_redo;
               xml_data := '';
               data_len := 0;
               DBMS_LOB.CreateTemporary(xml_lob, TRUE);
             END;

          WHEN 'XML DOC WRITE' THEN
             BEGIN
               -- Continue to assemble XML data
               xml_data := xml_data || row_redo;
               data_len := data_len + length(row_redo);
               DBMS_LOB.WriteAppend(xml_lob, length(row_redo), row_redo);
             END;

          WHEN 'XML DOC END' THEN
             BEGIN
               -- Now that assembly is complete, we can use the XML document
              xml_doc := XMLType.createXML(xml_lob);
              insert into employee_xml_docs values
                        (extractvalue(xml_doc, '/EMPLOYEE/NAME'), stmt, xml_doc);
              commit;

              -- reset
              xml_data := '';
              data_len := 0;
              xml_lob := NULL;
             END;

          WHEN 'INSERT' THEN
             BEGIN
                stmt := row_redo;
             END;

          WHEN 'UPDATE' THEN
             BEGIN
                stmt := row_redo;
             END;

          WHEN 'INTERNAL' THEN
             DBMS_OUTPUT.PUT_LINE('Skip rows marked INTERNAL');

          ELSE
             BEGIN
                stmt := row_redo;
                DBMS_OUTPUT.PUT_LINE('Other - ' || stmt);
                IF row_status != DBMS_LOGMNR.VALID_SQL then
                   DBMS_OUTPUT.PUT_LINE('Skip rows marked non-executable');
                ELSE
                   dbms_output.put_line('Status : ' || row_status);
                END IF;
             END;

     END CASE;

 End LOOP;

End;
/

show errors;
```

次に、このプロシージャをコールして、SCOTT.XML_DATA_TABへの変更をマイニングし、DMLを適用できます。

```
EXECUTE MINE_AND_ASSEMBLE ('SCOTT', 'XML_DATA_TAB');
```

このプロシージャの結果、EMPLOYEE_XML_DOCS表には、変更されたXMLの表外の列それぞれに対する行が含まれます。EMPLOYEE_NAME列には、XML文書から抽出された値が含まれ、SQL_STMT列とXML_DOC列には、元の行の変更が反映されます。

次に、従業員名とSQL文のみが表示される結果表への問合せの例を示します。

```
SELECT EMPLOYEE_NAME, SQL_STMT FROM EMPLOYEE_XML_DOCS;

EMPLOYEE_NAME          SQL_STMT

Scott Davis          update "SCOTT"."XML_DATA_TAB" a set a."F3" = XMLType(:1)
                         where a."F1" = '5000' and a."F2" = 'Chen' and a."F5" = 'JJJ'

Richard Harry        update "SCOTT"."XML_DATA_TAB" a set a."F4" = XMLType(:1)
                         where a."F1" = '5000' and a."F2" = 'Chen' and a."F5" = 'JJJ'

Margaret Sally       update "SCOTT"."XML_DATA_TAB" a set a."F4" = XMLType(:1)
                         where a."F1" = '5006' and a."F2" = 'Janosik' and a."F5" =
'MMM'
```

親トピック: XMLType列およびXMLType表に基づいたV$LOGMNR_CONTENTSの問合せ

24.7 V$LOGMNR_CONTENTSに返されるデータのフィルタ処理および書式設定

LogMinerは、大量の情報を処理できます。V$LOGMNR_CONTENTSビューに返される情報、および情報が返される速度を制限できます。

次の項では、これらの制限を指定する方法、およびV$LOGMNR_CONTENTSを問い合せた場合に返されるデータへのその制限の影響を示します。

また、LogMinerには、V$LOGMNR_CONTENTSに返されたデータを書式設定するための機能もあります。この機能については、次の項を参照してください。

- 再実行のために再構築されたSQL文の書式設定
- 返されるデータの可読性向上のための表示方法の書式設定

これらのフィルタ処理機能および書式設定機能は、DBMS_LOGMNR.START_LOGMNRプロシージャのパラメータまたはオプションを使用して要求します。

- コミット済トランザクションのみの表示
  DBMS_LOGMNR.START_LOGMNRに対してCOMMITTED_DATA_ONLYオプションを使用した場合、コミット済トランザクションに属する行のみがV$LOGMNR_CONTENTSビューに表示されます。
- REDO破損のスキップ
  DBMS_LOGMNR.START_LOGMNRに対してSKIP_CORRUPTIONオプションを使用した場合、V$LOGMNR_CONTENTSビューからの選択操作中、REDOログ・ファイル内の破損はすべてスキップされます。
- 時刻によるデータのフィルタ処理
  データを時刻でフィルタ処理するには、DBMS_LOGMNR.START_LOGMNRプロシージャにSTARTTIMEパラメータおよびENDTIMEパラメータを設定します。

- SCNによるデータのフィルタ処理
  データをSCN(システム変更番号)でフィルタ処理するには、PL/SQL DBMS_LOGMNR.START_LOGMNRプロシージャに対してSTARTSCNパラメータおよびENDSCNパラメータを使用します。
- 再実行のために再構築されたSQL文の書式設定
  デフォルトでは、再構築されるSQL_REDO文およびSQL_UNDO文にはROWID句が含まれ、それらの文はセミコロンで終わります。
- 返されるデータの可読性向上のための表示方法の書式設定
  LogMinerには、読みやすくするために返されたデータの外観を書式設定するPRINT_PRETTY_SQLオプションが用意されています。

親トピック: REDOログ・ファイル分析のためのLogMinerの使用

24.7.1 コミット済トランザクションのみの表示

DBMS_LOGMNR.START_LOGMNRに対してCOMMITTED_DATA_ONLYオプションを使用した場合、コミット済トランザクションに属する行のみがV$LOGMNR_CONTENTSビューに表示されます。

このオプションを使用すると、ロールバックされたトランザクション、進行中のトランザクションおよび内部操作をフィルタ処理して除外できます。

このオプションを有効にするには、LogMiner起動時に次のように指定します。

```
EXECUTE DBMS_LOGMNR.START_LOGMNR(OPTIONS => -
  DBMS_LOGMNR.COMMITTED_DATA_ONLY);
```

COMMITTED_DATA_ONLYオプションを指定した場合、LogMinerは、同じトランザクションに属するすべてのDML操作をグループ化します。トランザクションはコミットされた順序で返されます。

> ノート:
>
> COMMITTED_DATA_ONLY オプションを指定して問合せを発行した場合、LogMiner は、1 つのトランザクションのコミット・レコードを検出するまで、そのトランザクション内のすべての REDO レコードをメモリーにステージングします。したがって、メモリーを使い果たして、「メモリー不足」エラーが返される場合があります。このエラーが発生した場合は、COMMITTED_DATA_ONLY オプションを指定せずに LogMiner を再起動し、問合せを再発行する必要があります。

LogMinerのデフォルトでは、すべてのトランザクションに対応する行が表示され、その行がREDOログ・ファイルで検出された順序で返されます。

たとえば、COMMITTED_DATA_ONLYオプションを指定せずにLogMinerを起動し、次の問合せを実行したとします。

```
SELECT (XIDUSN || '.' || XIDSLT || '.' || XIDSQN) AS XID,
   USERNAME, SQL_REDO FROM V$LOGMNR_CONTENTS WHERE USERNAME != 'SYS'
   AND SEG_OWNER IS NULL OR SEG_OWNER NOT IN ('SYS', 'SYSTEM');
```

出力は次のようになります。コミット済トランザクションとコミットされていないトランザクションの両方が返され、異なるトランザクションからの行が混在しています。

```
XID          USERNAME  SQL_REDO
1.15.3045    RON       set transaction read write;
1.15.3045    RON       insert into "HR"."JOBS"("JOB_ID","JOB_TITLE",
                       "MIN_SALARY","MAX_SALARY") values ('9782',
```

```
                        'HR_ENTRY',NULL,NULL);
1.18.3046   JANE       set transaction read write;
1.18.3046   JANE       insert into "OE"."CUSTOMERS"("CUSTOMER_ID",
                        "CUST_FIRST_NAME","CUST_LAST_NAME",
                        "CUST_ADDRESS","PHONE_NUMBERS","NLS_LANGUAGE",
                        "NLS_TERRITORY","CREDIT_LIMIT","CUST_EMAIL",
                        "ACCOUNT_MGR_ID") values ('9839','Edgar',
                        'Cummings',NULL,NULL,NULL,NULL,
                         NULL,NULL,NULL);
1.9.3041    RAJIV       set transaction read write;
1.9.3041    RAJIV       insert into "OE"."CUSTOMERS"("CUSTOMER_ID",
                         "CUST_FIRST_NAME","CUST_LAST_NAME","CUST_ADDRESS",
                         "PHONE_NUMBERS","NLS_LANGUAGE","NLS_TERRITORY",
                         "CREDIT_LIMIT","CUST_EMAIL","ACCOUNT_MGR_ID")
                         values ('9499','Rodney','Emerson',NULL,NULL,NULL,NULL,
                         NULL,NULL,NULL);
1.15.3045    RON         commit;
1.8.3054     RON         set transaction read write;
1.8.3054     RON         insert into "HR"."JOBS"("JOB_ID","JOB_TITLE",
                         "MIN_SALARY","MAX_SALARY") values ('9566',
                         'FI_ENTRY',NULL,NULL);
1.18.3046    JANE        commit;
1.11.3047    JANE        set transaction read write;
1.11.3047    JANE        insert into "OE"."CUSTOMERS"("CUSTOMER_ID",
                         "CUST_FIRST_NAME","CUST_LAST_NAME",
                         "CUST_ADDRESS","PHONE_NUMBERS","NLS_LANGUAGE",
                         "NLS_TERRITORY","CREDIT_LIMIT","CUST_EMAIL",
                         "ACCOUNT_MGR_ID") values ('8933','Ronald',
                         'Frost',NULL,NULL,NULL,NULL,NULL,NULL,NULL);
1.11.3047    JANE        commit;
1.8.3054     RON         commit;
```

次に、COMMITTED_DATA_ONLYオプションを指定してLogMinerを起動したとします。前の例の問合せを再度実行すると、出力は次のようになります。

```
1.15.3045   RON        set transaction read write;
1.15.3045   RON        insert into "HR"."JOBS"("JOB_ID","JOB_TITLE",
                       "MIN_SALARY","MAX_SALARY") values ('9782',
                       'HR_ENTRY',NULL,NULL);
1.15.3045    RON        commit;
1.18.3046   JANE       set transaction read write;
1.18.3046   JANE       insert into "OE"."CUSTOMERS"("CUSTOMER_ID",
                       "CUST_FIRST_NAME","CUST_LAST_NAME",
                       "CUST_ADDRESS","PHONE_NUMBERS","NLS_LANGUAGE",
                       "NLS_TERRITORY","CREDIT_LIMIT","CUST_EMAIL",
                       "ACCOUNT_MGR_ID") values ('9839','Edgar',
                       'Cummings',NULL,NULL,NULL,NULL,
                        NULL,NULL,NULL);
1.18.3046    JANE       commit;
1.11.3047    JANE       set transaction read write;
1.11.3047    JANE       insert into "OE"."CUSTOMERS"("CUSTOMER_ID",
                        "CUST_FIRST_NAME","CUST_LAST_NAME",
                        "CUST_ADDRESS","PHONE_NUMBERS","NLS_LANGUAGE",
                        "NLS_TERRITORY","CREDIT_LIMIT","CUST_EMAIL",
                        "ACCOUNT_MGR_ID") values ('8933','Ronald',
                        'Frost',NULL,NULL,NULL,NULL,NULL,NULL,NULL);
1.11.3047    JANE       commit;
1.8.3054     RON        set transaction read write;
1.8.3054     RON        insert into "HR"."JOBS"("JOB_ID","JOB_TITLE",
                        "MIN_SALARY","MAX_SALARY") values ('9566',
                        'FI_ENTRY',NULL,NULL);
1.8.3054     RON        commit;
```

1.15.3045トランザクションのCOMMIT文は、1.18.3046トランザクションのCOMMIT文の前に発行されたため、1.15.3045トランザクション全体が先に返されます。これは、1.18.3046トランザクションが1.15.3045トランザクションの前に開始されてい

た場合でも同様です。1.9.3041トランザクションではCOMMIT文が発行されなかったため、このトランザクションは返されません。

関連項目:

COMMITTED_DATA_ONLYオプションの使用例の詳細は、「LogMinerの使用例」を参照してください。

親トピック: V$LOGMNR_CONTENTSに返されるデータのフィルタ処理および書式設定

24.7.2 REDO破損のスキップ

DBMS_LOGMNR.START_LOGMNRに対してSKIP_CORRUPTIONオプションを使用した場合、V$LOGMNR_CONTENTSビューからの選択操作中、REDOログ・ファイル内の破損はすべてスキップされます。

破損したREDOレコードが検出されるたびに、OPERATION列にCORRUPTED_BLOCKS、STATUS列に1343、INFO列にスキップしたブロック数の入った行が返されます。

スキップされたレコード内には、破損したブロックで進行中のトランザクションに対する変更が含まれる場合がありますが、そのような変更は、V$LOGMNR_CONTENTSビューから返されるデータには反映されません。

デフォルトでは、選択操作は、REDOログ・ファイルで最初に破損が検出された時点で終了します。

次のSQLの例で、このオプションの動作を示します。

```
-- Add redo log files of interest.
--
EXECUTE DBMS_LOGMNR.ADD_LOGFILE(-
   logfilename => '/usr/oracle/data/db1arch_1_16_482701534.log' -
   options => DBMS_LOGMNR.NEW);
-- Start LogMiner
--
EXECUTE DBMS_LOGMNR.START_LOGMNR();
-- Select from the V$LOGMNR_CONTENTS view. This example shows corruptions are -- in
the redo log files.
--
SELECT rbasqn, rbablk, rbabyte, operation, status, info
   FROM V$LOGMNR_CONTENTS;
ERROR at line 3:
ORA-00368: checksum error in redo log block
ORA-00353: log corruption near block 6 change 73528 time 11/06/2011 11:30:23
ORA-00334: archived log: /usr/oracle/data/dbarch1_16_482701534.log
-- Restart LogMiner. This time, specify the SKIP_CORRUPTION option.
--
EXECUTE DBMS_LOGMNR.START_LOGMNR(-
   options => DBMS_LOGMNR.SKIP_CORRUPTION);
-- Select from the V$LOGMNR_CONTENTS view again. The output indicates that
-- corrupted blocks were skipped: CORRUPTED_BLOCKS is in the OPERATION
-- column, 1343 is in the STATUS column, and the number of corrupt blocks
-- skipped is in the INFO column.
--
SELECT rbasqn, rbablk, rbabyte, operation, status, info
   FROM V$LOGMNR_CONTENTS;
RBASQN  RBABLK RBABYTE  OPERATION        STATUS  INFO
13      2        76     START              0
13      2        76     DELETE             0
13      3       100     INTERNAL           0
13      3       380     DELETE             0
13      0         0     CORRUPTED_BLOCKS   1343  corrupt blocks 4 to 19 skipped
13      20      116     UPDATE             0
```

親トピック: V$LOGMNR_CONTENTSに返されるデータのフィルタ処理および書式設定

24.7.3 時刻によるデータのフィルタ処理

データを時刻でフィルタ処理するには、DBMS_LOGMNR.START_LOGMNRプロシージャにSTARTTIMEパラメータおよびENDTIMEパラメータを設定します。

PL/SQL DBMS_LOGMNR.START_LOGMNRプロシージャへのコールで日付書式を指定する必要がないように、最初にSQL ALTER SESSION SET NLS_DATE_FORMAT文を使用します。次に例を示します。

```
ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';
EXECUTE DBMS_LOGMNR.START_LOGMNR( -
   DICTFILENAME => '/oracle/database/dictionary.ora', -
   STARTTIME => '01-Jan-2019 08:30:00', -
   ENDTIME => '01-Jan-2019 08:45:00'-
   );
```

タイムスタンプは、REDOレコードの順序の推測には使用しないでください。REDOレコードの順序は、SCNを使用して推測できます。

> ノート:
>
> フィルタリングする前にログ・ファイルを追加する必要があります。連続ロギングはサポートされなくなりました。指定した時間または SCN に一致するログ・ファイルが追加されていない場合、DBMS_LOGMNR.START_LOGMNR はエラー1291 ORA-01291: missing logfile で失敗します。

親トピック: V$LOGMNR_CONTENTSに返されるデータのフィルタ処理および書式設定

24.7.4 SCNによるデータのフィルタ処理

データをSCN(システム変更番号)でフィルタ処理するには、PL/SQL DBMS_LOGMNR.START_LOGMNRプロシージャに対してSTARTSCNパラメータおよびENDSCNパラメータを使用します。

たとえば:

```
 EXECUTE DBMS_LOGMNR.START_LOGMNR(-
    STARTSCN => 621047, -
    ENDSCN   => 625695, -
    OPTIONS  => DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG + -
                );
```

STARTSCNパラメータおよびENDSCNパラメータは、すべてのパラメータを指定した場合、STARTTIMEパラメータおよびENDTIMEパラメータより優先されます。

> ノート:
>
> フィルタリングする前にログ・ファイルを追加する必要があります。連続ロギングはサポートされなくなりました。指定した時間または SCN に一致するログ・ファイルが追加されていない場合、DBMS_LOGMNR.START_LOGMNR はエラー1291 ORA-01291: missing logfile で失敗します。

親トピック: V$LOGMNR_CONTENTSに返されるデータのフィルタ処理および書式設定

24.7.5 再実行のために再構築されたSQL文の書式設定

デフォルトでは、再構築されるSQL_REDO文およびSQL_UNDO文にはROWID句が含まれ、それらの文はセミコロンで終わります。

ただし、次のようにこのデフォルト設定を上書きできます。

- LogMiner起動時にNO_ROWID_IN_STMTオプションを指定します。

  この指定によって、再構築される文からROWID句が除外されます。行IDはデータベース間で一貫性がないため、最初に実行したデータベースと異なるデータベースに対してSQL_REDO文またはSQL_UNDO文を再実行する場合は、LogMiner起動時にNO_ROWID_IN_STMTオプションを指定します。

- LogMiner起動時にNO_SQL_DELIMITERオプションを指定します。

  この指定によって、再構築される文からセミコロンが削除されます。この指定は、カーソルでオープンしてから、再構築された文を実行するアプリケーションに有効です。

V$LOGMNR_CONTENTSビューSTATUSフィールドに値2(invalid sql)が含まれている場合、関連付けられたSQL文は実行できません。

親トピック: V$LOGMNR_CONTENTSに返されるデータのフィルタ処理および書式設定

24.7.6 返されるデータの可読性向上のための表示方法の書式設定

LogMinerには、読みやすくするために返されたデータの外観を書式設定するPRINT_PRETTY_SQLオプションが用意されています。

問合せの結果、再構築されたSQL文を含む列が多数返され、表示内容が繁雑になり読みにくくなる場合があります。LogMinerでは、この問題に対応するためにPRINT_PRETTY_SQLオプションが用意されています。DBMS_LOGMNR.START_LOGMNRプロシージャに対してPRINT_PRETTY_SQLオプションを指定すると、再構築されたSQL文が次のように書式設定され、読みやすくなります。

```
insert into "HR"."JOBS"
 values
    "JOB_ID" = '9782',
    "JOB_TITLE" = 'HR_ENTRY',
    "MIN_SALARY" IS NULL,
    "MAX_SALARY" IS NULL;
  update "HR"."JOBS"
  set
    "JOB_TITLE" = 'FI_ENTRY'
  where
    "JOB_TITLE" = 'HR_ENTRY' and
    ROWID = 'AAAHSeAABAAAY+CAAX';
update "HR"."JOBS"
  set
    "JOB_TITLE" = 'FI_ENTRY'
  where
    "JOB_TITLE" = 'HR_ENTRY' and
    ROWID = 'AAAHSeAABAAAY+CAAX';
delete from "HR"."JOBS"
 where
    "JOB_ID" = '9782' and
    "JOB_TITLE" = 'FI_ENTRY' and
    "MIN_SALARY" IS NULL and
    "MAX_SALARY" IS NULL and
    ROWID = 'AAAHSeAABAAAY+CAAX';
```

PRINT_PRETTY_SQLオプションが有効な状態で再構築されたSQL文は、標準SQL構文を使用していないため、実行され

ません。

関連項目:

PRINT_PRETTY_SQLオプションの使用例の詳細は、「LogMinerの使用例」を参照してください。

親トピック: V$LOGMNR_CONTENTSに返されるデータのフィルタ処理および書式設定

24.8 V$LOGMNR_CONTENTSに返されたDDL文の再適用

ユーザーが発行した一部のDDL文によって、1つ以上の他のDDL文がOracleによって内部的に実行される場合があります。

V$LOGMNR_CONTENTSビューのSQL_REDO列またはSQL_UNDO列から、データベースに対して最初に適用したときと同様にSQL DDLを再適用する場合は、Oracleによって内部的に実行された文は実行しないでください。

> ノート:
>
> Oracle によって内部的に実行された DML 文を実行すると、データベースが破損する場合があります。例については、「例 4: REDO ログ・ファイル内の LogMiner ディクショナリの使用」のステップ 5 を参照してください。

ユーザーによって発行されたDDL文と、Oracleによって内部的に発行されたDDL文を区別するには、V$LOGMNR_CONTENTSのINFO列を問い合せます。INFO列の値は、DDLがユーザーによって実行されたか、Oracleによって実行されたかを示します。

最初に適用したときと同様にSQL DDLを再適用する場合は、INFO列に値USER_DDLが含まれる場合にのみ、V$LOGMNR_CONTENTSのSQL_REDO列またはSQL_UNDO列に含まれるDDL SQLを再実行する必要があります。

親トピック: REDOログ・ファイル分析のためのLogMinerの使用

24.9 DBMS_LOGMNR.START_LOGMNRの複数回のコール

DBMS_LOGMNR.START_LOGMNRのコールに成功し、V$LOGMNR_CONTENTSビューから選択した後でも、現在のLogMinerセッションを終了せずにDBMS_LOGMNR.START_LOGMNRを再度コールし、異なるオプションおよび時間範囲またはSCN範囲を指定できます。

次の場合にコールを複数回行います。

- LogMinerが分析するREDOデータの量を制限する。
- 異なるオプションを指定する。たとえば、PRINT_PRETTY_SQLオプションを指定する場合、コミット済トランザクションのみを表示する場合(COMMITTED_DATA_ONLYオプションを指定)などがあります。
- 分析する時間範囲またはSCN範囲を変更する。

例: DBMS_LOGMNR.START_LOGMNRの複数回のコール

次に、DBMS_LOGMNR.START_LOGMNRを複数回コールすると有効な例を示します。

例1: REDOログ・ファイルのデータのサブセットのみのマイニング

LogMinerがマイニングするREDOログ・ファイルのリストに、1週間の間に生成されたログ・ファイルが含まれているとします。ただし、それぞれの日の12:00から1:00の間に生成されたログ・ファイルのみを分析します。これを最も効率的に行う方法は次のと

おりです。

1. 月曜日に、この時間範囲を指定して、DBMS_LOGMNR.START_LOGMNRをコールします。
2. V$LOGMNR_CONTENTSビューから変更を選択します。
3. 一週間のそれぞれの日に対して、ステップ1と2を繰り返します。

この方法では、その週のREDOデータの合計量が多い場合、リスト内の各REDOログ・ファイルの小さなサブセットのみがLogMinerによって読み込まれるため、全体の分析が非常に速く行われます。

例2: 時間範囲またはSCN範囲の調整

REDOログ・ファイル・リストを指定し、LogMiner起動時に時間範囲(またはSCN範囲)を指定するとします。V$LOGMNR_CONTENTSビューを問い合せると、指定した時間範囲には、必要なデータの一部のみが含まれます。時間範囲を1時間拡張するために、またはSCN範囲を調整するためにDBMS_LOGMNR.START_LOGMNRを再度コールできます。

例3: リモート・データベースで受信された場合のREDOログ・ファイルの分析

変更を分析またはデータベース間で変更をレプリケートするためのアプリケーションを作成したとします。ソース・データベースは、そのREDOログ・ファイルをマイニング・データベースに送信し、オペレーティング・システム・ディレクトリに格納します。アプリケーションは、次の手順を実行します。

1. 現在ディレクトリにあるすべてのREDOログ・ファイルをREDOログ・ファイル・リストに追加します。
2. 適切な設定でDBMS_LOGMNR.START_LOGMNRをコールし、V$LOGMNR_CONTENTSビューから選択します。
3. ディレクトリに新しく受信されたREDOログ・ファイルを追加します。
4. 必要に応じて、ステップ2と3を繰り返します。

親トピック: REDOログ・ファイル分析のためのLogMinerの使用

24.10 サプリメンタル・ロギング

サプリメンタル・ロギングについて説明します。

一般に、REDOログ・ファイルは、インスタンス・リカバリおよびメディア・リカバリに使用されます。これらの操作に必要なデータは、REDOログ・ファイルに自動的に記録されます。ただし、REDOベースのアプリケーションでは、追加の列をREDOログ・ファイルに記録する必要がある場合があります。これらの追加の列を記録するプロセスは、サプリメンタル・ロギングと呼ばれます。

デフォルトでは、Oracle Databaseでサプリメンタル・ロギングは提供されません。つまり、デフォルトでは、LogMinerは使用できません。したがって、LogMinerで分析するログ・ファイルを生成する前に、少なくとも最小サプリメンタル・ロギングは有効にする必要があります。

次に、追加の列が必要な例を示します。

- 再構築されたSQL文を別のデータベースに適用するアプリケーションでは、行を一意に識別する列(主キーなど)で更新文を識別する必要があります。ROWIDはデータベースごとに異なり、他のデータベースでは意味を持たないため、V$LOGMNR_CONTENTSビューによって返される再構築されたSQLに示されるROWIDでは識別できません。
- アプリケーションは、行変更の追跡をより効率的にするために、変更された列のみでなく、行全体のビフォア・イメージを記録する必要がある場合があります。

サプリメンタル・ログ・グループは、サプリメンタル・ロギングが有効な場合に記録される追加の列です。サプリメンタル・ログ・グループは2種類あり、これらによって、ログ・グループの列を記録する時期が決定されます。

- 無条件のサプリメンタル・ログ・グループ: 指定した列に更新が影響したかどうかに関係なく、行が更新されるたびに指定した列のビフォア・イメージが記録されます。これは、ALWAYSログ・グループと呼ばれる場合もあります。
- 条件付きのサプリメンタル・ログ・グループ: ログ・グループの1つ以上の列が更新された場合にのみ、指定したすべての列のビフォア・イメージが記録されます。

サプリメンタル・ログ・グループは、システムで生成することも、ユーザーが定義することもできます。

次の項で説明するとおり、サプリメンタル・ロギングには、2つの種類に加えて2つのレベルがあります。

- データベース・レベルのサプリメンタル・ロギング
  LogMinerには、これらの項で説明するとおり、最小サプリメンタル・ロギング、識別キー・ロギングおよびプロシージャ型サプリメンタル・ロギングの異なるタイプのデータベース・レベルのサプリメンタル・ロギングがあります。
- データベース・レベルのサプリメンタル・ロギングの無効化
  データベース・レベルのサプリメンタル・ロギングを無効にするには、`DROP SUPPLEMENTAL LOGGING`句を指定してSQL `ALTER DATABASE`文を使用します。
- 表レベルのサプリメンタル・ロギング
  表レベルのサプリメンタル・ロギングでは、補助的に記録する列を表レベルで指定します。
- LogMinerディクショナリでのDDL文の追跡
  LogMinerでは、LogMiner起動時に指定したLogMinerディクショナリ(オンライン・カタログ、REDOログ・ファイルのディクショナリまたはフラット・ファイルのいずれか)から独自の内部ディクショナリが自動的に作成されます。
- DDL_DICT_TRACKINGおよびサプリメンタル・ロギングの設定
  ディクショナリ追跡とサプリメンタル・ロギングの各種設定を組み合せた場合に発生する相互作用を説明します。
- DDL_DICT_TRACKINGおよび指定された時間範囲またはSCN範囲
  LogMinerは、LogMinerディクショナリの一貫性を維持するためにDDL文が必要なため、`DDL_DICT_TRACKING`オプションが有効な場合、要求した開始時刻またはSCN(`DBMS_LOGMNR.START_LOGMNR`で指定)より前に、REDOログ・ファイルの読取りを開始する場合があります。

関連項目:

サプリメンタル・ロギング設定に関するビューの問合せ

親トピック: REDOログ・ファイル分析のためのLogMinerの使用

24.10.1 データベース・レベルのサプリメンタル・ロギング

LogMinerには、これらの項で説明するとおり、最小サプリメンタル・ロギング、識別キー・ロギングおよびプロシージャ型サプリメンタル・ロギングの異なるタイプのデータベース・レベルのサプリメンタル・ロギングがあります。

最小サプリメンタル・ロギングでは、REDOログ・ファイルを生成するデータベースに大きなオーバーヘッドが発生しません。ただし、データベース全体の識別キー・ロギングを有効にすると、REDOログ・ファイルを生成するデータベースにオーバーヘッドが発生する場合があります。LogMinerに対しては、少なくとも最小サプリメンタル・ロギングを有効にすることをお薦めします。

- 最小サプリメンタル・ロギング
  最小サプリメンタル・ロギングは、LogMinerでDML変更と関連付けられたREDO操作を識別、グループ化およびマージするために必要な最小限の情報を記録します。
- データベース・レベルの識別キー・ロギング
  識別キー・ロギングは、REDOログ・ファイルがソース・データベース・インスタンスでマイニングされない場合(REDOログ・ファイルがロジカル・スタンバイ・データベースでマイニングされる場合など)に必要です。

- プロシージャ型サプリメンタル・ロギング
  プロシージャ型サプリメンタル・ロギングを有効にすると、ローリング・アップグレードまたはOracle GoldenGateでレプリケートできるように、LogMinerによって特定のプロシージャ起動がREDOに記録されます。

親トピック: サプリメンタル・ロギング

24.10.1.1 最小サプリメンタル・ロギング

最小サプリメンタル・ロギングは、LogMinerでDML変更と関連付けられたREDO操作を識別、グループ化およびマージするために必要な最小限の情報を記録します。

また、LogMiner(およびLogMinerテクノロジに基づいた他の製品)に、連鎖行や様々な記憶域構成(クラスタ表、索引構成表など)のサポートに十分な情報を確保します。最小サプリメンタル・ロギングを有効にするには、次のSQL文を実行します。

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;
```

親トピック: データベース・レベルのサプリメンタル・ロギング

24.10.1.2 データベース・レベルの識別キー・ロギング

識別キー・ロギングは、REDOログ・ファイルがソース・データベース・インスタンスでマイニングされない場合(REDOログ・ファイルがロジカル・スタンバイ・データベースでマイニングされる場合など)に必要です。

データベース識別キー・ロギングを使用すると、次のオプションの1つ以上をSQLのALTER DATABASE ADD SUPPLEMENTAL LOG文に指定して、すべての更新に対してデータベース全体のビフォア・イメージ・ロギングを有効にできます。

- ALL: システムによって生成される無条件のサプリメンタル・ログ・グループ

  このオプションを指定すると、行が更新された場合、その行のすべての列(LOB、LONGS、ADTを除く)がREDOログ・ファイルに格納されます。

  データベース・レベルですべての列ロギングを有効にするには、次の文を実行します。

  ```
  SQL> ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
  ```

- PRIMARY KEY: システムによって生成される無条件のサプリメンタル・ログ・グループ

  このオプションを指定すると、主キーを含む行が更新された場合(主キーの値に変更がない場合も)、データベースによって行の主キーのすべての列がREDOログ・ファイルに格納されます。

  表に主キーが存在せず、1つ以上の非NULLの一意索引キー制約または索引キーが存在する場合は、一意索引キーのいずれかが、更新された行を一意に識別する手段としてロギング用に選択されます。

  表に主キーと非NULLの一意索引キーのいずれも存在しない場合は、LONGとLOBを除くすべての列が補助的に記録されます。これは、その行に対してALLサプリメンタル・ロギングを指定することと同様です。したがって、データベース・レベルの主キー・サプリメンタル・ロギングを使用する場合は、すべての表またはほとんどの表に主キーまたは一意索引キーが含まれるように定義しておくことをお薦めします。

  データベース・レベルで主キー・ロギングを有効にするには、次の文を実行します。

  ```
  SQL> ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
  ```

- UNIQUEシステムによって生成される条件付きのサプリメンタル・ログ・グループ

  このオプションを指定すると、コンポジット一意キーまたはビットマップ索引に属する列が変更された場合、データベースによって、行のコンポジット一意キーまたはビットマップ索引のすべての列がREDOログ・ファイルに格納されます。一意キー

は、一意制約または一意索引のいずれかが原因である可能性があります。

データベース・レベルで一意索引キーおよびビットマップ索引のロギングを有効にするには、次の文を実行します。

```
SQL> ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (UNIQUE) COLUMNS;
```

- FOREIGN KEY: システムによって生成される条件付きのサプリメンタル・ログ・グループ

  このオプションを指定すると、データベースによって、外部キーに属する列が変更された場合、行の外部キーのすべての列がREDOログ・ファイルに格納されます。

  データベース・レベルで外部キー・ロギングを有効にするには、次のSQL文を実行します。

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (FOREIGN KEY) COLUMNS;
```

> ノート:
>
> 識別キー・ロギングが有効であるかどうかに関係なく、常に、LogMiner によって返される SQL 文には ROWID 句が含まれます。DBMS_LOGMNR.START_LOGMNR プロシージャ・コールに対して NO_ROWID_IN_STMT オプションを使用すると、フィルタ処理で ROWID 句を除外できます。詳細は、「再実行のために再構築された SQL 文の書式設定」を参照してください。

識別キー・ロギングを使用する場合は、次のことに注意してください。

- 識別キー・ロギングが有効な場合にデータベースがオープンしていると、カーソル・キャッシュ内のすべてのDMLカーソルが無効になります。したがって、カーソル・キャッシュに再移入を行うまで、パフォーマンスに影響する場合があります。
- データベース・レベルで識別キー・ロギングを有効にすると、最小サプリメンタル・ロギングが暗黙で有効になります。
- 主キーにENABLE NOVALIDATEを指定した場合、主キーは有効な識別キーとはみなされません。有効な一意制約がない場合、すべてのスカラー列が記録されます。行外の列(LOB、XML、32k varcharなど)が補足的にログに記録されることはありません。
- サプリメンタル・ロギング文は累積的に実行されます。次のSQL文を発行すると、主キー・サプリメンタル・ロギングと一意キー・サプリメンタル・ロギングの両方が有効になります。

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (UNIQUE) COLUMNS;
```

親トピック: データベース・レベルのサプリメンタル・ロギング

24.10.1.3 プロシージャ型サプリメンタル・ロギング

プロシージャ型サプリメンタル・ロギングを有効にすると、ローリング・アップグレードまたはOracle GoldenGateでレプリケートできるように、LogMinerによって特定のプロシージャ起動がREDOに記録されます。

プロシージャ型サプリメンタル・ロギングは、AQキュー表、階層対応表およびSDO_TOPO_GEOMETRY列またはSDO_GEORASTER列がある表のレプリケーションをサポートするために、ローリング・アップグレードおよびOracle GoldenGateに対して有効にする必要があります。プロシージャ型サプリメンタル・ロギングを有効にするには、次のSQL文を使用します。

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA FOR PROCEDURAL REPLICATION END SUBHEADING
```

プロシージャ型サプリメンタル・ロギングが有効になっている場合、プロシージャ型サプリメンタル・ロギングを先に削除しないかぎり、最小サプリメンタル・ロギングを削除できません。

親トピック: データベース・レベルのサプリメンタル・ロギング

24.10.2 データベース・レベルのサプリメンタル・ロギングの無効化

データベース・レベルのサプリメンタル・ロギングを無効にするには、DROP SUPPLEMENTAL LOGGING句を指定してSQL ALTER DATABASE文を使用します。

サプリメンタル・ロギング属性は段階的に削除できます。たとえば、次のSQL文を次の順序で発行したとします。

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (UNIQUE) COLUMNS;
ALTER DATABASE DROP SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
ALTER DATABASE DROP SUPPLEMENTAL LOG DATA;
```

この文には、次のような効果があります。

- 最初の文の後、主キー・サプリメンタル・ロギングが有効になります。
- 2番目の文の後、主キー・サプリメンタル・ロギングおよび一意キー・サプリメンタル・ロギングが有効になります。
- 3番目の文の後、一意キー・サプリメンタル・ロギングのみが有効になります。
- 4番目の文の後、すべてのサプリメンタル・ロギングが無効になりません。次に示すエラーが返されます。ORA-32589: 最小限のサプリメンタル・ロギングは削除できません

すべてのデータベース・サプリメンタル・ロギングを無効にするには、最初に、有効になっている識別キー・ロギングをすべて無効にしてから、最小サプリメンタル・ロギングを無効にする必要があります。次の例に、正しい順序を示します。

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (UNIQUE) COLUMNS;
ALTER DATABASE DROP SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
ALTER DATABASE DROP SUPPLEMENTAL LOG DATA (UNIQUE) COLUMNS;
ALTER DATABASE DROP SUPPLEMENTAL LOG DATA;
```

最小サプリメンタル・ログ・データは、データベース・レベルの他のサプリメンタル・ロギングが有効になっていない場合にのみ削除できます。

親トピック: サプリメンタル・ロギング

24.10.3 表レベルのサプリメンタル・ロギング

表レベルのサプリメンタル・ロギングでは、補助的に記録する列を表レベルで指定します。

識別キー・ロギングまたはユーザー定義の条件付きのサプリメンタル・ログ・グループまたは無条件のサプリメンタル・ログ・グループを使用して、次の項で説明するとおり、補助情報を記録できます。

- 表レベルの識別キー・ロギング
  表レベルでの識別キー・ロギングでは、データベース・レベルの場合と同じオプション(すべて、主キー、外部キー、一意キー)が提供されます。
- 表レベルのユーザー定義サプリメンタル・ログ・グループ
  表レベルの識別キー・ロギングに加えて、ユーザー定義のサプリメンタル・ログ・グループもサポートされています。
- ユーザー定義のサプリメンタル・ログ・グループを使用する場合のノート
  ユーザー定義のサプリメンタル・ログ・グループを使用するためのヒント。

親トピック: サプリメンタル・ロギング

24.10.3.1 表レベルの識別キー・ロギング

表レベルでの識別キー・ロギングでは、データベース・レベルの場合と同じオプション(すべて、主キー、外部キー、一意キー)が提供されます。

ただし、表レベルで識別キー・ロギングを指定すると、指定した表のみが影響を受けます。たとえば、次のSQL文(データベース・レベルのサプリメンタル・ロギングの指定)を入力すると、データベース表の列が変更された場合は常に、その列を含む行全体(LOB、LONG、ADTの列を除く)がREDOログ・ファイルに格納されます。

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
```

ただし、次のSQL文(表レベルのサプリメンタル・ロギングの指定)を入力すると、employees表の列が変更された場合にのみ、行全体(LOB、LONG、ADTを除く)がREDOログ・ファイルに格納されます。departments表の列が変更されると、変更された列のみがREDOログ・ファイルに格納されます。

```
ALTER TABLE HR.EMPLOYEES ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
```

表レベルの識別キー・ロギングを使用する場合は、次のことに注意してください。

- 表で識別キー・ロギングが有効な場合にデータベースがオープンしていると、カーソル・キャッシュ内のその表のすべてのDMLカーソルが無効になります。したがって、カーソル・キャッシュに再移入を行うまで、パフォーマンスに影響する場合があります。
- サプリメンタル・ロギング文は累積的に実行されます。次のSQL文を発行すると、主キー・サプリメンタル・ロギングと一意索引キーの表レベルのサプリメンタル・ロギングの両方が有効になります。

  ```
  ALTER TABLE HR.EMPLOYEES
    ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
  ALTER TABLE HR.EMPLOYEES
    ADD SUPPLEMENTAL LOG DATA (UNIQUE) COLUMNS;
  ```

各識別キー・ロギング・オプションの詳細は、「データベース・レベルの識別キー・ロギング」を参照してください。

親トピック: 表レベルのサプリメンタル・ロギング

24.10.3.2 表レベルのユーザー定義サプリメンタル・ログ・グループ

表レベルの識別キー・ロギングによって、ユーザー定義のサプリメンタル・ログ・グループもサポートされています。

ユーザー定義サプリメンタル・ログ・グループによって、補助的に記録する列を指定できます。条件付きログ・グループまたは無条件ログ・グループは、次のとおり指定できます。

- ユーザー定義の無条件ログ・グループ

  ユーザー定義の無条件ログ・グループを使用するサプリメンタル・ロギングを有効にするには、次の例に示すとおり、ALWAYS句を使用します。

  ```
  ALTER TABLE HR.EMPLOYEES
     ADD SUPPLEMENTAL LOG GROUP emp_parttime (EMPLOYEE_ID, LAST_NAME,
     DEPARTMENT_ID) ALWAYS;
  ```

  この例では、hr.employees表でemp_parttimeという名前のログ・グループが作成されます。このグループは、employee_id列、last_name列、department_id列で構成されます。これらの列は、UPDATE文がhr.employees表で実行されるたびに、その更新がこれらの列に影響するかどうかに関係なく記録されます。(更新が行われるたびに行イメージ全体を記録する場合は、前述のとおり、表レベルのALL識別キー・ロギングを使用します。)

ノート:

LOB 列、LONG 列および ADT 列は補助的に記録できません。

- ユーザー定義の条件付きログ・グループ

  ユーザー定義の条件付きログ・グループを使用するサプリメンタル・ロギングを有効にするには、次の例に示すとおり、SQL ALTER TABLE文からALWAYS句を削除します。

```
ALTER TABLE HR.EMPLOYEES
   ADD SUPPLEMENTAL LOG GROUP emp_fulltime (EMPLOYEE_ID, LAST_NAME,
   DEPARTMENT_ID);
```

  この例では、表hr.employeesでemp_fulltimeという名前のログ・グループが作成されます。前述の例と同様に、このグループは、employee_id列、last_name列およびdepartment_id列で構成されます。ただし、ALWAYS句が省略されたため、列のビフォア・イメージは、1つ以上の列が更新された場合にのみ記録されます。

無条件および条件付きのいずれのユーザー定義サプリメンタル・ログ・グループでも、NO LOGオプションを指定して、ログ・グループ内の列をサプリメンタル・ロギングから除外することを明示的に指定できます。ログ・グループを指定し、NO LOGオプションを使用する場合は、次の例に示すとおり、NO LOGオプションが接続されていない列をログ・グループで1つ以上指定する必要があります。

```
ALTER TABLE HR.EMPLOYEES
   ADD SUPPLEMENTAL LOG GROUP emp_parttime(
   DEPARTMENT_ID NO LOG, EMPLOYEE_ID);
```

これによって、NO LOG列に変更を行うと、サプリメンタル・ログ・グループ内の他の列がREDOログ・ファイルに格納されるように、指定されたサプリメンタル・ログ・グループ内の他の列とこの列とを関連付けることができます。この方法は、たとえば、LONG列の変更時にグループの特定の列を記録する場合に有効です。LONG列自体は補助的に記録できませんが、この列に対して行った変更を使用して、同じ行の他の列のサプリメンタル・ロギングをトリガーすることはできます。

親トピック: 表レベルのサプリメンタル・ロギング

24.10.3.3 ユーザー定義のサプリメンタル・ログ・グループを使用する場合のノート

ユーザー定義のサプリメンタル・ログ・グループを使用するためのヒント。

ユーザー定義のサプリメンタル・ログ・グループを指定する場合は、次の点に注意してください。

- 1つの列を複数のサプリメンタル・ログ・グループに含めることができます。ただし、その列のビフォア・イメージの記録は1回のみです。
- 条件付きと無条件の両方で同じ列を記録することを指定した場合、その列は無条件に記録されます。

親トピック: 表レベルのサプリメンタル・ロギング

24.10.4 LogMinerディクショナリでのDDL文の追跡

LogMinerでは、LogMiner起動時に指定したLogMinerディクショナリ(オンライン・カタログ、REDOログ・ファイルのディクショナリまたはフラット・ファイルのいずれか)から独自の内部ディクショナリが自動的に作成されます。

このディクショナリでは、データベース・オブジェクトとその定義のスナップショットが提供されます。

LogMinerディクショナリがREDOログ・ファイルにある場合またはフラット・ファイルである場合は、PL/SQL

DBMS_LOGMNR.START_LOGMNRプロシージャに対してDDL_DICT_TRACKINGオプションを使用して、LogMinerでデータ定義言語(DDL)文を追跡することができます。DDL追跡によって、LogMinerは、表での列の追加や削除など、データベース・オブジェクトに対して行われた構造上の変更を問題なく追跡できます。たとえば:

```
EXECUTE DBMS_LOGMNR.START_LOGMNR(OPTIONS => -
   DBMS_LOGMNR.DDL_DICT_TRACKING + DBMS_LOGMNR.DICT_FROM_REDO_LOGS);
```

例の詳細は、「例5: 内部ディクショナリでのDDL文の追跡」を参照してください。

このオプションが設定されている場合、LogMinerは、REDOログ・ファイルで検出されたDDL文をその内部ディクショナリに適用します。

> ノート:
>
> サプリメンタル・ロギングおよび DDL 追跡機能が有効でない場合に DDL イベントが発生すると、LogMiner によって REDO データの一部がバイナリ・データとして返されるため、通常、これらの機能を有効にしておくことをお薦めします。また、メタデータ・バージョンの不一致が発生する可能性もあります。

DDL_DICT_TRACKINGを有効にすると、LogMinerディクショナリの抽出後に作成された表で実行されたデータ操作言語(DML)操作を正常に表示できます。

たとえば、employees表が、1回目の操作でgender列が追加され、次の操作でcommission_pct列が削除されるように、連続する2つのDDL操作で更新された場合、LogMinerによって、これらの変更のそれぞれに対するemployeesについての情報がバージョン化されます。つまり、これらのDDLの変更前と変更後のREDOログ・ファイルは、LogMinerによって正常にマイニングできるため、SQL_REDO列またはSQL_UNDO列にバイナリ・データは表示されません。

LogMinerは、データベース・メタデータに対して自動的にバージョンを割り当てるため、内部ディクショナリとREDOログ・ファイル内のディクショナリとの間で不一致を検出して通知します。LogMinerによって不一致が検出されると、V$LOGMNR_CONTENTSビューのSQL_REDO列でバイナリ・データが生成され、INFO列に「ディクショナリのバージョンの不一致」という文字列、STATUS列に値2が入力されます。

> ノート:
>
> LogMiner 内部ディクショナリが、フラット・ファイル、REDO ログ・ファイルまたはオンライン・カタログに含まれている LogMiner ディクショナリとは異なることを理解しておいてください。LogMiner は、LogMiner 内部ディクショナリは更新しますが、フラット・ファイル、REDO ログ・ファイルまたはオンライン・カタログに含まれているディクショナリは更新しません。

次に、DBMS_LOGMNR.START_LOGMNRプロシージャでDDL_DICT_TRACKINGオプションを指定するための要件を説明します。

- DDL_DICT_TRACKINGオプションは、DICT_FROM_ONLINE_CATALOGオプションとは併用できません。
- DDL_DICT_TRACKINGオプションでは、データベースがオープンしている必要があります。
- サプリメンタル・ロギングをデータベース全体で有効にするか、または対象となる表のログ・グループを作成しておく必要があります。

親トピック: サプリメンタル・ロギング

24.10.5 DDL_DICT_TRACKINGおよびサプリメンタル・ロギングの設定

ディクショナリ追跡とサプリメンタル・ロギングの各種設定を組み合せた場合に発生する相互作用を説明します。

次の点に注意してください。

- DDL_DICT_TRACKINGが有効で、サプリメンタル・ロギングが有効でない場合は、次のようになります。
  - REDOログ・ファイルでDDLトランザクションが検出された場合、V$LOGMNR_CONTENTSの問合せはORA-01347エラーを返して終了します。
  - REDOログ・ファイルでDMLトランザクションが検出された場合、LogMinerディクショナリ内の表(そのDMLの基礎となる表)の現在のバージョンがLogMinerによって正しいとは認識されず、V$LOGMNR_CONTENTSの列が次のように設定されます。
    - SQL_REDO列にバイナリ・データが含まれます。
    - STATUS列に値2が含まれます(SQLが有効でないことを示します)。
    - INFO列に「ディクショナリの不一致」という文字列が含まれます。
- DDL_DICT_TRACKINGおよびサプリメンタル・ロギングが有効ではない場合、DML操作で参照された列がLogMinerディクショナリ内の列と一致すると、LogMinerディクショナリの最新バージョンがLogMinerによって正しいと認識され、V$LOGMNR_CONTENTSの列が次のように設定されます。
  - LogMinerは、LogMinerディクショナリ内のオブジェクトの定義を使用して、SQL_REDO列とSQL_UNDO列の値を生成します。
  - STATUS列に値3が含まれます(SQLが正しいと保証できないことを示します)。
  - INFO列には、「サプリメンタル・ログ・データが見つかりません」という文字列が含まれます。
- DDL_DICT_TRACKINGおよびサプリメンタル・ロギングが有効ではない場合に、表のLogMinerディクショナリ定義で定義された列より、表のREDOログ・ファイル内の変更された列の方が多いときは、次のようになります。
  - SQL_REDO列とSQL_UNDO列に「ディクショナリのバージョンの不一致」という文字列が含まれます。
  - STATUS列に値2が含まれます(SQLが有効でないことを示します)。
  - INFO列に「ディクショナリの不一致」という文字列が含まれます。

  また、列のディクショナリ定義で定義されている型とその列の実際の型が異なる場合は、予期しない動作が発生する可能性があります。

親トピック: サプリメンタル・ロギング

24.10.6 DDL_DICT_TRACKINGおよび指定された時間範囲またはSCN範囲

LogMinerは、LogMinerディクショナリの一貫性を維持するためにDDL文が必要なため、DDL_DICT_TRACKINGオプションが有効な場合、要求した開始時刻またはSCN(DBMS_LOGMNR.START_LOGMNRで指定)より前に、REDOログ・ファイルの読取りを開始する場合があります。

LogMinerがREDOログ・ファイルの読取りを開始する実際の時刻またはSCNは、必須開始時刻または必須開始SCNと呼ばれます。

欠落REDOログ・ファイル(順序番号に基づく)は、必須開始時刻または必須開始SCNから読取りを開始できません。

LogMinerでは、REDOログ・データの読取りを開始する位置を次のとおり決定します。

- ディクショナリのロード後、DBMS_LOGMNR.START_LOGMNRを初めてコールする場合、LogMinerは、次のいずれかの値の早い方を基準に読取りを開始します。
  - ユーザーが要求した開始時刻またはSCN値
  - ディレクトリ・ダンプのコミットSCN
- DBMS_LOGMNR.START_LOGMNRに対するこれ以降のコールでは、LogMinerは、次のいずれかの値の最も早いものを基準に読取りを開始します。
  - ユーザーが要求した開始時刻またはSCN値
  - COMMIT文がLogMinerによって読み込まれていない最も早いDDLトランザクションの開始時
  - LogMinerによって読み込まれた最高のSCN

次に、前述の内容の使用例を示します。

5つのREDOログ・ファイルを含むREDOログ・ファイル・リストを作成するとします。ディクショナリは、最初のREDOファイルに含まれ、確認対象として指定した変更(DBMS_LOGMNR.START_LOGMNRを使用)は、3番目のREDOログ・ファイルに記録されているとします。ここで、次の手順を実行します。

1. DBMS_LOGMNR.START_LOGMNRをコールします。LogMinerによって、次のものが読み込まれます。
   a. ディクショナリをロードするための最初のログ・ファイル
   b. その中に含まれている可能性があるDDLをすべて取得するための2番目のREDOログ・ファイル
   c. 必要なデータを取得するための3番目のログ・ファイル
2. 同じ要求範囲で、再度DBMS_LOGMNR.START_LOGMNRをコールします。

   LogMinerは、REDOログ・ファイル3から読取りを開始します。REDOログ・ファイル2に含まれているDDL文はすでに処理されているため、REDOログ・ファイル2を読み込む必要はありません。
3. DBMS_LOGMNR.START_LOGMNRを再度コールします。今回は、REDOログ・ファイル5からデータを読み込む必要があるパラメータを指定します。

   LogMinerは、REDOログ・ファイル4から読取りを開始し、その中に含まれているすべてのDDL文を取得します。

LogMinerが実際に読取りを開始する位置を確認するには、V$LOGMNR_PARAMETERSビューのREQUIRED_START_DATE列またはREQUIRED_START_SCN列を問い合せます。LogMinerが読取りを開始する位置に関係なく、要求した範囲内の行のみがV$LOGMNR_CONTENTSビューから返されます。

親トピック: サプリメンタル・ロギング

24.11 ビューでのLogMiner操作情報へのアクセス

LogMiner操作情報(REDOデータではなく)は、ビューに含まれています。

他のビューと同様に、SQLを使用してこれらのビューを問い合せることができます。

- V$LOGMNR_DICTIONARY

  DBMS_LOGMNR.START_LOGMNRに対してSTORE_IN_FLAT_FILEオプションを使用して作成されたLogMinerディクショナリ・ファイルに関する情報を表示します。LogMinerディクショナリ作成の基礎となったデータベースに関する

情報などが表示されます。

- V$LOGMNR_LOGS

  指定したREDOログ・ファイルに関する情報を表示します。詳細は、「V$LOGMNR_LOGSの問合せ」を参照してください。

- V$LOGMNR_PARAMETERS

  オプションのLogMinerパラメータに関する情報を表示します。開始システム変更番号(SCN)と終了SCN、開始時刻と終了時刻などの情報が含まれます。

- V$DATABASE、DBA_LOG_GROUPS、ALL_LOG_GROUPS、USER_LOG_GROUPS、DBA_LOG_GROUP_COLUMNS、ALL_LOG_GROUP_COLUMNS、USER_LOG_GROUP_COLUMNS

  サプリメンタル・ロギングの現在の設定に関する情報を表示します。詳細は、「サプリメンタル・ロギング設定に関するビューの問合せ」を参照してください。

- LogMiner操作情報を表示するオプション
  LogMiner操作を確認するには、他のビューと同様に、SQLを使用してLogMinerビューを問い合せられます。
- V$LOGMNR_LOGSの問合せ
  LogMinerで分析するREDOログ・ファイルのリストに手動または自動で追加されたREDOログ・ファイルを特定するには、V$LOGMNR_LOGSビューを問い合せます。
- サプリメンタル・ロギング設定のビューの問合せ
  サプリメンタル・ロギングの現在の設定を確認するには、複数の異なるビューを問い合せられます。
- LogMinerを使用した個々のPDBの問合せ
  時間別またはSCN別にディクショナリ・ビルドを検索するには(たとえば、PDBごとのマイニングを開始する場合)、ソース・データベースでSYS.DBA_LOGMNR_DICTIONARY_BUILDLOGビューを使用できます。

親トピック: REDOログ・ファイル分析のためのLogMinerの使用

24.11.1 LogMiner操作情報を表示するためのオプション

LogMiner操作を確認するには、他のビューと同様に、SQLを使用してLogMinerビューを問い合せられます。

V$LOGMNR_CONTENTSに加えて、他のビューとその表示内容のリストを次に示します。

- V$LOGMNR_DICTIONARY

  DBMS_LOGMNR.START_LOGMNRに対してSTORE_IN_FLAT_FILEオプションを使用して作成されたLogMinerディクショナリ・ファイルに関する情報を表示します。LogMinerディクショナリ作成の基礎となったデータベースに関する情報などが表示されます。

- V$LOGMNR_LOGS

  指定したREDOログ・ファイルに関する情報を表示します。

- V$LOGMNR_PARAMETERS

  オプションのLogMinerパラメータに関する情報を表示します。開始システム変更番号(SCN)と終了SCN、開始時刻と終了時刻などの情報が含まれます。

- V$DATABASE, DBA_LOG_GROUPS, ALL_LOG_GROUPS, USER_LOG_GROUPS, DBA_LOG_GROUP_COLUMNS, ALL_LOG_GROUP_COLUMNS, USER_LOG_GROUP_COLUMN

  サプリメンタル・ロギングの現在の設定に関する情報を表示します。

- SYS.DBA_LOGMNR_DICTIONARY_BUILDLOG

  ディクショナリ・ビルドを時間またはSCNのいずれかで検索します。このビューは、CDB$ROOTログ・マイニングとPDBごとのログ・マイニングの両方で、Oracle Database 19c (リリース更新10以降)で使用できます。たとえば、PDBごとのログ・マイニングを取得する場合は、START_LOGMNRの実行時に時間またはSCNを指定する必要があります。

親トピック: ビューでのLogMiner操作情報へのアクセス

24.11.2 V$LOGMNR_LOGSの問合せ

LogMinerで分析するREDOログ・ファイルのリストに手動または自動で追加されたREDOログ・ファイルを特定するには、V$LOGMNR_LOGSビューを問い合せられます。

V$LOGMNR_LOGSには、REDOログ・ファイルごとに行が含まれます。このビューでは、各REDOログ・ファイルに関する重要な情報(ファイル名、SCN範囲、時間範囲、LogMinerディクショナリのすべてを含むか一部を含むかなど)が提供されます。

DBMS_LOGMNR.START_LOGMNRに対するコールが成功すると、V$LOGMNR_LOGSビューのSTATUS列には、次のいずれかの値が含まれます。

- 0

  REDOログ・ファイルが、V$LOGMNR_CONTENTSビューの問合せ中に処理されることを示します。

- 1

  対象のREDOログ・ファイルが、V$LOGMNR_CONTENTSビューに対する選択操作中にLogMinerによって処理される、最初のREDOログ・ファイルとなることを示します。

- 2

  REDOログ・ファイルが除外されたため、V$LOGMNR_CONTENTSビューの問合せ中にLogMinerによって処理されないことを示します。REDOログ・ファイルは、要求した時間範囲またはSCN範囲を満たすには不要なため、除外されました。

- 4

  LogMiner REDOログ・ファイル・リストから、REDOログ・ファイル(順序番号に基づく)が欠落していることを示します。

V$LOGMNR_LOGSビューには、次のように、リストから欠落している各REDOログ・ファイルに対する行が1行含まれます。

- FILENAME列には、順序番号の連続する範囲とSCN範囲の差異の合計が含まれます。

  たとえば: スレッド番号1、順序番号100から102の欠落ログ・ファイル。

- INFO列には、MISSING_LOGFILEという文字列が含まれます。

REDOログ・ファイル・リストから欠落しているファイルの情報は、次の場合に有効な場合があります。

- DBMS_LOGMNR.START_LOGMNRのコール時に指定可能なDDL_DICT_TRACKINGオプションによって、要求した時間範囲またはSCN範囲に対するLogMiner REDOログ・ファイル・リストから、REDOログ・ファイルが欠落しなくなります。DBMS_LOGMNR.START_LOGMNRに対するコールが失敗した場合、V$LOGMNR_LOGSビューのSTATUS列を問い合せることによって、リストから欠落しているREDOログ・ファイルを判別できます。その後、欠落しているREDOログ・ファイルを検索して、手動で追加すると、再度DBMS_LOGMNR.START_LOGMNRをコールできます。

ノート:

dbms_logmnr.start_logmnr パッケージの continuous_mine オプションは、Oracle

Database 19c (19.1)ではサポートされなくなり、使用できなくなりました。

- DBMS_LOGMNR.START_LOGMNRへのコール時に指定可能なその他のオプションでは、LogMiner REDOログ・ファイル・リストからファイルが欠落する場合がありますが、ファイルの欠落は、可能なかぎり回避する必要があります。V$LOGMNR_CONTENTSビューを問い合せる前に、V$LOGMNR_LOGSビューを問い合せて、必要なすべてのファイルがリストにあることを確認できます。リストに欠落ファイルがある場合にV$LOGMNR_CONTENTSビューを問い合せると、次の列の値を持つ行がV$LOGMNR_CONTENTSに返されます。
  - OPERATION列の値MISSING_SCN
  - STATUS列の値1291
  - INFO列で、欠落しているSCN範囲を示す文字列。たとえば: SCN 100から200がありません

親トピック: ビューでのLogMiner操作情報へのアクセス

24.11.3 サプリメンタル・ロギング設定に関するビューの問合せ

サプリメンタル・ロギングの現在の設定を確認するには、様々なビューを問い合せます。

必要な情報に応じて、複数のビューのいずれかを使用できます。

- V$DATABASEビュー
  - SUPPLEMENTAL_LOG_DATA_FK列

    この列には次のいずれかの値が入ります。
    - NO - FOREIGN KEYオプションを指定したデータベース・レベルの識別キー・ロギングが有効でない場合
    - YES - FOREIGN KEYオプションを指定したデータベース・レベルの識別キー・ロギングが有効な場合
  - SUPPLEMENTAL_LOG_DATA_ALL列

    この列には次のいずれかの値が入ります。
    - NO - ALLオプションを指定したデータベース・レベルの識別キー・ロギングが有効でない場合
    - YES - ALLオプションを指定したデータベース・レベルの識別キー・ロギングが有効な場合
  - SUPPLEMENTAL_LOG_DATA_UI列
    - NO - UNIQUEオプションを指定したデータベース・レベルの識別キー・ロギングが有効でない場合
    - YES - UNIQUEオプションを指定したデータベース・レベルの識別キー・ロギングが有効な場合
  - SUPPLEMENTAL_LOG_DATA_MIN列

    この列には次のいずれかの値が入ります。
    - NO - データベース・レベルのサプリメンタル・ロギングが有効でない場合
    - IMPLICIT - データベース・レベルの識別キー・ロギング・オプションが有効なため、最小サプリメンタル・ロギングが有効な場合
    - YES - SQL ALTER DATABASE ADD SUPPLEMENTAL LOG DATA文が発行されているため、最小サプリメンタル・ロギングが有効な場合

- DBA_LOG_GROUPS、ALL_LOG_GROUPS、USER_LOG_GROUPSの各ビュー
  - ALWAYS列

    この列には次のいずれかの値が入ります。

    - ALWAYS - このログ・グループの列は、関連付けられた行の列が更新された場合に補助的に記録されることを示します。
    - CONDITIONAL - このグループの列は、ログ・グループの列が更新された場合にのみ補助的に記録されることを示します。
  - GENERATED列

    この列には次のいずれかの値が入ります。

    - GENERATED NAME - LOG_GROUP名がシステムによって生成された場合です。
    - USER NAME - LOG_GROUP名がユーザーによって定義された場合です。
  - LOG_GROUP_TYPE列

    この列には、次のいずれかの値が入り、このログ・グループで定義されたロギングの種類を示します。USER LOG GROUPは、ログ・グループがユーザー定義である(システム生成ではない)ことを示します。

    - ALL COLUMN LOGGING
    - FOREIGN KEY LOGGING
    - PRIMARY KEY LOGGING
    - UNIQUE KEY LOGGING
    - USER LOG GROUP
- DBA_LOG_GROUP_COLUMNS、ALL_LOG_GROUP_COLUMNS、USER_LOG_GROUP_COLUMNSの各ビュー
  - LOGGING_PROPERTY列

    この列には次のいずれかの値が入ります。

    - LOG - ログ・グループのこの列が補助的に記録されることを示します。
    - NO LOG - ログ・グループのこの列が補助的に記録されないことを示します。

親トピック: ビューでのLogMiner操作情報へのアクセス

24.11.4 LogMinerを使用した個々のPDBの問合せ

ディクショナリ・ビルドを時間またはSCNで検索するには(たとえば、PDBごとのマイニングを開始する場合)、ソース・データベースでSYS.DBA_LOGMNR_DICTIONARY_BUILDLOGビューを使用できます。

Oracle Database 19c (リリース更新10以降)以降では、CDB$ROOTまたは個々のPDBのどちらに接続するかを選択できます。

従来のオンプレミス・ログ・マイニング・セッションでは、CDB$ROOTに接続すると、CDB$ROOTおよびPDBを含むマルチテナント・アーキテクチャ全体に対して問合せが実行されます。PDBごとのログ・マイニング・セッションでは、特定のPDBに接続すると、LogMinerは接続したPDBの行のみを返します。この方法は、Oracle Autonomous Cloud Platform ServicesでOracle Autonomous DatabaseのREDOログ・ファイルを問い合せる場合に必要です。

PDBのログ履歴情報を表示するには、引き続きV$LOGMNR_CONTENTSビューを使用します。ただし、PDBのLogMinerを

起動するために、ログ・ファイルを追加しなくなりました。かわりに、DBMS_LOGMNR.START_LOGMNRをコールし、表示するPDBログ履歴のシステム変更番号(SCN)を指定します。PDBのDBA_LOGMNR_DICTIONARY_BUILDLOGビューに表示される任意のSTART_SCN値を使用できます。

> ノート:
>
> LogMiner の起動時に、表示するログ履歴の ENDSCN または ENDTIME の値がわかっている場合は、これらの終了値のいずれかを指定できます。

例24-1 SYS.DBA_LOGMNR_DICTIONARYの問合せ

次の例では、PDBに接続した後、DBA_LOGMNR_DICTIONARY_BUILDLOGを問い合せてSTART_SCN値を特定し、DBMS_LOGMNR.START_LOGMNRでLogMinerを起動して、問い合せるログのSCN値を指定します。

```
SQL> execute dbms_logmnr_d.build(options => dbms_logmnr_d.store_in_redo_logs);
PL/SQL procedure successfully completed.
SQL> select date_of_build, start_scn from dba_logmnr_dictionary_buildlog;
DATE_OF_BUILD  START_SCN
-------------------- --------------------
09/02/2020 15:58:42 2104064
09/02/2020 19:35:36 3943026
09/02/2020 19:35:54 3943543
09/02/2020 19:35:57 3944009
09/02/2020 19:36:00 3944473
09/10/2020 20:13:22 5902422
09/15/2020 10:03:16 7196131
7 rows selected.
SQL> execute dbms_logmnr.start_logmnr(Options => dbms_logmnr.DDL_DICT_TRACKING +
dbms_logmnr.DICT_FROM_REDO_LOGS, startscn=>5902422);
PL/SQL procedure successfully completed.
SQL> select count(sql_redo) from v$logmnr_contents;
     COUNT(SQL_REDO)
--------------------
             619958
SQL>
```

親トピック: ビューでのLogMiner操作情報へのアクセス

24.12 LogMinerの使用例

データ・マイニングにLogMinerを使用する方法を確認するには、示されている例を確認してください。

> ノート:
>
> この項のすべての例では、最小サプリメンタル・ロギングが次のように有効にされているとします。
>
> ```
> SQL> ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;
> ```
>
> LogMiner のユースケース・シナリオの例を除くすべての例で、NLS_DATE_FORMAT パラメータが次のように設定されていると想定しています。
>
> ```
> SQL>  ALTER SESSION SET NLS_DATE_FORMAT = 'dd-mon-yyyy hh24:mi:ss';
> ```
>
> LogMiner ではユーザー・セッションに対してアクティブな NLS_DATE_FORMAT パラメータの設定を使用して日付

データが表示されるため、このステップはオプションです。ただし、このパラメータを明示的に設定すると、指定した日付書式が使用されます。

- 分析するREDOログ・ファイルの明示的指定によるマイニングの例
  例を使用して、REDOログ・ファイルを指定する方法を確認してください。
- LogMinerのユースケース・シナリオ
  LogMinerを使用してデータ・マイニング・タスクを実行する方法の一般的な例を確認してください。

関連トピック

- サプリメンタル・ロギング

親トピック: REDOログ・ファイル分析のためのLogMinerの使用

24.12.1 分析するREDOログ・ファイルの明示的指定によるマイニングの例

例を使用して、REDOログ・ファイルの指定方法を確認してください。

これらの例では、分析対象データが存在するREDOログ・ファイルがわかっている場合のLogMinerの使用方法を示します。それぞれの例は、前の例に基づいて作成されているため、順番に読み進むことをお薦めします。

SQL出力の画面上の書式設定は、この項で示す例とは異なる場合があります。

ノート:

`dbms_logmnr.start_logmnr` パッケージの `continuous_mine` オプションは、Oracle Database 19c (19.1)ではサポートされなくなり、使用できなくなりました。ログ・ファイルを手動で指定する必要があります

- 例1: 最後にアーカイブされたREDOログ・ファイルでのすべての変更の検索
  LogMinerのデフォルト動作では、トランザクションがコミット済であるかどうかに関係なく、分析するREDOログ・ファイルで検出されたすべての変更が表示されます。
- 例2: コミット済トランザクションへのDML文のグループ化
  LogMinerを使用してREDOログ・トランザクションをグループ化する方法について説明します。
- 例3: 再構築されたSQLの書式設定
  視覚的な検査を簡単に行うには、`PRINT_PRETTY_SQL`オプションを指定してLogMinerを実行します。
- 例4: REDOログ・ファイル内のLogMinerディクショナリの使用
  REDOログ・ファイルに抽出されたディクショナリの使用方法について説明します。
- 例5: 内部ディクショナリでのDDL文の追跡
  `DBMS_LOGMNR.DDL_DICT_TRACKING`オプションを使用して、REDOログ・ファイル内で発生したDDL文でLogMiner内部ディクショナリを更新する方法について説明します。
- 例6: 時間範囲による出力のフィルタ
  時間ごとに一連のREDOログをフィルタするには、時間範囲を指定してログ・ファイルを返す様々な方法について学習してください。

親トピック: LogMinerの使用例

24.12.1.1 例1: 最後にアーカイブされたREDOログ・ファイルでのすべての変更の検索

LogMinerのデフォルト動作では、トランザクションがコミット済であるかどうかに関係なく、分析するREDOログ・ファイルで検出されたすべての変更が表示されます。

データベースの変更履歴を確認する最も簡単な方法は、ソース・データベースでマイニングし、オンライン・カタログを使用してREDOログ・ファイルを変換する方法です。この例では、LogMinerを使用して最も簡単に分析を行う方法を示します。

この例では、最後にアーカイブされたREDOログ・ファイルをマイニングするとします。ここでは、最後にアーカイブされたREDOログ(データベースで生成)に含まれているすべての変更を検索します(データベースは、Oracle Real Application Clusters (Oracle RAC)データベースでないものとします)。

1. 最後にアーカイブされたREDOログ・ファイルを判別します。

```
SELECT NAME FROM V$ARCHIVED_LOG
   WHERE FIRST_TIME = (SELECT MAX(FIRST_TIME) FROM V$ARCHIVED_LOG);
NAME
--------------------------------------------
/usr/oracle/data/db1arch_1_16_482701534.dbf
```

2. 分析するREDOログ・ファイルのリストを指定します。この場合は、ステップ1の問合せで返されたREDOログ・ファイルです。

```
EXECUTE DBMS_LOGMNR.ADD_LOGFILE( -
  LOGFILENAME => '/usr/oracle/data/db1arch_1_16_482701534.dbf', -
  OPTIONS => DBMS_LOGMNR.NEW);
```

3. LogMinerを起動し、使用するディクショナリを指定します。

```
EXECUTE DBMS_LOGMNR.START_LOGMNR( -
   OPTIONS => DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG);
```

4. V$LOGMNR_CONTENTSビューを問い合せます。

   4つのトランザクションがあることに注意してください(そのうちの2つは、分析するREDOログ・ファイル内でコミット済であり、2つはコミットされていません)。出力には、実行された順序でDML文が表示されるため、トランザクションは交互に配置されます。

```
SELECT username AS USR, (XIDUSN || '.' || XIDSLT || '.' ||  XIDSQN) AS XID,
   SQL_REDO, SQL_UNDO FROM V$LOGMNR_CONTENTS WHERE username IN ('HR', 'OE');
USR    XID          SQL_REDO                          SQL_UNDO
----   ---------  -----------------------------------------------------
HR     1.11.1476  set transaction read write;
HR     1.11.1476  insert into "HR"."EMPLOYEES"(     delete from
"HR"."EMPLOYEES"
                  "EMPLOYEE_ID","FIRST_NAME",       where "EMPLOYEE_ID" = '306'
                  "LAST_NAME","EMAIL",              and "FIRST_NAME" =
'Nandini'
                  "PHONE_NUMBER","HIRE_DATE",       and "LAST_NAME" = 'Shastry'
                  "JOB_ID","SALARY",                and "EMAIL" = 'NSHASTRY'
                  "COMMISSION_PCT","MANAGER_ID",    and "PHONE_NUMBER" =
'1234567890'
                  "DEPARTMENT_ID") values           and "HIRE_DATE" =
TO_DATE('10-JAN-2012
                  ('306','Nandini','Shastry',       13:34:43', 'dd-mon-yyyy
hh24:mi:ss')
                  'NSHASTRY', '1234567890',         and "JOB_ID" = 'HR_REP' and
                  TO_DATE('10-jan-2012 13:34:43',   "SALARY" = '120000' and
                  'dd-mon-yyyy hh24:mi:ss'),         "COMMISSION_PCT" = '.05'
and
                  'HR_REP','120000', '.05',         "DEPARTMENT_ID" = '10' and
                  '105','10');                      ROWID =
'AAAHSkAABAAAY6rAAO';

OE     1.1.1484   set transaction read write;
OE     1.1.1484   update "OE"."PRODUCT_INFORMATION"  update
"OE"."PRODUCT_INFORMATION"
```

```
                   set "WARRANTY_PERIOD" =            set "WARRANTY_PERIOD" =
                   TO_YMINTERVAL('+05-00') where      TO_YMINTERVAL('+01-00')
where
                   "PRODUCT_ID" = '1799' and          "PRODUCT_ID" = '1799' and
                   "WARRANTY_PERIOD" =                "WARRANTY_PERIOD" =
                   TO_YMINTERVAL('+01-00') and        TO_YMINTERVAL('+05-00')
and
                   ROWID = 'AAAHTKAABAAAY9mAAB';      ROWID =
'AAAHTKAABAAAY9mAAB';

OE     1.1.1484   update "OE"."PRODUCT_INFORMATION"  update
"OE"."PRODUCT_INFORMATION"
                   set "WARRANTY_PERIOD" =            set "WARRANTY_PERIOD" =
                   TO_YMINTERVAL('+05-00') where      TO_YMINTERVAL('+01-00')
where
                   "PRODUCT_ID" = '1801' and          "PRODUCT_ID" = '1801' and
                   "WARRANTY_PERIOD" =                "WARRANTY_PERIOD" =
                   TO_YMINTERVAL('+01-00') and        TO_YMINTERVAL('+05-00')
and
                   ROWID = 'AAAHTKAABAAAY9mAAC';      ROWID
='AAAHTKAABAAAY9mAAC';
HR     1.11.1476  insert into "HR"."EMPLOYEES"(     delete from
"HR"."EMPLOYEES"
                   "EMPLOYEE_ID","FIRST_NAME",       "EMPLOYEE_ID" = '307' and
                   "LAST_NAME","EMAIL",              "FIRST_NAME" = 'John' and
                   "PHONE_NUMBER","HIRE_DATE",       "LAST_NAME" = 'Silver' and
                   "JOB_ID","SALARY",                "EMAIL" = 'JSILVER' and
                   "COMMISSION_PCT","MANAGER_ID",    "PHONE_NUMBER" =
'5551112222'
                   "DEPARTMENT_ID") values           and "HIRE_DATE" =
TO_DATE('10-jan-2012
                   ('307','John','Silver',           13:41:03', 'dd-mon-yyyy
hh24:mi:ss')
                    'JSILVER', '5551112222',         and "JOB_ID" ='105' and
"DEPARTMENT_ID"
                   TO_DATE('10-jan-2012 13:41:03',   = '50' and ROWID =
'AAAHSkAABAAAY6rAAP';
                   'dd-mon-yyyy hh24:mi:ss'),
                   'SH_CLERK','110000', '.05',
                   '105','50');
OE     1.1.1484   commit;
HR     1.15.1481   set transaction read write;
HR     1.15.1481  delete from "HR"."EMPLOYEES"      insert into
"HR"."EMPLOYEES"(
                   where "EMPLOYEE_ID" = '205' and   "EMPLOYEE_ID","FIRST_NAME",
                   "FIRST_NAME" = 'Shelley' and
"LAST_NAME","EMAIL","PHONE_NUMBER",
                   "LAST_NAME" = 'Higgins' and       "HIRE_DATE",
"JOB_ID","SALARY",
                   "EMAIL" = 'SHIGGINS' and
"COMMISSION_PCT","MANAGER_ID",
                   "PHONE_NUMBER" = '515.123.8080'   "DEPARTMENT_ID") values
                   and "HIRE_DATE" = TO_DATE(        ('205','Shelley','Higgins',
                   '07-jun-1994 10:05:01',           and
'SHIGGINS','515.123.8080',
                   'dd-mon-yyyy hh24:mi:ss')         TO_DATE('07-jun-1994
10:05:01',
                   and "JOB_ID" = 'AC_MGR'           'dd-mon-yyyy hh24:mi:ss'),
                   and "SALARY"= '12000'
'AC_MGR','12000',NULL,'101','110');
                   and "COMMISSION_PCT" IS NULL
                   and "MANAGER_ID"
                   = '101' and "DEPARTMENT_ID" =
                   '110' and ROWID =
                   'AAAHSkAABAAAY6rAAM';
OE     1.8.1484   set transaction read write;
```

```
OE      1.8.1484   update "OE"."PRODUCT_INFORMATION"  update
"OE"."PRODUCT_INFORMATION"
                   set "WARRANTY_PERIOD" =            set "WARRANTY_PERIOD" =
                   TO_YMINTERVAL('+12-06') where      TO_YMINTERVAL('+20-00')
where
                   "PRODUCT_ID" = '2350' and          "PRODUCT_ID" = '2350' and
                   "WARRANTY_PERIOD" =                "WARRANTY_PERIOD" =
                   TO_YMINTERVAL('+20-00') and        TO_YMINTERVAL('+20-00')
and
                   ROWID = 'AAAHTKAABAAAY9tAAD';       ROWID
='AAAHTKAABAAAY9tAAD';
HR      1.11.1476  commit;
```

5. LogMinerセッションを終了します。

```
SQL> EXECUTE DBMS_LOGMNR.END_LOGMNR();
```

親トピック: [分析するREDOログ・ファイルの明示的指定によるマイニングの例]

24.12.1.2 例2: コミット済トランザクションへのDML文のグループ化

LogMinerを使用してREDOログ・トランザクションをグループ化する方法を学習します。

例1に示すように、LogMinerのデフォルト動作では、トランザクションがコミット済であるかどうかに関係なく、分析するREDOログ・ファイルで検出されたすべての変更が表示されます。また、LogMinerでは、実行された順序で変更が表示されます。同じトランザクションに属するDML文がグループにまとめられていないため、出力を目視で簡単に確認することはできません。SQLを使用してトランザクションをグループ化することもできますが、LogMinerでは、より簡単な方法が提供されています。この例では、最後にアーカイブされたREDOログ・ファイルを再度分析しますが、コミット済トランザクションのみが返されます。

1. データベースによって最後にアーカイブされたREDOログ・ファイルを判別します。

```
SELECT NAME FROM V$ARCHIVED_LOG
   WHERE FIRST_TIME = (SELECT MAX(FIRST_TIME) FROM V$ARCHIVED_LOG);
NAME
--------------------------------------------
/usr/oracle/data/db1arch_1_16_482701534.dbf
```

2. ステップ1の問合せで返されたREDOログ・ファイルを指定します。このリストは、1つのREDOログ・ファイルで構成されます。

```
EXECUTE DBMS_LOGMNR.ADD_LOGFILE( -
   LOGFILENAME => '/usr/oracle/data/db1arch_1_16_482701534.dbf', -
   OPTIONS => DBMS_LOGMNR.NEW);
```

3. 使用するディクショナリとCOMMITTED_DATA_ONLYオプションを指定して、LogMinerを起動します。

```
EXECUTE DBMS_LOGMNR.START_LOGMNR( -
   OPTIONS => DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG + -
   DBMS_LOGMNR.COMMITTED_DATA_ONLY);
```

4. V$LOGMNR_CONTENTSビューを問い合せます。

   トランザクション1.11.1476は、トランザクション1.1.1484 (ステップ1で説明)より前に起動されましたが、これはトランザクション1.1.1484がコミットされた後にコミットされました。したがって、この例では、トランザクション1.1.1484は、その全体がトランザクション1.11.1476より前に示されます。分析中のREDOログ・ファイル内の、コミットされなかった2つのトランザクションは返されません。

```
SELECT username AS USR, (XIDUSN || '.' || XIDSLT || '.' ||  XIDSQN) AS XID,
SQL_REDO,
SQL_UNDO FROM V$LOGMNR_CONTENTS WHERE username IN ('HR', 'OE');
```

```
;
USR    XID          SQL_REDO                          SQL_UNDO
----   ---------    ------------------------------- ---------------------------
------

OE     1.1.1484   set transaction read write;
OE     1.1.1484   update "OE"."PRODUCT_INFORMATION"  update
"OE"."PRODUCT_INFORMATION"
                  set "WARRANTY_PERIOD" =            set "WARRANTY_PERIOD" =
                  TO_YMINTERVAL('+05-00') where      TO_YMINTERVAL('+01-00')
where
                  "PRODUCT_ID" = '1799' and          "PRODUCT_ID" = '1799' and
                  "WARRANTY_PERIOD" =                "WARRANTY_PERIOD" =
                  TO_YMINTERVAL('+01-00') and        TO_YMINTERVAL('+05-00')
and
                  ROWID = 'AAAHTKAABAAAY9mAAB';      ROWID =
'AAAHTKAABAAAY9mAAB';

OE     1.1.1484   update "OE"."PRODUCT_INFORMATION"  update
"OE"."PRODUCT_INFORMATION"
                  set "WARRANTY_PERIOD" =            set "WARRANTY_PERIOD" =
                  TO_YMINTERVAL('+05-00') where      TO_YMINTERVAL('+01-00')
where
                  "PRODUCT_ID" = '1801' and          "PRODUCT_ID" = '1801' and
                  "WARRANTY_PERIOD" =                "WARRANTY_PERIOD" =
                  TO_YMINTERVAL('+01-00') and        TO_YMINTERVAL('+05-00')
and
                  ROWID = 'AAAHTKAABAAAY9mAAC';      ROWID
='AAAHTKAABAAAY9mAAC';
OE     1.1.1484   commit;

HR     1.11.1476  set transaction read write;
HR     1.11.1476  insert into "HR"."EMPLOYEES"(     delete from
"HR"."EMPLOYEES"
                  "EMPLOYEE_ID","FIRST_NAME",       where "EMPLOYEE_ID" = '306'
                  "LAST_NAME","EMAIL",              and "FIRST_NAME" =
'Nandini'
                  "PHONE_NUMBER","HIRE_DATE",       and "LAST_NAME" = 'Shastry'
                  "JOB_ID","SALARY",                and "EMAIL" = 'NSHASTRY'
                  "COMMISSION_PCT","MANAGER_ID",    and "PHONE_NUMBER" =
'1234567890'
                  "DEPARTMENT_ID") values           and "HIRE_DATE" =
TO_DATE('10-JAN-2012
                  ('306','Nandini','Shastry',       13:34:43', 'dd-mon-yyyy
hh24:mi:ss')
                  'NSHASTRY', '1234567890',         and "JOB_ID" = 'HR_REP' and
                  TO_DATE('10-jan-2012 13:34:43',   "SALARY" = '120000' and
                  'dd-mon-yyy hh24:mi:ss'),         "COMMISSION_PCT" = '.05'
and
                  'HR_REP','120000', '.05',         "DEPARTMENT_ID" = '10' and
                  '105','10');                      ROWID =
'AAAHSkAABAAAY6rAAO';
HR     1.11.1476  insert into "HR"."EMPLOYEES"(     delete from
"HR"."EMPLOYEES"
                  "EMPLOYEE_ID","FIRST_NAME",       "EMPLOYEE_ID" = '307' and
                  "LAST_NAME","EMAIL",              "FIRST_NAME" = 'John' and
                  "PHONE_NUMBER","HIRE_DATE",       "LAST_NAME" = 'Silver' and
                  "JOB_ID","SALARY",                "EMAIL" = 'JSILVER' and
                  "COMMISSION_PCT","MANAGER_ID",    "PHONE_NUMBER" =
'5551112222'
                  "DEPARTMENT_ID") values           and "HIRE_DATE" =
TO_DATE('10-jan-2012
                  ('307','John','Silver',           13:41:03', 'dd-mon-yyyy
hh24:mi:ss')
                   'JSILVER', '5551112222',         and "JOB_ID" ='105' and
"DEPARTMENT_ID"
```

```
                    TO_DATE('10-jan-2012 13:41:03',   = '50' and ROWID =
'AAAHSkAABAAAY6rAAP';
                    'dd-mon-yyyy hh24:mi:ss'),
                    'SH_CLERK','110000', '.05',
                    '105','50');
HR      1.11.1476   commit;
```

5. LogMinerセッションを終了します。

```
EXECUTE DBMS_LOGMNR.END_LOGMNR();
```

親トピック: 分析するREDOログ・ファイルの明示的指定によるマイニングの例

24.12.1.3 例3: 再構築されたSQLの書式設定

視覚的な検査を簡単に行うには、PRINT_PRETTY_SQLオプションを指定してLogMinerを実行します。

例2で示したとおり、オンラインREDOログ・ファイルのディクショナリでCOMMITTED_DATA_ONLYオプションを使用する方法は、コミット済トランザクションのみを対象とするための簡単な方法です。ただし、目視で簡単に確認はできません。INSERT文内の列名とそれに対応する値との間の関連が明白でないためです。PRINT_PRETTY_SQLオプションを指定することで、この問題に対処できます。このオプションを指定すると、再構築されたSQL文の一部が実行不可能になることに注意してください。

1. 最後にアーカイブされたREDOログ・ファイルを判別します。

```
SELECT NAME FROM V$ARCHIVED_LOG
   WHERE FIRST_TIME = (SELECT MAX(FIRST_TIME) FROM V$ARCHIVED_LOG);
NAME
--------------------------------------------
/usr/oracle/data/db1arch_1_16_482701534.dbf
```

2. ステップ1の問合せで返されたREDOログ・ファイルを指定します。

```
EXECUTE DBMS_LOGMNR.ADD_LOGFILE( -
   LOGFILENAME => '/usr/oracle/data/db1arch_1_16_482701534.dbf', -
   OPTIONS => DBMS_LOGMNR.NEW);
```

3. 使用するディクショナリおよびCOMMITTED_DATA_ONLYオプションとPRINT_PRETTY_SQLオプションを指定して、LogMinerを起動します。

```
EXECUTE DBMS_LOGMNR.START_LOGMNR(-
   OPTIONS => DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG + -
              DBMS_LOGMNR.COMMITTED_DATA_ONLY + -
              DBMS_LOGMNR.PRINT_PRETTY_SQL);
```

DBMS_LOGMNR.PRINT_PRETTY_SQLオプションは、再構築されたSQLの書式のみを変更するため、目視用のレポートを生成する場合に有効です。

4. SQL_REDO文に対するV$LOGMNR_CONTENTSビューを問い合せます。

```
SELECT username AS USR, (XIDUSN || '.' || XIDSLT || '.' ||  XIDSQN) AS XID,
SQL_REDO
   FROM V$LOGMNR_CONTENTS;
USR     XID          SQL_REDO
----    ---------    -------------------------------------------------------
OE      1.1.1484     set transaction read write;
OE      1.1.1484     update "OE"."PRODUCT_INFORMATION"
                       set
                         "WARRANTY_PERIOD" = TO_YMINTERVAL('+05-00')
                       where
                         "PRODUCT_ID" = '1799' and
                         "WARRANTY_PERIOD" = TO_YMINTERVAL('+01-00') and
                         ROWID = 'AAAHTKAABAAAY9mAAB';
```

```
OE     1.1.1484   update "OE"."PRODUCT_INFORMATION"
                    set
                      "WARRANTY_PERIOD" = TO_YMINTERVAL('+05-00')
                    where
                      "PRODUCT_ID" = '1801' and
                      "WARRANTY_PERIOD" = TO_YMINTERVAL('+01-00') and
                      ROWID = 'AAAHTKAABAAAY9mAAC';
OE     1.1.1484   commit;

HR     1.11.1476  set transaction read write;
HR     1.11.1476  insert into "HR"."EMPLOYEES"
                   values
                     "EMPLOYEE_ID" = 306,
                     "FIRST_NAME" = 'Nandini',
                     "LAST_NAME" = 'Shastry',
                     "EMAIL" = 'NSHASTRY',
                     "PHONE_NUMBER" = '1234567890',
                     "HIRE_DATE" = TO_DATE('10-jan-2012 13:34:43',
                     'dd-mon-yyyy hh24:mi:ss',
                     "JOB_ID" = 'HR_REP',
                     "SALARY" = 120000,
                     "COMMISSION_PCT" = .05,
                     "MANAGER_ID" = 105,
                     "DEPARTMENT_ID" = 10;
HR     1.11.1476   insert into "HR"."EMPLOYEES"
                    values
                       "EMPLOYEE_ID" = 307,
                       "FIRST_NAME" = 'John',
                       "LAST_NAME" = 'Silver',
                       "EMAIL" = 'JSILVER',
                       "PHONE_NUMBER" = '5551112222',
                       "HIRE_DATE" = TO_DATE('10-jan-2012 13:41:03',
                       'dd-mon-yyyy hh24:mi:ss'),
                       "JOB_ID" = 'SH_CLERK',
                       "SALARY" = 110000,
                       "COMMISSION_PCT" = .05,
                       "MANAGER_ID" = 105,
                       "DEPARTMENT_ID" = 50;
HR     1.11.1476    commit;
```

5. 再構築されたSQL_UNDO文に対するV$LOGMNR_CONTENTSビューを問い合せます。

```
SELECT username AS USR, (XIDUSN || '.' || XIDSLT || '.' ||  XIDSQN) AS XID,
SQL_UNDO
   FROM V$LOGMNR_CONTENTS;
USR   XID        SQL_UNDO
----   ---------  ----------------------------------------------------

OE     1.1.1484   set transaction read write;
OE     1.1.1484   update "OE"."PRODUCT_INFORMATION"
                    set
                      "WARRANTY_PERIOD" = TO_YMINTERVAL('+01-00')
                    where
                      "PRODUCT_ID" = '1799' and
                      "WARRANTY_PERIOD" = TO_YMINTERVAL('+05-00') and
                      ROWID = 'AAAHTKAABAAAY9mAAB';

OE     1.1.1484   update "OE"."PRODUCT_INFORMATION"
                    set
                      "WARRANTY_PERIOD" = TO_YMINTERVAL('+01-00')
                    where
                      "PRODUCT_ID" = '1801' and
                      "WARRANTY_PERIOD" = TO_YMINTERVAL('+05-00') and
                      ROWID = 'AAAHTKAABAAAY9mAAC';
OE     1.1.1484   commit;
```

```
HR     1.11.1476  set transaction read write;
HR     1.11.1476  delete from "HR"."EMPLOYEES"
                  where
                     "EMPLOYEE_ID" = 306 and
                     "FIRST_NAME" = 'Nandini' and
                     "LAST_NAME" = 'Shastry' and
                     "EMAIL" = 'NSHASTRY' and
                     "PHONE_NUMBER" = '1234567890' and
                     "HIRE_DATE" = TO_DATE('10-jan-2012 13:34:43',
                     'dd-mon-yyyy hh24:mi:ss') and
                     "JOB_ID" = 'HR_REP' and
                     "SALARY" = 120000 and
                     "COMMISSION_PCT" = .05 and
                     "MANAGER_ID" = 105 and
                     "DEPARTMENT_ID" = 10 and
                     ROWID = 'AAAHSkAABAAAY6rAAO';
HR     1.11.1476   delete from "HR"."EMPLOYEES"
                    where
                        "EMPLOYEE_ID" = 307 and
                        "FIRST_NAME" = 'John' and
                        "LAST_NAME" = 'Silver' and
                        "EMAIL" = 'JSILVER' and
                        "PHONE_NUMBER" = '555122122' and
                        "HIRE_DATE" = TO_DATE('10-jan-2012 13:41:03',
                        'dd-mon-yyyy hh24:mi:ss') and
                        "JOB_ID" = 'SH_CLERK' and
                        "SALARY" = 110000 and
                        "COMMISSION_PCT" = .05 and
                        "MANAGER_ID" = 105 and
                        "DEPARTMENT_ID" = 50 and
                        ROWID = 'AAAHSkAABAAAY6rAAP';
HR     1.11.1476    commit;
```

6. LogMinerセッションを終了します。

```
EXECUTE DBMS_LOGMNR.END_LOGMNR();
```

親トピック: 分析するREDOログ・ファイルの明示的指定によるマイニングの例

24.12.1.4 例4: REDOログ・ファイル内のLogMinerディクショナリの使用

REDOログ・ファイルに抽出されたディクショナリを使用する方法を学習します。

オンライン・カタログ内のディクショナリを使用する場合は、オンライン・カタログを生成したデータベースと同じデータベース内のREDOログ・ファイルをマイニングする必要があります。REDOログ・ファイルに含まれているディクショナリを使用すると、別のデータベース内のREDOログ・ファイルをマイニングできます。

オンライン・カタログ内のディクショナリを使用する場合は、オンライン・カタログを生成したデータベースと同じデータベース内のREDOログ・ファイルをマイニングする必要があります。REDOログ・ファイルに含まれているディクショナリを使用すると、別のデータベース内のREDOログ・ファイルをマイニングできます。

1. データベースによって最後にアーカイブされたREDOログ・ファイルを判別します。

```
SELECT NAME, SEQUENCE# FROM V$ARCHIVED_LOG
   WHERE FIRST_TIME = (SELECT MAX(FIRST_TIME) FROM V$ARCHIVED_LOG);
NAME                                            SEQUENCE#
----------------------------------------------  --------------
/usr/oracle/data/db1arch_1_210_482701534.dbf    210
```

2. ディクショナリは、複数のREDOログ・ファイルに含まれている場合があります。したがって、ディクショナリの先頭と終わりが含まれているREDOログ・ファイルを判別する必要があります。次の手順を実行して、V$ARCHIVED_LOGビューを問

い合せます。

a. ディクショナリ抽出の終わりが含まれているREDOログ・ファイルを検索します。このREDOログ・ファイルは、分析するREDOログ・ファイルより前に作成されている必要がありますが、できるかぎり新しいものを使用します。

```
SELECT NAME, SEQUENCE#, DICTIONARY_BEGIN d_beg, DICTIONARY_END d_end
   FROM V$ARCHIVED_LOG
   WHERE SEQUENCE# = (SELECT MAX (SEQUENCE#) FROM V$ARCHIVED_LOG
   WHERE DICTIONARY_END = 'YES' and SEQUENCE# <= 210);
NAME                                            SEQUENCE#    D_BEG  D_END
----------------------------------------------  ----------   -----  ------
/usr/oracle/data/db1arch_1_208_482701534.dbf    208          NO     YES
```

b. データ・ディクショナリ抽出の先頭(前のステップで検出されたディクショナリの終わりに対応)が含まれているREDOログ・ファイルを検索します。

```
SELECT NAME, SEQUENCE#, DICTIONARY_BEGIN d_beg, DICTIONARY_END d_end
   FROM V$ARCHIVED_LOG
   WHERE SEQUENCE# = (SELECT MAX (SEQUENCE#) FROM V$ARCHIVED_LOG
   WHERE DICTIONARY_BEGIN = 'YES' and SEQUENCE# <= 208);
NAME                                            SEQUENCE#    D_BEG  D_END
----------------------------------------------  ----------   -----  ------
/usr/oracle/data/db1arch_1_207_482701534.dbf    207          YES     NO
```

c. 分析するREDOログ・ファイルのリストを指定します。ディクショナリの先頭と終わりが含まれているREDOログ・ファイル、および分析するREDOログ・ファイルを追加します。REDOログ・ファイルは任意の順序で追加できます。

```
EXECUTE DBMS_LOGMNR.ADD_LOGFILE(-
   LOGFILENAME => '/usr/oracle/data/db1arch_1_210_482701534.dbf', -
       OPTIONS => DBMS_LOGMNR.NEW);
EXECUTE DBMS_LOGMNR.ADD_LOGFILE(-
   LOGFILENAME => '/usr/oracle/data/db1arch_1_208_482701534.dbf');
EXECUTE DBMS_LOGMNR.ADD_LOGFILE(-
   LOGFILENAME => '/usr/oracle/data/db1arch_1_207_482701534.dbf');
```

d. V$LOGMNR_LOGSビューを問い合せて、分析するREDOログ・ファイルのリストをタイムスタンプとともに表示します。

出力では、LogMinerによって欠落しているREDOログ・ファイルに指摘フラグが付けられています。正常に機能するために欠落しているREDOログ・ファイルが必要なオプションを指定していない場合は、LogMinerでマイニングを続行できます。

3. 使用するディクショナリおよびCOMMITTED_DATA_ONLYオプションとPRINT_PRETTY_SQLオプションを指定して、LogMinerを起動します。

```
EXECUTE DBMS_LOGMNR.START_LOGMNR(-
   OPTIONS => DBMS_LOGMNR.DICT_FROM_REDO_LOGS + -
              DBMS_LOGMNR.COMMITTED_DATA_ONLY + -
              DBMS_LOGMNR.PRINT_PRETTY_SQL);
```

4. V$LOGMNR_CONTENTSビューを問い合せます。

問合せによって返される行数を減らすには、SYSスキーマまたはSYSTEMスキーマで実行されたすべてのDML文を問合せから除外します。(この問合せでは、ディクショナリ抽出に関連したトランザクションを除外するタイムスタンプを指定します。)

出力には3つのトランザクションが表示されます。2つのDDLトランザクションと1つのDMLトランザクションです。DDLトランザクション1.2.1594は表oe.product_trackingを作成し、1.18.1602は表

oe.product_informationでトリガーを作成します。いずれのトランザクションでも、システム表(SYSによって所有される表)に対して実行されたDML文は、問合せ条件に基づいてフィルタ処理で除外されます。

DMLトランザクション1.9.1598は、oe.product_information表を更新します。このトランザクションの更新操作は完全に変換されます。ただし、問合せの出力には、変換されない再構築されたSQL文も含まれます。これらの文は、データ・ディクショナリがREDOログ・ファイルに抽出された後で作成されたoe.product_tracking表で実行された可能性があります。

(次に示す例は、すべてのSQL文が完全に変換されるように、DDL_DICT_TRACKINGを使用したLogMinerの実行例です。バイナリ・データは返されていません。)

```
SELECT USERNAME AS usr, SQL_REDO FROM V$LOGMNR_CONTENTS
   WHERE SEG_OWNER IS NULL OR SEG_OWNER NOT IN ('SYS', 'SYSTEM') AND
   TIMESTAMP > '10-jan-2012 15:59:53';
USR             XID         SQL_REDO
---             --------    -----------------------------------
SYS             1.2.1594    set transaction read write;
SYS             1.2.1594    create table oe.product_tracking (product_id number
not null,
                            modified_time date,
                            old_list_price number(8,2),
                            old_warranty_period interval year(2) to month);
SYS             1.2.1594    commit;
SYS             1.18.1602   set transaction read write;
SYS             1.18.1602   create or replace trigger
oe.product_tracking_trigger
                            before update on oe.product_information
                            for each row
                            when (new.list_price <> old.list_price or
                                  new.warranty_period <> old.warranty_period)
                            declare
                            begin
                            insert into oe.product_tracking values
                               (:old.product_id, sysdate,
                                :old.list_price, :old.warranty_period);
                            end;
SYS             1.18.1602   commit;
OE              1.9.1598    update "OE"."PRODUCT_INFORMATION"
                              set
                                "WARRANTY_PERIOD" = TO_YMINTERVAL('+08-00'),
                                "LIST_PRICE" = 100
                              where
                                "PRODUCT_ID" = 1729 and
                                "WARRANTY_PERIOD" = TO_YMINTERVAL('+05-00') and
                                "LIST_PRICE" = 80 and
                                ROWID = 'AAAHTKAABAAAY9yAAA';
OE              1.9.1598    insert into "UNKNOWN"."OBJ# 33415"
                              values
                                "COL 1" = HEXTORAW('c2121e'),
                                "COL 2" = HEXTORAW('7867010d110804'),
                                "COL 3" = HEXTORAW('c151'),
                                "COL 4" = HEXTORAW('800000053c');
OE              1.9.1598    update "OE"."PRODUCT_INFORMATION"
                              set
                                "WARRANTY_PERIOD" = TO_YMINTERVAL('+08-00'),
                                "LIST_PRICE" = 92
                              where
                                "PRODUCT_ID" = 2340 and
                                "WARRANTY_PERIOD" = TO_YMINTERVAL('+05-00') and
                                "LIST_PRICE" = 72 and
                                ROWID = 'AAAHTKAABAAAY9zAAA';
OE              1.9.1598    insert into "UNKNOWN"."OBJ# 33415"
                              values
                                "COL 1" = HEXTORAW('c21829'),
```

```
                                    "COL 2" = HEXTORAW('7867010d110808'),
                                    "COL 3" = HEXTORAW('c149'),
                                    "COL 4" = HEXTORAW('800000053c');
OE              1.9.1598     commit;
```

5. 必要に応じて、追加の問合せを発行します。

CREATE TABLE DDL文の一部として実行されたすべてのDML文を表示します。この中には、ユーザーによって実行された文、Oracleによって内部的に実行された文などが含まれます。

> ノート:
>
> ここで示すような問合せによって、表示された文を再適用する場合は、DDL 文のみを再適用します。Oracle によって内部的に実行された DML 文は再適用しないでください。データベースを破損する危険性があります。次の出力では、再適用操作で使用する文は CREATE TABLE OE.PRODUCT_TRACKING 文のみです。

```
SELECT SQL_REDO FROM V$LOGMNR_CONTENTS
   WHERE XIDUSN  = 1 and XIDSLT = 2 and XIDSQN = 1594;
SQL_REDO
--------------------------------------------------------------------------------
-
set transaction read write;
insert into "SYS"."OBJ$"
 values
    "OBJ#" = 33415,
    "DATAOBJ#" = 33415,
    "OWNER#" = 37,
    "NAME" = 'PRODUCT_TRACKING',
    "NAMESPACE" = 1,
    "SUBNAME" IS NULL,
    "TYPE#" = 2,
    "CTIME" = TO_DATE('13-jan-2012 14:01:03', 'dd-mon-yyyy hh24:mi:ss'),
    "MTIME" = TO_DATE('13-jan-2012 14:01:03', 'dd-mon-yyyy hh24:mi:ss'),
    "STIME" = TO_DATE('13-jan-2012 14:01:03', 'dd-mon-yyyy hh24:mi:ss'),
    "STATUS" = 1,
    "REMOTEOWNER" IS NULL,
    "LINKNAME" IS NULL,
    "FLAGS" = 0,
    "OID$" IS NULL,
    "SPARE1" = 6,
    "SPARE2" = 1,
    "SPARE3" IS NULL,
    "SPARE4" IS NULL,
    "SPARE5" IS NULL,
    "SPARE6" IS NULL;
insert into "SYS"."TAB$"
 values
    "OBJ#" = 33415,
    "DATAOBJ#" = 33415,
    "TS#" = 0,
    "FILE#" = 1,
    "BLOCK#" = 121034,
    "BOBJ#" IS NULL,
    "TAB#" IS NULL,
    "COLS" = 5,
    "CLUCOLS" IS NULL,
    "PCTFREE$" = 10,
    "PCTUSED$" = 40,
    "INITRANS" = 1,
    "MAXTRANS" = 255,
```

```
    "FLAGS" = 1,
    "AUDIT$" = '--------------------------------------',
    "ROWCNT" IS NULL,
    "BLKCNT" IS NULL,
    "EMPCNT" IS NULL,
    "AVGSPC" IS NULL,
    "CHNCNT" IS NULL,
    "AVGRLN" IS NULL,
    "AVGSPC_FLB" IS NULL,
    "FLBCNT" IS NULL,
    "ANALYZETIME" IS NULL,
    "SAMPLESIZE" IS NULL,
    "DEGREE" IS NULL,
    "INSTANCES" IS NULL,
    "INTCOLS" = 5,
    "KERNELCOLS" = 5,
    "PROPERTY" = 536870912,
    "TRIGFLAG" = 0,
    "SPARE1" = 178,
    "SPARE2" IS NULL,
    "SPARE3" IS NULL,
    "SPARE4" IS NULL,
    "SPARE5" IS NULL,
    "SPARE6" = TO_DATE('13-jan-2012 14:01:05', 'dd-mon-yyyy hh24:mi:ss'),
insert into "SYS"."COL$"
 values
    "OBJ#" = 33415,
    "COL#" = 1,
    "SEGCOL#" = 1,
    "SEGCOLLENGTH" = 22,
    "OFFSET" = 0,
    "NAME" = 'PRODUCT_ID',
    "TYPE#" = 2,
    "LENGTH" = 22,
    "FIXEDSTORAGE" = 0,
    "PRECISION#" IS NULL,
    "SCALE" IS NULL,
    "NULL$" = 1,
    "DEFLENGTH" IS NULL,
    "SPARE6" IS NULL,
    "INTCOL#" = 1,
    "PROPERTY" = 0,
    "CHARSETID" = 0,
    "CHARSETFORM" = 0,
    "SPARE1" = 0,
    "SPARE2" = 0,
    "SPARE3" = 0,
    "SPARE4" IS NULL,
    "SPARE5" IS NULL,
    "DEFAULT$" IS NULL;
insert into "SYS"."COL$"
 values
    "OBJ#" = 33415,
    "COL#" = 2,
    "SEGCOL#" = 2,
    "SEGCOLLENGTH" = 7,
    "OFFSET" = 0,
    "NAME" = 'MODIFIED_TIME',
    "TYPE#" = 12,
    "LENGTH" = 7,
    "FIXEDSTORAGE" = 0,
    "PRECISION#" IS NULL,
    "SCALE" IS NULL,
    "NULL$" = 0,
    "DEFLENGTH" IS NULL,
    "SPARE6" IS NULL,
    "INTCOL#" = 2,
```

```
    "PROPERTY" = 0,
    "CHARSETID" = 0,
    "CHARSETFORM" = 0,
    "SPARE1" = 0,
    "SPARE2" = 0,
    "SPARE3" = 0,
    "SPARE4" IS NULL,
    "SPARE5" IS NULL,
    "DEFAULT$" IS NULL;
insert into "SYS"."COL$"
 values
    "OBJ#" = 33415,
    "COL#" = 3,
    "SEGCOL#" = 3,
    "SEGCOLLENGTH" = 22,
    "OFFSET" = 0,
    "NAME" = 'OLD_LIST_PRICE',
    "TYPE#" = 2,
    "LENGTH" = 22,
    "FIXEDSTORAGE" = 0,
    "PRECISION#" = 8,
    "SCALE" = 2,
    "NULL$" = 0,
    "DEFLENGTH" IS NULL,
    "SPARE6" IS NULL,
    "INTCOL#" = 3,
    "PROPERTY" = 0,
    "CHARSETID" = 0,
    "CHARSETFORM" = 0,
    "SPARE1" = 0,
    "SPARE2" = 0,
    "SPARE3" = 0,
    "SPARE4" IS NULL,
    "SPARE5" IS NULL,
    "DEFAULT$" IS NULL;
insert into "SYS"."COL$"
 values
    "OBJ#" = 33415,
    "COL#" = 4,
    "SEGCOL#" = 4,
    "SEGCOLLENGTH" = 5,
    "OFFSET" = 0,
    "NAME" = 'OLD_WARRANTY_PERIOD',
    "TYPE#" = 182,
    "LENGTH" = 5,
    "FIXEDSTORAGE" = 0,
    "PRECISION#" = 2,
    "SCALE" = 0,
    "NULL$" = 0,
    "DEFLENGTH" IS NULL,
    "SPARE6" IS NULL,
    "INTCOL#" = 4,
    "PROPERTY" = 0,
    "CHARSETID" = 0,
    "CHARSETFORM" = 0,
    "SPARE1" = 0,
    "SPARE2" = 2,
    "SPARE3" = 0,
    "SPARE4" IS NULL,
    "SPARE5" IS NULL,
    "DEFAULT$" IS NULL;
insert into "SYS"."CCOL$"
 values
    "OBJ#" = 33415,
    "CON#" = 2090,
    "COL#" = 1,
    "POS#" IS NULL,
```

```
    "INTCOL#" = 1,
    "SPARE1" = 0,
    "SPARE2" IS NULL,
    "SPARE3" IS NULL,
    "SPARE4" IS NULL,
    "SPARE5" IS NULL,
    "SPARE6" IS NULL;
insert into "SYS"."CDEF$"
 values
    "OBJ#" = 33415,
    "CON#" = 2090,
    "COLS" = 1,
    "TYPE#" = 7,
    "ROBJ#" IS NULL,
    "RCON#" IS NULL,
    "RRULES" IS NULL,
    "MATCH#" IS NULL,
    "REFACT" IS NULL,
    "ENABLED" = 1,
    "CONDLENGTH" = 24,
    "SPARE6" IS NULL,
    "INTCOLS" = 1,
    "MTIME" = TO_DATE('13-jan-2012 14:01:08', 'dd-mon-yyyy hh24:mi:ss'),
    "DEFER" = 12,
    "SPARE1" = 6,
    "SPARE2" IS NULL,
    "SPARE3" IS NULL,
    "SPARE4" IS NULL,
    "SPARE5" IS NULL,
    "CONDITION" = '"PRODUCT_ID" IS NOT NULL';
create table oe.product_tracking (product_id number not null,
  modified_time date,
  old_product_description varchar2(2000),
  old_list_price number(8,2),
  old_warranty_period interval year(2) to month);
update "SYS"."SEG$"
  set
    "TYPE#" = 5,
    "BLOCKS" = 5,
    "EXTENTS" = 1,
    "INIEXTS" = 5,
    "MINEXTS" = 1,
    "MAXEXTS" = 121,
    "EXTSIZE" = 5,
    "EXTPCT" = 50,
    "USER#" = 37,
    "LISTS" = 0,
    "GROUPS" = 0,
    "CACHEHINT" = 0,
    "HWMINCR" = 33415,
    "SPARE1" = 1024
  where
    "TS#" = 0 and
    "FILE#" = 1 and
    "BLOCK#" = 121034 and
    "TYPE#" = 3 and
    "BLOCKS" = 5 and
    "EXTENTS" = 1 and
    "INIEXTS" = 5 and
    "MINEXTS" = 1 and
    "MAXEXTS" = 121 and
    "EXTSIZE" = 5 and
    "EXTPCT" = 50 and
    "USER#" = 37 and
    "LISTS" = 0 and
    "GROUPS" = 0 and
    "BITMAPRANGES" = 0 and
```

```
    "CACHEHINT" = 0 and
    "SCANHINT" = 0 and
    "HWMINCR" = 33415 and
    "SPARE1" = 1024 and
    "SPARE2" IS NULL and
    ROWID = 'AAAAAIAABAAAdMOAAB';
insert into "SYS"."CON$"
 values
    "OWNER#" = 37,
    "NAME" = 'SYS_C002090',
    "CON#" = 2090,
    "SPARE1" IS NULL,
    "SPARE2" IS NULL,
    "SPARE3" IS NULL,
    "SPARE4" IS NULL,
    "SPARE5" IS NULL,
    "SPARE6" IS NULL;
commit;
```

6. LogMinerセッションを終了します。

```
EXECUTE DBMS_LOGMNR.END_LOGMNR();
```

親トピック: 分析するREDOログ・ファイルの明示的指定によるマイニングの例

24.12.1.5 例5: 内部ディクショナリでのDDL文の追跡

DBMS_LOGMNR.DDL_DICT_TRACKINGオプションを使用して、REDOログ・ファイルで発生したDDL文でLogMinerの内部ディクショナリを更新する方法を学習します。

1. データベースによって最後にアーカイブされたREDOログ・ファイルを判別します。

```
SELECT NAME, SEQUENCE# FROM V$ARCHIVED_LOG
   WHERE FIRST_TIME = (SELECT MAX(FIRST_TIME) FROM V$ARCHIVED_LOG);
NAME                                            SEQUENCE#
----------------------------------------------  --------------
/usr/oracle/data/db1arch_1_210_482701534.dbf    210
```

2. ディクショナリは複数のREDOログ・ファイルに含まれている場合があるため、データ・ディクショナリの先頭と終わりが含まれているREDOログ・ファイルを判別します。これを行うには、次のとおりV$ARCHIVED_LOGビューを問い合せます。

   a. データ・ディクショナリ抽出の終わりが含まれているREDOログを検索します。このREDOログ・ファイルは、分析するREDOログ・ファイルより前に作成されている必要がありますが、できるかぎり新しいものを使用します。

```
SELECT NAME, SEQUENCE#, DICTIONARY_BEGIN d_beg, DICTIONARY_END d_end
   FROM V$ARCHIVED_LOG
   WHERE SEQUENCE# = (SELECT MAX (SEQUENCE#) FROM V$ARCHIVED_LOG
   WHERE DICTIONARY_END = 'YES' and SEQUENCE# < 210);
NAME                                            SEQUENCE#    D_BEG  D_END
----------------------------------------------  ----------   -----  ------
/usr/oracle/data/db1arch_1_208_482701534.dbf    208          NO     YES
```

   b. データ・ディクショナリ抽出の先頭(前のSQL文で検出されたディクショナリの終わりに対応)が含まれている、REDOログ・ファイルを検索します。

```
SELECT NAME, SEQUENCE#, DICTIONARY_BEGIN d_beg, DICTIONARY_END d_end
   FROM V$ARCHIVED_LOG
   WHERE SEQUENCE# = (SELECT MAX (SEQUENCE#) FROM V$ARCHIVED_LOG
   WHERE DICTIONARY_BEGIN = 'YES' and SEQUENCE# <= 208);
NAME                                            SEQUENCE#    D_BEG  D_END
----------------------------------------------  ----------   -----  ------
/usr/oracle/data/db1arch_1_208_482701534.dbf    207          YES     NO
```

3. REDOログ・ファイルの完全なリストがあることを確認します。

REDOログ・ファイルで検出されたDDL文を正しく適用するために、マイニングするREDOログ・ファイルのリストにすべてのファイルが含まれていることを確認します。順序番号209に対応する欠落ログ・ファイルが、このリストに含まれている必要があります。次の問合せを発行して、リストに追加する必要があるREDOログ・ファイルの名前を判別します。

```
SELECT NAME FROM V$ARCHIVED_LOG
   WHERE SEQUENCE# >= 207 AND SEQUENCE# <= 210
   ORDER BY SEQUENCE# ASC;
NAME
---------------------------------------------
/usr/oracle/data/db1arch_1_207_482701534.dbf
/usr/oracle/data/db1arch_1_208_482701534.dbf
/usr/oracle/data/db1arch_1_209_482701534.dbf
/usr/oracle/data/db1arch_1_210_482701534.dbf
```

4. 分析するREDOログ・ファイルのリストを指定します。

ディクショナリの先頭と終わりが含まれているREDOログ・ファイル、マイニングするREDOログ・ファイル、および差異のないリストを作成するために必要なすべてのREDOログ・ファイルを含めます。REDOログ・ファイルは任意の順序で追加できます。

```
EXECUTE DBMS_LOGMNR.ADD_LOGFILE(-
   LOGFILENAME => '/usr/oracle/data/db1arch_1_210_482701534.dbf', -

       OPTIONS => DBMS_LOGMNR.NEW);

EXECUTE DBMS_LOGMNR.ADD_LOGFILE(-
   LOGFILENAME => '/usr/oracle/data/db1arch_1_209_482701534.dbf');
EXECUTE DBMS_LOGMNR.ADD_LOGFILE(-
   LOGFILENAME => '/usr/oracle/data/db1arch_1_208_482701534.dbf');
EXECUTE DBMS_LOGMNR.ADD_LOGFILE(-
   LOGFILENAME => '/usr/oracle/data/db1arch_1_207_482701534.dbf');
```

5. 使用するディクショナリおよびDDL_DICT_TRACKING、COMMITTED_DATA_ONLY、PRINT_PRETTY_SQLの各オプションを指定して、LogMinerを起動します。

```
EXECUTE DBMS_LOGMNR.START_LOGMNR(-
   OPTIONS => DBMS_LOGMNR.DICT_FROM_REDO_LOGS + -
              DBMS_LOGMNR.DDL_DICT_TRACKING + -
              DBMS_LOGMNR.COMMITTED_DATA_ONLY + -
              DBMS_LOGMNR.PRINT_PRETTY_SQL);
```

6. V$LOGMNR_CONTENTSビューを問い合せます。

返される行数を減らすため、SYSスキーマまたはSYSTEMスキーマで実行されたすべてのDML文を問合せから除外します。(この問合せでは、ディクショナリ抽出に関連したトランザクションを除外するタイムスタンプを指定します。)

この問合せは、正常に変換されたすべての再構築SQL文、およびトリガーを実行したためoe.product_tracking表に対して実行されたすべての挿入操作を返します。

```
SELECT USERNAME AS usr,(XIDUSN || '.' || XIDSLT || '.' || XIDSQN) as XID,
SQL_REDO FROM
   V$LOGMNR_CONTENTS
   WHERE SEG_OWNER IS NULL OR SEG_OWNER NOT IN ('SYS', 'SYSTEM') AND
   TIMESTAMP > '10-jan-2012 15:59:53';
USR             XID         SQL_REDO
-----------     --------    -----------------------------------
SYS             1.2.1594    set transaction read write;
SYS             1.2.1594    create table oe.product_tracking (product_id number
not null,
```

```
                              modified_time date,
                              old_list_price number(8,2),
                              old_warranty_period interval year(2) to month);
SYS             1.2.1594      commit;
SYS             1.18.1602     set transaction read write;
SYS             1.18.1602     create or replace trigger
oe.product_tracking_trigger
                              before update on oe.product_information
                              for each row
                              when (new.list_price <> old.list_price or
                                    new.warranty_period <> old.warranty_period)
                              declare
                              begin
                              insert into oe.product_tracking values
                                 (:old.product_id, sysdate,
                                  :old.list_price, :old.warranty_period);
                              end;
SYS             1.18.1602     commit;
OE              1.9.1598      update "OE"."PRODUCT_INFORMATION"
                                set
                                  "WARRANTY_PERIOD" = TO_YMINTERVAL('+08-00'),
                                  "LIST_PRICE" = 100
                                where
                                  "PRODUCT_ID" = 1729 and
                                  "WARRANTY_PERIOD" = TO_YMINTERVAL('+05-00') and
                                  "LIST_PRICE" = 80 and
                                  ROWID = 'AAAHTKAABAAAY9yAAA';
OE              1.9.1598      insert into "OE"."PRODUCT_TRACKING"
                                values
                                  "PRODUCT_ID" = 1729,
                                  "MODIFIED_TIME" = TO_DATE('13-jan-2012
16:07:03',
                                  'dd-mon-yyyy hh24:mi:ss'),
                                  "OLD_LIST_PRICE" = 80,
                                  "OLD_WARRANTY_PERIOD" = TO_YMINTERVAL('+05-
00');
OE              1.9.1598      update "OE"."PRODUCT_INFORMATION"
                                set
                                  "WARRANTY_PERIOD" = TO_YMINTERVAL('+08-00'),
                                  "LIST_PRICE" = 92
                                where
                                  "PRODUCT_ID" = 2340 and
                                  "WARRANTY_PERIOD" = TO_YMINTERVAL('+05-00') and
                                  "LIST_PRICE" = 72 and
                                  ROWID = 'AAAHTKAABAAAY9zAAA';
OE              1.9.1598      insert into "OE"."PRODUCT_TRACKING"
                                values
                                  "PRODUCT_ID" = 2340,
                                  "MODIFIED_TIME" = TO_DATE('13-jan-2012
16:07:07',
                                  'dd-mon-yyyy hh24:mi:ss'),
                                  "OLD_LIST_PRICE" = 72,
                                  "OLD_WARRANTY_PERIOD" = TO_YMINTERVAL('+05-
00');
OE              1.9.1598       commit;
```

7. LogMinerセッションを終了します。

```
EXECUTE DBMS_LOGMNR.END_LOGMNR();
```

親トピック: 分析するREDOログ・ファイルの明示的指定によるマイニングの例

24.12.1.6 例6: 時間範囲による出力のフィルタ処理

時間ごとに一連のREDOログをフィルタするには、時間範囲を指定してログ・ファイルを返す様々な方法について学習してくださ

い。

例4と例5では、問合せにタイムスタンプ・ベースの述語(timestamp > '10-jan-2012 15:59:53')を指定して行をフィルタする方法を確認しました。ただし、タイムスタンプ値に基づいてREDOレコードを除外するには、この例で示すとおり、DBMS_LOGMNR.START_LOGMNRプロシージャ・コールで時間範囲を指定する方法がより効率的です。

1. マイニングするREDOログ・ファイルのリストを作成します。

   指定した時刻以降に生成されたREDOログ・ファイルをマイニングするとします。次のプロシージャは、指定した時刻に基づいてREDOログ・ファイルのリストを作成します。その後のSQL EXECUTE文は、プロシージャをコールし、開始時刻を2012年1月13日の午後2時に指定します。

```
--
-- my_add_logfiles
-- Add all archived logs generated after a specified start_time.
--
CREATE OR REPLACE PROCEDURE my_add_logfiles (in_start_time  IN DATE) AS
  CURSOR  c_log IS
    SELECT NAME FROM V$ARCHIVED_LOG
      WHERE FIRST_TIME >= in_start_time;
count      pls_integer := 0;
my_option  pls_integer := DBMS_LOGMNR.NEW;
BEGIN
  FOR c_log_rec IN c_log
  LOOP
    DBMS_LOGMNR.ADD_LOGFILE(LOGFILENAME => c_log_rec.name,
                            OPTIONS => my_option);
    my_option := DBMS_LOGMNR.ADDFILE;
    DBMS_OUTPUT.PUT_LINE('Added logfile ' || c_log_rec.name);
  END LOOP;
END;
/
EXECUTE my_add_logfiles(in_start_time => '13-jan-2012 14:00:00');
```

2. Redoログ・ファイルのリストを表示するには、V$LOGMNR_LOGSビューを問い合せます。

   この例では、出力にREDOログ・ファイルのサイズを含めます。

```
SELECT FILENAME name, LOW_TIME start_time, FILESIZE bytes
    FROM V$LOGMNR_LOGS;
NAME                                 START_TIME            BYTES
------------------------------------ --------------------  ----------------
/usr/orcl/arch1_310_482932022.dbf    13-jan-2012 14:02:35  23683584
/usr/orcl/arch1_311_482932022.dbf    13-jan-2012 14:56:35  2564096
/usr/orcl/arch1_312_482932022.dbf    13-jan-2012 15:10:43  23683584
/usr/orcl/arch1_313_482932022.dbf    13-jan-2012 15:17:52  23683584
/usr/orcl/arch1_314_482932022.dbf    13-jan-2012 15:23:10  23683584
/usr/orcl/arch1_315_482932022.dbf    13-jan-2012 15:43:22  23683584
/usr/orcl/arch1_316_482932022.dbf    13-jan-2012 16:03:10  23683584
/usr/orcl/arch1_317_482932022.dbf    13-jan-2012 16:33:43  23683584
/usr/orcl/arch1_318_482932022.dbf    13-jan-2012 17:23:10  23683584
```

3. REDOログ・ファイルのリストを調整します。

   午後3時から4時の間に生成されたREDOログ・ファイルのみをマイニングする必要があるとします。

   問合せ条件(timestamp > '13-jan-2012 15:00:00' and timestamp < '13-jan-2012 16:00:00')を使用してこれを行います。ただし、問合せ条件はLogMinerによって返される各行で評価されるため、問合せ条件に基づく行のフィルタ処理は、内部マイニング・エンジンでは行われません。したがって、REDOログ・ファイルarch1_311_482932022.dbfからarch1_315_482932022.dbfまでの行のみを取得する場合でも、この問合せでは、LogMinerセッションに登録されたすべてのREDOログ・ファイルがマイニングされます。

また、問合せ条件を使用し、分析する時間範囲外のREDOログ・ファイルを手動で削除することもできますが、最も簡単な解決法は、DBMS_LOGMNR.START_LOGMNRプロシージャ・コールで分析する時間範囲を指定する方法です。

この方法では、REDOログ・ファイルのリストは変更されませんが、指定した時間範囲内のREDOログ・ファイルのみがLogMinerによってマイニングされます。

```
EXECUTE DBMS_LOGMNR.START_LOGMNR(-
   STARTTIME => '13-jan-2012 15:00:00', -
   ENDTIME   => '13-jan-2012 16:00:00', -
   OPTIONS   => DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG + -
                DBMS_LOGMNR.COMMITTED_DATA_ONLY + -
                DBMS_LOGMNR.PRINT_PRETTY_SQL);
```

4. V$LOGMNR_CONTENTSビューを問い合せます。

```
SELECT TIMESTAMP, (XIDUSN || '.' || XIDSLT || '.' || XIDSQN) AS XID,

 SQL_REDO FROM V$LOGMNR_CONTENTS WHERE SEG_OWNER = 'OE';
TIMESTAMP              XID          SQL_REDO
---------------------  -----------  --------------------------------
13-jan-2012 15:29:31   1.17.2376    update "OE"."PRODUCT_INFORMATION"
                                      set
                                        "WARRANTY_PERIOD" = TO_YMINTERVAL('+05-
00')
                                      where
                                        "PRODUCT_ID" = 3399 and
                                        "WARRANTY_PERIOD" = TO_YMINTERVAL('+02-
00') and
                                        ROWID = 'AAAHTKAABAAAY9TAAE';
13-jan-2012 15:29:34   1.17.2376      insert into "OE"."PRODUCT_TRACKING"
                                        values
                                        "PRODUCT_ID" = 3399,
                                        "MODIFIED_TIME" = TO_DATE('13-jan-2012
15:29:34',
                                        'dd-mon-yyyy hh24:mi:ss'),
                                        "OLD_LIST_PRICE" = 815,
                                        "OLD_WARRANTY_PERIOD" =
TO_YMINTERVAL('+02-00');
13-jan-2012 15:52:43   1.15.1756      update "OE"."PRODUCT_INFORMATION"
                                        set
                                          "WARRANTY_PERIOD" =
TO_YMINTERVAL('+05-00')
                                        where
                                          "PRODUCT_ID" = 1768 and
                                          "WARRANTY_PERIOD" =
TO_YMINTERVAL('+02-00') and
                                          ROWID = 'AAAHTKAABAAAY9UAAB';
13-jan-2012 15:52:43   1.15.1756      insert into "OE"."PRODUCT_TRACKING"
                                        values
                                        "PRODUCT_ID" = 1768,
                                        "MODIFIED_TIME" = TO_DATE('13-jan-2012
16:52:43',
                                        'dd-mon-yyyy hh24:mi:ss'),
                                        "OLD_LIST_PRICE" = 715,
                                        "OLD_WARRANTY_PERIOD" =
TO_YMINTERVAL('+02-00');
```

5. LogMinerセッションを終了します。

```
EXECUTE DBMS_LOGMNR.END_LOGMNR();
```

親トピック: 分析するREDOログ・ファイルの明示的指定によるマイニングの例

24.12.2 LogMinerのユース・ケースのシナリオ

LogMinerを使用してデータ・マイニング・タスクを実行する方法の一般的な例を確認してください。

- LogMinerを使用した特定のユーザーによる変更の追跡
  LogMinerを使用して、単一ユーザーによる特定の時間範囲内にデータベースに対して行われたすべての変更を識別する方法について説明します。
- LogMinerを使用した表アクセス統計の計算
  LogMinerを使用して指定の時間範囲にわたる表アクセス統計を計算する方法について説明します。

親トピック: LogMinerの使用例

24.12.2.1 LogMinerを使用した特定のユーザーによる変更の追跡

LogMinerを使用して、単一のユーザーによって特定の時間範囲内にデータベースに対して行われたすべての変更を特定する方法を学習します。

ユーザーjoedevoが特定の時間範囲内にデータベースに加えたすべての変更を確認するとします。このタスクを実行するには、LogMinerを使用できます。

1. データベースへの接続。
2. LogMinerディクショナリ・ファイルを作成します。

   LogMinerを使用してjoedevoのデータを分析するには、joedevoが使用する表に対して表定義の変更が行われる前にLogMinerディクショナリ・ファイルを作成するか、LogMinerの起動時にオンライン・カタログを使用する必要があります。この例では、REDOログ・ファイルに抽出されたLogMinerディクショナリを使用します。

3. REDOログ・ファイルを追加します。

   joedevoがデータベースに対して変更を行ったとします。分析するREDOログ・ファイルの名前は、次のように指定できます。

```
EXECUTE DBMS_LOGMNR.ADD_LOGFILE( -
   LOGFILENAME => 'log1orc1.ora', -
   OPTIONS => DBMS_LOGMNR.NEW);
```

   必要に応じて、次のようにREDOログ・ファイルを追加します。

```
EXECUTE DBMS_LOGMNR.ADD_LOGFILE( -
   LOGFILENAME => 'log2orc1.ora', -
   OPTIONS => DBMS_LOGMNR.ADDFILE);
```

4. LogMinerを起動し、検索を指定した時間範囲に制限します。

```
EXECUTE DBMS_LOGMNR.START_LOGMNR( -
   DICTFILENAME => 'orcldict.ora', -
   STARTTIME => TO_DATE('01-Jan-1998 08:30:00','DD-MON-YYYY HH:MI:SS'), -
   ENDTIME => TO_DATE('01-Jan-1998 08:45:00', 'DD-MON-YYYY HH:MI:SS'));
```

5. V$LOGMNR_CONTENTSビューを問い合せます。

   この時点で、V$LOGMNR_CONTENTSビューを問い合せることが可能です。ユーザーjoedevoがsalary表に対して行ったすべての変更を検索します。次のSELECT文を実行します。

```
SELECT SQL_REDO, SQL_UNDO FROM V$LOGMNR_CONTENTS
   WHERE USERNAME = 'joedevo' AND SEG_NAME = 'salary';
```

SQL_REDO列とSQL_UNDO列の両方に対して、2つの行が返されます(データ表示の書式は画面では異なる場合があります)。joedevoが2つの操作を要求したことがわかります。joedevoは、古い給与を削除した後、新しい昇給後の給与を挿入しました。これらは、この操作を元に戻すために必要なデータです。

```
SQL_REDO                                SQL_UNDO
--------                                --------
delete from SALARY                      insert into SALARY(NAME, EMPNO, SAL)
where EMPNO = 12345                      values ('JOEDEVO', 12345, 500)
and NAME='JOEDEVO'
and SAL=500;
insert into SALARY(NAME, EMPNO, SAL)    delete from SALARY
values('JOEDEVO',12345, 2500)           where EMPNO = 12345
                                        and NAME = 'JOEDEVO'
2 rows selected                         and SAL = 2500;
```

6. LogMinerセッションを終了します。

   LogMinerセッションを正常に終了するには、DBMS_LOGMNR.END_LOGMNRプロシージャを使用します。

```
DBMS_LOGMNR.END_LOGMNR( );
```

親トピック: LogMinerのユースケースのシナリオ

24.12.2.2 LogMinerを使用した表アクセス統計の計算

LogMinerを使用して、指定の時間範囲にわたる表アクセス統計を計算する方法を学習します。

この例では、ダイレクト・マーケティング・データベースを管理し、1月の2週間の収益でのカスタマ・コンタクトの有効性を確認するとします。この場合、すでにLogMinerディクショナリを作成し、検索するREDOログ・ファイルを追加していることを前提としています。これらのコンタクトを識別するには、次のように1月の時間範囲でログを検索します。

1. LogMinerを起動し、時間範囲を指定します。

```
EXECUTE DBMS_LOGMNR.START_LOGMNR( -
   STARTTIME => TO_DATE('07-Jan-2012 08:30:00','DD-MON-YYYY HH:MI:SS'), -
   ENDTIME => TO_DATE('21-Jan-2012 08:45:00','DD-MON-YYYY HH:MI:SS'), -
   DICTFILENAME => '/usr/local/dict.ora');
```

2. 次の

```
V$LOGMNR_CONTENTS
```

   ビューを問い合せて、次の例に示すとおり、指定した時間範囲で変更された表を判別します。(この問合せでは、フィルタ処理で除外されるのは、従来どおり名前に次の

```
$
```

   を含むシステム表です。)

```
SELECT SEG_OWNER, SEG_NAME, COUNT(*) AS Hits FROM
   V$LOGMNR_CONTENTS WHERE SEG_NAME NOT LIKE '%$' GROUP BY
   SEG_OWNER, SEG_NAME ORDER BY Hits DESC;
```

3. 次のデータが表示されます。(画面の書式は異なる場合があります。)

```
SEG_OWNER          SEG_NAME          Hits
---------          --------          ----
CUST               ACCOUNT            384
UNIV               EXECDONOR          325
UNIV               DONOR              234
UNIV               MEGADONOR           32
```

```
HR                    EMPLOYEES            12
SYS                   DONOR                12
```

次の

```
Hits
```

列の値は、問合せで指定した2週間の間に、指定した表に対して挿入、削除、更新の操作が行われた回数を示しています。この例では

```
cust.account
```

表が、指定された2週間の期間中に最も修正され、

```
hr.employees
```

および

```
sys.donor
```

表は、同じ期間内に最も少なく変更されました。

4. LogMinerセッションを終了します。

次の

```
DBMS_LOGMNR.END_LOGMNR
```

プロシージャを使用して、LogMinerセッションを適切に終了します。

```
DBMS_LOGMNR.END_LOGMNR( );
```

親トピック: LogMinerのユースケースのシナリオ

24.13 サポートされるデータ型、記憶域属性、およびデータベースとREDOログ・ファイルのバージョン

データ型と記憶域属性のサポートに関する情報およびサポートされるデータベースとREDOログ・ファイルのリリースを説明します。

- サポートされるデータ型と表記憶域属性
  サポートされるデータ型と表記憶域属性を説明します。
- サポートされないデータ型と表記憶域属性
  サポートされないデータ型と表記憶域属性を説明します。
- サポートされるデータベースとREDOログ・ファイルのバージョン
  サポートされるデータベースのリリースとREDOログ・ファイルのバージョンを説明します。
- SecureFiles LOBに関する考慮点
  SecureFiles LOBは、データベース互換性が11.2以上に設定されている場合にサポートされます。

親トピック: REDOログ・ファイル分析のためのLogMinerの使用

24.13.1 サポートされるデータ型と表記憶域属性

サポートされるデータ型と表記憶域属性を説明します。

ノート:

Oracle Database 12c リリース 1 (12.1)から、VARCHAR2、NVARCHAR2 および RAW データ型の最大サイズが 32KB に増加しました(COMPATIBLE 初期化パラメータが 12.0 以上に設定され、MAX_STRING_SIZE 初期化パラメータが EXTENDED に設定されている場合)。

LogMiner では、サプリメンタル・ロギングの目的で 32KB の列が LOB として扱われます。

32KB の列は、ALWAYS サプリメンタル・ロギング・グループの一部にはできません。

LogMinerは、次のデータ型と表記憶域属性をサポートします。互換性要件で説明するとおり、一部のデータ型は特定のリリースでのみサポートされます。

データ型

- BINARY_DOUBLE
- BINARY_FLOAT
- BLOB
- CHAR
- CLOBおよびNCLOB
- DATE
- INTERVALYEARTOMONTH
- INTERVALDAYTOSECOND
- SecureFilesとして格納されるLOB (11.2以上の互換性で動作するデータベースが必要)
- LONG
- LONG RAW
- NCHAR
- NUMBER
- NVARCHAR2
- VARRAYとして格納されるオブジェクト
- オブジェクト(コレクションなしの単純なネストされたADT)

  (SDO_GEOMETRYやORDIMAGEなどOracle提供の型を含む)オブジェクトがサポートされるようにするには、12.0.0.0以上のREDO互換性が設定されたOracle Database 12c リリース1 (12.1)以上が実行されている必要があります。XMLデータに関連する操作用のSQL_REDO列の内容は、有効なSQLまたはPL/SQLではありません。
- Oracle Text
- RAW
- TIMESTAMP
- TIMESTAMP WITH TIMEZONE
- TIMESTAMP WITH LOCAL TIMEZONE
- VARCHARおよびVARCHAR2
- XDB

- 次のプライマリ・データベース互換性要件を仮定した場合は、すべてのストレージ・モデルのXMLTypeデータ。
    - CLOB形式として格納されるXMLTypeでは、11.0以上の互換性設定で動作するデータベースが必要です(Oracle Database 12cリリース1 (12.1)からはCLOBとして格納されるXMLTypeが非推奨になったため)。
    - オブジェクト・リレーショナル形式またはバイナリXMLで格納されるXMLTypeには、11.2.0.3以上のREDO互換性が設定されたOracle Database 11gリリース2 (11.2.0.3)以上でデータベースを実行している必要があります。XMLデータに関連する操作用のSQL_REDO列の内容は、有効なSQLまたはPL/SQLではありません。

表記憶域タイプ

- クラスタ表(索引クラスタおよびヒープ・クラスタを含む)
- 索引構成表(IOT) (オーバーフロー・セグメントを含むパーティション化および非パーティション化)
- ヒープ構成表(パーティション化および非パーティション化)
- 高度な行圧縮および基本表圧縮。これらのオプションはいずれも、データベースの互換性設定が11.1.0以上である必要があります。
- SecureFilesとして格納されるLOB列を含む表(データベース互換性が11.2以上に設定されている場合)
- ハイブリッド列圧縮を使用する表(データベース互換性が11.2.0.2以上に設定されている場合)

関連項目:

- ハイブリッド列圧縮の詳細は、『Oracle Database概要』を参照してください。

- 互換性要件
  LogMinerによる一部のデータ型および表記憶域属性のサポートには、データベース互換性要件があります。

親トピック: サポートされるデータ型、記憶域属性、およびデータベースとREDOログ・ファイルのバージョン

24.13.1.1 互換性要件

LogMinerによる一部のデータ型および表記憶域属性のサポートには、データベース互換性要件があります。

具体的には次のとおりです。

- マルチバイトのCLOBサポートには、10.1以上の互換性で動作するデータベースが必要です。
- LOBおよびオーバーフローなしのIOTサポートには、10.1以上の互換性で動作するデータベースが必要です。
- LOBおよびオーバーフローありのIOTサポートには、10.2以上の互換性で動作するデータベースが必要です。
- TDEおよびTSEサポートには、11.1以上の互換性で動作するデータベースが必要です。
- 基本圧縮および高度な行圧縮では、11.1以上の互換性で動作するデータベースが必要です。
- ハイブリッド列圧縮のサポートは、基礎となるストレージ・システムに依存し、11.2以上の互換性で動作するデータベースが必要です。

関連項目:

- ハイブリッド列圧縮の詳細は、『Oracle Database概要』を参照してください。

親トピック: サポートされるデータ型と表記憶域属性

24.13.2 サポートされないデータ型と表記憶域属性

サポートされないデータ型と表記憶域属性を説明します。

LogMinerは、次のデータ型と表記憶域属性をサポートしません。これらのサポートされていないデータ型のいずれかを使用する列が表に含まれる場合、表全体がLogMinerによって無視されます。

- BFILE
- ネストされた表
- ネストされた表のあるオブジェクト
- アイデンティティ列を含む表
- 時間的な有効性列
- PKREF列
- PKOID列
- ネストした表属性およびスタンドアロンのネストした表の列

親トピック: サポートされるデータ型、記憶域属性、およびデータベースとREDOログ・ファイルのバージョン

24.13.3 サポートされるデータベースとREDOログ・ファイルのバージョン

サポートされるデータベースのリリースとREDOログ・ファイルのバージョンを説明します。

LogMinerは、リリース8.1以上のデータベースでのみ実行されますが、リリース8.0のデータベースからのREDOログ・ファイルの分析に使用することができます。ただし、LogMinerでREDOログ・ファイルから取得することができる情報は、使用中のデータベースのリリースではなく、ログのバージョンに依存します。たとえば、サプリメンタル・ロギングが有効な場合は、Oracle9iのREDOログ・ファイルを拡張して追加情報を取得できます。これによって、LogMinerの機能を最大限使用できます。旧リリースのOracleで作成されたREDOログ・ファイルには、追加データが含まれていないため、LogMinerによってサポートされる操作およびデータ型が制限される場合があります。

関連項目:

一般的なLogMinerセッションのステップおよびサプリメンタル・ロギング

親トピック: サポートされるデータ型、記憶域属性、およびデータベースとREDOログ・ファイルのバージョン

24.13.4 SecureFiles LOBに関する考慮点

SecureFiles LOBは、データベース互換性が11.2以上に設定されている場合にサポートされます。

SecureFiles LOB列には、SQL_REDO列のみを入力できます。SQL_UNDO列は入力されません。

透過的データ暗号化(TDE)およびデータ圧縮は、プライマリ・データベースのSecureFiles LOB列で有効化できます。

SecureFiles LOB列の重複除外は完全にサポートされます。フラグメント操作はサポートされません。

LogMinerは、サポートされない操作によって生成されたREDOを検出すると、OPERATION列がUNSUPPORTEDに設定された行を生成します。これらのREDOレコードには、SQL_REDOまたはSQL_UNDOは生成されません。

親トピック: サポートされるデータ型、記憶域属性、およびデータベースとREDOログ・ファイルのバージョン

24.12 一般的なLogMinerセッションのステップ

LogMinerを使用してデータを抽出およびマイニングできる一般的な方法について学習します。

- LogMinerセッションの実行方法の理解
  オンプレミスでのLogMinerセッションの実行方法とOracle Autonomous Cloud Platform Servicesは似ていますが、異なるユーザーが必要です。
- 一般的なLogMinerセッション・タスク1: サプリメンタル・ロギングの有効化
  REDOログ・ファイルとともにLogMinerを使用できるようにするには、サプリメンタル・ロギングを有効にする必要があります。
- 一般的なLogMinerセッション・タスク2: LogMinerディクショナリの抽出
  LogMinerを使用するには、LogMinerにデータベース・ディクショナリを提供するオプションを選択する必要があります。
- 一般的なLogMinerセッション・タスク3: 分析用のREDOログ・ファイルの指定
  LogMinerを起動する前に`DBMS_LOGMNR_ADD_LOGFILE`を使用して、分析するREDOログ・ファイルを指定する必要があります。
- LogMinerの起動
  LogMinerの起動方法、およびREDOログ・ファイル、フィルタ基準、その他のセッション特性の分析に使用できるオプションを確認してください。
- V$LOGMNR_CONTENTSの問合せ
  LogMinerを起動した後、Oracle Databaseの`V$LOGMNR_CONTENTS`ビューを問い合せることができます。
- 一般的なLogMinerセッション・タスク6: LogMinerセッションの終了
  LogMinerセッションの終了。

親トピック: REDOログ・ファイル分析のためのLogMinerの使用

24.12.1 LogMinerセッションの実行方法の理解

オンプレミスとOracle Autonomous Cloud Platform Services LogMinerセッションは似ていますが、異なるユーザーが必要です。

従来のLogMinerセッションでは、`CDB$ROOT`でLogMinerを実行する場合、`SYS`が所有するPL/SQLパッケージを使用してLogMinerを実行します。LogMinerを使用するには、LogMinerで使用するユーザー・アカウントに関する要件があります。

オンプレミスOracle DatabaseでLogMinerを実行する場合、`CDB$ROOT`取得抽出を作成して複数のPDBからデータを同時に取得するか、Oracle GoldenGateを使用して複数の個別のPDBログをマイニングし、それぞれが単一のPDBからデータを取得できます。ただし、`CDB$ROOT`にアクセスできないOracle Autonomous Database Cloud Platform Servicesの場合は、PDBごとの取得方法を使用する必要があります。このモードでは、事前定義済の権限セットを持つローカル・ユーザーを、ログを確認するソースPDBにプロビジョニングします。すべてのLogMiner処理は、このPDBのみに制限されます。

オンプレミスPDBでは、リソースで許可されている数のセッションを開始できます。ただし、クラウド構成では、`CDB$ROOT`で多くの同時セッションを開始できますが、LogMiner PL/SQLパッケージを使用して各PDBに対して開始できるセッションは1つのみです。

`CDB$ROOT`でLogMinerを実行するには、PL/SQLパッケージ`DBMS_LOGMNR.ADD_LOGFILE`を使用し、ログ・ファイルを明示的に追加します。また、オンライン・カタログを使用せずにLogMinerディクショナリを抽出することを選択する場合は、`DBMS_LOGMNR_D`パッケージも使用できます。

個々のPDBでLogMinerを実行する場合、プロシージャは若干異なります。`DBMS_LOGMNR.ADD_LOGFILE`を使用するか

わりに、PDBのログ・ファイルを確認する期間を指定します。問い合せるログのSCN値をstartScnで指定し、endScnまたはstartTimeを選択した場合はendTimeで指定します。次に、DBMS_LOGMNR.START_LOGMNRを使用してLogMinerを起動します。DBMS_LOGMNR.START_LOGMNRは、分析するREDOログを自動的に追加します。

DBMS_LOGMNRパッケージには、LogMinerの初期化および実行に使用するプロシージャ(REDOログ・ファイルの名前、フィルタ基準、セッション特性を指定するためのインタフェースなど)が含まれます。DBMS_LOGMNR_Dパッケージは、現在のデータベースのデータベース・ディクショナリ表を問い合せてLogMinerディクショナリ・ファイルを作成します。

個々のPDBに対してLogMinerを実行するための要件

LogMinerを実行して個々のPDBを問い合せるには、プロシージャ・コールDBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGEを使用して、必要な権限をローカル・ユーザーにプロビジョニングする必要があります。また、GGADMIN権限を持つユーザーは、PDBごとの取得抽出を実行できます。

また、個々のPDBでは、マイニングするアーカイブ・ログを指定しません。かわりに、マイニングするPDBに接続し、dbms_logmnr_d.STORE_IN_REDO_LOGSを実行します。たとえば:

```
SQL> execute dbms_logmnr_d.build(option=>dbms_logmnr_d.STORE_IN_REDO_LOGS);
```

その後、PDBに接続してSCNを識別し、dbms_logmnr.start_logmnrを実行して、表示するPDBログ履歴の開始点システム変更番号(SCN)およびエンド・ポイントSCN (選択した場合)をログ・ファイルに問い合せられます。マイニングは、V$LOGMNR_CONTENTSビューに対する従来のLogMiner問合せと同様に、その時点で続行されます。ただし、使用できるのは、接続しているPDBに対して生成されたREDOのみです

> ノート:
>
> Extract および LogMiner プロセスの実行中に PDB を停止すると、これらのプロセスは他のアクティブ・セッションと同様に終了します。PDB を再オープンしたら、Extract マイニングの再起動を通常どおり続行する必要があります。PDB を切断する場合、特別なアクションは必要ありません。ただし、PDB を切断した後に接続すると、PDB に以前存在していたすべての LogMiner および Capture セッションが削除されます。

SYSとして接続していない場合に従来のLogMinerセッションを実行するための要件

オンプレミス・ログ・マイニングでは、LogMiner PL/SQLパッケージはSYSスキーマによって所有されます。したがって、ユーザーSYSとして接続していない場合は、次の条件が必要になります。

- コールにSYSを含める必要があります。たとえば:

  ```
  EXECUTE SYS.DBMS_LOGMNR.END_LOGMNR;
  ```

- EXECUTE_CATALOG_ROLEロールを付与されている必要があります。

関連トピック

- LogMinerを使用した個々のPDBの問合せ
- DBMS_LOGMNR
- PL/SQLパッケージの概要

親トピック: 一般的なLogMinerセッションのステップ

24.12.2 一般的なLogMinerセッション・タスク1: 追加ロギングの有効化

REDOログ・ファイルとともにLogMinerを使用できるようにするには、サプリメンタル・ロギングを有効にする必要があります。

REDOベースのアプリケーションでは、追加の列をREDOログ・ファイルに記録する必要がある場合があります。これらの追加の列を記録するプロセスは、**サプリメンタル・ロギング**と呼ばれますデフォルトでは、Oracle Databaseはサプリメンタル・ロギングが有効になっていません。少なくともLogMinerを使用するには、最小限のサプリメンタル・ロギングを有効にする必要があります。

例24-2 最小サプリメンタル・ロギングの有効化

サプリメンタル・ロギングを有効にするには、次の文を入力します。

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;
```

関連トピック

- サプリメンタル・ロギング

親トピック: 一般的なLogMinerセッションのステップ

24.12.3 一般的なLogMinerセッション・タスク2: LogMinerディクショナリの抽出

LogMinerを使用するには、LogMinerにデータベース・ディクショナリを提供するオプションを選択する必要があります。

次のいずれかのオプションを選択します。

- LogMiner起動時に`DICT_FROM_ONLINE_CATALOG`オプションでオンライン・カタログの使用を指定。
- REDOログ・ファイルにデータベース・ディクショナリ情報を抽出。
- フラット・ファイルにデータベース・ディクショナリ情報を抽出。

関連トピック

- オンライン・カタログの使用
- REDOログ・ファイルへのLogMinerディクショナリの抽出
- フラット・ファイルへのLogMinerディクショナリの抽出

親トピック: 一般的なLogMinerセッションのステップ

24.12.4 一般的なLogMinerセッション・タスク3: 分析用のREDOログ・ファイルの指定

LogMinerを起動する前に`DBMS_LOGMNR_ADD_LOGFILE`を使用して、分析するREDOログ・ファイルを指定する必要があります。

オンプレミスの`CDB$ROOT`のログを問い合せるには、LogMinerを起動する前に、分析するREDOログ・ファイルを指定する必要があります。ログ・ファイルを指定するには、次のステップで示すとおり、`DBMS_LOGMNR.ADD_LOGFILE`プロシージャを実行します。REDOログ・ファイルは、任意の順序で追加および削除できます。

ノート:

個々の PDB のログを問い合せるには、若干異なる手順を使用します。PDB に接続した後、DBA_LOGMNR_DICTIONARY_BUILDLOG を問い合せて START_SCN 値を特定し、DBMS_LOGMNR.START_LOGMNR で LogMiner を起動して、確認するログの SCN 値を指定します。DBMS_LOGMNR.START_LOGMNR は、分析する REDO ログを自動的に追加します。例は、「LogMiner を使用した個々の PDB の問合せ」を参照してください。

1. データベースをマウントした状態またはマウントしていない状態で、SQL*Plusを使用してOracle Databaseインスタンスを起動します。たとえば、SQLプロンプトでSTARTUP文を入力します。

```
STARTUP
```

2. REDOログ・ファイルのリストを作成します。PL/SQLプロシージャDBMS_LOGMNR.ADD_LOGFILEのNEWオプションを指定して、新規リストの開始であることを指定します。たとえば、REDOログ・ファイル/oracle/logs/log1.fを指定するには、次のように入力します。

```
EXECUTE DBMS_LOGMNR.ADD_LOGFILE( -
   LOGFILENAME => '/oracle/logs/log1.f', -
   OPTIONS => DBMS_LOGMNR.NEW);
```

3. 必要に応じて、PL/SQLプロシージャDBMS_LOGMNR.ADD_LOGFILEのADDFILEオプションを指定し、さらにREDOログ・ファイルを追加します。たとえば、REDOログ・ファイル/oracle/logs/log2.fを追加するには、次のように入力します。

```
EXECUTE DBMS_LOGMNR.ADD_LOGFILE( -
   LOGFILENAME => '/oracle/logs/log2.f', -
   OPTIONS => DBMS_LOGMNR.ADDFILE);
```

   REDOログ・ファイルを追加する場合、OPTIONSパラメータの指定はオプションです。たとえば、次のように入力できます。

```
EXECUTE DBMS_LOGMNR.ADD_LOGFILE( -
   LOGFILENAME=>'/oracle/logs/log2.f');
```

4. 必要に応じて、PL/SQLプロシージャDBMS_LOGMNR.REMOVE_LOGFILEを使用してREDOログ・ファイルを削除します。たとえば、REDOログ・ファイル/oracle/logs/log2.fを削除するには、次のように入力します。

```
EXECUTE DBMS_LOGMNR.REMOVE_LOGFILE( -
   LOGFILENAME => '/oracle/logs/log2.f');
```

関連トピック

- LogMinerを使用した個々のPDBの問合せ

親トピック: 一般的なLogMinerセッションのステップ

24.12.5 LogMinerの起動

LogMinerの起動方法、およびREDOログ・ファイル、フィルタ基準、その他のセッション特性の分析に使用できるオプションを確認してください。

LogMinerディクショナリ・ファイルを作成し、分析するREDOログ・ファイルを指定した後、LogMinerを起動し、Oracle Databaseトランザクションを分析できます。

1. LogMinerを起動するには、DBMS_LOGMNR.START_LOGMNRプロシージャを実行します。

   LogMinerディクショナリ・オプションを指定することをお薦めします。ディクショナリ・オプションを指定していない場合、LogMinerでは、内部オブジェクトの識別子とデータ型を、オブジェクト名と外部データ書式に変換できません。この結

果、LogMinerから内部オブジェクトIDが返され、データはバイナリ・データとして表されます。また、ディクショナリがない場合、MINE_VALUE機能およびCOLUMN_PRESENT機能は使用できません。

フラット・ファイルのLogMinerディクショナリの名前を指定する場合は、そのディクショナリ・ファイルに完全修飾されたファイル名を指定する必要があります。たとえば、/oracle/database/dictionary.oraを使用してLogMinerを起動するには、次の文を発行します。

```
EXECUTE DBMS_LOGMNR.START_LOGMNR( -
   DICTFILENAME =>'/oracle/database/dictionary.ora');
```

フラット・ファイルのディクショナリ名を指定しない場合は、OPTIONSパラメータを使用して、DICT_FROM_REDO_LOGSオプションまたはDICT_FROM_ONLINE_CATALOGオプションのいずれかを指定します。

DICT_FROM_REDO_LOGSを指定した場合、LogMinerでは、DBMS_LOGMNR.ADD_LOGFILEプロシージャで指定したREDOログ・ファイルに、ディクショナリが含まれていると認識されます。ディクショナリが含まれているREDOログ・ファイルを判別するには、V$ARCHIVED_LOGビューを使用します。このタスクの例を確認するには、「REDOログ・ファイルへのLogMinerディクショナリの抽出」を参照してください。

> ノート:
>
> LogMiner の起動後に REDO ログ・ファイルを追加した場合は、LogMiner を再起動する必要があります。LogMiner は、DBMS_LOGMNR.START_LOGMNR に対する前回のコールに含まれていたオプションを保持しないため、使用するオプションを再度指定する必要があります。ただし、DBMS_LOGMNR.START_LOGMNR の現在のコールでディクショナリを指定しない場合、LogMiner は、以前のコールのディクショナリ仕様を保持します。

2. 必要に応じて、問合せをフィルタ処理またはフォーマットしたり、OPTIONSパラメータを使用してLogMinerセッションの特性を追加指定できます。たとえば、オンライン・カタログをLogMinerディクショナリとして使用して、V$LOGMNR_CONTENTSビューにコミット済トランザクションのみを表示できます。次のように入力します。

```
EXECUTE DBMS_LOGMNR.START_LOGMNR(OPTIONS => -
   DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG + -
   DBMS_LOGMNR.COMMITTED_DATA_ONLY);
```

DBMS_LOGMNR.START_LOGMNRプロシージャは、毎回異なるオプションを指定して、複数回実行できます。たとえば、V$LOGMNR_CONTENTSの問合せから必要な結果を取得できなかった場合、他のオプションを指定してLogMinerを再起動できます。LogMinerディクショナリを再指定する必要がないかぎり、DBMS_LOGMNR.START_LOGMNRに対する前回のコールでREDOログ・ファイルがすでに追加されている場合は、再度追加する必要はありません。

関連トピック

- REDOログ・ファイルへのLogMinerディクショナリの抽出
- オンライン・カタログの使用

親トピック: 一般的なLogMinerセッションのステップ

24.12.6 V$LOGMNR_CONTENTSの問合せ

LogMinerを起動した後、Oracle DatabaseのV$LOGMNR_CONTENTSビューを問い合せることができます。

たとえば:

```
SELECT (XIDUSN || '.' || XIDSLT || '.' || XIDSQN) AS XID,
   USERNAME, SQL_REDO FROM V$LOGMNR_CONTENTS WHERE USERNAME != 'SYS'
   AND SEG_OWNER IS NULL OR SEG_OWNER NOT IN ('SYS', 'SYSTEM');
XID         USERNAME  SQL_REDO
1.15.3045   RON       set transaction read write;
1.15.3045   RON       insert into "HR"."JOBS"("JOB_ID","JOB_TITLE",
                      "MIN_SALARY","MAX_SALARY") values ('9782',
                      'HR_ENTRY',NULL,NULL);
1.18.3046   JANE      set transaction read write;
1.18.3046   JANE      insert into "OE"."CUSTOMERS"("CUSTOMER_ID",
                      "CUST_FIRST_NAME","CUST_LAST_NAME",
                      "CUST_ADDRESS","PHONE_NUMBERS","NLS_LANGUAGE",
                      "NLS_TERRITORY","CREDIT_LIMIT","CUST_EMAIL",
                      "ACCOUNT_MGR_ID") values ('9839','Edgar',
                      'Cummings',NULL,NULL,NULL,NULL,
                       NULL,NULL,NULL);
1.9.3041    RAJIV      set transaction read write;
1.9.3041    RAJIV      insert into "OE"."CUSTOMERS"("CUSTOMER_ID",
                       "CUST_FIRST_NAME","CUST_LAST_NAME","CUST_ADDRESS",
                       "PHONE_NUMBERS","NLS_LANGUAGE","NLS_TERRITORY",
                       "CREDIT_LIMIT","CUST_EMAIL","ACCOUNT_MGR_ID")
                       values ('9499','Rodney','Emerson',NULL,NULL,NULL,NULL,
                       NULL,NULL,NULL);
1.15.3045    RON       commit;
1.8.3054     RON       set transaction read write;
1.8.3054     RON       insert into "HR"."JOBS"("JOB_ID","JOB_TITLE",
                       "MIN_SALARY","MAX_SALARY") values ('9566',
                       'FI_ENTRY',NULL,NULL);
1.18.3046    JANE      commit;
1.11.3047    JANE      set transaction read write;
1.11.3047    JANE      insert into "OE"."CUSTOMERS"("CUSTOMER_ID",
                       "CUST_FIRST_NAME","CUST_LAST_NAME",
                       "CUST_ADDRESS","PHONE_NUMBERS","NLS_LANGUAGE",
                       "NLS_TERRITORY","CREDIT_LIMIT","CUST_EMAIL",
                       "ACCOUNT_MGR_ID") values ('8933','Ronald',
                       'Frost',NULL,NULL,NULL,NULL,NULL,NULL,NULL);
1.11.3047    JANE      commit;
1.8.3054     RON       commit;
```

別の例を確認するには、「V$LOGMNR_CONTENTSに返されるデータのフィルタ処理および書式設定」を参照してください。

関連トピック

- V$LOGMNR_CONTENTSに返されるデータのフィルタ処理および書式設定

親トピック: 一般的なLogMinerセッションのステップ

24.12.7 一般的なLogMinerセッション・タスク6: LogMinerセッションの終了

LogMinerセッションの終了。

LogMinerセッションを正常に終了するには、次に示すとおり、PL/SQLプロシージャDBMS_LOGMNR.END_LOGMNRを使用します。

```
EXECUTE DBMS_LOGMNR.END_LOGMNR;
```

このプロシージャを実行すると、すべてのREDOログ・ファイルがクローズされ、LogMinerによって割り当てられたすべてのデータベースとシステム・リソースを解放できます。

このプロシージャを実行しない場合、LogMinerは、コールを実行したOracleセッションが終了するまで、割り当てられたすべてのリソースを保持します。DDL_DICT_TRACKINGオプションまたはDICT_FROM_REDO_LOGSオプションのいずれかを使用した場合は、LogMinerセッションの終了に、このプロシージャを使用する必要があります。

親トピック: 一般的なLogMinerセッションのステップ

25 メタデータAPIの使用

DBMS_METADATA APIの使用。

DBMS_METADATA APIを使用すると、次の操作を行うことができます。

- オブジェクトのメタデータのXMLとしての取得
- SQL DDLへの変換を含む様々な方法でのXMLの変換
- 取得で抽出したオブジェクトを再作成するためのXMLの送信

DBMS_METADATA_DIFF APIを使用すると、データベース間でオブジェクトを比較し、同じ型のオブジェクトのうち時間の経過によってメタデータが変化したものを識別できます。

- DBMS_METADATA APIを使用する理由
  DBMS_METADATA APIを使用すると、メタデータを抽出するために独自のコードを記述して管理する必要がなくなります。
- DBMS_METADATA APIの概要
  DBMS_METADATA APIの概要。
- オブジェクトのメタデータを取得するためのDBMS_METADATA APIの使用
  DBMS_METADATA APIの取得インタフェースでは、取得するオブジェクトの種類を指定できます。
- 取得したオブジェクトを再作成するためのDBMS_METADATA APIの使用
  オブジェクトのメタデータをフェッチする場合は、これを使用して異なるデータベースまたはスキーマでオブジェクトを再作成することをお薦めします。
- 異なるオブジェクト型のコレクションを取得するためのDBMS_METADATA APIの使用
  異なるオブジェクト型のコレクションの取得例および説明。
- オブジェクトのメタデータを比較するためのDBMS_METADATA_DIFF APIの使用
  DBMS_METADATAおよびDBMS_METADATA_DIFFの取得インタフェース、比較インタフェースおよび送信インタフェースの説明および使用例。この例では、2つの表のメタデータをフェッチして比較し、1つの表を他方と同じ内容に変更するALTER文を生成します。
- DBMS_METADATA APIのプログラム・インタフェースに関するパフォーマンスのヒント
  DBMS_METADATA APIのプログラム・インタフェース使用時にパフォーマンスを向上する方法について説明します。
- DBMS_METADATA APIの使用例
  DBMS_METADATA APIの使用方法の例。
- DBMS_METADATAプロシージャの要約
  DBMS_METADATA APIによって提供されるプロシージャを簡単に説明します。
- DBMS_METADATA_DIFFプロシージャの要約
  DBMS_METADATA_DIFF APIによって提供されるプロシージャおよびファンクションを簡単に説明します。

親トピック: その他のユーティリティ

25.1 DBMS_METADATA APIを使用する理由

DBMS_METADATA APIを使用すると、メタデータを抽出するために独自のコードを記述して管理する必要がなくなります。

独自のコードをOracle Databaseで開発して、ディクショナリからメタデータを抽出したり、メタデータを操作(列の追加、列のデータ型の変更など)したり、メタデータをDDLに変換して同じまたは別のデータベースでオブジェクトを再作成できるようにする場

合、メンテナンスが問題になります。ディクショナリの新機能をサポートするよう、そのコードを更新された状態に保つことは困難です。

Oracle Databaseでは、ディクショナリ・メタデータの抽出、操作および再作成に関する集中的な機能が提供されます。また、Oracle Databaseでは、すべてのディクショナリ・オブジェクトを最新のレベルでサポートしています。

DBMS_METADATA APIは、記述および管理するカスタム・コードの量を大幅に削減しますが、通常のデータベース手順の変更はありません。DBMS_METADATA APIはデータ・ディクショナリ・ビューと同様に、データベースのインストール時にcatproc.sqlを実行し、SQLスクリプトを実行することでインストールできます。DBMS_METADATAをインストールすると、制限モードであっても、インスタンスが動作中はいつでも使用可能です。

DBMS_METADATA APIを使用してデータベースのリリースを変更する場合、ソースコードを変更する必要はありません。DBMS_METADATA APIを使用すると、異なるOracle Databaseリリース間でコードに上位互換性を持たせることができます。1つのリリースで取得したXML文書を、同じまたはそれ以上のリリースの送信インタフェースで処理できます。たとえば、Oracle Database 10gリリース2 (10.2)データベースで取得したXML文書は、Oracle Database 12cに送信できます。

親トピック: メタデータAPIの使用

25.2 DBMS_METADATA APIの概要

DBMS_METADATA APIの概要。

DBMS_METADATA APIにおいては、データベース内のすべてのエンティティは、オブジェクト型に属するオブジェクトとして形成されています。たとえば、scott.emp表はオブジェクトであり、オブジェクト型はTABLEです。オブジェクトのメタデータをフェッチする場合は、オブジェクト型を指定する必要があります。

オブジェクト型の特定のオブジェクトまたは一連のオブジェクトをフェッチするには、フィルタを指定します。各オブジェクト型に対して異なるフィルタが定義されます。たとえば、TABLEオブジェクト型に対して定義されている2つのフィルタはSCHEMAおよびNAMEです。これにより、たとえば、スキーマがscottで名前がempである表を必要としていることを表現できます。

DBMS_METADATA APIは、XML(Extensible Markup Language)およびXSLT(Extensible Stylesheet Language Transformation)を使用します。XMLは解析や変換が容易な汎用形式であるため、DBMS_METADATA APIは、オブジェクト・メタデータをXMLとして表現します。DBMS_METADATA APIは、XSLTを使用してXML文書を他のXML文書またはSQL DDLのいずれかに変換します。

DBMS_METADATA APIを使用して、メタデータをフェッチするとき(または再送信するとき)に、1つ以上の変換(XSLTスクリプト)をXMLに適用するよう指定できます。APIには、XML文書をSQL作成DDLに変換する名前付きDDL以外にも、いくつかの事前定義変換があります。

次に変換パラメータを使用して、変換の条件を指定できます。また、オブジェクトのメタデータの特定の属性にアクセスするために、オプションの解析項目も指定できます。

これらのオプションの詳細と実装例については、次の項を参照してください。

- オブジェクトのメタデータを取得するためのDBMS_METADATA APIの使用
- 取得したオブジェクトを再作成するためのDBMS_METADATA APIの使用
- 異なるオブジェクト型のコレクションを取得するためのDBMS_METADATA APIの使用

ビューを使用した有効なDBMS_METADATAオプションの確認

次のビューを使用して、各オブジェクト型変換、各変換のパラメータおよびその解析項目に使用できるDBMS_METADATA変換を確認できます。

- DBMS_METADATA_TRANSFORMS - DBMS_METADATAパッケージで使用されるすべての有効なOracle提供の変換を示します。
- DBMS_METADATA_TRANSFORM_PARAMS - 各変換について有効な変換パラメータを示します。
- DBMS_METADATA_PARSE_ITEMS- 有効な解析項目を示します。

たとえば、INDEXオブジェクトに使用できる変換を確認するとします。次の問合せは、INDEXオブジェクトに対して有効な変換、必要な入力型および結果の出力型を返します。

```
SQL> SELECT transform, output_type, input_type, description
2 FROM dbms_metadata_transforms
3 WHERE object_type='INDEX';
TRANSFORM  OUTPUT_TYP INPUT_TYPE           DESCRIPTION
---------- ---------- -------------------- ----------------------------------------
----------------------------
ALTERXML   ALTER_XML  SXML difference doc  Generate ALTER_XML from SXML difference
document
SXMLDDL    DDL        SXML                 Convert SXML to DDL
MODIFY     XML        XML                  Modify XML document according to transform
parameters
SXML       SXML       XML                  Convert XML to SXML
DDL        DDL        XML                  Convert XML to SQL to create the object
ALTERDDL   ALTER_DDL  ALTER_XML            Convert ALTER_XML to ALTER_DDL
MODIFYSXML SXML       SXML                 Modify SXML document
```

次に、DDL変換に対して有効な変換パラメータを確認するとします。

```
SQL> SELECT param, datatype, default_val, description
2 FROM dbms_metadata_transform_params
3 WHERE object_type='INDEX' and transform='DDL'
4 ORDER BY param;
PARAM                      DATATYPE   DEFAULT_VA  DESCRIPTION
-------------------------- ---------- ----------  -----------------------------------
----------------------------------
INCLUDE_PARTITIONS         TEXT                   Include generated interval and list
partitions in DDL
                                                       transformation
INDEX_COMPRESSION_CLAUSE   TEXT       ""          Text of user-specified index
compression clause
PARTITIONING               BOOLEAN    TRUE        Include partitioning clauses in
transformation
PARTITION_NAME             TEXT       ""          Name of partition selected for the
transformation
PCTSPACE                   NUMBER     ""          Percentage by which space allocation
is to be modified
SEGMENT_ATTRIBUTES         BOOLEAN    TRUE        Include segment attribute clauses
(physical attributes, storage
                                                       attribues, tablespace, logging)
in transformation
STORAGE                    BOOLEAN    TRUE        Include storage clauses in
transformation
SUBPARTITION_NAME          TEXT       ""          Name of subpartition selected for
the transformation
TABLESPACE                 BOOLEAN    TRUE        Include tablespace clauses in
transformation
```

INDEXオブジェクト型に関する特定のメタデータを返す次の問合せを実行することもできます。

```
SQL> SELECT parse_item, description
2 FROM dbms_metadata_parse_items
3 WHERE object_type='INDEX' and convert='Y';
PARSE_ITEM           DESCRIPTION
-------------------- ------------------------------------------------------------
OBJECT_TYPE          Object type
```

```
TABLESPACE            Object tablespace (default tablespace for partitioned objects)
BASE_OBJECT_SCHEMA    Schema of the base object
SCHEMA                Object schema, if any
NAME                  Object name
BASE_OBJECT_NAME      Name of the base object
BASE_OBJECT_TYPE      Object type of the base object
SYSTEM_GENERATED      Y = system-generated object; N = not system-generated
```

親トピック: メタデータAPIの使用

25.3 オブジェクトのメタデータを取得するためのDBMS_METADATA APIの使用

DBMS_METADATA APIの取得インタフェースでは、取得するオブジェクトの種類を指定できます。

種類は、特定のオブジェクト型(表、索引、プロシージャなど)または論理単位を形成する異なるオブジェクト型のコレクション(データベース・エクスポート、スキーマ・エクスポートなど)のいずれかです。デフォルトでは、フェッチされたメタデータがXML文書に返されます。

> ノート:
>
> スキーマにないオブジェクトにアクセスするには SELECT_CATALOG_ROLE ロールが必要です。ただし、ロールは多くの PL/SQL オブジェクト(ストアド・プロシージャ、ファンクション、定義者権限 API)内で無効とされます。そのため、他のスキーマのオブジェクト(一般的には、SELECT_CATALOG_ROLE ロールを必要とするすべてのオブジェクト)内にアクセスする PL/SQL プログラムを記述する場合は、実行者権限 API にコードを入力する必要があります。

プログラム・インタフェースを使用して、簡単な参照を実行したり、アプリケーションを開発できます。システム・メタデータの非定型の問合せを行う場合は、ブラウザ・インタフェースを使用します。アプリケーションの一部として、ディクショナリ・メタデータを抽出する場合は、プログラム・インタフェースを使用します。その場合、同じことをするために現在使用しているSQLスクリプトおよびカスタマイズしたコードのかわりに、DBMS_METADATA APIで提供されているプロシージャを使用することもできます。

- 基本的なメタデータ取得の通常ステップ
  メタデータを取得する場合は、DBMS_METADATA PL/SQL APIを使用します。
- 複数のオブジェクトの取得
  複数オブジェクトの取得例および説明。
- 変換の条件指定
  変換パラメータを使用して、追加する変換の条件を指定できます。
- 特定のメタデータ属性へのアクセス
  特定のメタデータ属性へのアクセス例および説明。

親トピック: メタデータAPIの使用

25.3.1 基本的なメタデータ取得の通常ステップ

メタデータを取得する場合は、DBMS_METADATA PL/SQL APIを使用します。

次に、プログラム・インタフェースおよびブラウザ・インタフェースの例を示します。

DBMS_METADATAプログラム・インタフェースの例では、DBMS_METADATAプログラム・インタフェースを使用して1つの表のメタデータを取得する基本的な方法を説明します。この例では、get_table_mdというファンクションを作成する

DBMS_METADATAプログラムを作成します。ファンクションは、1つの表のメタデータを返します。

DBMS_METADATAブラウザ・インタフェースの例では、ブラウザ・インタフェースを使用して同じ結果を得る方法を説明します。

例25-1 データ取得のためのDBMS_METADATAプログラム・インタフェースの使用

1. hrスキーマ内の1つの表(timecards)のメタデータを返すファンクションget_table_mdを作成する

   DBMS_METADATAプログラムを作成します。プログラムの内容は次のようになります。(この例ではプログラムにmetadata_program.sqlという名前を付けます)。

```
CREATE OR REPLACE FUNCTION get_table_md RETURN CLOB IS
-- Define local variables.
h NUMBER; --handle returned by OPEN
th NUMBER; -- handle returned by ADD_TRANSFORM
doc CLOB;
BEGIN
-- Specify the object type.
h := DBMS_METADATA.OPEN('TABLE');
-- Use filters to specify the particular object desired.
DBMS_METADATA.SET_FILTER(h,'SCHEMA','HR');
DBMS_METADATA.SET_FILTER(h,'NAME','TIMECARDS');
 -- Request that the metadata be transformed into creation DDL.
th := DBMS_METADATA.ADD_TRANSFORM(h,'DDL');
 -- Fetch the object.
doc := DBMS_METADATA.FETCH_CLOB(h);
 -- Release resources.
DBMS_METADATA.CLOSE(h);
RETURN doc;
END;
/
```

2. ユーザーhrとして接続します。

3. プログラムを実行してget_table_mdファンクションを作成します。

   SQL> @metadata_program

4. 新しく作成したget_table_mdファンクションを選択操作で使用します。完全で中断のない出力を生成するには、問合せの実行前に、次のとおり、PAGESIZEを0(ゼロ)にし、またLONGを大きい数に設定します。

```
SQL> SET PAGESIZE 0
SQL> SET LONG 1000000
SQL> SELECT get_table_md FROM dual;
```

5. hrスキーマのtimecards表のメタデータを示す出力は、次のようになります。

```
  CREATE TABLE "HR"."TIMECARDS"
   (    "EMPLOYEE_ID" NUMBER(6,0),
        "WEEK" NUMBER(2,0),
        "JOB_ID" VARCHAR2(10),
        "HOURS_WORKED" NUMBER(4,2),
         FOREIGN KEY ("EMPLOYEE_ID")
          REFERENCES "HR"."EMPLOYEES" ("EMPLOYEE_ID") ENABLE
   ) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
  TABLESPACE "EXAMPLE"
```

例25-2 データ取得のためのDBMS_METADATAブラウザ・インタフェースの使用

```
SQL> SET PAGESIZE 0
SQL> SET LONG 1000000
SQL> SELECT DBMS_METADATA.GET_DDL('TABLE','TIMECARDS','HR') FROM dual;
```

この問合せの結果は、DBMS_METADATAプログラム・インタフェースの例のステップ5で示された結果と同じです。

親トピック: オブジェクトのメタデータを取得するためのDBMS_METADATA APIの使用

25.3.2 複数のオブジェクトの取得

複数オブジェクトの取得例および説明。

前の例「データ取得のためのDBMS_METADATAプログラム・インタフェースの使用」では、オブジェクトが1つしか存在しないことがわかっていたため、FETCH_CLOBプロシージャは1回のみコールされました。scottスキーマのすべての表などの複数のオブジェクトも取得できます。それを行うには、次の構成メンバーを使用する必要があります。

```
LOOP
  doc := DBMS_METADATA.FETCH_CLOB(h);
  --
  -- When there are no more objects to be retrieved, FETCH_CLOB returns NULL.
  --
  EXIT WHEN doc IS NULL;
END LOOP;
```

次の例に、構成メンバーの使用および複数のオブジェクトの取得を示します。この例では、ユーザーscottとして接続します。パスワードは、tigerです。

例25-3 複数のオブジェクトの取得

1. 次のように、my_metadataという名前の表と、get_tables_mdという名前のプロシージャを作成します。すべてのオブジェクトを返すことはできないため、それらを表に格納して最後に問合せを実行します。

```
DROP TABLE my_metadata;
CREATE TABLE my_metadata (md clob);
CREATE OR REPLACE PROCEDURE get_tables_md IS
-- Define local variables
h       NUMBER;          -- handle returned by 'OPEN'
th      NUMBER;          -- handle returned by 'ADD_TRANSFORM'
doc     CLOB;            -- metadata is returned in a CLOB
BEGIN
 -- Specify the object type.
 h := DBMS_METADATA.OPEN('TABLE');
 -- Use filters to specify the schema.
 DBMS_METADATA.SET_FILTER(h,'SCHEMA','SCOTT');
 -- Request that the metadata be transformed into creation DDL.
 th := DBMS_METADATA.ADD_TRANSFORM(h,'DDL');
 -- Fetch the objects.
 LOOP
   doc := DBMS_METADATA.FETCH_CLOB(h);
  -- When there are no more objects to be retrieved, FETCH_CLOB returns NULL.
   EXIT WHEN doc IS NULL;
   -- Store the metadata in a table.
   INSERT INTO my_metadata(md) VALUES (doc);
   COMMIT;
 END LOOP;

 -- Release resources.
 DBMS_METADATA.CLOSE(h);
END;
/
```

2. 次のようにプロシージャを実行します。

```
EXECUTE get_tables_md;
```

3. my_metadata表を問い合せて、取得された内容を確認します。

```
SET LONG 9000000
SET PAGES 0
SELECT * FROM my_metadata;
```

親トピック: オブジェクトのメタデータを取得するためのDBMS_METADATA APIの使用

25.3.3 変換の条件指定

変換パラメータを使用して、追加する変換の条件を指定できます。

SET_TRANSFORM_PARAMプロシージャを使用します。たとえば、TABLEオブジェクトに対してDDL変換を追加した場合、SEGMENT_ATTRIBUTES変換パラメータを指定して、物理、ストレージ、ロギングなどのセグメント属性をDDLに表示しないようにすることができます。デフォルトでは、セグメント属性がDDLに表示されます。

例25-4に、SET_TRANSFORM_PARAMプロシージャの使用方法を示します。

例25-4 変換の条件指定

1. 次のように、get_table_mdという名前のファンクションを作成します。

```
CREATE OR REPLACE FUNCTION get_table_md RETURN CLOB IS
 -- Define local variables.
 h    NUMBER;   -- handle returned by 'OPEN'
 th   NUMBER;   -- handle returned by 'ADD_TRANSFORM'
 doc  CLOB;
BEGIN
 -- Specify the object type.
 h := DBMS_METADATA.OPEN('TABLE');
 -- Use filters to specify the particular object desired.
 DBMS_METADATA.SET_FILTER(h,'SCHEMA','HR');
 DBMS_METADATA.SET_FILTER(h,'NAME','TIMECARDS');
 -- Request that the metadata be transformed into creation DDL.
 th := dbms_metadata.add_transform(h,'DDL');
 -- Specify that segment attributes are not to be returned.
 -- Note that this call uses the TRANSFORM handle, not the OPEN handle.
DBMS_METADATA.SET_TRANSFORM_PARAM(th,'SEGMENT_ATTRIBUTES',false);
 -- Fetch the object.
 doc := DBMS_METADATA.FETCH_CLOB(h);
 -- Release resources.
 DBMS_METADATA.CLOSE(h);
 RETURN doc;
END;
/
```

2. 次の問合せを実行します。

```
SQL> SELECT get_table_md FROM dual;
```

出力は、次のようになります。

```
  CREATE TABLE "HR"."TIMECARDS"
   (    "EMPLOYEE_ID" NUMBER(6,0),
        "WEEK" NUMBER(2,0),
        "JOB_ID" VARCHAR2(10),
        "HOURS_WORKED" NUMBER(4,2),
         FOREIGN KEY ("EMPLOYEE_ID")
          REFERENCES "HR"."EMPLOYEES" ("EMPLOYEE_ID") ENABLE
   )
```

ここまでの例ではDDL変換という1つの変換を使用しました。DBMS_METADATA APIでは、最初の出力が2番目の入力になり、2番目の出力が3番目の入力になるというように、複数の変換を指定することもできます。

Oracleには、XML文書を変更するMODIFYと呼ばれる変換があります。スキーマ名や表領域名の変更などが行えます。これには、再マップ・パラメータおよびSET_REMAP_PARAMプロシージャを使用します。

例25-5に、SET_REMAP_PARAMプロシージャの使用例を示します。まずMODIFY変換を追加し、再マップ・パラメータを指定して、スキーマ名をhrからscottに変更します。次にDDL変換を追加します。MODIFY変換の出力は、DDL変換への入力となるXML文書です。最終的な結果は、timecards表の作成DDLであり、hrスキーマのすべてのインスタンスはscottに変更されます。

例25-5 XML文書の変更

1. 次のように、remap_schemaという名前のファンクションを作成します。

```
CREATE OR REPLACE FUNCTION remap_schema RETURN CLOB IS
-- Define local variables.
h NUMBER; --handle returned by OPEN
th NUMBER; -- handle returned by ADD_TRANSFORM
doc CLOB;
BEGIN
-- Specify the object type.
h := DBMS_METADATA.OPEN('TABLE');
-- Use filters to specify the particular object desired.
DBMS_METADATA.SET_FILTER(h,'SCHEMA','HR');
DBMS_METADATA.SET_FILTER(h,'NAME','TIMECARDS');
-- Request that the schema name be modified.
th := DBMS_METADATA.ADD_TRANSFORM(h,'MODIFY');
DBMS_METADATA.SET_REMAP_PARAM(th,'REMAP_SCHEMA','HR','SCOTT');
-- Request that the metadata be transformed into creation DDL.
th := DBMS_METADATA.ADD_TRANSFORM(h,'DDL');
-- Specify that segment attributes are not to be returned.
DBMS_METADATA.SET_TRANSFORM_PARAM(th,'SEGMENT_ATTRIBUTES',false);
-- Fetch the object.
doc := DBMS_METADATA.FETCH_CLOB(h);
-- Release resources.
DBMS_METADATA.CLOSE(h);
RETURN doc;
END;
/
```

2. 次の問合せを実行します。

```
SELECT remap_schema FROM dual;
```

出力は、次のようになります。

```
  CREATE TABLE "SCOTT"."TIMECARDS"
   (     "EMPLOYEE_ID" NUMBER(6,0),
         "WEEK" NUMBER(2,0),
         "JOB_ID" VARCHAR2(10),
         "HOURS_WORKED" NUMBER(4,2),
          FOREIGN KEY ("EMPLOYEE_ID")
           REFERENCES "SCOTT"."EMPLOYEES" ("EMPLOYEE_ID") ENABLE
   )
```

XSLTをよく理解しているユーザーであれば、独自の変換を追加してXMLを処理できます。

親トピック: オブジェクトのメタデータを取得するためのDBMS_METADATA APIの使用

25.3.4 特定のメタデータ属性へのアクセス

特定のメタデータ属性へのアクセス例および説明。

名前やスキーマなどのオブジェクトのメタデータの特定の属性へアクセスすることが必要な場合もあります。属性の情報は返され

たメタデータを解析して取得できますが、DBMS_METADATA APIには、別の機能もあります。メタデータから解析され、別のデータ構造に返される特定の属性を解析項目に指定できます。これを行うには、SET_PARSE_ITEMプロシージャを使用します。

例25-6では、スキーマ内のすべての表をフェッチします。各表では、名前を取得するために解析項目を使用します。次に表のすべての索引を取得するために名前を使用します。次の例に、sys.ku$_ddlsオブジェクトのメタデータを返すFETCH_DDLファンクションの使用を示します。

この例では、いくつかの表および索引を含むスキーマに接続していることとします。また、my_metadataという名前の表も作成します。

例25-6 特定のメタデータ属性にアクセスするための解析項目の使用

1. 次のように、my_metadataという名前の表と、get_tables_and_indexesという名前のプロシージャを作成します。

```
DROP TABLE my_metadata;
CREATE TABLE my_metadata (
  object_type   VARCHAR2(30),
  name          VARCHAR2(30),
  md            CLOB);
CREATE OR REPLACE PROCEDURE get_tables_and_indexes IS
-- Define local variables.
h1      NUMBER;         -- handle returned by OPEN for tables
h2      NUMBER;         -- handle returned by OPEN for indexes
th1     NUMBER;         -- handle returned by ADD_TRANSFORM for tables
th2     NUMBER;         -- handle returned by ADD_TRANSFORM for indexes
doc     sys.ku$_ddls;   -- metadata is returned in sys.ku$_ddls,
                        --  a nested table of sys.ku$_ddl objects
ddl     CLOB;           -- creation DDL for an object
pi      sys.ku$_parsed_items;   -- parse items are returned in this object
                                -- which is contained in sys.ku$_ddl
objname VARCHAR2(30);   -- the parsed object name
idxddls sys.ku$_ddls;   -- metadata is returned in sys.ku$_ddls,
                        --  a nested table of sys.ku$_ddl objects
idxname VARCHAR2(30);   -- the parsed index name
BEGIN
 -- This procedure has an outer loop that fetches tables,
 -- and an inner loop that fetches indexes.

 -- Specify the object type: TABLE.
 h1 := DBMS_METADATA.OPEN('TABLE');

 -- Request that the table name be returned as a parse item.
 DBMS_METADATA.SET_PARSE_ITEM(h1,'NAME');

 -- Request that the metadata be transformed into creation DDL.
 th1 := DBMS_METADATA.ADD_TRANSFORM(h1,'DDL');

 -- Specify that segment attributes are not to be returned.
 DBMS_METADATA.SET_TRANSFORM_PARAM(th1,'SEGMENT_ATTRIBUTES',false);

 -- Set up the outer loop: fetch the TABLE objects.
 LOOP
   doc := dbms_metadata.fetch_ddl(h1);

-- When there are no more objects to be retrieved, FETCH_DDL returns NULL.
   EXIT WHEN doc IS NULL;

-- Loop through the rows of the ku$_ddls nested table.
   FOR i IN doc.FIRST..doc.LAST LOOP
     ddl := doc(i).ddlText;
     pi := doc(i).parsedItems;
```

```
      -- Loop through the returned parse items.
      IF pi IS NOT NULL AND pi.COUNT > 0 THEN
        FOR j IN pi.FIRST..pi.LAST LOOP
          IF pi(j).item='NAME' THEN
            objname := pi(j).value;
          END IF;
        END LOOP;
      END IF;
      -- Insert information about this object into our table.
      INSERT INTO my_metadata(object_type, name, md)
        VALUES ('TABLE',objname,ddl);
      COMMIT;
    END LOOP;

    -- Now fetch indexes using the parsed table name as
    --  a BASE_OBJECT_NAME filter.

    -- Specify the object type.
    h2 := DBMS_METADATA.OPEN('INDEX');

    -- The base object is the table retrieved in the outer loop.
    DBMS_METADATA.SET_FILTER(h2,'BASE_OBJECT_NAME',objname);

    -- Exclude system-generated indexes.
    DBMS_METADATA.SET_FILTER(h2,'SYSTEM_GENERATED',false);

    -- Request that the index name be returned as a parse item.
    DBMS_METADATA.SET_PARSE_ITEM(h2,'NAME');

    -- Request that the metadata be transformed into creation DDL.
    th2 := DBMS_METADATA.ADD_TRANSFORM(h2,'DDL');

    -- Specify that segment attributes are not to be returned.
    DBMS_METADATA.SET_TRANSFORM_PARAM(th2,'SEGMENT_ATTRIBUTES',false);


    LOOP
     idxddls := dbms_metadata.fetch_ddl(h2);

     -- When there are no more objects to  be retrieved, FETCH_DDL returns NULL.
     EXIT WHEN idxddls IS NULL;

       FOR i in idxddls.FIRST..idxddls.LAST LOOP
         ddl := idxddls(i).ddlText;
         pi  := idxddls(i).parsedItems;
         -- Loop through the returned parse items.
         IF pi IS NOT NULL AND pi.COUNT > 0 THEN
           FOR j IN pi.FIRST..pi.LAST LOOP
             IF pi(j).item='NAME' THEN
               idxname := pi(j).value;
             END IF;
           END LOOP;
          END IF;

          -- Store the metadata in our table.
           INSERT INTO my_metadata(object_type, name, md)
             VALUES ('INDEX',idxname,ddl);
          COMMIT;
        END LOOP;  -- for loop
   END LOOP;
   DBMS_METADATA.CLOSE(h2);
  END LOOP;
  DBMS_METADATA.CLOSE(h1);
END;
/
```

2. 次のようにプロシージャを実行します。

```
EXECUTE get_tables_and_indexes;
```

3. 次の問合せを実行して、取得された内容を確認します。

```
SET LONG 9000000
SET PAGES 0
SELECT * FROM my_metadata;
```

親トピック: オブジェクトのメタデータを取得するためのDBMS_METADATA APIの使用

25.4 取得したオブジェクトを再作成するためのDBMS_METADATA APIの使用

オブジェクトに対しメタデータをフェッチする場合、そのメタデータを使用して、オブジェクトを異なるデータベースまたはスキーマで再作成することがあります。

メタデータをフェッチする際に、再マップの実行を決定していない場合もあります。また、この決定を先送りにしたい場合があります。再マップを先送りにするには、メタデータをXMLとしてフェッチし、ファイルまたは表に格納します。後で送信インタフェースを使用しオブジェクトを再作成します。

送信インタフェースは取得インタフェースに類似した構成です。送信インタフェースには、作成するオブジェクトのオブジェクト型を指定するOPENWプロシージャが存在します。変換の指定、パラメータの変換および項目の解析ができます。CONVERTファンクションをコールしてXMLからDDLに変換する、またはPUTファンクションをコールしてXMLをDDLに変換し、DDLを送信してオブジェクトを作成できます。

例25-7は、1つのスキーマの表のXMLをフェッチし、次に送信インタフェースを使用して別のスキーマの表を再作成します。

例25-7 取得したオブジェクトを再作成するための送信インタフェースの使用

1. 次のように、権限を持つユーザーとして接続します。

```
CONNECT system
Enter password: password
```

2. 実行者権限パッケージを作成してプロシージャを保持します。これは、別のスキーマ内のオブジェクトにアクセスするには、SELECT_CATALOG_ROLEロールが必要なためです。定義者権限PL/SQLオブジェクト(プロシージャ、ファンクションなど)では、ロールは使用できません。

```
CREATE OR REPLACE PACKAGE example_pkg AUTHID current_user IS
  PROCEDURE move_table(
         table_name  in VARCHAR2,
         from_schema in VARCHAR2,
         to_schema   in VARCHAR2 );
END example_pkg;
/
CREATE OR REPLACE PACKAGE BODY example_pkg IS
PROCEDURE move_table(
         table_name  in VARCHAR2,
         from_schema in VARCHAR2,
         to_schema   in VARCHAR2 ) IS
-- Define local variables.
h1       NUMBER;          -- handle returned by OPEN
h2       NUMBER;          -- handle returned by OPENW
th1      NUMBER;          -- handle returned by ADD_TRANSFORM for MODIFY
th2      NUMBER;          -- handle returned by ADD_TRANSFORM for DDL
xml      CLOB;            -- XML document
```

```
errs    sys.ku$_SubmitResults := sys.ku$_SubmitResults();
err     sys.ku$_SubmitResult;
result  BOOLEAN;
BEGIN
-- Specify the object type.
h1 := DBMS_METADATA.OPEN('TABLE');
-- Use filters to specify the name and schema of the table.
DBMS_METADATA.SET_FILTER(h1,'NAME',table_name);
DBMS_METADATA.SET_FILTER(h1,'SCHEMA',from_schema);
-- Fetch the XML.
xml := DBMS_METADATA.FETCH_CLOB(h1);
IF xml IS NULL THEN
    DBMS_OUTPUT.PUT_LINE('Table ' || from_schema || '.' || table_name
|| ' not found');
    RETURN;
  END IF;
-- Release resources.
DBMS_METADATA.CLOSE(h1);
-- Use the submit interface to re-create the object in another schema.
-- Specify the object type using OPENW (instead of OPEN).
h2 := DBMS_METADATA.OPENW('TABLE');
-- First, add the MODIFY transform.
th1 := DBMS_METADATA.ADD_TRANSFORM(h2,'MODIFY');
-- Specify the desired modification: remap the schema name.
DBMS_METADATA.SET_REMAP_PARAM(th1,'REMAP_SCHEMA',from_schema,to_schema);
-- Now add the DDL transform so that the modified XML can be
--  transformed into creation DDL.
th2 := DBMS_METADATA.ADD_TRANSFORM(h2,'DDL');
-- Call PUT to re-create the object.
result := DBMS_METADATA.PUT(h2,xml,0,errs);
DBMS_METADATA.CLOSE(h2);
  IF NOT result THEN
    -- Process the error information.
    FOR i IN errs.FIRST..errs.LAST LOOP
      err := errs(i);
      FOR j IN err.errorLines.FIRST..err.errorLines.LAST LOOP
        dbms_output.put_line(err.errorLines(j).errorText);
      END LOOP;
    END LOOP;
  END IF;
END;
END example_pkg;
/
```

3. ここで、次のように、スキーマSCOTTでmy_exampleという名前の表を作成します。

```
CONNECT scott
Enter password:
-- The password is tiger.
DROP TABLE my_example;
CREATE TABLE my_example (a NUMBER, b VARCHAR2(30));
CONNECT system
Enter password: password
SET LONG 9000000
SET PAGESIZE 0
SET SERVEROUTPUT ON SIZE 100000
```

4. 次のように、my_example表をSYSTEMスキーマにコピーします。

```
DROP TABLE my_example;
EXECUTE example_pkg.move_table('MY_EXAMPLE','SCOTT','SYSTEM');
```

5. 次の問合せを実行して、正しく動作したことを確認します。

```
SELECT DBMS_METADATA.GET_DDL('TABLE','MY_EXAMPLE') FROM dual;
```

25.5 異なるオブジェクト型のコレクションを取得するための DBMS_METADATA APIの使用

異なるオブジェクト型のコレクションの取得例および説明。

異なる型のオブジェクトであるにもかかわらず、1つの論理単位を形成するオブジェクトのコレクションを取得する必要がある場合があります。たとえば、データベースまたはスキーマのすべてのオブジェクト、または表および依存する索引、制約、権限付与、監査などすべてを取得する必要があるとします。DBMS_METADATA APIでは、こうした取得を可能にするために、複数の異種オブジェクト型を提供しています。異種オブジェクト型は、オブジェクト型の順序付けられた集合です。

Oracleでは、次の異種オブジェクト型を提供しています。

- TABLE_EXPORT - 表およびその依存オブジェクト
- SCHEMA_EXPORT - スキーマおよびその内容
- DATABASE_EXPORT - データベース内のオブジェクト

これらのオブジェクト型は、データ・ポンプ・エクスポート・ユーティリティで使用するために開発されましたが、ユーザー独自のアプリケーションでも使用できます。

これらの型では、ブラウザ・インタフェースまたは送信インタフェースではなく、プログラム取得インタフェース(OPEN, FETCH, CLOSE)のみが使用できます。

同種型と同様に、異種オブジェクト型に対してもフィルタを指定できます。たとえば、TABLE_EXPORTに対してSCHEMAおよびNAMEフィルタを指定でき、SCHEMA_EXPORTに対してSCHEMAフィルタを指定できます。

例25-8に、scottスキーマ内のオブジェクト型を取得する方法を示します。ユーザーscottとして接続します。パスワードは、tigerです。

例25-8 異種オブジェクト型の取得方法

1. 取得したオブジェクトを格納するための表を作成します。

```
DROP TABLE my_metadata;
CREATE TABLE my_metadata (md CLOB);
CREATE OR REPLACE PROCEDURE get_schema_md IS
-- Define local variables.
h        NUMBER;          -- handle returned by OPEN
th       NUMBER;          -- handle returned by ADD_TRANSFORM
doc      CLOB;            -- metadata is returned in a CLOB
BEGIN
-- Specify the object type.
 h := DBMS_METADATA.OPEN('SCHEMA_EXPORT');
 -- Use filters to specify the schema.
 DBMS_METADATA.SET_FILTER(h,'SCHEMA','SCOTT');
 -- Request that the metadata be transformed into creation DDL.
 th := DBMS_METADATA.ADD_TRANSFORM(h,'DDL');
 -- Fetch the objects.
 LOOP
   doc := DBMS_METADATA.FETCH_CLOB(h);
   -- When there are no more objects to be retrieved, FETCH_CLOB returns NULL.
   EXIT WHEN doc IS NULL;
   -- Store the metadata in the table.
   INSERT INTO my_metadata(md) VALUES (doc);
   COMMIT;
 END LOOP;
```

```
 -- Release resources.
 DBMS_METADATA.CLOSE(h);
END;
/
```

2. 次のようにプロシージャを実行します。

```
EXECUTE get_schema_md;
```

3. 次の問合せを実行して、取得された内容を確認します。

```
SET LONG 9000000
SET PAGESIZE 0
SELECT * FROM my_metadata;
```

この例では、オブジェクトは、オブジェクト型による順序で返されます。たとえば、すべての表が返され、次に表のすべての権限が返され、次に表のすべての索引が返されます。順序は一般的に有効な作成順序となります。このため、返された順序でオブジェクトを取得し、送信インタフェースを使用して別のスキーマまたはデータベースで同じ順序で再作成した場合、通常、エラーは発生しません。(通常、例外は、循環参照の場合に発生します。たとえば、パッケージAにパッケージBへのコールが含まれ、パッケージBにパッケージAへのコールが含まれる場合、2回目にいずれかのパッケージを再コンパイルする必要があります。)

- 異種オブジェクト型の戻り値のフィルタ
  異種オブジェクト型の戻り値のフィルタ例および説明。

親トピック: メタデータAPIの使用

25.5.1 異種オブジェクト型の戻り値のフィルタ

異種オブジェクト型の戻り値のフィルタ例および説明。

返されるオブジェクトに対して、より詳細な制御を行うには、SET_FILTERプロシージャを使用して特定のメンバー型にのみフィルタを適用するよう指定します。これを行うには、SET_FILTERの4つ目のパラメータとして、メンバー型のパス名を指定します。また、EXCLUDE_PATH_EXPRフィルタを使用して、オブジェクト型のすべてのオブジェクトを除外できます。有効なパス名のリストは、TABLE_EXPORT_OBJECTSカタログ・ビューを参照してください。

例25-9に、返されるオブジェクトに対してさらに詳細な制御を指定するためのSET_FILTERの使用方法を示します。ユーザーscottとして接続します。パスワードは、tigerです。

例25-9 異種オブジェクト型の戻り値のフィルタ

1. 取得したオブジェクトを格納するための表my_metadataを作成します。さらに、プロシージャget_schema_md2を作成します。

```
DROP TABLE my_metadata;
CREATE TABLE my_metadata (md CLOB);
CREATE OR REPLACE PROCEDURE get_schema_md2 IS
-- Define local variables.
h       NUMBER;          -- handle returned by 'OPEN'
th      NUMBER;          -- handle returned by 'ADD_TRANSFORM'
doc     CLOB;            -- metadata is returned in a CLOB
BEGIN
 -- Specify the object type.
 h := DBMS_METADATA.OPEN('SCHEMA_EXPORT');
 -- Use filters to specify the schema.
 DBMS_METADATA.SET_FILTER(h,'SCHEMA','SCOTT');
 -- Use the fourth parameter to SET_FILTER to specify a filter
 -- that applies to a specific member object type.
 DBMS_METADATA.SET_FILTER(h,'NAME_EXPR','!='''MY_METADATA''','TABLE');
```

```
 -- Use the EXCLUDE_PATH_EXPR filter to exclude procedures.
 DBMS_METADATA.SET_FILTER(h,'EXCLUDE_PATH_EXPR','=''PROCEDURE''');
 -- Request that the metadata be transformed into creation DDL.
 th := DBMS_METADATA.ADD_TRANSFORM(h,'DDL');
 -- Use the fourth parameter to SET_TRANSFORM_PARAM to specify a parameter
 --  that applies to a specific member object type.
DBMS_METADATA.SET_TRANSFORM_PARAM(th,'SEGMENT_ATTRIBUTES',false,'TABLE');
 -- Fetch the objects.
 LOOP
   doc := dbms_metadata.fetch_clob(h);
   -- When there are no more objects to be retrieved, FETCH_CLOB returns NULL.
   EXIT WHEN doc IS NULL;
   -- Store the metadata in the table.
   INSERT INTO my_metadata(md) VALUES (doc);
   COMMIT;
 END LOOP;

 -- Release resources.
 DBMS_METADATA.CLOSE(h);
END;
/
```

2. 次のようにプロシージャを実行します。

```
EXECUTE get_schema_md2;
```

3. 次の問合せを実行して、取得された内容を確認します。

```
SET LONG 9000000
SET PAGESIZE 0
SELECT * FROM my_metadata;
```

親トピック: 異なるオブジェクト型のコレクションを取得するためのDBMS_METADATA APIの使用

25.6 オブジェクトのメタデータを比較するための DBMS_METADATA_DIFF APIの使用

DBMS_METADATAおよびDBMS_METADATA_DIFFの取得インタフェース、比較インタフェースおよび送信インタフェースの説明および使用例。この例では、2つの表のメタデータをフェッチして比較し、1つの表を他方と同じ内容に変更するALTER文を生成します。

わかりやすくするために、この例全体でファンクションの変形を使用しています。

例25-10 オブジェクト・メタデータの比較

1. 次のように、2つの表TAB1、TAB2を作成します。

```
SQL> CREATE TABLE TAB1
  2     (    "EMPNO" NUMBER(4,0),
  3          "ENAME" VARCHAR2(10),
  4          "JOB" VARCHAR2(9),
  5          "DEPTNO" NUMBER(2,0)
  6     ) ;

Table created.

SQL> CREATE TABLE TAB2
  2     (    "EMPNO" NUMBER(4,0) PRIMARY KEY ENABLE,
  3          "ENAME" VARCHAR2(20),
  4          "MGR" NUMBER(4,0),
  5          "DEPTNO" NUMBER(2,0)
  6     ) ;
```

```
Table created.
```

表TAB1とTAB2の差異に注意します。

- 表名が異なります。
- TAB2には主キー制約がありますが、TAB1にはありません。
- それぞれの表で、ENAME列の長さが異なります。
- TAB1にはJOB列がありますが、TAB2にはありません。
- TAB2にはMGR列がありますが、TAB1にはありません。

2. SXML形式の表メタデータを返すファンクションを作成します。DBMS_METADATA_DIFF APIを使用するときに、SXMLに関して注意する必要がある重要な点について示します。

- SXMLはオブジェクト・メタデータのXML表現です。
- 返されるSXMLは、DBMS_METADATA.GET_XMLによって返されるXMLと同じではありません。その場合のXMLは、複雑かつ不透明であり、バイナリ値やインスタンス固有の値などが含まれます。
- SXMLは、SQL作成DDLを直接XMLに変換したものと考えることができます。タグ名と構造は、『Oracle Database SQL言語リファレンス』で説明されている名前に対応しています。
- SXMLは、編集と比較を行えるように設計されています。

ここでは例を単純にするため、変換パラメータを使用して物理プロパティを抑止します。

```
SQL> CREATE OR REPLACE FUNCTION get_table_sxml(name IN VARCHAR2) RETURN CLOB IS
  2   open_handle NUMBER;
  3   transform_handle NUMBER;
  4   doc CLOB;
  5  BEGIN
  6   open_handle := DBMS_METADATA.OPEN('TABLE');
  7   DBMS_METADATA.SET_FILTER(open_handle,'NAME',name);
  8   --
  9   -- Use the 'SXML' transform to convert XML to SXML
 10   --
 11   transform_handle := DBMS_METADATA.ADD_TRANSFORM(open_handle,'SXML');
 12   --
 13   -- Use this transform parameter to suppress physical properties
 14   --
 15   DBMS_METADATA.SET_TRANSFORM_PARAM(transform_handle,'PHYSICAL_PROPERTIES',
 16                                     FALSE);
 17   doc := DBMS_METADATA.FETCH_CLOB(open_handle);
 18   DBMS_METADATA.CLOSE(open_handle);
 19   RETURN doc;
 20  END;
 21  /

Function created.
```

3. 次のように、get_table_sxmlファンクションを使用して、2つの表の表SXMLをフェッチします。

```
SQL> SELECT get_table_sxml('TAB1') FROM dual;

  <TABLE xmlns="http://xmlns.oracle.com/ku" version="1.0">
   <SCHEMA>SCOTT</SCHEMA>
   <NAME>TAB1</NAME>
   <RELATIONAL_TABLE>
```

```
      <COL_LIST>
         <COL_LIST_ITEM>
            <NAME>EMPNO</NAME>
            <DATATYPE>NUMBER</DATATYPE>
            <PRECISION>4</PRECISION>
            <SCALE>0</SCALE>
         </COL_LIST_ITEM>
         <COL_LIST_ITEM>
            <NAME>ENAME</NAME>
            <DATATYPE>VARCHAR2</DATATYPE>
            <LENGTH>10</LENGTH>
         </COL_LIST_ITEM>
         <COL_LIST_ITEM>
            <NAME>JOB</NAME>
            <DATATYPE>VARCHAR2</DATATYPE>
            <LENGTH>9</LENGTH>
         </COL_LIST_ITEM>
         <COL_LIST_ITEM>
            <NAME>DEPTNO</NAME>
            <DATATYPE>NUMBER</DATATYPE>
            <PRECISION>2</PRECISION>
            <SCALE>0</SCALE>
         </COL_LIST_ITEM>
      </COL_LIST>
   </RELATIONAL_TABLE>
</TABLE>

1 row selected.

SQL> SELECT get_table_sxml('TAB2') FROM dual;

  <TABLE xmlns="http://xmlns.oracle.com/ku" version="1.0">
   <SCHEMA>SCOTT</SCHEMA>
   <NAME>TAB2</NAME>
   <RELATIONAL_TABLE>
      <COL_LIST>
         <COL_LIST_ITEM>
            <NAME>EMPNO</NAME>
            <DATATYPE>NUMBER</DATATYPE>
            <PRECISION>4</PRECISION>
            <SCALE>0</SCALE>
         </COL_LIST_ITEM>
         <COL_LIST_ITEM>
            <NAME>ENAME</NAME>
            <DATATYPE>VARCHAR2</DATATYPE>
            <LENGTH>20</LENGTH>
         </COL_LIST_ITEM>
         <COL_LIST_ITEM>
            <NAME>MGR</NAME>
            <DATATYPE>NUMBER</DATATYPE>
            <PRECISION>4</PRECISION>
            <SCALE>0</SCALE>
         </COL_LIST_ITEM>
         <COL_LIST_ITEM>
            <NAME>DEPTNO</NAME>
            <DATATYPE>NUMBER</DATATYPE>
            <PRECISION>2</PRECISION>
            <SCALE>0</SCALE>
         </COL_LIST_ITEM>
      </COL_LIST>
      <PRIMARY_KEY_CONSTRAINT_LIST>
         <PRIMARY_KEY_CONSTRAINT_LIST_ITEM>
            <COL_LIST>
               <COL_LIST_ITEM>
                  <NAME>EMPNO</NAME>
               </COL_LIST_ITEM>
            </COL_LIST>
```

```
        </PRIMARY_KEY_CONSTRAINT_LIST_ITEM>
      </PRIMARY_KEY_CONSTRAINT_LIST>
   </RELATIONAL_TABLE>
</TABLE>

1 row selected.
```

4. 次のように、DBMS_METADATAブラウザAPIを使用して結果を比較します。

```
SQL> SELECT dbms_metadata.get_sxml('TABLE','TAB1') FROM dual;
SQL> SELECT dbms_metadata.get_sxml('TABLE','TAB2') FROM dual;
```

5. DBMS_METADATA_DIFF APIを使用してファンクションを作成し、2つの表のメタデータを比較します。このファンクションでは、ステップ2で定義したget_table_sxmlファンクションを使用します。

```
SQL> CREATE OR REPLACE FUNCTION compare_table_sxml(name1 IN VARCHAR2,
  2                                                name2 IN VARCHAR2) RETURN CLOB IS
  3   doc1 CLOB;
  4   doc2 CLOB;
  5   diffdoc CLOB;
  6   openc_handle NUMBER;
  7  BEGIN
  8   --
  9   -- Fetch the SXML for the two tables
 10   --
 11   doc1 := get_table_sxml(name1);
 12   doc2 := get_table_sxml(name2);
 13   --
 14   -- Specify the object type in the OPENC call
 15   --
 16   openc_handle := DBMS_METADATA_DIFF.OPENC('TABLE');
 17   --
 18   -- Add each document
 19   --
 20   DBMS_METADATA_DIFF.ADD_DOCUMENT(openc_handle,doc1);
 21   DBMS_METADATA_DIFF.ADD_DOCUMENT(openc_handle,doc2);
 22   --
 23   -- Fetch the SXML difference document
 24   --
 25   diffdoc := DBMS_METADATA_DIFF.FETCH_CLOB(openc_handle);
 26   DBMS_METADATA_DIFF.CLOSE(openc_handle);
 27   RETURN diffdoc;
 28  END;
 29  /

Function created.
```

6. 次のようにファンクションを使用して、2つの表のSXML差分ドキュメントをフェッチします。

```
SQL> SELECT compare_table_sxml('TAB1','TAB2') FROM dual;
<TABLE xmlns="http://xmlns.oracle.com/ku" version="1.0">
  <SCHEMA>SCOTT</SCHEMA>
  <NAME value1="TAB1">TAB2</NAME>
  <RELATIONAL_TABLE>
    <COL_LIST>
      <COL_LIST_ITEM>
        <NAME>EMPNO</NAME>
        <DATATYPE>NUMBER</DATATYPE>
        <PRECISION>4</PRECISION>
        <SCALE>0</SCALE>
      </COL_LIST_ITEM>
      <COL_LIST_ITEM>
        <NAME>ENAME</NAME>
        <DATATYPE>VARCHAR2</DATATYPE>
```

```
        <LENGTH value1="10">20</LENGTH>
      </COL_LIST_ITEM>
      <COL_LIST_ITEM src="1">
        <NAME>JOB</NAME>
        <DATATYPE>VARCHAR2</DATATYPE>
        <LENGTH>9</LENGTH>
      </COL_LIST_ITEM>
      <COL_LIST_ITEM>
        <NAME>DEPTNO</NAME>
        <DATATYPE>NUMBER</DATATYPE>
        <PRECISION>2</PRECISION>
        <SCALE>0</SCALE>
      </COL_LIST_ITEM>
      <COL_LIST_ITEM src="2">
        <NAME>MGR</NAME>
        <DATATYPE>NUMBER</DATATYPE>
        <PRECISION>4</PRECISION>
        <SCALE>0</SCALE>
      </COL_LIST_ITEM>
    </COL_LIST>
    <PRIMARY_KEY_CONSTRAINT_LIST src="2">
      <PRIMARY_KEY_CONSTRAINT_LIST_ITEM>
        <COL_LIST>
          <COL_LIST_ITEM>
            <NAME>EMPNO</NAME>
          </COL_LIST_ITEM>
        </COL_LIST>
      </PRIMARY_KEY_CONSTRAINT_LIST_ITEM>
    </PRIMARY_KEY_CONSTRAINT_LIST>
  </RELATIONAL_TABLE>
</TABLE>

1 row selected.
```

SXML差分ドキュメントでは、2つのSXMLドキュメントの結合内容が示され、差分を識別するXML属性value1およびsrcが使用されます。ある要素が1つのドキュメントのみに存在するときは、その要素がsrcでマーキングされます。したがって、<COL_LIST_ITEM src="1">は、この要素が最初のドキュメント(TAB1)に含まれていて、2番目のドキュメントに含まれていないことを表します。ある要素が両方のドキュメントに存在し、値がそれぞれ異なる場合、この要素の値は2番目のドキュメント内の値とされ、value1によって最初のドキュメント内のその値が示されます。たとえば、<LENGTH value1="10">20</LENGTH>は、長さがTAB1(最初のドキュメント)では10、TAB2では20であることを表します。

7. 次のように、DBMS_METADATA_DIFFブラウザAPIを使用して、結果を比較します。

```
SQL> SELECT dbms_metadata_diff.compare_sxml('TABLE','TAB1','TAB2') FROM dual;
```

8. DBMS_METADATA.CONVERT APIを使用してファンクションを作成し、ALTERXMLドキュメントを生成します。これは、片方のオブジェクトをもう一方のオブジェクトと同様にするためのALTER文が含まれているXML文書です。解析項目を使用して、個々のALTER文に関する情報を取得することもできます。(この例ではこれまでに定義したファンクションを使用しています。)

```
SQL> CREATE OR REPLACE FUNCTION get_table_alterxml(name1 IN VARCHAR2,
  2                                                name2 IN VARCHAR2) RETURN CLOB IS
  3   diffdoc CLOB;
  4   openw_handle NUMBER;
  5   transform_handle NUMBER;
  6   alterxml CLOB;
  7  BEGIN
  8   --
  9   -- Use the function just defined to get the difference document
```

```
 10   --
 11   diffdoc := compare_table_sxml(name1,name2);
 12   --
 13   -- Specify the object type in the OPENW call
 14   --
 15   openw_handle := DBMS_METADATA.OPENW('TABLE');
 16   --
 17   -- Use the ALTERXML transform to generate the ALTER_XML document
 18   --
 19   transform_handle := DBMS_METADATA.ADD_TRANSFORM(openw_handle,'ALTERXML');
 20   --
 21   -- Request parse items
 22   --
 23   DBMS_METADATA.SET_PARSE_ITEM(openw_handle,'CLAUSE_TYPE');
 24   DBMS_METADATA.SET_PARSE_ITEM(openw_handle,'NAME');
 25   DBMS_METADATA.SET_PARSE_ITEM(openw_handle,'COLUMN_ATTRIBUTE');
 26   --
 27   -- Create a temporary LOB
 28   --
 29   DBMS_LOB.CREATETEMPORARY(alterxml, TRUE );
 30   --
 31   -- Call CONVERT to do the transform
 32   --
 33   DBMS_METADATA.CONVERT(openw_handle,diffdoc,alterxml);
 34   --
 35   -- Close context and return the result
 36   --
 37   DBMS_METADATA.CLOSE(openw_handle);
 38   RETURN alterxml;
 39  END;
 40  /

Function created.
```

9. 次のようにファンクションを使用して、ALTER_XML文書をフェッチします。

```
SQL> SELECT get_table_alterxml('TAB1','TAB2') FROM dual;

<ALTER_XML xmlns="http://xmlns.oracle.com/ku" version="1.0">
   <OBJECT_TYPE>TABLE</OBJECT_TYPE>
   <OBJECT1>
      <SCHEMA>SCOTT</SCHEMA>
      <NAME>TAB1</NAME>
   </OBJECT1>
   <OBJECT2>
      <SCHEMA>SCOTT</SCHEMA>
      <NAME>TAB2</NAME>
   </OBJECT2>
   <ALTER_LIST>
      <ALTER_LIST_ITEM>
         <PARSE_LIST>
            <PARSE_LIST_ITEM>
               <ITEM>NAME</ITEM>
               <VALUE>MGR</VALUE>
            </PARSE_LIST_ITEM>
            <PARSE_LIST_ITEM>
               <ITEM>CLAUSE_TYPE</ITEM>
               <VALUE>ADD_COLUMN</VALUE>
            </PARSE_LIST_ITEM>
         </PARSE_LIST>
         <SQL_LIST>
            <SQL_LIST_ITEM>
               <TEXT>ALTER TABLE "SCOTT"."TAB1" ADD ("MGR" NUMBER(4,0))</TEXT>
            </SQL_LIST_ITEM>
         </SQL_LIST>
      </ALTER_LIST_ITEM>
```

```
      <ALTER_LIST_ITEM>
         <PARSE_LIST>
            <PARSE_LIST_ITEM>
               <ITEM>NAME</ITEM>
               <VALUE>JOB</VALUE>
            </PARSE_LIST_ITEM>
            <PARSE_LIST_ITEM>
               <ITEM>CLAUSE_TYPE</ITEM>
               <VALUE>DROP_COLUMN</VALUE>
            </PARSE_LIST_ITEM>
         </PARSE_LIST>
         <SQL_LIST>
            <SQL_LIST_ITEM>
               <TEXT>ALTER TABLE "SCOTT"."TAB1" DROP ("JOB")</TEXT>
            </SQL_LIST_ITEM>
         </SQL_LIST>
      </ALTER_LIST_ITEM>
      <ALTER_LIST_ITEM>
         <PARSE_LIST>
            <PARSE_LIST_ITEM>
               <ITEM>NAME</ITEM>
               <VALUE>ENAME</VALUE>
            </PARSE_LIST_ITEM>
            <PARSE_LIST_ITEM>
               <ITEM>CLAUSE_TYPE</ITEM>
               <VALUE>MODIFY_COLUMN</VALUE>
            </PARSE_LIST_ITEM>
            <PARSE_LIST_ITEM>
               <ITEM>COLUMN_ATTRIBUTE</ITEM>
               <VALUE> SIZE_INCREASE</VALUE>
            </PARSE_LIST_ITEM>
         </PARSE_LIST>
         <SQL_LIST>
            <SQL_LIST_ITEM>
               <TEXT>ALTER TABLE "SCOTT"."TAB1" MODIFY
                     ("ENAME" VARCHAR2(20))
               </TEXT>
            </SQL_LIST_ITEM>
         </SQL_LIST>
      </ALTER_LIST_ITEM>
      <ALTER_LIST_ITEM>
         <PARSE_LIST>
            <PARSE_LIST_ITEM>
               <ITEM>CLAUSE_TYPE</ITEM>
               <VALUE>ADD_CONSTRAINT</VALUE>
            </PARSE_LIST_ITEM>
         </PARSE_LIST>
         <SQL_LIST>
            <SQL_LIST_ITEM>
               <TEXT>ALTER TABLE "SCOTT"."TAB1" ADD  PRIMARY KEY
                      ("EMPNO") ENABLE
               </TEXT>
            </SQL_LIST_ITEM>
         </SQL_LIST>
      </ALTER_LIST_ITEM>
      <ALTER_LIST_ITEM>
         <PARSE_LIST>
            <PARSE_LIST_ITEM>
               <ITEM>NAME</ITEM>
               <VALUE>TAB1</VALUE>
            </PARSE_LIST_ITEM>
            <PARSE_LIST_ITEM>
               <ITEM>CLAUSE_TYPE</ITEM>
               <VALUE>RENAME_TABLE</VALUE>
            </PARSE_LIST_ITEM>
         </PARSE_LIST>
         <SQL_LIST>
```

```
            <SQL_LIST_ITEM>
               <TEXT>ALTER TABLE "SCOTT"."TAB1" RENAME TO "TAB2"</TEXT>
            </SQL_LIST_ITEM>
         </SQL_LIST>
      </ALTER_LIST_ITEM>
   </ALTER_LIST>
</ALTER_XML>


1 row selected.
```

10. 次のように、DBMS_METADATA_DIFFブラウザAPIを使用して、結果を比較します。

```
SQL> SELECT dbms_metadata_diff.compare_alter_xml('TABLE','TAB1','TAB2') FROM
dual;
```

11. ALTER_XML文書には、それぞれの変更のALTER_LISTが含まれています。それぞれのALTER_LIST_ITEMには、名前と値のペアとしての解析項目が含まれたPARSE_LISTと、特定の変更のSQLが含まれたSQL_LISTがあります。PARSE_LIST内の情報を使用してこの文書を解析し、どのSQL文を実行するかを決定できます。(たとえば、この場合は、変更の1つがDROP_COLUMNであり、それを実行しないことを決定できます。)

12. 次のように、DBMS_METADATA.CONVERT APIおよびALTER DDL変換を使用する最後の1つのファンクションを作成して、ALTER_XML文書をSQL DDLに変換します。

```
SQL> CREATE OR REPLACE FUNCTION get_table_alterddl(name1 IN VARCHAR2,
  2                                                name2 IN VARCHAR2) RETURN CLOB IS
  3   alterxml CLOB;
  4   openw_handle NUMBER;
  5   transform_handle NUMBER;
  6   alterddl CLOB;
  7  BEGIN
  8   --
  9   -- Use the function just defined to get the ALTER_XML document
 10   --
 11   alterxml := get_table_alterxml(name1,name2);
 12   --
 13   -- Specify the object type in the OPENW call
 14   --
 15   openw_handle := DBMS_METADATA.OPENW('TABLE');
 16   --
 17   -- Use ALTERDDL transform to convert the ALTER_XML document to SQL DDL
 18   --
 19   transform_handle := DBMS_METADATA.ADD_TRANSFORM(openw_handle,'ALTERDDL');
 20   --
 21   -- Use the SQLTERMINATOR transform parameter to append a terminator
 22   -- to each SQL statement
 23   --
 24   DBMS_METADATA.SET_TRANSFORM_PARAM(transform_handle,'SQLTERMINATOR',true);
 25   --
 26   -- Create a temporary lob
 27   --
 28   DBMS_LOB.CREATETEMPORARY(alterddl, TRUE );
 29   --
 30   -- Call CONVERT to do the transform
 31   --
 32   DBMS_METADATA.CONVERT(openw_handle,alterxml,alterddl);
 33   --
 34   -- Close context and return the result
 35   --
 36   DBMS_METADATA.CLOSE(openw_handle);
 37   RETURN alterddl;
```

```
 38  END;
 39  /

Function created.
```

13. 次のようにファンクションを使用して、SQL ALTER文をフェッチします。

```
SQL> SELECT get_table_alterddl('TAB1','TAB2') FROM dual;
ALTER TABLE "SCOTT"."TAB1" ADD ("MGR" NUMBER(4,0))
/
  ALTER TABLE "SCOTT"."TAB1" DROP ("JOB")
/
  ALTER TABLE "SCOTT"."TAB1" MODIFY ("ENAME" VARCHAR2(20))
/
  ALTER TABLE "SCOTT"."TAB1" ADD  PRIMARY KEY ("EMPNO") ENABLE
/
  ALTER TABLE "SCOTT"."TAB1" RENAME TO "TAB2"
/

1 row selected.
```

14. 次のように、DBMS_METADATA_DIFFブラウザAPIを使用して、結果を比較します。

```
SQL> SELECT dbms_metadata_diff.compare_alter('TABLE','TAB1','TAB2') FROM dual;
ALTER TABLE "SCOTT"."TAB1" ADD ("MGR" NUMBER(4,0))
  ALTER TABLE "SCOTT"."TAB1" DROP ("JOB")
  ALTER TABLE "SCOTT"."TAB1" MODIFY ("ENAME" VARCHAR2(20))
  ALTER TABLE "SCOTT"."TAB1" ADD  PRIMARY KEY ("EMPNO") USING INDEX
  PCTFREE 10 INITRANS 2 STORAGE ( INITIAL 16384 NEXT 16384 MINEXTENTS 1
  MAXEXTENTS 505 PCTINCREASE 50 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL
  DEFAULT)  ENABLE ALTER TABLE "SCOTT"."TAB1" RENAME TO "TAB2"

1 row selected.
```

親トピック: メタデータAPIの使用

25.7 DBMS_METADATA APIのプログラム・インタフェースに関するパフォーマンスのヒント

DBMS_METADATA APIのプログラム・インタフェース使用時にパフォーマンスを向上する方法について説明します。

具体的には次のとおりです。

1. ある型のすべてのオブジェクトを、次の型のオブジェクトをフェッチする前にフェッチします。たとえば、スキーマ内に含まれるすべてのオブジェクトの定義を取得する場合は、まずすべての表をフェッチし、次にすべての索引をフェッチし、その次にすべてのトリガーをフェッチします。この方法は、OPENコンテキストをネストする方法(表をフェッチした後でそのすべての索引、権限およびトリガーをフェッチし、その後、次の表をフェッチした後でそのすべての索引、権限およびトリガーをフェッチするという方法)より非常に高速です。「DBMS_METADATA APIの使用例」では、この非効率な後者の方法が示されていますが、その例の目的はほとんどのプログラム・コールを示すことであり、この方法がその目的に最適です。

2. SET_COUNTプロシージャを使用して、複数のオブジェクトを一度に取得します。これによってサーバーのラウンドトリップが最小化され、多くの冗長なファンクション・コールが削減されます。

3. DBMS_METADATA APIをコールするPL/SQLパッケージを記述する場合、LOB変数およびLOBを含むオブジェクト(SYS.KU$_DDLSなど)は、個々のファンクション内ではなく、パッケージ・スコープで宣言します。これによって、ファンクションの開始および終了時に、負荷の高い操作である、LOBの存続時間構造の作成および削除が実行されなくなり

ます。

親トピック: メタデータAPIの使用

25.8 DBMS_METADATA APIの使用例

DBMS_METADATA APIの使用方法の例。

デモを自動的に実行するスクリプトの手順は、次のとおりです。

- スキーマ(MDDEMO)および何人かの従業員名簿のユーザーを確立します。
- スキーマ内に従業員名簿に似せた表を3つ作成し、関連する索引、トリガーおよび権限を作成します。
- DBMS_METADATA APIを使用するPAYROLL_DEMOというパッケージを作成します。PAYROLL_DEMOパッケージには、PAYROLLから始まるMDDEMOスキーマ内の2つの表のDDLを取得するGET_PAYROLL_TABLESというプロシージャを含めます。各表に対して、表の関連する依存オブジェクト、索引、権限およびトリガーのDDLを取得します。すべてのDDLは、MDDEMO.DDLという名前の表に書き込まれます。

例を実行するには、次の手順に従います。

1. ユーザーsystemとしてSQL*Plusを起動します。パスワードの入力を求められます。

   ```
   sqlplus system
   ```

2. デモをインストールします。このデモは、rdbms/demoにあるファイルmddemo.sqlに含まれています。

   ```
   SQL> @mddemo
   ```

   このステップで行われる処理の詳細は、「DBMS_METADATAの例で行われる処理」を参照してください。

3. ユーザーmddemoとして接続します。パスワードを入力するように要求されます。パスワードもmddemoです。

   ```
   SQL> CONNECT mddemo
   Enter password:
   ```

4. 問合せの出力を完全なものにして読取り可能にするために、次のパラメータを設定します

   ```
   SQL> SET PAGESIZE 0
   SQL> SET LONG 1000000
   ```

5. 次のとおりGET_PAYROLL_TABLESプロシージャを実行します。

   ```
   SQL> CALL payroll_demo.get_payroll_tables();
   ```

6. 次のSQL問合せを実行します。

   ```
   SQL> SELECT ddl FROM DDL ORDER BY SEQNO;
   ```

   生成される出力は、GET_PAYROLL_TABLESプロシージャの実行結果です。デモのインストール時に、ステップ2で行われたすべてのDDLが示されます。実際の出力のリストについては、「GET_PAYROLL_TABLESプロシージャで生成される出力」を参照してください。

- DBMS_METADATAの例で行われる処理
  DBMS_METADATAの例の説明。
- GET_PAYROLL_TABLESプロシージャで生成される出力
  GET_PAYROLL_TABLESプロシージャから生成される出力の説明。

25.8.1 DBMS_METADATAの例で行われる処理

DBMS_METADATAの例の説明。

mddemoスクリプトを実行すると、次のステップが行われます。これらのステップをユーザーの状況に合わせることができます。

1. ユーザーが存在する場合は、次のとおり削除します。これで、新しいデータを使用した作業が開始できます。ユーザーが存在しない場合は、その影響を示すメッセージが表示されますが、問題はありません。デモは継続して実行されます。

```
CONNECT system
Enter password: password
SQL> DROP USER mddemo CASCADE;
SQL> DROP USER mddemo_clerk CASCADE;
SQL> DROP USER mddemo_mgr CASCADE;
```

2. mddemoで識別されるユーザーmddemoを作成します。

```
SQL> CREATE USER mddemo IDENTIFIED BY mddemo;
SQL> GRANT resource, connect, create session,
  1     create table,
  2     create procedure,
  3     create sequence,
  4     create trigger,
  5     create view,
  6     create synonym,
  7     alter session,
  8  TO mddemo;
```

3. clerkで識別されるユーザーmddemo_clerkを作成します。

```
CREATE USER mddemo_clerk IDENTIFIED BY clerk;
```

4. mgrで識別されるユーザーmddemo_mgrを作成します。

```
CREATE USER mddemo_mgr IDENTIFIED BY mgr;
```

5. mddemoとしてSQL*Plusに接続します(パスワードもmddemoです)。

```
CONNECT mddemo
Enter password:
```

6. いくつかの従業員名簿型の表を作成します。

```
SQL> CREATE TABLE payroll_emps
  2  ( lastname VARCHAR2(60) NOT NULL,
  3  firstname VARCHAR2(20) NOT NULL,
  4  mi VARCHAR2(2),
  5  suffix VARCHAR2(10),
  6  dob DATE NOT NULL,
  7  badge_no NUMBER(6) PRIMARY KEY,
  8  exempt VARCHAR(1) NOT NULL,
  9  salary NUMBER (9,2),
  10 hourly_rate NUMBER (7,2) )
  11 /
SQL> CREATE TABLE payroll_timecards
  2  (badge_no NUMBER(6) REFERENCES payroll_emps (badge_no),
  3  week NUMBER(2),
  4  job_id NUMBER(5),
  5  hours_worked NUMBER(4,2) )
  6 /
```

7. audit_trailというダミーの表を作成します。この表は、payrollで始まらない表は、GET_PAYROLL_TABLES プロシージャでは取得されないことを示すために使用します。

```
SQL> CREATE TABLE audit_trail
  2  (action_time DATE,
  3  lastname VARCHAR2(60),
  4  action LONG )
  5  /
```

8. 作成された表の権限を作成します。

```
SQL> GRANT UPDATE (salary,hourly_rate) ON payroll_emps TO mddemo_clerk;
SQL> GRANT ALL ON payroll_emps TO mddemo_mgr WITH GRANT OPTION;
SQL> GRANT INSERT,UPDATE ON payroll_timecards TO mddemo_clerk;
SQL> GRANT ALL ON payroll_timecards TO mddemo_mgr WITH GRANT OPTION;
```

9. 作成された表の索引を作成します。

```
SQL> CREATE INDEX i_payroll_emps_name ON payroll_emps(lastname);
SQL> CREATE INDEX i_payroll_emps_dob ON payroll_emps(dob);
SQL> CREATE INDEX i_payroll_timecards_badge ON payroll_timecards(badge_no);
```

10. 作成された表のトリガーを作成します。

```
SQL> CREATE OR REPLACE PROCEDURE check_sal( salary in number) AS BEGIN
  2  RETURN;
  3  END;
  4  /
```

例を簡単にするために、セキュリティは比較的低く設定しています。

```
SQL> CREATE OR REPLACE TRIGGER salary_trigger BEFORE INSERT OR UPDATE OF salary
ON payroll_emps
FOR EACH ROW WHEN (new.salary > 150000)
CALL check_sal(:new.salary)
/
SQL> CREATE OR REPLACE TRIGGER hourly_trigger BEFORE UPDATE OF hourly_rate ON
payroll_emps
FOR EACH ROW
BEGIN :new.hourly_rate:=:old.hourly_rate;END;
/
```

11. 生成されたDDLを保持する表を設定します。

```
CREATE TABLE ddl (ddl CLOB, seqno NUMBER);
```

12. DBMS_METADATAプロシージャを使用できる例を提供するPAYROLL_DEMOパッケージを作成します。

```
SQL> CREATE OR REPLACE PACKAGE payroll_demo AS PROCEDURE get_payroll_tables;
END;
/
```

ノート:

PAYROLL_DEMO パッケージの内容を含めて、この例のスクリプト全体を参照するには、$ORACLE_HOME/rdbms/demo ディレクトリにあるファイル mddemo.sql を参照してください。

親トピック: DBMS_METADATA APIの使用例

25.8.2 GET_PAYROLL_TABLESプロシージャで生成される出力

GET_PAYROLL_TABLESプロシージャから生成される出力の説明。

mddemo.payroll_demo.get_payroll_tables プロシージャの実行後に、次の問合せを実行できます。

```
SQL> SELECT ddl FROM ddl ORDER BY seqno;
```

結果は次のとおりです。前述の項で説明したとおり、スクリプトによって実行されたすべてのDDLが反映されます。

```
CREATE TABLE "MDDEMO"."PAYROLL_EMPS"
   (    "LASTNAME" VARCHAR2(60) NOT NULL ENABLE,
        "FIRSTNAME" VARCHAR2(20) NOT NULL ENABLE,
        "MI" VARCHAR2(2),
        "SUFFIX" VARCHAR2(10),
        "DOB" DATE NOT NULL ENABLE,
        "BADGE_NO" NUMBER(6,0),
        "EXEMPT" VARCHAR2(1) NOT NULL ENABLE,
        "SALARY" NUMBER(9,2),
        "HOURLY_RATE" NUMBER(7,2),
 PRIMARY KEY ("BADGE_NO") ENABLE
   ) ;
  GRANT UPDATE ("SALARY") ON "MDDEMO"."PAYROLL_EMPS" TO "MDDEMO_CLERK";
  GRANT UPDATE ("HOURLY_RATE") ON "MDDEMO"."PAYROLL_EMPS" TO "MDDEMO_CLERK";
  GRANT ALTER ON "MDDEMO"."PAYROLL_EMPS" TO "MDDEMO_MGR" WITH GRANT OPTION;
  GRANT DELETE ON "MDDEMO"."PAYROLL_EMPS" TO "MDDEMO_MGR" WITH GRANT OPTION;
  GRANT INDEX ON "MDDEMO"."PAYROLL_EMPS" TO "MDDEMO_MGR" WITH GRANT OPTION;
  GRANT INSERT ON "MDDEMO"."PAYROLL_EMPS" TO "MDDEMO_MGR" WITH GRANT OPTION;
  GRANT SELECT ON "MDDEMO"."PAYROLL_EMPS" TO "MDDEMO_MGR" WITH GRANT OPTION;
  GRANT UPDATE ON "MDDEMO"."PAYROLL_EMPS" TO "MDDEMO_MGR" WITH GRANT OPTION;
  GRANT REFERENCES ON "MDDEMO"."PAYROLL_EMPS" TO "MDDEMO_MGR" WITH GRANT OPTION;
  GRANT ON COMMIT REFRESH ON "MDDEMO"."PAYROLL_EMPS" TO "MDDEMO_MGR" WITH GRANT
OPTION;
  GRANT QUERY REWRITE ON "MDDEMO"."PAYROLL_EMPS" TO "MDDEMO_MGR" WITH GRANT OPTION;
  CREATE INDEX "MDDEMO"."I_PAYROLL_EMPS_DOB" ON "MDDEMO"."PAYROLL_EMPS" ("DOB")
  PCTFREE 10 INITRANS 2 MAXTRANS 255
  STORAGE(INITIAL 10240 NEXT 10240 MINEXTENTS 1 MAXEXTENTS 121 PCTINCREASE 50
  FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT) TABLESPACE "SYSTEM" ;
  CREATE INDEX "MDDEMO"."I_PAYROLL_EMPS_NAME" ON "MDDEMO"."PAYROLL_EMPS" ("LASTNAME")
  PCTFREE 10 INITRANS 2 MAXTRANS 255
  STORAGE(INITIAL 10240 NEXT 10240 MINEXTENTS 1 MAXEXTENTS 121 PCTINCREASE 50
  FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT) TABLESPACE "SYSTEM" ;
  CREATE OR REPLACE TRIGGER hourly_trigger before update of hourly_rate on
payroll_emps
for each row
begin :new.hourly_rate:=:old.hourly_rate;end;
/
ALTER TRIGGER "MDDEMO"."HOURLY_TRIGGER" ENABLE;
  CREATE OR REPLACE TRIGGER salary_trigger before insert or update of salary on
payroll_emps
for each row
WHEN (new.salary > 150000)  CALL check_sal(:new.salary)
/
ALTER TRIGGER "MDDEMO"."SALARY_TRIGGER" ENABLE;
CREATE TABLE "MDDEMO"."PAYROLL_TIMECARDS"
   (    "BADGE_NO" NUMBER(6,0),
        "WEEK" NUMBER(2,0),
        "JOB_ID" NUMBER(5,0),
        "HOURS_WORKED" NUMBER(4,2),
 FOREIGN KEY ("BADGE_NO")
  REFERENCES "MDDEMO"."PAYROLL_EMPS" ("BADGE_NO") ENABLE
   ) ;
  GRANT INSERT ON "MDDEMO"."PAYROLL_TIMECARDS" TO "MDDEMO_CLERK";
  GRANT UPDATE ON "MDDEMO"."PAYROLL_TIMECARDS" TO "MDDEMO_CLERK";
  GRANT ALTER ON "MDDEMO"."PAYROLL_TIMECARDS" TO "MDDEMO_MGR" WITH GRANT OPTION;
  GRANT DELETE ON "MDDEMO"."PAYROLL_TIMECARDS" TO "MDDEMO_MGR" WITH GRANT OPTION;
```

```
  GRANT INDEX ON "MDDEMO"."PAYROLL_TIMECARDS" TO "MDDEMO_MGR" WITH GRANT OPTION;
  GRANT INSERT ON "MDDEMO"."PAYROLL_TIMECARDS" TO "MDDEMO_MGR" WITH GRANT OPTION;
  GRANT SELECT ON "MDDEMO"."PAYROLL_TIMECARDS" TO "MDDEMO_MGR" WITH GRANT OPTION;
  GRANT UPDATE ON "MDDEMO"."PAYROLL_TIMECARDS" TO "MDDEMO_MGR" WITH GRANT OPTION;
  GRANT REFERENCES ON "MDDEMO"."PAYROLL_TIMECARDS" TO "MDDEMO_MGR" WITH GRANT OPTION;
  GRANT ON COMMIT REFRESH ON "MDDEMO"."PAYROLL_TIMECARDS" TO "MDDEMO_MGR" WITH GRANT
OPTION;
  GRANT QUERY REWRITE ON "MDDEMO"."PAYROLL_TIMECARDS" TO "MDDEMO_MGR" WITH GRANT
OPTION;
  CREATE INDEX "MDDEMO"."I_PAYROLL_TIMECARDS_BADGE" ON "MDDEMO"."PAYROLL_TIMECARDS"
("BADGE_NO")
  PCTFREE 10 INITRANS 2 MAXTRANS 255
  STORAGE(INITIAL 10240 NEXT 10240 MINEXTENTS 1 MAXEXTENTS 121 PCTINCREASE 50
  FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT) TABLESPACE "SYSTEM" ;
```

親トピック: DBMS_METADATA APIの使用例

25.9 DBMS_METADATAプロシージャの要約

DBMS_METADATA APIによって提供されるプロシージャを簡単に説明します。

これらのプロシージャの詳細は、『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』を参照してください。

次の表では、複数オブジェクトの取得に使用する、DBMS_METADATAプログラム・インタフェースのプロシージャを簡単に説明します。

表25-1 複数オブジェクトの取得に使用するDBMS_METADATAプロシージャ

PL/SQLプロシージャ名	説明
DBMS_METADATA.OPEN()	取得するオブジェクトの型、メタデータのバージョンおよびオブジェクト・モデルを指定します。
DBMS_METADATA.SET_FILTER()	取得するオブジェクトに、オブジェクト名、スキーマなどの制限を指定します。
DBMS_METADATA.SET_COUNT()	1 回の FETCH_xxx コールで取得するオブジェクトの最大数を指定します。
DBMS_METADATA.GET_QUERY()	FETCH_xxx で使用される問合せのテキストを返します。これをデバッグの機能として使用できます。
DBMS_METADATA.SET_PARSE_ITEM()	解析して返すオブジェクトの属性を指定して、出力を解析に利用します。DBMS_METADATA_PARSE_ITEMS を問い合せて、すべての有効な解析項目を確認できます。
DBMS_METADATA.ADD_TRANSFORM()	取得したオブジェクトの XML 表現に、FETCH_xxx によって適用される変換を指定します。DBMS_METADATA_TRANSFORMS ビューを問い合せて、すべての有効な Oracle 提供の変換を確認できます。

PL/SQLプロシージャ名	説明
DBMS_METADATA.SET_TRANSFORM_PARAM()	transform_handle.で識別される XSLT スタイルシートのパラメータを指定します DBMS_METADATA_TRANSFORM_PARAMS ビューを問い合せて、各変換で有効なすべての変換パラメータを確認できます。
DBMS_METADATA.SET_REMAP_PARAM()	transform_handle で識別される XSLT スタイルシートのパラメータを指定します。
DBMS_METADATA.FETCH_xxx()	OPEN、SET_FILTER、SET_COUNT、ADD_TRANSFORM などで確立された基準を満たすオブジェクトのメタデータを返します。
DBMS_METADATA.CLOSE()	OPEN によって返されたハンドルを無効にし、関連付けられた状態をクリーンアップします。

次の表では、DBMS_METADATAブラウザ・インタフェースで提供されるプロシージャを簡単に説明します。これらのファンクションは、1つ以上の依存オブジェクトまたは権限付与されたオブジェクトのメタデータを返します。これらのプロシージャでは、異種オブジェクト・タイプを処理できません。

表25-2 ブラウザ・インタフェースで使用するDBMS_METADATAプロシージャ

PL/SQLプロシージャ名	説明
DBMS_METADATA.GET_xxx()	シングル・オブジェクトに対してメタデータを返す方法を提供します。各 GET_xxx コールは、OPEN プロシージャ、1 つか 2 つの SET_FILTER コール、ADD_TRANSFORM プロシージャ(オプション)、FETCH_xxx コールおよび CLOSE プロシージャで構成されます。 object_type パラメータには、OPEN プロシージャと同じセマンティクスが含まれます。schema および name はフィルタ処理に使用されます。 変換を指定した場合、セッション・レベルの変換フラグが継承されます。
DBMS_METADATA.GET_DEPENDENT_xxx()	XML または DDL で指定したとおり、1 つ以上の依存オブジェクトに対してメタデータが返されます。
DBMS_METADATA.GET_GRANTED_xxx()	XML または DDL で指定したとおり、権限が付与された 1 つ以上のオブジェクトに対してメタデータが返されます。

次の表では、XMLの送信で使用されるDBMS_METADATAプロシージャおよびファンクションを簡単に説明します。

表25-3 XMLデータの送信に使用するDBMS_METADATAのプロシージャおよびファンクション

PL/SQL名	説明
DBMS_METADATA.OPENW()	書込みコンテキストをオープンします。

PL/SQL名	説明
DBMS_METADATA.ADD_TRANSFORM()	XML 文書の変換を指定します。
DBMS_METADATA.SET_TRANSFORM_PARAM() and DBMS_METADATA.SET_REMAP_PARAM()	SET_TRANSFORM_PARAM は、変換のパラメータを指定します。 SET_REMAP_PARAM は、変換の再マップを指定します。
DBMS_METADATA.SET_PARSE_ITEM()	解析するオブジェクト属性を指定します。
DBMS_METADATA.CONVERT()	XML 文書を DDL に変換します。
DBMS_METADATA.PUT()	データベースに XML 文書を送信します。
DBMS_METADATA.CLOSE()	OPENW でオープンしたコンテキストをクローズします。

親トピック: メタデータAPIの使用

25.10 DBMS_METADATA_DIFFプロシージャの要約

DBMS_METADATA_DIFF APIによって提供されるプロシージャおよびファンクションを簡単に説明します。

これらのプロシージャの詳細は、『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』を参照してください。

表25-4 DBMS_METADATA_DIFFのプロシージャおよびファンクション

PL/SQLプロシージャ名	説明
OPENC ファンクション	比較するオブジェクトの型を指定します。
ADD_DOCUMENT プロシージャ	比較する SXML ドキュメントを指定します。
FETCH_CLOB ファンクションおよびプロシージャ	ADD_DOCUMENT で指定した 2 つのドキュメント間の差異を示す CLOB を返します。
CLOSE プロシージャ	OPENC によって返されたハンドルを無効にし、関連付けられた状態をクリーンアップします。

親トピック: メタデータAPIの使用

26 オリジナルのエクスポート

Oracle Databaseのデータをオペレーティング・システムのファイルにバイナリ形式で書き込むために、オリジナルのエクスポート・ユーティリティ(exp)が使用されました。

このファイルはデータベースの外部に格納されるため、オリジナルのインポート・ユーティリティを使用して、別のOracle Databaseに読み込むことができます。

> ノート:
>
> オリジナルのエクスポートは、Oracle Database 11g からは原則としてサポートされなくなりました。Oracle Database 11g 以降のリリースでサポートされているオリジナルのエクスポートの使用方法は、XMLType データの Oracle Database 10g リリース 2 (10.2)以前への下位移行のみです。新しい Oracle Data Pump エクスポートおよびインポート・ユーティリティを使用することをお薦めします。このガイドラインの唯一の例外は、元のエクスポートおよびインポートが必要な次の状況です。
>
> - オリジナルのエクスポート・ユーティリティ(exp)を使用して作成されたファイルをインポートする必要がある。
> - オリジナルのインポート・ユーティリティ(imp)を使用してインポートされるファイルをエクスポートする必要がある。このような例として考えられるのは、Oracle Database 10g からデータをエクスポートし、それより前のデータベース・リリースにインポートする場合です。

- エクスポート・ユーティリティとは
  オリジナルのエクスポート・ユーティリティ(exp)は、ハードウェアやソフトウェアの構成に関係なく、Oracle Databaseインスタンス間でデータ・オブジェクトを転送するための簡単な手段を提供します。
- エクスポート・ユーティリティを使用する前に
  オリジナルのエクスポート(exp)ユーティリティを使用する前に、準備チェックリストを確認し、サーバーやOracle Databaseのインスタンスで必要に応じたタスクの実行を完了しておきます。
- エクスポート・ユーティリティの起動
  3種類のいずれかの方法を使用して、エクスポート・ユーティリティを起動してパラメータを指定できます。
- エクスポート・モード
  エクスポート・ユーティリティには、4種類の操作モードがあります。
- エクスポート・パラメータ
  エクスポート・コマンドライン・パラメータについて説明します。
- エクスポート・セッションの例
  エクスポート・セッションのタイプの例。
- 警告、エラーおよび完了メッセージ
  これらの項では、エクスポート・ユーティリティによって発行される各種のメッセージと、ログ・ファイルへのメッセージの保存方法について説明します。
- 終了コードによる結果の検査と表示
  エクスポートでは、操作の完了後、すぐに結果を表示できます。プラットフォームによっては、実行結果はプロセス終了コードに通知され、ログ・ファイルにも記録されます。
- 従来型パス・エクスポートとダイレクト・パス・エクスポート
  エクスポートでは、2つの方法で表データをエクスポートできます。

- ダイレクト・パス・エクスポートの起動
  ダイレクト・パス・エクスポートを使用するには、コマンドラインまたはパラメータ・ファイルでDIRECT=yパラメータを指定します。
- オリジナルのOracle Data Pumpエクスポートのネットワークに関する考慮事項
  ネットワーク経由でオリジナルのエクスポート(exp)を使用する場合は、プロトコルと接続修飾文字列を確認してください。
- 文字セットおよびグローバリゼーション・サポートに関する考慮点
  これらの項では、ユーザー・データおよびデータ定義言語(DDL)の文字セット変換に関連するエクスポート・ユーティリティのグローバリゼーション・サポートの動作について説明します。
- エクスポート・ユーティリティおよびインポート・ユーティリティでのインスタンス親和性の使用
  インスタンス親和性を使用して、インポートおよびエクスポートするデータベース内のインスタンスにジョブを関連付けることができます。
- データベース・オブジェクトのエクスポート時の考慮点
  これらの項では、特定のデータベース・オブジェクトをエクスポートする際に考慮する必要がある点について説明します。
- トランスポータブル表領域
  トランスポータブル表領域機能は、一連の表領域を、あるOracle Databaseから他のOracle Databaseに移動できる機能です。
- 読取り専用データベースからのエクスポート
  読取り専用データベースからのエクスポートについて説明します。
- エクスポート・ユーティリティおよびインポート・ユーティリティを使用したデータベース移行のパーティション化
  エクスポート・ユーティリティおよびインポート・ユーティリティを使用して大規模データベースを移行する場合、移行を複数のエクスポート・ジョブおよびインポート・ジョブにパーティション化するとより効率的です。
- リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用
  リリースが異なるエクスポート・ユーティリティおよびOracle Databaseを使用することに関連した互換性の問題について説明します。

親トピック: その他のユーティリティ

26.1 エクスポート・ユーティリティとは

オリジナルのエクスポート・ユーティリティ(exp)は、ハードウェアやソフトウェアの構成に関係なく、Oracle Databaseインスタンス間でデータ・オブジェクトを転送するための簡単な手段を提供します。

ノート:

オリジナルのエクスポートは、Oracle Database 11g からは原則としてサポートされなくなりました。Oracle Database 11g 以降のリリースでサポートされているオリジナルのエクスポートの使用方法は、XMLType データの Oracle Database 10g リリース 2 (10.2)以前への下位移行のみです。新しい Oracle Data Pump エクスポートおよびインポート・ユーティリティを使用することをお薦めします。このガイドラインの唯一の例外は、元のエクスポートおよびインポートが必要な次の状況です。

- オリジナルのエクスポート・ユーティリティ(exp)を使用して作成されたファイルをインポートする必要がある。
- オリジナルのインポート・ユーティリティ(imp)を使用してインポートされるファイルをエクスポートする必要がある。このような例として考えられるのは、Oracle Database 10g からデータをエクスポートし、それより前の

データベース・リリースにインポートする場合です。

Oracle Databaseインスタンスに対してエクスポート・ユーティリティを実行すると、オブジェクト(表など)が抽出された後で、それに関連するオブジェクト(索引、コメント、権限など)があれば、それらのオブジェクトが抽出されます。

エクスポート・ファイルは、通常はディスクまたはテープに格納されるOracleバイナリ形式のダンプ・ファイルです。ダンプ・ファイルは、FTPを使用して転送できます。また、物理的に(テープなどで)別のサイトに転送することもできます。その後、このファイルをインポート・ユーティリティで使用すると、ネットワーク接続されていないデータベース間でデータを転送できます。このデータ転送は、データベースが異なるハードウェア構成およびソフトウェア構成のプラットフォームに存在していても可能です。このファイルは、通常のバックアップ手順とともに、追加のバックアップとして使用することもできます。

オリジナルのエクスポート・ダンプ・ファイルは、オリジナルのOracleインポート・ユーティリティでのみ読み取れます。ダンプ・ファイルの作成に使用したエクスポート・ユーティリティのバージョンよりも古いバージョンのインポート・ユーティリティは使用できません。

実際にインポートを実行せずに、エクスポート・ファイルの内容を表示することもできます。内容を表示するには、インポートのSHOWパラメータを使用します。ASCII固定形式ファイルまたはデリミタ付きファイルからデータをロードする場合は、SQL*Loaderユーティリティを使用します。

関連トピック

- SHOW

親トピック: オリジナルのエクスポート

26.2 エクスポート・ユーティリティを使用する前に

オリジナルのエクスポート(exp)ユーティリティを使用する前に、準備チェックリストを確認し、サーバーやOracle Databaseのインスタンスで必要に応じたタスクの実行を完了しておきます。

- エクスポート・ユーティリティを使用するための準備チェックリスト
  オリジナルのエクスポート・ユーティリティ(exp)の使用開始前に、次のチェックを実行します。
- catexp.sqlまたはcatalog.sqlの実行
  エクスポート・ユーティリティを使用するには、データベースを作成するかまたは新しいリリースに移行した後で、スクリプト`catexp.sql`または(`catexp.sql`を実行する)`catalog.sql`を実行する必要があります。
- エクスポート操作に必要な十分なディスク領域の確保
  エクスポートを実行する前に、エクスポート・ファイルを書き込むための十分なディスクまたはテープのストレージが存在することを確認してください。
- エクスポート操作およびインポート操作のアクセス権の確認
  エクスポート・ユーティリティを使用するには、Oracle Databaseに対する`CREATE SESSION`権限が必要です。

親トピック: オリジナルのエクスポート

26.2.1 エクスポート・ユーティリティを使用するための準備チェックリスト

オリジナルのエクスポート・ユーティリティ(exp)の使用開始前に、次のチェックを実行します。

問題の発生を防止するために、次の各条件が満たされていることを確認します。

- データベースを手動で作成した場合は、`catexp.sql`または`catalog.sql`スクリプトが実行されたことの確認。Database Configuration Assistant(DBCA)を使用してデータベースを作成した場合は、これらのスクリプトを実行する必要はありません。

- エクスポート・ファイルを書き込むための十分なディスクまたはテープ記憶域が存在することの確認
- 必要なアクセス権限を所有していることの確認

親トピック: エクスポート・ユーティリティを使用する前に

26.2.2 catexp.sqlまたはcatalog.sqlの実行

エクスポート・ユーティリティを使用するには、データベースを作成するかまたは新しいリリースに移行した後で、スクリプトcatexp.sqlまたは(catexp.sqlを実行する)catalog.sqlを実行する必要があります。

データベースに対して、catexp.sqlまたはcatalog.sqlを実行するのは1回のみです。スクリプトを実行すると、次の処理が行われ、データベースはエクスポートおよびインポート操作に備えて調整されます。

- データ・ディクショナリへの必要なエクスポート・ビューおよびインポート・ビューの作成
- EXP_FULL_DATABASEロールおよびIMP_FULL_DATABASEロールの作成
- EXP_FULL_DATABASEおよびIMP_FULL_DATABASEロールへのすべての必要な権限の割当て
- DBAロールへのEXP_FULL_DATABASEおよびIMP_FULL_DATABASEの割当て
- インストールされているcatexp.sqlのバージョンの記録

EXP_FULL_DATABASEロールおよびIMP_FULL_DATABASEロールは強力です。これらのロールをユーザーに付与する際は、十分に注意する必要があります。

親トピック: エクスポート・ユーティリティを使用する前に

26.2.3 エクスポート操作に必要な十分なディスク領域の確保

エクスポートを実行する前に、エクスポート・ファイルを書き込むための十分なディスクまたはテープのストレージが存在することを確認してください。

十分な領域がないと、エクスポート・ユーティリティは書込み失敗エラーが発生して終了します。

表のサイズから、必要な最大領域を見積もることもできます。表のサイズは、OracleデータベースのUSER_SEGMENTSビューで確認できます。次の問合せでは、すべての表のディスク使用率が表示されます。

```
SELECT SUM(BYTES) FROM USER_SEGMENTS WHERE SEGMENT_TYPE='TABLE';
```

問合せの結果には、LOB(ラージ・オブジェクト)またはVARRAY列あるいはパーティション表に格納されているデータ用のディスク領域は含まれません。

関連項目:

ディクショナリ・ビューの詳細は、『Oracle Databaseリファレンス』を参照してください

親トピック: エクスポート・ユーティリティを使用する前に

26.2.4 エクスポート操作およびインポート操作のアクセス権の確認

エクスポート・ユーティリティを使用するには、Oracle DatabaseにCREATE SESSION権限が必要です。

この権限は、データベースの作成時に設定されるCONNECTロールに含まれます。別のユーザーが所有している表をエクスポートするには、EXP_FULL_DATABASEロールを有効にする必要があります。このロールは、すべてのデータベース管理者(DBA)に

付与されます。

EXP_FULL_DATABASEロールに含まれているシステム権限を持っていないと、別のユーザーのスキーマ内のオブジェクトをエクスポートできません。たとえば、そのシノニムを作成した場合でも、別のユーザーのスキーマ内の表をエクスポートできません。

一部のシステム・スキーマは、ユーザー・スキーマではないため、エクスポートできません。システム・スキーマには、Oracleが管理するデータおよびメタデータが含まれています。エクスポートされないスキーマの例としては、SYS、ORDSYS、MDSYSなどがあります。

親トピック: エクスポート・ユーティリティを使用する前に

26.3 エクスポート・ユーティリティの起動

3種類のいずれかの方法を使用して、エクスポート・ユーティリティを起動してパラメータを指定できます。

これらのいずれかの方法を使用する前に、使用可能なパラメータについての説明を必ず読んでください。詳細は、「エクスポート・パラメータ」を参照してください。

- SYSDBAでのエクスポート・ユーティリティの起動
  SYSDBAは内部的に使用され、一般ユーザーとは異なる特別な機能を持ちます。
- コマンドライン・エントリ
  すべての有効なパラメータおよびその値をコマンドラインから指定できます。
- パラメータ・ファイル
  オリジナルのエクスポート・ユーティリティを起動し、パラメータ・ファイルに、有効なすべてのエクスポート・パラメータおよびその値を指定できます。
- 対話方式モード
  各パラメータの値をプロンプトで入力するには、コマンドラインでexpを指定します。
- オンライン・ヘルプの利用
  エクスポート・ユーティリティには、オンライン・ヘルプがあります。エクスポート・ユーティリティのヘルプを表示するには、コマンドラインでexp help=yを入力します。

親トピック: オリジナルのエクスポート

26.3.1 SYSDBAでのエクスポート・ユーティリティの起動

SYSDBAは内部的に使用され、一般ユーザーとは異なる特別な機能を持ちます。

そのため、通常、次の場合以外は、エクスポート・ユーティリティをSYSDBAで起動する必要はありません。

- Oracleサポート・サービスから要求された場合
- トランスポータブル表領域セットをインポートする場合

親トピック: エクスポート・ユーティリティの起動

26.3.2 コマンドライン・エントリ

すべての有効なパラメータおよびその値をコマンドラインから指定できます。

次の構文を使用します(ユーザー名とパスワードの入力を求められます)。

```
exp PARAMETER=value
```

または

```
exp PARAMETER=(value1,value2,...,valuen)
```

システムでのコマンドラインの最大長を超える数のパラメータは指定できません。

親トピック: エクスポート・ユーティリティの起動

26.3.3 パラメータ・ファイル

オリジナルのエクスポート・ユーティリティを起動し、パラメータ・ファイルに、有効なすべてのエクスポート・パラメータおよびその値を指定できます。

複数のパラメータを1つのファイルに格納することによって、これらを簡単に変更したり再利用することができます。この方法を使用してオリジナルのエクスポート・ユーティリティを起動することをお薦めします。データベースごとに別のパラメータを使用する場合は、複数のパラメータ・ファイルを作成できるので、それぞれにパラメータ・ファイルを用意すると便利です。

フラット・ファイル用のテキスト・エディタを使用してパラメータ・ファイルを作成します。コマンドライン・オプション`PARFILE=filename`を指定すると、エクスポート・ユーティリティは、コマンドラインからではなく指定されたファイルからパラメータを読み込みます。たとえば:

パラメータ・ファイル指定の構文は、次のいずれかを使用します。

```
PARAMETER=value
PARAMETER=(value)
PARAMETER=(value1, value2, ...)
```

次に、パラメータ・ファイル内のリストの一部を示します。

```
FULL=y
FILE=dba.dmp
GRANTS=y
INDEXES=y
CONSISTENT=y
```

ノート:

パラメータ・ファイルの最大サイズは、オペレーティング・システムによって制限されます。また、パラメータ・ファイル名はオペレーティング・システムのネーミング規則に従います。

ポンド(#)記号を使用すると、パラメータ・ファイルにコメントを追加できます。ポンド(#)の右側にある文字はすべて無視されます。

コマンドラインでのパラメータの入力と同時にパラメータ・ファイルを指定できます。実際、パラメータ・ファイルとコマンドラインの両方に同じパラメータを指定することもできます。コマンドラインでの`PARFILE`パラメータと他のパラメータの位置によって、優先されるパラメータが決まります。たとえば、パラメータ・ファイル`params.dat`にパラメータ`INDEXES=y`,を指定し、エクスポート・ユーティリティを次のコマンドラインで起動するとします。

```
exp PARFILE=params.dat INDEXES=n
```

この場合、`INDEXES=n`は`PARFILE=params.dat`の後にあるため、パラメータ・ファイルに指定されている`INDEXES`パラメータの値は、`INDEXES=n`によって上書きされます。

親トピック: エクスポート・ユーティリティの起動

26.3.4 対話方式モード

各パラメータの値をプロンプトで入力するには、コマンドラインでexpを指定します。

プロンプトでユーザー名とパスワードを入力した後、一般的に使用されるパラメータが表示されます。

デフォルトのパラメータ値を受け入れるか(示された場合)、または異なる値を入力できます。コマンドライン対話方式では、すべての機能に関してプロンプトが表示されるわけではありません。また、コマンドライン対話方式は下位互換用のみに提供されています。対話方式のインタフェースを使用する場合は、Oracle Enterprise Managerのエクスポート・ウィザードを使用することをお薦めします。

- 対話方式でのエクスポートを使用する際の制限事項
  対話方式でのエクスポートを使用する際の制限事項について説明します。

親トピック: エクスポート・ユーティリティの起動

26.3.4.1 対話方式でのエクスポートを使用する際の制限事項

対話方式でエクスポートを使用する際の制限事項について説明します。

対話方式による方法を使用する場合は、次のことに注意してください。

- ユーザー・モードでは、データをエクスポートする前にすべてのユーザー名をエクスポートに含めるように要求されます。ユーザー・リストの終わりを指定して現在のエクスポート・セッションを開始するには、[Enter]を押します。
- 表モードでは、スキーマの接頭辞を指定しないと、デフォルトによりエクスポート実行者のスキーマ、または現在のセッション内の最後にエクスポートされた表が含まれているスキーマになります。

  たとえば、bethが表モードでエクスポートする特権ユーザーの場合、別のスキーマが指定されるまで、すべての表がbethスキーマに存在するとみなされます。別のユーザーのスキーマ内の表をエクスポートできるのは、特権ユーザー(EXP_FULL_DATABASEロールを持っているユーザー)のみです。
- エクスポートする表を指定するプロンプトで、NULL表リストを指定すると、エクスポート・ユーティリティは終了します。

親トピック: 対話方式モード

26.3.5 オンライン・ヘルプの利用

エクスポート・ユーティリティには、オンライン・ヘルプがあります。エクスポート・ユーティリティのヘルプを表示するには、コマンドラインでexp help=yを入力します。

親トピック: エクスポート・ユーティリティの起動

26.4 エクスポート・モード

エクスポート・ユーティリティには、4種類の操作モードがあります。

具体的には次のとおりです。

- 全体エクスポート: データベース全体をエクスポートします。EXP_FULL_DATABASEロールを所有するユーザーのみが使用できます。FULLパラメータを使用してこのモードを指定します。
- 表領域モード: 特権ユーザーが表領域をOracle Database間で移動できます。TRANSPORT_TABLESPACEパラメータを使用してこのモードを指定します。
- ユーザー・モード: 所有するすべてのオブジェクト(表、権限、索引、プロシージャなど)をエクスポートできます。特権ユー

ザーがユーザー・モードでインポートする場合、指定したユーザー・グループのユーザーのスキーマにあるすべてのオブジェクトをインポートできます。エクスポート・ユーティリティでOWNERパラメータを使用してこのモードを指定します。

- 表モード: 特定の表およびパーティションをエクスポートできます。特権ユーザーは、インポートする表を含むスキーマを指定して、その表を修飾できます。スキーマ名が指定されていない表は、デフォルトでエクスポート実行者のスキーマ名になります。TABLESパラメータを使用してこのモードを指定します。

表26-1に、各モードでエクスポートおよびインポートされるオブジェクトのリストを示します。

ノート:

遅延セグメント作成によって作成された表に、作成されたセグメントがない場合、オリジナルのエクスポート・ユーティリティではエクスポートされません。セグメントを作成する最も一般的な方法は表に行を格納することですが、ALTER TABLE ALLOCATE EXTENTS のような他の操作では、セグメントも作成されます。セグメントが存在する表をエクスポートすると、SEGMENT CREATION DEFERRED 句は、オリジナルのインポート・ユーティリティによって実行される CREATE TABLE 文に含まれません。

従来型パス・エクスポートまたはダイレクト・パス・エクスポートを使用すると、表領域モード以外のどのモードでもエクスポートできます。従来型パス・エクスポートとダイレクト・パス・エクスポートの違いについては、「従来型パス・エクスポートとダイレクト・パス・エクスポート」を参照してください。

表26-1 各モードでエクスポートされるオブジェクト

オブジェクト	表モード	ユーザー・モード	全データベース・モード	表領域モード
分析クラスタ	なし	あり	あり	なし
分析表/統計	あり	あり	あり	あり
アプリケーション・コンテキスト	なし	なし	あり	なし
監査情報	あり	あり	あり	なし
B ツリー索引、ビットマップ索引、ドメイン・ファンクション索引	あり脚注 1	あり	あり	あり
クラスタ定義	なし	あり	あり	あり
列コメントおよび表コメント	あり	あり	あり	あり
データベース・リンク	なし	あり	あり	なし
デフォルト・ロール	なし	なし	あり	なし

オブジェクト	表モード	ユーザー・モード	全データベース・モード	表領域モード
ディメンション	なし	あり	あり	なし
ディレクトリ別名	なし	なし	あり	なし
外部表(データなし)	あり	あり	あり	なし
外部関数ライブラリ	なし	あり	あり	なし
表の所有者以外のユーザーが所有する索引	あり(特権ユーザーのみ)	あり	あり	あり
索引タイプ	なし	あり	あり	なし
Java リソースおよび Java クラス	なし	あり	あり	なし
ジョブ・キュー	なし	あり	あり	なし
ネストした表のデータ	あり	あり	あり	あり
オブジェクト権限	あり(表および索引のみ)	あり	あり	あり
表で使用されるオブジェクト型定義	あり	あり	あり	あり
オブジェクト型	なし	あり	あり	なし
演算子	なし	あり	あり	なし
パスワード履歴	なし	なし	あり	なし
インスタンスの事後処理およびオブジェクト	なし	なし	あり	なし
スキーマの事後プロシージャ処理およびオブジェクト	なし	あり	あり	なし

オブジェクト	表モード	ユーザー・モード	全データベース・モード	表領域モード
表の事後処理	あり	あり	あり	あり
表の事後プロシージャ処理およびオブジェクト	あり	あり	あり	あり
スキーマの事前プロシージャ・オブジェクトおよび処理	なし	あり	あり	なし
表の事前処理	あり	あり	あり	あり
表の事前プロシージャ処理	あり	あり	あり	あり
プライベート・シノニム	なし	あり	あり	なし
プロシージャ・オブジェクト	なし	あり	あり	なし
プロファイル	なし	なし	あり	なし
パブリック・シノニム	なし	なし	あり	なし
参照整合性制約	あり	あり	あり	なし
リフレッシュ・グループ	なし	あり	あり	なし
リソース・コスト	なし	なし	あり	なし
ロール権限	なし	なし	あり	なし
ロール	なし	なし	あり	なし
ロールバック・セグメント定義	なし	なし	あり	なし
表のセキュリティ・ポリシー	あり	あり	あり	あり
順序番号	なし	あり	あり	なし
スナップショット・ログ	なし	あり	あり	なし

オブジェクト	表モード	ユーザー・モード	全データベース・モード	表領域モード
スナップショットおよびマテリアライズド・ビュー	なし	あり	あり	なし
システム権限	なし	なし	あり	なし
表制約(主キー制約、一意制約、CHECK 制約)	あり	あり	あり	あり
表データ	あり	あり	あり	あり
表定義	あり	あり	あり	あり
表領域定義	なし	なし	あり	なし
表領域割当て制限	なし	なし	あり	なし
トリガー	あり	あり脚注 2	あり脚注 3	あり
他のユーザーが所有するトリガー	あり(特権ユーザーのみ)	なし	なし	なし
ユーザー定義	なし	なし	あり	なし
ユーザー・プロキシ	なし	なし	あり	なし
ユーザー・ビュー	なし	あり	あり	なし
ユーザー・ストアド・プロシージャ、ユーザー・ストアド・パッケージおよびユーザー・ストアド・ファンクション	なし	あり	あり	なし

脚注1

非特権ユーザーがエクスポートおよびインポートできるのは、そのユーザー自身が所有する表に関する索引のみです。他のユーザーが所有する表に関する索引や、ユーザー自身が所有する表に関して他のユーザーが作成した索引はエクスポートできません。特権ユーザーは、エクスポートおよびインポート対象に指定したユーザーの表に関する索引が、表の所有者以外のユーザーが作成したものであっても、その索引をエクスポートおよびインポートできます。指定したユーザーが他のユーザーの表に関する索引を所有しているときは、エクスポートするユーザーのリストに表の所有者であるユーザーを指定しないかぎり、その索引はエクスポートされません。

脚注2

特権ユーザーも非特権ユーザーも、そのユーザー自身が所有するすべてのトリガーを(他のユーザーが所有する表に関するトリガーの場合でも)、エクスポートおよびインポートできます。

脚注3

全体エクスポートでは、スキーマSYSが所有するトリガーはエクスポートされません。SYSトリガーは、全体インポートの前後のいずれかに手動で再作成する必要があります。このトリガーによってインポートの進行を妨げるような処理が定義されないように、インポートの後に再作成することをお薦めします。

- 表レベル・エクスポートおよびパーティション・レベル・エクスポート
  表、パーティションおよびサブパーティションをエクスポートできます。

親トピック: オリジナルのエクスポート

26.4.1 表レベル・エクスポートおよびパーティション・レベル・エクスポート

表、パーティションおよびサブパーティションをエクスポートできます。

どのモードでも、パーティションデータは、パーティションまたはサブパーティションを選択してインポートできる形式でエクスポートされます。

- 表レベル・エクスポート
  指定された表からすべてのデータをエクスポートします。
- パーティション・レベル・エクスポート
  指定されたソース・パーティションまたはサブパーティションのデータのみエクスポートします。

親トピック: エクスポート・モード

26.4.1.1 表レベル・エクスポート

指定した表からすべてのデータをエクスポートします。

表レベル・エクスポートでは、その索引や他の表依存オブジェクトとともに表(パーティション表および非パーティション表)全体をエクスポートできます。表がパーティション化されている場合は、そのすべてのパーティションとサブパーティションもエクスポートされます。これは、ダイレクト・パス・エクスポートと従来型パス・エクスポートの両方に当てはまります。どのエクスポート・モードでも、表レベル・エクスポートを実行できます。

親トピック: 表レベル・エクスポートおよびパーティション・レベル・エクスポート

26.4.1.2 パーティション・レベル・エクスポート

指定されたソース・パーティションまたはサブパーティションのデータのみエクスポートします。

パーティション・レベル・エクスポートでは、表内の指定された1つ以上のパーティションまたはサブパーティションをエクスポートできます。パーティション・レベル・エクスポートは、表モードでのみ実行できます。

表レベルおよびパーティション・レベルのエクスポートを指定する方法の詳細は、「TABLES」を参照してください。

親トピック: 表レベル・エクスポートおよびパーティション・レベル・エクスポート

26.5 エクスポート・パラメータ

エクスポート・コマンドライン・パラメータについて説明します。

- BUFFER
- COMPRESS
- CONSISTENT
- CONSTRAINTS
- DIRECT
  エクスポート・ユーティリティのDIRECTパラメータでは、ダイレクト・パス・エクスポートを使用することを指定します。
- FEEDBACK
  エクスポート・ユーティリティのFEEDBACKパラメータでは、エクスポート・ユーティリティにプログレス・バー(n行分のエクスポートを1つのピリオドで示す)を表示するように指定します。
- FILE
  FILEエクスポート・ユーティリティ・パラメータでは、エクスポート・ダンプ・ファイルの名前を指定します。
- FILESIZE
  エクスポート・ユーティリティのFILESIZEパラメータでは、ダンプ・ファイルのサイズを指定します。
- FLASHBACK_SCN
  FLASHBACK_SCNエクスポート・ユーティリティ・パラメータでは、エクスポート・ユーティリティでフラッシュバックを有効にするために使用するシステム変更番号(SCN)を指定します。
- FLASHBACK_TIME
  FLASHBACK_TIMEエクスポート・ユーティリティ・パラメータを使用すると、タイムスタンプを指定できます。エクスポート・ユーティリティは、指定されたタイムスタンプに最も近いSCNを検索します。このSCNを使用して、フラッシュバックを使用可能にします。
- FULL
  FULLエクスポート・パラメータは、エクスポートが全データベース・モード・エクスポートであることを示します(つまり、データベース全体をエクスポート)。
- GRANTS
  GRANTSエクスポート・ユーティリティ・パラメータでは、エクスポート・ユーティリティによりオブジェクト権限をエクスポートするかどうかを指定します。
- HELP
  エクスポート・ユーティリティのHELPパラメータを使用すると、エクスポート・ユーティリティのパラメータの説明が表示されます。
- INDEXES
  エクスポート・ユーティリティのINDEXESパラメータでは、エクスポート・ユーティリティで索引をエクスポートするかどうかを指定します。
- LOG
  情報メッセージとエラー・メッセージを受け取るファイル名(export.logなど)を指定します。
- OBJECT_CONSISTENT
  エクスポート・ユーティリティでSET TRANSACTION READ ONLY文を使用するかどうかを指定します。これにより、エクスポートされたデータには特定の時点における一貫性があり、エクスポート中に変更されないことが保証されます。
- OWNER
  エクスポートがユーザー・モード・エクスポートであることを指示して、エクスポートするオブジェクトの所有ユーザーを表示します。
- PARFILE
  エクスポート・ユーティリティのパラメータのリストを格納するファイルのファイル名を指定します。
- QUERY

このパラメータを使用すると、表モード・エクスポートの実行時に、表のセットから行のサブセットを選択できます。

- RECORDLENGTH
  ファイル・レコードの長さをバイト単位で指定します。
- RESUMABLE
  RESUMABLEパラメータは、再開可能な領域割当てを有効または無効にする場合に使用します。
- RESUMABLE_NAME
  RESUMABLE_NAMEパラメータの値によって、再開可能な文を識別します。
- RESUMABLE_TIMEOUT
  RESUMABLE_TIMEOUTパラメータの値では、エラー修正までに必要な時間を指定します。
- ROWS
  表のデータ行をエクスポートするかどうかを指定します。
- STATISTICS
  エクスポートしたデータのインポート時に生成するデータベース・オプティマイザ統計のタイプを指定します。オプションは、ESTIMATE、COMPUTEおよびNONEです。
- TABLES
- TABLESPACES
  TABLESPACESパラメータでは、指定した表領域内のすべての表をエクスポート・ダンプ・ファイルにエクスポートすることを指定します。
- TRANSPORT_TABLESPACE
  このパラメータにyを指定すると、トランスポータブル表領域のメタデータをエクスポートできるようになります。
- TRIGGERS
  エクスポート・ユーティリティでトリガーをエクスポートするかどうかを指定します。
- TTS_FULL_CHECK
  TTS_FULL_CHECKをyに設定すると、エクスポート・ユーティリティは、リカバリ・セット(リカバリする一連の表領域)がリカバリ・セットの外部のオブジェクトへの依存性(特に、INポインタ)を持たないこと(その逆も)を検証します。
- USERID (ユーザー名/パスワード)
  エクスポートを実行するユーザーのユーザー名、パスワードおよびオプションの接続文字列を指定します。
- VOLSIZE
  各テープ・ボリュームのエクスポート・ファイルに最大バイト数を指定します。

親トピック: オリジナルのエクスポート

26.5.1 BUFFER

デフォルト: オペレーティング・システムによって異なります。このパラメータのデフォルト値については、ご使用のオペレーティング・システム固有のOracleマニュアルを参照してください。

行のフェッチに使用するバッファのサイズをバイト単位で指定します。その結果、エクスポート・ユーティリティによってフェッチされる配列内の行の最大数が決まります。次の式を使用して、バッファ・サイズを計算します。

```
buffer_size = rows_in_array * maximum_row_size
```

ゼロを指定すると、1回に1行のみをフェッチします。

LOB型、LONG型、BFILE型、REF型、ROWID型、LOGICAL ROWID型またはDATE型の列を持つ表は、1回に1行ずつフェッチされます。

ノート:

BUFFER パラメータは、従来型パス・エクスポートのみに適用されます。ダイレクト・パス・エクスポートには影響しません。ダイレクト・パス・エクスポートの場合は、RECORDLENGTH パラメータを使用して、エクスポート・ユーティリティでエクスポート・ファイルへの書き込みに使用されるバッファのサイズを指定します。

- 例: バッファ・サイズの計算

親トピック: エクスポート・パラメータ

26.5.1.1 例: バッファ・サイズの計算

この項では、バッファ・サイズを計算する方法の例を示します。

次の表が作成されます。

```
CREATE TABLE sample (name varchar(30), weight number);
```

name列の最大サイズは、30バイトにインジケータ用の2バイトを加えた値です。weight列の最大サイズは、22バイト(Oracle数値の内部表現のサイズ)にインジケータ用の2バイトを加えた値です。

したがって、最大行サイズは56(30+2+22+2)です。

100行の配列操作を実行するには、バッファ・サイズを5600に指定する必要があります。

親トピック: BUFFER

26.5.2 COMPRESS

デフォルト: y

エクスポート・ユーティリティおよびインポート・ユーティリティで、表データの初期エクステントをどのように管理するかを指定します。

エクスポート・ユーティリティでは、デフォルトによりCOMPRESS=yによって、インポート時に1つの初期エクステントに統合するためのフラグが表データに設定されます。エクステント・サイズが大きいと(たとえば、PCTINCREASEパラメータが使用されているため)、割り当てられる領域がデータの保持に必要な領域よりも大きくなります。

COMPRESS=nを指定すると、エクスポート・ユーティリティでは初期エクステント・サイズと次のエクステント・サイズの値を含めた現在の記憶域パラメータが使用されます。パラメータの値には、CREATE TABLE文またはALTER TABLE文で指定された値、あるいはデータベース・システムによって変更された値を使用できます。たとえば、NEXTエクステント・サイズの値は、表が大きくなった場合、およびPCTINCREASEパラメータがゼロ以外の値の場合に変更できます。

COMPRESSパラメータは、ビットマップ化表領域では機能しません。

ノート:

実際の統合はインポート時に実行されますが、COMPRESS パラメータを指定できるのは、インポート時ではなくエクスポート時のみです。インポート・ユーティリティではなくエクスポート・ユーティリティによって、記憶域パラメータ定義などのデータ定義が生成されます。したがって、エクスポート時に COMPRESS=y を指定した場合は、統合形式でのみデータをインポートできます。

ノート:

LOB データもサブパーティション・データも圧縮されません。エクスポート時の初期エクステント・サイズと次のエクステント・サイズの値が使用されます。

親トピック: エクスポート・パラメータ

26.5.3 CONSISTENT

デフォルト: n

エクスポート・ユーティリティから見たデータが単一時点に対して一貫性がありexpコマンドの実行時に変更されないことを保証するために、エクスポート・ユーティリティでSET TRANSACTION READ ONLY文を使用するかどうかを指定します。エクスポートの開始後に、他のアプリケーションによるターゲット・データの更新が見込まれる場合は、CONSISTENT=yを指定する必要があります。

CONSISTENT=nを使用する場合、通常は1つのトランザクションでそれぞれの表がエクスポートされます。ただし、表にネストされた表が含まれている場合は、外部表とそれぞれの内部表が個別のトランザクションとしてエクスポートされます。表がパーティション化されている場合は、それぞれのパーティションが個別のトランザクションとしてエクスポートされます。

したがって、ネストされた表およびパーティション化された表が他のアプリケーションによって更新されると、エクスポートされるデータに不整合が生じる可能性があります。この可能性を最小限に抑えるには、更新が行われていないときにそれらの表をエクスポートします。

表26-2に、2人のユーザーによるイベントの順序を示します。user1は、表のパーティションをエクスポートし、user2はその表内のデータを更新します。

表26-2 2人のユーザーによる更新時のイベント順序

時間順序	user1	user2
1	表:P1 のエクスポートを開始	アクティビティなし
2	アクティビティなし	表:P2 を更新 表:P1 を更新 トランザクションをコミット
3	表:P1 のエクスポートを終了	アクティビティなし
4	表:P2 をエクスポート	アクティビティなし

エクスポートでCONSISTENT=yを使用すると、user2による更新はエクスポート・ファイルに書き込まれません。

エクスポートでCONSISTENT=nを使用すると、表:P1に対する更新はエクスポート・ファイルに書き込まれません。ただし、表:P2に対する更新はエクスポート・ファイルに書き込まれます。これは、更新トランザクションが表:P2のエクスポート開始前にコミットされるためです。その結果、user2のトランザクションが部分的にのみエクスポート・ファイルに記録されて一貫性がなくなります。

CONSISTENT=yを使用し、更新のボリュームが大きい場合は、ロールバック・セグメントの使用量が大きくなります。また、それぞれの表のエクスポート速度が低下します。これはそのロールバック・セグメントで、コミットされていないトランザクションをスキャン

する必要があるためです。

CONSISTENT=yの使用については、次のことに注意してください。

- ユーザーSYSとして接続しているか、AS SYSDBAを使用している(あるいはその両方)ときに実行されるエクスポートでは、CONSISTENT=yはサポートされません。
- ある特定のメタデータのエクスポートでは、再帰的SQL内でSYSスキーマを使用することが必要になる場合があります。そのような場合は、CONSISTENT=yを使用しても無視されます。CONSISTENT=yが選択されているエクスポート・プロセスの実行時は、メタデータの変更を行わないことをお薦めします。
- そのようなエクスポートに必要な時間と領域を最小限に抑えるには、一貫性を維持する必要がある表を、一貫性を維持する必要のない表とは別にエクスポートする必要があります。たとえば、一貫性エクスポートでemp表およびdept表を一緒にエクスポートし、データベースの他の部分を2番目のパスでエクスポートします。
- ロールバック領域を使い切り、コミットされたトランザクションで使用された領域を新しいトランザクションに再利用すると、スナップショットが古すぎるというエラーが発生します。ロールバック・セグメントで領域を再利用すると、最小限の領域要件でデータベースの一貫性を維持できますが、読取り一貫性イメージを保持できる時間は制限されます。

  コミットされたトランザクションが上書きされ、データベースの読取り一貫性ビューに情報が必要になると、スナップショットが古すぎることによるエラーが発生します。

  このエラーを回避するには、読取り一貫性エクスポートにかかる時間を最小限に抑える必要があります。(これを行うには、エクスポートされるオブジェクトの数を制限し、可能な場合は、データベースのトランザクション・レートを低下させます。)また、ロールバック・セグメントを可能なかぎり大きくします。

> ノート:
>
> ロールバック・セグメントは将来の Oracle Database リリースでは非推奨になります。かわりに、自動 UNDO 管理を使用することをお薦めします。

関連項目:

「OBJECT_CONSISTENT」

親トピック: エクスポート・パラメータ

26.5.4 CONSTRAINTS

デフォルト: y

エクスポート・ユーティリティで表の制約をエクスポートするかどうかを指定します。

親トピック: エクスポート・パラメータ

26.5.5 DIRECT

エクスポート・ユーティリティのDIRECTパラメータでは、ダイレクト・パス・エクスポートを使用することを指定します。

デフォルト

n

```
DIRECT=[Y|N]
```

エクスポート・ユーティリティでDIRECT=yを指定すると、データを直接読み取って、SQLコマンド処理レイヤー(評価バッファ)をバイパスすることでデータが抽出されます。この方式は、従来型パス・エクスポートよりも非常に高速になります。

エクスポートのプロシージャには、セキュリティとパフォーマンスに関する考慮事項があります。関連トピックを参照してください。

関連トピック

- ダイレクト・パス・エクスポートの開始

親トピック: エクスポート・パラメータ

26.5.6 FEEDBACK

エクスポート・ユーティリティのFEEDBACKパラメータでは、エクスポート・ユーティリティにプログレス・バー(n行分のエクスポートを1つのピリオドで示す)を表示するように指定します。

デフォルト: 0 (ゼロ)

n行分のエクスポートを1つのピリオドで示すプログレス・バーの表示を指定します。たとえば、FEEDBACK=10を指定すると、10行分のエクスポートが終了するたびにピリオドが1つ表示されます。FEEDBACK値は、エクスポートされるすべての表に適用されるため、各表に対して個別には設定できません。

親トピック: エクスポート・パラメータ

26.5.7 FILE

FILEエクスポート・ユーティリティ・パラメータでは、エクスポート・ダンプ・ファイルの名前を指定します。

デフォルト: expdat.dmp

エクスポート・ダンプ・ファイル名を指定します。デフォルトの拡張子は、.dmpですが、任意の拡張子を指定できます。エクスポート・ユーティリティは複数のエクスポート・ファイルをサポートするので、使用する複数のファイル名を指定できます。たとえば:

```
exp scott FILE = dat1.dmp, dat2.dmp, dat3.dmp FILESIZE=2048
```

エクスポート・ユーティリティは、最大FILESIZEに指定された値に到達すると、現在のファイルへの書込みを停止し、FILEパラメータによって指定された次の名前を持つ別のエクスポート・ファイルを開いて、完了するかFILESIZEの最大値に再度到達するまで操作を続行します。エクスポートを完了するために十分なエクスポート・ファイル名を指定しないと、エクスポート・ユーティリティによって、追加のファイル名を指定するように要求されます。

親トピック: エクスポート・パラメータ

26.5.8 FILESIZE

エクスポート・ユーティリティのFILESIZEパラメータでは、ダンプ・ファイルのサイズを指定します。

デフォルト: データは、表26-3で指定されているように、最大サイズになるまで1つのファイルに書き込まれます。

エクスポート・ユーティリティでは、複数のエクスポート・ファイルへの書込みがサポートされており、インポート・ユーティリティでは、複数のエクスポート・ファイルからの読取りが可能です。FILESIZEパラメータの値(バイト制限)を指定すると、エクスポート・ユーティリティでは、それぞれのダンプ・ファイルに指定したバイト数のみが書き込まれます。

エクスポート・ユーティリティは、書き込む必要があるデータの量がFILESIZEに指定された最大値を超えていると、FILEパラメータから次のエクスポート・ファイルの名前を取得するか、FILEパラメータに指定された名前をすべて使用した場合は、新しいエクスポート・ファイルの名前を指定するように要求します。FILESIZEに値を指定しない場合(値0はFILESIZEを指定しないことに相当)、エクスポート・ユーティリティはFILEパラメータに指定されたファイル数に関係なく1つのファイルのみに書き込みます。

ノート:

エクスポート・ファイルの領域要件が使用可能なディスク領域を超えていると、エクスポート・ユーティリティは終了します。十分な使用可能ディスク領域を有効にした後で、エクスポート・ユーティリティを繰り返す必要があります。

FILESIZEパラメータの最大値は、64ビットで格納できる最大値と同じです。

表26-3は、ダンプ・ファイルの最大サイズが、使用しているオペレーティング・システムおよび使用しているOracle Databaseによって異なることを示しています。

表26-3 ダンプ・ファイルの最大サイズ

オペレーティング・システム	Oracle Databaseのリリース	最大サイズ
任意	8.1.5 より前	2GB
32 ビット	8.1.5	2GB
64 ビット	8.1.5 以上	無制限
32 ビットで 32 ビットファイルを使用	任意	2GB
32 ビットで 64 ビットファイルを使用	8.1.6 以上	無制限

ファイルに格納できる最大値は、オペレーティング・システムによって異なります。この最大値については、FILESIZEを指定する前に、ご使用のオペレーティング・システム固有のOracleマニュアルで確認してください。エクスポート・ユーティリティ用に指定するファイル・サイズが、インポート・ユーティリティを実行するシステムでサポートされていることも確認する必要があります。

FILESIZEの値は、数字にKB(キロバイト数)を付けて指定することもできます。たとえば、FILESIZE=2KBは、FILESIZE=2048と同じです。同様に、MBはメガバイト(1024×1024)を、GBはギガバイト(1024の3乗)を表します。Bはバイトの省略です。この場合、最終的なファイル・サイズの算出に乗算は不要です(FILESIZE=2048Bは、FILESIZE=2048と同じです)。

親トピック: エクスポート・パラメータ

26.5.9 FLASHBACK_SCN

FLASHBACK_SCNエクスポート・ユーティリティ・パラメータでは、エクスポート・ユーティリティでフラッシュバックを有効にするために使用するシステム変更番号(SCN)を指定します。

デフォルト: なし

エクスポート・ユーティリティでフラッシュバックを有効にするために使用するシステム変更番号(SCN)を指定します。エクスポート

操作は、この指定されたSCNにおけるデータの一貫性を維持したまま実行されます。

関連項目:

- フラッシュバック・リカバリの実行方法の詳細は、『Oracle Databaseバックアップおよびリカバリ・ユーザーズ・ガイド』を参照してください。

次に、SCNの指定例を示します。このエクスポートを実行すると、SCN 3482971におけるデータの一貫性が維持されます。

```
> exp FILE=exp.dmp FLASHBACK_SCN=3482971
```

親トピック: エクスポート・パラメータ

26.5.10 FLASHBACK_TIME

FLASHBACK_TIMEエクスポート・ユーティリティ・パラメータを使用すると、タイムスタンプを指定できます。エクスポート・ユーティリティは、指定されたタイムスタンプに最も近いSCNを検索します。このSCNを使用して、フラッシュバックを使用可能にします。

デフォルト: なし

タイムスタンプを指定できます。エクスポート・ユーティリティは、指定されたタイムスタンプに最も近いSCNを検索します。このSCNを使用して、フラッシュバックを使用可能にします。エクスポート操作は、このSCNにおけるデータの一貫性を維持したまま実行されます。

DBMS_FLASHBACK.ENABLE_AT_TIMEプロシージャで使用可能な形式で時刻を指定できます。つまり、次のいずれかの方法で指定できます。

```
> exp FILE=exp.dmp FLASHBACK_TIME="TIMESTAMP '2006-05-01 11:00:00'"
> exp FILE=exp.dmp FLASHBACK_TIME="TO_TIMESTAMP('12-02-2005 14:35:00', 'DD-MM-YYYY
HH24:MI:SS')"
```

また、次の例に示す古い形式も下位互換性を維持するために引き続き指定できます。

```
> exp FILE=exp.dmp FLASHBACK_TIME="'2006-05-01 11:00:00'"
```

関連項目:

- フラッシュバック・リカバリの実行方法の詳細は、『Oracle Databaseバックアップおよびリカバリ・ユーザーズ・ガイド』を参照してください。
- DBMS_FLASHBACK PL/SQLパッケージの詳細は、『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』を参照してください。

親トピック: エクスポート・パラメータ

26.5.11 FULL

FULLエクスポート・パラメータは、エクスポートが全データベース・モード・エクスポートであることを示します(つまり、データベース全体をエクスポート)。

デフォルト: n

エクスポートが全データベース・モード・エクスポートであることを示します(つまり、データベース全体をエクスポート)。FULL=yを指

定して、全データベース・モードでエクスポートします。このモードでエクスポートするには、EXP_FULL_DATABASEロールが必要です。

- 全データベースのエクスポートおよびインポートについての考慮点
  全データベースのエクスポートおよびインポートは、データベースの複製およびクリーン・アップに有効です。

親トピック: エクスポート・パラメータ

26.5.11.1 全データベースのエクスポートおよびインポートについての考慮点

全データベースのエクスポートおよびインポートは、データベースの複製およびクリーンアップに有効です。

ただし、問題が発生しないように、次の点に注意する必要があります。

- 全体エクスポートでは、スキーマSYSが所有するトリガーはエクスポートされません。SYSトリガーは、全体インポートの前後のいずれかに手動で再作成する必要があります。このトリガーによってインポートの進行を妨げるような処理が定義されないように、インポートの後に再作成することをお薦めします。
- 全体エクスポートでは、デフォルトのプロファイルもエクスポートされません。ソース・データベースでデフォルトのプロファイルを変更した場合(たとえば、スキーマSYSが所有するパスワード確認機能を追加するなど)は、事前に手動でその機能を作成し、インポートの完了後にターゲット・データベースでデフォルトのプロファイルを変更する必要があります。
- 可能な場合、開始前にエクスポートされたデータベースおよびインポートするデータベースの物理コピーを作成します。これによって、誤った操作をやり直すことができます。
- エクスポートの開始前に、次の情報を含むレポートを作成することをお薦めします。
  - 表領域およびデータ・ファイルのリスト
  - ロールバック・セグメントのリスト
  - ユーザーごとの各オブジェクト型(表、索引など)の数

  この情報で、表領域がすでに作成され、インポートが正常に完了したことを確認できます。
- エクスポートから完全に新しいデータベースを作成する場合、SYSTEMに余分なロールバック・セグメントを作成し、初期化パラメータ・ファイル(`init.ora`)で使用可能にしてからインポートを開始します。
- インポートの実行時に、正しいインスタンスを指していることを確認します。一部のUNIXシステムでは、サブシェルの入力のみで、インポート操作を行ったデータベースが変更される場合があります。
- すべての表領域が作成されるまで、複数のデータベースを含むシステムでは全体インポートを行わないでください。全体インポートは、エクスポートされたデータベースと同じデータ・ファイル名を使用して、未定義の表領域を作成します。このため、次のような状況で問題が発生します。
  - データ・ファイルが他のいずれかのデータベースに属する場合、データ・ファイルは破損します。このことは特に、エクスポートされるデータベースが同じシステムに存在する場合に当てはまります(そのデータ・ファイルがインポート先のデータベースによって再利用されるため)。
  - データ・ファイルの名前が既存のオペレーティング・システム・ファイルの名前と同じ場合も問題が発生します。

親トピック: FULL

26.5.12 GRANTS

GRANTSエクスポート・ユーティリティ・パラメータでは、エクスポート・ユーティリティによりオブジェクト権限をエクスポートするかどう

かを指定します。

デフォルト: y

エクスポート・ユーティリティでオブジェクト権限をエクスポートするかどうかを指定します。エクスポートされるオブジェクト権限は、全データベースとユーザー・モードのどちらを使用するかによって異なります。全データベース・モードでは、表のすべての権限がエクスポートされます。ユーザー・モードでは、表の所有者によって付与されている権限のみがエクスポートされます。システム権限は常にエクスポートされます。

親トピック: エクスポート・パラメータ

26.5.13 HELP

エクスポート・ユーティリティのHELPパラメータを使用すると、エクスポート・ユーティリティのパラメータの説明が表示されます。

デフォルト: なし

エクスポート・パラメータの説明を表示します。ヘルプの内容を表示するには、コマンドラインでexp help=yを入力します。

親トピック: エクスポート・パラメータ

26.5.14 INDEXES

エクスポート・ユーティリティのINDEXESパラメータでは、エクスポート・ユーティリティで索引をエクスポートするかどうかを指定します。

デフォルト: y

エクスポート・ユーティリティで索引をエクスポートするかどうかを指定します。

親トピック: エクスポート・パラメータ

26.5.15 LOG

情報メッセージおよびエラー・メッセージを受け取るファイル名(export.logなど)を指定します。

デフォルト: なし

このパラメータを指定すると、メッセージがログ・ファイルに記録されて、端末画面に表示されます。

親トピック: エクスポート・パラメータ

26.5.16 OBJECT_CONSISTENT

エクスポートされたデータが単一時点に対して一貫性がありエクスポート中に変更されないことを保証するために、エクスポート・ユーティリティでSET TRANSACTION READ ONLY文を使用するかどうかを指定します。

デフォルト: n

OBJECT_CONSISTENTをyに設定すると、パーティション化されている場合でも、自身の読取り専用トランザクションで各オブジェクトがエクスポートされます。一方、CONSISTENTパラメータを使用した場合は、1つの読取り専用トランザクションのみが存在します。

関連項目:

CONSISTENT

親トピック: エクスポート・パラメータ

26.5.17 OWNER

エクスポートがユーザー・モード・エクスポートであることと、オブジェクトがエクスポートされるユーザーを指定します。

デフォルト: なし

エクスポートを開始するユーザーがデータベース管理者(DBA)の場合は、複数のユーザーをリストできます。

ユーザー・モード・エクスポートを使用して、1人以上のデータベース・ユーザーをバックアップできます。たとえばDBAは、削除されたユーザーの表を一定期間バックアップすることが必要な場合があります。ユーザー・モードは、自分のデータをバックアップする必要があるユーザー、または1人の所有者から別の所有者にオブジェクトを移動する必要があるユーザーにも適しています。

親トピック: エクスポート・パラメータ

26.5.18 PARFILE

エクスポート・パラメータのリストを格納するファイルのファイル名を指定します。

デフォルト: なし

パラメータ・ファイルの使用方法の詳細は、「エクスポート・ユーティリティの起動」を参照してください。

親トピック: エクスポート・パラメータ

26.5.19 QUERY

このパラメータでは、表モード・エクスポートの実行時に、表のセットから行のサブセットを選択できます。

デフォルト

ありません。

用途

問合せパラメータの値は、TABLESパラメータにリストされたすべての表(または表のパーティション)に適用されるSQL SELECT文のWHERE句が含まれている文字列です。

たとえば、ユーザーscottが、肩書きがSALESMANで給与が1600未満の従業員のみをエクスポートする場合は、次のように指定できます(この例はUNIXに基づいています)。

```
exp scott TABLES=emp QUERY=¥"WHERE job=¥'SALESMAN¥' and sal ¥<1600¥"
```

> ノート:
>
> QUERY パラメータの値には空白が含まれるため、ほとんどのオペレーティング・システムでは、WHERE job=¥'SALESMAN¥'および sal¥<1600 の文字列全体を二重引用符で囲むか、なんらかの方法でリテラルとしてマークする必要があります。オペレーティング・システムで予約されている文字も、エスケープ文字を前に付ける必要があります。システムの特殊文字および予約文字の詳細は、ご使用のオペレーティング・システム固有の Oracle マニュアルを参照してください。

この問合せの実行中に、エクスポート・ユーティリティでは次のようなSQL SELECT文が構築されます。

```
SELECT * FROM emp WHERE job='SALESMAN' and sal <1600;
```

QUERYパラメータに指定された値は、TABLESパラメータにリストされているすべての表(または表のパーティション)に適用されます。たとえば、次の文では問合せに一致するempとbonusの両方にある行がアンロードされます。

```
exp scott TABLES=emp,bonus QUERY=¥"WHERE job=¥'SALESMAN¥' and sal¥<1600¥"
```

また、エクスポート・ユーティリティでは次のようなSQL文が実行されます。

```
SELECT * FROM emp WHERE job='SALESMAN' and sal <1600;
SELECT * FROM bonus WHERE job='SALESMAN' and sal <1600;
```

表にQUERY句で指定された列が見つからないと、エラー・メッセージが生成され、その不適切な表については行がエクスポートされません。

- QUERYパラメータを使用する際の制限事項
  QUERYパラメータを使用する際の制限事項について説明します。

親トピック: エクスポート・パラメータ

26.5.19.1 QUERYパラメータを使用する際の制限事項

QUERYパラメータを使用する際の制限事項について説明します。

- QUERYパラメータは、全モード、ユーザー・モードまたは表領域モードのエクスポートには指定できません。
- QUERYパラメータは、指定されたすべての表に適用可能である必要があります。
- QUERYパラメータは、ダイレクト・パス・エクスポート(DIRECT=y)では指定できません。
- QUERYパラメータは、ネストされた内部表がある表には指定できません。
- データがQUERYエクスポートの結果かどうかを、エクスポート・ファイルの内容から判断することはできません。

親トピック: QUERY

26.5.20 RECORDLENGTH

ファイル・レコードの長さをバイト単位で指定します。

デフォルト

オペレーティング・システムによって異なります。

用途

RECORDLENGTHパラメータは、異なるデフォルト値を使用する別のオペレーティング・システムにエクスポート・ファイルを転送する場合に指定する必要があります。

このパラメータを指定しない場合、ご使用のプラットフォーム固有のバッファ・サイズの値がデフォルトになります。

RECORDLENGTHは、ご使用のシステムのバッファ・サイズの値以上の任意の値に設定できます。(最大値は64KBです。)RECORDLENGTHパラメータの変更により影響を受けるのは、ディスクに書き出す前に累積されるデータのサイズのみです。オペレーティング・システム・ファイルのブロック・サイズには影響しません。

ノート:

このパラメータは、エクスポート・ユーティリティの I/O バッファのサイズ指定に使用できます。

親トピック: エクスポート・パラメータ

26.5.21 RESUMABLE

RESUMABLEパラメータを使用して、再開可能な領域割当てを有効または無効にします。

デフォルト

n

用途

このパラメータはデフォルトでは無効なため、関連するRESUMABLE_NAMEおよびRESUMABLE_TIMEOUTを使用するには、RESUMABLE=yを設定する必要があります。

関連項目:

再開可能な領域割当ての詳細は、『Oracle Database管理者ガイド』を参照してください

親トピック: エクスポート・パラメータ

26.5.22 RESUMABLE_NAME

RESUMABLE_NAMEパラメータの値によって、再開可能な文を識別します。

デフォルト

```
'User USERNAME (USERID), Session SESSIONID, Instance INSTANCEID'
```

用途

この値はユーザー定義のテキスト文字列で、USER_RESUMABLEまたはDBA_RESUMABLEビューに挿入して、一時停止されている特定の再開可能な文を識別できます。

RESUMABLEパラメータをyに設定して、再開可能な領域割当てを有効にしないかぎり、このパラメータは無視されます。

親トピック: エクスポート・パラメータ

26.5.23 RESUMABLE_TIMEOUT

RESUMABLE_TIMEOUTパラメータの値では、エラー修正までに必要な時間を指定します。

デフォルト

7200秒(2時間)

用途

タイムアウト時間内にエラーを修正できない場合は、文の実行が途中で終了します。

RESUMABLEパラメータをyに設定して、再開可能な領域割当てを有効にしないかぎり、このパラメータは無視されます。

親トピック: エクスポート・パラメータ

26.5.24 ROWS

表のデータ行をエクスポートするかどうかを指定します。

デフォルト

y

親トピック: エクスポート・パラメータ

26.5.25 STATISTICS

エクスポートされたデータのインポート時に生成するデータベース・オプティマイザ統計のタイプを指定します。オプションは、ESTIMATE、COMPUTEおよびNONEです。

デフォルト: ESTIMATE

エクスポート・ユーティリティによって計算済の統計情報がエクスポート・ファイルに含まれることがあります。また、ANALYZE文によって統計情報が再生成されることもあります。

ただし、表にシステム生成名を持つ列がある場合、計算済オプティマイザ統計情報はエクスポート時に使用されません。

計算済オプティマイザ統計情報のエクスポート時には、次の場合に、問題ありというフラグが付きます。

- エクスポート中に行エラーが発生している
- クライアントの文字セットまたはNCHAR文字セットが、サーバーの文字セットまたはNCHAR文字セットと一致していない
- QUERY句が指定されている
- 特定のパーティションまたはサブパーティションのみがエクスポートされている

ノート:

ROWS=n を指定しても、計算済統計情報をエクスポート・ファイルに保存することはできます。この機能により、本番データベースからの統計情報を使用して、非本番データベース上で問合せの実行計画のチューニングを行うことができます。

親トピック: エクスポート・パラメータ

26.5.26 TABLES

デフォルト: なし

表モード・エクスポートでエクスポートすることを指定します。また、エクスポートの対象となる表名、パーティション名およびサブパーティション名を指定します。表の名前を指定するときは、次のように指定します。

- schemanameは、表またはパーティションのエクスポート元のユーザー・スキーマの名前を指定します。スキーマ名が指定されていない場合は、エクスポート実行者のスキーマがデフォルトとして使用されます。ORDSYS、MDSYS、CTXSYS、LBACSYS、ORDPLUGINSなどのシステム・スキーマ名はエクスポート・ユーティリティによって予約されています。
- tablenameは、エクスポートされる表の名前を指定します。表レベル・エクスポートでは、パーティション表または非パーティション表全体をエクスポートできます。リストにパーティション表が含まれる場合、パーティション名を指定しないと、すべてのパーティションおよびサブパーティションがエクスポートされます。

  表名は、任意の数の「%」パターン一致文字を含むことができます。各「%」パターン一致文字は、データベースの表オ

ブジェクトに対し表名の0個以上の文字と一致します。指定されたパターンに一致する関連のスキーマ内のすべての表が、それぞれの表名がパラメータで明示的に指定されているとおりにエクスポートに選択されます。

- partition_nameは、エクスポートがパーティション・レベルのエクスポートであることを示します。パーティション・レベル・エクスポートでは、表内の指定された1つ以上のパーティションまたはサブパーティションをエクスポートできます。

構文の形式は、次のとおりです。

```
schemaname.tablename:partition_name
schemaname.tablename:subpartition_name
```

tablename:partition_nameを使用する場合、指定された表はパーティション化されている必要があり、partition_nameはパーティションまたはサブパーティションのうちのいずれかの名前である必要があります。指定された表がパーティション化されていない場合、partition_nameは無視され、表全体がエクスポートされます。

パーティション・レベル・エクスポートの例については、「パーティション・レベル・エクスポートでのエクスポート・セッションの例」を参照してください。

- 表名の制限
  エクスポート・ユーティリティの表名の制限についての説明します。

親トピック: エクスポート・パラメータ

26.5.26.1 表名の制限

これは、エクスポート・ユーティリティの表名の制限についての説明です。

表名には次の制限があります。

- デフォルトでは、表名は大文字でデータベースに格納されます。表名が大文字と小文字または小文字のみで表記され、大/小文字を区別する場合、名前を引用符で囲む必要があります。したがって、表名は、データベースに格納されている表名と完全に一致するように指定する必要があります。

  ただし、オペレーティング・システムによっては、コマンドラインの引用符自体をエスケープする必要がある場合があります。次に、異なるエクスポート・モードで大/小文字の区別を保持する方法を示します。

  - コマンドライン・モード

    ```
    TABLES='¥"Emp¥"'
    ```

  - 対話方式モード

    ```
    Table(T) to be exported: "Emp"
    ```

  - パラメータ・ファイル・モード

    ```
    TABLES='"Emp"'
    ```

- 表名を引用符で囲まないかぎり、コマンドラインで指定する表名にポンド(#)記号は使用できません。同様に、パラメータ・ファイルでは、表名が引用符で囲まれていないかぎり、表名にポンド(#)記号を使用すると、エクスポート・ユーティリティではポンド(#)記号より右側の文字がコメントとして解釈されます。

  たとえば、パラメータ・ファイルに次の行が記述されている場合、エクスポート・ユーティリティはemp#の右側をすべてコメントとして解釈するため、表deptとmydataはエクスポートされません。

  ```
  TABLES=(emp#, dept, mydata)
  ```

ただし、次の例では、emp#が引用符で囲まれているため、3つの表はすべてエクスポートされます。

```
TABLES=("emp#", dept, mydata)
```

ノート:

オペレーティング・システムによっては、一重引用符を使用する必要がある場合と、二重引用符を使用する必要がある場合があります。表のネーミング方法に制限があるオペレーティング・システムもあります。

親トピック: TABLES

26.5.27 TABLESPACES

TABLESPACESパラメータは、指定された表領域内のすべての表をエクスポートのダンプ・ファイルにエクスポートすることを指定します。

デフォルト: なし

これには、指定した表領域に含まれるすべての表と、指定した表領域にあるパーティションを持つすべての表が含まれます。索引は、索引の格納場所に関係なくその表とともにエクスポートされます。

TABLESPACESを使用して表領域内のすべての表をエクスポートするには、EXP_FULL_DATABASEロールが必要です。

TABLESPACESとTRANSPORT_TABLESPACE=yを組み合せて使用するときは、データベースからエクスポート・ファイルにエクスポートする表領域の制限付きリストを指定できます。

親トピック: エクスポート・パラメータ

26.5.28 TRANSPORT_TABLESPACE

yを指定すると、トランスポータブル表領域のメタデータをエクスポートできるようになります。

デフォルト: n

暗号化された列は、トランスポータブル表領域モードではサポートされていません。

ノート:

トランスポータブル表領域をエクスポートした後、それよりも古いリリース・レベルのデータベースにインポートすることはできません。ターゲット・データベースのリリース・レベルは、ソース・データベース以上である必要があります。

関連項目:

- トランスポータブル表領域
- トランスポータブル表領域の詳細は、『Oracle Database管理者ガイド』を参照してください。

親トピック: エクスポート・パラメータ

26.5.29 TRIGGERS

エクスポート・ユーティリティでトリガーをエクスポートするかどうかを指定します。

デフォルト: y

親トピック: エクスポート・パラメータ

26.5.30 TTS_FULL_CHECK

TTS_FULL_CHECKをyに設定すると、エクスポート・ユーティリティは、リカバリ・セット(リカバリする一連の表領域)がリカバリ・セットの外部のオブジェクトへの依存性(特に、INポインタ)を持たないこと(その逆も)を検証します。

デフォルト: n

親トピック: エクスポート・パラメータ

26.5.31 USERID (username/password)

エクスポートを実行するユーザーのユーザー名、パスワードおよびオプションの接続文字列を指定します。

デフォルト: なし

パスワードを省略すると、指定するように要求されます。

ユーザーSYSとして接続する場合は、接続文字列にAS SYSDBAも指定する必要があります。オペレーティング・システムによっては、AS SYSDBAを特殊文字列とみなし、その文字列全体を引用符で囲む必要があります。

関連項目:

Oracle Netに接続文字列を指定する方法の詳細は、ご使用のOracle Netプロトコルのユーザーズ・ガイドを参照してください

親トピック: エクスポート・パラメータ

26.5.32 VOLSIZE

各テープ・ボリュームのエクスポート・ファイルに最大バイト数を指定します。

デフォルト: なし

VOLSIZEパラメータの最大値は、ご使用のプラットフォーム上に64ビットで格納できる最大値と同じです。

VOLSIZEの値は、数字にKB(キロバイト数)を付けて指定できます。たとえば、VOLSIZE=2KBは、VOLSIZE=2048と同じです。同様に、MBはメガバイト(1024×1024)を、GBはギガバイト(1024の3乗)を表します。Bはバイトの省略です。この場合、最終的なファイル・サイズの算出に乗算は不要です(VOLSIZE=2048Bは、VOLSIZE=2048と同じです)。

親トピック: エクスポート・パラメータ

26.6 エクスポート・セッションの例

エクスポート・セッションのタイプの例。

それぞれの例では、コマンドライン方式とパラメータ・ファイル方式の両方の使用方法が示されます。一部の例では、縦長の省略記号を使用して、長すぎてすべてを表示できない出力例のセクションを示しています。

- 全体データベース・モードのエクスポート・セッションの例
- ユーザー・モードのエクスポート・セッションの例

- 表モードのエクスポート・セッションの例
- パーティション・レベル・エクスポートでのエクスポート・セッションの例

親トピック: オリジナルのエクスポート

26.6.1 全体データベース・モードのエクスポート・セッションの例

DBAロールまたはEXP_FULL_DATABASEロールを所有するユーザーのみが全データベース・モードでのエクスポートを実行できます。この例では、データベース全体がすべてのGRANTSおよびデータとともにファイルdba.dmpにエクスポートされます。

パラメータ・ファイル方式

```
> exp PARFILE=params.dat
```

params.datファイルには、次の情報が格納されています。

```
FILE=dba.dmp
GRANTS=y
FULL=y
ROWS=y
```

コマンドライン方式

```
> exp FULL=y FILE=dba.dmp GRANTS=y ROWS=y
```

エクスポート・メッセージ

使用しているエクスポート・ユーティリティのリリース情報および接続しているOracle Databaseのリリース情報が表示されます。状態メッセージは、データベース全体がエクスポートされた時点で書き出されます。エクスポートが警告なしで完了すると、最終的な完了メッセージが返されます。

親トピック: エクスポート・セッションの例

26.6.2 ユーザー・モードのエクスポート・セッションの例

ユーザー・モード・エクスポートを使用して、1人以上のデータベース・ユーザーをバックアップできます。たとえばDBAは、削除されたユーザーの表を一定期間バックアップすることが必要な場合があります。ユーザー・モードは、自分のデータをバックアップする必要があるユーザー、または1人の所有者から別の所有者にオブジェクトを移動する必要があるユーザーにも適しています。この例では、ユーザーscottが自分の表をエクスポートします。

パラメータ・ファイル方式

```
> exp scott PARFILE=params.dat
```

params.datファイルには、次の情報が格納されています。

```
FILE=scott.dmp
OWNER=scott
GRANTS=y
ROWS=y
COMPRESS=y
```

コマンドライン方式

```
> exp scott FILE=scott.dmp OWNER=scott GRANTS=y ROWS=y COMPRESS=y
```

エクスポート・メッセージ

使用しているエクスポート・ユーティリティのリリース情報および接続しているOracle Databaseのリリース情報が表示されます。その後で、次に示すような状態メッセージが表示されます。

```
.
.
. about to export SCOTT's tables via Conventional Path ...
. . exporting table                          BONUS          0 rows exported
. . exporting table                           DEPT          4 rows exported
. . exporting table                            EMP         14 rows exported
. . exporting table                       SALGRADE          5 rows exported
.
.
.
Export terminated successfully without warnings.
```

親トピック: エクスポート・セッションの例

26.6.3 表モードのエクスポート・セッションの例

表モードでは、表データまたは表定義をエクスポートできます。(行がエクスポートされない場合は、CREATE TABLE文が権限および索引とともに(指定されている場合)エクスポート・ファイルに書き込まれます。)

EXP_FULL_DATABASEロールを持っているユーザーは、表モードを使用しTABLES=schemaname.tablenameを指定して、任意のユーザーのスキーマから表をエクスポートできます。

schemanameが指定されていない場合、デフォルトでエクスポート実行者のスキーマ名になります。次の例では、表aおよび表cは、デフォルトでSYSTEMスキーマになります。

```
> exp TABLES=(a, scott.b, c, mary.d)
```

EXP_FULL_DATABASEロールを持っているユーザーは、他のユーザーによって所有されている依存オブジェクトもエクスポートできます。権限のないユーザーは、所有している指定済の表の依存オブジェクトのみをエクスポートできます。

表モードでのエクスポートには、クラスタ定義は含まれません。その結果、データはクラスタ化されていない表としてエクスポートされます。したがって、表モードを使用して、表のクラスタ化を解除できます。

- 例1: DBAによる2人のユーザー用の表のエクスポート
- 例2: ユーザーによる自身が所有している表のエクスポート
- 例3: パターン一致を使用した様々な表のエクスポート

親トピック: エクスポート・セッションの例

26.6.3.1 例1: DBAによる2人のユーザー用の表のエクスポート

この例では、DBAが2人のユーザーの指定済表をエクスポートします。

パラメータ・ファイル方式

```
> exp PARFILE=params.dat
```

params.datファイルには、次の情報が格納されています。

```
FILE=expdat.dmp
TABLES=(scott.emp,blake.dept)
GRANTS=y
INDEXES=y
```

コマンドライン方式

```
> exp FILE=expdat.dmp TABLES=(scott.emp,blake.dept) GRANTS=y INDEXES=y
```

エクスポート・メッセージ

使用しているエクスポート・ユーティリティのリリース情報および接続しているOracle Databaseのリリース情報が表示されます。その後で、次に示すような状態メッセージが表示されます。

```
.
.
.
About to export specified tables via Conventional Path ...
Current user changed to SCOTT
. . exporting table                          EMP          14 rows exported
Current user changed to BLAKE
. . exporting table                         DEPT           8 rows exported
Export terminated successfully without warnings.
```

親トピック: 表モードのエクスポート・セッションの例

26.6.3.2 例2: ユーザーによる自身が所有している表のエクスポート

この例では、ユーザーblakeが所有している選択済の表をエクスポートします。

パラメータ・ファイル方式

```
> exp blake PARFILE=params.dat
```

params.datファイルには、次の情報が格納されています。

```
FILE=blake.dmp
TABLES=(dept,manager)
ROWS=y
COMPRESS=y
```

コマンドライン方式

```
> exp blake FILE=blake.dmp TABLES=(dept, manager) ROWS=y COMPRESS=y
```

エクスポート・メッセージ

使用しているエクスポート・ユーティリティのリリース情報および接続しているOracle Databaseのリリース情報が表示されます。その後で、次に示すような状態メッセージが表示されます。

```
.
.
.
About to export specified tables via Conventional Path ...
. . exporting table                         DEPT           8 rows exported
. . exporting table                      MANAGER           4 rows exported
Export terminated successfully without warnings.
```

親トピック: 表モードのエクスポート・セッションの例

26.6.3.3 例3: パターン一致を使用した様々な表のエクスポート

この例では、パターン一致を使用して、ユーザーscottおよびblakeの様々な表をエクスポートします。

パラメータ・ファイル方式

```
> exp PARFILE=params.dat
```

params.datファイルには、次の情報が格納されています。

```
FILE=misc.dmp
TABLES=(scott.%P%,blake.%,scott.%S%)
```

コマンドライン方式

```
> exp FILE=misc.dmp TABLES=(scott.%P%,blake.%,scott.%S%)
```

エクスポート・メッセージ

使用しているエクスポート・ユーティリティのリリース情報および接続しているOracle Databaseのリリース情報が表示されます。その後で、次に示すような状態メッセージが表示されます。

```
.
.
.
About to export specified tables via Conventional Path ...
Current user changed to SCOTT
. . exporting table                          DEPT          4 rows exported
. . exporting table                           EMP         14 rows exported
Current user changed to BLAKE
. . exporting table                          DEPT          8 rows exported
. . exporting table                       MANAGER          4 rows exported
Current user changed to SCOTT
. . exporting table                         BONUS          0 rows exported
. . exporting table                      SALGRADE          5 rows exported
Export terminated successfully without warnings.
```

親トピック: 表モードのエクスポート・セッションの例

26.6.4 パーティション・レベル・エクスポートでのエクスポート・セッションの例

パーティション・レベルのエクスポートでは、エクスポートする表のパーティションおよびサブパーティションを指定できます。

- 例1: パーティションを指定しない表のエクスポート
- 例2: パーティションを指定した表のエクスポート
- 例3: コンポジット・パーティションのエクスポート

親トピック: エクスポート・セッションの例

26.6.4.1 例1: パーティションを指定しない表のエクスポート

empは従業員名でパーティション化されている表であると想定します。mおよびzの2つのパーティションがあります。この例に示すとおり、パーティションを指定せずに表をエクスポートすると、すべてのパーティションがエクスポートされます。

パラメータ・ファイル方式

```
> exp scott PARFILE=params.dat
```

`params.dat`ファイルには、次の情報が格納されています。

```
TABLES=(emp)
ROWS=y
```

コマンドライン方式

```
> exp scott TABLES=emp rows=y
```

エクスポート・メッセージ

使用しているエクスポート・ユーティリティのリリース情報および接続しているOracle Databaseのリリース情報が表示されます。

その後で、次に示すような状態メッセージが表示されます。

```
.
.
.
About to export specified tables via Conventional Path ...
. . exporting table                          EMP
. . exporting partition                            M          8 rows exported
. . exporting partition                            Z          6 rows exported
Export terminated successfully without warnings.
```

親トピック: パーティション・レベル・エクスポートでのエクスポート・セッションの例

26.6.4.2 例2: パーティションを指定した表のエクスポート

empは従業員名でパーティション化されている表であると想定します。mおよびzの2つのパーティションがあります。この例に示すとおり、表をエクスポートするときにパーティションを指定すると、指定されたパーティションのみがエクスポートされます。

パラメータ・ファイル方式

```
 > exp scott PARFILE=params.dat
```

params.datファイルには、次の情報が格納されています。

```
TABLES=(emp:m)
ROWS=y
```

コマンドライン方式

```
> exp scott TABLES=emp:m rows=y
```

エクスポート・メッセージ

使用しているエクスポート・ユーティリティのリリース情報および接続しているOracle Databaseのリリース情報が表示されます。その後で、次に示すような状態メッセージが表示されます。

```
.
.
.
About to export specified tables via Conventional Path ...
. . exporting table                          EMP
. . exporting partition                            M          8 rows exported
Export terminated successfully without warnings.
```

親トピック: パーティション・レベル・エクスポートでのエクスポート・セッションの例

26.6.4.3 例3: コンポジット・パーティションのエクスポート

empは、2つのパーティションm、zがあるパーティション表であるとします。表empは、コンポジット方式を使用してパーティション化されています。パーティションmには、サブパーティションsp1およびsp2があり、パーティションzには、サブパーティションsp3およびsp4があります。この例に示すとおり、コンポジット・パーティションmをエクスポートすると、そのすべてのサブパーティション(sp1およびsp2)がエクスポートされます。表をエクスポートするときにサブパーティション(sp4)を指定すると、指定されたパーティションのみがエクスポートされます。

パラメータ・ファイル方式

```
> exp scott PARFILE=params.dat
```

params.datファイルには、次の情報が格納されています。

```
TABLES=(emp:m,emp:sp4)
ROWS=y
```

コマンドライン方式

```
> exp scott TABLES=(emp:m, emp:sp4) ROWS=y
```

エクスポート・メッセージ

使用しているエクスポート・ユーティリティのリリース情報および接続しているOracle Databaseのリリース情報が表示されます。その後で、次に示すような状態メッセージが表示されます。

```
.
.
.
About to export specified tables via Conventional Path ...
. . exporting table                          EMP
. . exporting composite partition              M
. . exporting subpartition                   SP1          1 rows exported
. . exporting subpartition                   SP2          3 rows exported
. . exporting composite partition              Z
. . exporting subpartition                   SP4          1 rows exported
Export terminated successfully without warnings.
```

親トピック: パーティション・レベル・エクスポートでのエクスポート・セッションの例

26.7 警告、エラーおよび完了メッセージ

これらの項では、エクスポート・ユーティリティによって発行される各種のメッセージと、ログ・ファイルへのメッセージの保存方法について説明します。

- ログ・ファイル
  すべてのエクスポート・メッセージをログ・ファイルに保存するには、LOGパラメータを使用するか、出力をファイルにリダイレクトします(リダイレクトできるシステムの場合)。
- 警告メッセージ
  エクスポート・ユーティリティは、リカバリ可能なエラーの発生後は終了しません。これらのリカバリ可能なエラーは警告と呼ばれます。
- リカバリ不能なエラー・メッセージ
  エラーによってはリカバリ不能で、エクスポート・セッションが終了する場合があります。
- 完了メッセージ
  エラーが発生することなくエクスポートが完了すると、その状態を示すメッセージが表示されます。

親トピック: オリジナルのエクスポート

26.7.1 ログ・ファイル

すべてのエクスポート・メッセージをログ・ファイルに保存するには、LOGパラメータを使用するか、出力をファイルにリダイレクトします(リダイレクトできるシステムの場合)。

成功したアンロード、および発生している可能性があるエラーに関して、詳細な情報のログが記録されます。

親トピック: 警告、エラーおよび完了メッセージ

26.7.2 警告メッセージ

エクスポート・ユーティリティは、リカバリ可能なエラーの発生後は終了しません。これらのリカバリ可能なエラーは警告と呼ばれます。

たとえば、表のエクスポート中にエラーが発生すると、エクスポート・ユーティリティはエラー・メッセージを表示(またはログに記録)して次の表にスキップし、処理を続行します。

エクスポート・ユーティリティは、無効なオブジェクトが検出された場合も警告メッセージを表示します。

たとえば、存在しない表が表モード・エクスポートの一部として指定されても、それ以外のすべての表をエクスポートします。その後で警告メッセージを表示して、正常に終了します。

親トピック: 警告、エラーおよび完了メッセージ

26.7.3 リカバリ不能なエラー・メッセージ

エラーによってはリカバリ不能で、エクスポート・セッションが終了する場合があります。

通常、これらのエラーは、内部的な問題、またはメモリーなどのリソースが使用不能か不足しているなどの原因によって発生します。たとえば、`catexp.sql`スクリプトが実行されない場合、エクスポート・ユーティリティは次のようなリカバリ不能メッセージを表示します。

```
EXP-00024: Export views not installed, please notify your DBA
```

親トピック: 警告、エラーおよび完了メッセージ

26.7.4 完了メッセージ

エラーが発生することなくエクスポートが完了すると、その状態を示すメッセージが表示されます。

たとえば:

```
Export terminated successfully without warnings
```

リカバリ可能なエラーが1つ以上発生した場合に、ジョブが続行されて完了すると、次のようなメッセージが表示されます。

```
Export terminated successfully with warnings
```

リカバリ不能なメッセージが発生してジョブがすぐに終了した場合は、その状態を示す次のようなメッセージが表示されます。

```
Export terminated unsuccessfully
```

親トピック: 警告、エラーおよび完了メッセージ

26.8 終了コードによる結果の検査と表示

エクスポート・ユーティリティでは、操作の完了後、すぐに実行結果を確認できます。プラットフォームによっては、実行結果はプロセス終了コードに通知され、ログ・ファイルにも記録されます。

これによって、コマンドラインやスクリプトからの出力を確認できます。表26-4に、様々な結果の終了コードを示します。

表26-4 エクスポート・ユーティリティの終了コード

結果	終了コード

結果	終了コード
エクスポートは警告なしで正常終了しました。	EX_SUCC
エクスポートは正常に終了しましたが、警告が発生しました。	EX_OKWARN
エラーが発生したためエクスポートを終了します。	EX_FAIL

UNIXの場合、終了コードは次のようになります。

```
EX_SUCC   0
EX_OKWARN 0
EX_FAIL   1
```

親トピック: オリジナルのエクスポート

26.9 従来型パス・エクスポートとダイレクト・パス・エクスポート

エクスポート・ユーティリティで表データをエクスポートするには、2つの方法があります。

具体的には次のとおりです。

- 従来型パス・エクスポート
- ダイレクト・パス・エクスポート

従来型パス・エクスポートでは、SQL SELECT文を使用して表からデータを抽出します。データはディスクからバッファ・キャッシュへと読み取られ、行は評価バッファに転送されます。データは、式の評価が渡された後にエクスポート・クライアントに転送され、クライアントによってエクスポート・ファイルに書き込まれます。

ダイレクト・パス・エクスポートは、従来型パス・エクスポートよりも非常に高速です。これは、データがディスクからバッファ・キャッシュに読み取られて、行が直接エクスポート・クライアントに転送されるためです。評価バッファ(SQLコマンド処理レイヤーに相当)はバイパスされます。データはすでにエクスポート・ユーティリティ向けの形式になっているため、不要なデータ変換を回避できます。データは、エクスポート・クライアントに転送され、クライアントによってエクスポート・ファイルに書き込まれます。

親トピック: オリジナルのエクスポート

26.10 ダイレクト・パス・エクスポートの開始

ダイレクト・パス・エクスポートを使用するには、コマンドラインまたはパラメータ・ファイルで、DIRECT=yパラメータを指定します。

デフォルトは、従来型パスを使用して表データを抽出するDIRECT=nです。ダイレクト・パス・エクスポート操作に関するセキュリティ、パフォーマンスおよび制限事項について確認してください。

- ダイレクト・パス・エクスポートに関するセキュリティ上の考慮点
  Oracle Virtual Private Database (VPD)およびOracle Label Securityは、ダイレクト・パス・エクスポートの実行時には施行されません。
- ダイレクト・パス・エクスポートに関するパフォーマンス上の考慮点
  ダイレクト・パス・エクスポートを起動するときに、RECORDLENGTHパラメータの値を大きくすることによるパフォーマンスの向上。
- ダイレクト・パス・エクスポートに関する制限事項

ダイレクト・パス・モードの使用に関する制限事項。

親トピック: オリジナルのエクスポート

26.10.1 ダイレクト・パス・エクスポートに関するセキュリティ上の考慮点

Oracle Virtual Private Database(VPD)およびOracle Label Securityは、ダイレクト・パス・エクスポートの実行時には施行されません。

次のユーザーは、データベースからのデータの抽出に使用されるエクスポート・モード、アプリケーションまたはユーティリティに関係なく、Virtual Private DatabaseおよびOracle Label Security施行の対象外となります。

- データベース・ユーザーSYS
- 直接またはデータベース・ロールを通じてEXEMPT ACCESS POLICY権限を付与されているデータベース・ユーザー

つまり、EXEMPT ACCESS POLICY権限を付与されているどのユーザーも、VPDおよびOracle Label Securityの施行の完全な対象外となります。これは強力な権限であるため、注意して管理する必要があります。この権限は、SELECT、INSERT、UPDATE、DELETEなどの従来のオブジェクト権限の施行に影響しません。これらの権限は、ユーザーにEXEMPT ACCESS POLICY権限が付与されている場合も施行されます。

関連項目:

- ファイングレイン・アクセス・コントロールに対するサポート
- 『Oracle Label Security管理者ガイド』
- VPDを使用してデータ・アクセスを制御する方法の詳細は、『Oracle Databaseセキュリティ・ガイド』を参照してください。

親トピック: ダイレクト・パス・エクスポートの開始

26.10.2 ダイレクト・パス・エクスポートに関するパフォーマンス上の考慮点

ダイレクト・パス・エクスポートを起動するときに、RECORDLENGTHパラメータの値を大きくすることによるパフォーマンスの向上。

実際に得られるパフォーマンスは、次の要因によって決まります。

- DB_BLOCK_SIZE
- 表内の列の型
- 使用するI/Oレイアウト(エクスポート・ファイルを受け取るドライブは、データベース・ファイルが存在するディスク・ドライブとは別にする必要があります。)

一般に、RECORDLENGTHには次の値が推奨されます。

- ファイル・システムI/Oブロック・サイズの整数倍
- DB_BLOCK_SIZEの整数倍

ダイレクト・パス・エクスポートを使用して作成されたエクスポート・ファイルをインポートするには、従来型パス・エクスポートを使用して作成されたエクスポート・ファイルの場合と同じ時間がかかります。

親トピック: ダイレクト・パス・エクスポートの開始

26.10.3 ダイレクト・パス・エクスポートに関する制限事項

ダイレクト・パス・モードの使用に関する制限事項。

具体的には次のとおりです。

- ダイレクト・パス・エクスポートを起動するには、コマンドライン方式またはパラメータ・ファイルを使用する必要があります。対話方式を使用して、ダイレクト・パス・エクスポートを起動することはできません。
- BUFFERエクスポート・パラメータは、従来型パス・エクスポートのみに適用されます。ダイレクト・パス・エクスポートの場合は、RECORDLENGTHパラメータを使用して、エクスポート・ユーティリティでエクスポート・ファイルへの書き込みに使用されるバッファのサイズを指定します。
- 表領域モード(TRANSPORT_TABLESPACES=Y)でエクスポートするときは、ダイレクト・パスを使用できません。
- QUERYパラメータは、ダイレクト・パス・エクスポートでは指定できません。
- ダイレクト・パス・エクスポートでデータをエクスポートできるのは、エクスポートを起動するセッションのNLS_LANG環境変数がデータベースの文字セットに一致する場合のみです。NLS_LANGが設定されていないか、データベースの文字セットと異なる場合は、警告メッセージが表示されて、エクスポートが続行されません。NLS_LANG環境変数のデフォルト値は、AMERICAN_AMERICA.US7ASCIIです。

親トピック: ダイレクト・パス・エクスポートの開始

26.11 オリジナルのOracle Data Pumpエクスポートのネットワーク考慮事項

ネットワークを通じてオリジナルのエクスポート(exp)ユーティリティを使用するときには、プロトコルと接続修飾文字列を確認してください。

- ネットワークを介してエクスポート・ファイルを転送する方法
  エクスポート・ファイルはバイナリ形式であるため、ネットワークを介してそのエクスポート・ファイルを転送するときは、バイナリ転送をサポートしているプロトコルを使用して、ファイルが破損しないようにしてください。
- Oracle Netを使用したエクスポート
  Oracle Netを使用すると、ネットワークを介してエクスポートを実行できます。

親トピック: オリジナルのエクスポート

26.11.1 ネットワークを介してエクスポート・ファイルを転送する方法

エクスポート・ファイルはバイナリ形式であるため、ネットワークを介してそのエクスポート・ファイルを転送するときは、バイナリ転送をサポートしているプロトコルを使用して、ファイルが破損しないようにしてください。

たとえば、FTPなどのファイル転送プロトコルを使用して、バイナリ・モードでファイルを転送します。エクスポート・ファイルをキャラクタ・モードで送信すると、ファイルのインポート時にエラーが発生します。

親トピック: オリジナルのOracle Data Pumpエクスポートのネットワーク考慮事項

26.11.2 Oracle Netを使用したエクスポート

Oracle Netを使用すると、ネットワークを介してエクスポートを実行できます。

たとえば、エクスポート・ユーティリティをローカルで実行した場合に、リモートのOracle Databaseからローカルのエクスポート・

ファイルにデータを書き込むことができます。

Oracle Netとともにエクスポート・ユーティリティを使用するには、expコマンドにユーザー名とパスワードを入力するときに接続修飾文字列@connect_stringを指定する必要があります。この句の構文の詳細は、ご使用のOracle Netプロトコルのユーザーズ・ガイドを参照してください。

関連項目:

- Oracle Database Net Services管理者ガイド

親トピック: オリジナルのOracle Data Pumpエクスポートのネットワーク考慮事項

26.12 文字セットおよびグローバリゼーション・サポートに関する考慮点

これらの項では、ユーザー・データおよびデータ定義言語(DDL)の文字セット変換に関連するエクスポート・ユーティリティのグローバリゼーション・サポートの動作について説明します。

- ユーザー・データ
  エクスポート・ユーティリティは、常に、エクスポート・サーバーの文字セットでUnicodeデータを含むユーザー・データをエクスポートします。(文字セットは、データベース作成時に指定されます。)
- データ定義言語(DDL)
  エクスポートおよびインポート操作時に、データ定義言語(DDL)に対して最大3回の文字セット変換が必要です。
- シングルバイト文字セットとエクスポートおよびインポート
  8ビット文字セットのエクスポート・ファイルをインポートすると、一部の8ビット・キャラクタが失われる(つまり7ビットの対応するキャラクタに変換される)場合があります。
- マルチバイト文字セットとエクスポートおよびインポート
  ターゲットの文字セットに同等の文字がないエクスポート・ファイル中の文字は、変換時にデフォルトの文字に置換されます。(デフォルトの文字は、ターゲット文字セットによって定義されます)。

親トピック: オリジナルのエクスポート

26.12.1 ユーザー・データ

エクスポート・ユーティリティは、常に、エクスポート・サーバーの文字セットでUnicodeデータを含むユーザー・データをエクスポートします。(文字セットは、データベース作成時に指定されます。)

ソース・データベースの文字セットが、インポート・データベースの文字セットと異なる場合、自動的にデータをインポート・サーバーの文字セットに変換するための文字セット変換が実行されます。

- 変換による文字セットのソート順への影響
  エクスポート文字セットのソート順が、インポート文字セットと異なる場合、キャラクタ列をパーティション化した表では、結果が保証されません。

親トピック: 文字セットおよびグローバリゼーション・サポートに関する考慮点

26.12.1.1 変換による文字セットのソート順への影響

エクスポート文字セットのソート順が、インポート文字セットと異なる場合、キャラクタ列をパーティション化した表では、結果が保証されません。

たとえば、次のようなASCII文字セットのデータベースの表定義について考えてみます。

```
CREATE TABLE partlist
   (
   part     VARCHAR2(10),
   partno   NUMBER(2)
   )
PARTITION BY RANGE (part)
  (
  PARTITION part_low VALUES LESS THAN ('Z')
    TABLESPACE tbs_1,
  PARTITION part_mid VALUES LESS THAN ('z')
    TABLESPACE tbs_2,
  PARTITION part_high VALUES LESS THAN (MAXVALUE)
    TABLESPACE tbs_3
  );
```

ASCII文字セットでは、Zの後にzがあるため、このパーティション・スキームには意味があります。

この表がEBCDIC文字セット・ベースのデータベースにインポートされると、EBCDIC文字セットでは、zがZの前にあるため、part_midパーティションのすべての行が、part_lowパーティションに移行します。希望する結果を得るには、partlist表の所有者は、インポート後に表を再パーティション化する必要があります。

関連項目:

文字セットの詳細は、『Oracle Databaseグローバリゼーション・サポート・ガイド』を参照してください。

親トピック: ユーザー・データ

26.12.2 DDL

エクスポートおよびインポート操作時に、データ定義言語(DDL)に対して最大3回の文字セット変換が必要です。

具体的には次のとおりです。

1. エクスポート・ファイルは、環境変数NLS_LANGでユーザー・セッション用に指定された文字セットで書き出されます。NLS_LANGの値が、データベースの文字セットと異なる場合は、文字セット変換が実行されます。
2. エクスポート・ファイルの文字セットが、インポート先ユーザー・セッション用の文字セットと異なる場合、ユーザー・セッションの文字セットに変換されます。シングルバイト文字セットの場合のみにこの変換が実行されます。マルチバイト文字セットの場合は、インポート・ファイルの文字セットがエクスポート・ファイルの文字セットと同じである必要があります。
3. ターゲット・データベースの文字セットが、インポート・ユーザー・セッション用の文字セットと異なる場合、最後の文字セット変換が実行される場合があります。

文字セット変換によるデータの損失を最小限にするには、エクスポート・データベース、エクスポート・ユーザー・セッション、インポート・ユーザー・セッションおよびインポート・データベースのすべてにおいて、同一の文字セットを使用するようにしてください。

親トピック: 文字セットおよびグローバリゼーション・サポートに関する考慮点

26.12.3 シングルバイト文字セットとエクスポートおよびインポート

8ビット文字セットのエクスポート・ファイルをインポートすると、一部の8ビット・キャラクタが失われる(つまり7ビットの対応するキャラクタに変換される)場合があります。

これが発生するのは、インポートを実行するシステムに、システム固有の7ビット文字セットが存在するか、オペレーティング・システム環境変数NLS_LANGが7ビット文字セットに設定されている場合です。アクセント記号が付いている文字からアクセントが消去されるのが最もよく見られる例です。

このような状況を回避するために、オペレーティング・システム環境変数NLS_LANGにエクスポート・ファイルの文字セットを設定できます。

親トピック: 文字セットおよびグローバリゼーション・サポートに関する考慮点

26.12.4 マルチバイト文字セットとエクスポートおよびインポート

ターゲット文字セットに同等の文字がないエクスポート・ファイル中の文字は、変換時にデフォルトの文字に置換されます(デフォルトの文字は、ターゲット文字セットによって定義されます)。(デフォルトの文字は、ターゲット文字セットによって定義されます)。

100%完全に変換されるためには、ターゲット文字セットはソース文字セットのスーパーセットであるか、ソース文字セットと同等である必要があります。

> ノート:
>
> エクスポート・サーバーとインポート・サーバーとの間で文字セットの幅が異なるとき、変換が原因でデータの拡張が発生する場合にデータの切捨てが発生することがあります。切捨てが発生する場合、インポート・ユーティリティに警告メッセージが表示されます。

親トピック: 文字セットおよびグローバリゼーション・サポートに関する考慮点

26.13 エクスポート・ユーティリティおよびインポート・ユーティリティでのインスタンス親和性の使用

インスタンス親和性を使用して、インポートおよびエクスポートするデータベース内のインスタンスにジョブを関連付けることができます。

複数のリリースを組み合せて使用している場合は、互換性の問題に注意してください。

関連項目:

- 親和性の詳細は、『Oracle Database管理者ガイド』を参照してください。

親トピック: オリジナルのエクスポート

26.14 データベース・オブジェクトのエクスポート時の考慮点

これらの項では、特定のデータベース・オブジェクトをエクスポートする場合の考慮点について説明します。

- エクスポート順序
  エクスポート中もトランザクションから順序番号へのアクセスが継続すると、順序番号がスキップされる場合があります。
- LONGおよびLOBデータ型のエクスポート
  LONGおよびLOBデータ型のエクスポートについて説明します。
- 外部関数ライブラリのエクスポート
  外部関数ライブラリのエクスポートについて説明します。
- オフライン・ローカル管理表領域のエクスポート
  オフライン・ローカル管理表領域のエクスポートについて説明します。

- ディレクトリ別名のエクスポート
  ディレクトリ別名定義のエクスポートについて説明します。
- BFILE列および属性のエクスポート
  BFILE列および属性のエクスポートについて説明します。
- 外部表のエクスポート
  外部表のエクスポートについて説明します。
- オブジェクトの型定義のエクスポート
  オブジェクトの型定義のエクスポートについて説明します。
- ネストした表のエクスポート
  ネストした表のエクスポートについて説明します。
- AQ表のエクスポート
  AQ表のエクスポートについて説明します。
- シノニムのエクスポート
  シノニムのエクスポートについて説明します。
- Javaシノニムに関連して発生する可能性があるエクスポート・エラー
  Javaシノニムに関連して発生する可能性があるエクスポート・エラーについて説明します。
- ファイングレイン・アクセス・コントロールに対するサポート
  ファイングレイン・アクセス・コントロールに対するサポートについて説明します。

親トピック: オリジナルのエクスポート

26.14.1 エクスポート順序

エクスポート中もトランザクションから順序番号へのアクセスが継続すると、順序番号がスキップされる場合があります。

順序番号がスキップされないようにする最良の方法として、エクスポート中に順序にアクセスしないようにします。

順序番号をスキップできるのは、キャッシュされた順序番号が使用されている場合のみです。順序番号のキャッシュが割り当てられていると、それらを現在のデータベースで使用できます。エクスポートされる値は、次の順序番号(キャッシュされた値の後)です。キャッシュされていても未使用の順序番号は、順序がインポートされたときに失われます。

親トピック: データベース・オブジェクトのエクスポート時の考慮点

26.14.2 LONGおよびLOBデータ型のエクスポート

LONGおよびLOBデータ型のエクスポートについて説明します。

エクスポート時に、LONGデータ型はセクションごとにフェッチされます。ただし、LONGデータを含めて、各行のすべての内容を保持できる十分な使用可能メモリーが必要です。

LONG列の長さは、最大2GBです。

LOB列内のデータはすべてを同時にメモリーに保持する必要はありません。LOBデータはセクションごとにロードおよびアンロードされます。

ノート:

既存の LONG 列を LOB 列に変換することもお薦めします。LOB 列の制限の数は、LONG 列の制限の数よりも、かなり少なくなっています。LOB 機能はリリースごとに拡張されていますが、LONG 機能は最近のリリースでは変更さ

れていません。

親トピック: データベース・オブジェクトのエクスポート時の考慮点

26.14.3 外部関数ライブラリのエクスポート

外部関数ライブラリのエクスポートについて説明します。

外部関数ライブラリの内容はエクスポート・ファイルに含まれません。かわりに、ライブラリの指定(名前、場所)のみが全体データベース・モードおよびユーザー・モード・エクスポートに含まれます。データベースを新しい場所に移動する場合は、ライブラリの実行可能ファイルを移動してライブラリの指定を更新する必要があります。

親トピック: データベース・オブジェクトのエクスポート時の考慮点

26.14.4 オフライン・ローカル管理表領域のエクスポート

オフライン・ローカル管理表領域のエクスポートについて説明します。

エクスポートするデータにオフライン・ローカル管理表領域が含まれていると、エクスポート・ユーティリティでは完全な表領域定義をエクスポートできず、エラー・メッセージが表示されます。その場合もデータをインポートはできますが、インポートの前にオフライン・ローカル管理表領域を作成して、欠落している表領域を参照する場合があるDDLコマンドに失敗しないようにする必要があります。

親トピック: データベース・オブジェクトのエクスポート時の考慮点

26.14.5 ディレクトリ別名のエクスポート

ディレクトリ別名定義のエクスポートについて説明します。

ディレクトリ別名の定義は、全体データベース・モード・エクスポートのみに含まれます。データベースを新しい場所に移動する場合、データベース管理者は、新しい場所を指すように、ディレクトリ別名を更新する必要があります。

ディレクトリ別名はユーザー・モードまたは表モード・エクスポートには含まれません。したがって、ディレクトリ別名を使用する前に、ディレクトリ別名をターゲット・システム上に作成しておく必要があります。

親トピック: データベース・オブジェクトのエクスポート時の考慮点

26.14.6 BFILE列および属性のエクスポート

BFILE列および属性のエクスポートについて説明します。

エクスポート・ファイルには、BFILE列または属性によって参照される外部ファイルの内容は保持されません。かわりに、ファイルの名前とディレクトリ別名のみがエクスポート時にコピーされて、インポート時にリストアされます。データベースを移動し、元のディレクトリに含まれているファイルにアクセスできない場合、データベース管理者(DBA)は、指定されたファイルが含まれているディレクトリを、アクセス可能な新しい場所に移動する必要があります。

親トピック: データベース・オブジェクトのエクスポート時の考慮点

26.14.7 外部表のエクスポート

外部表のエクスポートについて説明します。

外部表の内容はエクスポート・ファイルに含まれません。かわりに、表の指定(名前、場所)のみが全体データベース・モードおよび

ユーザー・モード・エクスポートに含まれます。データベースを新しい場所に移動する場合は、外部データを手動で移動して表の指定を更新する必要があります。

親トピック: データベース・オブジェクトのエクスポート時の考慮点

26.14.8 オブジェクトの型定義のエクスポート

オブジェクトの型定義のエクスポートについて説明します。

どのエクスポート・モードでも、エクスポート・ユーティリティには、エクスポートする表で使用されるオブジェクトの型定義に関する情報が含まれます。ターゲット・システム上のオブジェクト型とエクスポート・ファイルに含まれているオブジェクトの状態との間に一貫性があるかどうかを確認するには、オブジェクト名、オブジェクト識別子、オブジェクト・ジオメトリなどの情報が必要です。これにより、表に必要なオブジェクト型が、同じオブジェクト識別子でインポート時に作成されます。

ただし、エクスポート・ユーティリティを実行するユーザーが、対象のオブジェクト型に対する実行権限を持っていない場合、表モード、ユーザー・モード、表領域モードでは、表に必要な完全オブジェクト定義がエクスポート・ファイルに含まれません。この場合、その型の存在確認に必要な情報のみが、同じオブジェクト識別子およびジオメトリでインポート・ターゲット・システム上に書き込まれます。

ユーザーは、DBAとともに適切な型定義を作成するか、またはDBAが実行した全体データベース・モード・エクスポートまたはユーザー・モード・エクスポートからそれらの型定義をインポートすることによって、適切なデータ型がターゲット・システムに存在するようにする必要があります。

すべてのオブジェクト型定義を保持するには、全体データベース・モード・エクスポートを定期的に実行することが重要です。また、異なるスキーマのオブジェクト型定義を使用する場合、DBAは、該当するユーザーのユーザー・モード・エクスポートを実行する必要があります。たとえば、ユーザー`scott`に属している`table1`に、`blake`の型`type1`に存在する列が含まれている場合、表に必要な型定義を保持するには、`blake`と`scott`の両方のユーザー・モード・エクスポートを実行する必要があります。

親トピック: データベース・オブジェクトのエクスポート時の考慮点

26.14.9 ネストした表のエクスポート

ネストした表のエクスポートについて説明します。

ネストした内部表データは、外部表がエクスポートされると必ずエクスポートされます。ネストした内部表は名前を指定できますが、個別にエクスポートすることはできません。

親トピック: データベース・オブジェクトのエクスポート時の考慮点

26.14.10 AQ表のエクスポート

AQ表のエクスポートについて説明します。

キューは表上で実装されます。キューのエクスポートおよびインポートは、基礎となるキュー表と関連のディクショナリ表のエクスポートおよびインポートによって構成されます。キューのエクスポートおよびインポートは、キュー表単位のレベルでのみ実行できます。

キュー表をエクスポートすると、表定義情報とキュー・データの両方がエクスポートされます。キュー表データと表定義がエクスポートされるため、ユーザーは、キュー表データのインポート時に、アプリケーション・レベル・データの整合性のメンテナンスを行います。

関連項目:

親トピック: データベース・オブジェクトのエクスポート時の考慮点

26.14.11 シノニムのエクスポート

シノニムのエクスポートについて説明します。

シノニムおよび別のオブジェクトとして使用される名前を参照するコンパイル済オブジェクトをエクスポートするときは、注意する必要があります。これらのオブジェクトのエクスポートおよびインポートを実行すると、再コンパイルが施行されて、オブジェクト定義が変更されることがあります。

この問題について、次の例に示します。

```
CREATE PUBLIC SYNONYM emp FOR scott.emp;
CONNECT blake/paper;
CREATE TRIGGER t_emp BEFORE INSERT ON emp BEGIN NULL; END;
CREATE VIEW emp AS SELECT * FROM dual;
```

前述の例のデータベースがエクスポートされると、トリガー内のempへの参照によって、scottの表ではなくblakeのビューが参照されます。そのため、インポート・ユーティリティでt_empトリガーを再確立しようとすると、エラーが発生します。

親トピック: データベース・オブジェクトのエクスポート時の考慮点

26.14.12 Javaシノニムに関連して発生する可能性があるエクスポート・エラー

Javaシノニムに関連して発生する可能性があるエクスポート・エラーについて説明します。

対応するDBMS_JAVAパッケージがないとき、またはJavaがロードされないか、正しくロードされないときに、エクスポート操作でDBMS_JAVAという名前のシノニムをエクスポートしようとすると、エクスポートが正常に完了されないまま終了します。生成されるエラー・メッセージには、EXP-00008、ORA-00904、ORA-29516などが含まれます。

Javaが有効化されている場合、エクスポートを再実行する前にDBMS_JAVAシノニムとDBMS_JAVAパッケージの両方が作成されて有効になっていることを確認してください。

Javaが有効化されていない場合は、エクスポートを再実行する前にJava関連オブジェクトを削除します。

親トピック: データベース・オブジェクトのエクスポート時の考慮点

26.14.13 ファイングレイン・アクセス・コントロールに対するサポート

ファイングレイン・アクセス・コントロール・ポリシーのサポートについて説明します。

ファイングレイン・アクセス・コントロール・ポリシーを使用可能にして、表をエクスポートできます。その場合は、次の点を考慮してください。

- ファイングレイン・アクセス・コントロール・ポリシーが使用可能な表を含むエクスポート・ファイルをインポートするユーザーは、適切な権限(特にDBMS_RLSパッケージに対するEXECUTE権限。表のセキュリティ・ポリシーを回復するため)を持っている必要があります。適切な権限を持っていないユーザーが、ファイングレイン・アクセス・ポリシーが使用可能な表をエクスポートしようとすると、エクスポート実行者が読取り権限を持つ行のみがエクスポートされます。
- SELECT文に対してファイングレイン・アクセス・コントロールを使用可能にすると、従来型パス・エクスポートでは表全体をエクスポートできない場合があります。これは、ファイングレイン・アクセスでは問合せが再作成される可能性があるためです。

- ファイングレイン・アクセス・コントロールが設定されている表上でダイレクト・パス・エクスポートを実行できるのは、ユーザーSYS、EXP_FULL_DATABASEロールが使用可能なユーザー、またはEXEMPT ACCESS POLICYを付与されているユーザーのみです。

親トピック: データベース・オブジェクトのエクスポート時の考慮点

26.15 トランスポータブル表領域

トランスポータブル表領域機能は、一連の表領域を、あるOracle Databaseから他のOracle Databaseに移動できる機能です。

トランスポータブル表領域をエクスポートした後、それよりも古いリリース・レベルのデータベースにインポートすることはできません。ターゲット・データベースのリリース・レベルは、ソース・データベース以上である必要があります。

一連の表領域を移動またはコピーするには、表領域を読取り専用にし、表領域のデータ・ファイルをコピーしてから、エクスポートおよびインポートを使用して、データ・ディクショナリに格納されているデータベース情報(メタデータ)を移動します。データ・ファイルとメタデータ・エクスポート・ファイルの両方を、ターゲット・データベースにコピーする必要があります。これらのファイルの転送は、オペレーティング・システムのコピー機能、バイナリ・モードFTP、CD-ROMへの出力などのような、フラット・バイナリ・ファイルのコピー機能を使用して行われます。

データ・ファイルのコピーおよびメタデータのエクスポートの後、表領域を任意に読み書き両用モードにできます。

次のパラメータで、トランスポータブル表領域のメタデータを移動できます。

- TABLESPACES
- TRANSPORT_TABLESPACE

エクスポート操作時のこれらのパラメータの使用方法の詳細は、「TABLESPACES」および「TRANSPORT_TABLESPACE」を参照してください。

関連項目:

- トランスポータブル表領域の管理の詳細は、『Oracle Database管理者ガイド』を参照してください。

親トピック: オリジナルのエクスポート

26.16 読取り専用データベースからのエクスポート

読取り専用データベースからのエクスポートについて説明します。

ソース・データベースからメタデータを抽出するには、エクスポート・ユーティリティで順序付け句(ソート操作)が含まれる問合せを使用します。これらの問合せを正しく実行するには、エクスポートを実行するユーザーがソート・セグメントを指定できる必要があります。これらのソート・セグメントを読取り専用データベースに指定するには、ユーザーの一時表領域が、一時的なローカル管理表領域を指すように設定します。

親トピック: オリジナルのエクスポート

26.17 エクスポート・ユーティリティおよびインポート・ユーティリティを使用したデータベース移行のパーティション化

エクスポート・ユーティリティおよびインポート・ユーティリティを使用して大規模データベースを移行する場合、移行を複数のエクス

ポート・ジョブおよびインポート・ジョブにパーティション化するとより効率的です。

移行をパーティション化する場合は、次のメリットおよびデメリットに注意してください。

- 移行をパーティション化する場合のメリット
  移行をパーティション化する場合のメリットについて説明します。
- 移行をパーティション化する場合のデメリット
  移行をパーティション化する場合のデメリットについて説明します。
- エクスポート・ユーティリティおよびインポート・ユーティリティを使用したデータベース移行のパーティション化方法
  エクスポート・ユーティリティおよびインポート・ユーティリティを使用したデータベース移行のパーティション化方法について説明します。

親トピック: オリジナルのエクスポート

26.17.1 移行をパーティション化する場合のメリット

移行をパーティション化する場合のメリットについて説明します。

移行をパーティション化すると、次のメリットがあります。

- 多くのサブジョブをパラレルに実行できるため、移行に必要な時間を削減できます。
- 最初のエクスポート・ジョブが完了するとすぐにインポート・ユーティリティを起動できます。

親トピック: エクスポート・ユーティリティおよびインポート・ユーティリティを使用したデータベース移行のパーティション化

26.17.2 移行をパーティション化する場合のデメリット

移行をパーティション化する場合のデメリットについて説明します。

移行をパーティション化すると、次のデメリットがあります。

- エクスポートおよびインポートのプロセスがより複雑になります。
- 特定の型のオブジェクトに対する相互スキーマ参照のサポートが損なわれます。たとえば、異なるスキーマの表に対する外部キー制約を持つ表がスキーマに含まれている場合、その表を依存スキーマにインポートしたときに、必要な親レコードが存在しない場合があります。

親トピック: エクスポート・ユーティリティおよびインポート・ユーティリティを使用したデータベース移行のパーティション化

26.17.3 エクスポート・ユーティリティおよびインポート・ユーティリティを使用したデータベース移行のパーティション化方法

エクスポート・ユーティリティおよびインポート・ユーティリティを使用したデータベース移行のパーティション化方法について説明します。

データベースの移行をパーティション化方法で実行するには、次のステップに従います。

1. データベースのすべての最上位メタデータに、次のコマンドを発行します。
   a. `exp FILE=full FULL=y CONSTRAINTS=n TRIGGERS=n ROWS=n INDEXES=n`
   b. `imp FILE=full FULL=y`
2. データベースの各スキーマnに、次のコマンドを発行します。
   a. `exp OWNER=scheman FILE=scheman`

b. `imp FILE=scheman FROMUSER=scheman TOUSER=scheman IGNORE=y`

すべてのエクスポートはパラレルで実行できます。`full.dmp`のインポートが完了すると、残りのインポートもパラレルで実行できます。

親トピック: エクスポート・ユーティリティおよびインポート・ユーティリティを使用したデータベース移行のパーティション化

26.18 リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用

リリースが異なるエクスポート・ユーティリティおよびOracle Databaseを使用することに関連した互換性の問題について説明します。

異なるリリースのOracle Database間でデータを移動するときは、必ず次の基本的な規則に従います。

- インポート・ユーティリティとデータのインポート先であるデータベース(ターゲット・データベース)のリリースは同じである必要があります。たとえば、インポート・ユーティリティ9.2.0.7を使用して9.2.0.8のデータベースにインポートしようとすると、エラーが発生する場合があります。
- エクスポート・ユーティリティのバージョンは、ソースまたはターゲット・データベースの古い方のリリースに合わせる必要があります。

  たとえば、新しいリリースのデータベースにインポートするためにエクスポート・ファイルを作成する場合は、ソース・データベースと同じエクスポート・ユーティリティのバージョンを使用します。一方、古いリリースのデータベースにインポートするためのエクスポート・ファイルを作成する場合は、ターゲット・データベースのリリースと同じバージョンのエクスポート・ユーティリティを使用します。

  - 通常は、Oracle8リリースのエクスポート・ユーティリティを使用して、Oracle9iサーバーからエクスポートし、Oracle8エクスポート・ファイルを作成できます。

- リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用時の制限事項
  異なるリリースのエクスポート・ユーティリティおよびインポート・ユーティリティを使用する場合の制限について説明します。
- リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用例
  リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用例を示します。

親トピック: オリジナルのエクスポート

26.18.1 リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用時の制限事項

異なるリリースのエクスポート・ユーティリティおよびインポート・ユーティリティを使用する場合の制限について説明します。

具体的には次のとおりです。

- エクスポート・ダンプ・ファイルは、特別なバイナリ形式で格納されているため、インポート・ユーティリティによる読取り専用です。
- すべてのエクスポート・ダンプ・ファイルは、上位リリースのOracle Databaseにインポートできます。
- インポート・ユーティリティでは、上位のメンテナンス・リリースのエクスポート・ユーティリティで作成されたエクスポート・ダンプ・ファイルの読取りはできません。たとえば、リリース9.2のエクスポート・ダンプ・ファイルは、リリース9.0.1のインポート・ユーティリティではインポートできません。

- 下位バージョンのエクスポート・ユーティリティを上位リリースのOracle Databaseで実行すると、下位リリースに存在しないデータベース・オブジェクトのカテゴリは、常にエクスポートから除外されます。
- Oracle9iのエクスポート(ダイレクト・パスまたは従来型パス)によって生成されたエクスポート・ファイルは、以前のリリースのインポート・ユーティリティと互換性がありません。したがって、インポートにはOracle9iのインポートのみを使用できます。下位互換性が問題になる場合は、Oracle9iデータベースよりも古いリリースのエクスポート・ユーティリティを使用してください。

親トピック: リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用

26.18.2 リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用例

リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用例を示します。

表26-5に、異なるリリースのOracle Database間でデータを移動させる場合に使用するエクスポート・ユーティリティおよびインポート・ユーティリティのリリースの例を示します。

表26-5 リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用

エクスポート元>インポート先	使用するエクスポート・ユーティリティのリリース	使用するインポート・ユーティリティのリリース
8.1.6 -> 8.1.6	8.1.6	8.1.6
8.1.5 -> 8.0.6	8.0.6	8.0.6
8.1.7 -> 8.1.6	8.1.6	8.1.6
9.0.1 -> 8.1.6	8.1.6	8.1.6
9.0.1 -> 9.0.2	9.0.1	9.0.2
9.0.2 -> 10.1.0	9.0.2	10.1.0
10.1.0 -> 9.0.2	9.0.2	9.0.2

表26-5に、オリジナルのエクスポートおよびインポート・ユーティリティ間のデータ移動を示します。Oracle Database 10gリリース1 (10.1)以上では、ほとんどの場合にデータ・ポンプ・エクスポートおよびデータ・ポンプ・インポート・ユーティリティを使用することをお薦めします。これらのユーティリティは、オリジナルのエクスポートおよびインポート・ユーティリティと比較して、パフォーマンスが著しく向上しています。

関連項目:

10.1以上のリリースを含めた異なるリリース間でのデータのエクスポートおよびインポートの詳細は、『Oracle Databaseアップグレード・ガイド』を参照してください。

親トピック: リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用

27 オリジナルのインポート

オリジナルのインポート・ユーティリティ(imp)は、オリジナルのエクスポート・ユーティリティ(exp)を使用して作成されたダンプ・ファイルをインポートします。

- インポート・ユーティリティとは
  オリジナル・インポート・ユーティリティ(imp)は、オリジナル・エクスポート・ユーティリティ(exp)によって作成されたダンプ・ファイルからオブジェクト定義および表データを読み込みます。
- インポート・ユーティリティを使用する前に
  インポート・ユーティリティを使用する前に実行する必要がある事柄について説明します。
- 既存の表へのインポート
  これらの項では、既存の表にデータをインポートする場合に考慮する必要がある点について説明します。
- インポート操作上のスキーマおよびデータベース・トリガーの影響
  特定のスキーマに対するDDLイベントまたはデータベースに対するDDL関連イベントで実行するように定義されているトリガーは、システム・トリガーです。
- インポート・ユーティリティの起動
  オリジナルのインポート・ユーティリティを起動してパラメータを指定するには、異なる3つの方法のいずれかを使用します。
- インポート・ユーティリティのモード
  インポート・ユーティリティには、次の4種類の操作モードがあります。
- インポート・パラメータ
  これらの項では、インポート・コマンドライン・パラメータについて説明します。
- インポート・セッションの例
  これらの項では、インポート・セッションの例をいくつか取り上げ、パラメータ・ファイル方式およびコマンドライン方式の使用方法を示します。
- 終了コードによる結果の検査と表示
  インポート・ユーティリティでは、操作の完了後、すぐに結果が表示されます。プラットフォームによっては、実行結果はプロセス終了コードに通知され、ログ・ファイルにも記録されます。
- インポート中のエラー処理
  これらの項では、データベース・オブジェクトのインポート時に発生する可能性があるエラーについて説明します。
- 表レベル・インポートおよびパーティション・レベル・インポート
  表、パーティションおよびサブパーティションをインポートできます。
- 索引作成およびメンテナンスの制御
  これらの項では、索引作成およびメンテナンスに関連するインポート・ユーティリティの動作について説明します。
- オリジナルのインポート・ユーティリティでOracle Netを使用する際のネットワーク考慮事項
  ネットワーク経由でインポートを実行する場合は、Oracle NetでOracle Data Pumpのオリジナルのインポート・ユーティリティ(imp)を使用できます。
- 文字セットおよびグローバリゼーション・サポートに関する考慮点
  これらの項では、ユーザー・データおよびデータ定義言語(DDL)の文字セット変換に関連するインポート・ユーティリティのグローバリゼーション・サポートの動作について説明します。
- インスタンス親和性の使用方法
  インスタンス親和性を使用して、インポートおよびエクスポートするデータベース内のインスタンスにジョブを関連付けることができます。
- データベース・オブジェクトのインポートに関する考慮点

これらの項では、特定のデータベース・オブジェクトをインポートする場合の制限事項および考慮点について説明します。

- ファイングレイン・アクセス・コントロールに対するサポート
  ファイングレイン・アクセス・コントロール・ポリシーをリストアするには、ファイングレイン・アクセス・コントロール・ポリシーが使用可能な表を含むエクスポート・ファイルからインポートするユーザーが、表のセキュリティ・ポリシーを回復するためのDBMS_RLSパッケージに対するEXECUTE権限が必要です。
- スナップショットおよびスナップショット・ログ
  特定の状況、特にデータ・ウェアハウスに関連する場合、スナップショットは、マテリアライズド・ビューと呼ばれます。これらの項では、そのような場合でもスナップショットという用語を使用します。
- トランスポータブル表領域
  トランスポータブル表領域機能は、一連の表領域を、あるOracle Databaseから他のOracle Databaseに移動できる機能です。
- 記憶域パラメータ
  デフォルトでは、表は元の表領域にインポートされます。
- 読取り専用表領域
  読取り専用表領域はエクスポート可能です。インポートでは、表領域がターゲット・データベース内に存在しない場合、読取り/書込み表領域として表領域が作成されます。
- 表領域を削除する方法
  インポート前に、オブジェクトに別の表領域を使用するように再定義すると、表領域を削除できます。impコマンドの発行時には、IGNORE=yを指定します。
- 表領域を再編成する方法
  ユーザーの割当て制限で許可される場合、そのユーザーの表はエクスポート元と同じ表領域にインポートされます。
- 統計情報のインポート
  統計情報がエクスポート時に必要で、表にアナライザ統計が利用できる場合、表の統計情報の再計算に使用されるANALYZE文が、エクスポートによってダンプ・ファイルに含まれます。
- エクスポート・ユーティリティおよびインポート・ユーティリティを使用したデータベース移行のパーティション化
  エクスポート・ユーティリティおよびインポート・ユーティリティを使用して大規模データベースを移行する場合、移行を複数のエクスポート・ジョブおよびインポート・ジョブにパーティション化するとより効率的です。
- インポート操作のチューニングに関する考慮点
  これらのセクションでは、インポート操作のパフォーマンスを向上させるためのいくつかの方法について説明します。
- リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用
  これらのセクションでは、リリースが異なるエクスポート・ユーティリティおよびOracle Databaseの使用に関連した互換性の問題について説明します。

親トピック: その他のユーティリティ

27.1 インポート・ユーティリティとは

オリジナル・インポート・ユーティリティ(imp)は、オリジナル・エクスポート・ユーティリティ(exp)によって作成されたダンプ・ファイルからオブジェクト定義および表データを読み込みます。

ノート:

オリジナルのエクスポートは、Oracle Database 11g からは原則としてサポートされなくなりました。Oracle Database 11g 以降のリリースでサポートされているオリジナルのエクスポートの使用方法は、XMLType データの

Oracle Database 10g リリース 2 (10.2)以前への下位移行のみです。新しい Oracle Data Pump エクスポートおよびインポート・ユーティリティを使用することをお薦めします。このガイドラインの唯一の例外は、元のエクスポートおよびインポートが必要な次の状況です。

- オリジナルのエクスポート・ユーティリティ(exp)を使用して作成されたファイルをインポートする必要がある。
- オリジナルのインポート・ユーティリティ(imp)を使用してインポートされるファイルをエクスポートする必要がある。このような例として考えられるのは、Oracle Database 10g からデータをエクスポートし、それより前のデータベース・リリースにインポートする場合です。

元のインポートを使用する場合、次の条件がtrueである必要があります。

- ダンプ・ファイルはOracleのバイナリ形式になっており、オリジナルのインポートを使用した場合のみ読取りが可能です。
- ダンプ・ファイルの作成に使用したエクスポート・ユーティリティより前のバージョンのインポート・ユーティリティは使用できません。
- 表オブジェクト: インポートの順序
  表オブジェクトは、エクスポート・ダンプ・ファイルから読み取られた順序でインポートされます。

親トピック: オリジナルのインポート

27.1.1 表オブジェクト: インポートの順序

表オブジェクトは、エクスポート・ダンプ・ファイルから読み込まれたとおりにインポートされます。

ダンプ・ファイルには、次の順序でオブジェクトが格納されています。

1. 型定義
2. 表定義
3. 表データ
4. 表索引
5. 整合性制約、ビュー、プロシージャおよびトリガー
6. ビットマップ索引、ファンクション索引およびドメイン索引

インポートの順序は次のとおりです。まず、新しい表が作成されます。次にデータがインポートされ、索引が作成されます。その後トリガーがインポートされ、整合性制約が新しい表で使用可能になり、ビットマップ索引、ファンクション索引またはドメイン索引(あるいはそれらのすべて)が作成されます。このようにインポートされると、表のインポート順序が原因でデータが拒否されることがなくなります。また、同じデータについて、トリガーが重複して2回(最初の挿入時に1回、インポート中に1回)起動することもなくなります。

親トピック: インポート・ユーティリティとは

27.2 インポート・ユーティリティを使用する前に

インポート・ユーティリティを使用する前に、実行する必要がある事柄について説明します。

インポート・ユーティリティを使用する前に、次のことを行う必要があります(詳細は以降の項を参照してください)。

- データベースを手動で作成した場合は、`catexp.sql`または`catalog.sql`スクリプトが実行されたことの確認。Database Configuration Assistant(DBCA)を使用してデータベースを作成した場合は、これらのスクリプトを実

行する必要はありません。

- 必要なアクセス権限を所有していることの確認

- catexp.sqlまたはcatalog.sqlの実行
  インポート・ユーティリティを使用するには、データベースを作成するかまたは新しいバージョンに移行した後で、スクリプトcatexp.sqlまたは(catexp.sqlを実行する)catalog.sqlを実行する必要があります。
- インポート操作のアクセス権の確認
  インポートを使用するには、Oracle DatabaseにCREATE SESSION権限を持っている必要があります。この権限は、データベースの作成時に設定されるCONNECTロールに含まれます。
- 処理上の制限事項
  インポート・ユーティリティを使用してデータを処理する場合、制限が適用されます。

親トピック: オリジナルのインポート

27.2.1 catexp.sqlまたはcatalog.sqlの実行

インポート・ユーティリティを使用するには、データベースを作成するかまたは新しいバージョンに移行した後で、スクリプトcatexp.sqlまたは(catexp.sqlを実行する)catalog.sqlを実行する必要があります。

データベースに対して、catexp.sqlまたはcatalog.sqlを実行するのは1回のみです。スクリプトを実行すると、次の処理が行われ、データベースはエクスポートおよびインポート操作に備えて調整されます。

- データ・ディクショナリへの必要なインポート・ビューの作成
- EXP_FULL_DATABASEロールおよびIMP_FULL_DATABASEロールの作成
- EXP_FULL_DATABASEおよびIMP_FULL_DATABASEロールへのすべての必要な権限の割当て
- DBAロールへのEXP_FULL_DATABASEおよびIMP_FULL_DATABASEの割当て
- インストールされているcatexp.sqlのバージョンの記録

親トピック: インポート・ユーティリティを使用する前に

27.2.2 インポート操作のアクセス権の確認

インポート・ユーティリティを使用するには、Oracle DatabaseにCREATE SESSION権限が必要です。この権限は、データベースの作成時に設定されるCONNECTロールに含まれます。

他のユーザーが作成したエクスポート・ファイルをインポートすることもできます。ただし、EXP_FULL_DATABASEロールを所有するユーザーが作成したエクスポート・ファイルをインポートするには、IMP_FULL_DATABASEロールが必要です。通常、データベース管理者(DBA)にはこれらの両方のロールが割り当てられています。

- オブジェクトをスキーマにインポートする方法
  自分のスキーマにオブジェクトをインポートするために必要な権限。
- 権限のインポート
  他のユーザーに付与されている権限をインポートするには、インポートを開始したユーザーがそのオブジェクトの所有者であるか、WITH GRANT OPTION付きのオブジェクト権限を所有している必要があります。
- 他のスキーマへのオブジェクトのインポート
  オブジェクトを他のユーザーのスキーマにインポートする場合は、IMP_FULL_DATABASEロールを使用可能にしておく必要があります。
- システム・オブジェクトのインポート

システム・オブジェクトを全データベース・エクスポート・ファイルからインポートする場合は、IMP_FULL_DATABASEロールを使用可能にする必要があります。

親トピック: インポート・ユーティリティを使用する前に

27.2.2.1 オブジェクトをスキーマにインポートする方法

自分のスキーマにオブジェクトをインポートするために必要な権限。

表27-1に、自分のスキーマにオブジェクトをインポートするために必要な権限を示します。これらのすべての権限は、あらかじめRESOURCEロールに含まれています。

表27-1 自分のスキーマにオブジェクトをインポートするために必要な権限

オブジェクト	必要な権限(該当する場合の権限タイプ)
クラスタ	CREATE CLUSTER(システム)または UNLIMITED TABLESPACE(システム)。ユーザーに表領域割当て制限が割り当てられている必要があります。
データベース・リンク	CREATE DATABASE LINK(システム)およびリモート・データベースの場合は CREATE SESSION(システム)
表のトリガー	CREATE TRIGGER (システム)
スキーマのトリガー	CREATE ANY TRIGGER(システム)
索引	CREATE INDEX(システム)または UNLIMITED TABLESPACE(システム)。ユーザーに表領域割当て制限が割り当てられている必要があります。
整合性制約	ALTER TABLE (オブジェクト)
ライブラリ	CREATE ANY LIBRARY(システム)
パッケージ	CREATE PROCEDURE (システム)
プライベート・シノニム	CREATE SYNONYM (システム)
順序	CREATE SEQUENCE (システム)
スナップショット	CREATE SNAPSHOT (システム)
ストアド・ファンクション	CREATE PROCEDURE (システム)
ストアド・プロシージャ	CREATE PROCEDURE (システム)

オブジェクト	必要な権限(該当する場合の権限タイプ)
表データ	INSERT TABLE (オブジェクト)
表定義(コメントおよび監査オプションを含む)	CREATE TABLE(システム)または UNLIMITED TABLESPACE(システム)ユーザーに表領域割当て制限が割り当てられている必要があります。
ビュー	実表の場合は CREATE VIEW (システム)および SELECT (オブジェクト)、または SELECT ANY TABLE (システム)
オブジェクト型	CREATE TYPE (システム)
外部関数ライブラリ	CREATE LIBRARY (システム)
ディメンション	CREATE DIMENSION (システム)
演算子	CREATE OPERATOR (システム)
索引タイプ	CREATE INDEXTYPE (システム)

親トピック: インポート操作のアクセス権の確認

27.2.2.2 権限のインポート

他のユーザーに付与されている権限をインポートするには、インポートを開始したユーザーがそのオブジェクトの所有者であるか、WITH GRANT OPTION付きのオブジェクト権限を所有している必要があります。

表27-2に、権限がターゲット・システムで有効となるために必要な条件を示します。

表27-2 権限のインポートに必要な権限

権限	条件
オブジェクト権限	オブジェクトがユーザーのスキーマに存在しているか、 ユーザーが With Grant Option 付きのオブジェクト権限を所有しているか、または IMP_FULL_DATABASE ロールを使用可能にする必要があります。
システム権限	ユーザーが SYSTEM 権限および With Admin Option を所有している必要があります。

親トピック: インポート操作のアクセス権の確認

27.2.2.3 他のスキーマへのオブジェクトのインポート

オブジェクトを他のユーザーのスキーマにインポートする場合は、IMP_FULL_DATABASEロールを使用可能にしておく必要があります。

親トピック: インポート操作のアクセス権の確認

27.2.2.4 システム・オブジェクトのインポート

システム・オブジェクトを全データベース・エクスポート・ファイルからインポートする場合は、IMP_FULL_DATABASEロールを使用可能にする必要があります。

パラメータFULLを指定すると、次のシステム・オブジェクトがインポート対象に含まれます。

- プロファイル
- パブリック・データベース・リンク
- パブリック・シノニム
- ロール
- ロールバック・セグメント定義
- リソース・コスト
- 外部関数ライブラリ
- コンテキスト・オブジェクト
- システム・プロシージャ・オブジェクト
- システム監査オプション
- システム権限
- 表領域定義
- 表領域割当て制限
- ユーザー定義
- ディレクトリ別名
- システム・イベント・トリガー

親トピック: インポート操作のアクセス権の確認

27.2.3 処理上の制限事項

インポート・ユーティリティを使用してデータを処理する場合、制限が適用されます。

具体的には次のとおりです。

- 型定義が進化し、その進化した型を参照するデータがエクスポートされた場合、インポート・システムの型定義も同様に進化させる必要があります。
- エクスポートおよびインポート時には、表およびパーティションの表圧縮属性が保持されます。ただし、インポート処理ではダイレクト・パスAPIが使用されないため、データは、インポート時に圧縮された形式では格納されません。

親トピック: インポート・ユーティリティを使用する前に

27.3 既存の表へのインポート

これらの項では、既存の表にデータをインポートする場合に考慮する必要がある点について説明します。

- データのインポート前に手動で表を作成する方法
  データをインポートする前に表を手動で作成できます。
- 参照制約を使用禁止にする方法
  参照制約を使用禁止にする方法について説明します。
- 手動によるインポートの順序付け
  手動によるインポートの順序付けについて説明します。

親トピック: オリジナルのインポート

27.3.1 データのインポート前に手動で表を作成する方法

データをインポートする前に表を手動で作成できます。

エクスポート・ファイルから表にデータをインポートする前に手動で表を作成する場合は、以前使用した表定義を使用するか、互換性のある形式を使用して表を作成します。たとえば、列幅の増加および列順序の変更はできますが、次のことはできません。

- NOT NULL列の追加
- 列のデータ型の互換性のないデータ型への変更(LONGからNUMBERなど)
- 表で使用されているオブジェクト型定義の変更
- DEFAULT列値の変更

ノート:

データをインポートする前に表を手動で作成した場合は、表がすでに存在するため、エクスポート・ダンプ・ファイルで CREATE TABLE 文を実行するとエラーが発生します。このエラーを回避して、表へのデータのロードを継続するには、インポート・パラメータを IGNORE=y に設定します。設定しない場合、表作成エラーが発生し、データは表にロードされません。

親トピック: 既存の表へのインポート

27.3.2 参照制約を使用禁止にする方法

参照制約を使用禁止にする方法について説明します。

通常のインポートの順序では、参照制約はすべての表がインポートされた後にインポートされます。この順序でインポートすることによって、まだインポートされていないデータに対する参照整合性制約が存在する場合に発生するエラーを回避できます。

データが既存の表にロードされる場合に、このようなエラーが発生することがあります。たとえば、表empでmgr列に参照整合性制約が定義されており、その制約によって表emp内のマネージャ番号が検証される場合、マネージャの行がインポートされていない時点では、従業員の行が基準を満たしていても、参照整合性制約の違反になることがあります。

このようなエラーが発生すると、エラー・メッセージが生成され、失敗した行をバイパスして、引き続き他の行が表にインポートされます。制約を手動で使用禁止にすると、このエラーを回避できます。

複数の表にまたがって参照制約が存在すると、問題になることがあります。たとえば、emp表の順序がエクスポート・ダンプ・ファイ

ル内でdept表より先であるにもかかわらず、emp表からdept表へ参照チェックが行われると、参照制約違反のため、emp表のいくつかの行がインポートされないことがあります。

このようなエラーが発生しないようにするには、データを既存の表にインポートするときに、参照整合性制約を使用禁止にします。

親トピック: 既存の表へのインポート

27.3.3 手動によるインポートの順序付け

手動によるインポートの順序付けについて説明します。

インポート後に制約が再び使用可能にされると、表全体がチェックされますが、大きな表の場合はチェックに時間がかかることがあります。表のチェックにかかる時間が長すぎる場合、インポートを手動で順序付ける方が効率的なこともあります。

そのためには、エクスポート・ファイルからのインポートを1回ではなく複数回に分けて実行します。まず、参照チェックのターゲットである表をインポートします。次に、これらの表を参照する表をインポートします。表が循環的に相互参照している場合、および表がその表自体を参照している場合を除き、この方法は有効です。

親トピック: 既存の表へのインポート

27.4 インポート操作上のスキーマおよびデータベース・トリガーの影響

特定のスキーマに対するDDLイベントまたはデータベースに対するDDL関連イベントで実行するように定義されているトリガーは、システム・トリガーです。

これらのトリガーは、特定のインポート操作に悪い影響を与える場合があります。たとえば、表などのデータベース・オブジェクトを正常に再作成できない場合があります。これによって、トリガーが原因で問題が発生したということがわからないエラーが返されます。

データベース管理者およびシステム・トリガーを作成するユーザーは、このようなトリガーによって、ユーザーが、権限を持つデータベースの操作を実行できなくなることがないように確認する必要があります。システム・トリガーをテストするには、次のステップに従います。

1. トリガーを定義します。
2. データベース・オブジェクトを作成します。
3. 表モードまたはユーザーモードでオブジェクトをエクスポートします。
4. オブジェクトを削除します。
5. オブジェクトをインポートします。
6. オブジェクトが正常に再作成されていることを確認します。

ノート:

全体エクスポートでは、スキーマ SYS が所有するトリガーはエクスポートされません。SYS トリガーは、全体インポートの前後のいずれかに手動で再作成する必要があります。このトリガーによってインポートの進行を妨げるような処理が定義されないように、インポートの後に再作成することをお薦めします。

親トピック: オリジナルのインポート

27.5 インポートの起動

オリジナルのインポート・ユーティリティを起動してパラメータを指定するには、異なる3つの方法のいずれかを使用します。

オリジナルのインポート・ユーティリティを開始するための3つの方法は次のとおりです。

- コマンドライン・エントリ
- パラメータ・ファイル
- 対話方式モード

これらのいずれかの方法を使用する前に、使用可能なパラメータについての説明を必ず読んでください。

- コマンドライン・エントリ
  すべての有効なパラメータおよびその値をコマンドラインから指定できます。
- パラメータ・ファイル
  パラメータ・ファイルに、すべての有効なパラメータおよびその値を指定できます。
- 対話方式モード
  各パラメータの値をプロンプトで入力するには、コマンドラインでimpを指定するだけです。
- SYSDBAでのインポート・ユーティリティの起動
  オリジナルのインポート・ユーティリティをSYSDBAとして起動することは特別な手順であり、特定の状況下でのみ実行する必要があります。
- オンライン・ヘルプの利用
  インポート・ユーティリティにはオンライン・ヘルプがあります。インポート・ユーティリティのヘルプを表示するには、`imp help=y`と入力します。

関連トピック

- インポート・パラメータ

親トピック: オリジナルのインポート

27.5.1 コマンドライン・エントリ

すべての有効なパラメータおよびその値をコマンドラインから指定できます。

次の構文を使用します(ユーザー名とパスワードの入力を求められます)。

```
imp PARAMETER=value
```

または

```
imp PARAMETER=(value1,value2,...,valuen)
```

システムでのコマンドラインの最大長を超える数のパラメータは指定できません。

親トピック: インポート・ユーティリティの起動

27.5.2 パラメータ・ファイル

パラメータ・ファイルに、すべての有効なパラメータおよびその値を指定できます。

複数のパラメータを1つのファイルに格納することによって、これらを簡単に変更したり再利用することができます。データベースごとに別のパラメータを使用する場合は、複数のパラメータ・ファイルを作成できるので、それぞれにパラメータ・ファイルを用意すると便

利です。

フラット・ファイル用のテキスト・エディタを使用してパラメータ・ファイルを作成します。コマンドライン・オプション`PARFILE=filename`を指定すると、インポート・ユーティリティは、コマンドラインからではなく指定されたファイルからパラメータを読み込みます。たとえば:

パラメータ・ファイルは、次のいずれかの構文を使用して指定します。

```
PARAMETER=value
PARAMETER=(value)
PARAMETER=(value1, value2, ...)
```

次に、パラメータ・ファイル内のリストの一部を示します。

```
FULL=y
FILE=dba.dmp
GRANTS=y
INDEXES=y
CONSISTENT=y
```

ノート:

パラメータ・ファイルの最大サイズは、オペレーティング・システムによって制限されます。また、パラメータ・ファイル名はオペレーティング・システムのネーミング規則に従います。

ポンド(#)記号を使用すると、パラメータ・ファイルにコメントを追加できます。ポンド(#)の右側にある文字はすべて無視されます。

コマンドラインでのパラメータの入力と同時にパラメータ・ファイルを指定できます。実際、パラメータ・ファイルとコマンドラインの両方に同じパラメータを指定することもできます。コマンドラインでの`PARFILE`パラメータと他のパラメータの位置によって、優先されるパラメータが決まります。たとえば、パラメータ・ファイル`params.dat`でパラメータ`INDEXES=y`を指定し、インポート・ユーティリティを次のコマンドで起動するとします。

```
imp PARFILE=params.dat INDEXES=n
```

この場合、`INDEXES=n`は`PARFILE=params.dat`の後にあるため、パラメータ・ファイルに指定されている`INDEXES`パラメータの値は、`INDEXES=n`によって上書きされます。

関連項目:

- インポート・パラメータ
- リモートのデータベースからのエクスポートの指定方法の詳細は、「ネットワークに関する考慮点」を参照してください。

親トピック: インポート・ユーティリティの起動

27.5.3 対話方式モード

各パラメータの値をプロンプトで入力するには、コマンドラインで`imp`を指定するだけです。

ユーザー名とパスワードを入力するように要求されます。

その後、一般的に使用されるパラメータが表示されます。デフォルト値を受け入れるか(示された場合)、または異なる値を入力できます。コマンドライン対話方式では、すべての機能に関してプロンプトが表示されるわけではありません。また、コマンドライン

対話方式は下位互換用のみに提供されています。

親トピック: インポート・ユーティリティの起動

27.5.4 SYSDBAでのインポート・ユーティリティの起動

オリジナルのインポート・ユーティリティをSYSDBAとして起動することは特別な手順であり、特定の状況下でのみ実行する必要があります。

SYSDBAは内部的に使用され、一般ユーザーとは異なる特別な役割があります。そのため、通常、次の場合以外は、インポート・ユーティリティをSYSDBAで起動しないでください。

- Oracleサポート・サービスから要求された場合
- トランスポータブル表領域セットをインポートする場合

親トピック: インポート・ユーティリティの起動

27.5.5 オンライン・ヘルプの利用

インポート・ユーティリティでは、オンライン・ヘルプが提供されます。インポート・ユーティリティのヘルプを表示するには、`imp help=y`と入力します。

親トピック: インポート・ユーティリティの起動

27.6 インポート・モード

インポート・ユーティリティには、次の4種類の操作モードがあります。

具体的には次のとおりです。

- 全体インポート: データベース全体をインポートします。`IMP_FULL_DATABASE`ロールを所有するユーザーのみが使用できます。`FULL`パラメータを使用してこのモードを指定します。
- 表領域モード: 特権ユーザーが表領域をOracle Database間で移動できます。`TRANSPORT_TABLESPACE`パラメータを使用してこのモードを指定します。
- ユーザー・モード: 所有するすべてのオブジェクト(表、権限、索引、プロシージャなど)をインポートできます。特権ユーザーがユーザー・モードでインポートする場合、指定したユーザー・グループのユーザーのスキーマにあるすべてのオブジェクトをインポートできます。このモードを指定するには、`FROMUSER`パラメータを使用します。
- 表モード: 特定の表およびパーティションをインポートできます。特権ユーザーは、インポートする表を含むスキーマを指定して、その表を修飾できます。`TABLES`パラメータを使用してこのモードを指定します。

> ノート:
>
> 表モードを使用して `ANYDATA` 型の列を含む表をインポートする場合、次のエラーが発生する場合があります。
>
> ORA-22370: メソッドの使用方法が正しくありません。Nonexistent type。
>
> このメッセージは、`ANYDATA` 列が、データベースに存在しない他の型に依存することを示します。表モード

を使用して ANYDATA 型を使用する表をインポートする前に、手動で、ターゲット・データベース内に依存型を作成する必要があります。

IMP_FULL_DATABASEロールを持つユーザーは、これらのモードのいずれかを指定する必要があります。指定しない場合、インポートはエラーになります。IMP_FULL_DATABASEロールを持たないユーザーがこれらのモードをいずれも指定しない場合は、ユーザー・レベルのインポートが実行されます。

ノート:

Oracle Database 12c リリース 2 (12.2)から、インポート・ユーティリティ(imp)では、セキュリティ上の理由から、ユーザーS
イルにユーザーSYS として再作成する必要があるオブジェクトが含まれている場合、imp ユーティリティでは、かわりにユーザーS
SYSTEM によってオブジェクトを再作成できない場合は、インポートが完了した後、ユーザー自身がオブジェクトを手動で再作成

インポート・ジョブが DBA ロールを持つユーザーによって実行され、一部のオブジェクトをユーザーSYSTEM によって再作成できな
す。

```
IMP-00403:
Warning: This import generated a separate SQL file "logfilename_sys" which contains
```

生成される SQL ファイルには、ユーザーSYSTEM によって再作成できなかったオブジェクトの失敗した DDL が含まれます。これ
失敗した DDL を手動で実行する必要があります。

SQL ファイルには、LOG パラメータで指定されたファイル名に'_sys.sql'を追加した名前が自動的に付けられます。たとえば、
JulyImport_sys.sql となります。

ログ・ファイルが指定されていない場合、SQL ファイルのデフォルト名は import_sys.sql となります。

ノート: SQL ファイルはすべてのインポート・ジョブで生成されるわけではなく、ユーザーDBA として実行されたジョブによってのみ

次の表に、各モードでインポートされるオブジェクトのリストを示します。

表27-3 各モードでインポートされるオブジェクト

オブジェクト	表モード	ユーザー・モード	全データベース・モード	表領域モード
分析クラスタ	なし	あり	あり	なし
分析表/統計	あり	あり	あり	あり
アプリケーション・コンテキスト	なし	なし	あり	なし
監査情報	あり	あり	あり	なし
B ツリー索引、ビットマップ索引、ド	あり脚注 1	あり	あり	あり

オブジェクト	表モード	ユーザー・モード	全データベース・モード	表領域モード
メイン・ファンクション索引				
クラスタ定義	なし	あり	あり	あり
列コメントおよび表コメント	あり	あり	あり	あり
データベース・リンク	なし	あり	あり	なし
デフォルト・ロール	なし	なし	あり	なし
ディメンション	なし	あり	あり	なし
ディレクトリ別名	なし	なし	あり	なし
外部表(データなし)	あり	あり	あり	なし
外部関数ライブラリ	なし	あり	あり	なし
表の所有者以外のユーザーが所有する索引	あり(特権ユーザーのみ)	あり	あり	あり
索引タイプ	なし	あり	あり	なし
Java リソースおよび Java クラス	なし	あり	あり	なし
ジョブ・キュー	なし	あり	あり	なし
ネストした表のデータ	あり	あり	あり	あり
オブジェクト権限	あり(表および索引のみ)	あり	あり	あり
表で使用されるオブジェクト型定義	あり	あり	あり	あり
オブジェクト型	なし	あり	あり	なし

オブジェクト	表モード	ユーザー・モード	全データベース・モード	表領域モード
演算子	なし	あり	あり	なし
パスワード履歴	なし	なし	あり	なし
インスタンスの事後処理およびオブジェクト	なし	なし	あり	なし
スキーマの事後プロシージャ処理およびオブジェクト	なし	あり	あり	なし
表の事後処理	あり	あり	あり	あり
表の事後プロシージャ処理およびオブジェクト	あり	あり	あり	あり
スキーマの事前プロシージャ・オブジェクトおよび処理	なし	あり	あり	なし
表の事前処理	あり	あり	あり	あり
表の事前プロシージャ処理	あり	あり	あり	あり
プライベート・シノニム	なし	あり	あり	なし
プロシージャ・オブジェクト	なし	あり	あり	なし
プロファイル	なし	なし	あり	なし
パブリック・シノニム	なし	なし	あり	なし
参照整合性制約	あり	あり	あり	なし
リフレッシュ・グループ	なし	あり	あり	なし
リソース・コスト	なし	なし	あり	なし
ロール権限	なし	なし	あり	なし

オブジェクト	表モード	ユーザー・モード	全データベース・モード	表領域モード
ロール	なし	なし	あり	なし
ロールバック・セグメント定義	なし	なし	あり	なし
表のセキュリティ・ポリシー	あり	あり	あり	あり
順序番号	なし	あり	あり	なし
スナップショット・ログ	なし	あり	あり	なし
スナップショットおよびマテリアライズド・ビュー	なし	あり	あり	なし
システム権限	なし	なし	あり	なし
表制約(主キー制約、一意制約、CHECK 制約)	あり	あり	あり	あり
表データ	あり	あり	あり	あり
表定義	あり	あり	あり	あり
表領域定義	なし	なし	あり	なし
表領域割当て制限	なし	なし	あり	なし
トリガー	あり	あり[脚注 2]	あり[脚注 3]	あり
他のユーザーが所有するトリガー	あり(特権ユーザーのみ)	なし	なし	なし
ユーザー定義	なし	なし	あり	なし
ユーザー・プロキシ	なし	なし	あり	なし
ユーザー・ビュー	なし	あり	あり	なし

オブジェクト	表モード	ユーザー・モード	全データベース・モード	表領域モード
ユーザー・ストアド・プロシージャ、ユーザー・ストアド・パッケージおよびユーザー・ストアド・ファンクション	なし	あり	あり	なし

脚注1

非特権ユーザーがエクスポートおよびインポートできるのは、そのユーザー自身が所有する表に関する索引のみです。他のユーザーが所有する表に関する索引や、ユーザー自身が所有する表に関して他のユーザーが作成した索引はエクスポートできません。特権ユーザーは、エクスポートおよびインポート対象に指定したユーザーの表に関する索引が、表の所有者以外のユーザーが作成したものであっても、その索引をエクスポートおよびインポートできます。指定したユーザーが他のユーザーの表に関する索引を所有しているときは、エクスポートするユーザーのリストに表の所有者であるユーザーを指定しないかぎり、その索引はエクスポートされません。

脚注2

特権ユーザーも非特権ユーザーも、そのユーザー自身が所有するすべてのトリガーを(他のユーザーが所有する表に関するトリガーの場合でも)、エクスポートおよびインポートできます。

脚注3

全体エクスポートでは、スキーマSYSが所有するトリガーはエクスポートされません。SYSトリガーは、全体インポートの前後のいずれかに手動で再作成する必要があります。このトリガーによってインポートの進行を妨げるような処理が定義されないように、インポートの後に再作成することをお薦めします。

親トピック: オリジナルのインポート

27.7 インポート・パラメータ

これらの項では、インポート・コマンドライン・パラメータについて説明します。

- BUFFER
- COMMIT
- COMPILE
- CONSTRAINTS
- DATA_ONLY
  DATA_ONLYインポート・パラメータを使用すると、ダンプ・ファイルからデータのみをインポートできます。
- DATAFILES
  DATAFILESインポート・パラメータを使用すると、データベースに転送されるデータ・ファイルをリストできます。
- DESTROY
  DESTROYインポート・パラメータでは、データベースを構成する既存のデータ・ファイルを再利用する必要があるかどうかを指定します。
- FEEDBACK
  インポート・ユーティリティのFEEDBACKパラメータでは、インポート・ユーティリティにプログレス・バー(n行分のエクスポートを1つのピリオドで示す)を表示するように指定します。
- FILE

FILEインポート・ユーティリティ・パラメータでは、インポートするエクスポート・ファイルの名前を指定します。

- FILESIZE
  FILESIZEインポート・ユーティリティ・パラメータを使用すると、エクスポート時に指定したサイズと同じ最大ダンプ・ファイル・サイズを指定できます。
- FROMUSER
  インポート・ユーティリティのFROMUSERパラメータを使用すると、複数のスキーマを含むエクスポート・ファイルからスキーマのサブセットをインポートできます。
- FULL
  FULLインポート・ユーティリティ・パラメータでは、エクスポート・ダンプ・ファイル全体をインポートするかどうかを指定します。
- GRANTS
  オブジェクト権限をインポートするかどうかを指定します。
- HELP
  インポート・ユーティリティのHELPパラメータを使用すると、インポート・パラメータの説明が表示されます。
- IGNORE
  IGNOREインポート・ユーティリティ・パラメータでは、オブジェクト作成エラーをどのように処理する必要があるかを指定します。
- INDEXES
  インポート・ユーティリティのINDEXESパラメータでは、索引をインポートするかどうかを指定します。
- INDEXFILE
  インポート・ユーティリティのINDEXFILEパラメータでは、索引作成文を受け取るファイルを指定します。
- LOG
  情報メッセージおよびエラー・メッセージを受け取るファイル(import.logなど)を指定します。
- PARFILE
  インポート・パラメータのリストを格納するファイルのファイル名を指定します。
- RECORDLENGTH
  ファイル・レコードの長さをバイト単位で指定します。
- RESUMABLE
  RESUMABLEパラメータは、再開可能な領域割当てを有効または無効にする場合に使用します。
- RESUMABLE_NAME
  RESUMABLE_NAMEパラメータの値によって、再開可能な文を識別します。
- RESUMABLE_TIMEOUT
  RESUMABLE_TIMEOUTパラメータの値では、エラー修正までに必要な時間を指定します。
- ROWS
  表のデータ行をインポートするかどうかを指定します。
- SHOW
  インポートの前に、エクスポート・ファイルの内容を表示します。
- SKIP_UNUSABLE_INDEXES
  インポート・ユーティリティとOracleデータベースの両方でSKIP_UNUSABLE_INDEXESパラメータを指定します。
- STATISTICS
  インポート時のデータベース・オプティマイザ統計の処理方法を指定します。
- STREAMS_CONFIGURATION
  エクスポート・ダンプ・ファイル内に存在する可能性のあるすべての一般的なGoldenGateレプリケーション・メタデータをインポートするかどうかを指定します。

- STREAMS_INSTANTIATION
  エクスポート・ダンプ・ファイル内に存在するStreamsインスタンス化メタデータをインポートするかどうかを指定します。
- TABLES
- TABLESPACES
  インポート・ユーティリティのTABLESPACESパラメータ。
- TOID_NOVALIDATE
  TOID_NOVALIDATEパラメータは、TOIDと比較しない型を指定するために使用します。
- TOUSER
  インポートの対象になるスキーマを所有するユーザー名のリストを指定します。
- TRANSPORT_TABLESPACE
  yを指定すると、インポート・ユーティリティはエクスポート・ファイルからトランスポータブル表領域のメタデータをインポートします。
- TTS_OWNERS
  TRANSPORT_TABLESPACEにyを指定した場合は、このパラメータを使用して一連のトランスポータブル表領域のデータの所有者ユーザーを表示します。
- USERID (ユーザー名/パスワード)
  インポートを実行するユーザーのユーザー名、パスワードおよびオプションの接続文字列を指定します。
- VOLSIZE
  各テープ・ボリュームのダンプ・ファイルに最大バイト数を指定します。

親トピック: オリジナルのインポート

27.7.1 BUFFER

デフォルト: オペレーティング・システムによって異なります

BUFFERに指定された整数は、転送したデータ行を格納するバッファのバイト数です。

BUFFERによって、インポートで挿入される配列の行数が決定します。所定の行配列の挿入に必要なバッファ・サイズは、次のように計算できます。

```
buffer_size = rows_in_array * maximum_row_size
```

LOB、LONG、BFILE、REF、ROWID、UROWIDまたはTIMESTAMPの各列が含まれている表には、1回に1行ずつ挿入されます。バッファ・サイズは(LOBおよびLONG列の場合以外は)、行全体を格納できるだけの容量が必要です。バッファ・サイズが足りずに表の最長の行を格納できない場合、インポート・ユーティリティはさらに大きいサイズのバッファを割り当てようとします。

DATE列の場合は、バッファの大きさが十分であれば一度に複数の行が挿入されます。

> ノート:
>
> このパラメータのデフォルト値については、ご使用のオペレーティング・システム固有の Oracle マニュアルを参照してください。

親トピック: インポート・パラメータ

27.7.2 COMMIT

デフォルト: n

配列を挿入するたびにコミットするかどうかを指定します。デフォルトでは、各表はロードされた後にのみコミットされ、エラーが発生した場合はロールバックを実行してから次のオブジェクトに進みます。

表にネストした表の列または属性が含まれている場合、ネストした表の内容は、それぞれ別の表としてインポートされます。したがって、ネストした表の内容は、常に、外部表をコミットしたトランザクションとは別のトランザクションとしてコミットされます。

パーティション表の場合にCOMMIT=nを指定すると、エクスポート・ファイルのそれぞれのパーティションおよびサブパーティションは、別のトランザクションとしてインポートされます。

LOB、LONG、BFILE、REF、ROWID、UROWIDまたはTIMESTAMPの各列が含まれている表では、配列の挿入は行われません。COMMIT=yが指定されていると、各行の挿入後に表がコミットされます。

親トピック: インポート・パラメータ

27.7.3 COMPILE

デフォルト: y

パッケージ、プロシージャおよびファンクションを、作成時にインポート・ユーティリティでコンパイルするかどうかを指定します。

COMPILE=nの場合、これらのユニットは最初の使用時にコンパイルされます。たとえば、ドメイン索引作成に使用されるパッケージは、ドメイン索引作成時にコンパイルされます。

関連項目:

「ストアド・プロシージャ、ファンクションおよびパッケージのインポート」

親トピック: インポート・パラメータ

27.7.4 CONSTRAINTS

デフォルト: y

表の制約をインポートするかどうかを指定します。デフォルトでは、制約をインポートします。制約をインポートしないようにするには、このパラメータ値をnに設定します。

IOTおよびオブジェクト表の主キー制約は、常にインポートされます。

親トピック: インポート・パラメータ

27.7.5 DATA_ONLY

DATA_ONLYインポート・パラメータを使用すると、ダンプ・ファイルからデータのみをインポートできます。

デフォルト

n

データのみ(メタデータなし)をダンプ・ファイルからインポートする場合は、DATA_ONLY=yを指定します。

DATA_ONLY=yを指定すると、コマンドライン(またはパラメータ・ファイル)で入力されたインポート・パラメータのうち、メタデータと関連のあるパラメータは無効になります。つまり、ダンプ・ファイルからメタデータはインポートされないことになります。

メタデータに関連するパラメータは、COMPILE、CONSTRAINTS、DATAFILES、DESTROY、GRANTS、IGNORE、INDEXES、INDEXFILE、ROWS=n、SHOW、SKIP_UNUSABLE_INDEXES、STATISTICS、STREAMS_CONFIGURATION、STREAMS_INSTANTIATION、TABLESPACES、TOID_NOVALIDATE、TRANSPORT_TABLESPACE、TTS_OWNERSです。

親トピック: インポート・パラメータ

27.7.6 DATAFILES

DATAFILESインポート・パラメータを使用すると、データベースに転送されるデータ・ファイルをリストできます。

デフォルト

ありません。

TRANSPORT_TABLESPACEにyを指定した場合、データベースに転送するデータ・ファイルを、このパラメータを使用して表示します。

関連項目:

TRANSPORT_TABLESPACE

親トピック: インポート・パラメータ

27.7.7 DESTROY

DESTROYインポート・パラメータでは、データベースを構成する既存のデータ・ファイルを再利用する必要があるかどうかを指定します。

デフォルト

```
n
```

データベースを構成する既存のデータ・ファイルを再利用するかどうかを指定します。つまり、DESTROY=yと指定すると、インポートではSQL CREATE TABLESPACE文のデータ・ファイル句にREUSEオプションが含められ、これにより、オリジナル・データベースのデータ・ファイルの内容が削除された後に、データ・ファイルが再利用されます。

エクスポート・ファイルには、各表領域内で使用されるデータ・ファイル名が含まれていることに注意してください。DESTROY=yを指定して、2つ目のデータベースを(テストまたはその他の目的で)同じシステム上に作成しようとすると、インポート・ユーティリティでは、表領域を作成するときに最初のデータベースのデータ・ファイルを上書きします。このような場合には、デフォルトのDESTROY=nを使用して、表領域が作成されるときにデータ・ファイルがすでに存在する場合、エラーが発生するようにする必要があります。また、オリジナル・データベースにインポートする必要がある場合は、既存のデータ・ファイルを置換しないでデータ・ファイルに追加するように、IGNORE=yを指定する必要があります。

ノート:

データ・ファイルが RAW デバイスに格納されている場合は、 DESTROY=n を指定しても、ファイルは上書きされます。

親トピック: インポート・パラメータ

27.7.8 FEEDBACK

インポート・ユーティリティのFEEDBACKパラメータでは、インポート・ユーティリティにプログレス・バー(n行分のエクスポートを1つのピリオドで示す)を表示するように指定します。

デフォルト: 0 (ゼロ)

n行分のインポートを1つのピリオドで示すプログレス・バーの表示を指定します。たとえば、FEEDBACK=10を指定すると、10行分のインポートが終了するたびにピリオドが1つ表示されます。FEEDBACK値は、インポートされるすべての表に適用されるため、各表に対して個別には設定できません。

親トピック: インポート・パラメータ

27.7.9 FILE

FILEインポート・ユーティリティ・パラメータでは、インポートするエクスポート・ファイルの名前を指定します。

デフォルト: expdat.dmp

説明

インポートするエクスポート・ファイル名を指定します。デフォルトの拡張子は、.dmpです。エクスポート・ユーティリティは複数のエクスポート・ファイルをサポートしているため、インポートする複数のファイル名の指定が必要になる場合があります。

エクスポート・ファイルをエクスポートしたユーザーである必要はありません。ただし、そのファイルへの読取りアクセス権限が必要です。自分のユーザーIDでファイルをエクスポートしなかった場合は、IMP_FULL_DATABASEロールも付与されている必要があります。

例

```
imp scott IGNORE=y FILE = dat1.dmp, dat2.dmp, dat3.dmp FILESIZE=2048
```

親トピック: インポート・パラメータ

27.7.10 FILESIZE

FILESIZEインポート・ユーティリティ・パラメータを使用すると、エクスポート時に指定したサイズと同じ最大ダンプ・ファイル・サイズを指定できます。

デフォルト: オペレーティング・システムによって異なります。

エクスポート時に指定したサイズと同じ最大ダンプ・ファイル・サイズを指定できます。

ノート:

許容最大サイズは、オペレーティング・システムによって異なります。この最大値については、FILESIZEを指定する前に、ご使用のオペレーティング・システム固有のOracleマニュアルで確認してください。

FILESIZEの値は、数字にKB(キロバイト数)を付けて指定できます。たとえば、FILESIZE=2KBは、FILESIZE=2048と同じです。同様に、MBはメガバイト(1024×1024)を、GBはギガバイト(1024の3乗)を表します。Bはバイトの省略です。この場合、最終的なファイル・サイズの算出に乗算は不要です(FILESIZE=2048Bは、FILESIZE=2048と同じです)。

親トピック: インポート・パラメータ

27.7.11 FROMUSER

インポート・ユーティリティのFROMUSERパラメータを使用すると、複数のスキーマを含むエクスポート・ファイルからスキーマのサブセットをインポートできます。

デフォルト: なし

インポートするスキーマをカンマで区切ったリスト。このパラメータは、IMP_FULL_DATABASEロールを持つユーザーにのみ関係があります。このパラメータで、複数のスキーマを含むエクスポート・ファイル(たとえば、全体エクスポート・ダンプ・ファイルまたは複数スキーマのユーザー、ユーザー・モードのエクスポート・ダンプ・ファイルなど)からスキーマのサブセットをインポートできます。

ファンクション索引、ファンクション、プロシージャ、トリガー、型本体、ビューなどの内部に表示されるスキーマ名は、FROMUSERまたはTOUSERの処理には影響されません。影響を受けるのは、オブジェクト名のみです。インポートの完了後、TOUSERスキーマに含まれる項目が古いスキーマ(FROMUSER)を参照しているかどうかを手動で確認し、必要に応じて修正する必要があります。

通常は、インポート・パラメータTOUSERとFROMUSERを組み合せて使用し、インポートのターゲットとなるスキーマの所有者ユーザー名のリストを指定します。インポート操作の前にターゲット・データベース内にTOUSERで指定するユーザーが存在している必要があります。存在していない場合、エラーが返されます。

TOUSERを指定しない場合、インポート操作では次のことが行われます。

- エクスポート・ファイルが、全データベース・モードのダンプ・ファイル、または複数スキーマ、ユーザー・モードのエクスポート・ダンプ・ファイルの場合、FROMUSERのスキーマへオブジェクトをインポートします。
- エクスポート・ファイルが単一のスキーマで、ユーザー・モードのエクスポート・ダンプ・ファイルが権限のないユーザーに作成された場合、(インポート時にFROMUSERスキーマが存在するかどうかにかかわらず)インポートするユーザーのスキーマにオブジェクトを作成します。

ノート:

FROMUSER=SYSTEM を指定しても、システム・オブジェクトはインポートされず、ユーザーSYSTEM が所有するスキーマ・オブジェクトのみがインポートされます。

親トピック: インポート・パラメータ

27.7.12 FULL

FULLインポート・ユーティリティ・パラメータでは、エクスポート・ダンプ・ファイル全体をインポートするかどうかを指定します。

デフォルト: y

エクスポート・ダンプ・ファイル全体をインポートするかどうかを指定します。

- 全データベースのエクスポートおよびインポートについての考慮点
  全データベースのエクスポートおよびインポートは、データベースの複製およびクリーン・アップに有効です。

親トピック: インポート・パラメータ

27.7.12.1 全データベースのエクスポートおよびインポートについての考慮点

全データベースのエクスポートおよびインポートは、データベースの複製およびクリーンアップに有効です。

ただし、問題が発生しないように、次の点に注意する必要があります。

- 全体エクスポートでは、スキーマSYSが所有するトリガーはエクスポートされません。SYSトリガーは、全体インポートの前

後のいずれかに手動で再作成する必要があります。このトリガーによってインポートの進行を妨げるような処理が定義されないように、インポートの後に再作成することをお薦めします。

- 全体エクスポートでは、デフォルトのプロファイルもエクスポートされません。ソース・データベースでデフォルトのプロファイルを変更した場合(たとえば、スキーマSYSが所有するパスワード確認機能を追加するなど)は、事前に手動でその機能を作成し、インポートの完了後にターゲット・データベースでデフォルトのプロファイルを変更する必要があります。
- 可能な場合、開始前にエクスポートされたデータベースおよびインポートするデータベースの物理コピーを作成します。これによって、誤った操作をやり直すことができます。
- エクスポートの開始前に、次の情報を含むレポートを作成することをお薦めします。
  - 表領域およびデータ・ファイルのリスト
  - ロールバック・セグメントのリスト
  - ユーザーごとの各オブジェクト型(表、索引など)の数

  この情報で、表領域がすでに作成され、インポートが正常に完了したことを確認できます。
- エクスポートから完全に新しいデータベースを作成する場合、SYSTEMに余分なロールバック・セグメントを作成し、初期化パラメータ・ファイル(`init.ora`)で使用可能にしてからインポートを開始します。
- インポートの実行時に、正しいインスタンスを指していることを確認します。一部のUNIXシステムでは、サブシェルの入力のみで、インポート操作を行ったデータベースが変更される場合があります。
- すべての表領域が作成されるまで、複数のデータベースを含むシステムでは全体インポートを行わないでください。全体インポートは、エクスポートされたデータベースと同じデータ・ファイル名を使用して、未定義の表領域を作成します。このため、次のような状況で問題が発生します。
  - データ・ファイルが他のいずれかのデータベースに属する場合、データ・ファイルは破損します。このことは特に、エクスポートされるデータベースが同じシステムに存在する場合に当てはまります(そのデータ・ファイルがインポート先のデータベースによって再利用されるため)。
  - データ・ファイルの名前が既存のオペレーティング・システム・ファイルの名前と同じ場合も問題が発生します。

親トピック: FULL

27.7.13 GRANTS

オブジェクト権限をインポートするかどうかを指定します。

デフォルト: y

デフォルトでは、エクスポートされたオブジェクト権限はすべてインポートされます。ユーザー・モードでエクスポートが実行されている場合は、第1レベルのオブジェクト権限(所有者によって付与されているもの)のみがエクスポート・ファイルにインポートされます。

全データベース・モードでエクスポートが実行されている場合は、下位レベルのオブジェクト権限(`WITH GRANT OPTION`で権限が付与されたユーザーによって付与されているもの)を含むすべての権限が、エクスポート・ファイルにインポートされます。`GRANTS=n`を指定すると、オブジェクト権限はインポートされません(`GRANTS=n`を指定しても、システム権限はインポートされます)。

ノート:

エクスポート・ユーティリティでは、セキュリティ上の理由から、インポートに影響するデータ・ディクショナリ・ビューの権限はエクスポートされません。このような権限がエクスポートされると、インポートしたユーザーが気付かないうちに、アクセス権が変更される場合があります。

親トピック: インポート・パラメータ

27.7.14 HELP

インポート・ユーティリティのHELPパラメータを使用すると、インポート・パラメータの説明が表示されます。

デフォルト: なし

インポート・パラメータの説明を表示します。ヘルプの内容を表示するには、コマンドラインで`imp HELP=y`を入力します。

親トピック: インポート・パラメータ

27.7.15 IGNORE

IGNOREインポート・ユーティリティ・パラメータでは、オブジェクト作成エラーをどのように処理する必要があるかを指定します。

デフォルト: n

オブジェクト作成エラーの処理方法を指定します。デフォルトの`IGNORE=n`が指定されている場合は、オブジェクト作成エラーがログに記録または表示されて、インポートが続行されます。

`IGNORE=y`を指定すると、データベース・オブジェクトの作成時に作成エラーが発生しても、このエラーは無視され、エラーはレポートされずに継続します。

無視されるのはオブジェクト作成エラーのみです。オペレーティング・システム、データベース、SQLなどのエラーは無視されず、場合によっては処理が停止します。

`IGNORE=y`が指定され、1つのエクスポート・ファイルから何回もリフレッシュが行われる場合、オブジェクトが何回も作成される場合があります(ただし、各オブジェクトには一意のシステム定義名が付けられます)。特定のオブジェクト(たとえば、制約など)に対しては、`CONSTRAINTS=n`を指定してインポートを実行すると、この問題を回避できます。`CONSTRAINTS=n`を指定して全インポートを実行すると、表の制約はインポートされません。

表がすでに存在する場合、`IGNORE=y`を指定すると、行は既存の表にインポートされ、エラーやメッセージは出力されません。新しい記憶域パラメータを使用するため、またはクラスタにすでに表を作成済であるため、すでに存在する表にデータをインポートする場合があります。

表がすでに存在する場合、`IGNORE=n`を指定すると、エラーがレポートされ、表は、行が挿入されないままスキップされます。また、表に依存するオブジェクト(索引、権限、制約など)は作成されません。

ノート:

既存の表へのインポート時に、表の列の索引が一意でない場合、行データが重複することがあります。

親トピック: インポート・パラメータ

27.7.16 INDEXES

インポート・ユーティリティのINDEXESパラメータでは、索引をインポートするかどうかを指定します。

デフォルト: y

索引をインポートするかどうかを指定します。LOB索引、OID索引、一意制約索引など、システムによって作成される索引は、このパラメータの指定に関係なく、インポート・ユーティリティによって自動的に再作成されます。

INDEXES=nを指定すると、すべてのユーザー作成索引をインポートの終了後に作成できます。

インポート時、ターゲット表にすでに索引が存在する場合は、データ挿入時にターゲット表の索引のメンテナンスを実行します。

親トピック: インポート・パラメータ

27.7.17 INDEXFILE

インポート・ユーティリティのINDEXFILEパラメータでは、索引作成文を受け取るファイルを指定します。

デフォルト: なし

索引作成文を受け取るファイルを指定します。

このパラメータを指定すると、指定したモードでの索引作成文は、データベース中に索引を作成するために使用されるのではなく、抽出されて指定のファイルに書き込まれます。データベース・オブジェクトはインポートされません。

インポート・パラメータCONSTRAINTSにyを設定している場合、索引ファイルに表制約も書き込まれます。

その後、このファイルを編集して(記憶域パラメータの変更など)、索引を作成するためのSQLスクリプトとして使用できます。

ファイル内で定義されている索引をさらに簡単に識別するために、エクスポート・ファイルのCREATE TABLE文およびCREATE CLUSTER文がコメントとして含まれます。

この機能を使用するには、次のステップを実行します。

1. INDEXFILEパラメータを使用してインポートを行い、索引作成文のファイルを作成します。
2. ファイルを編集して、有効なパスワードをconnect文字列に追加します。
3. INDEXES=nを指定してインポートを再実行します。

   (このステップでデータベース・オブジェクトがインポートされますが、エクスポート・ファイルに格納されている索引定義は使用されません。)
4. 索引作成文のファイルをSQLスクリプトとして実行し、索引を作成します。

   INDEXFILEパラメータを指定できるのは、FULL=y、FROMUSER、TOUSERまたはTABLESパラメータを指定した場合のみです。

親トピック: インポート・パラメータ

27.7.18 LOG

情報メッセージおよびエラー・メッセージを受け取るファイル(import.logなど)を指定します。

デフォルト: なし

ログ・ファイルを指定すると、端末画面およびログ・ファイルの両方にインポートに関する情報が書き込まれます。

親トピック: インポート・パラメータ

27.7.19 PARFILE

インポート・パラメータのリストを格納するファイルのファイル名を指定します。

デフォルト: なし

パラメータ・ファイルの使用方法の詳細は、「パラメータ・ファイル」を参照してください。

親トピック: インポート・パラメータ

27.7.20 RECORDLENGTH

ファイル・レコードの長さをバイト単位で指定します。

デフォルト

オペレーティング・システムによって異なります。

用途

RECORDLENGTHパラメータは、異なるデフォルト値を使用する別のオペレーティング・システムにエクスポート・ファイルを転送する場合に指定する必要があります。

このパラメータを指定しない場合、ご使用のプラットフォーム固有のBUFSIZの値がデフォルトになります。

RECORDLENGTHは、ご使用のシステムのBUFSIZの値以上の任意の値に設定できます。(最大値は64KBです。)RECORDLENGTHパラメータの変更により影響を受けるのは、データベースに書き出す前に累積されるデータのサイズのみです。オペレーティング・システム・ファイルのブロック・サイズには影響しません。

このパラメータは、インポート・ユーティリティのI/Oバッファのサイズ指定にも使用できます。

親トピック: インポート・パラメータ

27.7.21 RESUMABLE

RESUMABLEパラメータを使用して、再開可能な領域割当てを有効または無効にします。

デフォルト

n

用途

このパラメータはデフォルトでは無効なため、関連するRESUMABLE_NAMEおよびRESUMABLE_TIMEOUTを使用するには、RESUMABLE=yを設定する必要があります。

関連項目:

再開可能な領域割当ての詳細は、『Oracle Database管理者ガイド』を参照してください

親トピック: インポート・パラメータ

27.7.22 RESUMABLE_NAME

RESUMABLE_NAMEパラメータの値によって、再開可能な文を識別します。

デフォルト

```
'User USERNAME (USERID), Session SESSIONID, Instance INSTANCEID'
```

用途

この値はユーザー定義のテキスト文字列で、USER_RESUMABLEまたはDBA_RESUMABLEビューに挿入して、一時停止されている特定の再開可能な文を識別できます。

RESUMABLEパラメータをyに設定して、再開可能な領域割当てを有効にしないかぎり、このパラメータは無視されます。

親トピック: インポート・パラメータ

27.7.23 RESUMABLE_TIMEOUT

RESUMABLE_TIMEOUTパラメータの値では、エラー修正までに必要な時間を指定します。

デフォルト

7200秒(2時間)

用途

タイムアウト時間内にエラーを修正できない場合は、文の実行が途中で終了します。

RESUMABLEパラメータをyに設定して、再開可能な領域割当てを有効にしないかぎり、このパラメータは無視されます。

親トピック: インポート・パラメータ

27.7.24 ROWS

表のデータ行をインポートするかどうかを指定します。

デフォルト

y

用途

ROWS=nを指定すると、インポート操作の終了後、インポートされたすべての表の統計がロックされます。

親トピック: インポート・パラメータ

27.7.25 SHOW

インポートの前に、エクスポート・ファイルの内容を表示します。

デフォルト

n

構文および説明

SHOW=yを指定すると、エクスポート・ダンプ・ファイルの内容が画面に表示されますが、インポートは実行されません。エクスポート・ファイルに含まれるSQL文は、インポート・ユーティリティでその文が実行される順序で表示されます。

SHOWパラメータを指定できるのは、FULL=y、FROMUSER、TOUSERまたはTABLESパラメータを指定した場合のみです。

親トピック: インポート・パラメータ

27.7.26 SKIP_UNUSABLE_INDEXES

SKIP_UNUSABLE_INDEXESパラメータは、インポート・ユーティリティおよびOracle Databaseの両方で提供されます。

デフォルト: 初期化パラメータ・ファイルで指定した、Oracle Databaseの構成パラメータSKIP_UNUSABLE_INDEXESの値

インポート・ユーティリティのSKIP_UNUSABLE_INDEXESパラメータは、インポート・ユーティリティのコマンドラインで指定します。Oracle DatabaseのSKIP_UNUSABLE_INDEXESパラメータは、初期化パラメータ・ファイルの構成パラメータとして指定します。相互に与える影響を理解しておく必要があります。

インポート・ユーティリティのコマンドラインでSKIP_UNUSABLE_INDEXESの値を指定しない場合、初期化パラメータ・ファイルで指定した、SKIP_UNUSABLE_INDEXES構成パラメータのデータベース設定が使用されます。

インポート・コマンドラインでSKIP_UNUSABLE_INDEXESの値を指定すると、インポートは、初期化パラメータ・ファイルに指定されているSKIP_UNUSABLE_INDEXES構成パラメータの値を上書きします。

yの値を指定すると、(システムまたはユーザーのいずれかによって)索引使用禁止状態に設定されている索引の作成が、インポート・ユーティリティではスキップされます。他の索引(事前に索引使用禁止に設定されていない索引)に対しては、行の挿入時にメンテナンス処理が行われます。

このパラメータを使用すると、選択した索引パーティションに対する索引のメンテナンスを、行データの挿入が完了するまで延期できます。インポート後、影響を受けた索引パーティションの再作成が必要です。

ノート:

一意で使用禁止状態の索引に対しては、索引メンテナンスをスキップできません。したがって、一意の索引はSKIP_UNUSABLE_INDEXES パラメータの影響を受けません。

INDEXES=nを指定してINDEXFILEパラメータを使用すると、索引の再作成を行うSQLスクリプトを作成できます。SKIP_UNUSABLE_INDEXESパラメータを指定しない場合、行挿入によって使用禁止索引をメンテナンスすると、その行挿入によってエラーが発生します。

関連項目:

『Oracle Database SQL言語リファレンス』のALTER SESSION文についての説明を参照してください

親トピック: インポート・パラメータ

27.7.27 STATISTICS

インポート時のデータベース・オプティマイザ統計の処理方法を指定します。

デフォルト: ALWAYS

メニュー項目は次のとおりです。

- ALWAYS

  データベース・オプティマイザ統計に問題があるかどうかにかかわらず、常にインポートします。

- NONE

  データベース・オプティマイザ統計をインポートまたは再計算しません。

- SAFE

  データベース・オプティマイザ統計に問題がない場合のみにインポートします。問題がある場合は、オプティマイザ統計を再計算します。

- RECALCULATE

  データベース・オプティマイザ統計をインポートしません。かわりに、インポート時に再計算します。この場合、ダンプ・ファイ

ルを作成した元のエクスポート操作で、必要なANALYZE文を生成している(STATISTICS=NONEを指定してエクスポートを実行していない)必要があります。ANALYZE文は、ダンプ・ファイルに含まれ、インポート操作で表統計の再計算に使用されます。

関連項目:

- オプティマイザおよびオプティマイザが使用する統計の詳細は、『Oracle Database概要』を参照してください。
- 統計情報のインポート

親トピック: インポート・パラメータ

27.7.28 STREAMS_CONFIGURATION

エクスポート・ダンプ・ファイル内に存在する可能性のあるすべての一般的なGoldenGateレプリケーション・メタデータをインポートするかどうかを指定します。

デフォルト: y

親トピック: インポート・パラメータ

27.7.29 STREAMS_INSTANTIATION

エクスポート・ダンプ・ファイル内に存在するStreamsインスタンス化メタデータをインポートするかどうかを指定します。

デフォルト: n

ストリーム環境でインスタンス化の一部としてインポートする場合は、yを指定します。

親トピック: インポート・パラメータ

27.7.30 TABLES

デフォルト: なし

表モード・インポートでインポートすることを指定します。インポートの対象となる表名、パーティション名およびサブパーティション名をリストとして指定します。表レベル・インポートでは、パーティション表または非パーティション表全体をインポートできます。TABLESパラメータでは、インポート対象は、表およびその関連オブジェクトに限定されます(「インポート・モード」を参照してください)。TABLESパラメータでは、次の値を指定できます。

- tablenameには、インポートされる表の名前を指定します。リストにパーティション表が含まれる場合、パーティション名を指定しないと、すべてのパーティションおよびサブパーティションがインポートされます。エクスポートされたすべての表をインポートするには、アスタリスク(*)のみを表名パラメータとして指定します。

  tablenameは、任意の数の「%」パターン一致文字を含むことができます。各「%」パターン一致文字は、エクスポート・ファイルの表名の0個以上の文字と一致します。リスト中の固有の表名の指定されたすべてのパターンと一致する名前を持つすべての表が、選択されてインポートされます。すべてのパターン一致文字で構成され、パーティション名を含まないリスト中の表名によって、エクスポートされたすべての表がインポートされます。

- partition_nameおよびsubpartition_nameによって、パーティション表内で指定された1つ以上のパーティションまたはサブパーティションにインポートが制限されます。

構文の形式は、次のとおりです。

```
tablename:partition_name
tablename:subpartition_name
```

tablename:partition_nameを使用する場合、指定された表はパーティション化されている必要があり、partition_nameはそのいずれかのパーティションまたはサブパーティションの名前である必要があります。指定された表がパーティション化されていない場合、partition_nameは無視されて表全体がインポートされます。

一度に指定できる表の数は、コマンドラインの制限によって決まります。

エクスポート・ファイルの処理中、エクスポート・ファイルの各表名は、パラメータで指定された順に、リスト中の各表名と比較されます。あいまいな処理が行われたり、処理時間が極端に長くならないように、固有の表名がリストの始めに表示され、一般的な表名(パターンを使用したもの)がリストの終わりに表示される必要があります。

エクスポート時には表名をスキーマ名(scott.empなど)で修飾できますが、インポート時にはできません。次に、間違って指定されたTABLESパラメータの例を示します。

```
imp TABLES=(jones.accts, scott.emp, scott.dept)
```

これらの表をインポートするには、次のように指定します。

```
imp FROMUSER=jones TABLES=(accts)
imp FROMUSER=scott TABLES=(emp,dept)
```

詳細は、「パターン一致を使用して様々な表をインポートする例」を参照してください。

ノート:

UNIX など一部のオペレーティング・システムで、カッコなどの特殊文字を使用する場合は、特殊文字として扱われないように、その文字の前にエスケープ文字を使用する必要があります。UNIX では、次の例に示すように、エスケープ文字としてバックスラッシュ(¥)を使用します。

```
TABLES=¥(emp,dept¥)
```

- 表名の制限
  インポート・ユーティリティの表名の制限についての説明します。

親トピック: インポート・パラメータ

27.7.30.1 表名の制限

インポート・ユーティリティの表名の制限について説明します。

表名には次の制限があります。

- デフォルトでは、表名は大文字でデータベースに格納されます。表名が大文字と小文字または小文字のみで表記され、大/小文字を区別する場合、名前を引用符で囲む必要があります。したがって、表名は、データベースに格納されている表名と完全に一致するように指定する必要があります。

  ただし、オペレーティング・システムによっては、コマンドラインの引用符自体をエスケープする必要がある場合があります。次に、異なるインポート・モードで大/小文字の区別を保持する方法を示します。

  - コマンドライン・モード

    ```
    tables='¥"Emp¥"'
    ```

- 対話方式モード

```
Table(T) to be exported: "Exp"
```

- パラメータ・ファイル・モード

```
tables='"Emp"'
```

- 表名を引用符で囲まないかぎり、コマンドラインで指定する表名にポンド(#)記号は使用できません。同様に、パラメータ・ファイルでは、表名が引用符で囲まれていないかぎり、表名にポンド(#)記号を使用すると、インポート・ユーティリティではポンド(#)記号より右側の文字がコメントとして解釈されます。

  たとえば、パラメータ・ファイルに次の行が記述されている場合、インポート・ユーティリティはemp#の右側をすべてコメントとして解釈するため、表deptとmydataはインポートされません。

```
TABLES=(emp#, dept, mydata)
```

  ただし、次の例では、emp#が引用符で囲まれているため、3つの表はすべてインポートされます。

```
TABLES=("emp#", dept, mydata)
```

> ノート:
>
> オペレーティング・システムによっては、一重引用符を使用する必要がある場合と、二重引用符を使用する必要がある場合があります。ご使用のオペレーティング・システム固有のドキュメントで確認してください。表のネーミング方法に制限があるオペレーティング・システムもあります。
>
> たとえば、UNIX の C シェルではドル記号($)やポンド記号(#)(またはその他の特殊文字)には特別な意味があります。これらの文字をシェルを介してインポートするには、エスケープ文字を使用する必要があります。

親トピック: TABLES

27.7.31 TABLESPACES

インポート・ユーティリティのTABLESPACESパラメータ。

デフォルト: なし

TRANSPORT_TABLESPACEにyを指定した場合、データベースに転送する表領域を、このパラメータを使用して表示します。エクスポート・ファイルに複数の表領域がある場合は、そのすべての表領域をインポート操作の一部として指定する必要があります。

詳細は、「TRANSPORT_TABLESPACE」を参照してください。

親トピック: インポート・パラメータ

27.7.32 TOID_NOVALIDATE

TOID_NOVALIDATEパラメータは、TOIDと比較しない型を指定するために使用します。

デフォルト: なし

型を参照している表のインポート時に、その名前の型がすでにデータベースに存在している場合は、その既存の型が、実際にその表で使用されているかどうか(実際は異なる型で、単に同じ名前であるだけではないか)を確認します。

この検証のために、型の一意の識別子(TOID)と、エクスポート・ファイルに格納された識別子が比較されます。TOIDが一致しない場合、その表の行はインポートされません。

この妥当性チェックをしてはいけない型もあります(たとえば、その型がカートリッジのインストールによって作成された場合)。TOID_NOVALIDATEパラメータを使用して、TOIDと比較しない型を指定できます。

構文は次のとおりです。

```
TOID_NOVALIDATE=([schemaname.]typename [, ...])
```

たとえば:

```
imp scott TABLES=jobs TOID_NOVALIDATE=typ1
imp scott TABLES=salaries TOID_NOVALIDATE=(fred.typ0,sally.typ2,typ3)
```

その型にスキーマ名を指定しない場合、インポートするユーザーのスキーマがデフォルトになります。たとえば、前述の最初の例では、型typ1はscott.typ1がデフォルトになり、2番目の例では、型typ3はscott.typ3がデフォルトになります。

TOID_NOVALIDATEは、表の列型のみを処理します。表型には影響しません。

通常のインポートでは、削除される型が含まれていると、次のように出力されます。

```
[...]
. importing IMP3's objects into IMP3
. . skipping TOID validation on type IMP2.TOIDTYP0
. . importing table                  "TOIDTAB3"
[...]
```

ノート:

型の識別子を比較しないように指定する場合は、ユーザーの責任において、インポートされる型の属性リストを既存の型の属性リストと一致させるようにしてください。これらの属性リストが一致しない場合、インポート結果は保証されません。

親トピック: インポート・パラメータ

27.7.33 TOUSER

インポートの対象となるスキーマを所有するユーザー名のリストを指定します。

デフォルト: なし

インポート操作の前にユーザー名が存在している必要があります。存在していない場合は、エラーが返されます。このパラメータを使用するには、IMP_FULL_DATABASEロールが必要です。オブジェクトがもともと入っていたスキーマと異なるスキーマにインポートする場合は、TOUSERを指定してください。たとえば:

```
imp FROMUSER=scott TOUSER=joe TABLES=emp
```

複数のスキーマを指定する場合、スキーマ名は対で指定します。次の例では、scottのオブジェクトをjoeのスキーマにインポートし、fredのオブジェクトをtedのスキーマにインポートします。

```
imp FROMUSER=scott,fred TOUSER=joe,ted
```

FROMUSERリストがTOUSERリストより長い場合、残りのスキーマは、通常のデフォルトの規則に従って、FROMUSERスキーマにインポートされるか、またはインポートを実行するユーザーのスキーマにインポートされます。余分なオブジェクトがTOUSERスキーマにインポートされるようにするには、次の構文を使用します。

```
imp FROMUSER=scott,adams TOUSER=ted,ted
```

ユーザーtedは2回指定されています。

関連項目:

FROMUSERおよびTOUSERを使用する場合の制限の詳細は、「FROMUSER」を参照してください

親トピック: インポート・パラメータ

27.7.34 TRANSPORT_TABLESPACE

yを指定した場合、エクスポート・ファイルからトランスポータブルの表領域メタデータがインポートされます。

デフォルト: n

暗号化された列は、トランスポータブル表領域モードではサポートされていません。

ノート:

トランスポータブル表領域をエクスポートした後、それよりも古いリリース・レベルのデータベースにインポートすることはできません。ターゲット・データベースのリリース・レベルは、ソース・データベース以上である必要があります。

親トピック: インポート・パラメータ

27.7.35 TTS_OWNERS

TRANSPORT_TABLESPACEにyを指定した場合、このパラメータを使用して、一連のトランスポータブル表領域のデータの所有者ユーザーを表示できます。

デフォルト: なし

「TRANSPORT_TABLESPACE」を参照してください。

親トピック: インポート・パラメータ

27.7.36 USERID (username/password)

インポートを実行するユーザーのユーザー名、パスワードおよびオプションの接続文字列を指定します。

デフォルト: なし

ユーザーSYSとして接続する場合は、接続文字列にAS SYSDBAも指定する必要があります。オペレーティング・システムによっては、AS SYSDBAを特殊文字列とみなし、その文字列全体を引用符で囲む必要があります。

関連項目:

Oracle Netに接続文字列を指定する方法の詳細は、ご使用のOracle Netプロトコルのユーザーズ・ガイドを参照してくださ

い。

親トピック: インポート・パラメータ

27.7.37 VOLSIZE

各テープ・ボリュームのダンプ・ファイルに最大バイト数を指定します。

デフォルト: なし

VOLSIZEパラメータの最大値は、ご使用のプラットフォーム上に64ビットで格納できる最大値と同じです。

VOLSIZEの値は、数字にKB(キロバイト数)を付けて指定できます。たとえば、VOLSIZE=2KBは、VOLSIZE=2048と同じです。同様に、MBはメガバイト(1024×1024)を、GBはギガバイト(1024の3乗)を表します。Bはバイトの省略です。この場合、最終的なファイル・サイズの算出に乗算は不要です(VOLSIZE=2048Bは、VOLSIZE=2048と同じです)。

親トピック: インポート・パラメータ

27.8 インポート・セッションの例

これらの項では、インポート・セッションの例をいくつか取り上げ、パラメータ・ファイル方式およびコマンドライン方式の使用方法を示します。

- 特定のユーザーの表を選択してインポートする例
- 別のユーザーによってエクスポートされた表をインポートする例
- あるユーザーの表を別のユーザーへインポートする例
- パーティション・レベル・インポートでのインポート・セッションの例
- パターン一致を使用して様々な表をインポートする例

親トピック: オリジナルのインポート

27.8.1 特定のユーザーの表を選択してインポートする例

この例では、管理者が全データベース・エクスポート・ファイルを使用して、dept表およびemp表をscottのスキーマにインポートします。

パラメータ・ファイル方式

```
> imp PARFILE=params.dat
```

params.datファイルには、次の情報が格納されています。

```
FILE=dba.dmp
SHOW=n
IGNORE=n
GRANTS=y
FROMUSER=scott
TABLES=(dept,emp)
```

コマンドライン方式

```
> imp FILE=dba.dmp FROMUSER=scott TABLES=(dept,emp)
```

インポート・メッセージ

使用しているインポート・ユーティリティのリリース情報および接続しているOracle Databaseのリリース情報が表示されます。ま

た、状態メッセージも表示されます。

親トピック: インポート・セッションの例

27.8.2 別のユーザーによってエクスポートされた表をインポートする例

この例では、blakeがエクスポートしたファイルからunit表およびmanager表をscottのスキーマにインポートします。

パラメータ・ファイル方式

```
> imp PARFILE=params.dat
```

params.datファイルには、次の情報が格納されています。

```
FILE=blake.dmp
SHOW=n
IGNORE=n
GRANTS=y
ROWS=y
FROMUSER=blake
TOUSER=scott
TABLES=(unit,manager)
```

コマンドライン方式

```
> imp FROMUSER=blake TOUSER=scott FILE=blake.dmp TABLES=(unit,manager)
```

インポート・メッセージ

使用しているインポート・ユーティリティのリリース情報および接続しているOracle Databaseのリリース情報が表示されます。また、状態メッセージも表示されます。

親トピック: インポート・セッションの例

27.8.3 あるユーザーの表を別のユーザーへインポートする例

この例では、データベース管理者(DBA)がユーザーscottのすべての表をユーザーblakeのアカウントにインポートします。

パラメータ・ファイル方式

```
 > imp PARFILE=params.dat
```

params.datファイルには、次の情報が格納されています。

```
FILE=scott.dmp
FROMUSER=scott
TOUSER=blake
TABLES=(*)
```

コマンドライン方式

```
> imp FILE=scott.dmp FROMUSER=scott TOUSER=blake TABLES=(*)
```

インポート・メッセージ

使用しているインポート・ユーティリティのリリース情報および接続しているOracle Databaseのリリース情報が表示されます。その後で、次に示すような状態メッセージが表示されます。

```
.
.
.
```

```
Warning: the objects were exported by SCOTT, not by you
import done in WE8DEC character set and AL16UTF16 NCHAR character set
. importing SCOTT's objects into BLAKE
. . importing table                          "BONUS"          0 rows imported
. . importing table                           "DEPT"          4 rows imported
. . importing table                            "EMP"         14 rows imported
. . importing table                        "SALGRADE"          5 rows imported
Import terminated successfully without warnings.
```

親トピック: インポート・セッションの例

27.8.4 パーティション・レベル・インポートでのインポート・セッションの例

この項では、複数のパーティションがある表、パーティションとサブパーティションがある表、および異なる列で再パーティション化した表のインポートについて説明します。

- 例1: パーティション・レベル・インポート
- 例2: コンポジット・パーティション表のパーティション・レベル・インポート
- 例3: 別の列での表の再パーティション化

親トピック: インポート・セッションの例

27.8.4.1 例1: パーティション・レベル・インポート

この例では、empは、P1、P2およびP3で構成されているパーティション表です。

表レベルのエクスポート・ファイルを作成するには、次のコマンドを使用します。

```
> exp scott TABLES=emp FILE=exmpexp.dat ROWS=y
```

エクスポート・メッセージ

使用しているエクスポート・ユーティリティのリリース情報および接続しているOracle Databaseのリリース情報が表示されます。その後で、次に示すような状態メッセージが表示されます。

```
.
.
.
About to export specified tables via Conventional Path ...
. . exporting table                             EMP
. . exporting partition                              P1          7 rows exported
. . exporting partition                              P2         12 rows exported
. . exporting partition                              P3          3 rows exported
Export terminated successfully without warnings.
```

パーティション・レベルのインポートでは、インポートの対象に、エクスポートした表の特定のパーティションを指定できます。この例では、emp表のP1およびP3を指定します。

```
> imp scott TABLES=(emp:p1,emp:p3) FILE=exmpexp.dat ROWS=y
```

インポート・メッセージ

使用しているインポート・ユーティリティのリリース情報および接続しているOracle Databaseのリリース情報が表示されます。また、状態メッセージも表示されます。

親トピック: パーティション・レベル・インポートでのインポート・セッションの例

27.8.4.2 例2: コンポジット・パーティション表のパーティション・レベル・インポート

この例では、コンポジット・パーティション表のパーティションおよびサブパーティションがインポートされます。empは、2つのコンポジット・パーティションP1およびP2のパーティション表です。パーティションP1には、3つのサブパーティションP1_SP1、P1_SP2およびP1_SP3があります。パーティションP2には、2つのサブパーティションP2_SP1およびP2_SP2があります。

表レベルのエクスポート・ファイルを作成するには、次のコマンドを使用します。

```
> exp scott TABLES=emp FILE=exmpexp.dat ROWS=y
```

エクスポート・メッセージ

使用しているエクスポート・ユーティリティのリリース情報および接続しているOracle Databaseのリリース情報が表示されます。その後で、次に示すような状態メッセージが表示されます。

コマンドラインで実行されると次のエクスポート・メッセージが表示されます。

```
.
.
.
About to export specified tables via Conventional Path ...
. . exporting table                          EMP
. . exporting composite partition                P1
. . exporting subpartition                    P1_SP1          2 rows exported
. . exporting subpartition                    P1_SP2         10 rows exported
. . exporting subpartition                    P1_SP3          7 rows exported
. . exporting composite partition                P2
. . exporting subpartition                    P2_SP1          4 rows exported
. . exporting subpartition                    P2_SP2          2 rows exported
Export terminated successfully without warnings.
```

次のインポート・コマンドでは、emp表にあるコンポジット・パーティションP1のサブパーティションP1_SP2およびP1_SP3と、emp表にあるコンポジット・パーティションP2のすべてのサブパーティションがインポートされます。

```
> imp scott TABLES=(emp:p1_sp2,emp:p1_sp3,emp:p2) FILE=exmpexp.dat ROWS=y
```

インポート・メッセージ

使用しているインポート・ユーティリティのリリース情報および接続しているOracle Databaseのリリース情報が表示されます。その後で、次に示すような状態メッセージが表示されます。

```
.
.
.
. importing SCOTT's objects into SCOTT
. . importing subpartition              "EMP":"P1_SP2"         10 rows imported
. . importing subpartition              "EMP":"P1_SP3"          7 rows imported
. . importing subpartition              "EMP":"P2_SP1"          4 rows imported
. . importing subpartition              "EMP":"P2_SP2"          2 rows imported
Import terminated successfully without warnings.
```

親トピック: パーティション・レベル・インポートでのインポート・セッションの例

27.8.4.3 例3: 別の列での表の再パーティション化

この例では、emp表に、empno列に基づく2つのパーティションがあります。emp表をdeptno列で再パーティション化します。

別の列で表を再パーティション化するには、次のステップを実行してください。

1. エクスポートを実行して、データを保存します。
2. データベースから表を削除します。

3. 表を新しいパーティションに分割して再作成します。
4. 表データをインポートします。

次に、これらのステップの例を示します。

```
> exp scott table=emp file=empexp.dat
.
.
.
About to export specified tables via Conventional Path ...
. . exporting table                          EMP
. . exporting partition                      EMP_LOW          4 rows exported
. . exporting partition                     EMP_HIGH         10 rows exported
Export terminated successfully without warnings.
SQL> connect scott
Connected.
SQL> drop table emp cascade constraints;
Statement processed.
SQL> create table emp
  2    (
  3    empno    number(4) not null,
  4    ename    varchar2(10),
  5    job      varchar2(9),
  6    mgr      number(4),
  7    hiredate date,
  8    sal      number(7,2),
  9    comm     number(7,2),
 10    deptno   number(2)
 11    )
 12 partition by range (deptno)
 13   (
 14   partition dept_low values less than (15)
 15     tablespace tbs_1,
 16   partition dept_mid values less than (25)
 17     tablespace tbs_2,
 18   partition dept_high values less than (35)
 19     tablespace tbs_3
 20   );
Statement processed.
SQL> exit
> imp scott tables=emp file=empexp.dat ignore=y
.
.
.
import done in WE8DEC character set and AL16UTF16 NCHAR character set
. importing SCOTT's objects into SCOTT
. . importing partition                "EMP":"EMP_LOW"          4 rows imported
. . importing partition               "EMP":"EMP_HIGH"         10 rows imported
Import terminated successfully without warnings.
```

次に示すSQLのSELECT文では、データはdeptno列でパーティション化されます。

```
SQL> connect scott
Connected.
SQL> select empno, deptno from emp partition (dept_low);
EMPNO      DEPTNO
---------- ----------
      7782         10
      7839         10
      7934         10
3 rows selected.
SQL> select empno, deptno from emp partition (dept_mid);
EMPNO      DEPTNO
---------- ----------
      7369         20
      7566         20
```

```
      7788         20
      7876         20
      7902         20
5 rows selected.
SQL> select empno, deptno from emp partition (dept_high);
EMPNO      DEPTNO
---------- ----------
      7499         30
      7521         30
      7654         30
      7698         30
      7844         30
      7900         30
6 rows selected.
SQL> exit;
```

親トピック: パーティション・レベル・インポートでのインポート・セッションの例

27.8.5 パターン一致を使用して様々な表をインポートする例

この例では、パターン一致を使用して、ユーザーscottの様々な表をインポートします。

パラメータ・ファイル方式

```
imp PARFILE=params.dat
```

params.datファイルには、次の情報が格納されています。

```
FILE=scott.dmp
IGNORE=n
GRANTS=y
ROWS=y
FROMUSER=scott
TABLES=(%d%,b%s)
```

コマンドライン方式

```
imp FROMUSER=scott FILE=scott.dmp TABLES=(%d%,b%s)
```

インポート・メッセージ

使用しているインポート・ユーティリティのリリース情報および接続しているOracle Databaseのリリース情報が表示されます。その後で、次に示すような状態メッセージが表示されます。

```
.
.
.
import done in US7ASCII character set and AL16UTF16 NCHAR character set
import server uses JA16SJIS character set (possible charset conversion)
. importing SCOTT's objects into SCOTT
. . importing table                  "BONUS"          0 rows imported
. . importing table                   "DEPT"          4 rows imported
. . importing table               "SALGRADE"          5 rows imported
Import terminated successfully without warnings.
```

親トピック: インポート・セッションの例

27.9 終了コードによる結果の検査と表示

インポート・ユーティリティでは、操作の完了後、すぐに実行結果を確認できます。プラットフォームによっては、実行結果はプロセス終了コードに通知され、ログ・ファイルにも記録されます。

これによって、コマンドラインやスクリプトからの出力を確認できます。表27-4に、様々な結果の終了コードを示します。

表27-4 インポート・ユーティリティの終了コード

結果	終了コード
インポートは警告なしで正常終了しました。	EX_SUCC
インポートは正常に終了しましたが、警告が発生しました。	EX_OKWARN
エラーが発生したためインポートを終了します。	EX_FAIL

UNIXの場合、終了コードは次のようになります。

```
EX_SUCC   0
EX_OKWARN 0
EX_FAIL   1
```

親トピック: オリジナルのインポート

27.10 インポート中のエラー処理

これらの項では、データベース・オブジェクトのインポート時に発生する可能性のあるエラーについて説明します。

- 行エラー
  整合性制約違反またはデータが無効なために行のインポートが拒否されると、警告メッセージが表示されますが、その表の残りの行は引き続き処理されます。
- データベース・オブジェクトのインポートでのエラー
  データベース・オブジェクトをインポートするときにエラーが発生する理由にはいろいろありますが、これらの項ではその理由について説明します。

親トピック: オリジナルのインポート

27.10.1 行エラー

整合性制約違反またはデータが無効なために行のインポートが拒否されると、警告メッセージが表示されますが、その表の残りの行は引き続き処理されます。

「tablespace full」というエラーなど、後続のすべての行に影響するエラーもあります。このようなエラーの場合には、現行の表の処理は停止され、次の表にスキップします。

RESUMABLE=yパラメータが指定されている場合、「tablespace full」エラーによって、インポートが一時停止することもあります。

- 整合性制約違反
  システムで有効な整合性制約に違反している行があると行エラーが発生します。
- 無効なデータ
  データベース内の表の列定義が、エクスポート・ファイル内の列定義と異なるときに、行エラーが発生する場合もあります。

親トピック: インポート中のエラー処理

27.10.1.1 整合性制約違反

システムで有効な整合性制約のいずれかに行が違反している場合、行エラーが生成されます。

たとえば、次のようなものがあります。

- NOT NULL制約
- 一意制約
- 主キー(NOT NULLおよび一意)制約
- 参照整合性制約
- CHECK制約

関連項目:

- アプリケーションで整合性制約を使用する方法の詳細は、『Oracle Database開発ガイド』を参照してください。
- 整合性制約の詳細は、『Oracle Database概要』を参照してください。

親トピック: 行エラー

27.10.1.2 無効なデータ

データベース内の表の列定義が、エクスポート・ファイル内の列定義と異なるときにも行エラーが発生します。

無効データ・エラーは、新しい表の列より長いデータの挿入、無効なデータ型またはその他のINSERTエラーによって発生します。

親トピック: 行エラー

27.10.2 データベース・オブジェクトのインポートでのエラー

データベース・オブジェクトをインポートするときにエラーが発生する理由にはいろいろありますが、これらの項ではその理由について説明します。

これらのエラーが発生すると、現行のデータベース・オブジェクトのインポートは中断されます。その後、インポート・ユーティリティでは、エクスポート・ファイルの次のデータベース・オブジェクトが継続して処理されます。

- 既存オブジェクト
  インポートするオブジェクトがデータベース中にすでに存在していると、オブジェクト作成エラーが発生します。
- 順序
  インポート処理で、順序番号をエクスポート・ファイルの値に設定しなおす必要がある場合は、順序を削除してください。
- リソース・エラー
  リソースの制限によって、オブジェクトがインポートされないことがあります。たとえば、表のインポート中に、内部的な問題またはメモリーなどのリソース不足によって、リソース・エラーが発生する場合があります。
- ドメイン索引メタデータ
  ドメイン索引は、無名PL/SQLブロックでインポートされる、アプリケーション固有のメタデータと関連付けることができます。

親トピック: インポート中のエラー処理

27.10.2.1 オブジェクトはすでに存在します

インポートするオブジェクトがデータベース中にすでに存在していると、オブジェクト作成エラーが発生します。

これ以降の処理は、IGNOREパラメータに指定されている値によって異なります。

IGNORE=n(デフォルト)が指定されている場合、エラーが報告され、次のデータベース・オブジェクトが継続して処理されます。現行のデータベース・オブジェクトは置き換えられません。オブジェクトが表の場合、エクスポート・ファイル内の行はインポートされません。

IGNORE=yが指定されている場合、オブジェクト作成エラーは報告されません。データベース・オブジェクトは置き換えられません。オブジェクトが表の場合、行がインポートされます。無視できるエラーはオブジェクト作成エラーのみです。他のすべてのエラー(オペレーティング・システムのエラー、データベースのエラー、SQLのエラーなど)は報告されます。また、処理が停止することもあります。

ノート:

IGNORE=y を指定した場合、表の 1 つ以上の列に対して UNIQUE 整合性制約を指定しないかぎり、その表に対して重複した行が挿入されます。たとえば、誤って 2 回インポートを実行した場合などがこれに該当します。

親トピック: データベース・オブジェクトのインポートでのエラー

27.10.2.2 順序

インポート処理で、順序番号をエクスポート・ファイルの値に設定しなおす必要がある場合は、順序を削除してください。

インポートでは、既存の順序の削除と再作成は行われません。そのため、順序は、インポートの前に削除されない場合、エクスポート・ファイルに保存されている値には設定されません。順序がすでに存在している場合、エクスポート・ファイルのCREATE SEQUENCE文は失敗し、その順序はインポートされません。

親トピック: データベース・オブジェクトのインポートでのエラー

27.10.2.3 リソース・エラー

リソースの制限によって、オブジェクトがインポートされないことがあります。たとえば、表のインポート中に、内部的な問題またはメモリーなどのリソース不足によって、リソース・エラーが発生する場合があります。

行のインポート中にリソース・エラーが発生すると、現行の表の処理が中止され、次の表にスキップします。COMMIT=yを指定している場合、現行の表のインポート済の部分がコミットされます。指定していない場合は、現行の表の処理がロールバックされた後で、インポートが続行されます。「COMMIT」を参照してください。

親トピック: データベース・オブジェクトのインポートでのエラー

27.10.2.4 ドメイン索引のメタデータ

ドメイン索引は、無名PL/SQLブロックでインポートされる、アプリケーション固有のメタデータと関連付けることができます。

これらのPL/SQLブロックは、インポート時にCREATE INDEX文より優先して実行されます。PL/SQLブロックにエラーが発生した場合、メタデータが索引の一部分とみなされるため、関連付けられた索引は作成されません。

親トピック: データベース・オブジェクトのインポートでのエラー

27.11 表レベル・インポートおよびパーティション・レベル・インポート

表、パーティションおよびサブパーティションをインポートできます。

具体的には次のとおりです。

- 表レベル・インポート: エクスポート・ファイルのすべてのデータをインポートします。
- パーティション・レベル・インポート: 指定されたソース・パーティションまたはサブパーティションのデータのみインポートします。
- 表レベル・インポートの使用に関するガイドライン
  表レベル・インポートでは、指定された各表に対して、表のすべての行がインポートされます。
- パーティション・レベル・インポートの使用に関するガイドライン
  パーティション・レベル・インポートは、表モードでのみ指定できます。パーティション・レベル・インポートでは、エクスポート・ファイル内の特定のパーティションまたはサブパーティションを選択してデータをロードすることができます。
- パーティションと表の間のデータ移行
  コンポジット・パーティションのパーティション名を指定しない場合、コンポジット・パーティション内のすべてのサブパーティションが、ソースとして使用されます。

親トピック: オリジナルのインポート

27.11.1 表レベル・インポートの使用に関するガイドライン

表レベル・インポートでは、指定した各表に関して表のすべての行がインポートされます。

表レベル・インポートの特長は次のとおりです。

- (TRANSPORT_TABLESPACESを除く)どのエクスポート・モードでエクスポートされた場合でも、表レベル・インポートを使用してエクスポートされたすべての表をインポートできます。
- ユーザーは、表レベルでエクスポートされたすべての表(パーティション表または非パーティション表)、パーティションまたはサブパーティションを、同じ名前のターゲット表(パーティション表または非パーティション表)にインポートできます。

表が存在しない場合で、なおかつエクスポートされた表がパーティション表である場合は、表レベル・インポートによってパーティション表が作成されます。表が正常に作成されると、エクスポート・ファイルからすべてのソース・データが読み込まれ、ターゲット表に書き込まれます。インポート後、ターゲット表には、エクスポート・ファイル内のソース表に対応付けられたすべてのパーティションおよびサブパーティションに関するパーティション定義が格納されます。この処理によって、ソース・パーティションの物理属性および論理属性(パーティションの境界を含む)が、インポート時に維持されます。

親トピック: 表レベル・インポートおよびパーティション・レベル・インポート

27.11.2 パーティション・レベル・インポートの使用に関するガイドライン

パーティション・レベル・インポートを指定できるのは、表モードでインポートを実行する場合のみです。パーティション・レベル・インポートでは、エクスポート・ファイル内の特定のパーティションまたはサブパーティションを選択してデータをロードすることができます。

パーティション・レベル・インポートを使用する場合は、次のガイドラインに注意してください。

- インポートでは、常にターゲット表のパーティション化スキームに従って行が格納されます。
- パーティション・レベル・インポートでは、指定されたソース・パーティションまたはサブパーティションの行データのみ挿入されます。

- ターゲット表がパーティション表の場合、パーティション・レベル・インポートを行うと、そのターゲット表の最上位パーティションより上に入る行はすべて拒否されます。
- エクスポートされた表が非パーティション表の場合は、パーティション・レベル・インポートを実行できません。ただし、表レベル・インポートを使用すると、エクスポートされた非パーティション表からパーティション表をインポートできます。
- ソース表(エクスポート時にtablenameに指定された表)がパーティション表で、エクスポート・ファイルに存在する場合のみ、パーティション・レベル・インポートは正常に実行されます。
- エクスポート・ファイルに存在しないパーティション名またはサブパーティション名を指定すると、警告が発行されます。
- パラメータの中で指定するパーティション名またはサブパーティション名には、エクスポート・ファイルにあるパーティションまたはサブパーティションのみを指定します。エクスポート・ファイルには、エクスポート・ソース・システム上の表のすべてのデータが含まれているとはかぎりません。
- ROWS=y(デフォルト)が指定されていて、表がインポート先のシステムに存在しない場合、表が作成され、すべての行が、ソース・パーティションまたはサブパーティションから、インポート先の表のパーティションまたはサブパーティションに挿入されます。
- ROWS=y(デフォルト)およびIGNORE=yが指定されていて、対象となる表の表名がインポート前に存在している場合は、指定された表のパーティションまたはサブパーティションの行がすべて、同名の表に挿入されます。インポートでは、常に、ターゲット表の既存のパーティション化スキーマに従って行が格納されます。
- ROWS=nが指定されている場合、データはターゲット表に挿入されず、エクスポート・ファイル中の表およびパーティションまたはサブパーティションに関連する他のオブジェクトに対する処理が、継続して行われます。
- ターゲット表が非パーティション表の場合、パーティションおよびサブパーティションは表全体にインポートされます。1つ以上のパーティションまたはサブパーティションを、エクスポート・ファイルからインポート・ターゲット・システム上の非パーティション表にインポートするには、IGNORE=yを指定します。

親トピック: 表レベル・インポートおよびパーティション・レベル・インポート

27.11.3 パーティションと表の間のデータ移行

コンポジット・パーティションのパーティション名を指定しない場合、コンポジット・パーティション内のすべてのサブパーティションが、ソースとして使用されます。

次の例では、パーティション名によって指定されたパーティションは、コンポジット・パーティションです。すべてのサブパーティションがインポートされます。

```
imp SYSTEM FILE=expdat.dmp FROMUSER=scott TABLES=b:py
```

次の例では、表scott.eのパーティションqcおよびqdの行データが、表scott.eにインポートされます。

```
imp scott FILE=expdat.dmp TABLES=(e:qc, e:qd) IGNORE=y
```

インポート・ターゲット・データベースに表eが存在しない場合は、表eの作成後、同じパーティションにデータが挿入されます。インポート前にターゲット・システムに表eが存在する場合、行データは、挿入可能な範囲を持つパーティションに挿入されます。行データは、最終的にqcおよびqd以外の名前のパーティションに挿入することもできます。

ノート:

既存の表にパーティション・レベル・インポートを実行する場合は、ターゲット・パーティションまたはサブパーティション

を正しく設定し、IGNORE=y を指定してください。

親トピック: 表レベル・インポートおよびパーティション・レベル・インポート

27.12 索引作成およびメンテナンスの制御

これらの項では、索引作成およびメンテナンスに関連するインポートの動作について説明します。

- 索引作成の延期
  インポート・ユーティリティには、索引の作成およびメンテナンスの実行を、インポートが完了し、エクスポート・データの挿入が終了するまで延期させる機能が用意されています。
- 索引作成およびメンテナンスの制御
  索引作成およびメンテナンスの制御について説明します。

親トピック: オリジナルのインポート

27.12.1 索引作成の延期

インポート・ユーティリティには、索引の作成およびメンテナンスの実行を、インポートが完了し、エクスポート・データの挿入が終了するまで延期させる機能が用意されています。

インポート完了後に索引の作成、再作成またはメンテナンスを実行すると、通常、その処理時間は、インポートで各行が挿入されるたびにメンテナンスを実行するより短くなります。

索引作成には時間がかかるため、他のすべてのオブジェクトのインポートが完了してから行った方が効率的です。INDEXES=nを指定すると、インポートの終了後に索引を作成できます。(デフォルトはINDEXES=yです。)その後、INDEXFILEパラメータを使用してインポートを実行し、SQLスクリプト内の未作成の索引定義を格納できます。索引作成文は、このように指定しない場合、インポート・ユーティリティから発行されますが、このように指定した場合、指定されたファイルに書き込まれます。

インポート完了後、索引を作成する必要があります。索引を作成するには、通常、CONNECT文にパスワードを指定した後、INDEXFILEで指定したファイルの内容をSQLスクリプトとして使用します。

親トピック: 索引作成およびメンテナンスの制御

27.12.2 索引作成およびメンテナンスの制御

索引作成およびメンテナンスの制御について説明します。

SKIP_UNUSABLE_INDEXES=yを指定すると、インポート前に索引使用禁止に設定されていた索引のメンテナンスはすべて延期されます。他の索引(事前に索引使用禁止に設定されていない索引)に対しては、行の挿入時にメンテナンス処理が行われます。これにより、既存の表をインポートする間、索引の更新が保存されます。

索引のメンテナンスが延期されると、その索引で設定されている既存の一意整合性制約に対して違反が発生することがあります。表に一意整合性制約が存在しても、INDEXES=nを指定してインポートした表内の重複キーは回避できません。このため、その索引は、重複キーが削除されて索引が再構築されるまでは、UNUSABLE状態となります。

- 索引更新延期の例
  索引更新延期の例を示します。

親トピック: 索引作成およびメンテナンスの制御

27.12.2.1 索引更新延期の例

索引更新延期の例を示します。

パーティションp1およびp2を持つパーティション表tが、インポート・ターゲット・システムに存在するとします。また、パーティションp1にローカル索引p1_ind、パーティションp2にローカル索引p2_indが存在するとします。このパーティションp1には既存の表tのデータが入っており、そのデータ量は、エクスポート・ファイル(expdat.dmp)を使用して挿入されるデータの量よりはるかに多いとします。一方、パーティションp2はその逆であるとします。

表データ挿入時にp1_indの索引メンテナンスを実行すると、パーティション索引の再作成時に実行するより、処理効率が高くなります。p2_indについては、この逆になります。

また、p2_indについては、インポート中のローカル索引のメンテナンスを延期できます。延期するには、次のステップを実行します。

1. インポート前に、次のSQL文を発行します。

```
ALTER TABLE t MODIFY PARTITION p2 UNUSABLE LOCAL INDEXES;
```

2. 次のインポート・コマンドを発行します。

```
imp scott FILE=expdat.dmp TABLES = (t:p1, t:p2) IGNORE=y
SKIP_UNUSABLE_INDEXES=y
```

   この例では、インポートの実行前にALTER SESSION SET SKIP_UNUSABLE_INDEXES=y文を実行します。

3. インポート後に次のSQL文を発行します。

```
ALTER TABLE t MODIFY PARTITION p2 REBUILD UNUSABLE LOCAL INDEXES;
```

この例では、p1のローカル索引p1_indは、インポート中、表データがパーティションp1に挿入されるときにメンテナンスされます。一方、p2のローカル索引p2_indは、インポート後の索引再作成時にメンテナンスされます。

親トピック: 索引作成およびメンテナンスの制御

27.13 オリジナルのインポート・ユーティリティでOracle Netを使用する際のネットワーク考慮事項

ネットワーク経由でインポートを実行する場合は、Oracle NetでOracle Data Pumpのオリジナルのインポート・ユーティリティ(imp)を使用できます。

たとえば、インポート・ユーティリティをローカルで実行しているときに、リモートのOracle Databaseインスタンスにデータを読み込むことができます。

Oracle Netでインポート・ユーティリティを使用するには、impコマンドを実行してユーザー名とパスワードを入力するときに、接続修飾文字列@connect_stringを含めます。この句の構文の詳細は、ご使用のOracle Netプロトコルのユーザーズ・ガイドを参照してください。

関連トピック

- Oracle Database Net Services管理者ガイド

親トピック: オリジナルのインポート

27.14 文字セットおよびグローバリゼーション・サポートに関する考慮点

これらの項では、ユーザー・データおよびデータ定義言語(DDL)の文字セット変換に関連するインポート・ユーティリティのグローバリゼーション・サポートの動作について説明します。

- ユーザー・データ
  エクスポート・ユーティリティは、常に、エクスポート・サーバーの文字セットでUnicodeデータを含むユーザー・データをエクスポートします。(文字セットは、データベース作成時に指定されます。)
- データ定義言語(DDL)
  エクスポートおよびインポート操作時に、データ定義言語(DDL)に対して最大3回の文字セット変換が必要です。
- シングルバイト文字セット
  8ビット文字セットのエクスポート・ファイルをインポートすると、一部の8ビット・キャラクタが失われる(つまり7ビットの対応するキャラクタに変換される)場合があります。
- マルチバイト文字セット
  ターゲット文字セットに同等の文字がないエクスポート・ファイル中の文字は、変換時にデフォルトの文字に置換されます(デフォルトの文字は、ターゲット文字セットによって定義されます)。(デフォルトの文字は、ターゲット文字セットによって定義されます)。

親トピック: オリジナルのインポート

27.14.1 ユーザー・データ

エクスポート・ユーティリティは、常に、エクスポート・サーバーの文字セットでUnicodeデータを含むユーザー・データをエクスポートします。(文字セットは、データベース作成時に指定されます。)

ソース・データベースの文字セットが、インポート・データベースの文字セットと異なる場合、自動的にデータをインポート・サーバーの文字セットに変換するための文字セット変換が実行されます。

- 変換による文字セットのソート順への影響
  エクスポート文字セットのソート順が、インポート文字セットと異なる場合、キャラクタ列をパーティション化した表では、結果が保証されません。

親トピック: 文字セットおよびグローバリゼーション・サポートに関する考慮点

27.14.1.1 変換による文字セットのソート順への影響

エクスポート文字セットのソート順が、インポート文字セットと異なる場合、キャラクタ列をパーティション化した表では、結果が保証されません。

たとえば、次のようなASCII文字セットのデータベースの表定義について考えてみます。

```
CREATE TABLE partlist
   (
   part     VARCHAR2(10),
   partno   NUMBER(2)
   )
PARTITION BY RANGE (part)
  (
  PARTITION part_low VALUES LESS THAN ('Z')
    TABLESPACE tbs_1,
  PARTITION part_mid VALUES LESS THAN ('z')
    TABLESPACE tbs_2,
  PARTITION part_high VALUES LESS THAN (MAXVALUE)
    TABLESPACE tbs_3
  );
```

ASCII文字セットでは、Zの後にzがあるため、このパーティション・スキームには意味があります。

この表がEBCDIC文字セット・ベースのデータベースにインポートされると、EBCDIC文字セットでは、zがZの前にあるため、part_midパーティションのすべての行が、part_lowパーティションに移行します。希望する結果を得るには、partlist表の所有者は、インポート後に表を再パーティション化する必要があります。

関連項目:

文字セットの詳細は、『Oracle Databaseグローバリゼーション・サポート・ガイド』を参照してください。

親トピック: ユーザー・データ

27.14.2 DDL

エクスポートおよびインポート操作時に、データ定義言語(DDL)に対して最大3回の文字セット変換が必要です。

具体的には次のとおりです。

1. エクスポート・ファイルは、環境変数NLS_LANGでユーザー・セッション用に指定された文字セットで書き出されます。NLS_LANGの値が、データベースの文字セットと異なる場合は、文字セット変換が実行されます。
2. エクスポート・ファイルの文字セットが、インポート先ユーザー・セッション用の文字セットと異なる場合、ユーザー・セッションの文字セットに変換されます。シングルバイト文字セットの場合のみにこの変換が実行されます。マルチバイト文字セットの場合は、インポート・ファイルの文字セットがエクスポート・ファイルの文字セットと同じである必要があります。
3. ターゲット・データベースの文字セットが、インポート・ユーザー・セッション用の文字セットと異なる場合、最後の文字セット変換が実行される場合があります。

文字セット変換によるデータの損失を最小限にするには、エクスポート・データベース、エクスポート・ユーザー・セッション、インポート・ユーザー・セッションおよびインポート・データベースのすべてにおいて、同一の文字セットを使用するようにしてください。

親トピック: 文字セットおよびグローバリゼーション・サポートに関する考慮点

27.14.3 シングルバイト文字セット

8ビット文字セットのエクスポート・ファイルをインポートすると、一部の8ビット・キャラクタが失われる(つまり7ビットの対応するキャラクタに変換される)場合があります。

これが発生するのは、インポートを実行するシステムに、システム固有の7ビット文字セットが存在するか、オペレーティング・システム環境変数NLS_LANGが7ビット文字セットに設定されている場合です。アクセント記号が付いている文字からアクセントが消去されるのが最もよく見られる例です。

このような状況を回避するために、オペレーティング・システム環境変数NLS_LANGにエクスポート・ファイルの文字セットを設定できます。

親トピック: 文字セットおよびグローバリゼーション・サポートに関する考慮点

27.14.4 マルチバイト文字セット

ターゲット文字セットに同等の文字がないエクスポート・ファイル中の文字は、変換時にデフォルトの文字に置換されます(デフォルトの文字は、ターゲット文字セットによって定義されます)。(デフォルトの文字は、ターゲット文字セットによって定義されます)。

100%完全に変換されるためには、ターゲット文字セットはソース文字セットのスーパーセットであるか、ソース文字セットと同等

である必要があります。

ノート:

エクスポート・サーバーとインポート・サーバーとの間で文字セットの幅が異なるとき、変換が原因でデータの拡張が発生する場合にデータの切捨てが発生することがあります。切捨てが発生する場合、インポート・ユーティリティに警告メッセージが表示されます。

親トピック: 文字セットおよびグローバリゼーション・サポートに関する考慮点

27.15 インスタンス親和性の使用方法

インスタンス親和性を使用して、インポートおよびエクスポートするデータベース内のインスタンスにジョブを関連付けることができます。

複数のリリースを組み合せて使用している場合は、互換性の問題に注意してください。

関連項目:

- 親和性の詳細は、『Oracle Database管理者ガイド』を参照してください。

親トピック: オリジナルのインポート

27.16 データベース・オブジェクトのインポートに関する考慮点

これらの項では、特定のデータベース・オブジェクトをインポートする場合の、制限事項および考慮点について説明します。

- オブジェクト識別子のインポート
- 既存のオブジェクト表およびオブジェクト型の含まれている表のインポート
  既存のオブジェクト表およびオブジェクト型が含まれている表のインポートは、データベース・オブジェクトをインポートする場合の考慮点の1つです。表を作成する場合、以前(記憶域パラメータ以外に対して)使用していた定義と同じ定義で作成するか、互換性のある形式で作成する必要があります。
- ネストした表のインポート
- REFデータのインポート
  REFデータのインポートは、データベース・オブジェクトをインポートする場合の考慮点の1つです。REF列および属性には、参照されている型のインスタンスを示すROWIDが隠されていることがあります。
- BFILE列およびディレクトリ別名のインポート
  BFILE列およびディレクトリ別名のインポートは、データベース・オブジェクトをインポートする場合の考慮点の1つです。BFILE列を含む表データをインポートする場合、BFILEロケータは、ディレクトリ別名およびエクスポート時のファイル名でインポートされます。
- 外部関数ライブラリのインポート
  外部関数ライブラリのインポートは、データベース・オブジェクトをインポートする場合の考慮点の1つです。インポート・ユーティリティでは、外部関数ライブラリの参照先が正しいかどうかの確認は行われません。
- ストアド・プロシージャ、ファンクションおよびパッケージのインポート
  ローカルのストアド・プロシージャ、ストアド・ファンクションまたはパッケージがインポートされるときのインポート・ユーティリティの動作は、COMPILEパラメータがyとnのどちらに設定されているかによって異なります。

- Javaオブジェクトのインポート
  Javaオブジェクトのインポートは、データベース・オブジェクトをインポートする場合の考慮点の1つです。Javaオブジェクトを任意のスキーマにインポートしても、リゾルバはインポート・ユーティリティによって変更されません(リゾルバとは、Javaのフルネームの解決に使用されるスキーマのリストです)。
- 外部表のインポート
  外部表のインポートは、データベース・オブジェクトをインポートする場合の考慮点の1つです。インポート・ユーティリティでは、外部表によって参照される場所が正しいかどうかを検証しません。
- AQ表のインポート
  アドバンスト・キュー表のインポートは、データベース・オブジェクトをインポートする場合の考慮点の1つです。キュー表をインポートすると、基礎となっているキューや関連するディクショナリ情報もインポートされます。
- LONG列のインポート
  LONG列のインポートは、データベース・オブジェクトをインポートする場合の考慮点の1つです。インポートおよびエクスポート時には、LONG列は各行の残りのデータとともにメモリーに収まるサイズである必要があります。
- トリガーが存在する場合のLOB列のインポート
  トリガーが存在する場合のLOB列のインポートは、データベース・オブジェクトをインポートする場合の考慮点の1つです。インポート・ユーティリティでは、エクスポート時に空であるすべてのLOBがインポート後に自動的にNULLに変更されます。
- ビューのインポート
  別のスキーマの表を参照しているビューをインポートするには、インポートの実行者にREAD ANY TABLE権限またはSELECT ANY TABLE権限が付与されている必要があります。
- パーティション表のインポート
  パーティション表のインポートは、データベース・オブジェクトをインポートする場合の考慮点の1つです。エクスポートしたパーティション表と同じパーティション名またはサブパーティション名を使用してパーティション表を作成するために、SYS_Pnnn形式の名前もインポートされます

親トピック: オリジナルのインポート

27.16.1 オブジェクト識別子のインポート

Oracle Databaseでは、オブジェクト型、オブジェクト表およびオブジェクト表内の行を一意に識別できるように、オブジェクト識別子が割り当てられます。オブジェクト識別子はインポート・ユーティリティによって保持されます。

型を参照している表のインポート時に、その名前の型がすでにデータベースに存在している場合は、その既存の型が、実際にその表で使用されているかどうか(実際は異なる型で、単に同じ名前であるだけではないか)を確認します。

この確認のために、型の一意の識別子(TOID)とエクスポート・ファイルに格納された識別子が比較されます。これらの識別子が一致する場合は、型の一意のハッシュ・コードとエクスポート・ファイルに格納されたハッシュ・コードが比較されます。TOIDまたはハッシュ・コードが一致しない場合、その表の行はインポートされません。

この妥当性チェックをしてはいけない型もあります(たとえば、その型がカートリッジのインストールによって作成された場合)。パラメータTOID_NOVALIDATEを使用して、TOIDおよびハッシュ・コードと比較しない型を指定できます。詳細は、「TOID_NOVALIDATE」を参照してください。

ノート:

型比較は、不正なデータを発生させないための非常に重要な機能であるため、TOID_NOVALIDATE の使用には、特に注意してください。この機能を使用禁止にする場合は、データ型の妥当性チェックとその処理について十分

な知識を持つユーザーが行ってください。

次の基準によって、オブジェクト型、オブジェクト表およびオブジェクト表の行の処理方法が決まります。

- オブジェクト型に関して、IGNORE=yが指定されていて、オブジェクト型がすでに存在し、そのオブジェクト識別子、ハッシュ・コードおよび型記述子が一致する場合は、エラーは通知されません。オブジェクト識別子またはハッシュ・コードが一致しない場合、パラメータTOID_NOVALIDATEにそのオブジェクト型を無視する設定がされていないと、エラーが通知され、そのオブジェクト型を使用している表はインポートされません。
- オブジェクト型に関して、IGNORE=nが指定されていて、そのオブジェクト型がすでに存在する場合は、エラーが通知されます。オブジェクト識別子、ハッシュ・コードまたは型記述子が一致しない場合、パラメータTOID_NOVALIDATEにそのオブジェクト型を無視するように設定されていないと、エラーが通知され、そのオブジェクト型を使用している表はインポートされません。
- オブジェクト表に関して、IGNORE=yが指定されていて、表がすでに存在し、そのオブジェクト識別子、ハッシュ・コードおよび型記述子が一致する場合、エラーは通知されません。行はオブジェクト表にインポートされます。同じオブジェクト識別子の行が、すでにそのオブジェクト表に存在している場合、行のインポートはエラーになります。オブジェクト識別子、ハッシュ・コードまたは型記述子が一致しない場合、パラメータTOID_NOVALIDATEがそのオブジェクト型を無視するように設定されていないと、エラーが通知され、そのオブジェクト型を使用している表はインポートされません。
- オブジェクト表に関して、IGNORE=nが指定されていて、オブジェクト表がすでに存在する場合は、エラーが通知され、オブジェクト表はインポートされません。

インポート・ユーティリティにオブジェクト型とオブジェクト表に関するオブジェクト識別子が保持されるため、FROMUSERパラメータおよびTOUSERパラメータを使用して、あるユーザー・スキーマから別のユーザー・スキーマにオブジェクトをインポートする場合は、次のことを考慮してください。

- FROMUSERのオブジェクト型およびオブジェクト表がターゲット表にすでに存在する場合は、TOUSERのオブジェクト型およびオブジェクト表の識別子がすでに使用されているため、エラーが発生します。インポート開始前に、FROMUSERのオブジェクト型およびオブジェクト表をシステムから削除する必要があります。
- オブジェクト表作成時に、OID ASオプションを指定して他の表と同じオブジェクト識別子が割り当てられている場合は、同じオブジェクト識別子を持つ表を両方インポートすることはできません。1つ目の表はインポートできますが、同じオブジェクト識別子がすでに使用されているため、2つ目の表をインポートするとエラーになります。

親トピック: データベース・オブジェクトのインポートに関する考慮点

27.16.2 既存のオブジェクト表およびオブジェクト型の含まれている表のインポート

既存のオブジェクト表およびオブジェクト型が含まれている表のインポートは、データベース・オブジェクトをインポートする場合の考慮点の1つです。表を作成する場合、以前(記憶域パラメータ以外に対して)使用していた定義と同じ定義で作成するか、互換性のある形式で作成する必要があります。

表領域の使用方法または表の記憶域パラメータを変更するため、インポートの前に表を作成することがよくあります。表を作成する場合、以前(記憶域パラメータ以外に対して)使用していた定義と同じ定義で作成するか、互換性のある形式で作成する必要があります。オブジェクト表や、オブジェクト型の列を含む表の場合は、形式の互換性がさらに制限されます。

オブジェクト表およびオブジェクト列を含む表の場合、表が参照する各オブジェクトは、その名前、構造およびバージョン情報をエクスポート・ファイルに書き出されます。エクスポート・ユーティリティでは、必要に応じて、異なるスキーマのオブジェクト型の情報も含まれます。

インポート・ユーティリティは、表データをインポートする前に、表に必要な各オブジェクト型の存在を確認します。この確認機能には、オブジェクト型の名前の確認、インポート・システムのオブジェクト型の構造およびバージョンと、エクスポート・ファイルに書き込まれた情報との比較が含まれます。

オブジェクト型名がインポート・システムで検出され、構造またはバージョンがエクスポート・ファイルと一致しない場合、エラー・メッセージが生成され、表データはインポートされません。

インポート・パラメータTOID_NOVALIDATEを使用して、特定のオブジェクトのオブジェクト型の構造およびバージョンの確認機能を使用禁止にできます。

親トピック: データベース・オブジェクトのインポートに関する考慮点

27.16.3 ネストした表のインポート

内部のネストした表は外部表とは別にエクスポートされます。したがって、内部のネストした表が正しくインポートされない場合、次のような状況が予想されます。

- 内部にネストした表を持つ表がエクスポートされ、インポート時に、表も表内の行も削除されなかったとします。IGNORE=yパラメータが指定されている場合、外部表に各行を挿入すると、制約違反が発生します。ただし、内部のネストした表のデータは正常にインポートされることがあり、その場合、内部表の行データが重複します。
- 外部表へデータをインポート中にリカバリ不能なエラーが発生した場合は、外部表の残りのデータはスキップされますが、対応する内部表の行はスキップされません。その結果、内部表の行は外部表のどの行からも参照されなくなります。
- リカバリ可能なエラーの後で内部表への挿入がエラーになる場合、外部表の行はすでに外部表にインポートされています。また、外部表および他の内部表へのデータのインポートは続行されます。その結果、不完全な論理行が作成されます。
- 内部表へのデータのインポート中にリカバリ不能なエラーが発生した場合は、その内部表の残りのデータはスキップされますが、外部表やその他のネストした表はスキップされません。

常にログ・ファイルを調べて、外部表および内部表にエラーがないかどうかを確認する必要があります。データに一貫性を持たせるためには、表データの変更や削除が必要になることがあります。

内部にネストした表は、外部表とは別にインポートされるので、インポート中に、このネストした表のデータにアクセスしようとしても失敗することがあります。たとえば、内部表の行がインポートされる前に、外部表の行にアクセスすると、ユーザーには不完全な行が返されます。

親トピック: データベース・オブジェクトのインポートに関する考慮点

27.16.4 REFデータのインポート

REFデータのインポートは、データベース・オブジェクトをインポートする場合の考慮点の1つです。REF列および属性には、参照されている型のインスタンスを示すROWIDが隠されていることがあります。

REF列および属性には、参照されている型のインスタンスを示すROWIDが隠されていることがあります。インポート・ユーティリティでは、ターゲット・データベースに対するROWIDは、自動的には再設定されません。ROWIDを適切な値に再設定するには、次の文を実行します。

```
ANALYZE TABLE [schema.]table VALIDATE REF UPDATE;
```

関連項目:

ANALYZE文の詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

親トピック: データベース・オブジェクトのインポートに関する考慮点

27.16.5 BFILE列およびディレクトリ別名のインポート

BFILE列およびディレクトリ別名のインポートは、データベース・オブジェクトをインポートする場合の考慮点の1つです。BFILE列を含む表データをインポートする場合、BFILEロケータは、ディレクトリ別名およびエクスポート時のファイル名でインポートされます。

BFILE列および属性で参照されているデータは、ソース・データベースからターゲット・データベースへはコピーされません。BFILE列で参照されているファイルの名前とディレクトリ別名が伝達されるのみです。BFILE列および属性で参照されている実際のファイルは、DBAまたはユーザーが移動してください。

BFILE列を含む表データをインポートする場合、BFILEロケータは、ディレクトリ別名およびエクスポート時のファイル名でインポートされます。インポート・ユーティリティでは、そのディレクトリ別名またはファイルが存在するかどうかの確認は行われません。ディレクトリ別名またはファイルが存在しない場合、ユーザーがBFILEデータにアクセスするとエラーが発生します。

ディレクトリ別名に関しては、エクスポート・システムで使用しているオペレーティング・システムのディレクトリ構文がインポート・システムで有効でない場合でも、インポート時にエラーは通知されません。ユーザーがインポート後に、そのファイルのデータにアクセスするとエラーが返されます。ディレクトリ別名がインポート・システムで有効かどうかは、DBAまたはユーザーが確認してください。

親トピック: データベース・オブジェクトのインポートに関する考慮点

27.16.6 外部関数ライブラリのインポート

外部関数ライブラリのインポートは、データベース・オブジェクトをインポートする場合の考慮点の1つです。インポート・ユーティリティでは、外部関数ライブラリの参照先が正しいかどうかの確認は行われません。

インポート・ユーティリティでは、外部関数ライブラリの参照先が正しいかどうかの確認は行われません。エクスポート・ファイル上のライブラリの指定で使用されているディレクトリやファイル名の形式がインポート・システムで無効であっても、インポート時にエラーは通知されません。この場合、インポート後にそのファンクションを呼び出そうとすると、エラーが返されます。

DBAまたはユーザーが手動でライブラリを移動し、ライブラリの指定がインポート・システムで有効になるようにしてください。

親トピック: データベース・オブジェクトのインポートに関する考慮点

27.16.7 ストアド・プロシージャ、ファンクションおよびパッケージのインポート

ローカルのストアド・プロシージャ、ストアド・ファンクションまたはパッケージがインポートされるときのインポート・ユーティリティの動作は、COMPILEパラメータがyとnのどちらに設定されているかによって異なります。

ローカルのストアド・プロシージャ、ストアド・ファンクションまたはパッケージがインポートされるときのインポート・ユーティリティの動作は、COMPILEパラメータがyまたはnに設定されているかどうかによって異なります。

ローカルのストアド・プロシージャ、ストアド・ファンクションまたはパッケージがインポートされ、COMPILE=yが指定されている場合は、プロシージャ、ファンクション、パッケージはインポート時に再コンパイルされ、元のタイムスタンプ仕様が保持されます。コンパイルが成功すると、リモート・プロシージャによってアクセスしてもエラーは発生しません。

COMPILE=nが指定されている場合も、プロシージャ、ストアド・ファンクションまたはパッケージはインポートされます。ただし、元のタイムスタンプは失われます。次に、プロシージャ、ファンクションまたはパッケージを使用するときに、コンパイルが実行されます。

関連項目:

COMPILE

親トピック: データベース・オブジェクトのインポートに関する考慮点

27.16.8 Javaオブジェクトのインポート

Javaオブジェクトのインポートは、データベース・オブジェクトをインポートする場合の考慮点の1つです。Javaオブジェクトを任意のスキーマにインポートしても、リゾルバはインポート・ユーティリティによって変更されません(リゾルバとは、Javaのフルネームの解決に使用されるスキーマのリストです)。

Javaオブジェクトを任意のスキーマにインポートしても、リゾルバはインポート・ユーティリティによって変更されません(リゾルバとは、Javaのフルネームの解決に使用されるスキーマのリストです)。(リゾルバとは、Javaのフルネームの解決に使用されるスキーマのリストです)。インポートの終了後、明示的または暗黙的に再検証しないかぎり、すべてのユーザー・クラスが無効な状態のままになります。暗黙的な再検証は、クラスが最初に参照されるときに実行されます。明示的な再検証は、SQLのALTER JAVA CLASS...RESOLVE文を使用すると実行されます。いずれの方法でもクラスは正常に解決され、有効になります。

親トピック: データベース・オブジェクトのインポートに関する考慮点

27.16.9 外部表のインポート

外部表のインポートは、データベース・オブジェクトをインポートする場合の考慮点の1つです。インポート・ユーティリティでは、外部表によって参照される場所が正しいかどうかを検証しません。

インポート・ユーティリティでは、外部表によって参照される場所が正しいかどうかを検証しません。エクスポート・ファイルの表の指定で使用されるディレクトリ名およびファイル名の書式がインポート・システムで無効な場合、エラーはインポート時に報告されません。後でコールアウト関数を使用するときにエラーを受け取ります。

DBAまたはユーザーが手動で表を移動し、表の指定がインポート・システムで有効になるようにしてください。

親トピック: データベース・オブジェクトのインポートに関する考慮点

27.16.10 AQ表のインポート

アドバンスト・キュー表のインポートは、データベース・オブジェクトをインポートする場合の考慮点の1つです。キュー表をインポートすると、基礎となっているキューや関連するディクショナリ情報もインポートされます。

キュー表をインポートすると、基礎となっているキューや関連するディクショナリ情報もインポートされます。キューのインポートは、キュー表単位のレベルでのみ実行できます。キュー表のインポートでは、エクスポートの表処理プロシージャの前後に、キュー・ディクショナリがメンテナンスされます。

関連項目:

Oracle Databaseアドバンスト・キューイング・ユーザーズ・ガイド

親トピック: データベース・オブジェクトのインポートに関する考慮点

27.16.11 LONG列のインポート

LONG列のインポートは、データベース・オブジェクトをインポートする場合の考慮点の1つです。インポートおよびエクスポート時には、LONG列は各行の残りのデータとともにメモリーに収まるサイズである必要があります。

LONG列の長さは、最大2GBです。インポートおよびエクスポート時には、LONG列は各行の残りのデータとともにメモリーに収ま

るサイズである必要があります。ただし、LONGデータはセクション単位でロードされるため、LONG列を格納するメモリーが連続している必要はありません。

インポート・ユーティリティを使用して、LONG列をCLOB列に変換できます。このためには、まず、新しいCLOB列を指定する表を作成します。インポートを実行すると、LONGデータはCLOB形式に変換されます。同じ方法で、LONG RAW列をBLOB列に変換できます。

ノート:

既存の LONG 列を LOB 列に変換することをお薦めします。LOB 列は、LONG 列ほど制限は多くありません。LOB 機能はリリースごとに拡張されていますが、LONG 機能は最近のリリースでは変更されていません。

親トピック: データベース・オブジェクトのインポートに関する考慮点

27.16.12 トリガーが存在する場合のLOB列のインポート

トリガーが存在する場合のLOB列のインポートは、データベース・オブジェクトをインポートする場合の考慮点の1つです。インポート・ユーティリティでは、エクスポート時に空であるすべてのLOBがインポート後に自動的にNULLに変更されます。

Oracle Database 10gでは、トリガーが正常に動作し、LOBのロード中に高いパフォーマンスが保たれるようにLOB処理が改善されています。こうした改善を実現するために、インポート・ユーティリティでは、エクスポート時に空であるすべてのLOBがインポート後に自動的にNULLに変更されるように変更されています。

LOBがNULLではなく、空であることが必要なアプリケーションを使用している場合は、インポート後に、各LOB列に対してSQL UPDATE文を発行できます。LOB列型がBLOBかCLOBかによって、構文は次のいずれかになります。

```
UPDATE <tablename> SET <lob column> = EMPTY_BLOB() WHERE <lob column> = IS NULL;
UPDATE <tablename> SET <lob column> = EMPTY_CLOB() WHERE <lob column> = IS NULL;
```

インポートの実行後、NULLのLOB列と空のLOB列を区別する方法はありません。そのため、この情報がデータの整合性上重要である場合は、インポートの実行前にNULLのLOB列と空のLOB列を確認しておいてください。

親トピック: データベース・オブジェクトのインポートに関する考慮点

27.16.13 ビューのインポート

他のスキーマの表を参照しているビューをインポートする場合は、インポートを実行するユーザーに、READ ANY TABLEまたはSELECT ANY TABLE権限が必要です。

ビューは、依存順序でエクスポートされます。状況によっては、データベースから順序を取得するのではなく、エクスポートで順序付けをする必要があります。この場合、常に正しい順序を複製できるとはかぎりません。順序が正しくない場合、ビューのインポート時にコンパイル上の警告が発行され、そのビューに関する列コメントはインポートされません。

特に、viewaでストアド・プロシージャprocbが使用され、procbでビューviewcが使用されている場合、エクスポート・ユーティリティでは、ビューviewaとviewcの正しい順序付けはできません。viewaがviewcより先にエクスポートされ、procbがインポート・システムにすでに存在する場合は、viewaのインポート時にコンパイル上の警告が出されます。

ビューに関する権限は、ビューにコンパイル・エラーがあってもインポートされます。ビューの作成時に、そのビューの基礎になっているオブジェクト(たとえば、表、プロシージャ、他のビューなど)が存在していない場合、ビューにコンパイル・エラーが発生する場合があります。実表が存在しない場合、実表に対する権限を付与したユーザー自身が、その実表に対してGRANTオプション付きの適正な権限を持っているかどうかを、サーバーでは検証できません。権限を付与したユーザーが適正な権限を持っていない場合、

インポートされなかった表の作成後にその表にアクセスしようとすると、エラーが発生します。

他のスキーマの表を参照しているビューをインポートする場合は、インポートを実行するユーザーに、READ ANY TABLEまたはSELECT ANY TABLE権限が必要です。この権限がない場合、ビューは、コンパイルされていない状態でインポートされます。ロールに権限を付与するのみでは不十分です。ビューのコンパイルには、インポートするユーザーに直接権限を付与する必要があります。

親トピック: データベース・オブジェクトのインポートに関する考慮点

27.16.14 パーティション表のインポート

パーティション表のインポートは、データベース・オブジェクトをインポートする場合の考慮点の1つです。エクスポートしたパーティション表と同じパーティション名またはサブパーティション名を使用してパーティション表を作成するために、SYS_Pnnn形式の名前もインポートされます

エクスポートしたパーティション表と同じパーティション名またはサブパーティション名を使用してパーティション表を作成するために、SYS_Pnnn形式の名前もインポートされます同じ名前のパーティション表がすでに存在している場合、これ以降の処理はIGNOREパラメータに指定されている値によって異なります。

SKIP_UNUSABLE_INDEXES=yが指定されていないかぎり、インポート時に非パーティション索引またはパーティション索引が(索引使用禁止に設定されているか、またはその他の不適合が理由で)メンテナンスできない場合は、エクスポート・データはターゲット表にインポートできません。

親トピック: データベース・オブジェクトのインポートに関する考慮点

27.17 ファイングレイン・アクセス・コントロールに対するサポート

ファイングレイン・アクセス・コントロール・ポリシーをリストアするには、ファイングレイン・アクセス・コントロール・ポリシーが使用可能な表を含むエクスポート・ファイルからインポートするユーザーが、表のセキュリティ・ポリシーを回復するためのDBMS_RLSパッケージに対するEXECUTE権限が必要です。

ファイングレイン・アクセス・コントロール・ポリシーが使用可能な表を含むエクスポートファイルからインポートを行うための正しい権限が付与されていない場合は、警告メッセージが発行されます。

親トピック: オリジナルのインポート

27.18 スナップショットおよびスナップショット・ログ

特定の状況、特にデータ・ウェアハウスに関連する場合、スナップショットは、マテリアライズド・ビューと呼ばれます。これらの項では、そのような場合でもスナップショットという用語を使用します。

- スナップショット・ログ
  インポート先のデータベースにマスター表がすでに存在し、そのマスター表にスナップショット・ログがある場合は、ダンプ・ファイルのスナップショット・ログがインポートされます。
- スナップショット
  エクスポート・ファイルからリストアされたスナップショットは、前の状態に戻ってしまいます。

親トピック: オリジナルのインポート

27.18.1 スナップショット・ログ

インポート先のデータベースにマスター表がすでに存在し、そのマスター表にスナップショット・ログがある場合は、ダンプ・ファイルのスナップショット・ログがインポートされます。

ROWIDスナップショット・ログのエクスポートでは、スナップショット・ログに記録されているROWIDはインポートした後には意味を持ちません。このため、各ROWIDのスナップショットによる最初の高速リフレッシュは失敗し、完全リフレッシュが必要であることを示すエラーが発生します。

リフレッシュのエラーを回避するには、ROWIDのスナップショット・ログをインポートしてから完全リフレッシュを実行してください。完全リフレッシュを実行すると、後続の高速リフレッシュが適切に行われます。これに対し、主キー・スナップショット・ログをエクスポートした場合は、インポートした後でも主キーの値は意味を持ちます。したがって、インポートした後でも、主キーのスナップショットにより、高速リフレッシュを実行できます。

親トピック: スナップショットおよびスナップショット・ログ

27.18.2 スナップショット

エクスポート・ファイルからリストアされたスナップショットは、前の状態に戻ってしまいます。

インポートでは、最後のリフレッシュが実行された時刻が、スナップショット表定義の一部としてインポートされます。次のリフレッシュ時刻を計算する機能もインポートされます。

各リフレッシュによって、署名が付けられます。高速リフレッシュでは、スナップショットを最新に保つため、その署名の時刻から日付を決定するログ・エントリが使用されます。高速リフレッシュが完了した時点で署名は削除され、新しい署名が付けられます。他のスナップショットのリフレッシュに必要でないログ・エントリ(残っている最も古い署名よりも前の時刻を持つすべてのログ・エントリ)も削除されます。

- スナップショットのインポート
  エクスポート・ファイルからスナップショットをリストアすると、問題が発生する場合があります。
- 異なるスキーマへのスナップショットのインポート
  スナップショットおよび関連項目は、DDL文で指定されたスキーマ名でエクスポートされます。

親トピック: スナップショットおよびスナップショット・ログ

27.18.2.1 スナップショットのインポート

エクスポート・ファイルからスナップショットをリストアすると、問題が発生する場合があります。

スナップショットが時刻Aにリフレッシュされ、時刻Bにエクスポートされ、時刻Cに再びリフレッシュされたとします。破損などの問題が発生したため、スナップショットを削除してインポートしなおすことによってリストアする必要があります。新規にインポートされたバージョンは、時刻Aとして記録された最終リフレッシュ時刻を保持しています。高速リフレッシュに必要となるログ・エントリが存在しなくなっている可能性があります。ログ・エントリが存在する場合は(たとえば、リフレッシュする必要のある別のスナップショットに必要なため)、このエントリが使用され、高速リフレッシュは正常に完了します。ログ・エントリが存在しない場合、高速リフレッシュは失敗し、完全リフレッシュが必要であることを示すエラーが発生します。

親トピック: スナップショット

27.18.2.2 異なるスキーマへのスナップショットのインポート

スナップショットおよび関連項目は、DDL文で指定されたスキーマ名でエクスポートされます。

異なるスキーマにインポートする場合は、FROMUSERパラメータおよびTOUSERパラメータを使用します。これは、異なるスキーマ

にインポートできないスナップショット・ログには適用されません。

> ノート:
>
> ファンクション索引、ファンクション、プロシージャ、トリガー、型本体、ビューなどの内部に表示されるスキーマ名は、FROMUSER または TOUSER の処理には影響されません。影響を受けるのは、オブジェクト名のみです。インポートの完了後、TOUSER スキーマに含まれる項目が古いスキーマ(FROMUSER)を参照しているかどうかを手動で確認し、必要に応じて修正する必要があります。

親トピック: スナップショット

27.19 トランスポータブル表領域

トランスポータブル表領域機能は、一連の表領域を、あるOracle Databaseから他のOracle Databaseに移動できる機能です。

> ノート:
>
> トランスポータブル表領域をエクスポートした後、それよりも古いリリース・レベルのデータベースにインポートすることはできません。ターゲット・データベースのリリース・レベルは、ソース・データベース以上である必要があります。

一連の表領域を移動またはコピーするには、表領域を読取り専用にし、表領域のデータ・ファイルをターゲット・データベースに手動でコピーしてから、エクスポートおよびインポートを使用して、データ・ディクショナリに格納されているデータベース情報(メタデータ)をターゲット・データベースに移動します。データ・ファイルの転送は、オペレーティング・システムのコピー機能、バイナリ・モードFTP、CD-ROMへの出力などのような、フラット・バイナリ・ファイルのコピー機能を使用して行われます。

データ・ファイルのコピーおよびメタデータのエクスポートの後、表領域を任意に読み書き両用モードにできます。

次のパラメータで、トランスポータブル表領域のメタデータを移動できます。

- TABLESPACES
- TRANSPORT_TABLESPACE

インポート操作時のこれらのパラメータの使用方法の詳細は、「TABLESPACES」および「TRANSPORT_TABLESPACE」を参照してください。

関連項目:

- トランスポータブル表領域の管理の詳細は、『Oracle Database管理者ガイド』を参照してください。

親トピック: オリジナルのインポート

27.20 記憶域パラメータ

デフォルトでは、表は、元の表領域にインポートされます。

その表領域が存在しない場合、またはユーザーがその表領域に十分な割当て制限を持っていない場合、次の表の場合を除いて、そのユーザーにはデフォルトの表領域が割り当てられます。

- パーティション表
- 特定の型の表
- LOB型列、VARRAY型列またはOPAQUE型列を含む表
- オーバーフロー・セグメントがあるIOTを含む表

ユーザーがデフォルトの表領域に対する十分な割当て制限を持っていない場合、そのユーザーの表はインポートされません。この制限の利用方法の詳細は、「表領域を再編成する方法」を参照してください。

- OPTIMALパラメータ
  ロールバック・セグメントのための記憶域パラメータOPTIMALは、エクスポートおよびインポート時には保持されません。
- OID索引とLOB列の記憶域パラメータ
  表は、それぞれの現行の記憶域パラメータを使用してエクスポートされます。
- 記憶域パラメータの上書き
  インポート・ユーティリティを使用してデータをインポートする前に、別の記憶域パラメータで、事前に大きな表を作成した方がよい場合があります。

親トピック: オリジナルのインポート

27.20.1 OPTIMALパラメータ

ロールバック・セグメントのための記憶域パラメータOPTIMALは、エクスポートおよびインポート時には保持されません。

親トピック: 記憶域パラメータ

27.20.2 OID索引とLOB列の記憶域パラメータ

表は、その表の現行の記憶域パラメータを使用してエクスポートされます。

オブジェクト表に関しては、OIDINDEXの作成時に、OIDINDEXの現行の記憶域パラメータおよび名前が設定されている場合は、それらを使用して作成されます。LOB型列、VARRAY型列またはOPAQUE型列が含まれている表に関しては、LOB型データ、VARRAY型データまたはOPAQUE型データは、それらの現行の記憶域パラメータを使用して作成されます。

エクスポートの前に、ユーザーが既存の表の記憶域パラメータを変更する場合もありますが、このような場合、表は変更された記憶域パラメータを使用してエクスポートされます。ただし、LOBデータの記憶域パラメータは、エクスポートの前には変更できません(たとえば、LOB列のチャンク・サイズ、LOB列がCACHEまたはNOCACHEか、など)。

LOBデータは、含んでいる表と同じ表領域に存在しない場合があることに、注意してください。このデータの表領域は、インポート時に読取り/書込みが可能である必要があります。そうでない場合、表はインポートされません。

LOBデータまたはLOB索引がインポート時に存在しない表領域にある場合、またはユーザーがその表領域に対して必要な割当て制限を持っていない場合、表はインポートされません。表領域の句は、表に関する句も含めて複数の句を同時に指定できるため、インポート時にエラーが発生しても、インポート・ユーティリティではどの表領域句が原因のエラーかを特定できません。

親トピック: 記憶域パラメータ

27.20.3 記憶域パラメータの上書き

インポート・ユーティリティを使用してデータをインポートする前に、別の記憶域パラメータで、事前に大きな表を作成した方がよい場合があります。

その場合は、コマンドラインまたはパラメータ・ファイルにIGNORE=yを指定します。

親トピック: 記憶域パラメータ

27.21 読取り専用表領域

読取り専用表領域はエクスポート可能です。インポートでは、表領域がターゲット・データベース内に存在しない場合、読取り/書込み表領域として表領域が作成されます。

読取り専用機能を取得する場合は、インポート後にその表領域を手動で読取り専用にする必要があります。

ターゲット・データベース内に表領域がすでに存在し、読取り専用である場合は、インポート前にこの表領域を読取り/書込み可能にする必要があります。

親トピック: オリジナルのインポート

27.22 表領域の削除

インポート前に、オブジェクトに別の表領域を使用するように再定義すると、表領域を削除できます。impコマンドの発行時には、IGNORE=yを指定します。

表領域を削除するには、通常、全データベース・エクスポートを実行し、(ログオフの前に)削除する表領域と同名の表領域をブロック数0(ゼロ)で作成します。IGNORE=yが指定されていると、インポート時にその表領域に関するCREATE TABLESPACE文はエラーとなります。これにより、削除対象である不要な表領域は作成されません。

その表領域のすべてのオブジェクト(ただし、パーティション表、特定の型の表、LOB列またはVARRAY列を含む表またはオーバーフロー・セグメントのあるIOTを除く)が、そのオブジェクトの所有者のデフォルトの表領域にインポートされます。インポート・ユーティリティでは、エラーの原因となった表領域を特定できません。かわりに、ユーザー自身が表の作成後、IGNORE=yを指定して表のインポートを実行する必要があります。

その表領域が存在しない場合、またはデフォルトの表領域に対するユーザーの割当て制限が十分でない場合は、オブジェクトはデフォルトの表領域にインポートされません。

親トピック: オリジナルのインポート

27.23 表領域を再編成する方法

ユーザーの割当て制限が十分な場合、そのユーザーの表はエクスポート元と同じ表領域にインポートされます。

その表領域がもう存在しないか、またはユーザーの割当て制限が十分でない場合は、そのユーザーに対するデフォルトの表領域が適用されます。ただし、パーティション表、LOB列またはVARRAY列が含まれている表、特定の型の表およびオーバーフロー・セグメントのある索引構成表には適用されません。この条件を利用して、表領域間でユーザーの表を移動できます。

たとえば、全データベース・エクスポートを実行した後、joeの表を表領域Aから表領域Bに移動する必要があるとします。この場合には、次のステップを実行します。

1. joeがUNLIMITED TABLESPACE権限を持っている場合、その権限を取り消します。表領域Aに対するjoeの割当て制限を0(ゼロ)に設定します。さらに、このような権限または割当て制限を含む可能性のあるすべてのロールを取り消します。

   ロールの取消しでは、カスケード効果はありません。したがって、joeによって他のロールを付与されたユーザーは影響を受けません。

2. joeの表をエクスポートします。

3. 表領域Aからjoeの表を削除します。
4. joeに表領域Bの割当て制限を付与し、joeのデフォルトの表領域とします。
5. joeの表をインポートします(デフォルトでは、joeの表は表領域Bにインポートされます)。

親トピック: オリジナルのインポート

27.24 統計情報のインポート

統計情報がエクスポート時に必要で、表にアナライザ統計が利用できる場合、表の統計情報の再計算に使用されるANALYZE文が、エクスポートによってダンプ・ファイルに含まれます。

ほとんどの場合、表、索引および列に対する計算済オプティマイザ統計情報も、ダンプ・ファイルにエクスポートされます。インポート・パラメータの詳細は、「STATISTICS」を参照してください。

ANALYZE文の実行には時間がかかるため、通常のインポートでは、エクスポートによって保存されるANALYZE文を計算するのではなく、表(およびその索引や列)の計算済オプティマイザ統計情報を使用してください。デフォルトでは、エクスポート・ダンプ・ファイルにある計算済統計情報が使用されます。

エクスポート・ユーティリティによって、計算済統計情報に問題ありというフラグが付けられる場合もあります。次の状況では、計算済統計情報ではなく、問題のない統計情報のみをインポートする必要がある場合があります。

- ダンプ・ファイル、インポート・クライアント、インポート・データベース間の文字セット変換(計算済統計情報で暗黙的に照合順番が変更されている可能性が高いため)。
- 表のインポート時に行エラーが発生した場合。
- パーティション・レベル・インポートが実行された場合(列統計情報が、すでに正確ではないため)。

> ノート:
>
> ROWS=n を指定しても、計算済統計情報は使用できます。この機能により、本番データベースからの統計情報を使用して、非本番データベース上で問合せの実行計画のチューニングを行うことができます。このような場合は、STATISTICS=SAFE を指定する必要があります。

場合によっては、インポート時に、計算済統計情報ではなく、常にANALYZE文を使用する必要があります。たとえば、分散データベースから収集した統計情報は、そのデータが圧縮形式でインポートされると、適切でなくなる場合があります。このような場合は、インポート時にSTATISTICS=RECALCULATEを指定して、統計情報を再計算する必要があります。

インポート時に統計情報を確定しない場合は、STATISTICS=NONEを指定する必要があります。

親トピック: オリジナルのインポート

27.25 エクスポート・ユーティリティおよびインポート・ユーティリティを使用したデータベース移行のパーティション化

エクスポート・ユーティリティおよびインポート・ユーティリティを使用して大規模データベースを移行する場合、移行を複数のエクスポート・ジョブおよびインポート・ジョブにパーティション化するとより効率的です。

移行をパーティション化する場合は、次のメリットおよびデメリットに注意してください。

- 移行をパーティション化する場合のメリット

移行をパーティション化する場合のメリットについて説明します。

- 移行をパーティション化する場合のデメリット
  移行をパーティション化する場合のデメリットについて説明します。
- エクスポート・ユーティリティおよびインポート・ユーティリティを使用したデータベース移行のパーティション化方法
  エクスポート・ユーティリティおよびインポート・ユーティリティを使用した移行のパーティション化方法について説明します。

親トピック: オリジナルのインポート

27.25.1 移行をパーティション化する場合のメリット

移行をパーティション化する場合のメリットについて説明します。

具体的には次のとおりです。

- 多くのサブジョブをパラレルに実行できるため、移行に必要な時間を削減できます。
- 最初のエクスポート・ジョブが完了するとすぐにインポート・ユーティリティを起動できます。

親トピック: エクスポート・ユーティリティおよびインポート・ユーティリティを使用したデータベース移行のパーティション化

27.25.2 移行をパーティション化する場合のデメリット

移行をパーティション化する場合のデメリットについて説明します。

具体的には次のとおりです。

- エクスポートおよびインポートのプロセスがより複雑になります。
- 特定の型のオブジェクトに対する相互スキーマ参照のサポートが損なわれます。たとえば、異なるスキーマの表に対する外部キー制約を持つ表がスキーマに含まれている場合、その表を依存スキーマにインポートしたときに、必要な親レコードが存在しない場合があります。

親トピック: エクスポート・ユーティリティおよびインポート・ユーティリティを使用したデータベース移行のパーティション化

27.25.3 エクスポート・ユーティリティおよびインポート・ユーティリティを使用したデータベース移行のパーティション化方法

エクスポート・ユーティリティおよびインポート・ユーティリティを使用した移行のパーティション化方法について説明します。

データベースの移行をパーティション化方法で実行するには、次のステップに従います。

1. データベースのすべての最上位メタデータに、次のコマンドを発行します。
   a. `exp FILE=full FULL=y CONSTRAINTS=n TRIGGERS=n ROWS=n INDEXES=n`
   b. `imp FILE=full FULL=y`
2. データベースの各スキーマnに、次のコマンドを発行します。
   a. `exp OWNER=scheman FILE=scheman`
   b. `imp FILE=scheman FROMUSER=scheman TOUSER=scheman IGNORE=y`

すべてのエクスポートはパラレルで実行できます。`full.dmp`のインポートが完了すると、残りのインポートもパラレルで実行できます。

親トピック: エクスポート・ユーティリティおよびインポート・ユーティリティを使用したデータベース移行のパーティション化

27.26 インポート操作のチューニングに関する考慮点

これらの項では、インポート操作のパフォーマンスを向上させる方法を説明します。

- システム・レベル・オプションの変更
  インポート操作のパフォーマンス向上に役立つ可能性があるシステム・レベル・オプションについて説明します。
- 初期化パラメータの変更
  初期化パラメータ・ファイルの設定に関してこれらの点を考慮すると、インポート操作のパフォーマンスの向上につながる場合があります。
- インポート・オプションの変更
  インポート・オプションの使用について、これらの点を考慮すると、パフォーマンスの向上に役立つ場合があります。
- 大量のLOBデータの処理
  大量のLOBデータのインポートについて説明します。
- 大量のLONGデータの処理
  LONG列を持つ表をインポートすると、I/Oおよびディスクの使用率が高くなり、その結果、インポート操作のパフォーマンスが低くなります。

親トピック: オリジナルのインポート

27.26.1 システム・レベル・オプションの変更

インポート操作のパフォーマンス向上に役立つ可能性があるシステム・レベル・オプションについて説明します。

具体的には次のとおりです。

- 1つの大きなロールバック・セグメントを作成および使用し、他のすべてのロールバック・セグメントをオフラインにします。一般に、インポートする最大の表の半分のサイズが、十分なロールバック・セグメントのサイズです。また、同じサイズの2つ以上のエクステントで、ロールバック・セグメントを作成することも有効です。

  ノート:

  ロールバック・セグメントのかわりに、自動 UNDO 管理を使用することをお薦めします。

- インポートが完了するまでデータベースをNOARCHIVELOGモードにします。これによって、アーカイブ・ログの作成および管理によるオーバーヘッドが削減されます。
- いくつかの大きいREDOファイルを作成し、小さいREDOログ・ファイルをすべてオフラインにします。これによって、作成されるログ・スイッチが少なくなります。
- 可能な場合、ロールバック・セグメント、表データおよびREDOログ・ファイルをすべて個別のディスクに配置します。I/O競合が削減され、スループットが増加します。
- 可能な場合、システム・リソースに対するインポート操作との競合が発生する可能性がある他のジョブは、同時に実行しないでください。
- ディクショナリ表に統計が存在しないことを確認してください。
- sqlnet.oraファイルにTRACE_LEVEL_CLIENT=OFFを設定します。
- 可能な場合、データベースの再作成時にDB_BLOCK_SIZEの値を増加します。ブロック・サイズを大きくすると、必要なI/Oサイクル数は少なくなります。この変更は永続的であるため、行う前にすべての影響を考慮してください。

親トピック: インポート操作のチューニングに関する考慮点

27.26.2 初期化パラメータの変更

初期化パラメータ・ファイルの設定に関してこれらの点を考慮すると、インポート操作のパフォーマンスの向上につながる場合があります。

- LOG_CHECKPOINT_INTERVALをREDOログ・ファイルのサイズより大きい数に設定します。オペレーティング・システム・ブロックの数(ほとんどのUNIXシステムでは512)です。これによって、(ログ・スイッチ時の)チェックポイントが最小まで削減されます。
- SORT_AREA_SIZEの値を増加します。増加する量は、システムで行われている他のアクティビティおよび使用可能な空きメモリーの量によって異なります。(システムでスワッピングおよびページングが開始された場合、設定値が高くなりすぎる可能性があります)。
- DB_BLOCK_BUFFERSおよびSHARED_POOL_SIZEの値を増加します。

親トピック: インポート操作のチューニングに関する考慮点

27.26.3 インポート・オプションの変更

インポート・オプションの使用についてこれらの点を考慮すると、パフォーマンスの向上につながる場合があります。

「インポート・パラメータ」に示す使用可能なオプションの各説明も参照してください。

- COMMIT=Nを設定します。これによって、各バッファの後ではなく、各オブジェクト(表)の後にインポートがコミットされます。このため、1つの大きなロールバック・セグメントが必要となります。(ロールバック・セグメントは将来のリリースでは非推奨になるため、かわりに、自動UNDO管理の使用をお薦めします。)
- システム・アクティビティ、データベースのサイズなどに応じて、BUFFERまたはRECORDLENGTHに大きい値を指定します。サイズを大きくすると、データに対してエクスポート・ファイルがアクセスされる回数が削減されます。通常は数MBで十分です。バッファ・サイズが大きすぎることを示す、過剰なページングおよびスワッピング・アクティビティをチェックします。
- 索引はインポート後に作成できるため、時間に余裕がある場合は、INDEXES=Nの設定を考慮してください。設定する場合は、INDEXFILEパラメータを使用して索引作成のDLLを抽出するか、またはINDEXES=YおよびROWS=Nを指定してインポートを再実行します。

親トピック: インポート操作のチューニングに関する考慮点

27.26.4 大量のLOBデータの処理

大量のLOBデータのインポートについて説明します。

具体的には次のとおりです。

- 索引を削除すると、総インポート時間が大幅に削減されます。LOBロケータには、インポート時に明示的に削除または無視できない主キーが存在するため、LOBデータのインポート時には特別な注意が必要です。
- データのロードを完了するために使用できる十分な領域が、大きな連続したチャンクで存在することを確認してください。

親トピック: インポート操作のチューニングに関する考慮点

27.26.5 大量のLONGデータの処理

LONG列を持つ表をインポートすると、I/Oおよびディスクの使用率が高くなり、その結果、インポート操作のパフォーマンスが低くなります。

大量のLONGデータのインポート時にパフォーマンスを向上させる特定のパラメータは存在しませんが、この項で説明したチューニングに関する一般的な考慮点に注意することによって、パフォーマンス全体が向上する場合があります。

関連項目:

LONG列のインポート

親トピック: インポート操作のチューニングに関する考慮点

27.27 リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用

これらの項では、リリースが異なるエクスポート・ユーティリティおよびOracle Databaseを使用することに関連した互換性の問題について説明します。

異なるリリースのOracle Database間でデータを移動するときは、必ず次の基本的な規則に従います。

- インポート・ユーティリティとデータのインポート先データベース(ターゲット・データベース)は同じバージョンである必要があります。たとえば、インポート・ユーティリティ9.2.0.7を使用して9.2.0.8のデータベースにインポートしようとすると、エラーが発生する場合があります。
- エクスポート・ユーティリティのバージョンは、ソースまたはターゲット・データベースの古い方のバージョンに合わせる必要があります。

  たとえば、新しいリリースのデータベースにインポートするためにエクスポート・ファイルを作成する場合は、ソース・データベースと同じエクスポート・ユーティリティのバージョンを使用します。一方、古いリリースのデータベースにインポートするためのエクスポート・ファイルを作成する場合は、ターゲット・データベースのバージョンと同じバージョンのエクスポート・ユーティリティを使用します。

  - 通常は、Oracle8リリースのエクスポート・ユーティリティを使用して、Oracle9iサーバーからエクスポートし、Oracle8エクスポート・ファイルを作成できます。

- リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用時の制限事項
  リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用時には制限事項があります。
- リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用例
  リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用。

親トピック: オリジナルのインポート

27.27.1 リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用時の制限事項

異なるリリースのエクスポート・ユーティリティおよびインポート・ユーティリティを使用する場合、制限があります。

具体的には次のとおりです。

- エクスポート・ダンプ・ファイルは、特別なバイナリ形式で格納されているため、インポート・ユーティリティによる読取り専用です。
- すべてのエクスポート・ダンプ・ファイルは、上位リリースのOracle Databaseにインポートできます。
- インポート・ユーティリティでは、上位のメンテナンス・リリースまたはバージョンのエクスポート・ユーティリティで作成されたエクスポート・ダンプ・ファイルの読取りはできません。たとえば、リリース9.2のエクスポート・ダンプ・ファイルは、リリース9.0.1のインポート・ユーティリティではインポートできません。
- 下位バージョンのエクスポート・ユーティリティを上位バージョンのOracle Databaseで実行すると、下位バージョンに存在しないデータベース・オブジェクトのカテゴリは、常に、エクスポートから除外されます。
- Oracle9iのエクスポート(ダイレクト・パスまたは従来型パス)によって生成されたエクスポート・ファイルは、以前のリリースのインポート・ユーティリティと互換性がありません。したがって、インポートにはOracle9iのインポートのみを使用できます。下位互換性が問題になる場合は、Oracle9iデータベースよりも古いリリースのエクスポート・ユーティリティを使用してください。

親トピック: リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用

27.27.2 リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用例

リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用。

表27-5に、異なるリリースのOracle Database間でデータを移動させる場合に使用するエクスポート・ユーティリティおよびインポート・ユーティリティのリリースの例を示します。

表27-5 リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用

エクスポート元>インポート先	使用するエクスポート・ユーティリティのリリース	使用するインポート・ユーティリティのリリース
8.1.6 -> 8.1.6	8.1.6	8.1.6
8.1.5 -> 8.0.6	8.0.6	8.0.6
8.1.7 -> 8.1.6	8.1.6	8.1.6
9.0.1 -> 8.1.6	8.1.6	8.1.6
9.0.1 -> 9.0.2	9.0.1	9.0.2
9.0.2 -> 10.1.0	9.0.2	10.1.0
10.1.0 -> 9.0.2	9.0.2	9.0.2

表27-5に、オリジナルのエクスポートおよびインポート・ユーティリティ間のデータ移動を示します。Oracle Database 10gリリース1 (10.1)以上では、ほとんどの場合にデータ・ポンプ・エクスポートおよびデータ・ポンプ・インポート・ユーティリティを使用するこ

とをお薦めします。これらのユーティリティは、オリジナルのエクスポートおよびインポート・ユーティリティと比較して、パフォーマンスが著しく向上しています。

関連項目:

10.1以上のリリースを含めた異なるリリース間でのデータのエクスポートおよびインポートの詳細は、『Oracle Databaseアップグレード・ガイド』を参照してください。

親トピック: リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用

第V部 付録

この項では、次の項目について説明します。

A SQL*Loaderの構文図

この付録では、SQL*Loaderの構文を図形式(線路図またはDDL構文図ともいいます)で示します。

構文図の読み方

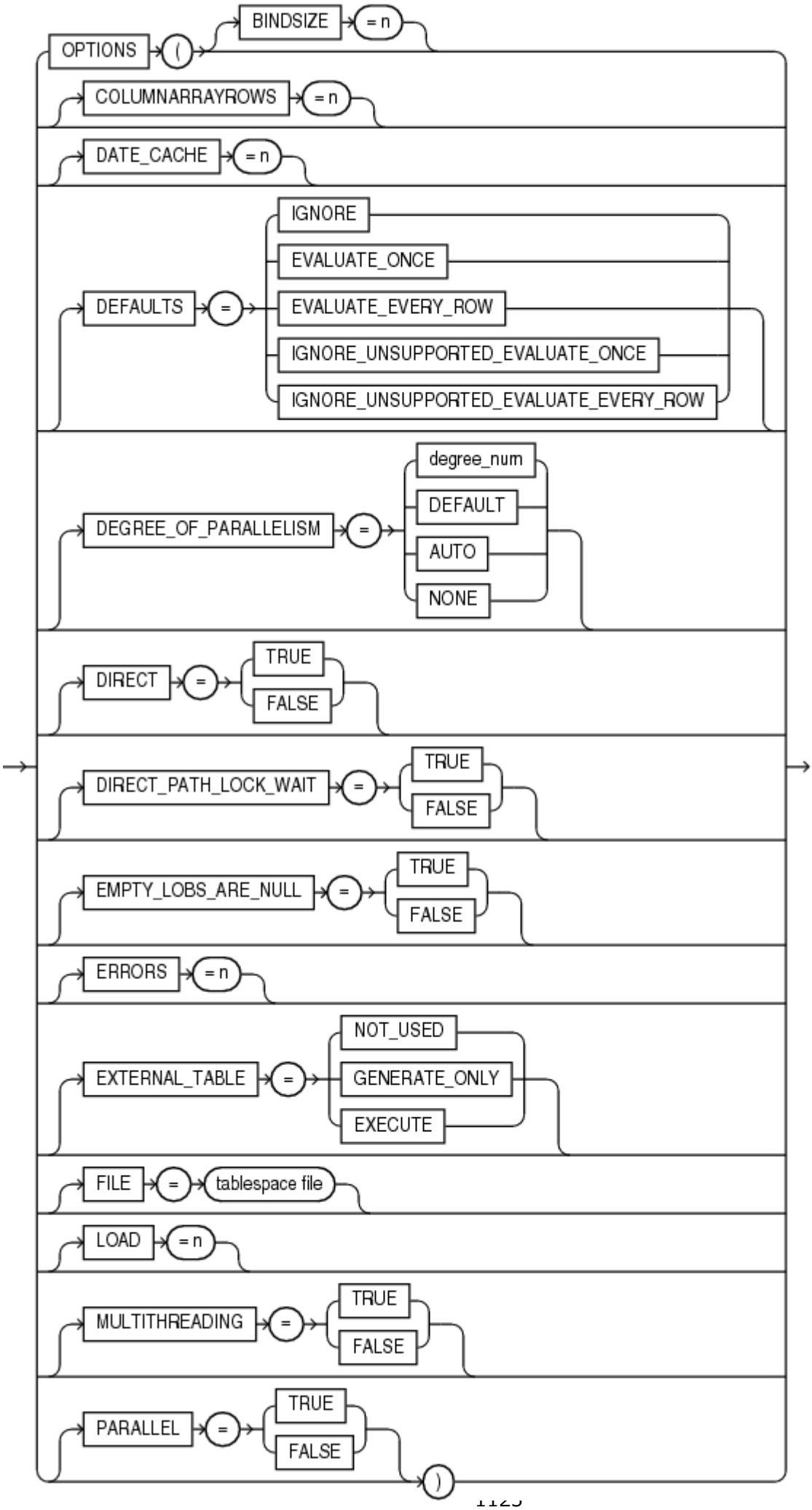
構文図とは、SQLの有効な構文を図で示したものです。構文図は、矢印が示す方向に左から右へ読んでください。

標準のSQL構文表記法の詳細は、次を参照してください。

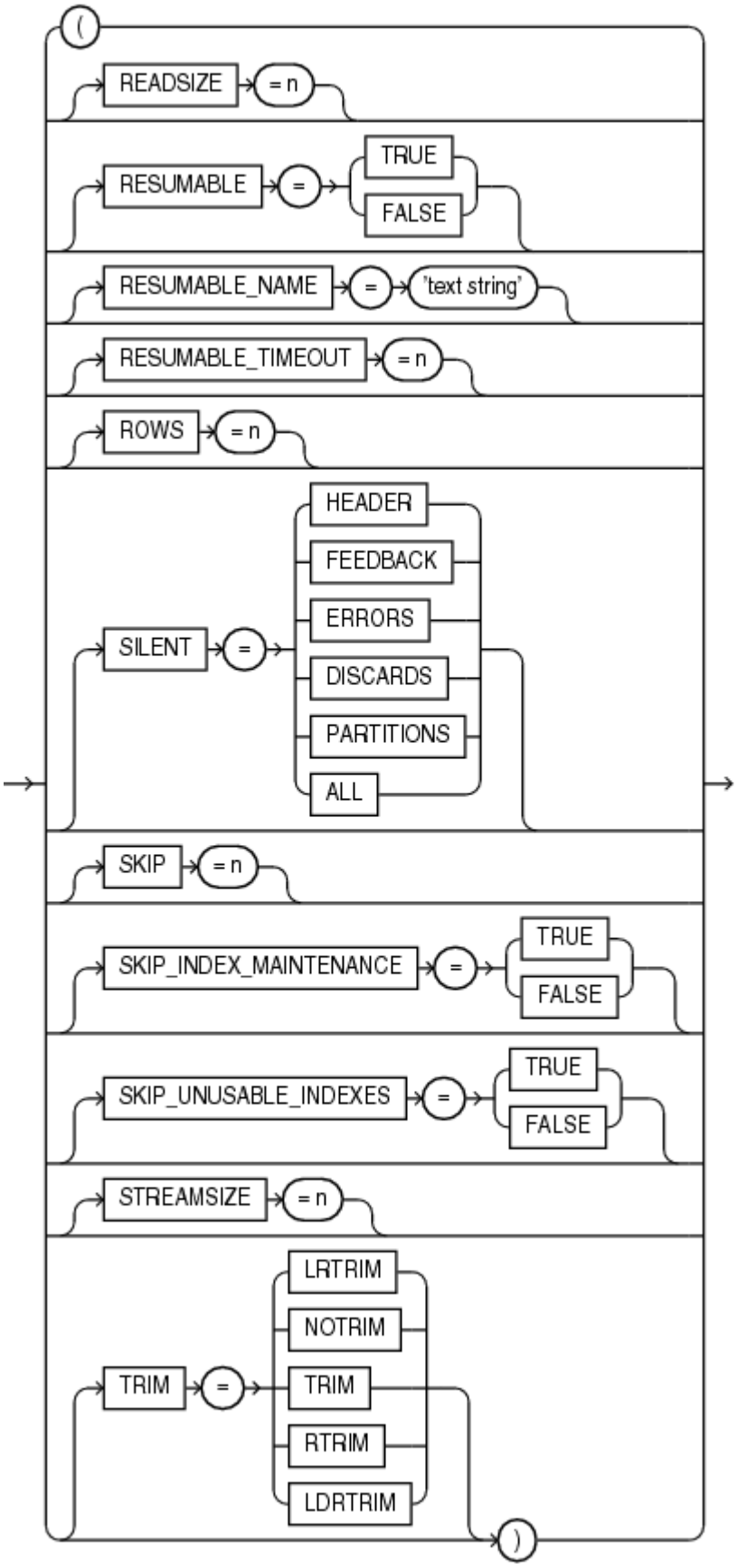
『Oracle Database SQL言語リファレンス』の「構文図の読み方」

次に示す構文図は、特定の句(pos_spec句など)が省略された形で示されています。この付録では、これらの構文図を拡張して詳しく説明します。

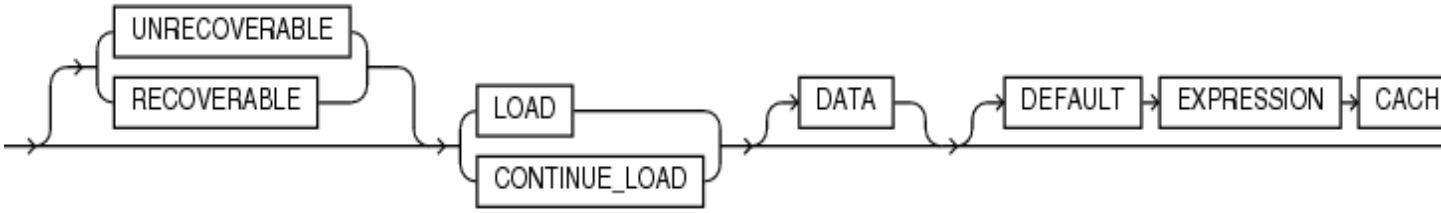
OPTIONS句

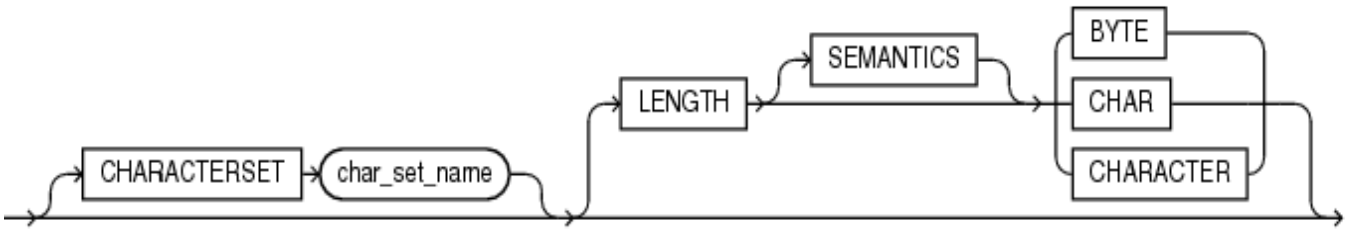

OPTIONS
(
BINDSIZE
= n
COLUMNARRAYROWS
= n
DATE_CACHE
= n
DEFAULTS
=
IGNORE
EVALUATE_ONCE
EVALUATE_EVERY_ROW
IGNORE_UNSUPPORTED_EVALUATE_ONCE
IGNORE_UNSUPPORTED_EVALUATE_EVERY_ROW
DEGREE_OF_PARALLELISM
=
degree_num
DEFAULT
AUTO
NONE
DIRECT
=
TRUE
FALSE
DIRECT_PATH_LOCK_WAIT
=
TRUE
FALSE
EMPTY_LOBS_ARE_NULL
=
TRUE
FALSE
ERRORS
= n
EXTERNAL_TABLE
=
NOT_USED
GENERATE_ONLY
EXECUTE
FILE
=
tablespace file
LOAD
= n
MULTITHREADING
=
TRUE
FALSE
PARALLEL
=
TRUE
FALSE
)

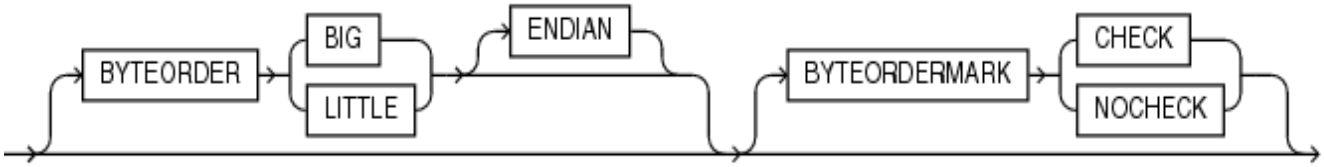
Options_Cont

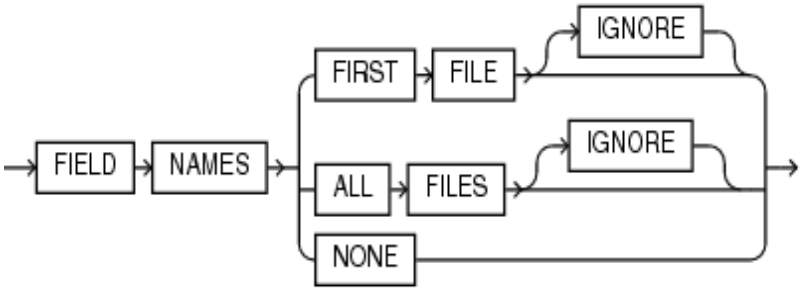


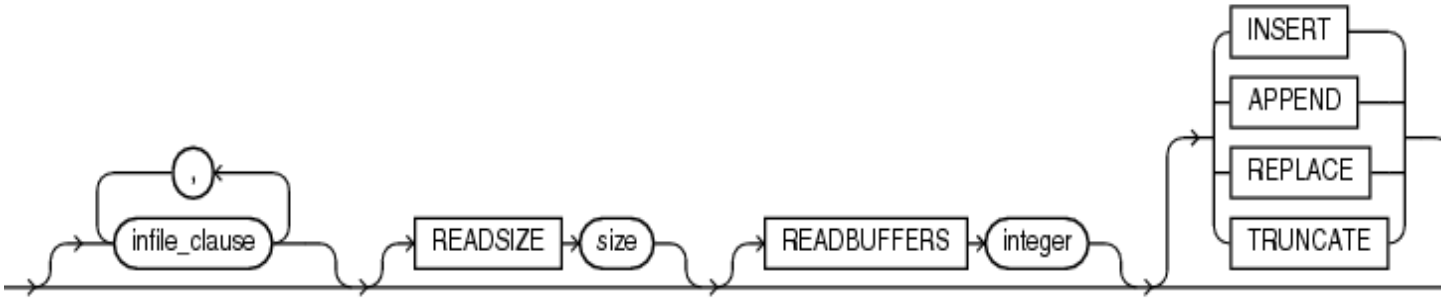
Load文

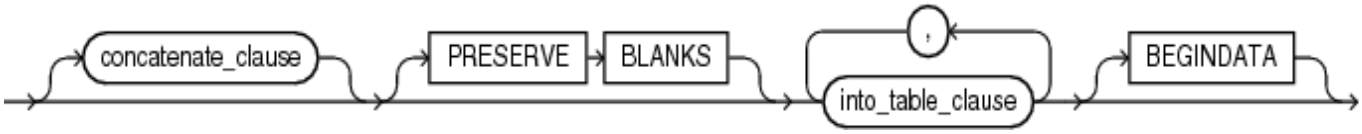




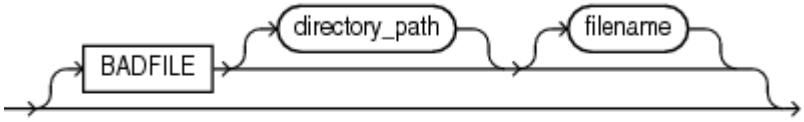


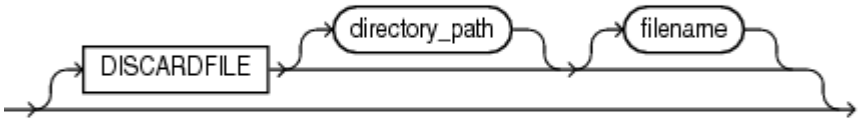


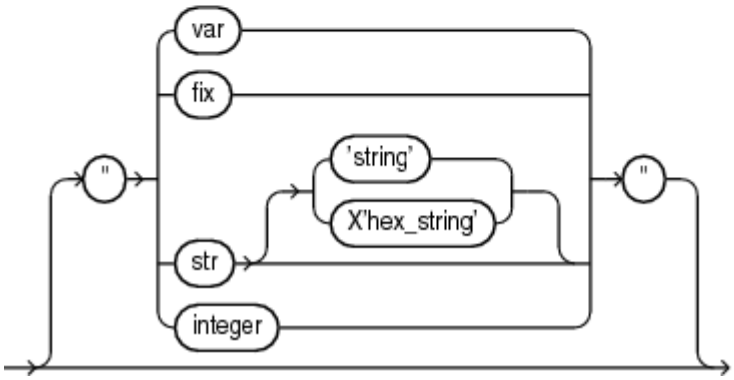




infile_clause



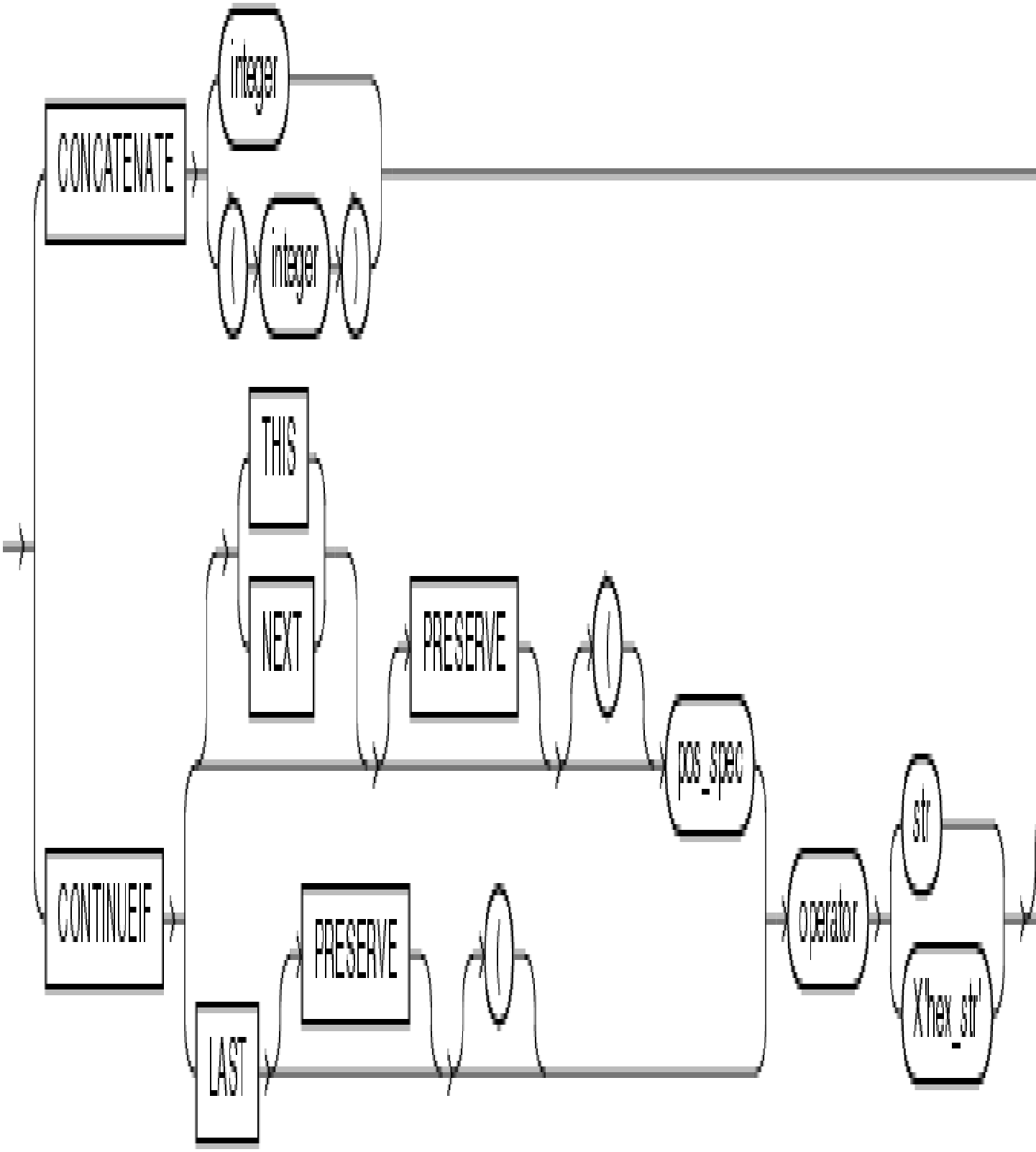




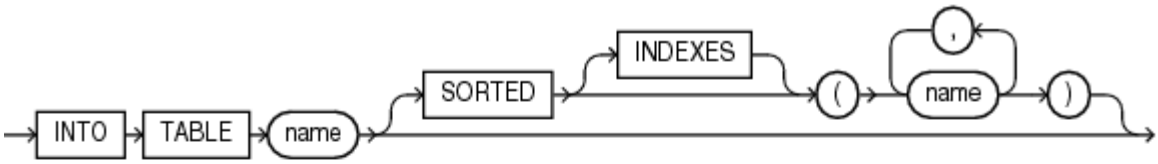
ノート:

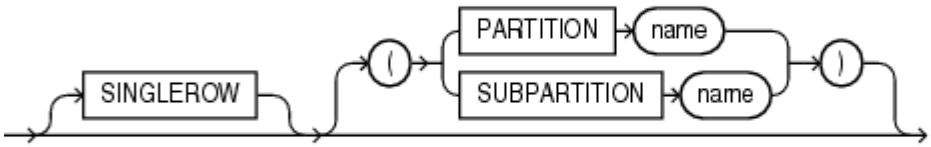
`BADFILE` および `DISCARDFILE` 句で、ディレクトリ・パスまたはファイル名(あるいはその両方)を指定する必要があります。

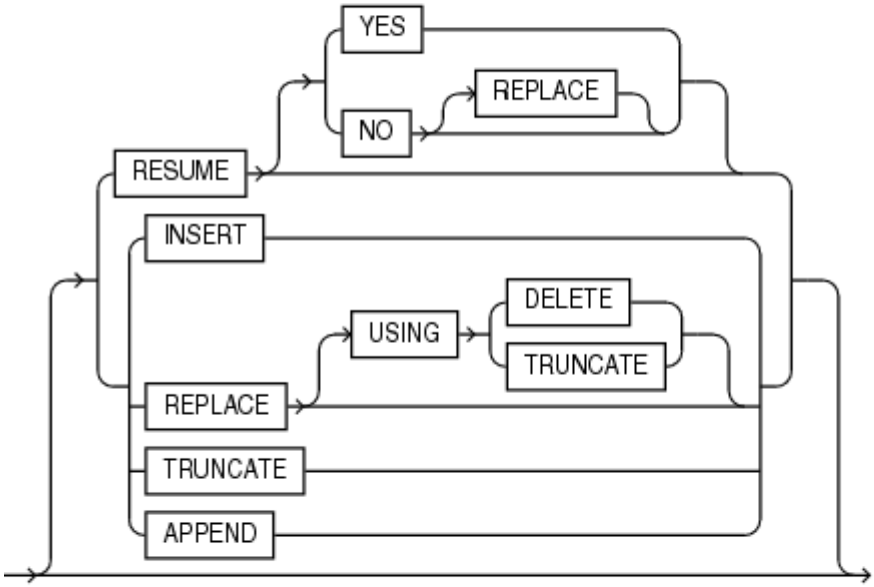
concatenate_clause

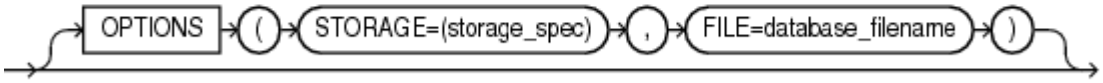


into_table_clause

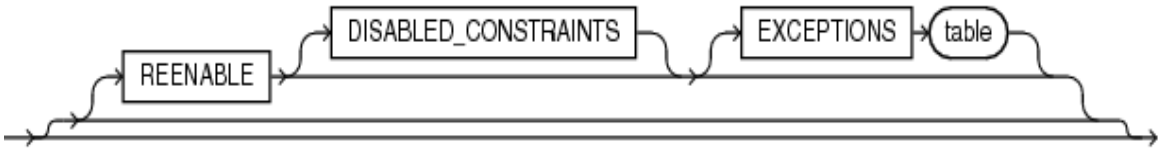


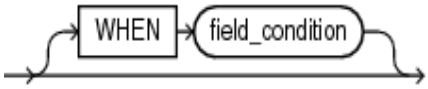


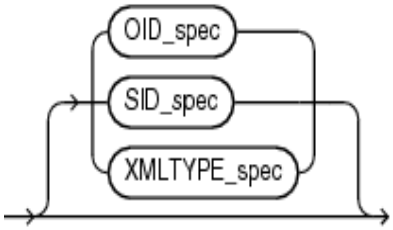


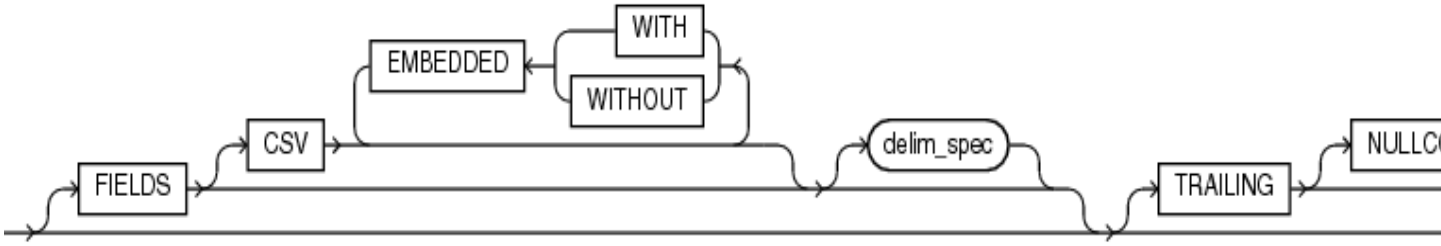


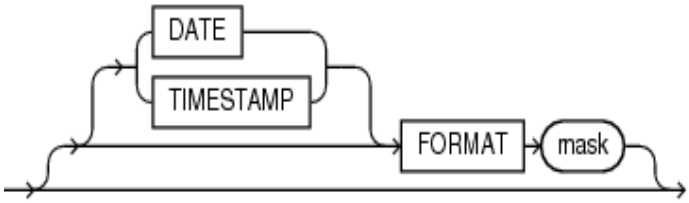
into_table_clause_continued

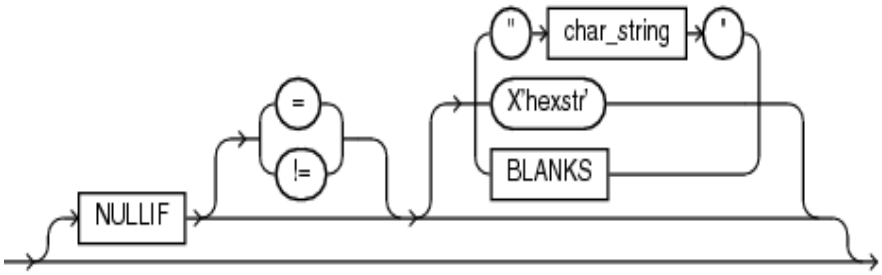


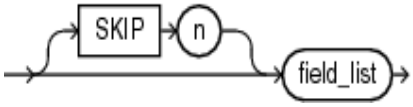




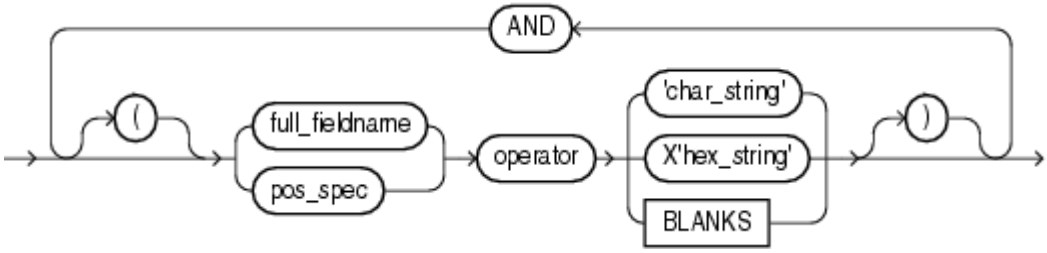




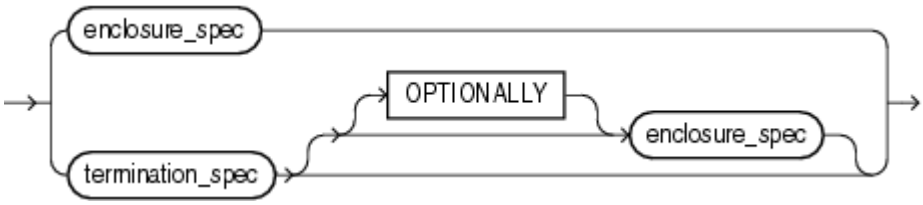




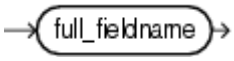
field_condition



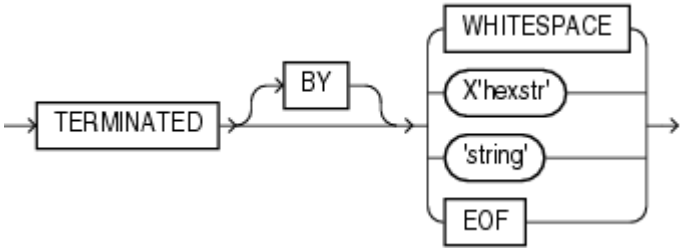
delim_spec



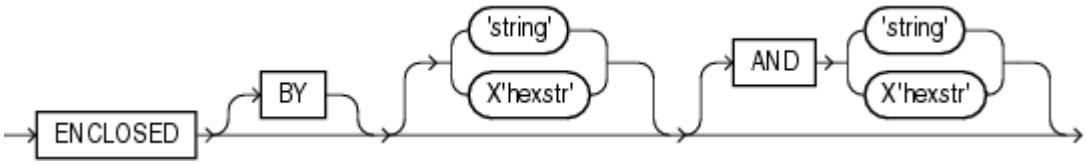
full_fieldname



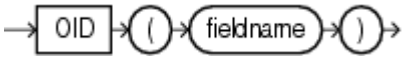
termination_spec



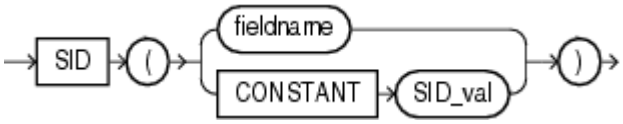
enclosure_spec



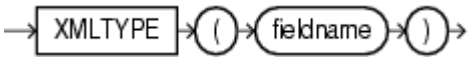
oid_spec



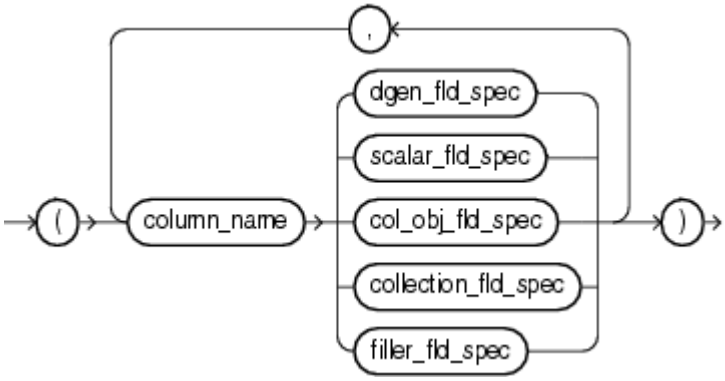
sid_spec



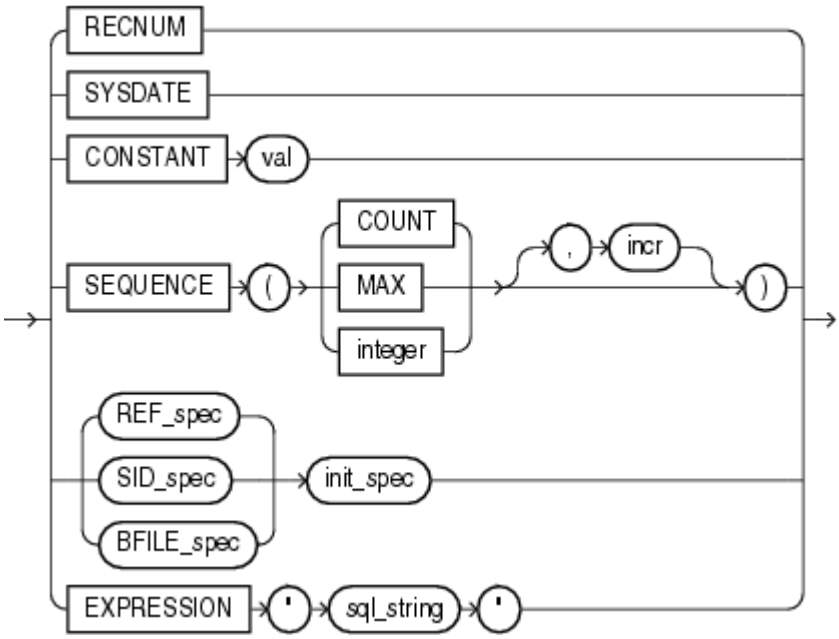
xmltype_spec



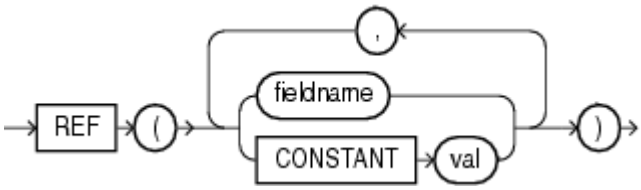
field_list



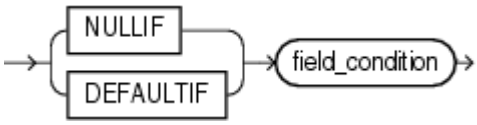
dgen_fld_spec



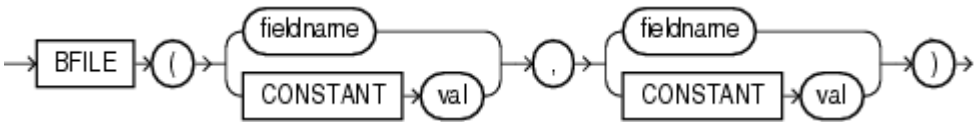
ref_spec



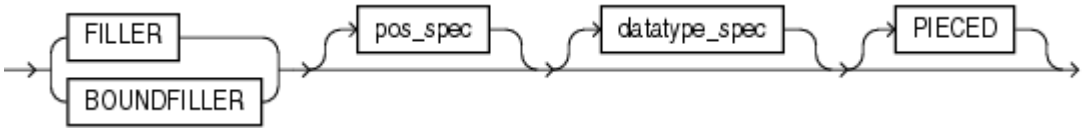
init_spec



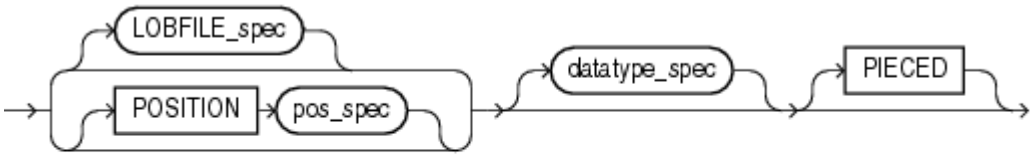
bfile_spec



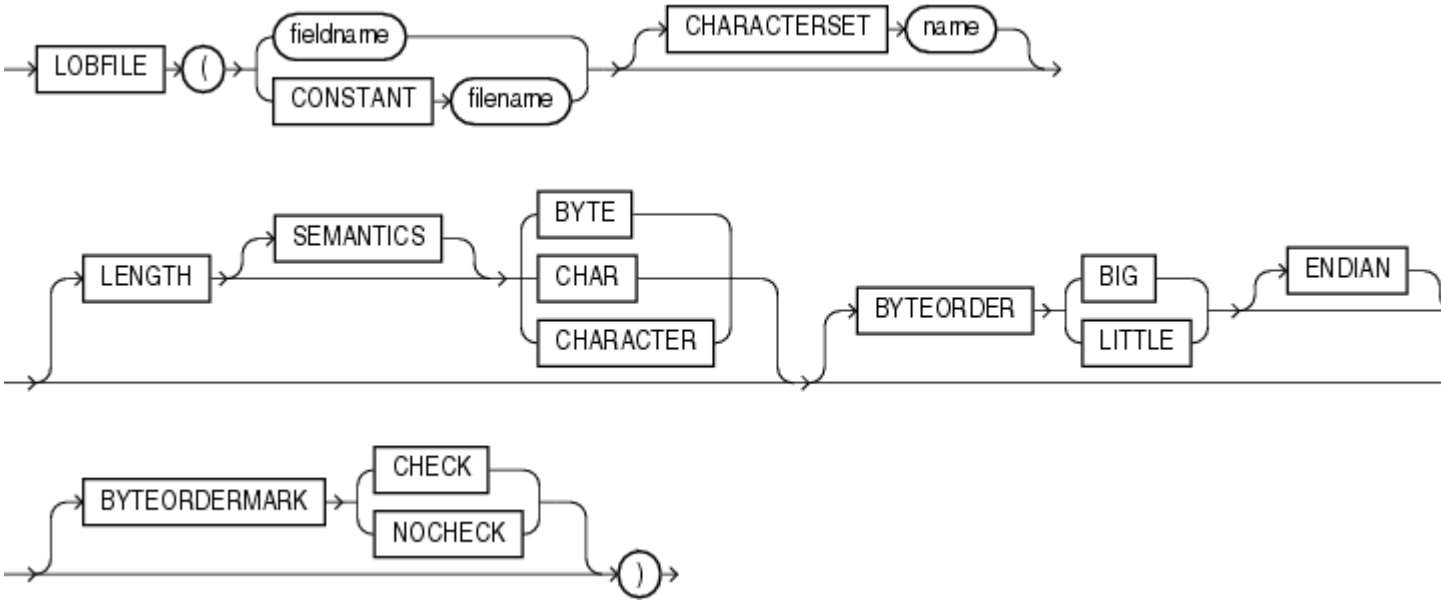
filler_fld_spec



scalar_fld_spec

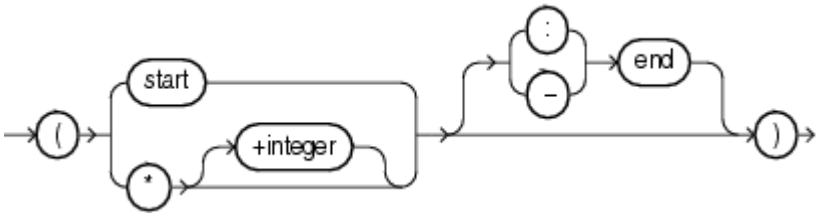


lobfile_spec

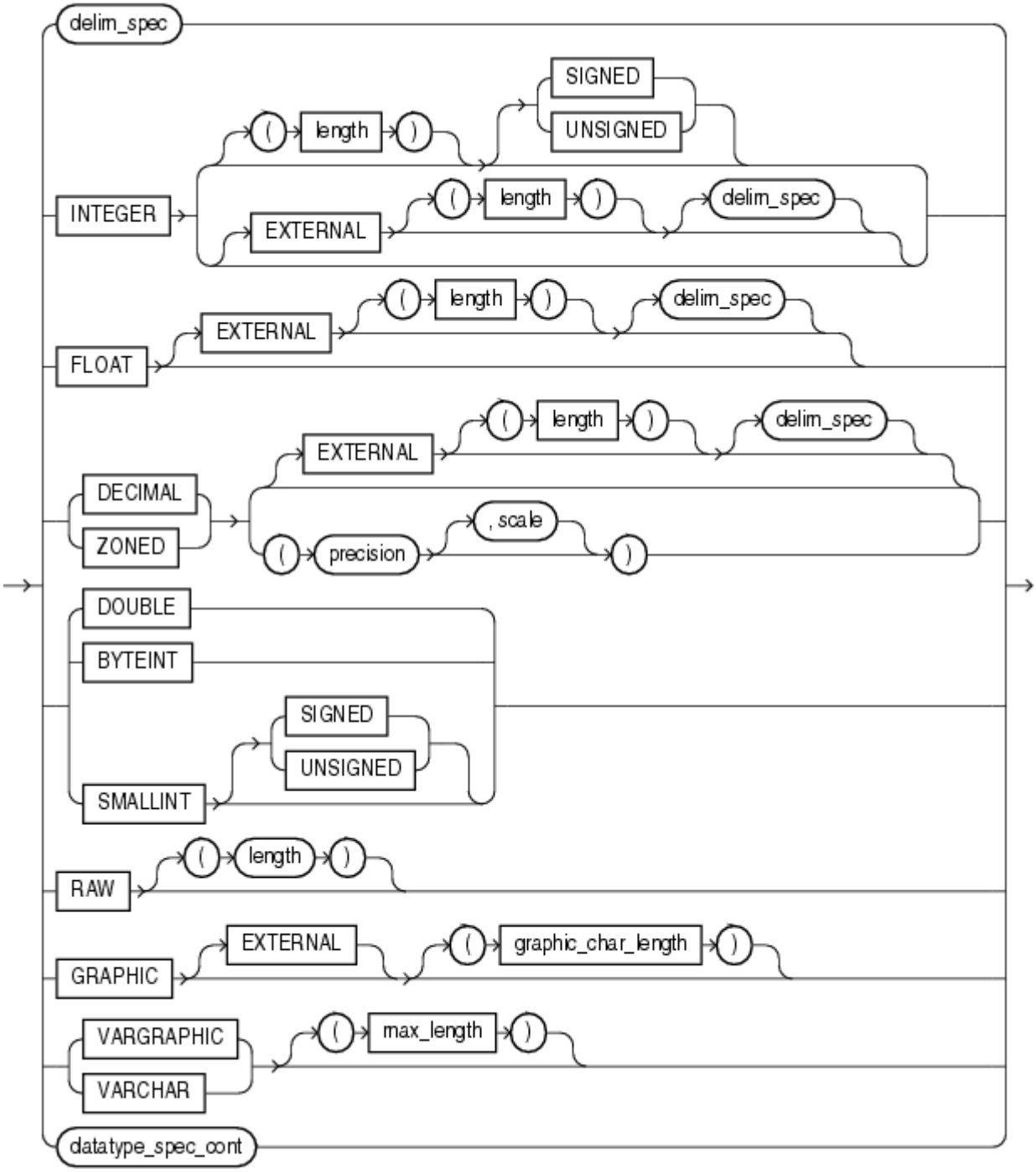


lls_field_spec

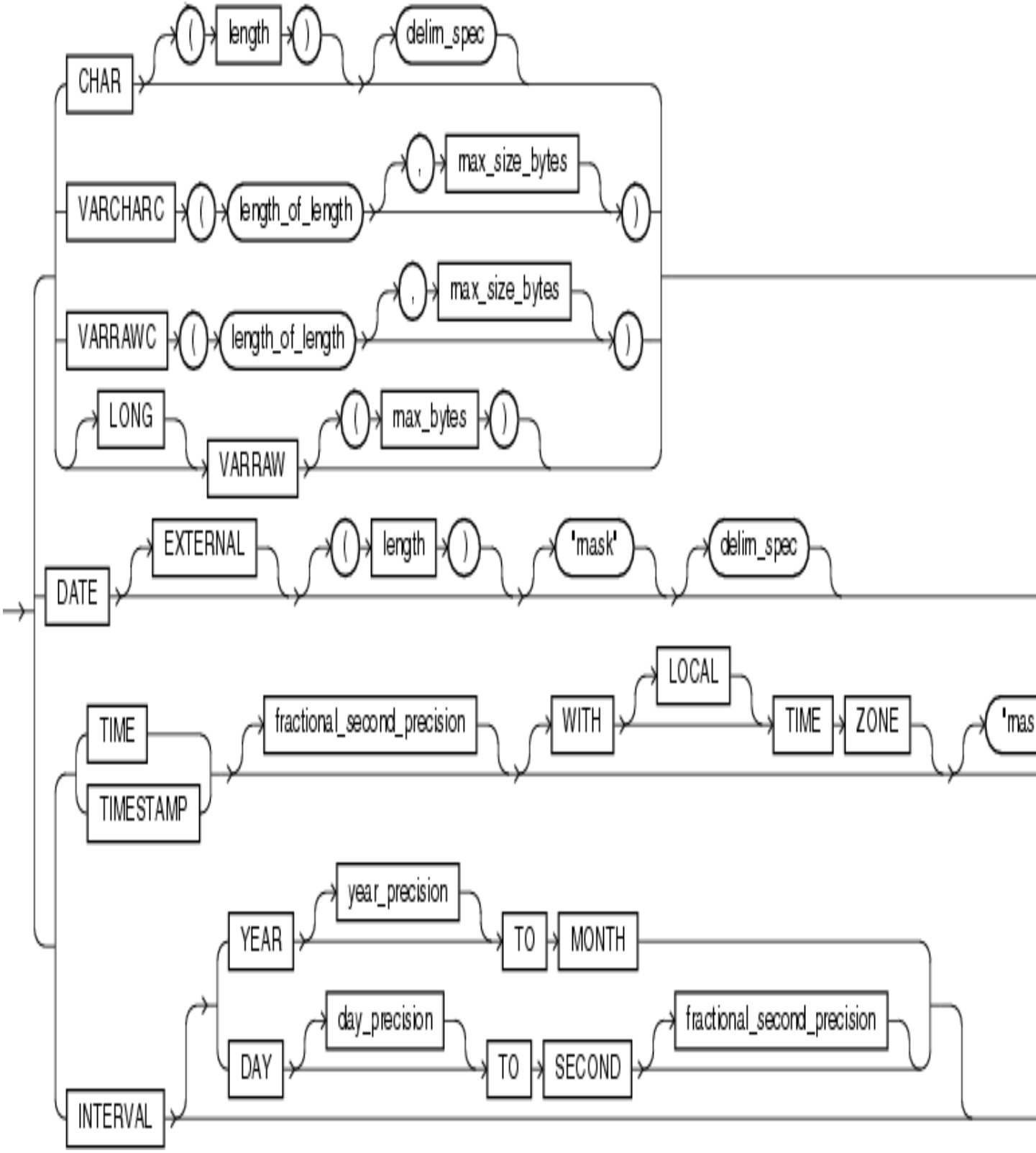
pos_spec



datatype_spec



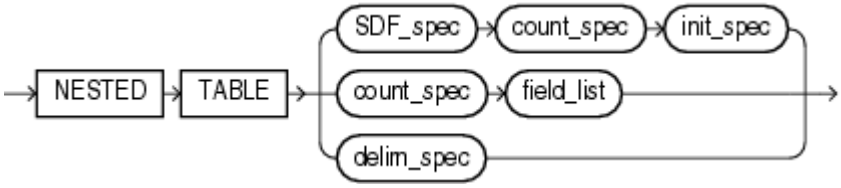
datatype_spec_cont



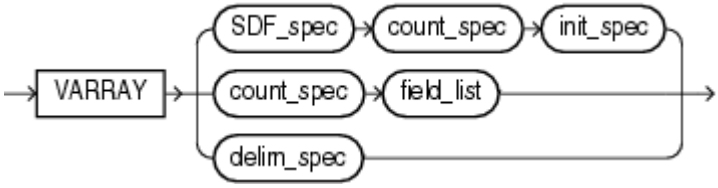
col_obj_fld_spec

collection_fld_spec

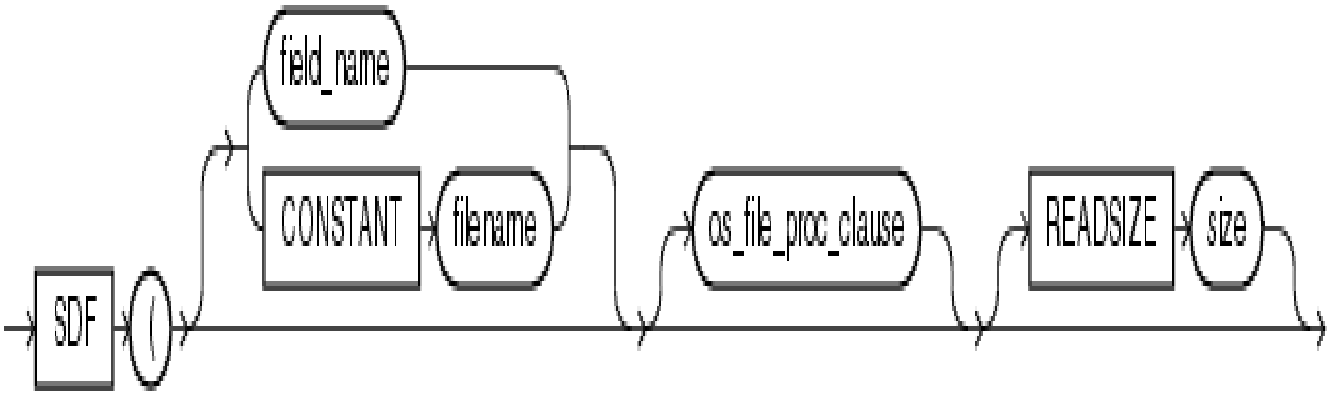
nested_table_spec

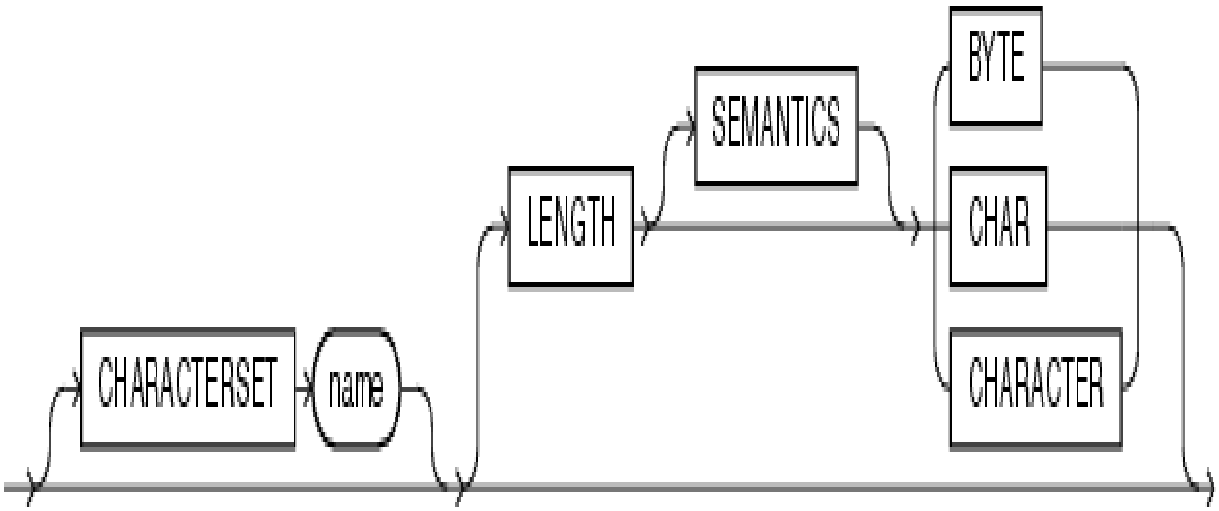


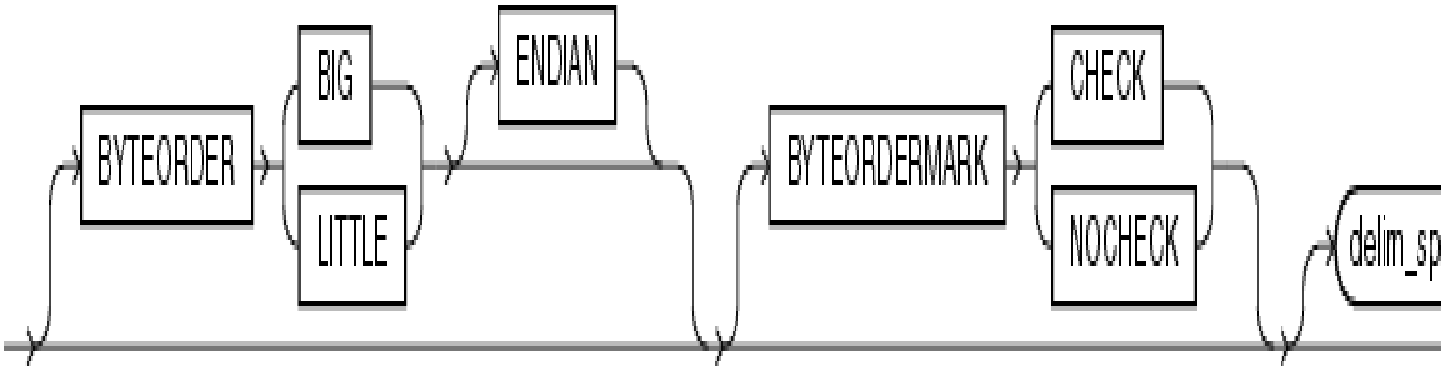
varray_spec



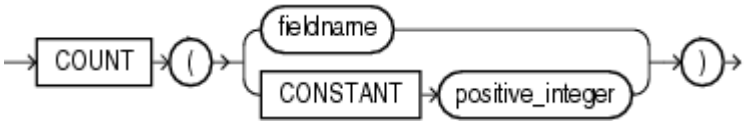
sdf_spec


SDF
(
field_name
CONSTANT
filename
os_file_proc_clause
READSIZE
size


CHARACTERSET
name
LENGTH
SEMANTICS
BYTE
CHAR
CHARACTER


BYTEORDER
BIG
LITTLE
ENDIAN
BYTEORDERMARK
CHECK
NOCHECK
delim_sp

count_spec


COUNT
(
fieldname
CONSTANT
positive_integer
)

B SQL*Loader、エクスポートおよびインポートのInstant Client

Oracle Instant Clientを使用すると、標準のOracleクライアントをインストールしたりOracleホームを使用せずにアプリケーションを実行できます。

次の内容について説明します。

- ツールInstant Clientとは
  ツールInstant Clientパッケージは、Oracle Call Interface (OCI) Instant Client対応のプラットフォームで使用できます。
- インストールするInstant Clientの選択
  ツールInstant Clientのツール・パッケージをインストールする前に、まず基本Instant ClientまたはInstant Client Lightを選択する必要があります。
- OTNからのダウンロードによるツールInstant Clientのインストール
  Linux用のOTNダウンロードはRPMパッケージです。UNIXおよびWindows用のOTNダウンロードは、zipファイルです。
- 12c Clientリリース・メディアからのツールInstant Clientのインストール
  ツールInstant Clientパッケージは Clientリリース・メディアからインストールできます。
- ツールInstant Clientパッケージの構成
  ツールInstant Clientパッケージ実行可能ファイルは、バージョンが一致するOCI Instant Clientとともにのみ使用する必要があります。環境変数ORACLE_HOMEまたはORACLE_SIDを設定する必要はありません。
- ツールInstant Clientパッケージによるデータベースへの接続
  ツールInstant Clientパッケージをインストールして構成した後、データベースに接続できます。
- Instant Clientのアンインストール
  ツールInstant Clientパッケージは、OCI Instant Clientとは別に削除できます。

B.1 ツールInstant Clientとは

ツールInstant Clientパッケージは、Oracle Call Interface (OCI) Instant Clientがサポートされているプラットフォームで使用できます。

ツール・パッケージには、SQL*Loader、Oracle Data Pump Export、Oracle Data Pump Import、オリジナルの(クラシック)エクスポート、オリジナルの(クラシック)インポートなどの複数のコマンドライン・ユーティリティが含まれています。Instant Clientのインストールはスタンドアロンで、製品のコマンドライン・バージョンのすべての機能が備えられています。Instant Clientは既存のリモートOracle Databaseに接続しますが、独自のデータベースはありません。簡単にインストールでき、製品のコマンドライン・バージョンの使用に必要なOracle Database Clientの完全なインストールと比較すると、非常に少ないディスク領域を使用します。

ツールInstant Clientの使用に必要なステップの概要

ツールInstant Clientを使用するには、2つのパッケージが必要です。

- ツールInstant Clientパッケージ
- 基本OCI Instant ClientパッケージまたはOCI Instant Client Lightパッケージ。

ツールInstant Clientを使用するために必要な基本ステップは次のとおりです。これらの各ステップの詳細は、この付録で説明します。

1. 使用するOCIパッケージ(基本またはLight)を選択し、Instant Clientファイルをインストールするディレクトリも選択します。
2. インストールしたOracleインスタンスから、ツールInstant Clientパッケージおよび選択したOCI Instant Clientパッケージをコピーするか、またはOTNからダウンロードします。
3. ツールInstant ClientパッケージおよびOCIパッケージをインストール(解凍)します。新しいディレクトリ`instantclient_12_2`がインストールの一部として作成されます。
4. Instant Clientを構成します。
5. 実行するユーティリティでリモート・インスタンスに接続します。

ツール・パッケージとOCIパッケージの両方がバージョン12.2.0.0.0以上のOracle Databaseであり、両方のバージョンが同じである必要があります。

OCI Instant Clientの詳細は、*Oracle Call Interface*プログラマーズ・ガイドを参照してください。

関連トピック

- Oracle Instant Clientについて

親トピック: SQL*Loader、エクスポートおよびインポートのInstant Client

B.2 インストールするInstant Clientの選択

ツールInstant Clientのツール・パッケージをインストールする前に、まず基本Instant ClientまたはInstant Client Lightを選択する必要があります。

基本Instant Client

ツールInstant Clientパッケージは、基本Instant Clientとともに使用する場合、Oracle Databaseでサポートされている`NLS_LANG`設定で動作します。Oracle Databaseで使用可能なすべての文字セットおよび言語設定をサポートしています。

Instant Client Light

Instant ClientのInstant Client Light (English)バージョンでは、クライアントのインストールに必要なディスク領域が削減されています。英語以外の言語のエラー・メッセージ・ファイルを削除し、サポートされる文字セット定義を約250の中からわずかな数のみ残すことで、ライブラリのサイズが削減されました。

Instant Client Lightは、`US7ASCII`、`WE8DEC`、`WE8ISO8859P1`、`WE8MSWIN1252`またはUnicode文字セットのいずれかを使用するアプリケーション向けです。`NLS_LANG`設定の`LANGUAGE`および`TERRITORY`フィールドに制限がないため、Instant Client Lightはあらゆる言語および地域設定で動作します。Instant Client Lightでは英語のエラー・メッセージのみ提供されているため、`NLS_LANG`が`AMERICAN`以外の言語に設定されている場合でも、Net接続エラーなどのクライアント側で生成されたエラー・メッセージは常に英語でレポートされます。データベース・インスタンスのOracleホームに適切な翻訳済メッセージ・ファイルがインストールされている場合、SQL文の構文エラーなどのデータベース側で生成されたエラー・メッセージは、選択した言語となります。

親トピック: SQL*Loader、エクスポートおよびインポートのInstant Client

B.3 OTNからのダウンロードによるツールInstant Clientのインストール

Linux用のOTNダウンロードは、RPMパッケージです。UNIXおよびWindows用のOTNダウンロードは、zipファイルです。

Instant Clientパッケージは、Oracleホームにはインストールしないでください。

Linux RPMパッケージからのInstant Clientのインストール

1. OTNのInstant Clientのページ(http://www.oracle.com/technetwork/database/database-technologies/instant-client/overview/index.html)から、ツールInstant ClientパッケージおよびOCIパッケージを含むRPMパッケージをダウンロードします。両方のパッケージがバージョン12.2.0.0.0以上で、かつ両方のバージョンが同じである必要があります。
2. RPMパッケージを初期インストールするには`rpm -i`、新しいバージョンのパッケージにアップグレードするには`rpm -u`を使用します。OCIパッケージを先にインストールします。
3. Instant Clientを構成します。「ツールInstant Clientパッケージの構成」を参照してください。

UNIXまたはWindows zipファイルからのInstant Clientのインストール

1. OTNのInstant Clientのページ(http://www.oracle.com/technetwork/database/database-technologies/instant-client/overview/index.html)から、ツールInstant ClientパッケージおよびOCIパッケージを含むRPMパッケージをダウンロードします。両方のパッケージがバージョン12.2.0.0.0以上で、かつ両方のバージョンが同じである必要があります。
2. 新規ディレクトリ(UNIXでは`//home/instantclient12_2`、Windowsでは`c:¥instantclient12_2`など)を作成します。
3. 2つのパッケージを新規ディレクトリに解凍します。OCIパッケージを先にインストールします。
4. Instant Clientを構成します。「ツールInstant Clientパッケージの構成」を参照してください。

ツールInstant Clientに必要なファイルのリスト

次の表に、ツールInstant Clientパッケージの必要なファイルを示します。また、いずれかのOCIパッケージ(基本またはLight)のファイルも必要です。インストールされるその他のファイルでこのリストに含まれていないファイルは、無視するか、ディスク領域を節約するために削除できます。

OCI Instant Clientの必要なファイルの詳細は、『Oracle Call Interfaceプログラマーズ・ガイド』を参照してください。

表B-1 ツール・パッケージ内のInstant Clientファイル

Linuxおよび UNIX	Windows	説明
exp	exp.exe	オリジナルの(クラシック)エクスポート実行可能ファイル
expdp	expdp.exe	データ・ポンプ・エクスポート実行可能ファイル
imp	imp.exe	オリジナルの(クラシック)インポート実行可能ファイル
impdp	impdp.exe	データ・ポンプ・インポート実行可能ファイル
libnfsodm12.	該当なし	Oracle Disk Manager (ODM)を使用するために SQL*Loader Instant Client で

Linuxおよび UNIX	Windows	説明
so		使用される共有ライブラリ。
sqlldr	sqlldr.exe	SQL*Loader 実行可能ファイル
TOOLS_README	該当なし	ツール Instant Client パッケージの Readme
wrc	wrc.exe	ツール Instant Client パッケージには、この付録で説明したもの以外のツールが含まれています。wrc ツールは、Oracle Database の再実行機能用のワークロードの再実行クライアント(wrc)です。wrc ツールは完全性のためにここに示されていますが、この付録の情報には含まれていません。

親トピック: SQL*Loader、エクスポートおよびインポートのInstant Client

B.4 12c Clientリリース・メディアからのツールInstant Clientのインストール

ツールInstant Clientパッケージは、Clientリリース・メディアからインストールできます。

1. Oracle Database 12c Clientリリース・メディアでインストーラを実行し、管理者オプションを選択します。
2. 新規ディレクトリ(UNIXおよびLinuxでは`/home/instantclient12_2`、Windowsでは`c:¥instantclient12_2`など)を作成します。
3. ツールInstant ClientパッケージおよびOCI Instant Clientファイルを新規ディレクトリにコピーします。すべてのファイルを同じOracleホームからコピーする必要があります。
4. ツールInstant Clientパッケージを構成します。「ツールInstant Clientパッケージの構成」を参照してください。

UNIXまたはLinuxでのツールInstant Clientのインストール

UNIXおよびLinuxで基本Instant Clientパッケージを使用してツールInstant Clientをインストールするには、次のファイルをコピーします。

```
$ORACLE_HOME/instantclient/libociei.so
$ORACLE_HOME/lib/libnfsodm12.so
$ORACLE_HOME/bin/exp
$ORACLE_HOME/bin/imp
$ORACLE_HOME/bin/expdp
$ORACLE_HOME/bin/impdp
$ORACLE_HOME/bin/sqlldr
```

UNIXおよびLinuxでInstant Client Lightパッケージを使用してツールInstant Clientをインストールするには、次のファイルをコピーします。

```
$ORACLE_HOME/instantclient/light/libociicus.so
$ORACLE_HOME/lib/libnfsodm12.so
$ORACLE_HOME/bin/exp
$ORACLE_HOME/bin/imp
$ORACLE_HOME/bin/expdp
$ORACLE_HOME/bin/impdp
$ORACLE_HOME/bin/sqlldr
```

WindowsでのツールInstant Clientのインストール

Windowsで基本Instant Clientパッケージを使用してツールInstant Clientをインストールするには、次のファイルをコピーします。

```
%ORACLE_HOME%¥instantclient¥oraociei12.dll
%ORACLE_HOME%¥bin¥exp.exe
%ORACLE_HOME%¥bin¥imp.exe
%ORACLE_HOME%¥bin¥expdp.exe
%ORACLE_HOME%¥bin¥impdp.exe
%ORACLE_HOME%¥bin¥sqlldr.exe
```

WindowsでInstant Client Lightパッケージを使用してツールInstant Clientをインストールするには、次のファイルをコピーします。

```
ORACLE_HOME%¥instantclient¥light¥oraociicus12.dll
%ORACLE_HOME%¥bin¥exp.exe
%ORACLE_HOME%¥bin¥imp.exe
%ORACLE_HOME%¥bin¥expdp.exe
%ORACLE_HOME%¥bin¥impdp.exe
%ORACLE_HOME%¥bin¥sqlldr.exe
```

親トピック: SQL*Loader、エクスポートおよびインポートのInstant Client

B.5 ツールInstant Clientパッケージの構成

ツールInstant Clientパッケージ実行可能ファイルは、バージョンが一致するOCI Instant Clientとともにのみ使用する必要があります。環境変数ORACLE_HOMEまたはORACLE_SIDを設定する必要はありません。

Linuxでの(RPMSからの)ツールInstant Clientパッケージの構成

OTNからダウンロードしたRPMは、/usrファイル・システム内のOracle固有のサブディレクトリにインストールされます。サブディレクトリ構造によって、複数のバージョンのInstant Clientを使用できます。

1. Instant Clientライブラリを含むディレクトリの名前をLD_LIBRARY_PATHに追加します。その他のOracleディレクトリは削除します。

   たとえば、BourneまたはKornシェルでLD_LIBRARY_PATHを設定するには、次の構文を使用します。

   ```
   LD_LIBRARY_PATH=/usr/lib/oracle/18.1/client/lib:${LD_LIBRARY_PATH}
   export LD_LIBRARY_PATH
   ```

   または、CシェルでLD_LIBRARY_PATHを設定するには、次の構文を使用します。

   ```
   % setenv LD_LIBRARY_PATH /usr/lib/oracle/18.1/client/lib:$LD_LIBRARY_PATH
   ```

2. たとえば、RPMからインストールしたTools実行可能ファイルが、PATHの先頭の実行可能ファイルであることを確認します。たとえば、このことをテストするためにwhich sqlldrと入力すると、/usr/bin/sqlldrが返されます。そうでない場合は、他のOracleディレクトリをPATHから削除するか、PATH内の他のTools実行可能ファイルの前に/usr/binを入力するか、あるいは絶対パスまたは相対パスを使用してツールInstant Clientを起動します。

   たとえば、bashシェルでPATHを設定するには、次のように指定します。

   ```
   PATH=/usr/bin:${PATH}
   export PATH
   ```

3. ロケールに必要なOracleグローバリゼーション変数を設定します。変数を設定しない場合は、デフォルトのロケールが指定されたものとみなされます。

   ```
   NLS_LANG=AMERICAN_AMERICA.UTF8
   ```

```
export NLS_LANG
```

LinuxおよびUNIXでの(Clientメディアまたはzipファイルからの)ツールInstant Clientパッケージの構成

1. Instant Clientファイルを含むディレクトリの名前を、適切な共有ライブラリ・パスLD_LIBRARY_PATH、LIBPATHまたはSHLIB_PATHに追加します。その他のOracleディレクトリは削除します。

   たとえば、BourneシェルまたはKornシェルでSolarisに設定するには、次のように指定します。

   ```
   LD_LIBRARY_PATH=/home/instantclient12_2:${LD_LIBRARY_PATH}
   export LD_LIBRARY_PATH
   ```

2. Instant Clientファイルを含むディレクトリを環境変数PATHに追加します。設定されていない場合は、絶対パスまたは相対パスを使用して、ツール・パッケージで提供されるユーティリティを起動する必要があります。その他のOracleディレクトリはPATHから削除します。たとえば:

   ```
   PATH=/home/instantclient12_2:${PATH}
   export PATH
   ```

3. ロケールに必要なOracleグローバリゼーション変数を設定します。変数を設定しない場合は、デフォルトのロケールが指定されたものとみなされます。たとえば:

   ```
   NLS_LANG=AMERICAN_AMERICA.UTF8
   export NLS_LANG
   ```

WindowsでのツールInstant Clientパッケージの構成

環境は、WindowsコマンドプロンプトでSETコマンドを使用して構成するか、またはシステム・プロパティの環境変数を設定して永続的にできます。

たとえば、Windows 2000でシステム・プロパティを使用して環境変数を設定するには、「コントロール パネル」から「システム」を開き、「詳細設定」タブをクリックしてから、「環境変数」をクリックします。

1. Instant Clientファイルを含むディレクトリをシステム環境変数PATHに追加します。その他のOracleディレクトリはPATHから削除します。

   たとえば、PATHの先頭にc:¥instantclient12_2を追加します。

2. ロケールに必要なOracleグローバリゼーション変数を設定します。変数を設定しない場合は、デフォルトのロケールが指定されたものとみなされます。

   たとえば、日本語環境用のNLS_LANGを設定するには、JAPANESE_JAPAN.JA16EUCに設定されたユーザー環境変数NLS_LANGを作成します。

親トピック: SQL*Loader、エクスポートおよびインポートのInstant Client

B.6 ツールInstant Clientパッケージによるデータベースへの接続

ツールInstant Clientパッケージをインストールして構成した後、データベースに接続できます。

ツールInstant Clientで提供されるユーティリティは、すべてのデータベース・サーバーから常にリモートの状態になります。サーバーでOracleインスタンスが稼働中で、かつTNSリスナーが実行中であることを前提とします。データ・ポンプ・エクスポートおよびインポートの場合、ダンプ・ファイルはリモート・サーバーに存在します。サーバー上のOracleディレクトリ・オブジェクトが存在し、適切な権限が割り当てられている必要があります。

データベースに接続するには、Oracle Net接続識別子を使用して、データベースを指定する必要があります。次の情報では

SQL*Loader (sqlldr)ユーティリティを使用しますが、この情報はツールInstant Clientパッケージで提供される他のユーティリティにも適用されます。

簡単な接続識別子を使用して、mymachineで実行されているMYDBデータベース内のHRスキーマに接続する例は、次のとおりです。

```
sqlldr hr/your_password@¥"//mymachine.mydomain:port/MYDB¥"
```

また、ネット・サービス名も使用できます。

```
sqlldr hr/your_password@MYDB
```

ネット・サービス名は、LDAPなどの様々な場所に格納できます。Oracle Database 12cの新機能を利用するには、LDAPを使用することをお薦めします

ローカルのOracle Net tnsnames.oraファイル内に構成されたネット・サービス名を使用する場合は、tnsnames.oraファイルを含むディレクトリに環境変数TNS_ADMINを設定します。たとえば、UNIXで、/home/user1内にあるtnsnames.oraファイルによってネット・サービス名MYDB2が定義されている場合は、次のように設定します。

```
TNS_ADMIN=/home/user1
export TNS_ADMIN
sqlldr hr@MYDB2
```

TNS_ADMINが設定されていない場合は、オペレーティング・システム依存のディレクトリ内でtnsnames.oraが検索されます。この検索パスにはnetwork/admin/tnsnames.oraも含まれ、ORACLE_HOME環境変数で指定されているディレクトリも検索されます。環境変数ORACLE_HOMEは、このためにのみ、SQL*Loader Instant Clientに設定します。Instant Clientアプリケーションの実行時にORACLE_HOMEを設定する場合は、既存のディレクトリに設定する必要があります。

この例は、環境変数ORACLE_HOMEが設定されていて、$ORACLE_HOME/network/admin/tnsnames.oraまたはORACLE_HOME¥network¥admin¥tnsnames.oraファイルによってネット・サービス名MYDB3が定義されていることを前提としています。

```
sqlldr hr@MYDB3
```

環境変数TWO_TASK (UNIXの場合)またはLOCAL (Windowsの場合)には接続識別子を設定できます。これによって、SQL*LoaderまたはSQL*Loader Instant Clientで接続が行われるたびに接続識別子を明示的に入力する必要がなくなります。このUNIXの例では、MYDB4という名前のデータベースに接続します。

```
TNS_ADMIN=/home/user1
export TNS_ADMIN
TWO_TASK=MYDB4
export TWO_TASK
sqlldr hr
```

Windowsでは、TNS_ADMINおよびLOCALは、システム・プロパティで設定できます。

親トピック: SQL*Loader、エクスポートおよびインポートのInstant Client

B.7 Instant Clientのアンインストール

ツールInstant Clientパッケージは、OCI Instant Clientとは別に削除できます。

ツールInstant Clientパッケージをアンインストールした後も、残されたOCI Instant Clientライブラリによって、カスタム作成されたOCIプログラムまたはサード・パーティのデータベース・ユーティリティでデータベースに接続できます。

ツールInstant Clientのアンインストール

1. LinuxでのRPMパッケージからのインストールの場合は、ツールInstant Clientパッケージに対してのみ`rpm -e`を使用します。

   または

   UNIXおよびWindowsでのインストールの場合、およびLinuxでのClientリリース・メディアからのインストールの場合は、ツールInstant Clientに固有のすべてのファイルを手動で削除します。削除するファイルは、インストール時に指定したInstant Clientディレクトリにあります。Oracleホームのファイルは削除しないでください。

   環境変数を再設定し、必要に応じて`tnsnames.ora`を削除します。

Instant Clientの完全なアンインストール

1. LinuxでのRPMパッケージからのインストールの場合は、`rpm -qa`を使用してツールInstant Clientおよび基本OCIパッケージの名前を検索し、`rpm -e`を実行してそれらを削除します。

   OR

   UNIXおよびWindowsでのインストールの場合、およびLinuxでのClientリリース・メディアからのインストールの場合は、Tools実行可能ファイルおよびOracleライブラリを含むディレクトリを手動で削除します。

2. `PATH`、`LD_LIBRARY_PATH`、`TNS_ADMIN`などの環境変数をリセットします。

3. 必要に応じて、`tnsnames.ora`を削除します。

親トピック: SQL*Loader、エクスポートおよびインポートのInstant Client

索引

記号 数値 A B C D E F G H I J K L M N O P Q R S T U V W X

記号

- * parameter*パラメータ
  - field_definitions句 15.4.5.2
  - pos_spec句 15.4.5.2

数字

- 1291 ORA-01291: ログ・ファイルがありません 24.7.3

A

- ABORT_STEPパラメータ
  - データ・ポンプ・エクスポート・ユーティリティ 2.4.2
  - データ・ポンプ・インポート・ユーティリティ 3.4.2
- ACCESS_METHODパラメータ
  - データ・ポンプ・エクスポート・ユーティリティ 2.4.3
  - データ・ポンプ・インポート・ユーティリティ 3.4.3
- ACCESS PARAMETERS句
  - 特殊文字 17.1, 19.3.1
- アクセス権限
  - エクスポートおよびインポート 26.2.4
- ADD_FILEパラメータ
  - データ・ポンプ・エクスポート・ユーティリティ
    - 対話方式コマンド・モード 2.5.2
- ADR
  - 「自動診断リポジトリ」を参照
- ADRベース
  - ADRCIユーティリティ内 21.2
- ADRCI
  - トラブルシューティング 21.10
- ADRCIヘルプ 21.3.2
- ADRCIユーティリティ 21
  - ADRベース 21.2
  - ADRホーム 21.2
  - バッチ・モード 21.3.3
  - コマンド 21.9
  - ヘルプの利用 21.3

    - ホームパス 21.2
    - 対話方式モード 21.3.1
    - 起動 21.3
- ADRホーム
    - ADRCIユーティリティ内 21.2
- アドバンスト・キューイング
    - アドバンスト・キュー表のエクスポート 26.14.10
    - アドバンスト・キュー表のインポート 27.16.10
- 別名
    - ディレクトリ
        - エクスポート 26.14.5
        - インポート 27.16.5
- アナライザ統計 27.24
- REDOログ・ファイルの分析 24
- ANYDATAタイプ
    - SQL文字列を使用してロード 10.14.6
- APPENDパラメータ
    - SQL*Loaderユーティリティ 9.19.2
- 表への追加
    - SQL*Loader 9.18.2.2.1
- アーカイブされたLOB
    - エクスポートに対する制限事項 2.4.51
- アーカイブ・ログ
    - データ・ポンプ・ジョブでの無効化 3.4.52
    - データ・ポンプ・ジョブでの有効化 3.4.52
- アーカイブ
    - 使用禁止
        - ダイレクト・パス・ロードの影響 12.6.5.1
- 配列のオーバーフロー 17.4.7
- 配列
    - 挿入後のコミット 27.7.2
- アトミックNULL 11.1.5.2
- 既存のジョブへの接続
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.4
- ATTACHパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.4
    - データ・ポンプ・インポート・ユーティリティ 3.4.4
- 属性
    - NULL 11.1.5.1
- 属性値コンストラクタ
    - 上書き 11.1.6
- データ・ポンプ・ジョブの監査
    - 統合監査

        - データ・ポンプ 1.12
- ダイレクト・パス・ロードの監査
    - SQL*Loaderの使用時 12.5.9
- 自動診断リポジトリ 21.2

B

- バックスラッシュ・エスケープ文字 9.4.3.2
- バックアップ
    - 削除されたスナップショットのリストア
        - インポート 27.18.2.1
- BADFILEパラメータ
    - SQL*Loaderユーティリティ 9.11.1
- 不良ファイル
    - SQL*Loaderへの指定 9.11.1
- BADパラメータ
    - SQL*Loaderコマンドライン 8.2.1
    - SQL*Loaderエクスプレス・モード 13.3.1
- BEGINDATAパラメータ
    - SQL*Loader制御ファイル 9.9
- BFILE
    - オリジナルのエクスポート 26.14.6
    - オリジナルのインポート 27.16.5
    - SQL*Loaderでのロード 11.4
- ビッグ・エンディアン・データ
    - 外部表 15.3.11
- BINARY_DOUBLE
    - 浮動小数点数 15.4.6.5
- BINARY_FLOAT
    - 浮動小数点数 15.4.6.5
- バイナリのオーバーフロー 17.4.7
- バインド配列
    - SQL*Loaderに対するサイズの指定 9.21.4
    - SQL*Loaderのメモリー所要量の最小化 9.21.5
    - 最小要件 9.21.1
    - 複数のSQL*LoaderのINTO TABLE文のサイズ 9.21.6
    - 最大サイズの指定 8.2.2
    - 行数の指定 8.2.31
    - SQL*Loaderパフォーマンスとの関連 9.21.2
- BINDSIZEパラメータ
    - SQL*Loaderコマンドライン 8.2.2, 9.21.3
- 空白
    - 空白のフィールドのロード 10.10

    - フィールド比較用のSQL*Loader BLANKSパラメータ 10.5.2
    - 後続 10.4.6.4
    - 切捨て 10.11
        - 外部表 8.2.38, 13.3.21, 15.4.2
    - 空白 10.11
- BLANKSパラメータ
    - SQL*Loaderユーティリティ 10.5.2
- BLOB
    - SQL*Loaderでのロード 11.4
- バッファ・キャッシュのサイズ
    - GoldenGateレプリケーションに関連するデータ・ポンプ操作 5.3.1
- BUFFERパラメータ
    - エクスポート・ユーティリティ 26.5.1
    - インポート・ユーティリティ 27.7.1
- バッファ
    - エクスポートの計算 26.5.1
    - SQL*LoaderのBINDSIZEパラメータを使用した指定 9.21.4
- バイト順序 10.9.1
    - ビッグ・エンディアン 10.9.1
    - リトル・エンディアン 10.9.1
    - SQL*Loaderの制御ファイルでの指定 10.9.2
- BYTEORDERMARKパラメータ
    - SQL*Loaderユーティリティ 10.9.4
- バイト順序マーク 10.9.3
    - 優先順位
        - 第1プライマリ・データ・ファイル 10.9.3
        - LOBFILEおよびSDF 10.9.3
    - 確認の抑止 10.9.4
- BYTEORDERパラメータ
    - SQL*Loaderユーティリティ 10.9.2

C

- キャッシュされた順序番号
    - エクスポート 26.14.1
- catalog.sqlスクリプト
    - エクスポートおよびインポートのためのデータベースの準備 26.2.2, 27.2.1
- catexp.sql スクリプト
    - エクスポートおよびインポートのためのデータベースの準備 26.2.2, 27.2.1
- CDB
    - Oracle Data Pumpのサポート 1.3
    - データ・ポンプを使用したデータの移動 1.3.2
- データベースIDの変更 23.5.1

- データベース名の変更 23.5.3
- 文字フィールド
    - デリミタとSQL*Loader 10.4.3, 10.4.6
    - SQL*Loaderに対する長さの指定 10.4.8
    - SQL*Loaderデータ型 10.4.3
- 文字長セマンティクス 9.15.5.4
- 文字のオーバーフロー 17.4.7
- CHARACTERSETパラメータ
    - SQL*Loaderエクスプレス・モード 13.3.2
    - SQL*Loaderユーティリティ 9.15.5.2, 13.3.2
- 文字セット
    - 変換
        - エクスポートおよびインポート中 26.12.1, 27.14.1
    - 8ビットから7ビットへの変換
        - エクスポートおよびインポート 26.12.3, 27.14.3
    - 外部表の識別 15.3.6
    - マルチバイト
        - エクスポートおよびインポート 26.12.4
        - SQL*Loader 9.15.1
    - シングルバイト
        - エクスポートおよびインポート 26.12.3, 27.14.3
    - SQL*Loader制御ファイル 9.15.5.3
    - SQL*Loaderでの変換 9.15
    - Unicode 9.15.2
- 文字列
    - 外部表
        - バイトまたは文字の指定 15.3.13
    - SQL*Loader 10.5.3
- CHARデータ型
    - デリミタ付き形式およびSQL*Loader 10.4.6
- CHECK制約
    - 使用禁止の変更 12.9.1.2
- CLOB
    - SQL*Loaderでのロード 11.4
- CLUSTERパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.5
    - データ・ポンプ・インポート・ユーティリティ 3.4.5
- コレクション 7.10
    - ロード 11.6
- SQL*Loaderでサポートされるコレクション型 7.10.2
- 配列行の列
    - 行数の指定 12.6.6
- COLUMNARRAYROWSパラメータ

    - SQL*Loaderコマンドライン 8.2.3
- 列マッピング 17.4.1
- 列オブジェクト
    - ロード11.1
        - ユーザー定義コンストラクタ 11.1.6
- 列
    - LONGデータ型のエクスポート 26.14.2
    - REF列のロード 11.3
    - 命名
        - SQL*Loader 10.3
    - オブジェクト
        - ネストした列オブジェクトのロード 11.1.3
        - ストリーム・レコード形式 11.1.1
        - 可変レコード形式 11.1.2
    - インポート実行前の順序変更 27.3.1
    - SQL*Loaderを使用した定数値への設定 10.15.2
    - SQL*Loaderを使用した式の値への設定 10.15.3
    - SQL*Loaderを使用した一意の順序番号への設定 10.15.6
    - SQL*Loaderを使用したNULLへの設定 10.15.2.1
    - SQL*Loaderを使用した現在の日付への設定 10.15.5
    - SQL*Loaderを使用したデータ・ファイルのレコード番号への設定 10.15.4
    - 指定
        - SQL*Loader 10.3
    - PIECEDとしての指定
        - SQL*Loader 12.5.8
    - using SQL*Loader 10.15.3
- com.oracle.bigdata.colmap 17.4.1
- com.oracle.bigdata.erroropt 17.4.2
- com.oracle.bigdata.fields 17.4.3
- com.oracle.bigdata.fileformat 17.4.4
- com.oracle.bigdata.log.exec 17.4.5
- com.oracle.bigdata.log.qc 17.4.6
- com.oracle.bigdata.overflow 17.4.7
- com.oracle.bigdata.rowformat 17.4.8
- com.oracle.bigdata.tablename 17.4.9
- コメント
    - エクスポートおよびインポートのパラメータ・ファイル 26.3.3, 27.5.2
    - 外部表 15.2, 16.2.1
- COMMITパラメータ
    - インポート・ユーティリティ 27.7.2
- COMPILEパラメータ
    - インポート・ユーティリティ 27.7.3
- 完了メッセージ

    - エクスポート 26.7.4
    - インポート 26.7.4
- 圧縮
    - データ・ポンプ・ジョブでのアルゴリズムの指定 2.4.7
    - データ・ポンプ・ジョブでの表の指定 3.4.52
    - 外部表へのレベルの指定 16.2.2
- COMPRESSION_ALGORITHMパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.7
- 圧縮アルゴリズム
    - データ・ポンプ・ジョブでの指定 2.4.7
- COMPRESSIONパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.6
- COMPRESSパラメータ
    - エクスポート・ユーティリティ 26.5.2
- CONCATENATEパラメータ
    - SQL*Loaderユーティリティ 9.17
- 従来型パスによる同時ロード 12.9.4
- 構成
    - LogMinerユーティリティ 24.2.1.1
- CONSISTENTパラメータ
    - エクスポート・ユーティリティ 26.5.3
        - ネストした表 26.5.3
        - パーティション表 26.5.3
- 統合
    - エクステント 26.5.2
- CONSTANTパラメータ
    - SQL*Loader 10.15.2
- 制約
    - 自動整合性およびSQL*Loader 12.9.2.2
    - ダイレクト・パス・ロード 12.9
    - 参照制約を使用禁止にする方法 27.3.2
    - 使用可能化
        - パラレル・ダイレクト・パス・ロード後 12.10.8
    - ダイレクト・ロードに対する施行 12.9.1.1
    - 失敗
        - インポート 27.10.1.1
    - ロード方法 12.4.9
- CONSTRAINTSパラメータ
    - エクスポート・ユーティリティ 26.5.4
    - インポート・ユーティリティ 27.7.4
- コンストラクタ
    - 属性値 11.1.6
        - 上書き 11.1.6

    - ユーザー定義 11.1.6
        - 列オブジェクトのロード 11.1.6
- CONTENTパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.8
    - データ・ポンプ・インポート・ユーティリティ 3.4.4
- CONTINUE_CLIENTパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ
        - 対話方式コマンド・モード 2.5.3
    - データ・ポンプ・インポート・ユーティリティ
        - 対話方式コマンド・モード 3.5.2
- CONTINUEIFパラメータ
    - SQL*Loaderユーティリティ 9.17
- 制御ファイル
    - 文字セット 9.15.5.3
    - データ定義言語の構文 9.1
    - データの指定 9.9
    - SQL*Loader廃棄ファイルの指定 9.12.1
    - SQL*Loader 9.1
- CONTROLパラメータ
    - SQL*Loaderコマンドライン 8.2.4
- 従来型パス・エクスポート
    - ダイレクト・パスとの比較 26.9
- 従来型パス・ロード
    - 中断された場合の動作 9.16.1
    - ダイレクト・パス・ロードとの比較 12.4.8
    - 同時 12.10.2
    - 単一パーティション 12.3
    - SQL*Loaderバインド配列 9.21.2
    - 使用する場合 12.3.2
- 文字セットの変換
    - エクスポートおよびインポート中 26.12.1, 27.14.1
    - 文字セットのソートの影響 26.12.1.1, 27.14.1.1
- データ変換
    - ダイレクト・パス・ロード時 12.4.2
- 入力文字の変換 9.15.5
- CREATE REPORTコマンド, ADRCIユーティリティ 21.9.1
- CREATE SESSION権限
    - エクスポート 26.2.4, 27.2.2
    - インポート 26.2.4, 27.2.2
- 作成
    - インシデント・パッケージ 21.8.2
    - 表
        - インポート実行前の手動操作 27.3.1

- CREDENTIALパラメータ
    - データ・ポンプ・インポート・ユーティリティ 3.4.7
- CSVパラメータ
    - SQL*Loaderエクスプレス・モード 13.3.3

D

- データ
    - 変換
        - ダイレクト・パス・ロード 12.4.2
    - デリミタで区切られたデータおよびSQL*Loader 10.4.6.2
    - SQL*Loaderへの異なる入力形式の区別 9.20
    - 異なる入力行オブジェクトのサブタイプの区別 9.20, 9.20.3
    - エクスポート 26.5.24
    - SQL*Loaderを使用した一意の値の生成 10.15.6
    - 制御ファイルへの組込み 9.9
    - SQL*Loader制御ファイルに組み込まれたデータのロード 10.15.1
    - セクションごとのロード
        - SQL*Loader 12.5.8
    - 複数表へのロード
        - SQL*Loader 9.20
    - SQL*Loaderのデリミタ付きデータの最大長 10.4.6.3
    - SQL*Loaderを使用したオペレーティング・システム間の移動 10.8
    - リカバリ
        - SQL*Loaderでのダイレクト・パス・ロード 12.5.6
    - ダイレクト・パス・ロード時の保存 12.5.5.1
    - 行の保存
        - SQL*Loader 12.6.4
    - 未ソート
        - SQL*Loader 12.6.3.3
    - SQL*Loaderのパフォーマンスのために最適化された値 10.15.1
- DATA_OPTIONSパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.9
    - データ・ポンプ・インポート・ユーティリティ 3.4.8
- データベースID (DBID)
    - 変更 23.5.1
- データベース識別子
    - 変更 23.5.1
- データベースの移行
    - パーティション化 26.17, 27.25
- データベース名(DBNAME)
    - 変更 23.5.3
- データベース・オブジェクト

    - LONG列のエクスポート 26.14.2
- データベース
    - データベースIDの変更 23.5.1
    - 名前の変更 23.5.3
    - 全体のエクスポート 26.5.11
    - 全体のインポート 27.7.12
    - エクスポートおよびインポートの権限 26.2.4, 27.2.2
    - 既存のデータ・ファイルの再利用
        - インポート 27.7.7
- データ・フィールド
    - SQL*Loaderデータ型の指定 10.3.2
- データファイル
    - インポート中の上書き防止 27.7.7
    - インポート時の再利用 27.7.7
    - 指定 8.2.5, 13.3.4
    - SQL*Loaderの形式の指定 9.10
- データ・ファイル
    - SQL*Loaderのバッファの指定 9.10
    - SQL*Loaderへの指定 9.7.1
- DATAFILESパラメータ
    - インポート・ユーティリティ 27.7.6
- DATAパラメータ
    - SQL*Loaderコマンドライン 8.2.5, 13.3.4
    - SQL*Loaderエクスプレス・モード 13.3.4
- DATAPUMP_EXP_FULL_DATABASEロール 1.4
- DATAPUMP_IMP_FULL_DATABASEロール 1.4
- データ・ポンプ・エクスポート・ユーティリティ
    - ABORT_STEPパラメータ 2.4.2
    - ACCESS_METHODパラメータ 2.4.3
    - 追加ダンプ・ファイルの追加 1.8.2
    - ATTACHパラメータ 2.4.4
    - CLUSTERパラメータ 2.4.5
    - コマンドライン・モード 2.4.1, 3.4
    - COMPRESSION_ALGORITHMパラメータ 2.4.7
    - COMPRESSIONパラメータ 2.4.6
    - CONTENTパラメータ 2.4.8
    - リソース消費の制御 5.2.1
    - DATA_OPTIONSパラメータ 2.4.9
    - DIRECTORYパラメータ 2.4.10
    - DUMPFILEパラメータ 2.4.11
    - ダンプ・ファイル・セット 2.1
    - ENCRYPTION_ALGORITHMパラメータ 2.4.14
    - ENCRYPTION_MODEパラメータ 2.4.15

- ENCRYPTION_PASSWORDパラメータ 2.4.16
- ENCRYPTION_PWD_PROMPTパラメータ 2.4.17
- SecureFilesの暗号化 2.4.13
- ENCRYPTIONパラメータ 2.4.13
- ESTIMATE_ONLYパラメータ 2.4.19
- ESTIMATEパラメータ 2.4.18
- EXCLUDEパラメータ 2.4.20
- オブジェクトの除外 2.4.20
- エクスポート・モード 2.2.2
- FILESIZEコマンド
    - 対話方式コマンド・モード 2.5.5
- FILESIZEパラメータ 2.4.21
- エクスポートされるデータのフィルタ処理
    - EXCLUDEパラメータの使用 2.4.20
    - INCLUDEパラメータの使用 2.4.26
- FLASHBACK_SCNパラメータ 2.4.22
- FLASHBACK_TIMEパラメータ 2.4.23
- FULLパラメータ 2.4.24
- HELPパラメータ
    - 対話方式コマンド・モード 2.5.6
- INCLUDEパラメータ 2.4.26
- 対話方式コマンド・モード 2.5
    - ADD_FILEパラメータ 2.5.2
    - CONTINUE_CLIENTパラメータ 2.5.3
    - EXIT_CLIENTパラメータ 2.5.4
    - FILESIZE 2.5.5
    - HELPパラメータ 2.5.6
    - KILL_JOBパラメータ 2.5.7
    - PARALLELパラメータ 2.5.8
    - START_JOBパラメータ 2.5.9
    - STATUSパラメータ 2.5.10, 3.5.8
    - STOP_JOBパラメータ 2.5.11, 3.5.9
- インタフェース 2.2.1
- 起動
    - SYSDBA 3.2
- JOB_NAMEパラメータ 2.4.27
- ジョブ名
    - 指定 2.4.27
- KEEP_MASTERパラメータ 2.4.28
- LOGFILEパラメータ 2.4.29
- LOGTIMEパラメータ 2.4.30
- METRICSパラメータ 2.4.31
- NETWORK_LINKパラメータ 2.4.32

- NOLOGFILEパラメータ 2.4.33
- PARALLEL_THRESHOLDパラメータ
  - コマンドライン・モード 2.4.35
- PARALLELパラメータ
  - コマンドライン・モード 2.4.34
  - 対話方式コマンド・モード 2.5.8
- PARFILEパラメータ 2.4.36
- QUERYパラメータ 2.4.37
- REMAP_DATAパラメータ 2.4.38
- REUSE_DUMPFILESパラメータ 2.4.39
- SAMPLEパラメータ 2.4.40
- SCHEMASパラメータ 2.4.41
- SecureFiles LOBに関する考慮点 1.10
- SERVICE_NAMEパラメータ 2.4.42
- SOURCE_EDITIONパラメータ 2.4.43
- ジョブ名の指定 2.4.27
- 起動
  - SYSDBA 2.2
- STATUSパラメータ 2.4.44
- 構文図 2.7
- TABLESPACESパラメータ 2.4.46
- TABLESパラメータ 2.4.45
- 透過的データ暗号化 2.4.16
- TRANSPORT_FULL_CHECKパラメータ 2.4.47
- TRANSPORT_TABLESPACESパラメータ 2.4.48
- TRANSPORTABLEパラメータ 2.4.49
- TTS_CLOSURE_CHECKパラメータ 2.4.50
- バージョニング 1.9
- VERSIONパラメータ 2.4.51
- VIEWS_AS_TABLESパラメータ 2.4.52

- データ・ポンプ・インポート・ユーティリティ
  - ABORT_STEPパラメータ 3.4.2
  - ACCESS_METHODパラメータ 3.4.3
  - 既存のジョブへの接続 3.4.4
  - ATTACHパラメータ 3.4.4
  - ソース・データ・ファイルの名前の変更 3.4.35
  - CLUSTERパラメータ 3.4.5
  - CONTENTパラメータ 3.4.6
  - リソース消費の制御 5.2.1
  - CREDENTIALパラメータ 3.4.7
  - DATA_OPTIONSパラメータ 3.4.8
  - DIRECTORYパラメータ 3.4.9
  - DUMPFILEパラメータ 3.4.10

- ENCRYPTION_PASSWORDパラメータ 3.4.12
- ENCRYPTION_PWD_PROMPTパラメータ 3.4.13
- ESTIMATEパラメータ 3.4.14
- ジョブ・サイズの見積り 3.4.14
- EXCLUDEパラメータ 3.4.15
- インポートされるデータをフィルタ処理
    - EXCLUDEパラメータの使用 3.4.15
    - INCLUDEパラメータの使用 3.4.20
- FLASHBACK_SCNパラメータ 3.4.16
- FLASHBACK_TIMEパラメータ 3.4.17
- 全体インポート・モード 3.2.2.2
- FULLパラメータ 3.4.18
- HELPパラメータ
    - コマンドライン・モード 3.4.19
    - 対話方式コマンド・モード 3.5.4
- INCLUDEパラメータ 3.4.20
- 対話方式コマンド・モード 3.5.1
    - CONTINUE_CLIENTパラメータ 3.5.2
    - EXIT_CLIENTパラメータ 3.5.3
    - HELPパラメータ 3.5.4
    - KILL_JOBパラメータ 3.5.5
    - PARALLELパラメータ 3.5.6
    - START_JOBパラメータ 3.5.7
    - STATUS 3.5.8
    - STOP_JOBパラメータ 3.5.9
- インタフェース 3.2.1
- JOB_NAMEパラメータ 3.4.21
- KEEP_MASTERパラメータ 3.4.22
- LOGTIMEパラメータ 3.4.24
- MASTER_ONLYパラメータ 3.4.25
- METRICSパラメータ 3.4.26
- NETWORK_LINKパラメータ 3.4.27
- NOLOGFILEパラメータ 3.4.28
- PARALLEL_THRESHOLDパラメータ
    - コマンドライン・モード 3.4.30
- PARALLELパラメータ
    - コマンドライン・モード 3.4.29
- PARFILEパラメータ 3.4.31
- PARTITION_OPTIONSパラメータ 3.4.32
- QUERYパラメータ 3.4.33
- REMAP_DATAFILEパラメータ 3.4.35
- REMAP_DATAパラメータ 3.4.34
- REMAP_SCHEMAパラメータ 3.4.37

    - REMAP_TABLEパラメータ 3.4.38
    - REMAP_TABLESPACEパラメータ 3.4.39
    - REUSE_DATAFILESパラメータ 3.4.48
    - スキーマ・モード 3.2.2.3
    - SCHEMASパラメータ 3.4.40
    - SERVICE_NAMEパラメータ 3.4.41
    - SKIP_UNUSABLE_INDEXESパラメータ 3.4.42
    - SOURCE_EDITIONパラメータ 3.4.43
    - ジョブ名の指定 3.4.21
    - 処理する資格証明オブジェクト名の指定 3.4.7
    - インポートするダンプ・ファイル・セットを指定 3.4.10
    - SQLFILEパラメータ 3.4.44
    - STATUSパラメータ 3.4.45
    - STREAMS_CONFIGURATIONパラメータ 3.4.46
    - 構文図 3.7
    - TABLE_EXISTS_ACTIONパラメータ 3.4.47
    - 表モード 3.2.2.4
    - 表領域モード 3.2.2.5
    - TABLESPACESパラメータ 3.4.50
    - TABLESパラメータ 3.4.49
    - TARGET_EDITIONパラメータ 3.4.51
    - TRANSFORMパラメータ 3.4.52
    - 透過的データ暗号化 3.4.12
    - TRANSPORT_FULL_CHECKパラメータ 3.4.54
    - TRANSPORT_TABLESPACESパラメータ 3.4.55
    - TRANSPORTABLEパラメータ 3.4.56
    - トランスポータブル表領域モード 3.2.2.6
    - バージョニング 1.9
    - VERSIONパラメータ 3.4.57
    - VIEWS_AS_TABLESパラメータ(ネットワーク・インポート) 3.4.58
- データ・ソース名 17.4.9
- データ型
    - BFILE
        - オリジナルのエクスポート 26.14.6
        - オリジナルのインポート 27.16.5
        - SQL*Loaderでのロード 11.4
    - BLOB
        - SQL*Loaderでのロード 11.4
    - CLOB
        - SQL*Loaderでのロード 11.4
    - SQL*Loaderの変換 10.4.4
    - 外部表フィールドの記述 15.4.6
    - SQL*Loaderに対する文字フィールド長の設定 10.4.8

    - DATE長の決定 10.4.8.3
    - 外部表の識別 15.4.4
    - ネイティブ
        - SQL*Loaderでの競合する長さ指定 10.4.3.10
    - NCLOB
        - SQL*Loaderでのロード 11.4
    - 非スカラー 11.1.5
    - SQL*Loaderでの指定 10.3.2
    - LogMinerユーティリティによるサポート 24.14.1
    - SQL*Loaderで使用される型 10.4
    - LogMinerユーティリティによりサポートされない 24.14.2
- データ型(HDFS) 17.4.3
- DATE_CACHEパラメータ
    - SQL*Loaderユーティリティ 8.2.6
- DATE_FORMATパラメータ
    - SQL*Loaderエクスプレス・モード 13.3.5
- 日付キャッシュ機能
    - DATE_CACHEパラメータ 8.2.6
    - 外部表 15.7
    - SQL*Loader 12.6.7
- DATEデータ型
    - デリミタ付き形式およびSQL*Loader 10.4.6
    - 長さの決定 10.4.8.3
    - マスク
        - SQL*Loader 10.4.8.3
- DBID (データベース識別子)
    - 変更 23.5.1
- DBMS_LOGMNR_D.ADD_LOGFILES PL/SQLプロシージャ
    - LogMinerユーティリティ 24.2.2
- DBMS_LOGMNR_D.BUILD PL/SQLプロシージャ
    - LogMinerユーティリティ 24.2.2
- DBMS_LOGMNR_D.END_LOGMNR PL/SQLプロシージャ
    - LogMinerユーティリティ 24.2.2
- DBMS_LOGMNR_D PL/SQLプロシージャ
    - LogMinerユーティリティ 24.2.2
- DBMS_LOGMNR.COLUMN_PRESENT関数 24.6.2
- DBMS_LOGMNR.MINE_VALUE関数 24.6.2
    - null値 24.6.2.1
- DBMS_LOGMNR.START_LOGMNR PL/SQLプロシージャ 24.5
    - 複数回のコール 24.9
    - COMMITTED_DATA_ONLYオプション 24.7.1
    - LogMinerユーティリティ 24.2.2
    - オプション 24.5

    - PRINT_PRETTY_SQLオプション 24.7.6
    - SKIP_CORRUPTIONオプション 24.7.2
- DBMS_LOGMNR.START_LOGMNRプロシージャ
    - ENDTIMEパラメータ 24.7.3
    - STARTTIMEパラメータ 24.7.3
- DBMS_LOGMNR PL/SQLプロシージャ
    - LogMinerユーティリティ 24.2.2
- PL/SQLパッケージDBMS_METADATA 25.3.1
- DBNAME
    - 変更 23.5.3
- DBNEWIDユーティリティ 23
    - データベースIDの変更 23.5.1
    - データベース名の変更 23.5.3
    - グローバル・データベース名への影響 23.3
    - 制限事項 23.6.2
    - 構文 23.6
    - データベースIDの変更のトラブルシューティング 23.5.4
- DBVERIFYユーティリティ
    - 出力 22.1.3
    - 制限事項 22
    - 構文 22.1.1
    - セグメントの検査 22.2
    - ディスク・ブロックの検査 22.1
- DEFAULTIFパラメータ
    - SQL*Loader 10.5
- デフォルト・スキーマ
    - SQL*Loaderが判別する 9.18.1.1
- DEGREE_OF_PARALLELISMパラメータ
    - SQL*Loaderコマンドライン 8.2.8
    - SQL*Loaderエクスプレス・モード 13.3.6
- DELETE ANY TABLE権限
    - SQL*Loader 9.18.2.2.4
- DELETE CASCADE
    - 空でない表へのデータのロードの影響 9.18.2.2.2
    - SQL*Loader 9.18.2.2.4
- DELETE権限
    - SQL*Loader 9.18.2.2.2
- デリミタ付きデータ
    - SQL*Loaderの最大長 10.4.6.3
- デリミタ付きフィールド
    - フィールド長 10.4.8.2
- デリミタ付きフィールドのLOB 11.4.2.2.3
- デリミタ付きテキスト・ファイル 17.4.8

- デリミタ
    - 外部表 15.3.4
    - 後続の空白のロード 10.4.6.4
    - データ内のマークおよびSQL*Loader 10.4.6.2
    - 外部表の指定 15.4.1
    - SQL*Loaderの指定 9.18.5, 10.4.6
    - SQL*Loaderの囲み 10.11.2.2
    - SQL*Loaderのフィールド指定 10.11.2.2
    - 終了 10.11.2.2
- DESTROYパラメータ
    - インポート・ユーティリティ 27.7.7
- ディクショナリ
    - LogMinerユーティリティの要件 24.2.1.3
- ディクショナリ・バージョンの不一致 24.10.4
- Direct NFSクライアント
    - 外部表との使用 15.3.28
- ディレクトリ別名
    - エクスポート 26.14.5
    - インポート 27.16.5
- ディレクトリ・オブジェクト
    - データ・ポンプでの使用
        - Oracle ASMの影響 1.8.3.3
- DIRECTORYパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.10
    - データ・ポンプ・インポート・ユーティリティ 3.4.9
- DIRECTパラメータ
    - エクスポート・ユーティリティ 26.5.5
    - SQL*Loaderコマンドライン 8.2.9
    - SQL*Loaderエクスプレス・モード 13.3.7
- ダイレクト・パス・エクスポート 26.9, 26.10
    - 従来型パスとの比較 26.9
    - EXEMPT ACCESS POLICY権限の効果 26.10.1
    - パフォーマンスの問題 26.10.2
    - 制限事項 26.10.3
    - セキュリティの考慮事項 26.10.1
- ダイレクト・パス・ロード
    - メリット 12.4.5
    - 中断された場合の動作 9.16.2
    - ソート順序の選択
        - SQL*Loader 12.6.3.5
    - 従来型パス・ロードとの比較 12.4.8
    - 同時 12.10.4
    - 使用の条件 12.4.6

    - データ・セーブ 12.5.5.1, 12.6.4
    - 索引の削除 12.8
    - アーカイブ使用禁止の影響 12.6.5
    - 主キー制約の影響 12.10.9
    - 一意キー制約の影響 12.10.9
    - フィールド・デフォルト 12.4.10
    - 不適切なソート
        - SQL*Loader 12.6.3.3
    - 索引 12.5.3
    - インスタンス・リカバリ 12.5.6
    - セグメント間同時処理 12.10.4
    - セグメント内同時処理 12.10.4
    - データ変換の位置 12.4.2
    - メディア・リカバリ 12.5.6.1
    - 複数CPUシステムの最適化 12.7
    - パーティション・ロード
        - SQL*Loader 12.9.4
    - パフォーマンス 12.5.3, 12.6.1
    - 記憶域の事前割当て 12.6.2
    - データの事前ソート 12.6.3
    - リカバリ 12.5.6
    - ROWSコマンドライン・パラメータ 12.5.5.2
    - 設定 12.5.1
    - 指定 12.5.2
    - 読み込む行数の指定 8.2.31
    - SQL*Loaderでのデータ・ロード方法 7.9.2
    - 表挿入トリガー 12.9.2
    - 一時セグメント記憶域要件 12.5.3.2
    - トリガー 12.9
    - 使用 12.4.8, 12.5
    - バージョンの要件 12.4.6
- 無効な一意索引
    - 表のロード 1.2
- 廃棄されたSQL*Loaderレコード 7.7
    - 原因 9.12.5
    - 廃棄ファイル 9.12.1
    - 制限 9.12.3
- 廃棄ファイル
    - SQL*Loader 9.12.1
        - 最大値の指定 9.12.7
- DISCARDMAXパラメータ
    - SQL*Loaderコマンドライン 8.2.12
- DISCARDパラメータ

    - SQL*Loaderコマンドライン 8.2.11
- 中断されたロード 9.16
    - 継続 9.16.5
    - 従来型パスの動作 9.16.1
    - ダイレクト・パスの動作 9.16.2
- DNFS_ENABLEパラメータ
    - SQL*Loaderコマンドライン 8.2.13
    - SQL*Loaderエクスプレス・モード 13.3.8
- DNFS_READBUFFERSパラメータ
    - record_format_info句 15.3.29
    - SQL*Loaderコマンドライン 8.2.14
    - SQL*Loaderエクスプレス・モード 13.3.9
- DOUBLE
    - 浮動小数点数 15.4.6.5
- 移植不能なDOUBLEデータ型
    - SQL*Loaderデータ型 10.4.2.5
- 削除されたスナップショット
    - インポート 27.18.2.1
- DUMPFILEパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.11
    - データ・ポンプ・インポート・ユーティリティ 3.4.10
- ダンプ・ファイル
    - 最大サイズ 26.5.8

E

- EBCDIC文字セット
    - インポート 26.12.3, 27.14.3
- ECHOコマンド, ADRCIユーティリティ 21.9.2
- 8ビット文字セットのサポート 26.12.3, 27.14.3
- ENCLOSED_BYパラメータ
    - SQL*Loaderエクスプレス・モード 13.3.10
- 囲まれたフィールド
    - 空白 10.11.6
- 囲みデリミタ 10.4.6
    - SQL*Loader 10.11.2.2
- 暗号化された列
    - 外部表 16.2.7
- ENCRYPTION_ALGORITHMパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.14
- ENCRYPTION_MODEパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.15
- ENCRYPTION_PASSWORDパラメータ

    - データ・ポンプ・エクスポート・ユーティリティ 2.4.16
    - データ・ポンプ・インポート・ユーティリティ 3.4.12
- ENCRYPTION_PWD_PROMPTパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.17
    - データ・ポンプ・インポート・ユーティリティ 3.4.13
- ENCRYPTIONパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.13
- 暗号化パスワード
    - データ・ポンプ・エクスポート 2.4.17
    - データ・ポンプ・インポート 3.4.13
- endパラメータ
    - field_definitions句 15.4.5.4
    - pos_spec句 15.4.5.4
- エラー処理 17.4.2
- エラー
    - SQL*Loaderデータ中のタブ文字が原因 10.2.1
    - LONGデータ 27.10.1.2
    - オブジェクト作成 27.10.2.1
        - インポート・パラメータIGNORE 27.7.15
    - インポート中のリソース・エラー 27.10.2.3
    - エクスポート・ログ・ファイルへの書込み 26.5.15
- ERRORSパラメータ
    - SQL*Loaderコマンドライン 8.2.16
- エスケープ文字
    - 引用符付き文字列 9.4.3.2
    - データ・ポンプ・エクスポートでの使用 2.4.1
    - データ・ポンプ・インポートでの使用 3.4
    - エクスポートでの使用方法 26.5.26.1
    - インポートでの使用方法 27.7.30
- ESTIMATE_ONLYパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.19
- ESTIMATEパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.18
    - データ・ポンプ・インポート・ユーティリティ 3.4.14
- ジョブ・サイズの見積り
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.18
- EVALUATE CHECK_CONSTRAINTS句 12.9.1.2
- EXCLUDEパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.20
    - データ・ポンプ・インポート・ユーティリティ 3.4.15
- 既存のオブジェクト表
    - インポート 27.16.2
- EXIT_CLIENTパラメータ

    - データ・ポンプ・エクスポート・ユーティリティ
        - 対話方式コマンド・モード 2.5.4
    - データ・ポンプ・インポート・ユーティリティ
        - 対話方式コマンド・モード 3.5.3
- 終了コード
    - エクスポートおよびインポート 26.8, 27.9
    - SQL*Loader 1.11, 8.3
- EXITコマンド, ADRCIユーティリティ 21.9.3
- EXP_FULL_DATABASEロール
    - エクスポートでの割当て 26.2.2, 27.2.1
- expdat.dmp
    - エクスポート出力ファイル 26.5.7
- エクスポート
    - BUFFERパラメータ 26.5.1
    - 文字セット変換 26.12.1, 27.14.1
    - COMPRESSパラメータ 26.5.2
    - CONSISTENTパラメータ 26.5.3
    - CONSTRAINTSパラメータ 26.5.4
    - 従来型パス 26.9
    - 作成
        - 必要な権限 26.2.2, 27.2.1
        - 必要なビュー 26.2.2, 27.2.1
    - DIRECTパラメータ 26.5.5
    - ダイレクト・パス 26.9
    - オンライン・ヘルプの表示 26.5.13
    - セッションの例 26.6
        - 全データベース・モード 26.6.1
        - パーティション・レベル 26.6.4
        - 表モード 26.6.3
        - ユーザー・モード 26.5.17, 26.6.2
    - 終了コード 26.8, 27.9
    - データベース全体のエクスポート 26.5.11
    - 索引のエクスポート 26.5.14
    - 順序番号のエクスポート 26.14.1
    - シノニムのエクスポート 26.14.11
    - 別のオペレーティング・システムへのエクスポート 26.5.20, 27.7.20
    - FEEDBACKパラメータ 26.5.6
    - FILEパラメータ 26.5.7
    - FILESIZEパラメータ 26.5.8
    - FLASHBACK_SCNパラメータ 26.5.9
    - FLASHBACK_TIMEパラメータ 26.5.10
    - 全データベース・モード
        - 例 26.6.1

- FULLパラメータ 26.5.11
- GRANTSパラメータ 26.5.12
- HELPパラメータ 26.5.13
- INDEXESパラメータ 26.5.14
- 起動 26.3, 27.5
- ログ・ファイル
    - 指定 26.5.15
- エラー・メッセージのログ 26.5.15
- LOGパラメータ 26.5.15
- LONG列 26.14.2
- OBJECT_CONSISTENTパラメータ 26.5.16
- オンライン・ヘルプ 26.3.5
- OWNERパラメータ 26.5.17
- パラメータ・ファイル 26.5.18
    - 最大サイズ 26.3.3, 27.5.2
- パラメータ構文 26.5
- PARFILEパラメータ 26.5.18
- データベース移行のパーティション化 26.17, 27.25
- QUERYパラメータ 26.5.19
- RECORDLENGTHパラメータ 26.5.20
- ログ・ファイルへの出力のリダイレクト 26.7.1
- リモート操作 26.11.2, 27.13
- 権限に基づく制限 26.2.4
- RESUMABLE_NAMEパラメータ 26.5.22
- RESUMABLE_TIMEOUTパラメータ 26.5.23
- RESUMABLEパラメータ 26.5.21
- ROWSパラメータ 26.5.24
- 順序番号 26.14.1
- 記憶域要件 26.2.3
- 表モード
    - セッションの例 26.6.3
- 表名に関する制限 26.5.26.1
- TABLESPACESパラメータ 26.5.27
- TABLESパラメータ 26.5.26
- TRANSPORT_TABLESPACEパラメータ 26.5.28
- TRIGGERSパラメータ 26.5.29
- TTS_FULL_CHECKパラメータ 26.5.30
- ユーザー・アクセス権限 26.2.4, 27.2.2
- USERIDパラメータ 26.5.31
- ユーザー・モード
    - セッションの例 26.5.17, 26.6.2
    - 指定 26.5.17
- VOLSIZEパラメータ 26.5.32

- エクスポート・ダンプ・ファイル
  - ファイル全体のインポート 27.7.12
- エクスポート・ファイル
  - インポート実行前の内容のリスト 27.7.25
  - 指定 26.5.7
- エクスポート
  - アーカイブされたLOB 2.4.51
- EXPRESSIONパラメータ
  - SQL*Loader 10.15.3, 10.15.3.1
- エクステント
  - 集計 26.5.2
- EXTERNAL_TABLEパラメータ
  - SQL*Loaderコマンドライン 8.2.17
  - SQL*Loaderエクスプレス・モード 13.3.11
- EXTERNALパラメータ
  - SQL*Loader 10.4.3.6
- EXTERNAL SQL*Loaderデータ型
  - 数値
    - 長さの決定 10.4.8
- 外部表 14
  - アクセス・パラメータ 14.3, 15.2, 16.2
  - 暗号化された列 16.2.7
  - ビッグ・エンディアン・データ 15.3.11
  - 読取り中のデータのキャッシュ 15.3.27
  - column_transforms句 15.2
  - データ型 15.4.6
  - 日付キャッシュ機能 15.7
  - デリミタ 15.3.4
  - フィールドのデータ型の記述 15.4.6
  - field_definitions句 15.2, 15.4
  - 固定長レコード 15.3.2
  - 文字セットの識別 15.3.6
  - データ型の識別 15.4.4
  - 使用時のパフォーマンスの向上
    - 日付キャッシュ機能 15.7
  - IO_OPTIONS句 15.3.27
  - リトル・エンディアン・データ 15.3.11
  - opaque_format_spec 14.3, 15.2, 16.2
  - データの前処理 15.3.8
  - record_format_info句 15.2, 15.3.1
  - 予約語 15.8, 16.7
  - 制限事項 15.8
  - フィールドのデフォルト値設定 15.4.7

    - フィールドのNULL設定 15.4.7
    - データのロード時のレコードのスキップ 15.3.18
    - 圧縮レベルの指定 16.2.2
    - デリミタの指定 15.4.1
    - ロード条件の指定 15.3.14
    - 空白の切捨て 8.2.38, 13.3.21, 15.4.2
    - SQL文字列の使用 15.8
    - コメントの使用 15.2, 16.2.1
    - 可変長レコード 15.3.3

F

- 致命的エラー
    - 「リカバリ不能なエラー・メッセージ」を参照
- FEEDBACKパラメータ
    - エクスポート・ユーティリティ 26.5.6
    - インポート・ユーティリティ 27.7.8
- field_definitions句
    - FLOAT [EXTERNAL] 15.4.6.7
- FIELD_NAMESパラメータ
    - SQL*Loaderエクスプレス・モード 13.3.12
- フィールド条件
    - SQL*Loaderへの指定 10.5
- フィールドの抽出 17.4.8
- フィールド長
    - SQL*Loaderの指定 10.11.2
- 移植可能なLENGTH-VALUEデータ型のフィールド長
    - SQL*Loaderデータ型 10.4.3.11
- フィールド位置
    - SQL*Loader 10.2
- フィールド名 17.4.3
- フィールド
    - SQL*Loaderを使用した文字列との比較 10.5.3
    - デリミタ付き
        - 長さの決定 10.4.8.2
        - SQL*Loader 10.4.6
    - すべての空白のロード 10.10
    - 事前のサイズ決定
        - 長さ 10.4.8.1
        - SQL*Loader 10.11.2.1
    - 相対的な位置指定およびSQL*Loader 10.11.3
    - SQL*Loaderへのデフォルトのデリミタの指定 9.18.5
    - SQL*Loaderへの指定 10.3

    - デリミタ付きのSQL*Loader
        - 指定 10.11.2.2
- FIELDS句
    - SQL*Loader 9.18.5
- データ・ポンプでのファイルの割当て
    - ダンプ・ファイルの追加 1.8.2
    - ファイルのデフォルトの場所 1.8.3.1
    - NFSエラー 1.8
- ファイル名
    - 引用符 9.4.3.1
    - 複数のSQL*Loaderの指定 9.7.3
    - SQL*Loader 9.4
    - SQL*Loader不良ファイル 9.11.1
- FILEパラメータ
    - エクスポート・ユーティリティ 26.5.7
    - インポート・ユーティリティ 27.7.9
    - SQL*Loaderコマンドライン 8.2.18
    - SQL*Loaderユーティリティ 12.10.7.1
- FILESIZEパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.21
    - エクスポート・ユーティリティ 26.5.8
    - インポート・ユーティリティ 27.7.10
- FILLERフィールド
    - init_specに対する引数としての使用 10.3.1
- データのフィルタ処理
    - データ・ポンプ・エクスポート・ユーティリティの使用方法 2.1
    - データ・ポンプ・インポート・ユーティリティの使用方法 3.1
- インポートされるメタデータをフィルタ処理
    - データ・ポンプ・インポート・ユーティリティ 3.4.15
- ファイナライズ
    - ADRCIユーティリティ内 21.2
- ファイングレイン・アクセスのサポート
    - エクスポートおよびインポート 27.17
- 固定形式レコード 7.4.1
- 固定長レコード
    - 外部表 15.3.2
- FLASHBACK_SCNパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.22
    - データ・ポンプ・インポート・ユーティリティ 3.4.16
    - エクスポート・ユーティリティ 26.5.9
- FLASHBACK_TIMEパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.23
    - データ・ポンプ・インポート・ユーティリティ 3.4.17

    - エクスポート・ユーティリティ 26.5.10
- FLOAT
    - 浮動小数点数 15.4.6.5
- FLOAT [EXTERNAL]パラメータ
    - datatype_spec句 15.4.6.7
- 浮動小数点数
    - datatype_spec句 15.4.6.5
- 外部関数ライブラリ
    - エクスポート 26.14.3
    - インポート 27.16.6, 27.16.9
- 形式
    - SQL*Loader入力レコード 9.20.2
- 書式エラー
    - SQL*Loader 9.11.1
- FROMUSERパラメータ
    - インポート・ユーティリティ 27.7.11
- 全データベース・モード
    - インポート 27.7.12
    - FULLでの指定 26.5.11
- 全体エクスポート・モード
    - データ・ポンプ・エクスポート・ユーティリティ 2.2.2.1
- FULLパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.24
    - データ・ポンプ・インポート・ユーティリティ 3.4.18
    - エクスポート・ユーティリティ 26.5.11
    - インポート・ユーティリティ 27.7.12
- 全体トランスポータブル・エクスポート 2.2.2.1
- 全体トランスポータブル・インポート 3.2.2.2

G

- グローバリゼーション
    - SQL*Loader 9.15
- データ・ポンプのGoldenGateレプリケーション環境
    - バッファ・キャッシュ・サイズの設定 5.3.1
- 権限
    - エクスポート 26.5.12
    - インポート 27.7.13
- GRANTSパラメータ
    - エクスポート・ユーティリティ 26.5.12
    - インポート・ユーティリティ 27.7.13
- GRAPHIC 10.4.3.4
- GRAPHICデータ型 10.4.3.4

H

- Hadoopログ・ファイル 17.4.5
- HELPパラメータ
  - データ・ポンプ・エクスポート・ユーティリティ
    - コマンドライン・モード 2.4.25
    - 対話方式コマンド・モード 2.5.6
  - データ・ポンプ・インポート・ユーティリティ
    - コマンドライン・モード 3.4.19
    - 対話方式コマンド・モード 3.5.4
  - エクスポート・ユーティリティ 26.5.13
  - インポート・ユーティリティ 27.7.14
- 16進文字列
  - SQL*Loader 10.5.3
- Hive表ソース 17.4.9
- ホームパス
  - ADRCIユーティリティ内 21.2
- HOSTコマンド, ADRCIユーティリティ 21.9.4

I

- IGNOREパラメータ
  - インポート・ユーティリティ 27.7.15
- IMP_FULL_DATABASEロール
  - インポートでの割当て 26.2.2, 27.2.1
- インポート
  - BUFFERパラメータ 27.7.1
  - 文字セット変換 26.12.1, 27.14.1, 26.12.3, 27.14.3
  - COMMITパラメータ 27.7.2
  - 配列挿入後のコミット 27.7.2
  - COMPILEパラメータ 27.7.3
  - CONSTRAINTSパラメータ 27.7.4
  - 作成
    - 必要な権限 26.2.2, 27.2.1
    - 必要なビュー 26.2.2, 27.2.1
  - 索引作成SQLスクリプトの作成 27.7.17
  - データベース・オプティマイザ統計 27.7.27
  - DATAFILESパラメータ 27.7.6
  - DESTROYパラメータ 27.7.7
  - 参照制約を使用禁止にする方法 27.3.2
  - オンライン・ヘルプの表示 27.7.14
  - 表領域を削除する方法 27.22

- データベース・オブジェクトのインポートでのエラー 27.10.2
- セッションの例 27.8
    - あるユーザーから別のユーザーへのすべての表のインポート 27.8.3
    - 特定のユーザー用に選択された表 27.8.1
    - 別のユーザーによってエクスポートされた表 27.8.2
    - パーティション・レベル・インポートの使用 27.8.4
- 終了コード 26.8, 27.9
- エクスポート・ファイル
    - ファイル全体のインポート 27.7.12
    - インポート実行前の内容のリスト 27.7.25
- FEEDBACKパラメータ 27.7.8
- FILEパラメータ 27.7.9
- FILESIZEパラメータ 27.7.10
- FROMUSERパラメータ 27.7.11
- FULLパラメータ 27.7.12
- 権限
    - インポート時の指定 27.7.13
- GRANTSパラメータ 27.7.13
- HELPパラメータ 27.7.14
- IGNOREパラメータ 27.7.15
- 権限のインポート 27.7.13
- 他のスキーマへのオブジェクトのインポート 27.2.2.3
- 表のインポート 27.7.30
- INDEXFILEパラメータ 27.7.17
- INSERTエラー 27.10.1.2
- 無効なデータ 27.10.1.2
- 起動 26.3, 27.5
- LOGパラメータ 27.7.18
- LONG列 27.16.11
- インポート実行前の手動による表の作成 27.3.1
- 手動による表の順序付け 27.3.3
- NLS_LANG環境変数 26.12.3, 27.14.3
- オブジェクト作成エラー 27.7.15
- オンライン・ヘルプ 26.3.5
- パラメータ・ファイル 27.7.19
    - 最大サイズ 26.3.3, 27.5.2
- パラメータ構文 27.7
- PARFILEパラメータ 27.7.19
- パーティション表 27.16.14
- パーティション・レベル 27.11
- 表名のパターン一致 27.7.30
- 読取り専用表領域 27.21
- RECORDLENGTHパラメータ 27.7.20

- レコード
  - 長さの指定 27.7.20
- ログ・ファイルへの出力のリダイレクト 26.7.1
- REFデータ 27.16.4
- リフレッシュ・エラー 27.18.1
- リモート操作 26.11.2, 27.13
- 表領域の再編成 27.23
- リソース・エラー 27.10.2.3
- 制限事項
  - 自分のスキーマへのインポート 27.2.2.1
- RESUMABLE_NAMEパラメータ 27.7.22
- RESUMABLE_TIMEOUTパラメータ 27.7.23
- RESUMABLEパラメータ 27.7.21
- 既存のデータ・ファイルの再利用 27.7.7
- スキーマ・オブジェクト 27.2.2.3
- 順序 27.10.2.2
- SHOWパラメータ 27.7.25
- シングルバイト文字セット 26.12.3, 27.14.3
- SKIP_UNUSABLE_INDEXESパラメータ 27.7.26
- スナップショット・マスター表 27.18.1
- スナップショット 27.18
  - 削除された場合のリストア 27.18.2.1
- ユーザーによる指定 27.7.11
- 索引作成コマンドの指定 27.7.17
- エクスポート・ファイルの指定 27.7.9
- STATISTICSパラメータ 27.7.27
- 記憶域パラメータ
  - 上書き 27.20.3
- ストアド・ファンクション 27.16.7
- ストアド・プロシージャ 27.16.7
- STREAMS_CONFIGURATIONパラメータ 27.7.28
- STREAMS_INSTANTIATIONパラメータ 27.7.29
- システム・オブジェクト 27.2.2.4
- 表レベル 27.11
- 表名に関する制限 2.4.45, 3.4.49, 27.7.30.1
- 表オブジェクト
  - インポート順序 27.1.1
- TABLESPACESパラメータ 27.7.31
- TABLESパラメータ 27.7.30
- TOID_NOVALIDATEパラメータ 27.7.32
- TOUSERパラメータ 27.7.33
- TRANSPORT_TABLESPACEパラメータ 27.7.34
- TTS_OWNERパラメータ 27.7.35

    - チューニングに関する考慮点 27.26
    - ユーザー・アクセス権限 26.2.4, 27.2.2
    - USERIDパラメータ 27.7.36
    - ビュー 27.16.13
    - VOLSIZEパラメータ 27.7.37
- インシデント
    - 障害診断インフラストラクチャ 21.2
    - パッケージ化 21.8
- インシデント・パッケージ
    - 障害診断インフラストラクチャ 21.2
- INCLUDEパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.26
    - データ・ポンプ・インポート・ユーティリティ 3.4.20
- incrementパラメータ
    - field_definitions句 15.4.5.3
    - pos_spec句 15.4.5.3
- 索引
    - 手動での作成 27.7.17
    - ダイレクト・パス・ロード
        - ダイレクト・ロードの状態 12.5.4
    - 削除
        - SQL*Loader 12.8
    - 記憶域要件の計算 12.5.3.1
    - エクスポート 26.5.14
    - 索引作成コマンド
        - インポート 27.7.17
    - 使用禁止状態 9.16.3, 12.6.3.3
    - 複数列
        - SQL*Loader 12.6.3.4
    - データの事前ソート
        - SQL*Loader 12.6.3
    - 使用禁止の場合のスキップ 8.2.36
    - SQL*Loader 9.19
    - ロード中断後の状態 9.16.3
    - 一意 27.7.16
- INDEXESパラメータ
    - エクスポート・ユーティリティ 26.5.14
- INDEXFILEパラメータ
    - インポート・ユーティリティ 27.7.17
- 索引オプション
    - SQL*LoaderでのSORTED INDEXES 9.19.1
    - SQL*Loader SINGLEROWパラメータ 9.19.2
- 索引使用禁止状態

    - 索引使用禁止状態のままの索引 9.16.3, 12.5.4
- INFILEパラメータ
    - SQL*Loaderユーティリティ 9.7.1
- 初期化パラメータ
    - MAX_DATAPUMP_JOBS_PER_PDB 5.3.2
    - MAX_DATAPUMP_PARALLEL_PER_JOB 5.3.2
- 挿入エラー
    - インポート 27.10.1.2
    - 指定 8.2.16
- 表へのINSERT
    - SQL*Loader 9.18.2.1
- インスタンス親和性
    - エクスポートおよびインポート 26.13
- インスタンス・リカバリ 12.5.6.2
- 整合性制約
    - ダイレクト・パス・ロード時に使用禁止 12.9.1.2
    - ダイレクト・パス・ロード時に使用可能 12.9.1.1
    - インポート時の違反 27.10.1.1
    - ロード方法 12.4.9
- 対話方式
    - データ・ポンプ・エクスポート・ユーティリティ 2.2.1
- 内部LOB
    - ロード11.4.1
- 中断されたロード 9.16
- INTERVAL句
    - date_format_spec 15.4.6.12.4
- INTERVALDAYTOSECOND
    - データ型 10.4.3.3.9
- INTO TABLE文
    - バインド配列サイズへの影響 9.21.6
    - SQL*Loaderでの複数の文 9.20
    - SQL*Loader 9.18
        - 列名 10.3
        - 廃棄 9.12.5
- 無効なデータ
    - インポート 27.10.1.2
- 起動
    - エクスポート 26.3, 27.5
        - コマンドライン 26.3.2, 27.5.1
        - ダイレクト・パス 26.10
        - 対話方式 26.3.4, 27.5.3
        - パラメータ・ファイル 26.3.3, 27.5.2
    - インポート 26.3, 27.5

        - SYSDBA 26.3.1, 27.5.4
        - コマンドライン 26.3.2, 27.5.1
        - 対話方式 26.3.4, 27.5.3
        - パラメータ・ファイル 26.3.3, 27.5.2
- IPSコマンド, ADRCIユーティリティ 21.9.5
- IPS SHOW FILES
    - IPS 21.9.5.18

J

- Javaオブジェクト
    - インポート 27.16.8
- JAVAVM 20.11.21
- JOB_NAMEパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.27
    - データ・ポンプ・インポート・ユーティリティ 3.4.21

K

- KEEP_MASTERパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.28
    - データ・ポンプ・インポート・ユーティリティ 3.4.22
- キー値
    - SQL*Loaderを使用した生成 10.15.6
- KILL_JOBパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ
        - 対話方式コマンド・モード 2.5.7
    - データ・ポンプ・インポート・ユーティリティ 3.5.5

L

- 先頭の空白
    - 定義 10.11.1
    - 切捨ておよびSQL*Loader 10.11.4
- Oracle Data Pumpのレガシー・モード 4
- 長さインジケータ
    - サイズの決定 9.21.4.1
- lengthパラメータ
    - field_definitions句 15.4.5.5
    - pos_spec句 15.4.5.5
- Length-Value Pairで指定したLOB 11.4.2.2.4
- ライブラリ

    - 外部関数
        - エクスポート 26.14.3
        - インポート 27.16.6, 27.16.9
- リトル・エンディアン・データ
    - 外部表 15.3.11
- ロード
    - コレクション 11.6
    - 列オブジェクト 11.1
        - 可変レコード形式 11.1.2
        - 導出サブタイプの使用 11.1.4
        - ユーザー定義コンストラクタ 11.1.6
    - タブを含むデータ・ファイル
        - SQL*Loader 10.2.1
    - 外部表データ
        - レコードのスキップ 15.3.18
        - 条件の指定 15.3.11, 15.3.24
    - LOB 11.4
    - ネストした列オブジェクト 11.1.3
    - オブジェクト表 11.2
    - サブタイプを使用したオブジェクト表 11.2.1
    - REF列 11.3
    - サブパーティション表 12.4.3
    - 表 12.4.3
- LOADパラメータ
    - SQL*Loaderコマンドライン 8.2.20, 13.3.13
    - SQL*Loaderエクスプレス・モード 13.3.13
- LOB列
    - LOB列のインポート 27.16.12
- LOBデータ
    - デリミタ付きフィールド 11.4.1.2
    - Length-Value Pairフィールド 11.4.1.3
    - 事前にサイズが決まっているフィールド 11.4.1.1
    - SQL*Loaderでのロード 11.4
    - エクスポート時の圧縮なし 26.5.2
    - 読取りバッファのサイズ 8.2.27
    - データ・ポンプでの記憶域の指定 3.4.52
    - SQL*Loaderでサポートされる型 7.10.3, 11.4
- LOBデータ型 7.5
- LOBFILE 7.5, 11.4, 11.4.2
- LOGFILEパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.29
    - データ・ポンプ・インポート・ユーティリティ 3.4.23
        - LOGFILEパラメータ 3.4.23

- ログ・ファイル 17.4.6
    - ロード中断後 9.16.4
    - エクスポート 26.5.15, 26.7.1
    - インポート 26.7.1, 27.7.18
    - データ・ポンプで書き込むデータの制限 3.4.52
    - SQL*Loaderへの指定 8.2.21
    - SQL*Loader 7.8
- 論理レコード
    - SQL*Loaderを使用した複数の物理レコードの統合 9.17
- LogMinerユーティリティ
    - 分析するREDOデータへのアクセス 24.6
    - REDOログ・ファイル・リストの調整 24.9
    - 出力の分析 24.6.1
    - 構成 24.2.1.1
    - DDL文を再適用する場合の注意 24.8
    - 現在のログ・ファイル・リスト
        - 格納される情報 24.11.2
    - DBMS_LOGMNR_D.ADD_LOGFILES PL/SQLプロシージャ 24.2.2
    - DBMS_LOGMNR_D.BUILD PL/SQLプロシージャ 24.2.2
    - DBMS_LOGMNR_D.END_LOGMNR PL/SQLプロシージャ 24.2.2
    - DBMS_LOGMNR_D PL/SQLプロシージャ 24.2.2
    - DBMS_LOGMNR.START_LOGMNR PL/SQLプロシージャ 24.2.2
    - DBMS_LOGMNR PL/SQLプロシージャ 24.2.2
    - DDL追跡
        - 時間範囲またはSCN範囲 24.10.6
    - 分析中のREDOログ・ファイルの調査 24.4.2
    - ディクショナリ
        - 目的 24.2.1.1
    - フラット・ファイルから抽出されたディクショナリ
        - 格納される情報 24.11
    - ディクショナリ・オプション 24.4.1
        - フラット・ファイル 24.4.1
        - オンライン・カタログ 24.4.1
        - REDOログ・ファイル 24.4.1
    - セッションの終了 24.12.7
    - 再構築されたSQLの実行 24.7.5
    - REDOログからのデータ値の抽出 24.6.2
    - SCNによるデータのフィルタ処理 24.7.4
    - 時間によるデータのフィルタ処理 24.7.3
    - 返されるデータの書式設定 24.7.6
    - グラフィカル・ユーザー・インタフェース 24
    - サプリメンタル・ロギングのレベル 24.10
    - LogMinerディクショナリの意味 24.2.1.1

- REDOログ・ファイルのデータのサブセットのマイニング 24.9
- マイニング・データベースの意味 24.2.1.1
- サポートされていないオブジェクトまたはデータ型 24.2.1.3
- 操作概要 24.2.2
- パラメータ
  - 格納される情報 24.11
- REDOログ・ファイル
  - リモート・データベース 24.9
  - 格納される情報 24.11
- ディクショナリの要件 24.2.1.3
- REDOログ・ファイルの要件 24.2.1.3
- ソース・データベースおよびマイニング・データベースの要件 24.2.1.3
- XMLTypeデータを使用する際の制限事項 24.6.3.2
- サンプル構成 24.2.1.2
- コミット済トランザクションのみの表示 24.7.1
- 破損のスキップ 24.7.2
- ソース・データベースの意味 24.2.1.1
- マイニングするREDOログ・ファイルの指定 24.4.2
  - 自動 24.4.2
  - 手動 24.4.2
- 分析対象のREDOログの指定 24.12.4
- 起動 24.5, 24.12.5
- 同一セッション内での複数回の起動 24.9
- フラット・ファイルに対してディクショナリを抽出するステップ 24.4.1.3
- REDOログ・ファイルに対してディクショナリを抽出するステップ 24.4.1.2
- オンライン・カタログにあるディクショナリを使用するステップ 24.4.1.1
- 一般的セッションのステップ 24.12.1
- サプリメンタル・ロギング 24.10
  - データベース・レベル 24.10.1
  - データベース・レベルの識別キー 24.10.1.2
  - データベース・レベルでの無効化 24.10.2
  - DDL追跡との相互作用 24.10.5
  - ログ・グループ 24.10
  - 最小 24.10.1.1
  - 格納される情報 24.11
  - 表レベルの識別キー 24.10.3.1
  - 表レベルのログ・グループ 24.10.3.2
  - ユーザー定義のログ・グループ 24.10.3.3
- サプリメンタル・ログ・グループ 24.10
  - 条件付き 24.10
  - 無条件 24.10
- サポートされるデータベース・バージョン 24.14.3
- サポートされるデータ型 24.14.1

    - サポートされるREDOログ・ファイル・バージョン 24.14.3
    - 透過的データ暗号化のサポート 24.6
    - SQL_REDOおよびSQL_UNDOでのデリミタの抑制 24.7.5
    - 表レベルのサプリメンタル・ロギング 24.10.3
    - DDL文の追跡 24.10.4
        - 要件 24.10.4
    - サポートされないデータ型 24.14.2
    - CDBでの使用 24.3
    - オンライン・カタログの使用 24.4.1.1
    - REDOログ・ファイル分析のための使用 24
    - V$DATABASEビュー 24.11
    - V$LOGMNR_CONTENTSビュー 24.2.2, 24.6.1, 24.7
    - V$LOGMNR_LOGSビュー
        - 問合せ 24.11.2
    - ビュー 24.11
- LogMinerビューア 24
- LOGパラメータ
    - エクスポート・ユーティリティ 26.5.15
    - インポート・ユーティリティ 27.7.18
    - SQL*Loaderコマンドライン 8.2.21
- LOGTIMEパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.30
    - データ・ポンプ・インポート・ユーティリティ 3.4.24
- LONGデータ
    - エクスポート 26.14.2
    - インポート 27.16.11

M

- MASTER_ONLYパラメータ
    - データ・ポンプ・インポート・ユーティリティ 3.4.25
- マスター表
    - スナップショット
        - オリジナルのインポート 27.18.1
- マテリアライズド・ビュー 27.18
- MAX_DATAPUMP_JOBS_PER_PDB初期化パラメータ 5.3.2
- MAX_DATAPUMP_PARALLEL_PER_JOB初期化パラメータ 5.3.2
- メディア・リカバリ
    - ダイレクト・パス・ロード 12.5.6.1
- メタデータAPI
    - パフォーマンスの向上 25.7
    - コレクションの取得 25.5
    - オブジェクト・メタデータの取得での使用 25.3

- METRICSパラメータ
  - データ・ポンプ・エクスポート・ユーティリティ 2.4.31
  - データ・ポンプ・インポート・ユーティリティ 3.4.26
- データ列の欠落
  - SQL*Loader 9.18.6
- マルチバイト文字セット
  - SQL*Loaderでの空白 10.5.2
  - SQL*Loader 9.15.1
- 複数列索引
  - SQL*Loader 12.6.3.4
- 複数CPUシステム
  - ダイレクト・パス・ロードの最適化 12.7
- 複数表へのロード
  - SQL*Loaderを使用した一意の順序番号の生成 10.15.7
  - SQL*Loader制御ファイルの指定 9.20
- マルチテナント・アーキテクチャ 24.3.2
- マルチテナント・コンテナ・データベース 1.3, 24.3
- マルチスレッド
  - 複数CPUシステム 12.7
- MULTITHREADINGパラメータ
  - SQL*Loaderコマンドライン 8.2.22

N

- 名前付きパイプ
  - 外部表ロード 7.9.4
- ネイティブなデータ型
  - 長さ指定との競合
    - SQL*Loader 10.4.3.10
- NCLOB
  - SQL*Loaderでのロード 11.4
- ネストした列オブジェクト
  - ロード 11.1.3
- ネストした表
  - エクスポート 26.14.9
    - 一貫性 26.5.3
  - インポート 27.16.3
- NETWORK_LINKパラメータ
  - データ・ポンプ・エクスポート・ユーティリティ 2.4.32
  - データ・ポンプ・インポート・ユーティリティ 3.4.27
- ネットワーク
  - エクスポートおよびインポート 26.11, 27.13
- データ・ポンプ使用時のNFSエラー 1.8

- NLS_LANG環境変数 26.12.2, 27.14.2
    - エクスポートおよびインポート 26.12.3, 27.14.3
- NO_INDEX_ERRORSパラメータ
    - SQL*Loaderコマンドライン 8.2.23
- NOLOGFILEパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.33
    - データ・ポンプ・インポート・ユーティリティ 3.4.28
- リカバリ不能なエラー・メッセージ
    - エクスポート 26.7.3
    - インポート 26.7.3
- 非スカラー・データ型 11.1.5
- NOT NULL制約
    - ロード方法 12.4.9
- NULLデータ
    - ロード中のレコードの終わりの欠落列 9.18.6
    - 指定されていない列およびSQL*Loader 10.3
- NULLIF
    - 文字フィールド 9.13
- NULLIF...BLANKS句
    - SQL*Loader 10.5.2
- NULLIF句
    - SQL*Loader 10.5, 10.10
- NULLIFパラメータ
    - SQL*Loaderエクスプレス・モード 13.3.14
- NULL
    - アトミック 11.1.5.2
    - 属性 11.1.5.1
- NULL値
    - オブジェクト 11.1.5
- NUMBERデータ型
    - SQL*Loader 10.4.4, 10.4.5
- 数値型EXTERNALデータ型
    - デリミタ付き形式およびSQL*Loader 10.4.6
    - 長さの決定 10.4.8

O

- OBJECT_CONSISTENTパラメータ
    - エクスポート・ユーティリティ 26.5.16
- オブジェクト識別子 11.2
    - インポート 27.16.1
- オブジェクト名
    - SQL*Loader 9.4

- オブジェクト 7.10
  - 作成エラー 27.10.2.1
  - インポート中の既存オブジェクトの無視 27.7.15
  - インポート作成エラー 27.7.15
  - ネストした列オブジェクトのロード 11.1.3
  - NULL値 11.1.5
  - ストリーム・レコード形式 11.1.1
  - 可変レコード形式 11.1.2
- オブジェクト表
  - ロード 11.2
  - サブタイプの使用
    - ロード 11.2.1
- オブジェクト型定義
  - エクスポート 26.14.8
- SQL*Loaderでサポートされるオブジェクト型 7.10.1
- オフライン・ローカル管理表領域
  - エクスポート 26.14.4
- OID
  - 「オブジェクト識別子」を参照
- オンライン・ヘルプ
  - エクスポートおよびインポート 26.3.5
- opaque_format_spec 14.3, 15.2, 16.2
- オペレーティング・システム
  - SQL*Loaderを使用した異なるシステムへのデータの移動 10.8
- OPTIMAL記憶域パラメータ
  - エクスポートおよびインポートでの使用 27.20.1
- オプティマイザ統計 27.24
- 最適化
  - ダイレクト・パス・ロード 12.6.1
  - SQL*Loader入力ファイル処理 9.10
- OPTIONALLY_ENCLOSED_BYパラメータ
  - SQL*Loaderエクスプレス・モード 13.3.15
- OPTIONALLY ENCLOSED BY句
  - SQL*Loader 10.11.2.2
- OPTIONSパラメータ
  - パラレル・ロード 9.18.3
  - SQL*Loaderユーティリティ 9.3.1
- ORA-00959表領域が存在しません 20.11.12, 20.11.13, 20.11.16, 20.11.17, 20.11.18, 20.11.20
- ORA-27486: 権限が不足しています 20.11.29, 20.11.30, 20.11.31
- ORA-39083: オブジェクト型INDEX_STATISTICSの作成がエラーで失敗しました 20.11.47
- ora-39095: ダンプ・ファイル領域を使い果たしました 2.4.34
- ORA-39127: コールからの予期しないエラー 20.11.32, 20.11.33, 20.11.34, 20.11.35, 20.11.36
- ORA-39173: 暗号化データがダンプ・ファイル・セットに暗号化されずに格納されました。 1.13

- ORA-39181: ファイングレイン・アクセス・コントロールのため、データの一部のみがエクスポートされました 1.4
- ORA-39346 1.9
- ORA-39346: オブジェクトの文字セット変換でデータが失われました 20.11.37
- ORA-39357: 警告: コンテナ・データベースのルートまたはシードに接続する場合、Oracle Data Pumpの操作は通常必要ありません。 1.3.1
- ORA-39777
  - long varcharデータ 8.2.31
- ORA-65094: ローカル・ユーザーまたはロール名が無効です 1.3.3
- ORACLE_DATAPUMPアクセス・ドライバ
  - SQL ENCRYPT句の使用による影響 16.2.7
  - 予約語 16, 16.8
- ORACLE_HIVE
  - アクセス・パラメータ 17.3
- ORACLE_LOADERアクセス・ドライバ
  - DNFS_READBUFFERSパラメータ 15.3.29
  - 予約語 15, 15.9
- Oracleアドバンスト・キューイング
  - 「アドバンスト・キューイング」を参照
- Oracle Automatic Storage Management(ASM)
  - データ・ポンプ 1.8.3.3
- Oracle Autonomous Database
  - DBMS_CLOUD.CREATE_CREDENTIAL() 3.4.7
  - インポート 3.6.4
- OracleバイナリJSON (OSON) 12.4.1
- Oracle Data Pump
  - ダイレクト・パス・ロード
    - 制限事項 1.2.2
  - 親ジョブ表 1.5.2
  - パフォーマンスのチューニング 5.2
- Oracle Data Pump API 6
  - クライアント・インタフェース 6.1
  - ジョブの状態 6.1.1
  - ジョブの進捗状況の監視 1.7
- Oracle Data Pumpのレガシー・モード 4
- ORCファイル 17.4.4
- OSON (OracleバイナリJSON) 12.4.1
- オーバーフロー処理 17.4.7
- OWNERパラメータ
  - エクスポート・ユーティリティ 26.5.17

P

- パッケージ

    - 作成 21.8.2
- 比較文字列の埋込み
    - SQL*Loader 10.5.3
- PARALLEL_THRESHOLDパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ
        - コマンドライン・インタフェース 2.4.35
    - データ・ポンプ・インポート・ユーティリティ
        - コマンドライン・インタフェース 3.4.30
- パラレル・ロード 12.10.1
    - ダイレクト・パスの制限 12.10.5
    - PREPROCESSOR句を使用する場合 15.3.8, 15.3.8.1
- PARALLELパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ
        - コマンドライン・インタフェース 2.4.34
        - 対話方式コマンド・モード 2.5.8
    - データ・ポンプ・インポート・ユーティリティ
        - コマンドライン・モード 3.4.29
        - 対話方式コマンド・モード 3.5.6
    - SQL*Loaderコマンドライン 8.2.24
- パラメータ・ファイル
    - エクスポート 26.5.18
    - エクスポートおよびインポート
        - コメント 26.3.3, 27.5.2
        - 最大サイズ 26.3.3, 27.5.2
    - インポート 27.7.19
    - SQL*Loader 8.2.25
- 親ジョブ表
    - Oracle Data Pump API 1.5.2
- PARFILEパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.36
    - データ・ポンプ・インポート・ユーティリティ 3.4.31
    - エクスポート・コマンドライン 26.5.18
    - インポート・コマンドライン 27.7.19
    - SQL*Loaderエクスプレス・モード 13.3.16
- Parquetファイル 17.4.4
- HDFSファイルの解析 17.4.8
- PARTITION_MEMORYパラメータ
    - SQL*Loaderユーティリティ 8.2.26
- PARTITION_OPTIONSパラメータ
    - データ・ポンプ・インポート・ユーティリティ 3.4.32
- パーティション・ロード
    - 従来型パスによる同時ロード 12.9.4
    - SQL*Loader 12.9.4

- SQL*Loaderでのパーティション・オブジェクトのサポート 7.11
- パーティション表
  - エクスポートの一貫性 26.5.3
  - エクスポート 26.4.1
  - インポート 27.8.1, 27.11.1
  - ロード12.4.3
- パーティション表
  - インポート 27.16.14
- データベース移行のパーティション化 26.17, 27.25
  - メリット 26.17, 27.25
  - デメリット 26.17, 27.25
  - エクスポート中の手順 19.2.1, 19.2.2, 26.17.3, 27.25.3
- パーティション・レベル・エクスポート 26.4.1
  - セッションの例 26.6.4
- パーティション・レベル・インポート 27.11
  - 指定 26.5.26
- パターン一致
  - インポート時の表名 27.7.30
- パフォーマンス
  - 整合性制約使用時の向上 12.9.4
  - ダイレクト・パス・ロードの最適化 12.6.1
  - SQL*Loaderデータ・ファイルの読取りの最適化 9.10
  - オリジナルのインポートのチューニング 27.26
- PIECEDパラメータ
  - SQL*Loader 12.5.8
- 移植可能なデータ型 10.4.3.4
- POSITIONパラメータ
  - タブを含むデータでの使用 10.2.1
  - 複数のSQL*LoaderのINTO TABLE句 9.20.2.1, 10.2, 10.2.2
- 事前にサイズが決まっているフィールド
  - SQL*Loader 10.11.2.1
- 事前に決められたサイズのLOB 11.4.2.2.2
- premigration -config-
  - clear_recovery_data 20.9.2.2
  - -console 20.9.2.2
  - -debug 20.9.2.2
  - -mode 20.9.2.2
  - -noconsole 20.9.2.2
  - -restore_on_fail 20.9.2.2
  - -zip 20.9.2.2
- 外部表用のデータの前処理 15.3.8
  - パラレル処理の影響 15.3.8, 15.3.8.1
- 前提条件

    - SQL*Loader 12.1
- PRESERVEパラメータ 9.17.2
- 保持
    - 空白 10.12
- PRIMARY KEY制約
    - ダイレクト・パス・ロードへの影響 12.10.9
- 主キーOID
    - 例 11.2
- 主キーREF列 11.3.3
- 権限
    - EXEMPT ACCESS POLICY
        - ダイレクト・パス・エクスポートへの影響 26.10.1
    - エクスポートおよびインポートで必要 26.2.2, 27.2.1
    - SQL*Loaderで必要 12.1
- 問題
    - 障害診断インフラストラクチャ 21.2
- 問題キー
    - 障害診断インフラストラクチャ 21.2
- PURGEコマンド, ADRCIユーティリティ 21.9.6

Q

- QUERYパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.37
    - データ・ポンプ・インポート・ユーティリティ 3.4.33
    - エクスポート・ユーティリティ 26.5.19
        - 制限事項 26.5.19.1
- QUITコマンド, ADRCIユーティリティ 21.9.7
- 引用符
    - エスケープ文字 9.4.3.2
    - ファイル名 9.4.3.1
    - SQL文字列 9.4.2
    - 表名 2.4.45, 3.4.49, 26.5.26.1, 27.7.30.1
    - データ・ポンプ・エクスポートでの使用 2.4.1
    - データ・ポンプ・インポートでの使用 3.4
    - データベース・オブジェクト名での使用 9.4.1

R

- RCファイル 17.4.4
- 読取り一貫性エクスポート 26.5.3
- 読取り専用表領域

    - インポート 27.21
- READSIZEパラメータ
    - SQL*Loaderコマンドライン 8.2.27
        - LOBへの影響 8.2.27
        - 最大サイズ 8.2.27
- RECNUMパラメータ
    - SQL*LoaderのSKIPパラメータでの使用 10.15.4
- RECORDLENGTHパラメータ
    - エクスポート・ユーティリティ 26.5.20
    - インポート・ユーティリティ 27.7.20
- レコード
    - 1つの論理レコードの作成
        - SQL*Loader 9.17
    - SQL*Loaderによる廃棄 7.7, 9.12.1
    - SQL*Loaderの異なる形式の区別 9.20.2
    - SQL*Loaderを使用した複数の論理レコードの抽出 9.20
    - 固定形式 7.4.1
    - ロード中のデータ列の欠落 9.18.6
    - SQL*Loaderによる拒否 7.7, 7.7.1.1, 7.7.1.2, 9.11.1
    - SQL*Loaderを使用したレコード番号への列の設定 10.15.4
    - ロード方法の指定 8.2.20, 13.3.13
    - エクスポート時の長さの指定 26.5.20
    - インポート時の長さの指定 27.7.20
    - ストリーム・レコード形式 7.4.3
- リカバリ
    - ダイレクト・パス・ロード
        - SQL*Loader 12.5.6
    - 行の置換え 9.18.2.2
- REDOログ・ファイル
    - LogMinerユーティリティ
        - サポートされるバージョン 24.14.3
- REDOログ・ファイル
    - 分析 24
    - LogMinerユーティリティの要件 24.2.1.3
    - LogMinerユーティリティに対する指定 24.4.2
- REDOログ
    - ダイレクト・パス・ロード 12.5.6.1
    - インスタンスおよびメディア・リカバリ
        - SQL*Loader 12.5.6.1
    - ダイレクト・パス・ロード中の使用の最小化 12.6.5
    - 領域の節約
        - ダイレクト・パス・ロード 12.6.5.2
- REF列 11.3

    - ロード 11.3
    - 主キー 11.3.3
    - システム生成 11.3.2
- REFデータ
    - インポート 27.16.4
- 参照整合性制約
    - インポート時の使用禁止 27.3.2
    - SQL*Loader 12.9
- リフレッシュ・エラー
    - スナップショット
        - インポート 27.18.1
- 拒否されたレコード
    - SQL*Loader 7.7, 9.11.1
- 拒否ファイル
    - SQL*Loaderへの指定 9.11.1
- 相対的なフィールド位置指定
    - フィールドの開始位置およびSQL*Loader 10.11.3
    - 複数のSQL*LoaderのINTO TABLE句 9.20.1.1
- REMAP_DATAFILEパラメータ
    - データ・ポンプ・インポート・ユーティリティ 3.4.35
- REMAP_DATAパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.38
    - データ・ポンプ・インポート・ユーティリティ 3.4.34
- REMAP_SCHEMAパラメータ
    - データ・ポンプ・インポート・ユーティリティ 3.4.37
- REMAP_TABLEパラメータ
    - データ・ポンプ・インポート・ユーティリティ 3.4.38
- REMAP_TABLESPACEパラメータ
    - データ・ポンプ・インポート・ユーティリティ 3.4.39
- リモート操作
    - エクスポートおよびインポート 26.11.2, 27.13
- REPLACE表
    - SQL*Loaderを使用した表の置換え 9.18.2.2.2
- 予約語
    - 外部表 15.8, 16.7
    - ORACLE_DATAPUMPアクセス・ドライバ 16, 16.8
    - ORACLE_LOADERアクセス・ドライバ 15, 15.9
    - SQL*Loader 7.3
- リソース消費
    - データ・ポンプ・エクスポート・ユーティリティにおける制御 5.2.1
    - データ・ポンプ・インポート・ユーティリティにおける制御 5.2.1
- リソース・エラー
    - インポート 27.10.2.3

- RESOURCEロール 27.2.2.1
- リソース使用率制限
  - データ・ポンプ・サポート 5.3.2
- 制限
  - 他のユーザーのスキーマへのインポート 27.2.2.3
  - エクスポート・パラメータ・ファイル内の表名 26.5.26.1
  - インポート・パラメータ・ファイル内の表名 2.4.45, 3.4.49, 27.7.30.1
- RESUMABLE_NAMEパラメータ
  - エクスポート・ユーティリティ 26.5.22
  - インポート・ユーティリティ 27.7.22
  - SQL*Loaderコマンドライン 8.2.29
- RESUMABLE_TIMEOUTパラメータ
  - エクスポート・ユーティリティ 26.5.23
  - インポート・ユーティリティ 27.7.23
  - SQL*Loaderコマンドライン 8.2.30
- RESUMABLEパラメータ
  - エクスポート・ユーティリティ 26.5.21
  - インポート・ユーティリティ 27.7.21
  - SQL*Loaderコマンドライン 8.2.28
- 再開可能領域割当て
  - 有効化と無効化 8.2.28, 26.5.21, 27.7.21
- オブジェクト・メタデータの取得
  - メタデータAPIの使用 25.3
- REUSE_DATAFILESパラメータ
  - データ・ポンプ・インポート・ユーティリティ 3.4.48
- REUSE_DUMPFILESパラメータ
  - データ・ポンプ・エクスポート・ユーティリティ 2.4.39
- ロール
  - DATAPUMP_EXP_FULL_DATABASE 1.4
  - DATAPUMP_IMP_FULL_DATABASE 1.4
  - EXP_FULL_DATABASE 26.2.2
  - IMP_FULL_DATABASE 27.2.1
  - RESOURCE 27.2.2.1
- ロールバック・セグメント
  - CONSISTENTエクスポート・パラメータの効果 26.5.3
- 行エラー
  - インポート 27.10.1.1
- 行形式の説明 17.4.4
- 行形式 17.4.8
- ROWID列
  - SQL*Loaderでのロード 12.2
- 行
  - SQL*Loaderを使用してロードする場合の選択 9.18.4

    - エクスポート 26.5.24
    - セーブ前に挿入する数の指定
        - SQL*Loader 12.5.5.2
    - SQL*Loaderを使用した既存の行の更新 9.18.2.2.3
- ROWSパラメータ
    - エクスポート・ユーティリティ 26.5.24
    - パフォーマンスの問題
        - SQL*Loader 12.6.4
    - SQL*Loaderコマンドライン 8.2.31
    - データ・セーブ時の指定での使用 12.5.5.2
- RUNコマンド, ADRCIユーティリティ 21.9.8

S

- SAMPLEパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.40
- スキーマ・モード・エクスポート
    - データ・ポンプ・エクスポート・ユーティリティ 2.2.2.2
- スキーマ
    - エクスポート時の指定 26.5.26
- SCHEMASパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.41
    - データ・ポンプ・インポート・ユーティリティ 3.4.40
- FLOAT EXTERNALの科学表記法 10.4.3.6
- スクリプト・ファイル
    - エクスポートおよびインポート前の実行 26.2.2, 27.2.1
- SDF
    - 「セカンダリ・データ・ファイル」を参照
- セカンダリ・データ・ファイル 7.5, 11.6.2
- SecureFiles
    - データ・ポンプのエクスポート中の暗号化 2.4.13
- SecureFiles LOB
    - エクスポートに関する考慮点 1.10
- セキュリティ上の考慮事項
    - ダイレクト・パス・エクスポート 26.10.1
- セグメント
    - 一時
        - SQL*LoaderのFILEパラメータ 12.10.7.1
- SELECTコマンド, ADRCIユーティリティ 21.9.9
    - 関数 21.9.9
- 順序ファイル 17.4.4
- 順序番号 10.15.6
- 順序番号

    - キャッシュ 26.14.1
    - エクスポート 26.14.1
    - 複数表へのロードおよびSQL*Loader 10.15.7
    - 読み込まれず生成された順序番号およびSQL*Loader 10.3
    - SQL*LoaderのSEQUENCE句による生成 10.15.6
- SerDe解析 17.4.8
- SERVICE_NAMEパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.42
    - データ・ポンプ・インポート・ユーティリティ 3.4.41
- SET BASEコマンド, ADRCIユーティリティ 21.9.10
- SET BROWSERコマンド, ADRCIユーティリティ 21.9.11
- SET CONTROLコマンド, ADRCIユーティリティ 21.9.12
- SET ECHOコマンド, ADRCIユーティリティ 21.9.13
- SET EDITORコマンド, ADRCIユーティリティ 21.9.14
- SET HOMEPATHコマンド, ADRCIユーティリティ 21.9.15
- SET TERMOUTコマンド, ADRCIユーティリティ 21.9.16
- データが欠落しているショート・レコード
    - SQL*Loader 9.18.6
- SHOW ALERTコマンド, ADRCIユーティリティ 21.9.17
- SHOW BASEコマンド, ADRCIユーティリティ 21.9.18
- SHOW CONTROLコマンド, ADRCIユーティリティ 21.9.19
- SHOW HM_RUNコマンド, ADRCIユーティリティ 21.9.20
- SHOW HOMEPATHコマンド, ADRCIユーティリティ 21.9.21
- SHOW HOMESコマンド, ADRCIユーティリティ 21.9.22
- SHOW INCDIRコマンド, ADRCIユーティリティ 21.9.23
- SHOW INCIDENTコマンド, ADRCIユーティリティ 21.9.24
- SHOW LOGコマンド, ADRCIユーティリティ 21.9.25
- SHOWパラメータ
    - インポート・ユーティリティ 27.7.25
- SHOW PROBLEMコマンド, ADRCIユーティリティ 21.9.26
- SHOW REPORTコマンド, ADRCIユーティリティ 21.9.27
- SHOW TRACEFILEコマンド, ADRCIユーティリティ 21.9.28
- SILENTパラメータ
    - SQL*Loaderコマンドライン 13.3.17
    - SQL*Loaderエクスプレス・モード 13.3.17
- シングルバイト文字セット
    - エクスポートおよびインポート 26.12.3, 27.14.3
- SINGLEROWパラメータ 9.19.2
- 単一表へのロード
    - 継続 9.16.5
- SKIP_INDEX_MAINTENANCEパラメータ
    - SQL*Loaderコマンドライン 8.2.35
- SKIP_UNUSABLE_INDEXESパラメータ

    - インポート・ユーティリティ 27.7.26
    - SQL*Loaderコマンドライン 8.2.36
- SKIP_USABLE_INDEXESパラメータ
    - データ・ポンプ・インポート・ユーティリティ 3.4.42
- SKIPパラメータ
    - SQL*Loader RECNUM指定への影響 10.15.4
    - SQL*Loaderコマンドライン 8.2.34
- 使用禁止索引のスキップ 8.2.36
- スナップショット・ログ
    - インポート 27.18.1
- スナップショット 27.18.2
    - インポート 27.18
    - マスター表
        - インポート 27.18.1
    - 削除された場合のリストア
        - インポート 27.18.2.1
- SORTED INDEXES句
    - ダイレクト・パス・ロード 9.19.1
    - SQL*Loader 12.6.3.2
- ソート
    - 複数列索引
        - SQL*Loader 12.6.3.4
    - 最適なソート順序
        - SQL*Loader 12.6.3.5
    - ダイレクト・パス・ロード時の事前ソート 12.6.3
    - SORTED INDEXES句
        - SQL*Loader 12.6.3.2
- SOURCE_EDITIONパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.43
    - データ・ポンプ・インポート・ユーティリティ 3.4.43
- ソース名 17.4.9
- SPOOLコマンド, ADRCIユーティリティ 21.9.29
- SQL*Loader
    - OSON 12.4.1
    - 表への行の追加 9.18.2.2.1
    - ダイレクト・パス・ロードの監査 12.5.9
    - BADコマンドライン・パラメータ 8.2.1
    - 不良ファイル 13.3.1
    - BADFILEパラメータ 9.11.1
    - 不良ファイル 8.2.1
    - バインド配列およびパフォーマンス 9.21.2
    - BINDSIZEコマンドライン・パラメータ 8.2.2, 9.21.3
    - ロードする行の選択 9.18.4

- COLUMNARRAYROWSコマンドライン・パラメータ 8.2.3
- コマンドライン・パラメータ 8.1
- 単一表へのロードの継続 9.16.5
- CONTROLコマンドライン・パラメータ 8.2.4
- 制御ファイル 9.1
- 従来型パス・ロード 7.9.1, 12.3
- DATAコマンドライン・パラメータ 8.2.5, 13.3.4
- データ変換 7.6
- データ定義言語
  - 構文図 A
- データ型の指定 7.6
- DATE_CACHEコマンドライン・パラメータ 8.2.6
- DEGREE_OF_PARALLELISMコマンドライン・パラメータ 8.2.8
- デフォルト・スキーマの判別 9.18.1.1
- DIRECTコマンドライン・パラメータ 8.2.9
- ダイレクト・パスによる方法 7.9.2
  - 監査 12.5.9
  - 日付キャッシュ機能を使用したパフォーマンスの向上 12.6.7
- DISCARDコマンドライン・パラメータ 8.2.11
- 廃棄されたレコード 7.7
- DISCARDFILEパラメータ 9.12.2
- DISCARDMAXコマンドライン・パラメータ 8.2.12
- DISCARDMAXパラメータ 9.12.3
- DISCARDSパラメータ 9.12.3
- DNFS_ENABLEコマンドライン・パラメータ 8.2.13
- DNFS_READBUFFERSコマンドライン・パラメータ 8.2.14
- 移植不能なDOUBLEデータ型 10.4.2.5
- タブが原因で発生するエラー 10.2.1
- ERRORSコマンドライン・パラメータ 8.2.16
- 排他的アクセス 12.9.3
- エクスプレス・モード 13
- EXTERNAL_TABLEコマンドライン・パラメータ 8.2.17
- 外部表ロード 7.9.4
- FILEコマンドライン・パラメータ 8.2.18
- ファイル名 9.4
- グローバリゼーション・テクノロジ 9.15
- 索引オプション 9.19
- 表への行の挿入 9.18.2.1
- INTO TABLE文 9.18
- 移植可能なLENGTH-VALUEデータ型 10.4.3.11
- LOADコマンドライン・パラメータ 8.2.20, 13.3.13
- 列オブジェクトのロード 11.1
- 異なるプラットフォーム間でのデータのロード 10.8

    - 制御ファイルに組み込まれたデータのロード 10.15.1
    - オブジェクト表のロード 11.2
    - ロード方法 12.1
    - LOGコマンドライン・パラメータ 8.2.21
    - ログ・ファイル 7.8
    - データをロードする方法 7.9
    - 複数のINTO TABLE文 9.20
    - MULTITHREADINGコマンドライン・パラメータ 8.2.22
    - NO_INDEX_ERRORSコマンドライン・パラメータ 8.2.23
    - オブジェクト名 9.4
    - PARALLELコマンドライン・パラメータ 8.2.24
    - パラレル・データ・ロード 12.10.1, 12.10.4, 12.11
    - 移植可能なデータ型 10.4.3
    - READSIZEコマンドライン・パラメータ 8.2.27
        - 最大サイズ 8.2.27
    - 拒否されたレコード 7.7
    - 表中の行の置換え 9.18.2.2.2
    - 必要な権限 12.1
    - RESUMABLE_NAMEコマンドライン・パラメータ 8.2.29
    - RESUMABLE_TIMEOUTコマンドライン・パラメータ 8.2.30
    - RESUMABLEコマンドライン・パラメータ 8.2.28
    - ROWSコマンドライン・パラメータ 8.2.31
    - SILENTコマンドライン・パラメータ 13.3.17
    - SINGLEROWパラメータ 9.19.2
    - SKIP_UNUSABLE_INDEXESコマンドライン・パラメータ 8.2.36
    - SKIPコマンドライン・パラメータ 8.2.34
    - ダイレクト・パス・ロードでのSORTED INDEXES 9.19.1
    - 列の指定 10.3
    - データ・ファイルの指定 9.7.1
    - フィールド条件の指定 10.5
    - フィールドの指定 10.3
    - ロード方法の指定 8.2.9
    - 複数のデータ・ファイルの指定 9.7.3
    - STREAMSIZEコマンドライン・パラメータ 8.2.37
    - メッセージの抑止 13.3.17
    - TRIMコマンドライン・パラメータ 8.2.38
    - USERIDコマンドライン・パラメータ 8.2.39, 13.3.22
- SQL*Loader制御ファイル
    - 作成時のガイドライン 7.3
- SQL*Loaderデータ型 10.4.3.4
- SQL*Loaderデータ型
    - 移植不能 10.4.2
- SQL*Loaderエクスプレス・モード 13, 13.2

    - BADパラメータ 13.3.1
    - バイト順序の処理 13.2.2
    - CHARACTERSETパラメータ 13.3.2
    - CSVパラメータ 13.3.3
    - DATAパラメータ 13.3.4
    - DATE_FORMATパラメータ 13.3.5
    - デフォルト値 13.2.1
    - DEGREE_OF_PARALLELISMパラメータ 13.3.6
    - DIRECTパラメータ 13.3.7
    - DNFS_ENABLEパラメータ 13.3.8
    - DNFS_READBUFFERSパラメータ 13.3.9
    - ENCLOSED_BYパラメータ 13.3.10
    - EXTERNAL_TABLEパラメータ 13.3.11
    - FIELD_NAMESパラメータ 13.3.12
    - LOADパラメータ 13.3.13
    - NULLIFパラメータ 13.3.14
    - OPTIONALLY_ENCLOSED_BYパラメータ 13.3.15
    - PARFILEパラメータ 13.3.16
    - SILENTパラメータ 13.3.17
    - 構文図 13.4
    - TABLEパラメータ 13.3.18
    - TERMINATED_BYパラメータ 13.3.19
    - TIMESTAMP_FORMATパラメータ 13.3.20
    - TRIMパラメータ 13.3.21
    - USERIDパラメータ 13.3.22
- SQL*Loaderユーティリティ
    - PARTITION_MEMORYパラメータ 8.2.26
- SQLFILEパラメータ
    - データ・ポンプ・インポート・ユーティリティ 3.4.44
- SQL演算子
    - フィールドへの適用 10.14
- SQL文字列
    - SQL演算子のフィールドへの適用 10.14
    - 引用符 9.4.2
- START_JOBパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ
        - 対話方式コマンド・モード 2.5.9
    - データ・ポンプ・インポート・ユーティリティ
        - 対話方式コマンド・モード 3.5.7
- 起動
    - LogMinerユーティリティ 24.5
- startパラメータ
    - field_definitions句 15.4.5.1

    - pos_spec句 15.4.5.1
- 統計
    - アナライザ 27.24
    - オプティマイザ 27.24
    - インポート時の指定 27.7.27
- STATISTICSパラメータ
    - インポート・ユーティリティ 27.7.27
- STATUSパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.44
        - 対話方式コマンド・モード 2.5.10
    - データ・ポンプ・インポート・ユーティリティ 3.4.45
        - 対話方式コマンド・モード 3.5.8
- STOP_JOBパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ
        - 対話方式コマンド・モード 2.5.11
    - データ・ポンプ・インポート・ユーティリティ
        - 対話方式コマンド・モード 3.5.9
- 記憶域パラメータ
    - エクスポート要件の見積り 26.2.3
    - OPTIMALパラメータ 27.20.1
    - 上書き
        - インポート 27.20.3
    - 事前割当て
        - ダイレクト・パス・ロード 12.6.2
    - ダイレクト・パス・ロード時の一時記憶域パラメータ 12.5.3.2
    - エクスポートおよびインポートでの使用 27.20
- ストアド・ファンクション
    - インポート 27.16.7
        - COMPILEパラメータの影響 27.16.7
- ストアド・パッケージ 27.16.7
- ストアド・パッケージ
    - インポート 27.16.7
- ストアド・プロシージャ
    - ダイレクト・パス・ロード 12.9.2.6
    - インポート 27.16.7
        - COMPILEパラメータの影響 27.16.7
- ストリーム・バッファ
    - ダイレクト・パスのサイズ指定 12.6.6
- ストリーム・レコード形式 7.4.3
    - 列オブジェクトのロード 11.1.1
- STREAMS_CONFIGURATIONパラメータ
    - データ・ポンプ・インポート・ユーティリティ 3.4.46
    - インポート・ユーティリティ 27.7.28

- STREAMS_INSTANTIATIONパラメータ
  - インポート・ユーティリティ 27.7.29
- STREAMSIZEパラメータ
  - SQL*Loaderコマンドライン 8.2.37
- 文字列の比較
  - SQL*Loader 10.5.3
- structのオーバーフロー 17.4.7
- サブパーティション表
  - ロード12.4.3
- サブタイプ
  - 複数のロード 9.20.3
- サプリメンタル・ロギング 24.10
  - 「LogMinerユーティリティ」も参照
  - LogMinerユーティリティ 24.10
    - データベース・レベルの識別キー 24.10.1.2
    - ログ・グループ 24.10
    - 表レベル 24.10.3
    - 表レベルの識別キー 24.10.3.1
    - 表レベルのログ・グループ 24.10.3.2
- シノニム
  - エクスポート 26.14.11
- 構文図
  - データ・ポンプ・エクスポート 2.7
  - データ・ポンプ・インポート 3.7
  - SQL*Loader A
  - SQL*Loaderエクスプレス・モード 13.4
- SYSDATEパラメータ
  - SQL*Loader 10.15.5
- システム生成OID型のREF列 11.3.2
- システム・オブジェクト
  - インポート 27.2.2.4
- システム・トリガー
  - インポート時の影響 27.4
  - テスト 27.4

T

- TABLE_EXISTS_ACTIONパラメータ
  - データ・ポンプ・インポート・ユーティリティ 3.4.47
- 表の圧縮
  - データ・ポンプ・ジョブでの型の指定 3.4.52
- 表レベル・エクスポート 26.4.1
- 表レベル・インポート 27.11

- 表モード・エクスポート
    - データ・ポンプ・エクスポート・ユーティリティ 2.2.2.3
    - 指定 26.5.26
- 表モード・インポート
    - 例 27.8.1
- 表名
    - 大/小文字の区別の保持 26.5.26.1
- TABLEパラメータ
    - SQL*Loaderエクスプレス・モード 13.3.18
- 表
    - アドバンスト・キューイング
        - エクスポート 26.14.10
        - インポート 27.16.10
    - SQL*Loaderを使用した行の追加 9.18.2.2.1
    - オブジェクト型を含む
        - インポート 27.16.2
    - インポート前の定義 27.3.1
    - 定義
        - インポート前の作成 27.3.1
    - ダイレクト・パス・ロード中の排他的アクセス
        - SQL*Loader 12.9.3
    - 外部 14
    - インポート 27.7.30
    - SQL*Loaderを使用した行の挿入 9.18.2.1
    - 挿入トリガー
        - SQL*Loaderでのダイレクト・パス・ロード 12.9.2
    - SQL*Loaderを使用した複数表へのデータのロード 9.20
    - オブジェクト表のロード 11.2
    - エクスポート時の一貫性の維持 26.5.3
    - 手動によるインポートの順序付け 27.3.3
    - マスター表
        - インポート 27.18.1
    - 名前の制限
        - エクスポート 26.5.26.1
        - インポート 2.4.45, 3.4.49, 27.7.30, 27.7.30.1
    - ネスト
        - エクスポート 26.14.9
        - インポート 27.16.3
    - オブジェクト
        - インポートの順序 27.1.1
    - パーティション化 26.4.1
    - SQL*Loaderを使用した行の置換え 9.18.2.2.2
    - エクスポート時の指定 26.5.26

    - 表モード・エクスポートの指定 26.5.26
    - 個々の表に対するSQL*Loaderの方法 9.18.2
    - 切捨て
        - SQL*Loader 9.18.2.2.4
    - SQL*Loaderを使用した既存の行の更新 9.18.2.2.3
- 表領域モード・エクスポート
    - データ・ポンプ・エクスポート・ユーティリティ 2.2.2.4
- 表領域
    - インポート中の削除 27.22
    - 一連のエクスポート 26.15, 27.19
    - メタデータ
        - 転送 27.7.34
    - 読取り専用
        - インポート 27.21
    - 再編成
        - インポート 27.23
- TABLESPACESパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.46
    - データ・ポンプ・インポート・ユーティリティ 3.4.50
    - エクスポート・ユーティリティ 26.5.27
    - インポート・ユーティリティ 27.7.31
- TABLESパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.45
    - データ・ポンプ・インポート・ユーティリティ 3.4.49
    - エクスポート・ユーティリティ 26.5.26
    - インポート・ユーティリティ 27.7.30
- タブ
    - タブを含むデータ・ファイルのロード 10.2.1
    - 切捨て 10.11
    - 空白 10.11
- TARGET_EDITIONパラメータ
    - データ・ポンプ・インポート・ユーティリティ 3.4.51
- 一時セグメント 12.10.7.1
    - FILEパラメータ
        - SQL*Loader 12.10.7.1
- ダイレクト・パス・ロード時の一時記憶域 12.5.3.2
- TERMINATED_BYパラメータ
    - SQL*Loaderエクスプレス・モード 13.3.19
- TERMINATED BY句
    - OPTIONALLY ENCLOSED BY付き 10.11.2.2
- 終端を指定されたフィールド
    - デリミタによる指定 10.11.2.2
- テキスト・ファイル 17.4.4

- テキストのオーバーフロー 17.4.7
- TIMESTAMP_FORMATパラメータ
  - SQL*Loaderエクスプレス・モード 13.3.20
- タイムスタンプ
  - データ・ポンプ・エクスポート操作時 2.4.30
  - データ・ポンプ・インポート操作時 3.4.24
- TOID_NOVALIDATEパラメータ
  - インポート・ユーティリティ 27.7.32
- TOUSERパラメータ
  - インポート・ユーティリティ 27.7.33
- トレース・ファイル
  - ADRCIを使用した表示 21.6
- 後続の空白
  - デリミタを使用したロード 10.4.6.4
- TRAILING NULLCOLSパラメータ
  - SQL*Loaderユーティリティ 9.1, 9.18.6.1
- 後続の空白
  - 切捨て 10.11.5
- TRANSFORMパラメータ
  - データ・ポンプ・インポート・ユーティリティ 3.4.52
- 透過的データ暗号化
  - データ・ポンプ・エクスポートによる処理 2.4.16
  - データ・ポンプ・インポートによる処理 3.4.12
  - LogMinerサポート 24.6
- TRANSPORT_DATAFILESパラメータ
  - データ・ポンプ・インポート・ユーティリティ 3.4.53
- TRANSPORT_FULL_CHECKパラメータ
  - データ・ポンプ・エクスポート・ユーティリティ 2.4.47
  - データ・ポンプ・インポート・ユーティリティ 3.4.54
- TRANSPORT_TABLESPACEパラメータ
  - エクスポート・ユーティリティ 26.5.28
  - インポート・ユーティリティ 27.7.34
- TRANSPORT_TABLESPACESパラメータ
  - データ・ポンプ・エクスポート・ユーティリティ 2.4.48
  - データ・ポンプ・インポート・ユーティリティ 3.4.55
- トランスポータブル・オプション
  - 全体モード・エクスポート時の使用 2.2.2.1
  - 全体モード・インポート時の使用 3.2.2.2
  - 表モード・エクスポート時の使用 2.4.45
- TRANSPORTABLEパラメータ
  - データ・ポンプ・エクスポート・ユーティリティ 2.4.49
  - データ・ポンプ・インポート・ユーティリティ 3.4.56
- トランスポータブル表領域モード・エクスポート

    - データ・ポンプ・エクスポート・ユーティリティ 2.2.2.5
- トランスポータブル表領域 26.15, 27.19
- トリガー
    - データベース挿入 12.9.2
    - LOB列
        - インポート 27.16.12
    - ログオン
        - SQL*Loaderでの影響 9.18.1.1
    - 永続的に使用禁止 12.9.3
    - 整合性制約への置換 12.9.2.1
    - システム
        - テスト 27.4
    - 更新
        - SQL*Loader 12.9.2.4
- TRIGGERSパラメータ
    - エクスポート・ユーティリティ 26.5.29
- 切捨て
    - サマリー 10.11
    - 後続の空白
        - SQL*Loader 10.11.5
- TRIMパラメータ
    - SQL*Loaderコマンドライン 8.2.38
    - SQL*Loaderエクスプレス・モード 13.3.21
- トラブルシューティング
    - ORA-39346 1.9
    - 移行前 20.10
- TTS_CLOSURE_CHECKパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.50
- TTS_FULL_CHECKパラメータ
    - エクスポート・ユーティリティ 26.5.30
- TTS_OWNERSパラメータ
    - インポート・ユーティリティ 27.7.35

U

- 統合監査
    - SQL*Loader操作時 12.5.9
- unionのオーバーフロー 17.4.7
- UNIQUE KEY制約
    - ダイレクト・パス・ロードへの影響 12.10.9
- 一意の値
    - SQL*Loaderを使用した生成 10.15.6
- データベース全体のアンロード

    - データ・ポンプ・エクスポート・ユーティリティ 2.2.2.1
- UNRECOVERABLE句
    - SQL*Loader 12.6.5.2
- 未ソートのデータ
    - ダイレクト・パス・ロード
        - SQL*Loader 12.6.3.3
- USER_SEGMENTSビュー
    - エクスポート 26.2.3
- ユーザー定義コンストラクタ 11.1.6
    - 列オブジェクトのロード 11.1.6
- USERIDパラメータ
    - エクスポート・ユーティリティ 26.5.31
    - インポート・ユーティリティ 27.7.36
    - SQL*Loaderコマンドライン 8.2.39, 13.3.22
    - SQL*Loaderエクスプレス・モード 13.3.22
- ユーザー・モード・エクスポート
    - 指定 26.5.17

V

- V$DATABASEビュー 24.11
- V$LOGMNR_CONTENTSビュー 24.6.1
    - 返される情報の書式設定 24.7
    - 問合せの影響 24.6.1
    - 含まれる情報 24.6
    - 返される情報の制限 24.7
    - LogMinerユーティリティ 24.2.2
    - 問合せの要件 24.5, 24.6.1
- V$LOGMNR_LOGSビュー 24.4.2
    - 問合せ 24.11.2
- V$SESSION_LONGOPSビュー
    - データ・ポンプ・ジョブの監視 1.7
- VARCHAR2データ型
    - SQL*Loader 10.4.4
- 可変長レコード
    - 外部表 15.3.3
- 可変レコード 7.4.2
    - 形式 11.1.2
- VARRAY列
    - ロード時のメモリーの問題 11.8.1
- VERSIONパラメータ
    - データ・ポンプ・エクスポート・ユーティリティ 2.4.51
    - データ・ポンプ・インポート・ユーティリティ 3.4.57

- 表示
  - ADRCIによるトレース・ファイルの表示 21.6
- ビュー
  - 表としてのエクスポート 2.4.52
- ビュー
  - インポート 27.16.13
- VIEWS_AS_TABLES
  - データ・ポンプ・エクスポート・パラメータ 2.4.52
- VIEWS_AS_TABLES (ネットワーク・インポート)
  - データ・ポンプ・インポート・パラメータ 3.4.58
- VIEWS_AS_TABLESパラメータ
  - データ・ポンプ・エクスポート・ユーティリティ 2.4.52
- VIEWS_AS_TABLESパラメータ(ネットワーク・インポート)
  - データ・ポンプ・インポート・ユーティリティ 3.4.58
- VOLSIZEパラメータ
  - エクスポート・ユーティリティ 26.5.32
  - インポート・ユーティリティ 27.7.37

W

- 警告メッセージ
  - エクスポート 26.7.2
  - インポート 26.7.2
- WHEN句
  - SQL*Loader 9.18.4, 10.5
  - SQL*Loaderの廃棄 9.12.5
- 空白
  - フィールドに含まれた 10.11.4
  - 先頭 10.11.1
  - 保持 10.12
  - フィールドの終了 10.11.4.1
  - 切捨て 10.11

X

- XML列
  - SQL*Loaderでのロード 11.4
  - SQL*Loaderによる処理 11.4
- XMLTYPE句
  - SQL*Loaderの制御ファイル 9.5
- XML型の表
  - SQL*Loaderで識別 9.5