Oracle® Database

ユーティリティ

11*g* リリース 1（11.1）

部品番号 : E05768-02

2007 年 11 月

ORACLE®

Oracle Database ユーティリティ , 11*g* リリース 1（11.1）

部品番号 : E05768-02

原本名 : Oracle Database Utilities, 11*g* Release 1 (11.1)

原本部品番号 : B28319-02

原本著者 : Kathy Rich

原本協力者 : Lee Barton、Ellen Batbouta、Janet Blowney、George Claborn、Jay Davison、Steve DiPirro、Marcus Fallen、Bill Fisher、Steve Fogel、Dean Gagne、John Galanes、John Kalogeropoulos、Jonathan Klein、Cindy Lim、Eric Magrath、Brian McCarthy、Rod Payne、Ray Pfau、Rich Phillips、Paul Reilly、Mike Sakayeda、Francisco Sanchez、Marilyn Saunders、Jim Stenoish、Carol Tagliaferri、Hailing Yu

Copyright © 1996, 2007, Oracle. All rights reserved.

制限付権利の説明

このプログラム（ソフトウェアおよびドキュメントを含む）には、オラクル社およびその関連会社に所有権のある情報が含まれています。このプログラムの使用または開示は、オラクル社およびその関連会社との契約に記された制約条件に従うものとします。著作権、特許権およびその他の知的財産権と工業所有権に関する法律により保護されています。

独立して作成された他のソフトウェアとの互換性を得るために必要な場合、もしくは法律によって規定される場合を除き、このプログラムのリバース・エンジニアリング、逆アセンブル、逆コンパイル等は禁止されています。

このドキュメントの情報は、予告なしに変更される場合があります。オラクル社およびその関連会社は、このドキュメントに誤りが無いことの保証は致し兼ねます。これらのプログラムのライセンス契約で許諾されている場合を除き、プログラムを形式、手段（電子的または機械的）、目的に関係なく、複製または転用することはできません。

このプログラムが米国政府機関、もしくは米国政府機関に代わってこのプログラムをライセンスまたは使用する者に提供される場合は、次の注意が適用されます。

U.S. GOVERNMENT RIGHTS

Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

このプログラムは、核、航空産業、大量輸送、医療あるいはその他の危険が伴うアプリケーションへの用途を目的としておりません。このプログラムをかかる目的で使用する際、上述のアプリケーションを安全に使用するために、適切な安全装置、バックアップ、冗長性（redundancy）、その他の対策を講じることは使用者の責任となります。万一かかるプログラムの使用に起因して損害が発生いたしましても、オラクル社およびその関連会社は一切責任を負いかねます。

Oracle、JD Edwards、PeopleSoft、Siebel は米国 Oracle Corporation およびその子会社、関連会社の登録商標です。その他の名称は、他社の商標の可能性があります。

このプログラムは、第三者の Web サイトへリンクし、第三者のコンテンツ、製品、サービスへアクセスすることがあります。オラクル社およびその関連会社は第三者の Web サイトで提供されるコンテンツについては、一切の責任を負いかねます。当該コンテンツの利用は、お客様の責任になります。第三者の製品またはサービスを購入する場合は、第三者と直接の取引となります。オラクル社およびその関連会社は、第三者の製品およびサービスの品質、契約の履行（製品またはサービスの提供、保証義務を含む）に関しては責任を負いかねます。また、第三者との取引により損失や損害が発生いたしましても、オラクル社およびその関連会社は一切の責任を負いかねます。

目次

2 データ・ポンプ・エクスポート

8 SQL*Loader 制御ファイル・リファレンス

9 SQL*Loader フィールド・リスト・リファレンス

10 オブジェクト、LOB およびコレクションのロード

11 従来型パス・ロードおよびダイレクト・パス・ロード

Vol.2

第 III 部　外部表

12　外部表の概要

13　ORACLE_LOADER アクセス・ドライバ

14 ORACLE_DATAPUMP アクセス・ドライバ

第 IV 部　その他のユーティリティ

15 ADRCI: ADR コマンド・インタプリタ

19 メタデータ API の使用

20 オリジナルのエクスポートおよびインポート

例一覧

図一覧

表一覧

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。この章の内容は、次のとおりです。

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 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（オリジナルおよびデータ・ポンプ版の両方）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 Oracle Database 間のデータ移動
- SQL*Loader を使用したオペレーティング・システムのファイルから Oracle 表へのデータのロード、または外部表機能を使用した外部ソースから Oracle 表へのデータのロード
- 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、データベース・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

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。

ドキュメントのアクセシビリティについて

オラクル社は、障害のあるお客様にもオラクル社の製品、サービスおよびサポート・ドキュメントを簡単にご利用いただけることを目標としています。オラクル社のドキュメントには、ユーザーが障害支援技術を使用して情報を利用できる機能が組み込まれています。HTML 形式のドキュメントで用意されており、障害のあるお客様が簡単にアクセスできるようにマークアップされています。標準規格は改善されつつあります。オラクル社はドキュメントをすべてのお客様がご利用できるように、市場をリードする他の技術ベンダーと積極的に連携して技術的な問題に対応しています。オラクル社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/accessibility/` を参照してください。

ドキュメント内のサンプル・コードのアクセシビリティについて

スクリーン・リーダーは、ドキュメント内のサンプル・コードを正確に読めない場合があります。コード表記規則では閉じ括弧だけを行に記述する必要があります。しかし JAWS は括弧だけの行を読まない場合があります。

外部 Web サイトのドキュメントのアクセシビリティについて

このドキュメントにはオラクル社およびその関連会社が所有または管理しない Web サイトへのリンクが含まれている場合があります。オラクル社およびその関連会社は、それらの Web サイトのアクセシビリティに関しての評価や言及は行っておりません。

Oracle サポート・サービスへの TTY アクセス

アメリカ国内では、Oracle サポート・サービスへ 24 時間年中無休でテキスト電話（TTY）アクセスが提供されています。TTY サポートについては、(800)446-2398 にお電話ください。アメリカ国外からの場合は、+1-407-458-2479 にお電話ください。

関連ドキュメント

詳細は、次の Oracle ドキュメントを参照してください。

特に、Oracle Database に関連するドキュメントは、次のとおりです。

- 『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』

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。

Oracle エラー・メッセージ・ドキュメントは、HTML でのみ参照可能です。Oracle ドキュメント CD のみを使用可能な場合は、範囲別にエラー・メッセージを参照できます。特定の範囲を検出後、ブラウザの「ページ内の検索」機能を使用して特定のメッセージを検索します。インターネットに接続した場合は、Oracle オンライン・ドキュメントのエラー・メッセージ検索機能を使用して、特定のエラー・メッセージを検索できます。

リリース・ノート、インストール関連ドキュメント、ホワイト・ペーパーまたはその他の関連ドキュメントは、OTN-J（Oracle Technology Network Japan）から、無償でダウンロードできます。OTN-J を使用するには、オンラインでの登録が必要です。登録は、次の Web サイトから無償で行えます。

```
http://otn.oracle.co.jp/membership/
```

すでに OTN-J のユーザー名およびパスワードを取得している場合は、次の URL で OTN-J Web サイトのドキュメントのセクションに直接接続できます。

```
http://otn.oracle.co.jp/document/
```

構文図

このマニュアルでは、様々な SQL、PL/SQL、あるいはその他のコマンド・ライン構成の構文記述に、図形式またはバッカス正規形（BNF）を使用しています。これらの記述を解釈する方法については、『Oracle Database SQL 言語リファレンス』を参照してください。

表記規則

この項では、このマニュアルで使用される表記規則について説明します。

規則	意味
太字	太字は、操作に関連付けられている Graphical User Interface あるいは本文中または用語集で定義されている用語を示します。
イタリック体	イタリック体は、特定の値を指定する必要があるプレースホルダや変数を示します。
固定幅フォント	固定幅フォントは、段落内のコマンド、URL、コード例、画面上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。

サポートおよびサービス

次の各項に、各サービスに接続するための URL を記載します。

Oracle サポート・サービス

オラクル製品サポートの購入方法、および Oracle サポート・サービスへの連絡方法の詳細は、次の URL を参照してください。

http://www.oracle.co.jp/support/

製品マニュアル

製品のマニュアルは、次の URL にあります。

http://otn.oracle.co.jp/document/

研修およびトレーニング

研修に関する情報とスケジュールは、次の URL で入手できます。

http://www.oracle.co.jp/education/

その他の情報

オラクル製品やサービスに関するその他の情報については、次の URL から参照してください。

http://www.oracle.co.jp
http://otn.oracle.co.jp

注意： ドキュメント内に記載されている URL や参照ドキュメントには、Oracle Corporation が提供する英語の情報も含まれています。日本語版の情報については、前述の URL を参照してください。

データベース・ユーティリティの新機能

ここでは、Oracle Database 11*g* ユーティリティの新機能について説明します。また、各機能の参照先も示します。旧リリースの Oracle Database で導入された機能については、そのリリースのドキュメントを参照してください。

Oracle Database 11*g* リリース 1 の新機能

この項では、Oracle Database 11*g* リリース 1（11.1）で追加された新機能を示します。

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

データ・ポンプ・エクスポート製品およびデータ・ポンプ・インポート製品については、新しい機能が追加されたことで次の操作が可能になりました。

- エクスポート時に、データとメタデータの両方、データのみまたはメタデータのみを圧縮するか、またはデータを圧縮しないかを指定できます。詳細は、2-9 ページの「COMPRESSION」を参照してください。
- 次の操作で、補足的な暗号化オプションを指定できます。
  - エクスポート時に、データとメタデータの両方、データのみまたはメタデータのみを暗号化するか、データを暗号化しないか、または暗号化された列のみを選択できます。詳細は、2-13 ページの「ENCRYPTION」を参照してください。
  - エクスポート中に使用する特定の暗号化アルゴリズムを指定できます。詳細は、2-14 ページの「ENCRYPTION_ALGORITHM」を参照してください。
  - エクスポート時の暗号化および復号化の実行に使用するセキュリティ・タイプを指定できます。たとえば、ダンプ・ファイル・セットを別のデータベースまたはリモート・データベースにインポートする場合は、転送中もセキュリティで保護する必要があります。また、Oracle Encryption Wallet 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ますが、Oracle Encryption Wallet 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がある場合もあります。詳細は、2-14 ページの「ENCRYPTION_MODE」を参照してください。
- トランスポータブル・メソッドを使用した、表モードのエクスポートおよびインポートを実行できます。エクスポートでのこの機能の使用方法の詳細は、2-36 ページの TRANSPORTABLE エクスポート・パラメータを参照してください。インポート中のこの機能の使用方法の詳細は、3-40 ページの TRANSPORTABLE インポート・パラメータを参照してください。
- インポート操作中に、パーティション化された表の処理方法を指定できます。インポートでのこのパラメータの使用方法の詳細は、3-23 ページの「PARTITION_OPTIONS」を参照してください。
- エクスポート操作中に、既存のダンプ・ファイルを上書きできます。詳細は、2-30 ページの「REUSE_DUMPFILES」を参照してください。
- インポート操作中に、表の名前を変更できます。詳細は、3-28 ページの「REMAP_TABLE」を参照してください。
- 非遅延の制約違反が起きた場合でも、データのロードを続行することを指定できます。これは、外部表によるアクセス方法を使用したインポート操作でのみ有効です。3-10 ページの DATA_OPTIONS インポート・パラメータを参照してください。
- XMLType 列に対して定義された XMLType 格納形式に関係なく、XMLType 列を非圧縮の CLOB 形式でエクスポートすることを指定できます。2-10 ページの DATA_OPTIONS エクスポート・パラメータを参照してください。
- エクスポート中に、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・ファイルの元の値をこの値に置き換えることができます。2-30 ページの REMAP_DATA エクスポート・パラメータを参照してください。
- インポート中に、新規データベースへのロード時に、データを再マップできます。3-25 ページの REMAP_DATA インポート・パラメータを参照してください。
- 同一インスタンス上のワーカーを自動再起動できます。

また、データ・ポンプでは、なんらかのエラーにより停止された（同一のインスタンス上にある）複数ワーカーを一度に自動再起動できるようになりました。たとえば、他のユーザーによりプロセスが手動で停止された場合、同一のインスタンス上のワーカーは一度に自動再起動されます。プロセスが二度目に停止された場合は、手動で再起動する必要があります。

外部表

外部表機能には、次の新機能が追加されました。

- データを圧縮してからダンプ・ファイル・セットに書き込む機能。詳細は、14-2 ページの「COMPRESSION」を参照してください。
- データを暗号化してからダンプ・ファイル・セットに書き込む機能。詳細は、14-3 ページの「ENCRYPTION」を参照してください。

LogMiner ユーティリティ

LogMiner については、次のサポートが追加されました。

- LogMiner ユーティリティは、`XMLType` のデータが `CLOB` 形式で格納されている場合に、これをサポートするようになりました。

詳細は、18-65 ページの「サポートされるデータ型と表記憶域属性」を参照してください。

自動診断リポジトリ・コマンド・インタプリタ（ADRCI）

自動診断リポジトリ・コマンド・インタプリタ（ADRCI）には、自動診断リポジトリ（ADR）内に格納される診断データを使用した作業方法が用意されています。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。ADR は、複数のインスタンスおよび複数の製品間で統一されたディレクトリ構造を使用しています。

詳細は、第 15 章「ADRCI: ADR コマンド・インタプリタ」を参照してください。

Enterprise Manager Configuration Assistant（EMCA）

Enterprise Manager Configuration Assistant（EMCA）の詳細は、このマニュアルを参照してください。Oracle Database 11*g* リリース 1 より前のリリースでは、『Oracle Enterprise Manager アドバンスト構成』に記載されていました。EMCA では、Database Control 構成のためにコマンドライン・インタフェースを使用できます。

詳細は、第 21 章「Enterprise Manager Configuration Assistant（EMCA）」を参照してください。

第 I 部

Oracle Data Pump

第 I 部には、次の章が含まれます。

- 第 1 章「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。

- 第 2 章「データ・ポンプ・エクスポート」

  この章では、Oracle Data Pump Export ユーティリティについて説明します。このユーティリティは、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、データおよびメタデータをアンロードする場合に使用します。

- 第 3 章「データ・ポンプ・インポート」

  この章では、Oracle Data Pump Import ユーティリティについて説明します。このユーティリティは、エクスポート・ダンプ・ファイル・セットをターゲット・システムにロードする場合に使用します。また、他の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ネットワーク・インポートの実行方法についても説明します。

- 第 4 章「データ・ポンプ・ユーティリティのパフォーマンス」

  この章では、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・ユーティリティが、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、パフォーマンスに優れている理由について説明します。また、エクスポートおよびインポート操作のパフォーマンスを向上させることができる特定の手順についても説明します。

- 第 5 章「データ・ポンプ API」

  この章では、データ・ポンプ API（`DBMS_DATAPUMP`）の機能について説明します。

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。

この章の内容は、次のとおりです。

データ・ポンプのコンポーネント

Oracle Data Pump は、次の 3 つの要素で構成されています。

- コマンドライン・クライアント expdp および impdp
- PL/SQL パッケージ DBMS_DATAPUMP（データ・ポンプ API とも呼ばれます）
- PL/SQL パッケージ DBMS_METADATA（メタデータ API とも呼ばれます）

データ・ポンプ・クライアントである expdp および impdp は、それぞれデータ・ポンプ・エクスポート・ユーティリティおよびデータ・ポンプ・インポート・ユーティリティを起動します。これらのユーティリティでは、オリジナルのエクスポート・ユーティリティ（exp）およびインポート・ユーティリティ（imp）とほぼ同じユーザー・インタフェースが提供されます。

注意： データ・ポンプ・エクスポート・ユーティリティで生成されるダンプ・ファイルと、オリジナルのエクスポート・ユーティリティで生成されたダンプ・ファイルには互換性がありません。そのため、オリジナルのエクスポート・ユーティリティ（exp）で生成されたファイルは、データ・ポンプ・インポート（impdp）ユーティリティではインポートできません。

ほとんどの場合、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・ユーティリティを使用することをお薦めします。これらの使用では、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すると、データ移動のパフォーマンスが向上します。

オリジナルのエクスポートおよびインポート・ユーティリティを使用する必要がある場合については、第 20 章「オリジナルのエクスポートおよびインポート」を参照してください。

---

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、操作および再送信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。

注意： ダンプ・ファイルの読取りおよび書込みを含むすべてのデータ・ポンプ・エクスポートおよびデータ・ポンプ・インポートの処理は、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。**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。ディレクトリ・オブジェクトの詳細は、1-10 ページの「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。

注意： フィジカル・スタンバイ・データベースまたはロジカル・スタンバイ・データベースで、データ・ポンプ・エクスポートおよびデータ・ポンプ・インポートはサポートされていません（ロジカル・スタンバイでの最初の表のインスタンス化を除く）。

参照： DBMS_DATAPUMP および DBMS_METADATA パッケージの詳細は、『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』を参照してください。

データ・ポンプによるデータの移動方法

データ・ポンプでは、データベースの内外へのデータ移動に、4 つのメカニズムを使用します。次に、それらを速度の速い順に示します。

- データ・ファイル・コピー
- ダイレクト・パス
- 外部表
- ネットワーク・リンク・インポート

注意： 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必要がある場合は、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。

注意： まれに、データ・ポンプで、ダイレクト・パスまたは外部表を使用してもデータを表にロードできない場合があります。これは、表属性の競合が存在する場合に起こります。たとえば、表に、（ダイレクト・パスによるアクセス方法を必要とする）LONG データ型の列が含まれているにもかかわらず、ダイレクト・パスによるアクセスの使用を禁止する条件も存在する場合、競合が発生します。このような場合、ORA-39242 エラー・メッセージが生成されます。この問題を解決するには、インポート前に、LONG 列でなく LOB 列を含む表を作成します。これにより、インポートを実行でき、APPEND または TRUNCATE のいずれかの値とともに TABLE_EXISTS_ACTION パラメータを使用できます。

次の項では、これらのデータ移動の各メカニズムの使用方法と、どの場合に使用するかを簡単に説明します。

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。この方法は、次のような場合に使用します。

- トランスポータブル・モード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。
- 表モードのエクスポート（TABLES 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。TABLES パラメータで指定した表、パーティションおよび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ます。

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は、データ・ファイルとエクスポート・ダンプ・ファイルの両方がロードされます。

データ・ファイル・コピーの使用によるデータ移動では、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キャラクタ・セットが同一である必要があります。したがって、データをコピーするのに加え、Recovery Manager（RMAN）の CONVERT コマンドを使用していくつかのデータ変換を実行し、準備することが必要な場合があります。これは、通常、ソース・データベースまたはターゲット・データベースのいずれかで実行できます。

参照：

- Recovery Manager の 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。

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のに、表の構造上可能な場合は、自動的にダイレクト・パスによる方法が使用されます。表に LONG データ型の列が含まれる場合は、ダイレクト・パスを使用する必要があります。

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による方法が使用されます。

- 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。これには、パーティション化されたオブジェクト表も含まれる。
- LOB 列のドメイン索引が存在する。
- 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。

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による方法が使用されます。

- 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。
- 表に最新ではない LONG 型または LONG RAW 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。

外部表を使用したデータ移動

データ・ファイル・コピーが選択されず、ダイレクト・パスでデータを移動できない場合、外部表によるメカニズムが使用されます。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、インポート時に APPEND ヒントが使用されて、データベースへのデータのコピーが高速化されます。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。したがって、データ・ポンプでは、エクスポート時にはダイレクト・パスによるメカニズムが使用されても、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が使用される場合があります。同様に、データ・ポンプでは、エクスポートに外部表、インポートにダイレクト・パスが使用される場合もあります。

特に、データ・ポンプでは、次のような場合に外部表が使用されます。

- パラレル SQL を効果的に使用できる状況で、非常に大きい表や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
- ロード時およびアンロード時に別の方法でパーティション化される表をロードする場合

注意： データ・ポンプがデータ・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、外部表を使用する際にデータ・ポンプで作成されるファイルは、ユーザーが SQL CREATE TABLE ... ORGANIZATION EXTERNAL 文を使用して外部表を手動で作成する際に作成されるファイルとは互換性がないことに注意してください。互換性がない一因としては、手動で作成される外部表は、データのみ（メタデータを含まず）をアンロードするのに対して、データ・ポンプでは、関連するすべてのオブジェクトについて、データとメタデータの両方の情報が保持されることが挙げられます。

参照： 第 14 章「ORACLE_DATAPUMP アクセス・ドライバ」

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、外部表による方法の変形が使用されます。この場合、データは、指定したネットワーク・リンク全体から選択され、外部表を使用してダンプ・ファイルに挿入されます。

参照：

- NETWORK_LINK エクスポート・パラメータの使用方法の詳細は、2-25 ページの「NETWORK_LINK」を参照してください。
- 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。

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・リンクの指定に NETWORK_LINK インポート・パラメータが使用されている場合、SQL が直接使用され、INSERT SELECT 文によりデータが移動されます。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。

データベース・リンクを介してエクスポートを実行すると、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・ファイルに書き込まれます。ソース・データベースは、読取り専用データベースである場合があります。

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。

ネットワーク全体からの読取りは、通常、ディスクからの読取りよりも時間がかかるため、ネットワーク・リンクは、データ・ポンプで使用する 4 つのアクセス方法のうちで最も低速です。大規模なジョブへの使用はお薦めしません。

サポートされるリンク・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（パブリックおよび共有の両方）
- 固定ユーザー
- 接続ユーザー

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、現在のユーザーのデータベース・リンク・タイプの使用はサポートされません。

参照：

- 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、2-25 ページの NETWORK_LINK エクスポート・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、3-20 ページの NETWORK_LINK インポート・パラメータを参照してください。
- データベース・リンクの詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

データ・ポンプ・ジョブ実行中に行われる処理

データ・ポンプ・ジョブでは、マスター表、マスター・プロセス、ワーカー・プロセスを使用して、処理が実行され、進捗状況が追跡されます。

ジョブの調整

すべてのデータ・ポンプ・エクスポート・ジョブおよびデータ・ポンプ・インポート・ジョブに対して、マスター・プロセスが作成されます。マスター・プロセスによって、ジョブ全体（クライアントとの通信、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。

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、ジョブ内の進捗状況の追跡にマスター表が使用されます。マスター表は、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・ジョブ用のマスター表固有の機能は、次のとおりです。

- エクスポート・ジョブの場合、マスター表には、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、マスター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、マスター表の内容がダンプ・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、マスター表は、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。

マスター表は、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マスター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。マスター表の名前は、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、データ・ポンプ・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、ジョブの再起動にマスター表の情報が使用されます。

マスター表は、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、マスター表は削除されます。
- 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マスター表は保持されます。
- 対話方式コマンド KILL_JOB を使用してジョブを中断すると、マスター表は削除され、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、マスター表は保持されます。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。
- ジョブが実行を開始する前（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、マスター表は削除されます。

> **参照：** ジョブ名の形式の詳細は、2-24 ページの「JOB_NAME」を参照してください。

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

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。

参照：

- 「エクスポート操作中のフィルタ処理」（2-6 ページ）
- 「インポート操作中のフィルタ処理」（3-6 ページ）

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場合は、表領域間で記憶域を再マップしたり、特定のオブジェク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と有効です。これは、データ・ポンプ・インポート・パラメータの REMAP_DATAFILE、REMAP_SCHEMA、REMAP_TABLE、REMAP_TABLESPACE、TRANSFORM および PARTITION_OPTIONS を使用して実行されます。

ジョブ・パフォーマンスの最大化

データ・ポンプでは、複数のワーカー・プロセスを採用し、それらをパラレルに実行して、ジョブのパフォーマンスを向上させることができます。PARALLEL パラメータを使用して、現状の環境で最大限の効果を得る並列度を設定します。たとえば、本番システムへのジョブの影響を制限する場合、データベース管理者（DBA）は並列度を制限する必要があります。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、運用時間中は特定のジョブの並列度が 2 に制限されるように PARALLEL を 2 に設定し、非運用時間中は 8 に再設定することができます。並列度の設定は、マスター・プロセスによって施行され、マスター・プロセスによって、1 回の操作でデータおよびメタデータの処理を実行するワーカー・プロセスに、実行対象の処理が割り当てられます。これらのワーカー・プロセスは、パラレルで動作します。通常、並列度は、インスタンス上の CPU の数の 2 倍を超えないように設定してください。

注意： 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。

データのロードおよびアンロード

ワーカー・プロセスとは、メタデータおよび表データを実際にパラレルでロードおよびアンロードするプロセスです。ワーカー・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。アクティブなワーカー・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。

注意： 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ワーカー・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、外部表によるアクセス方法が使用されることがあります。その場合、ワーカー・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処理は、パラレル I/O の実行プロセスの Oracle RAC 全体のプールから割り当てられたパラレル I/O の実行プロセス（スレーブとも呼ばれる）間で分割されます。

参照：

- PARALLEL エクスポート・パラメータの詳細は、2-27 ページを参照してください。
- PARALLEL インポート・パラメータの詳細は、3-22 ページを参照してください。

ジョブの状態の監視

データ・ポンプ・エクスポートおよびインポート・ユーティリティでは、対話方式コマンド・モードまたはロギング・モードのいずれでも、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、エクスポート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。

参照：

- コマンドライン・エクスポートでの状態表示の頻度を変更する方法については、2-43 ページの STATUS エクスポート・パラメータを参照してください。
- コマンドライン・インポートでの状態表示の頻度を変更する方法については、3-32 ページの STATUS インポート・パラメータを参照してください。

ジョブの状態は、対話方式コマンド・モードでのリクエストで表示できます。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。

参照：

- 対話方式の STATUS エクスポート・コマンドの詳細は、2-43 ページを参照してください。
- 対話方式の STATUS インポート・コマンドの詳細は、3-46 ページを参照してください。

オプションで、ジョブの実行中にログ・ファイルの書込みを行うこともできます。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、処理中に発生したエラーがリストされ、ジョブの完了状態が記録されます。

参照：

- エクスポート・ログ・ファイルのファイル指定を設定する方法については、2-24 ページの LOGFILE エクスポート・パラメータを参照してください。
- インポート・ログ・ファイルのファイル指定を設定する方法については、3-19 ページの LOGFILE インポート・パラメータを参照してください。

ジョブの状態を判断したり、データ・ポンプ・ジョブについての情報を表示するには、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方法もあります。これらのビューの説明は、『Oracle Database リファレンス』を参照してください。

ジョブの実行状況の監視

表データの転送を行うデータ・ポンプ操作（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メガバイト数）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。

COMPRESSION、ENCRYPTION、ENCRYPTION_ALGORITHM、ENCRYPTION_MODE、ENCRYPTION_PASSWORD、QUERY、REMAP_DATA、および SAMPLE パラメータの使用は、推定値の決定に反映されません。

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: ジョブ実行中に転送されたメガバイト数（MB）
- TOTALWORK: ジョブ内の推定メガバイト数（MB）
- UNITS : MB
- MESSAGE: 状態メッセージ。次の形式で表示されます。

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

ファイルの割当て

データ・ポンプ・ジョブは、次のタイプのファイルを管理します。

- 移動中のデータおよびメタデータを格納するダンプ・ファイル
- 操作に関連したメッセージを記録するログ・ファイル
- SQLFILE 操作の出力を記録する SQL ファイル。SQLFILE 操作は、データ・ポンプ・インポートの SQLFILE パラメータを使用して起動され、他のパラメータに基づいて Import が実行される SQL DDL として、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

データ・ポンプでのこれらのファイルの割当て方法および処理方法を理解すると、エクスポート・ユーティリティおよびインポート・ユーティリティを最大限に利用できます。

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、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。たとえば、エクスポート操作中に領域が不足した場合は、データ・ポンプ・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。

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

データ・ポンプは、クライアント・ベースではなく、サーバー・ベースであるため、ダンプ・ファイル、ログ・ファイルおよび SQL ファイルには、サーバー・ベースのディレクトリ・パスを基準としてアクセスします。データ・ポンプでは、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。

たとえば、次の SQL 文は、/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します。

```
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。次に例を示します。

- 入力ファイルのディレクトリ・パスの位置を指定する権限を付与された場合、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場合があります。
- 出力ファイルのディレクトリ・パスの位置を指定する権限を付与された場合、通常、ユーザーが削除権限を持たないファイルが、サーバーによって上書きされる場合があります。

UNIX および Windows NT 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、データベースが作成されるとき、またはデータベース・ディレクト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、特権ユーザーのみが使用できます。

権限のないユーザーの場合、データ・ポンプのエクスポート・ユーティリティまたはインポート・ユーティリティを実行できるようにするには、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 で指定されたディレクトリのユーザー hr のかわりに、Oracle Database がファイルを読取りまたは書込みできるようにするには、DBA が次のコマンドを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、Oracle Database によってそのファイルの読取りまたは書込みのみが実行されることを意味します。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。

データ・ポンプ・エクスポートおよびインポート・ユーティリティでは、次の順序でファイルの位置が判断されます。

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. ファイルにディレクトリ・オブジェクトが指定されていない場合は、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・ユーティリティが実行されるクライアント・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されます。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';
   ```

   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 は、ログ・ファイル export.log と同様、'/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・オブジェクトの位置を判断できない場合、特権ユーザーであれば、データ・ポンプ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、データベースが作成されるとき、またはデータベース・ディレクトリがアップグレードされるたびに自動的に作成されます。DATA_PUMP_DIR のパス定義は、次の SQL 問合せを使用して確認できます。

```
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。

自動ストレージ管理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

自動ストレージ管理（ASM）を使用可能にしてデータ・ポンプ・エクスポート・ユーティリティまたはインポート・ユーティリティを使用する場合は、オペレーティング・システムのディレクトリ・パスではなく ASM ディスク・グループ名が使用されるように、ダンプ・ファイルで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。オペレーティング・システムのディレクトリ・パスを示す、別のディレクトリ・オブジェクトがログ・ファイルに使用される必要があります。たとえば、ASM ダンプ・ファイルのディレクトリ・オブジェクトは、次のように作成します。

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

ログ・ファイル用の個別のディレクトリ・オブジェクトは、次のように作成します。

```
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、次に示すように、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;
```

この後に、次のデータ・ポンプ・エクスポート・ユーティリティのコマンドを使用します（パスワードを入力するように要求されます）。

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

参照：

- DIRECTORY エクスポート・パラメータの詳細は、2-11 ページを参照してください。
- DIRECTORY インポート・パラメータの詳細は、3-10 ページを参照してください。
- 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。
- 自動ストレージ管理（ASM）の詳細は、『Oracle Database 管理者ガイド』を参照してください。

並列度の設定

エクスポートおよびインポート操作では、並列度を、ダンプ・ファイル・セット内のダンプ・ファイル数以下に設定（PARALLEL パラメータで指定）する必要があります。ダンプ・ファイル数が不足すると、複数の実行スレッドが同じダンプ・ファイルへアクセスしようとするため、最適なパフォーマンスは得られません。

PARALLEL パラメータは、Oracle Database の Enterprise Edition でのみ有効です。

置換変数の使用方法

エクスポート操作で、特定のファイル名を指定するかわりに、または指定したうえで、ファイル名に置換変数（%U）を使用して DUMPFILE パラメータで複数のダンプ・ファイルを指定できます。これは、ダンプ・ファイル・テンプレートと呼ばれます。新しいダンプ・ファイルは、01（%U に対応）から 02、03 の順で、必要に応じて作成されます。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。FILESIZE パラメータで指定した最大サイズに達したためにダンプ・ファイルが一杯になると、ダンプ・ファイルはクローズされ、新しく生成された名前を持つ新しいダンプ・ファイルが、それにかわるファイルとして作成されます。

複数のダンプ・ファイル・テンプレートが提供されている場合、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 6 のジョブに、expa%U、expb%U および expc%U がすべて指定された場合、expa01.dmp、expb01.dmp、expc01.dmp、expa02.dmp、expb02.dmp および expc02.dmp が初期ダンプ・ファイルとして作成されます。

インポートおよび SQLFILE 操作では、ダンプ・ファイル指定 expa%U、expb%U および expc%U を指定した場合、ダンプ・ファイル expa01.dmp、expb01.dmp および expc01.dmp をオープンすると操作が開始されます。マスター表は複数のダンプ・ファイルにまたがることができます。したがって、マスター表のすべての部分が検出されるまで、ダンプ・ファイルは、置換変数の増加および新しいファイル名（expa02.dmp、expb02.dmp、expc02.dmp など）の検索を行うことによって、オープン状態を継続します。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 および expb02.dmp は検出され、expb03.dmp は検出されなかった場合、expb%U 指定を使用したファイルの検索は中止されます。マスター表全体が検出されると、その表を使用して、ダンプ・ファイル・セット内のすべてのダンプ・ファイルの位置が特定されているかどうかを確認できます。

データベース・バージョンが異なる場合のデータ移動

ほとんどのデータ・ポンプ操作はサーバー側で実行されるため、COMPATIBLE 以外のバージョンのデータベースを使用する場合は、そのバージョン情報をサーバーに提供する必要があります。指定しない場合は、エラーが発生することがあります。バージョン情報を指定するには、VERSION パラメータを使用します。

参照：

- VERSION エクスポート・パラメータの詳細は、2-37 ページを参照してください。
- VERSION インポート・パラメータの詳細は、3-41 ページを参照してください。

データ・ポンプのエクスポートおよびインポートを使用して、バージョンが異なるデータベース間でデータを移動する場合は、次の点に注意してください。

- 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場合は、特定の機能を使用できないことがあります。たとえば、VERSION=10.1 を指定した場合に、ジョブに対してデータ圧縮も指定してあるとエラーが発生します（データ圧縮は、10.1 ではサポートされていません）。

- データ・ポンプのエクスポートで、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。ただし、そのダンプ・ファイル・セットには、古いデータベース・バージョンでサポートされていないオブジェクトは含まれません。たとえば、バージョン 10.2 のデータベースからバージョン 10.1 のデータベースにエクスポートした場合、索引タイプに関するコメントはダンプ・ファイル・セットにエクスポートされません。
- データ・ポンプ・インポートは、常に、古いバージョンのデータベースで作成されたダンプ・ファイル・セットを読み込むことができます。
- データ・ポンプ・インポートは、現在のデータベース・バージョンより新しいデータベース・バージョンで作成されたダンプ・ファイル・セットを読み込むことができません（ただし、読み込むダンプ・ファイル・セットが、バージョン・パラメータをターゲット・データベースのバージョンに設定して作成されている場合は除きます）。したがって、ダウングレードを行う場合は、VERSION パラメータをターゲット・データベースのバージョンに設定して、データ・ポンプのエクスポートを実行するようにしてください。
- 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、リモート・データベースのバージョンは、ローカル・データベースのバージョンと同じか、1 つ前までのバージョンであることが必要です。たとえば、ローカル・データベースがバージョン 11.1 の場合、リモート・データベースのバージョンは 10.2 または 11.1 であることが必要です。

2

データ・ポンプ・エクスポート

この章では、Oracle Data Pump Export ユーティリティについて説明します。この章の内容は、次のとおりです。

データ・ポンプ・エクスポート・ユーティリティとは

> **注意：** データ・ポンプ・エクスポート・ユーティリティ（expdp）の機能は、オリジナルのエクスポート・ユーティリティ（exp）の機能と類似していますが、これらは完全に別のユーティリティであり、それぞれのファイルには互換性がありません。オリジナルのエクスポート・ユーティリティの詳細は、第 20 章「オリジナルのエクスポートおよびインポート」を参照してください。

データ・ポンプ・エクスポート（以降、エクスポート・ユーティリティと呼びます）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・ファイルにデータおよび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、データ・ポンプ・インポート・ユーティリティによってのみインポートできます。ダンプ・ファイル・セットは、同一システムでインポートするか、または別のシステムへ移動してそのシステムにロードできます。

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。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。

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、データベース管理者（DBA）は、ディレクトリ・オブジェクトを作成する必要があります。ディレクトリ・オブジェクトの詳細は、1-10 ページの「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。

データ・ポンプ・エクスポート・ユーティリティでは、データおよびメタデータのサブセットが、エクスポート・モードで設定されているとおりにジョブによって移動されるように指定できます。この指定には、エクスポート・パラメータで指定するデータ・フィルタおよびメタデータ・フィルタが使用されます。詳細は、2-6 ページの「エクスポート操作中のフィルタ処理」を参照してください。

データ・ポンプ・エクスポートを使用できる様々な方法の例については、2-45 ページの「データ・ポンプ・エクスポートの使用例」を参照してください。

データ・ポンプ・エクスポートの起動

データ・ポンプ・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。エクスポート操作の特性は、指定するエクスポート・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。

> **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。

エクスポート・ユーティリティの起動の詳細は、次の項を参照してください。

- 「データ・ポンプ・エクスポートのインタフェース」（2-3 ページ）
- 「データ・ポンプ・エクスポートのモード」（2-3 ページ）
- 「ネットワークに関する考慮点」（2-5 ページ）

> **注意：** データ・ポンプ・ジョブが Oracle Real Application Clusters（RAC）環境のあるインスタンス上で実行されている場合、その Oracle RAC 環境の他のインスタンス上でデータ・ポンプ・ジョブを起動および再起動することはできません。

データ・ポンプ・エクスポートのインタフェース

データ・ポンプ・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。

- 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。コマンドライン・インタフェースで選択可能なパラメータの詳細は、2-7 ページの「エクスポート・ユーティリティのコマンドライン・モードで使用可能なパラメータ」を参照してください。
- 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。詳細は、2-8 ページの「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。
- 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。

  対話方式コマンド・モードで使用可能なコマンドの詳細は、2-40 ページの「エクスポート・ユーティリティの対話方式コマンド・モードで使用可能なコマンド」を参照してください。

データ・ポンプ・エクスポートのモード

エクスポート・ユーティリティには、データベース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。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。使用可能なモードは次のとおりです。

- 「全体エクスポート・モード」(2-3 ページ)
- 「スキーマ・モード」(2-4 ページ)
- 「表モード」(2-4 ページ)
- 「表領域モード」(2-4 ページ)
- 「トランスポータブル表領域モード」(2-5 ページ)

注意 : 多数のシステム・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システム・スキーマの例としては、SYS、ORDSYS、MDSYS などがあります。

参照 : 「データ・ポンプ・エクスポートの使用例」(2-45 ページ)

全体エクスポート・モード

全体エクスポートは、FULL パラメータを使用して指定します。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、EXP_FULL_DATABASE ロールが必要です。

参照 : エクスポート・ユーティリティの FULL パラメータの詳細は、2-21 ページの「FULL」を参照してください。

スキーマ・モード

スキーマ・エクスポートは、SCHEMAS パラメータを使用して指定します。これがデフォルトのエクスポート・モードです。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。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。これは、指定されたスキーマ内の表に影響を及ぼす外部型定義についても同様です。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。

> **参照：** エクスポート・ユーティリティの SCHEMAS パラメータの詳細は、2-32 ページの「SCHEMAS」を参照してください。

表モード

表モード・エクスポートは、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アンロードされます。

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。異なるバージョン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）で処理することが必要な場合があります。

> **参照：** プラットフォーム間でのデータ転送の詳細は、『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。

自分のスキーマに存在しない表を指定するには、EXP_FULL_DATABASE ロールが必要です。指定するすべての表が、単一のスキーマ内に存在する必要があります。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。

> **参照：**
>
> - エクスポート・ユーティリティの TABLES パラメータの詳細は、2-33 ページの「TABLES」を参照してください。
> - エクスポート・ユーティリティの TRANSPORTABLE パラメータの詳細は、2-36 ページの「TRANSPORTABLE」を参照してください。

表領域モード

表領域エクスポートは、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。

> **参照：**
>
> - エクスポート・ユーティリティの TABLESPACES パラメータの詳細は、2-34 ページの「TABLESPACES」を参照してください。

トランスポータブル表領域モード

トランスポータブル表領域エクスポートは、TRANSPORT_TABLESPACES パラメータを使用して指定します。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。

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。

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。

注意： 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。

参照：

- 「TRANSPORT_FULL_CHECK」（2-35 ページ）
- 「TRANSPORT_TABLESPACES」（2-35 ページ）
- 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。

ネットワークに関する考慮点

データ・ポンプ・エクスポート・ユーティリティの起動時、接続文字列には接続識別子を指定できます。この識別子では、現行の Oracle システム識別子（SID）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名前を指定できます。これには、接続記述子を使用して検索できるアクティブ・リスナー（起動するには、lsnrctl start と入力）が必要です。次に、ユーザー hr が inst1 という接続記述子を使用してエクスポート・ユーティリティを起動する例を示します。

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

Export: Release 11.1.0.6.0 - Production on Monday, 27 August, 2007 10:15:45

Copyright (c) 2003, 2007, Oracle.  All rights reserved.

Password: password@inst1

Connected to: Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, Data Mining and Real Application Testing options
```

ローカルのエクスポート・クライアントは、接続記述子 inst1（通常は tnsnames.ora ファイルで定義される単純なネット・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、そのインスタンスのデータをエクスポートします。

接続識別子を使用したエクスポート・ユーティリティの起動と、エクスポート・ユーティリティのコマンドライン・パラメータ NETWORK_LINK を指定したエクスポート操作を混同しないでください。エクスポートを実行して NETWORK_LINK パラメータを使用すると、データベース・リンクを介してエクスポートが開始されます。一方、エクスポート操作を開始して接続識別子を指定すると、ローカルのエクスポート・クライアントは、コマンドライン接続文字列によって指定されるデータベース・インスタンスに接続し、データベース・リンクによって指定されるデータベース・インスタンスからエクスポートするデータを取得して、接続したデータベース・インスタンスのダンプ・ファイル・セットにそのデータを書き込みます。

参照：

- 「NETWORK_LINK」（2-25 ページ）
- 『Oracle Database Net Services 管理者ガイド』
- 『Oracle Database Heterogeneous Connectivity 管理者ガイド』

エクスポート操作中のフィルタ処理

データ・ポンプ・エクスポート・ユーティリティで提供されるデータおよびメタデータのフィルタ機能は、オリジナルのエクスポート・ユーティリティと比較すると大幅に拡張されています。

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、表のエクス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結果として、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。

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。

フィルタ処理できるオブジェクトを確認するには、DATABASE_EXPORT_OBJECTS（全体モードのエクスポートの場合）、SCHEMA_EXPORT_OBJECTS（スキーマ・モードのエクスポートの場合）および TABLE_EXPORT_OBJECTS（表モードおよび表領域モードのエクスポートの場合）の各ビューに対して問合せを実行します。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
```

参照： 2-18 ページの「EXCLUDE」および 2-22 ページの「INCLUDE」

エクスポート・ユーティリティのコマンドライン・モードで使用可能なパラメータ

この項では、データ・ポンプ・エクスポート・ユーティリティのコマンドライン・モードで使用可能なパラメータについて説明します。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。

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・ユーティリティが起動され、データベースとの接続が行われる前にパスワードの入力が要求されます。

  ```
  Export: Release 11.1.0.6.0 - Production on Monday, 27 August, 2007 11:45:35

  Copyright (c) 2003, 2007, Oracle.  All rights reserved.

  Password: password

  Connected to: Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 -
  Production
  With the Partitioning, Data Mining and Real Application Testing options
  ```

- 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 スキーマに付与されているものとします。ディレクトリ・オブジェクトの作成およびこれらへの権限の割当てについては、1-10 ページの「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。
- 一部の例で、EXP_FULL_DATABASE ロールおよび IMP_FULL_DATABASE ロールが必要です。このような例では、hr スキーマ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。

これらのパラメータの構文図は、2-47 ページの「データ・ポンプ・エクスポートの構文図」を参照してください。

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。

データ・ポンプ・コマンドラインでの引用符の使用

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必要がある場合があります。バックスラッシュがない場合、Export で使用するコマンドライン解析機能で引用符として認識されないため、引用符が削除されエラーが発生します。通常、そのような文は、パラメータ・ファイルに記述することをお薦めします。パラメータ・ファイルでは、エスケープ文字は不要なためです。

参照：

- デフォルトのディレクトリ・オブジェクトの作成については、1-10 ページの「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。
- 「データ・ポンプ・エクスポートの使用例」（2-45 ページ）
- 『Oracle Database サンプル・スキーマ』

注意： オリジナルのエクスポート・ユーティリティ（exp）を使い慣れている場合、オリジナルのインポート・ユーティリティと同様の操作の実行に使用するデータ・ポンプ・パラメータを特定できないことがあります。両者の対応関係は、2-38 ページの「オリジナルのエクスポート・ユーティリティのパラメータへのデータ・ポンプ・エクスポート・パラメータのマップ方法」を参照してください。

ATTACH

デフォルト：ユーザーのスキーマで現在実行されているジョブ（存在する場合）

用途

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・インタフェースを有効にします。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、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。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
```

参照： 「エクスポート・ユーティリティの対話方式コマンド・モードで使用可能なコマンド」(2-40 ページ)

COMPRESSION

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。
- 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。
- 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。これがデフォルトです。
- 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。

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。

CONTENT

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。これがデフォルトです。
- 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ます。表の行データはアンロードされません。

制限事項

- パラメータ CONTENT=METADATA_ONLY は、パラメータ TRANSPORT_TABLESPACES（トランスポータブル表領域モード）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場合は、デフォルトで hr のスキーマのスキーマ・モード・エクスポートになります。

DATA_OPTIONS

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 パラメータを使用すると、エクスポートおよびインポート中に特定のタイプのデータを処理するためのオプションが提供されます。エクスポート操作の場合、DATA_OPTIONS パラメータに対して唯一有効なオプションは XML_CLOBS です。

構文および説明

```
DATA_OPTIONS=XML_CLOBS
```

XML_CLOBS オプションを指定すると、XMLType 列は、列に対して定義されている XMLType の格納形式に関係なく非圧縮の CLOB 形式でエクスポートされます。

表に CLOB として格納されている XMLType 列しかない場合は、データ・ポンプにより自動的にこれらの列が CLOB 形式でエクスポートされるため、XML_CLOBS オプションを指定する必要はありません。表にオブジェクト・リレーショナル（スキーマベース）、バイナリまたは CLOB 形式の組合せで格納されている XMLType 列がある場合、デフォルトで、これらの列はデータ・ポンプにより圧縮形式でエクスポートされます。この方法をお薦めします。ただし、データを非圧縮の CLOB 形式でエクスポートする必要がある場合は、XML_CLOBS オプションを使用してデフォルトの設定を上書きできます。

> **参照 :** XMLType 表のエクスポートおよびインポートに固有の情報については、『Oracle XML DB 開発者ガイド』を参照してください。

制限事項

- XML_CLOBS オプションを使用するには、エクスポート時とインポート時の両方で同じ XML スキーマを使用する必要があります。
- 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。詳細は、2-37 ページの「VERSION」を参照してください。

例

この例では、hr.xdb_tab1 表の XMLType 列を、列に対して定義されている XMLType の格納形式に関係なく、非圧縮の CLOB 形式でエクスポートするエクスポート操作を示します。

```
> expdp hr TABLES=hr.xdb_tab1 DIRECTORY=dpump_dir1
DUMPFILE=hr_xml.dmp VERSION=11.1 DATA_OPTIONS=xml_clobs
```

DIRECTORY

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、データベースのディレクトリ・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）。インストール時に、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 へのアクセス権を持つユーザーが 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
```

ダンプ・ファイル employees.dmp は、ディレクトリ・オブジェクト dpump_dir1 に対応付けられたパスに書き込まれます。

参照 :

- デフォルトのディレクトリ・オブジェクトの詳細は、1-10 ページの「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。
- 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。

DUMPFILE

デフォルト：expdat.dmp

用途

名前を指定します。オプションで、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 パラメータで指定されている場合、*directory_object* はオプションです。ここで値を指定する場合は、すでに存在しアクセス権があるディレクトリ・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、複数のファイルを生成できることを示す置換変数（%U）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 ～ 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。

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（バイト）を上限とするため、拡張できなくなります。置換変数（%U）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 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・ユーティリティによって作成されます。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。

制限事項

- 作成されるダンプ・ファイルの名前がすでに存在しているダンプ・ファイルの名前と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。この動作を変更するには、エクスポート・ユーティリティのパラメータ REUSE_DUMPFILES=Y を指定します。

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。このジョブでは、3 つのパラレル処理がすべて実行されるため、ダンプ・ファイル exp201.dmp および exp202.dmp が作成され、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 に対応するパスに書き込まれます。

参照：

- 「ファイルの割当て」(1-10 ページ)

ENCRYPTION

デフォルト：デフォルト値は、使用される暗号化関連のパラメータの組合せによって決まります。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 と ENCRYPTION_PASSWORD のどちらも指定されていない場合は、デフォルトで ENCRYPTION が NONE に設定されます。

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。

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。

注意： 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・ポンプの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。

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 11*g* の Enterprise Edition でのみ有効です。

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
```

ENCRYPTION_ALGORITHM

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 }
```

暗号化アルゴリズムの詳細は、『Oracle Advanced Security 管理者ガイド』を参照してください。

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。
- このパラメータは、Oracle Database 11*g* の Enterprise Edition でのみ有効です。

例

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

ENCRYPTION_MODE

デフォルト : デフォルト・モードは、他に使用される暗号化関連のパラメータによって決まります。ENCRYPTION パラメータのみが指定されている場合、デフォルト・モードは TRANSPARENT になります。ENCRYPTION_PASSWORD パラメータが指定され、Oracle Encryption Wallet 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、Oracle Encryption Wallet 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、Oracle Encryption Wallet、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、Oracle Encryption Wallet 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、Oracle Encryption Wallet 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 パラメータも指定する必要があります。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 Oracle Encryption Wallet 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）の介入なしで作成することができます。したがって、ENCRYPTION_PASSWORD パラメータは必要なく、実際には、TRANSPARENT モードで使用するとエラーの原因となります。この暗号化モードは、ダンプ・ファイル・セットがエクスポート元と同じデータベースにインポートされる場合に最適です。

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があります。それ以外の場合は、エラーが返されます。
- このパラメータは、Oracle Database 11*g* の Enterprise Edition でのみ有効です。

例

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

ENCRYPTION_PASSWORD

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。

注意 : データ・ポンプの暗号化機能は、Oracle Database 11*g* 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。つまり、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場合は、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。エクスポート操作の対象に暗号化された表の列がある場合に暗号化パスワードが指定されていなければ、暗号化された列はクリア・テキストとしてダンプ・ファイル・セットに書き込まれ、警告が発行されます。

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。

注意： 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 11*g* の Enterprise Edition でのみ有効です。
- ENCRYPTION_PASSWORD が指定されているが、ENCRYPTION_MODE は指定されていない場合、ENCRYPTION_MODE がデフォルトで PASSWORD に設定されるため、透過的データ暗号化オプションを設定する必要はありません。
- ENCRYPTION_PASSWORD パラメータは、指定した暗号化モードが TRANSPARENT の場合は有効になりません。
- ENCRYPTION_MODE が DUAL に設定されている場合に ENCRYPTION_PASSWORD パラメータを使用するには、透過的データ暗号化オプションが設定されている必要があります。透過的データ暗号化オプションの詳細は、『Oracle Advanced Security 管理者ガイド』を参照してください。
- ネットワーク・エクスポートについては、ENCRYPTION_PASSWORD パラメータと ENCRYPTED_COLUMNS_ONLY の併用は、暗号化列があるユーザー定義の外部表ではサポートされていません。そのような表はスキップされ、エラー・メッセージが表示されますが、ジョブは続行されます。
- 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 列で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_dir
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。この例で作成された dpcd2be1.dmp ファイルを後でインポートするには、同じ暗号化パスワードを指定する必要があることに注意してください。
（ENCRYPTION_PASSWORD パラメータを使用したインポート操作の例は、3-12 ページの「ENCRYPTION_PASSWORD」を参照してください。）

ESTIMATE

デフォルト: BLOCKS

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。

制限事項

- ESTIMATE=BLOCKS が使用されている場合、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と、その圧縮表に対するデフォルトのサイズ見積りは不正確になります。これは、データが圧縮形式で保存されていたことが、サイズの見積りに反映されないためです。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、SAMPLE または REMAP_DATA 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ことがあります。

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
```

ESTIMATE_ONLY

デフォルト: n

用途

エクスポート・ユーティリティで、ジョブが消費する領域を実際のエクスポート操作を行わずに見積もります。

構文および説明

```
ESTIMATE_ONLY={y | n}
```

ESTIMATE_ONLY=y 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。

例

次に、ESTIMATE_ONLY パラメータを使用して、HR スキーマが使用するエクスポート領域の大きさを判断する場合の例を示します。

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

EXCLUDE

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] [, ...]
```

EXCLUDE 文に指定したオブジェクト型を除き、実行されるエクスポート・モードに対応するすべてのオブジェクト型が、エクスポート操作の対象になります。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。このオプションを使用すると、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。このオプションは、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。この名前句は、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）。名前句は、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の区切りに使用する必要があるため）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 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。

2 つ以上の EXCLUDE 文を指定できます。

エスケープ文字をコマンドラインで使用する必要がないように、EXCLUDE 句は、パラメータ・ファイルで指定することをお薦めします。

参照 :

- パラメータ・ファイルの使用例については、2-22 ページの「INCLUDE」を参照してください。
- 「データ・ポンプ・コマンドラインでの引用符の使用」（2-8 ページ）

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。

- NOT NULL 制約
- 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 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）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、次のフィルタを指定します（hr は除外するユーザーのスキーマ名です）。

```
EXCLUDE=SCHEMA:"='HR'"
```

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、DDL 文 CREATE USER hrDDL 文内で使用される情報のみが除外され、期待した結果が得られない場合があります。

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
```

これによって、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。

参照：

- EXCLUDE パラメータを使用した場合の効果の詳細は、2-6 ページの「エクスポート操作中のフィルタ処理」を参照してください。

FILESIZE

デフォルト：0（無制限）

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 | K | M | G]
```

integer の後に、B、K、M または G（それぞれバイト、キロバイト、メガバイト、ギガバイトを示す）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。

例

次に、サイズが 3MB のダンプ・ファイルを設定する例を示します。

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

3MB でも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、次のエラーが表示され、ジョブは中止されます。

```
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。この値は、ダンプ・ファイル領域がなくなった時点でエクスポート中だったオブジェクトのサイズのみを示しています。

この状況は、中止されたジョブに接続し、ADD_FILE コマンドを使用してファイルを 1 つ以上追加し、操作をやり直すことで解消できます。

FLASHBACK_SCN

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 におけるデータの一貫性を維持したまま実行されます。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。

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。

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
```

> **注意 :** 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。

FLASHBACK_TIME

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。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。

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。コマンドラインの場合は、引用符の前にエスケープ文字を入力する必要があります。詳細は、2-8 ページの「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。

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('25-08-2003 14:35:00', '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。

注意： 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。

参照： フラッシュバック問合せの使用方法の詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。

FULL

デフォルト : n

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={y | n}
```

FULL=y 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。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。詳細は、2-6 ページの「エクスポート操作中のフィルタ処理」を参照してください。

全体エクスポートを実行するには、EXP_FULL_DATABASE ロールが必要です。

注意： 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

- 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・スキーマがエクスポートされません。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。
- 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。

参照： レルムの構成の詳細は、『Oracle Database Vault 管理者ガイド』を参照してください。

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=y NOLOGFILE=y
```

HELP

デフォルト : N

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 = {y | n}
```

HELP=y が指定されている場合は、エクスポート・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

```
> expdp HELP = y
```

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。

INCLUDE

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] [, ...]
```

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、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。

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、INCLUDE パラメータで使用する有効なパスの一覧を表示できます。

name_clause は、オプションです。このオプションを使用すると、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。オプションの名前句は、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。この名前句は、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）。オプションの名前句は、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の区切りに使用する必要があるため）で囲む必要があります。

INCLUDE 文は、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・ファイルで指定しない場合、コマンドラインで引用符の前にオペレーティング・システム固有のエスケープ文字を記述することが必要な場合があります。詳細は、2-8 ページの「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。

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することができます。

```
> expdp hr parfile=hr.par
```

制約の追加

指定した *object_type* が CONSTRAINT である場合は、次に説明する影響があること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、INCLUDE 文の例およびその解釈を示します。

- INCLUDE=CONSTRAINT 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）を含めます。
- INCLUDE=REF_CONSTRAINT は、参照整合性（外部キー）制約を含めます。

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=y
```

JOB_NAME

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。ジョブ名は、現在のユーザーのスキーマでのマスター表の名前となります。マスター表は、エクスポート・ジョブの制御に使用されます。

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、30 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。

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 から始めて増加する 2 桁の整数です。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=y
```

LOGFILE

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）。この指定は、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

- 自動ストレージ管理（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、指定する LOGFILE パラメータには、ASM の + 表記法を含まないディレクトリ・オブジェクトを含める必要があります。つまり、ログ・ファイルはディスク・ファイルに書き込まれ、ASM の記憶域には書き込まれません。かわりに、NOLOGFILE=Y 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
```

注意： データ・ポンプ・エクスポート・ユーティリティは、データベースのキャラクタ・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キャラクタ・セットと異なるキャラクタ・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。

参照：

- 「STATUS」（2-32 ページ）
- 自動ストレージ管理とディレクトリ・オブジェクトの詳細は、1-12 ページの「自動ストレージ管理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」を参照してください。

NETWORK_LINK

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。つまり、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、データベース・リンクを作成する必要があります。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。

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には、デフォルトの一時表領域として、ローカル管理表領域が割り当てられている必要があります。それ以外の場合、ジョブは失敗します。詳細は、『Oracle Database 管理者ガイド』のローカル管理の一時表領域の作成に関する説明を参照してください。

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 Advanced Security 管理者ガイド』を参照してください。

制限事項

- NETWORK_LINK パラメータを TABLES パラメータと組み合せて使用する場合は、表全体のみをエクスポートできます（表のパーティションはエクスポートできません）。
- データ・ポンプ・エクスポートでサポートしているデータベース・リンクのタイプは、パブリック、固定ユーザーおよび接続ユーザーのみです。現在のユーザーのデータベース・リンクは、サポートされていません。

例

次に、NETWORK_LINK パラメータの使用例を示します。*source_database_link* には、すでに存在する有効なデータベース・リンク名を指定します。

```
> expdp hr DIRECTORY=dpump_dir1 NETWORK_LINK=source_database_link
  DUMPFILE=network_export.dmp LOGFILE=network_export.log
```

NOLOGFILE

デフォルト：n

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={y | n}
```

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=y 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場合に、NOLOGFILE=y 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=y
```

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・エクスポートが実行されます。

PARALLEL

デフォルト：1

用途

エクスポート・ジョブにかわり、アクティブな実行スレッドの最大数を指定します。この実行セットはワーカー・プロセスおよびパラレル I/O サーバーの処理の組合せで構成されています。パラレル問合せ操作で問合せコーディネータとして動作するマスター制御プロセスおよびワーカー・プロセスは、この合計数には加算されません。

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・プロセスまたは 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。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するのみです。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。

参照： 「リソース消費の制御」（4-2 ページ）

制限事項

- このパラメータは、Oracle Database 11*g* の Enterprise Edition でのみ有効です。

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 4 つのファイルが作成されます。

参照：

- 「DUMPFILE」（2-12 ページ）
- 「エクスポート・ユーティリティの対話方式コマンド・モードで使用可能なコマンド」（2-40 ページ）
- 「パラレル全データベース・エクスポートの実行」（2-46 ページ）

PARFILE

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*

Oracle Database によって作成され、書き込まれるダンプ・ファイルやログ・ファイルとは異なり、パラメータ・ファイルは、expdp イメージを実行しているクライアントによってオープンされ、読み込まれます。したがって、ディレクトリ・オブジェクトの名前は不要かつ不適切です。ディレクトリ・パスは、オペレーティング・システム固有のディレクトリ指定です。デフォルトは、ユーザーの現行のディレクトリです。

値の指定に引用符が必要なパラメータを使用する場合は、パラメータ・ファイルを使用することをお薦めします。詳細は、2-8 ページの「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。

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
```

QUERY

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。

問合せは一重引用符または二重引用符で囲みます。句内の文字列を一重引用符で囲む必要があるため、二重引用符の使用をお薦めします。オペレーティング・システム固有のエスケープ文字をコマンドラインで使用する必要がないように、QUERY は、パラメータ・ファイルで指定することをお薦めします。詳細は、2-8 ページの「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。

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 の 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)"'
```

  次の問合せのように、表別名として KU$ を使用しないと、すべての行がアンロードされることになります。

```
QUERY='sales:"WHERE EXISTS (SELECT cust_id FROM customers c
   WHERE cust_credit_limit > 10000 AND cust_id = c.cust_id)"'
```

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=y
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。

REMAP_DATA

デフォルト : デフォルト値は設定されていません。

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。これにより、権限のないユーザーに個人データを公開することなく、データで必要な書式と処理特性を保持できます。

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。これは、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。

schema1: 再マップされ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。

schema2: 再マップ・ファンクションを含むユーザー作成の PL/SQL パッケージを格納するスキーマ。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はともに、表内の指定した列のデータ型と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、コミットまたはロールバックを実行できません。

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、これらのファンクションは employees 表内の employee_id および first_name の値を変更すると想定しています。

```
> 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
```

REUSE_DUMPFILES

デフォルト : N

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={Y | N}
```

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、エクスポートを実行して DUMPFILE=hr.dmp および REUSE_DUMPFILES=Y を指定した場合、hr.dmp がすでに存在するときは、このファイルが上書きされます。このファイルの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=y
```

SAMPLE

デフォルト : デフォルト値は設定されていません。

用途

サンプリングしてソース・データベースからアンロードするデータ・ブロックの割合を指定できます。

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、行のブロック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。

sample_percent は特定の表に適用することができます。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することはできます（その場合は、現在のユーザーが使用されます）。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。

なお、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、3-36 ページの「TRANSFORM」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
```

SCHEMAS

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 [, ...]
```

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。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より詳細に制限できます（詳細は、2-6 ページの「エクスポート操作中のフィルタ処理」を参照してください）。

制限事項

- 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 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。

STATUS

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。

TABLES

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（詳細は、2-6 ページの「エクスポート操作中のフィルタ処理」を参照してください）。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。

指定する表名の先頭にスキーマ名を修飾できます。指定するすべての表名が、同じスキーマ内に存在する必要があります。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、EXP_FULL_DATABASE ロールが必要です。

ワイルド・カードは、エクスポート操作ごとに 1 つの表名に対して使用できます。たとえば、TABLES=emp% とした場合、EMP で始まる名前の表がすべてエクスポートされます。

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。

パーティション表がトランスポータブル・メソッドを使用してエクスポートされると、各パーティションおよびサブパーティションは固有の表に昇格されます。以降のインポート操作中は、自動的に表とパーティションの名前が組み合されて新しい表の名前（tablename_partitionname）になります。この自動名前付けオプションを変更するには、インポートの REMAP_TABLE パラメータを使用します。

参照 :

- 「TRANSPORTABLE」（2-36 ページ）
- REMAP_TABLE インポート・コマンド（3-28 ページ）
- 「データ・ファイル・コピーを使用したデータ移動」（1-3 ページ）

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。後でダンプ・ファイルをインポートするときにインポート先のデータベースに型が存在しない場合、表の作成は正常に実行されません。
- 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。
- NETWORK_LINK パラメータを使用した場合、個々の表パーティションのエクスポートはサポートされません。
- 表名にワイルド・カード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 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。
- 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 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_2000,sh.sales:sales_Q2_2000
```

この例では、sh スキーマの sales 表から、パーティション sales_Q1_2000 および sales_Q2_2000 をエクスポートします。

TABLESPACES

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（詳細は、2-6 ページの「エクスポート操作中のフィルタ処理」を参照してください）。

制限事項

- TABLESPACES パラメータに指定する表領域名リストの長さは、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。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。

TRANSPORT_FULL_CHECK

デフォルト: n

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={y | n}
```

TRANSPORT_FULL_CHECK=y 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・セットの内部にあるオブジェクトと外部にある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=n 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・セットの外部にあるオブジェクトの依存オブジェクトが、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。

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=y LOGFILE=tts.log
```

TRANSPORT_TABLESPACES

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・エクス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。

注意： 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・ジョブは、再開できません。
- トランスポータブル・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、EXP_FULL_DATABASE ロールが必要です。
- トランスポータブル・モードは、暗号化された列をサポートしていません。
- 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。
- SYS および SYSAUX 表領域は転送できません。
- 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。

例 1

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=y LOGFILE=tts.log
```

参照：

- 「トランスポータブル表領域モード」（2-5 ページ）
- 「データ・ファイル・コピーを使用したデータ移動」（1-3 ページ）
- データベース間で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。

TRANSPORTABLE

デフォルト：NEVER

用途

特定の表、パーティションおよびサブパーティションのメタデータをエクスポートするために、表モードのエクスポート（TABLES 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。トランスポータブル・オプションを使用すると、TABLES パラメータで指定した表、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ます。実際のデータ・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。詳細は、1-3 ページの「データ・ファイル・コピーを使用したデータ移動」を参照してください。

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。これがデフォルトです。

> **注意：** 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場合は、TRANSPORT_TABLESPACES パラメータを使用します。

制限事項

- TRANSPORTABLE パラメータは、表モードのエクスポートでのみ有効です。
- トランスポータブル・エクスポートを実行するスキーマには、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。
- トランスポータブル・モードではデータはエクスポートされません。データは、表領域データ・ファイルをソース・システムからターゲット・システムにコピーする際にコピーされます。コピーする必要のある表領域は、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。
- 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。
- 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 スキーマが EXP_FULL_DATABASE 権限を持ち、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
```

VERSION

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。このパラメータは、以前のリリースの Oracle Database と互換性のあるダンプ・ファイル・セットの作成に使用できます。なお、これは、10.1 より前のバージョンの Oracle Database でデータ・ポンプ・エクスポートが使用可能ということではありません。データ・ポンプ・エクスポートは、Oracle Database 10*g* 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。

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: 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 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、データベースのバージョンに対応します。
- *version_string*: 特定のデータベース・バージョン（11.1.0 など）。Oracle Database 11*g* の場合、9.2 未満の値は指定できません。

指定したバージョン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、指定したバージョン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。

> **参照：** 「データベース・バージョンが異なる場合のデータ移動」（1-13 ページ）

例

次の例は、メタデータのバージョンがデータベースのバージョン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=y
```

オリジナルのエクスポート・ユーティリティのパラメータへのデータ・ポンプ・エクスポート・パラメータのマップ方法

表 2-1 は、データ・ポンプ・エクスポート・パラメータをオリジナルのエクスポート・パラメータにできるかぎり正確にマップしたものです。機能の設計変更で、オリジナルのエクスポートのパラメータが不要になったため、対応するデータ・ポンプ・パラメータがない場合もあります。また、表に示すとおり、パラメータ名が同じ場合もありますが、機能は多少異なります。

表 2-1 オリジナルのエクスポート・パラメータと、それらに対応するデータ・ポンプ・エクスポートのパラメータ

オリジナルのエクスポート・パラメータ	対応するデータ・ポンプ・エクスポート・パラメータ
BUFFER	BUFFER に相当するパラメータは不要になりました。
COMPRESS	COMPRESS に相当するパラメータは不要になりました。
CONSISTENT	CONSISTENT に相当するパラメータは不要になりました。この機能には、FLASHBACK_SCN および FLASHBACK_TIME を使用します。
CONSTRAINTS	EXCLUDE=CONSTRAINT
DIRECT	DIRECT に相当するパラメータは不要になりました。データ・ポンプ・エクスポート・ユーティリティでは、最適な方法（ダイレクト・パス・モードまたは外部表モード）が自動的に選択されます。
FEEDBACK	STATUS
FILE	DUMPFILE
FILESIZE	FILESIZE

表 2-1 オリジナルのエクスポート・パラメータと、それらに対応するデータ・ポンプ・エクスポートのパラメータ（続き）

オリジナルのエクスポート・パラメータ	対応するデータ・ポンプ・エクスポート・パラメータ
FLASHBACK_SCN	FLASHBACK_SCN
FLASHBACK_TIME	FLASHBACK_TIME
FULL	FULL
GRANTS	EXCLUDE=GRANT
HELP	HELP
INDEXES	EXCLUDE=INDEX
LOG	LOGFILE
OBJECT_CONSISTENT	OBJECT_CONSISTENT に相当するパラメータは、不要になりました。
OWNER	SCHEMAS
PARFILE	PARFILE
QUERY	QUERY
RECORDLENGTH	サイズ指定が自動的に行われるため、RECORDLENGTH に相当するパラメータは不要になりました。
RESUMABLE	RESUMABLE に相当するパラメータは不要になりました。この機能は、EXP_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	RESUMABLE_NAME に相当するパラメータは不要になりました。この機能は、EXP_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	RESUMABLE_TIMEOUT に相当するパラメータは不要になりました。この機能は、EXP_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	CONTENT=METADATA_ONLY
ROWS=Y	CONTENT=ALL
STATISTICS	STATISTICS に相当するパラメータは不要になりました。統計は、常に表に保存されます。
TABLES	TABLES
TABLESPACES	TABLESPACES（同じパラメータですが、動作が多少異なります。）
TRANSPORT_TABLESPACE	TRANSPORT_TABLESPACES（同じパラメータですが、動作が多少異なります。）
TRIGGERS	EXCLUDE=TRIGGER
TTS_FULL_CHECK	TRANSPORT_FULL_CHECK
USERID	USERID に相当するパラメータは不要になりました。この情報は、エクスポート・ユーティリティの起動時に、ユーザー名とパスワードで指定します。
VOLSIZE	VOLSIZE に相当するパラメータは不要になりました。

この表に示されていないデータ・ポンプ・エクスポート・のコマンドライン・パラメータもあります。エクスポートのすべてのコマンドライン・パラメータの一覧は、2-7 ページの「エクスポート・ユーティリティのコマンドライン・モードで使用可能なパラメータ」を参照してください。

> **参照：** オリジナルのエクスポート・ユーティリティの詳細は、第 20 章「オリジナルのエクスポートおよびインポート」を参照してください。

エクスポート・ユーティリティ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、エクスポート・プロンプト（Export>）が表示されます。

> **注意：** データ・ポンプ・エクスポートの対話方式コマンド・モードは、オリジナルのエクスポート・ユーティリティの対話方式モードとは異なります。このモードでは、入力のためのプロンプトが表示されます。オリジナルのエクスポート・ユーティリティの対話方式モードについては、20-6 ページの「対話方式モード」を参照してください。

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。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場合に有効です。

表 2-2 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 2-2 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（2-41 ページ）
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（2-41 ページ）
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（2-41 ページ）
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（2-42 ページ）
使用可能なコマンドの概要を表示する。	HELP（2-42 ページ）
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（2-42 ページ）
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 11*g* の Enterprise Edition でのみ有効。	PARALLEL（2-43 ページ）
接続している停止ジョブを再開する。	START_JOB（2-43 ページ）
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（2-43 ページ）
後で再開するために、現行のジョブを停止する。	STOP_JOB（2-44 ページ）

次の項では、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について説明します。

ADD_FILE

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 [,...]
```

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。別の *directory_object* を指定することもできます。

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。

> **参照：** 置換変数を指定した場合の効果の詳細は、1-10 ページの「ファイルの割当て」を参照してください。

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
```

CONTINUE_CLIENT

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
```

EXIT_CLIENT

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
```

FILESIZE

用途

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・ファイルに適用します。

構文および説明

```
FILESIZE=number
```

ファイル・サイズの後に、B、K、M または G を指定して、それぞれバイト、キロバイト、メガバイト、ギガバイトを示すことができます。デフォルトは、B です。

ファイル・サイズ 0 は、新しいダンプ・ファイルに対してサイズ制限がないことを示します。ファイルは、そのファイルを含むデバイスの限界に到達するまで、必要に応じて大きくなります。

例

```
Export> FILESIZE=100M
```

HELP

用途

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。

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
```

KILL_JOB

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。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し、マスター表およびダンプ・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
```

PARALLEL

用途

現行のジョブに対してアクティブなプロセス（ワーカーおよびパラレル・スレーブ）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。（このパラメータは、Enterprise Edition のみで使用可能です。）必要な数のパラレル・プロセス（ワーカーおよびパラレル・スレーブ）を設定できます。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。値を小さくすると、ワーカー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はされません。

> **参照：** 並列度の詳細は、2-27 ページの「PARALLEL」を参照してください。

例

```
Export> PARALLEL=10
```

START_JOB

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、ジョブが現在実行中の場合を除き、接続している現行のジョブを開始します。ダンプ・ファイル・セットおよびマスター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・エクスポートは再開できません。

例

```
Export> START_JOB
```

STATUS

用途

現行の操作の説明とともにジョブの状態を累積的に表示します。ジョブの推定完了率も返されます。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
```

STOP_JOB

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・ユーティリティを終了します。

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マスター表およびダンプ・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。マスター・プロセスは、ワーカー・プロセスで現行のタスクが終了するまで待機はしません。
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
```

データ・ポンプ・エクスポートの使用例

この項では、データ・ポンプ・エクスポートの使用例を示します。

- 表モード・エクスポートの実行
- 選択した表および行のデータのみのアンロード
- 表モード・エクスポートに必要なディスク領域の見積り
- スキーマ・モード・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
- 対話方式モードを使用したジョブの停止および再接続

これらの例を正しく使用するために役立つ情報については、2-7 ページの「エクスポート・パラメータの使用例」を参照してください。

表モード・エクスポートの実行

例 2-1 に、TABLES パラメータを使用して、表モード・エクスポートを指定する例を示します。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するには、次のデータ・ポンプ・エクスポート・コマンドを発行します。

例 2-1 表モード・エクスポートの実行

```
expdp hr TABLES=employees,jobs DUMPFILE=dpump_dir1:table.dmp NOLOGFILE=y
```

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=y パラメータは、その操作の Export ログ・ファイルが生成されないことを示します。

選択した表および行のデータのみのアンロード

例 2-2 に、人事管理（hr）スキーマから、表 countries および regions を除くすべての表のデータのみをアンロードするために使用するパラメータ・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。

例 2-2 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"
```

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
```

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。

表モード・エクスポートに必要なディスク領域の見積り

例 2-3 に、表モード・エクスポートで使用される領域を、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しないで見積もる場合の例を示します。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。

例 2-3 表モード・エクスポートに必要なディスク領域の見積り

```
> expdp hr DIRECTORY=dpump_dir1 ESTIMATE_ONLY=y TABLES=employees,
departments, locations LOGFILE=estimate.log
```

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。

スキーマ・モード・エクスポートの実行

例 2-4 に、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 2-4 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
```

パラレル全データベース・エクスポートの実行

例 2-5 に、最大 3 つのパラレル・プロセス（ワーカーまたは PQ スレーブ）を持つ全データベース・エクスポートを示します。

例 2-5 パラレル全体エクスポート

```
> expdp hr FULL=y DUMPFILE=dpump_dir1:full1%U.dmp, dpump_dir2:full2%U.dmp
FILESIZE=2G PARALLEL=3 LOGFILE=dpump_dir1:expfull.log JOB_NAME=expfull
```

これは、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。最適なパフォーマンスを得るには、これらのファイルを個別の I/O チャネルに配置する必要があります。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、追加のファイルが作成されます。ジョブおよびマスター表は、expfull という名前になります。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。

対話方式モードを使用したジョブの停止および再接続

この例を試す前に、例 2-5 のパラレル全体エクスポートを再実行します。エクスポートの実行中、[Ctrl] を押しながら [C] を押します。これによって、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。

例 2-6 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
```

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。

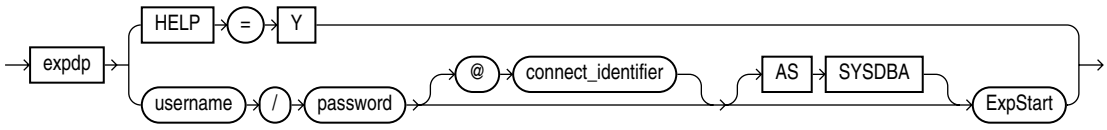
```
Export> CONTINUE_CLIENT
```

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。

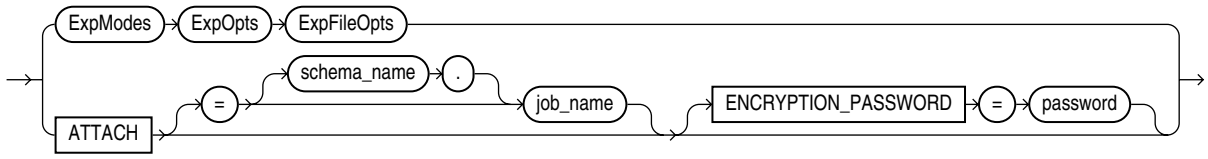
データ・ポンプ・エクスポートの構文図

この項では、データ・ポンプ・エクスポートの構文図を示します。これらの構文図では、標準SQL 構文の表記法を使用します。SQL 構文の表記については、『Oracle Database SQL 言語リファレンス』を参照してください。

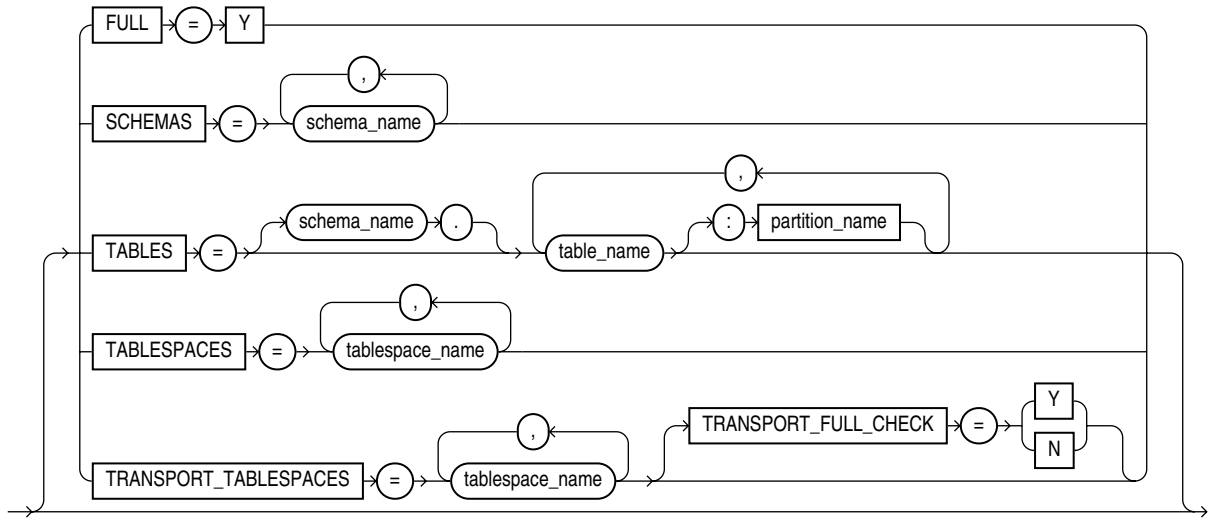
ExpInit



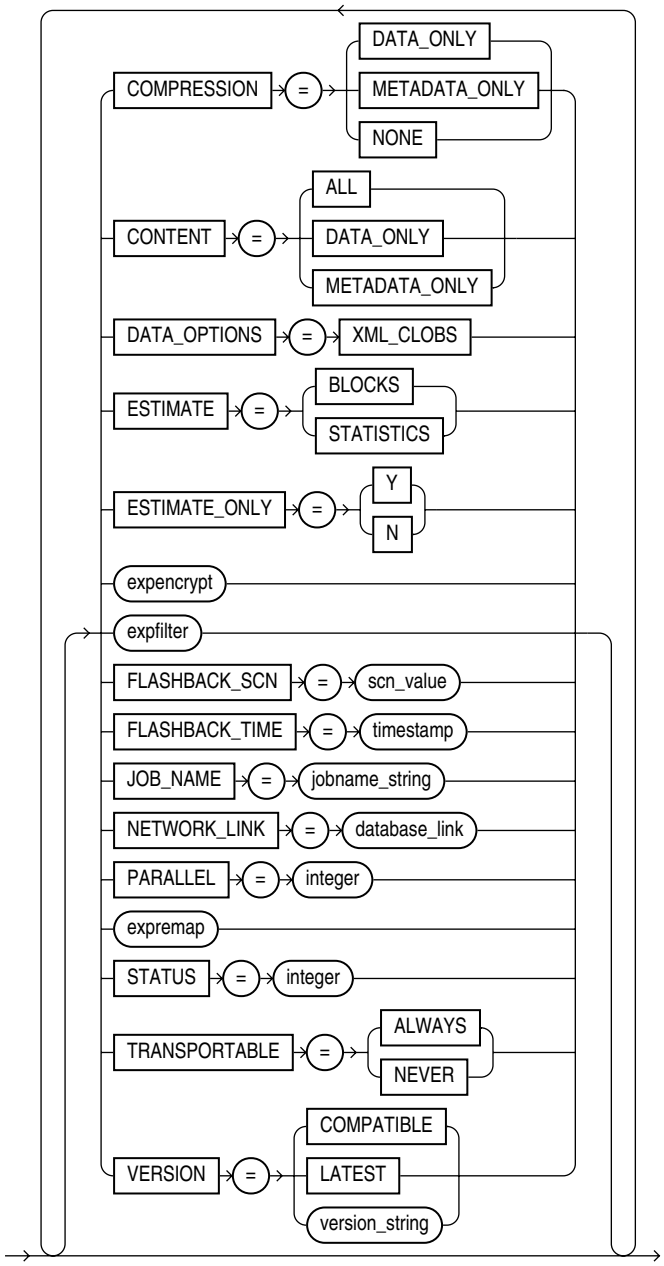
ExpStart



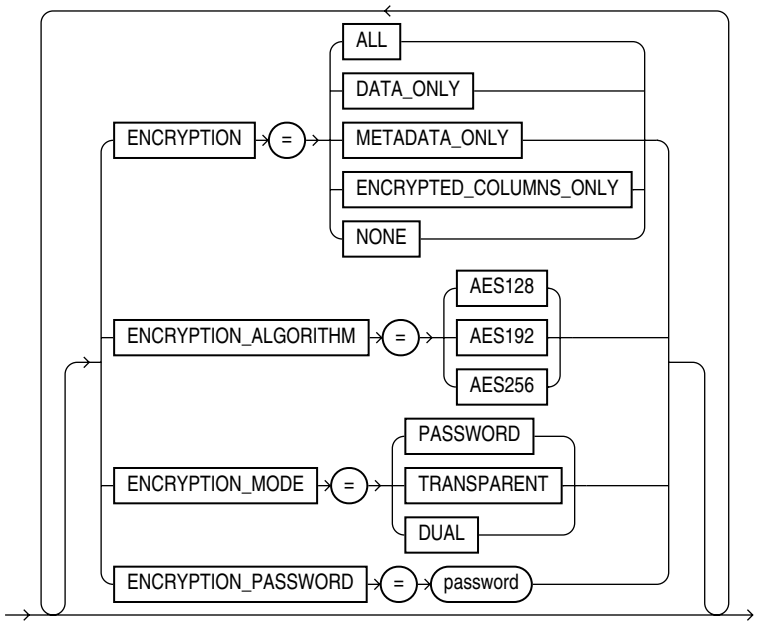
ExpModes



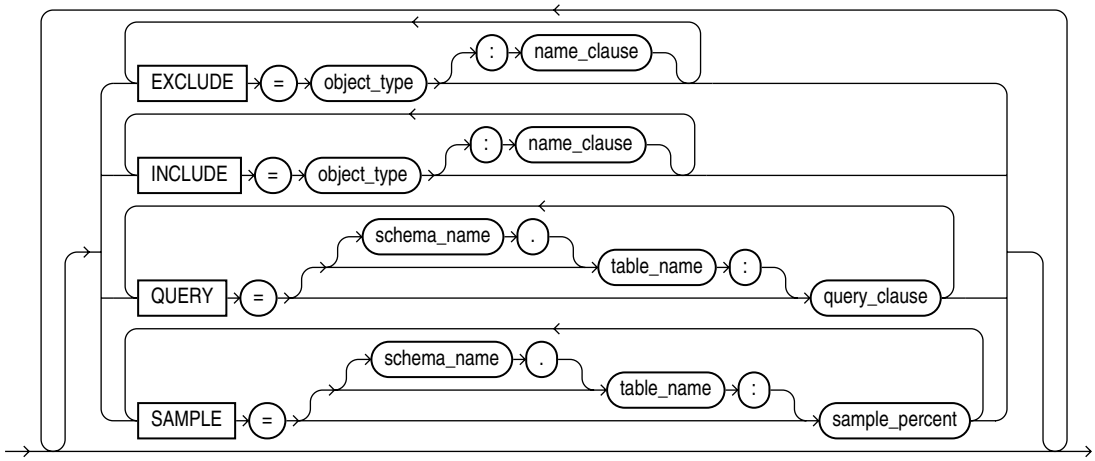
ExpOpts


COMPRESSION
=
DATA_ONLY
METADATA_ONLY
NONE
CONTENT
=
ALL
DATA_ONLY
METADATA_ONLY
DATA_OPTIONS
=
XML_CLOBS
ESTIMATE
=
BLOCKS
STATISTICS
ESTIMATE_ONLY
=
Y
N
expencrypt
expfilter
FLASHBACK_SCN
=
scn_value
FLASHBACK_TIME
=
timestamp
JOB_NAME
=
jobname_string
NETWORK_LINK
=
database_link
PARALLEL
=
integer
expremap
STATUS
=
integer
TRANSPORTABLE
=
ALWAYS
NEVER
VERSION
=
COMPATIBLE
LATEST
version_string

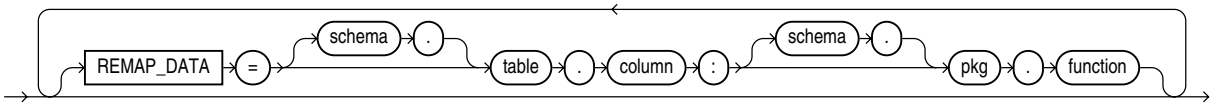
ExpEncrypt



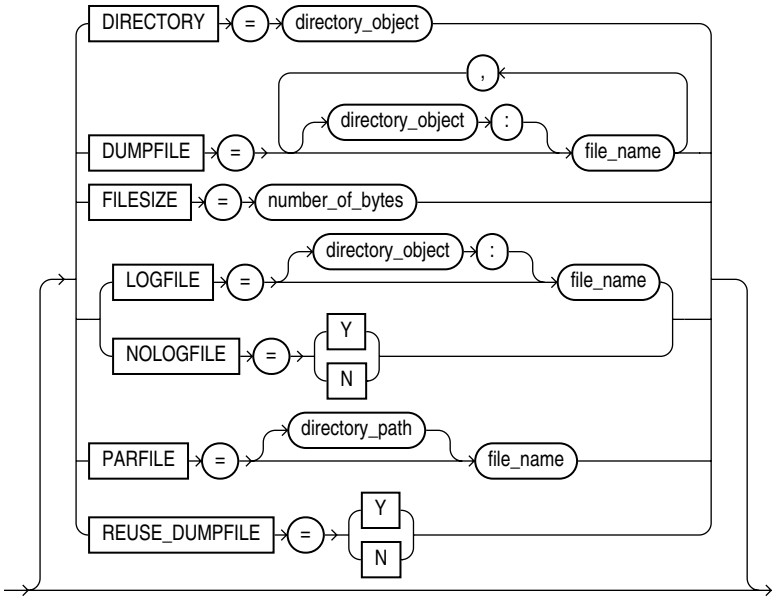
ExpFilter



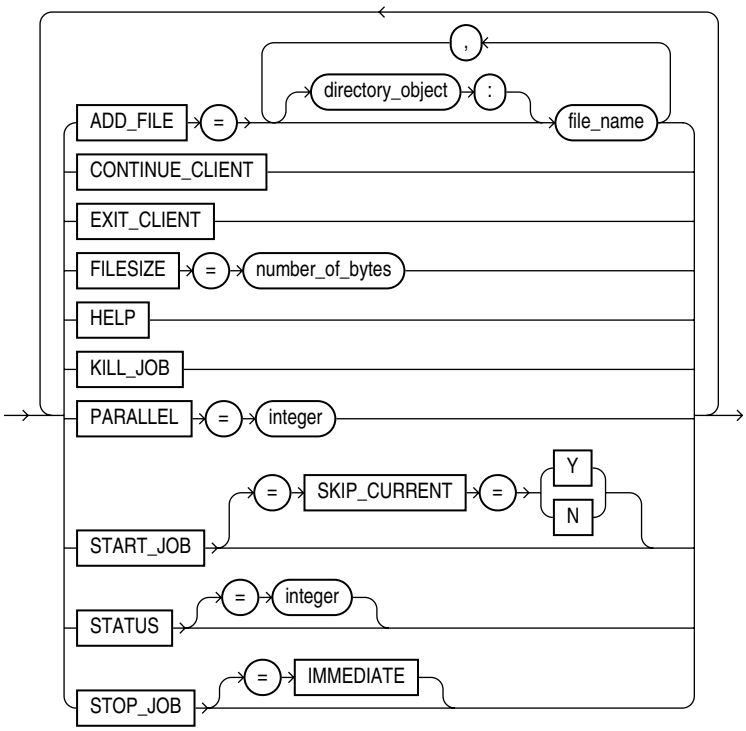
ExpRemap



ExpFileOpts



ExpDynOpts



3

データ・ポンプ・インポート

この章では、Oracle Data Pump Import ユーティリティについて説明します。この章の内容は、次のとおりです。

データ・ポンプ・インポート・ユーティリティとは

> **注意：** データ・ポンプ・インポート（impdp）の機能は、オリジナルのインポート・ユーティリティ（imp）の機能と類似していますが、これらは完全に別のユーティリティであり、それぞれのファイルには互換性がありません。オリジナルのインポート・ユーティリティの詳細は、第 20 章「オリジナルのエクスポートおよびインポート」を参照してください。

データ・ポンプ・インポート（以降、インポート・ユーティリティと呼びます）は、エクスポート・ダンプ・ファイル・セットをターゲット・システムに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、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。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使用することもできます。これはネットワーク・インポートと呼ばれます。

データ・ポンプ・インポート・ユーティリティでは、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・フィルタを使用して行います。詳細は、3-6 ページの「インポート操作中のフィルタ処理」を参照してください。

インポートを使用できる様々な方法の例については、3-47 ページの「データ・ポンプ・インポートの使用例」を参照してください。

データ・ポンプ・インポートの起動

データ・ポンプ・インポート・ユーティリティは、impdp コマンドを使用して起動します。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。

> **注意：** 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。

> **注意：** 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、マスター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。

インポート・ユーティリティの起動の詳細は、次の項を参照してください。

- 「データ・ポンプ・インポートのインタフェース」（3-3 ページ）
- 「データ・ポンプ・インポートのモード」（3-3 ページ）
- 「ネットワークに関する考慮点」（3-5 ページ）

注意： データ・ポンプ・ジョブが Oracle Real Application Clusters（RAC）環境のあるインスタンス上で実行されている場合、その Oracle RAC 環境の他のインスタンス上でデータ・ポンプ・ジョブを起動および再起動することはできません。

データ・ポンプ・インポートのインタフェース

データ・ポンプ・インポートは、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。

- 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・インタフェースで選択可能なパラメータの詳細は、3-7 ページの「インポート・ユーティリティのコマンドライン・モードで使用可能なパラメータ」を参照してください。
- 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。詳細は、3-8 ページの「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。
- 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。

  対話方式コマンド・モードで使用可能なコマンドの詳細は、3-43 ページの「インポート・ユーティリティの対話方式コマンド・モードで使用可能なコマンド」を参照してください。

データ・ポンプ・インポートのモード

インポート操作の重要な特性の 1 つはモードです。これは、インポートされる内容の大部分がモードによって決定されるためです。指定したモードは、操作のソース（ダンプ・ファイル・セットまたは NETWORK_LINK パラメータが指定されている場合は別のデータベース）に適用されます。

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。モードを指定していない場合、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。使用可能なモードは次のとおりです。

- 「全体インポート・モード」（3-4 ページ）
- 「スキーマ・モード」（3-4 ページ）
- 「表モード」（3-4 ページ）
- 「表領域モード」（3-5 ページ）
- 「トランスポータブル表領域モード」（3-5 ページ）

注意： 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。

注意： ジョブ（Oracle Database のジョブ・スケジューラにより作成）は、インポートを実行するユーザーのスキーマに常にインポートされます。インポート後に DBA_JOBS ビューを問い合せると、LOG_USER および PRIV_USER の値が、それらがエクスポート・プラットフォームでどのように設定されているかに関係なく、インポートを実行するユーザーに設定されて表示されます。

この問題に対処するには、エクスポートおよびインポートの両方をジョブの所有者として実行する必要があります。

全体インポート・モード

全体インポートは、FULL パラメータを使用して指定します。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。これは、ファイル・ベース・インポートのデフォルトです。ソースが別のデータベースの場合は、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。

NETWORK_LINK パラメータを完全インポートに使用する場合は、ターゲット・データベースでは IMP_FULL_DATABASE ロールが必要で、ソース・データベースでは EXP_FULL_DATABASE ロールが必要です。

参照： 「FULL」（3-17 ページ）

スキーマ・モード

スキーマ・インポートは、SCHEMAS パラメータを使用して指定します。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。IMP_FULL_DATABASE ロールを所有している場合は、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。

参照： 「SCHEMAS」（3-30 ページ）

表モード

表モードのインポートは、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、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。これには、NETWORK_LINK パラメータも使用する必要があることに注意してください。

参照：

- 「TABLES」（3-34 ページ）
- 「TRANSPORTABLE」（3-40 ページ）
- 「データ・ファイル・コピーを使用したデータ移動」（1-3 ページ）

表領域モード

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。

参照： 「TABLESPACES」（3-35 ページ）

トランスポータブル表領域モード

トランスポータブル表領域インポートは、TRANSPORT_TABLESPACES パラメータを使用して指定します。トランスポータブル表領域モードでは、トランスポータブル表領域エクスポート・ダンプ・ファイル・セットまたは別のデータベースからの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。

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。

このモードには、IMP_FULL_DATABASE ロールが必要です。

注意： 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。

参照：

- 「TRANSPORT_TABLESPACES」（3-39 ページ）
- 「TRANSPORT_FULL_CHECK」（3-38 ページ）
- 「TRANSPORT_DATAFILES」（3-38 ページ）

ネットワークに関する考慮点

データ・ポンプ・インポート・ユーティリティの起動時、接続文字列には接続識別子を指定できます。この識別子では、現行の Oracle システム識別子（SID）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名前を指定できます。これには、接続記述子を使用して検索できるアクティブ・リスナー（起動するには、lsnrctl start と入力）が必要です。

次に、ユーザー hr が inst1 という接続記述子を使用してインポート・ユーティリティを起動する例を示します。

```
> impdp hr DIRECTORY=dpump_dir1 DUMPFILE=hr.dmp TABLES=employees

Import: Release 11.1.0.6.0 - Production on Monday, 27 August, 2007 12:25:57

Copyright (c) 2003, 2007, Oracle.  All rights reserved.

Password: password@inst1

Connected to: Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, Data Mining and Real Application Testing options
```

ローカルのインポート・クライアントは、接続記述子 inst1（通常は tnsnames.ora ファイルで定義される単純なネット・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・セットのデータをそのデータベースにインポートします。

接続識別子を使用したインポート・ユーティリティの起動と、インポート・コマンドライン・パラメータ NETWORK_LINK を指定したインポート操作を混同しないでください。コマンドライン・パラメータを使用した方法では、データベース・リンクを使用してインポートが起動されます。この場合、ローカルのインポート・クライアントは、コマンドライン接続文字列によって指定されるデータベース・インスタンスに接続し、データベース・リンクによって指定されるデータベース・インスタンスからインポートするデータを取得して、接続したデータベース・インスタンスにそのデータを書き込みます。ダンプ・ファイル・セットは含まれません。

参照：

- 「NETWORK_LINK」（3-20 ページ）
- 『Oracle Database Net Services 管理者ガイド』
- 『Oracle Database Heterogeneous Connectivity 管理者ガイド』

インポート操作中のフィルタ処理

データ・ポンプ・インポート・ユーティリティで提供されるデータおよびメタデータのフィルタ機能は、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。

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結果として、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。

メタデータ・フィルタ

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。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、データ・ポンプ操作に含めるか、またはその操作から除外するオブジェクトを識別します。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 つのジョブ内で同一のフィルタ名を複数回指定できます。

フィルタ処理できるオブジェクトを確認するには、DATABASE_EXPORT_OBJECTS（全体モード・インポートの場合）、SCHEMA_EXPORT_OBJECTS（スキーマ・モード・インポートの場合）および TABLE_EXPORT_OBJECTS（表モードおよび表領域モード・インポートの場合）の各ビューに対して問合せを実行します。オブジェクトのフルパス名は、インポート・モードでなくエクスポート・モードで決定されることに注意してください。

例については、2-6 ページの「メタデータ・フィルタ」を参照してください。

参照：

- フィルタ処理の使用例は、2-6 ページの「メタデータ・フィルタ」を参照してください。
- EXCLUDE インポート・パラメータの詳細は、3-14 ページを参照してください。
- INCLUDE インポート・パラメータの詳細は、3-18 ページを参照してください。

インポート・ユーティリティのコマンドライン・モードで使用可能なパラメータ

この項では、データ・ポンプ・インポート・ユーティリティのコマンドライン・モードで使用可能なパラメータについて説明します。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。

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、データベースとの接続が行われる前にパスワードの入力が要求されます。

```
Import: Release 11.1.0.6.0 - Production on Monday, 27 August, 2007 12:15:55

Copyright (c) 2003, 2007, Oracle.  All rights reserved.

Password: password

Connected to: Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 -
Production
With the Partitioning, Data Mining and Real Application Testing options
```

- 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、第 2 章で Export の例を実行すると生成されるダンプ・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 スキーマに付与されているものとします。ディレクトリ・オブジェクトの作成およびこれらへの権限の割当てについては、1-10 ページの「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。
- 一部の例で、EXP_FULL_DATABASE ロールおよび IMP_FULL_DATABASE ロールが必要です。このような例では、hr スキーマ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。

これらのパラメータの構文図は、3-49 ページの「データ・ポンプ・インポートの構文図」を参照してください。

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。

データ・ポンプ・コマンドラインでの引用符の使用

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必要がある場合があります。バックスラッシュがない場合、インポートで使用するコマンドライン解析機能で引用符として認識されないため、引用符が削除されエラーが発生します。通常、そのような文は、パラメータ・ファイルに記述することをお薦めします。パラメータ・ファイルでは、エスケープ文字は不要なためです。

参照：

- デフォルトのディレクトリ・オブジェクトの作成については、1-10 ページの「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。
- 「データ・ポンプ・インポートの使用例」（3-47 ページ）
- 『Oracle Database サンプル・スキーマ』

注意： オリジナルのインポート・ユーティリティを使い慣れている場合、オリジナルのインポート・ユーティリティと同様の操作の実行に使用するデータ・ポンプ・パラメータを特定できないことがあります。両者の対応関係は、3-41 ページの「オリジナルのインポート・ユーティリティのパラメータへのデータ・ポンプ・インポート・パラメータのマップ方法」を参照してください。

ATTACH

デフォルト：ユーザーのスキーマで現在実行されているジョブ（実行中のジョブが 1 つのみの場合）

用途

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。このパラメータを指定するには、IMP_FULL_DATABASE ロールが必要です。

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、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、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。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。

> **参照：**「インポート・ユーティリティの対話方式コマンド・モードで使用可能なコマンド」(3-43 ページ)

CONTENT

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。これがデフォルトです。
- DATA_ONLY を指定すると、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 を指定すると、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。

制限事項

- CONTENT=METADATA_ONLY パラメータおよび値は、パラメータ TRANSPORT_TABLESPACES（トランスポータブル表領域モード）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。詳細は、2-21 ページの「FULL」を参照してください。

```
> impdp hr DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp CONTENT=METADATA_ONLY
```

このコマンドは、expfull.dmp ダンプ・ファイルのメタデータのみをロードする全体インポートを実行します。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。

DATA_OPTIONS

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 パラメータを使用すると、エクスポートおよびインポート中に特定のタイプのデータを処理するためのオプションが提供されます。インポート操作の場合、DATA_OPTIONS パラメータに対して唯一有効なオプションは SKIP_CONSTRAINT_ERRORS です。

構文および説明

```
DATA_OPTIONS=SKIP_CONSTRAINT_ERRORS
```

SKIP_CONSTRAINT_ERRORS オプションは、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）のロード中の非遅延の制約違反における処理方法に適用されます。遅延制約違反が発生しても、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 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。

制限事項

- SKIP_CONSTRAINT_ERRORS が使用され、データ・オブジェクトがロード時にそのデータ・オブジェクトに対して定義された一意の索引または制約を持つ場合、そのデータ・オブジェクトのロードに 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より時間がかかります。
- 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、それはログに記録されますが、インポートは完了するまで続行されます。

DIRECTORY

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、データベースのディレクトリ・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）。インストール時に、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 へのアクセス権を持つユーザーが 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・セット用に使用するディレクトリに対する読取り権限と、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に対する書込み権限が必要です。

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。詳細は、2-21 ページの「FULL」を参照してください。

```
> 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。

参照：

- デフォルトのディレクトリ・オブジェクトの詳細は、1-10 ページの「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。
- 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。

DUMPFILE

デフォルト：expdat.dmp

用途

Export 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。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。

構文および説明

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

DIRECTORY パラメータで指定されている場合、*directory_object* はオプションです。ここで値を指定する場合は、すでに存在しアクセス権があるディレクトリ・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。%U を使用した場合、インポート・ユーティリティは、テンプレートと一致する各ファイルを一致するファイルが検出されなくなるまで調べ、ダンプ・ファイル・セットの一部となるすべてのファイルの位置を特定します。%U は、01 から始まる 2 桁の整数に変換されます。

DUMPFILE パラメータでのファイル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の位置を特定するための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、エクスポート時と同じである必要はありません。

例

次に、インポートの DUMPFILE パラメータの使用例を示します。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。詳細は、2-12 ページの「DUMPFILE」を参照してください。

```
> 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、exp2<nn>.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。

参照：

- 「ファイルの割当て」（1-10 ページ）
- 「データのみ表モード・インポートの実行」（3-47 ページ）

ENCRYPTION_PASSWORD

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。

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

- このパラメータは、Oracle Database 11*g* の Enterprise Edition でのみ有効です。
- 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、ネットワーク・インポート・ジョブには無効です。
- 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 列で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（2-15 ページの「ENCRYPTION_PASSWORD」を参照）。

```
> impdp hr TABLES=employee_s_encrypt DIRECTORY=dpump_dir
  DUMPFILE=dpcd2be1.dmp ENCRYPTION_PASSWORD=123456
```

インポート操作時、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、複合化されてからインポートされます。

ESTIMATE

デフォルト : BLOCKS

用途

このパラメータによって、ネットワーク・インポート操作のソース・システムで、データの生成量が見積もられます。

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。

生成される見積りは、インポート・ジョブの完了率の確認に使用されます。

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 パラメータ、SAMPLE パラメータまたは REMAP_DATA パラメータを使用する場合、見積りが不正確になることがあります。

例

次の例では、*source_database_link* にソース・データベースに対する有効なリンク名を指定します。

```
> 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。

EXCLUDE

デフォルト : デフォルト値は設定されていません。

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*] [, ...]

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。このオプションを使用すると、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。オプションの名前句は、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。この名前句は、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）。オプションの名前句は、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の区切りに使用する必要があるため）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。

2 つ以上の EXCLUDE 文を指定できます。オペレーティング・システム固有のエスケープ文字をコマンドラインで使用する必要がないように、EXCLUDE 文は、パラメータ・ファイルで指定することをお薦めします。

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。

- NOT NULL 制約。
- 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 は、NOT NULL 制約および表の正常な作成とロードに必要な制約を除き、参照制約以外の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、次のフィルタを指定します（hr は除外するユーザーのスキーマ名です）。

```
EXCLUDE=SCHEMA: "= 'HR' "
```

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、DDL 文 CREATE USER hr のみが除外され、期待した結果が得られない場合があります。

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

DBA または 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。（例を試す場合は、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%' "
```

次のコマンドを発行します。このコマンドでは、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、2-21 ページの「FULL」を参照してください。

```
> impdp system DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp PARFILE=exclude.par
```

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。

> **参照：** EXCLUDE パラメータを使用した場合の効果の詳細は、3-6 ページの「インポート操作中のフィルタ処理」を参照してください。

FLASHBACK_SCN

デフォルト : デフォルト値は設定されていません。

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* におけるデータの一貫性を維持したまま実行されます。

注意： ロジカル・スタンバイ・システムを使用している場合は、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。

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。

例

次に、FLASHBACK_SCN パラメータの使用例を示します。

```
> impdp hr DIRECTORY=dpump_dir1 FLASHBACK_SCN=123456
NETWORK_LINK=source_database_link
```

この例の *source_database_link* には、データのインポート元であるソース・データベース名を指定します。

FLASHBACK_TIME

デフォルト : デフォルト値は設定されていません。

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 におけるデータの一貫性を維持したまま実行されます。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。コマンドラインの場合は、引用符の前にエスケープ文字を入力する必要があります。詳細は、3-8 ページの「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。

> **注意 :** ロジカル・スタンバイ・システムを使用している場合は、ロジカル・スタンバイによって SCN が選択されるため、FLASHBACK_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('25-08-2003 14:35:00', '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。

> **参照 :** フラッシュバックの使用方法の詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。

FULL

デフォルト：Y

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=y
```

FULL=y の値は、ソース（ダンプ・ファイル・セットまたは他のデータベース）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。

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（詳細は、3-6 ページの「インポート操作中のフィルタ処理」を参照してください）。

NETWORK_LINK パラメータが使用されている場合、インポート・ジョブを実行する USERID はターゲット・データベースの IMP_FULL_DATABASE ロールを持ち、そのユーザーは、ソース・データベースの EXP_FULL_DATABASE ロールも持っている必要があります。

ファイルのインポート権限が付与されていないユーザーの場合は、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。

FULL は、ファイル・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、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。詳細は、2-21 ページの「FULL」を参照してください。

```
> impdp hr DUMPFILE=dpump_dir1:expfull.dmp FULL=y
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。

HELP

デフォルト：n

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=y
```

HELP=y 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

```
> impdp HELP = Y
```

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。

INCLUDE

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] [, ...]
```

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。

name_clause は、オプションです。このオプションを使用すると、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。オプションの名前句は、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。この名前句は、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）。オプションの名前句は、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の区切りに使用する必要があるため）で囲む必要があります。

2 つ以上の INCLUDE 文を指定できます。オペレーティング・システム固有のエスケープ文字をコマンドラインで使用する必要がないように、INCLUDE 文は、パラメータ・ファイルで指定することをお薦めします。

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、INCLUDE パラメータで使用する有効なパスの一覧を表示できます。

制限事項

- INCLUDE および EXCLUDE は、相互に排他的なパラメータです。

例

DBA または IMP_FULL_DATABASE ロールを持つ他のユーザーに使用されているパラメータ・ファイル imp_include.par が、次のように指定されているとします。

```
INCLUDE=FUNCTION
INCLUDE=PROCEDURE
INCLUDE=PACKAGE
INCLUDE=INDEX:"LIKE 'EMP%' "
```

次のコマンドを発行します。

```
> 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。詳細は、2-21 ページの「FULL」を参照してください。

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 で始まる索引のみがロードされます。これは特権モードのインポート（ユーザーに 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。

JOB_NAME

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

ジョブ名は、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、後続処理でインポート・ジョブを指定するために使用されます。ジョブ名は、現在のユーザーのスキーマでのマスター表の名前となります。インポート・ジョブは、マスター表によって制御されます。

構文および説明

JOB_NAME=*jobname_string*

jobname_string には、このインポート・ジョブの名前を、30 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 Import' とします)。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。

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 から始めて増加する 2 桁の整数です。デフォルト名は、'SYS_IMPORT_TABLESPACE_02' などです。

例

次に、JOB_NAME パラメータの使用例を示します。この例では、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、2-21 ページの「FULL」を参照してください。

```
> impdp hr DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp JOB_NAME=impjob01
```

LOGFILE

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 には、DBA によって作成済で、自分にアクセス権があるディレクトリ・オブジェクトを指定する必要があります。この指定は、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。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。

file_name に指定したファイルがすでに存在する場合、そのファイルは上書きされます。

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 パラメータが指定されていないかぎり、常に、ログ・ファイルは作成されます。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。

注意： データ・ポンプ・インポート・ユーティリティは、データベースのキャラクタ・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 環境にデータベースのキャラクタ・セットと異なるキャラクタ・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ことがあります。

制限事項

- 自動ストレージ管理（ASM）を使用してデータ・ポンプ・インポートを実行する場合、指定する LOGFILE パラメータには、ASM の + 表記法を含まないディレクトリ・オブジェクトを含める必要があります。つまり、ログ・ファイルはディスク・ファイルに書き込まれ、ASM の記憶域には書き込まれません。かわりに、NOLOGFILE=Y 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。詳細は、2-21 ページの「FULL」を参照してください。

```
> 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。

参照：

- 「STATUS」（3-46 ページ）
- 自動ストレージ管理とディレクトリ・オブジェクトの詳細は、1-12 ページの「自動ストレージ管理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」を参照してください。

NETWORK_LINK

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イン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、データベース・リンクを作成する必要があります。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。

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、接続ユーザーには、ソース・データベース上のデフォルトの一時表領域として、ローカル管理表領域が割り当てられている必要があります。それ以外の場合、ジョブは失敗します。詳細は、『Oracle Database 管理者ガイド』のローカル管理の一時表領域の作成に関する説明を参照してください。

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。

> **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 Advanced Security 管理者ガイド』を参照してください。

制限事項

- 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。
- NETWORK_LINK パラメータを TABLES パラメータと組み合せて使用する場合は、表全体のみをインポートできます（表のパーティションはインポートできません）。唯一の例外は、TRANSPORTABLE=ALWAYS も指定されている場合で、この場合は、指定した表の単一または複数のパーティションをインポートできます。
- 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 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 EXP_FULL_DATABASE ロールも持っている必要があります。
- データ・ポンプ・インポートでサポートしているデータベース・リンクのタイプは、パブリック、固定ユーザーおよび接続ユーザーのみです。現在のユーザーのデータベース・リンクは、サポートされていません。

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。

```
> 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。

NOLOGFILE

デフォルト：n

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={y | n}
```

NOLOGFILE=Y 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場合に NOLOGFILE=Y 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=Y
```

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 y 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。

PARALLEL

デフォルト：1

用途

インポート・ジョブにかわり、アクティブな実行スレッドの最大数を指定します。

構文および説明

PARALLEL=*integer*

integer に指定する値は、インポート・ジョブの動作でアクティブな実行操作の最大スレッド数です。この実行セットはワーカー・プロセスおよびパラレル I/O サーバーの処理の組合せで構成されています。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動作するマスター制御プロセス、アイドル状態のワーカーおよびワーカー・プロセスは、この合計数には加算されません。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。

並列度は、ユーザー・データおよびパッケージ本体のロード、索引の作成に使用します。

参照：「リソース消費の制御」（4-2 ページ）

制限事項

- このパラメータは、Oracle Database 11*g* の Enterprise Edition でのみ有効です。

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
```

このコマンドは、Export 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・セットをインポートします。（詳細は、2-27 ページの「PARALLEL」を参照してください。）ダンプ・ファイル名は、par_exp01.dmp、par_exp02.dmp および par_exp03.dmp です。

PARFILE

デフォルト：デフォルト値は設定されていません。

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*

サーバーによって作成され、書き込まれるダンプ・ファイル、ログ・ファイル、SQL ファイルとは異なり、パラメータ・ファイルは、impdp イメージを実行しているクライアントによってオープンされ、読み込まれます。したがって、ディレクトリ・オブジェクトの名前は不要かつ不適切です。デフォルトは、ユーザーの現行のディレクトリです。値の指定に引用符が必要なパラメータを使用する場合は、パラメータ・ファイルを使用することをお薦めします。（詳細は、3-8 ページの「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。）

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_imp.par の内容は、次のとおりです。

```
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、次のコマンドを実行します。

```
> impdp hr PARFILE=hr_imp.par
```

表 countries、locations および regions は、Export 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。（詳細は、2-12 ページの「DUMPFILE」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 exp2<nn>.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。

PARTITION_OPTIONS

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どのように作成するかを指定します。

構文および説明

PARTITION_OPTIONS={none | departition | merge}

none の値を指定した場合、エクスポート操作が実行されたシステム上に存在していたのと同様に表が作成されます。エクスポートがパーティションまたはサブパーティション・フィルタとともにトランスポータブル・メソッドを使用して実行されている場合、none オプションまたは 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 の値を指定した場合、各パーティションまたはサブパーティションは、新しい個々の表に昇格します。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 の値を設定した場合、すべてのパーティションおよびサブパーティションは 1 つの表に統合されます。

制限事項

- 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おり、パーティションまたはサブパーティションが指定されている場合、インポート操作では 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ているオブジェクトに対する権限が存在する場合、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 スキーマ内のパーティション化されていない表にマージします。

```
> impdp system TABLES=sh.sales PARTITION_OPTIONS=merge
DIRECTORY=dpump_dir1 DUMPFILE=sales.dmp REMAP_SCHEMA=sh:scott
```

> **参照：** PARTITION_OPTIONS=DEPARTITION を使用したインポート操作の実行例は、2-36 ページの「TRANSPORTABLE」を参照してください。

QUERY

デフォルト：デフォルト値は設定されていません。

用途

インポート対象となる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い場合は、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。

問合せは一重引用符または二重引用符で囲みます。句内の文字列を一重引用符で囲む必要があるため、二重引用符の使用をお薦めします。オペレーティング・システム固有のエスケープ文字をコマンドラインで使用する必要がないように、QUERY は、パラメータ・ファイルで指定することをお薦めします。詳細は、3-8 ページの「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。

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 パラメータは次のパラメータとは併用できません。
  - 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。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 表のサブセットをインポートするとします。次の例では、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ることになります。

  ```
  QUERY='sales:"WHERE EXISTS (SELECT cust_id FROM customers c WHERE cust_credit_limit
  > 10000 AND cust_id = c.cust_id)"'
  ```

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、2-21 ページの「FULL」を参照してください。QUERY の値には引用符が使用されるため、コマンドラインでエスケープ文字を使用する必要がないように、パラメータ・ファイルを使用することをお薦めします。（詳細は、3-8 ページの「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。）

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=Y
```

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。

REMAP_DATA

デフォルト：デフォルト値は設定されていません。

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。これは、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: 再マップされる表を含むスキーマ。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。

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はともに、表内の指定した列のデータ型と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。

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
```

REMAP_DATAFILE

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・ファイルが指定されるすべての SQL 文（`CREATE TABLESPACE`、`CREATE LIBRARY`、`CREATE DIRECTORY` など）のソース・データ・ファイルの名前をターゲット・データ・ファイルの名前に変更します。

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。

このパラメータを指定するには、`IMP_FULL_DATABASE` ロールが必要です。

例

REMAP_DATAFILE の値には引用符が使用されるため、コマンドラインでエスケープ文字を使用する必要がないように、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。（詳細は、3-8 ページの「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。）たとえば、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=Y
DUMPFILE=db_full.dmp
REMAP_DATAFILE="'DB1$:[HRDATA.PAYROLL]tbs6.f':'/db1/hrdata/payroll/tbs6.f'"
```

次のコマンドを発行します。

```
> impdp hr PARFILE=payroll.par
```

この例では、インポート時に、すべての SQL DDL 文に対する VMS ファイル指定（DR1$:[HRDATA.PAYROLL]tbs6.f）を UNIX ファイル指定（/db1/hrdata/payroll/tbs6.f）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。

REMAP_SCHEMA

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・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。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、マッピングは完全ではない場合があります。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。

再マッピング先のスキーマが存在しない場合は、インポート操作によってそのスキーマが作成されます。ただし、ソース・スキーマに必要な CREATE USER メタデータがダンプ・ファイル・セットに含まれており、ユーザーが必要な権限を所有してインポートを実行していることが条件となります。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=y
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。これは、権限が付与されていないダンプ・ファイルには、インポート操作でスキーマを自動作成するための情報が含まれないためです。

インポート操作によってスキーマが作成された場合は、インポートの完了後、そのスキーマに有効なパスワードを割り当てて、接続できるようにする必要があります。パスワードを割り当てる SQL 文（権限が必要）は、次のとおりです。

```
SQL> ALTER USER schema_name IDENTIFIED BY new_password
```

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 のオブジェクトを BLAKE に再マップすることはできません。

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。これは、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。

REMAP_TABLE

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・メソッドを使用して実行された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

REMAP_TABLE=[*schema.*]*old_tablename*[*.partition*]*:new_tablename*

REMAP_TABLE パラメータを使用すると、表全体の名前を変更できます。

また、トランスポータブル・メソッドによりエクスポートされた表パーティションの自動名前付けオプションを変更する場合も、このパラメータを使用します。パーティション表がトランスポータブル・メソッドを使用してエクスポートされると、各パーティションおよびサブパーティションは固有の表に昇格され、その表は、デフォルトで、表とパーティションの名前を組み合せた名前（tablename_partitionname）になります。このデフォルト以外の名前を指定するには、REMAP_TABLE を使用します。

制限事項

- 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、既存の表で、TABLE_EXISTS_ACTION が TRUNCATE または APPEND に設定されている場合は、再マップされません。

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
```

REMAP_TABLESPACE

デフォルト : デフォルト値は設定されていません。

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方法は、REMAP_TABLESPACE パラメータを使用する方法のみです。これは、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 以降の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、既存の表の表領域で、TABLE_EXISTS_ACTION が SKIP、TRUNCATE または APPEND に設定されている場合は、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
```

REUSE_DATAFILES

デフォルト : n

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={y | n}
```

デフォルト（n）が使用され、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。この場合、データが失われる可能性があるため注意してください。

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。詳細は、2-21 ページの「FULL」を参照してください。

```
> impdp hr DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp LOGFILE=reuse.log
REUSE_DATAFILES=Y
```

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。

SCHEMAS

デフォルト : デフォルト値は設定されていません。

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* [,...]

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てい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。詳細は、3-6 ページの「インポート操作中のフィルタ処理」を参照してください。

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。詳細は、2-32 ページの「SCHEMAS」を参照してください。

```
> 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。

SKIP_UNUSABLE_INDEXES

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={y | n}
```

SKIP_UNUSABLE_INDEXES が y に設定されているときに、索引が使用禁止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 n 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（デフォルト値は y）が参照され、使用禁止の索引の処理が決定されます。

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。

> **注意：** このパラメータは、データを既存の表にインポートする場合にのみ有効です。インポート処理の一環として表が作成される場合は、表と索引が新規に作成され、使用禁止のマークは付けられないため、このパラメータによる実質的な効果はありません。

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。詳細は、2-21 ページの「FULL」を参照してください。

```
> impdp hr DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp LOGFILE=skip.log
SKIP_UNUSABLE_INDEXES=y
```

SQLFILE

デフォルト : デフォルト値は設定されていません。

用途

インポートが他のパラメータに基づいて実行するすべての SQL DDL の書込み先の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、インポート・ジョブが、ジョブで実行する DDL を書き込むファイル名を指定します。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、他の *directory_object* が明示的に指定されないかぎり、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、その文はコメントで置き換えられ、スキーマ名のみが表示されます。次の例では、ダッシュの後に続くのがコメントです。また、hr というスキーマ名は表示されていますが、パスワードは表示されていません。

```
-- 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して編集する必要があります。

Streams などの Oracle Database オプションでは、無名 PL/SQL ブロックが SQLFILE 出力に出現することがあります。これらは、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。
- 自動ストレージ管理（ASM）を使用して SQL ファイルへのデータ・ポンプ・インポートを実行する場合、指定する SQLFILE パラメータには、ASM の + 表記法を使用しないディレクトリ・オブジェクトを含める必要があります。つまり、SQL ファイルはディスク・ファイルに書き込まれ、ASM の記憶域には書き込まれません。

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。詳細は、2-21 ページの「FULL」を参照してください。

```
> 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。

STATUS

デフォルト：0

用途

ジョブ状態が表示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。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、2-21 ページの「FULL」を参照してください。

```
> impdp hr NOLOGFILE=y 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。

STREAMS_CONFIGURATION

デフォルト：y

用途

エクスポート・ダンプ・ファイル内に存在する Streams 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={y | n}
```

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。詳細は、2-21 ページの「FULL」を参照してください。

```
> impdp hr DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp STREAMS_CONFIGURATION=n
```

参照：『Oracle Streams レプリケーション管理者ガイド』

TABLE_EXISTS_ACTION

デフォルト：SKIP（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・ユーティリティに対して、作成しようとしている表がすでに存在する場合に行う操作を指定します。

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に考慮する事項は次のとおりです。

- 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 を使用する場合は、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無視されます。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除外（EXCLUDE を使用して）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、ソースにある行が既存の表に適合するかどうかがチェックされます。

  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。

制限事項

- TRUNCATE は、クラスタ化された表またはネットワーク・リンクを介しては使用できません。

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。詳細は、2-21 ページの「FULL」を参照してください。

```
> impdp hr TABLES=employees DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp
TABLE_EXISTS_ACTION=REPLACE
```

TABLES

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・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、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。

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。詳細は、3-6 ページの「インポート操作中のフィルタ処理」を参照してください。

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。

表名を指定する場合のワイルドカードの使用もサポートされていますが、指定できる表の式は 1 つのみです。たとえば、TABLES=emp% と指定すると、名前が EMP で始まるすべての表がインポートされます。

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。
- 複数の *table_name* を指定する場合は、それらのすべてが同じスキーマに存在する必要があります。
- 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。
- NETWORK_LINK パラメータを TABLES パラメータと組み合せて使用する場合は、表全体のみをインポートできます（表のパーティションはインポートできません）。唯一の例外は、TRANSPORTABLE=ALWAYS も指定されている場合で、この場合は、指定した表の単一または複数のパーティションをインポートできます。
- 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、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。詳細は、2-21 ページの「FULL」を参照してください。

```
> impdp hr DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp TABLES=employees,jobs
```

次に、TABLES パラメータを使用したパーティションのインポート例を示します。

```
> impdp hr DIRECTORY=dpump_dir1 DUMPFILE=expdat.dmp
TABLES=sh.sales:sales_Q1_2000,sh.sales:sales_Q2_2000
```

この例では、sh スキーマの sales 表のパーティション sales_Q1_2000 および sales_Q2_2000 をインポートします。

TABLESPACES

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・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。

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。詳細は、3-6 ページの「インポート操作中のフィルタ処理」を参照してください。

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。詳細は、2-21 ページの「FULL」を参照してください。

```
> 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。

TRANSFORM

デフォルト : デフォルト値は設定されていません。

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。使用可能なオプションは、次のとおりです。

- SEGMENT_ATTRIBUTES: 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。
- STORAGE: 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。
- OID: 値を n に指定すると、オブジェクトの表と型の作成時に、エクスポートされた OID の割当てが禁止されます。かわりに、新しい OID が割り当てられます。これは、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。デフォルト値は y です。
- PCTSPACE: 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。（詳細は、2-31 ページの「SAMPLE」を参照してください。）

指定する *value* の型は、使用する変換によって異なります。SEGMENT_ATTRIBUTES、STORAGE、OID の各変換では、ブール値（y/n）が必要です。PCTSPACE 変換では、整数値が必要です。

object_type はオプションです。このオプションで、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。表 3-1 に、変換ごとの有効なオブジェクト型を示します。

表 3-1 データ・ポンプ・エクスポートの TRANSFORM パラメータの有効なオブジェクト型

	SEGMENT_ ATTRIBUTES	STORAGE	OID	PCTSPACE
CLUSTER	X	X		X
CONSTRAINT	X	X		X
INC_TYPE			X	
INDEX	X	X		X
ROLLBACK_ SEGMENT	X	X		X
TABLE	X	X	X	X
TABLESPACE	X			X
TYPE			X	

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
```

TRANSPORT_DATAFILES

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・モード・インポート、または TRANSPORTABLE=ALWAYS がエクスポート中に設定されている場合は表モードで、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。ソース・データベース・システムからターゲット・データベース・システムに、それらのファイルを事前にコピーしておく必要があります。

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。

例

次に、TRANSPORT_DATAFILES パラメータの使用例を示します。TRANSPORT_DATAFILES の値は引用符で囲まれるため、コマンドラインでエスケープ文字を使用する必要がないように、パラメータ・ファイルを使用することをお薦めします。(詳細は、3-8 ページの「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。) 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.f'
```

次のコマンドを発行します。

```
> impdp hr PARFILE=trans_datafiles.par
```

TRANSPORT_FULL_CHECK

デフォルト : n

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={y | n}
```

TRANSPORT_FULL_CHECK=y 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・セットの内部にあるオブジェクトと外部にある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=n 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・セットの外部にあるオブジェクトの依存オブジェクトが、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）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.f というデータ・ファイルがすでに存在するものとします。

TRANSPORT_DATAFILES の値は引用符で囲まれるため、コマンドラインでエスケープ文字を使用する必要がないように、パラメータ・ファイルを使用することをお薦めします。（詳細は、3-8 ページの「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。）たとえば、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=y
TRANSPORT_DATAFILES='/wkdir/data/tbs6.f'
```

次のコマンドを発行します。

```
> impdp hr PARFILE=full_check.par
```

TRANSPORT_TABLESPACES

デフォルト : デフォルト値は設定されていません。

用途

ネットワーク・リンクを介したトランスポータブル表領域モード・インポートの実行を指定します。

構文および説明

```
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。

これ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する必要があります。

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。
- トランスポータブル・モードは、暗号化された列をサポートしていません。

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.f 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。TRANSPORT_DATAFILES の値は引用符で囲まれるため、コマンドラインでエスケープ文字を使用する必要がないように、パラメータ・ファイルを使用することをお薦めします。(詳細は、3-8 ページの「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。）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=n
TRANSPORT_DATAFILES='user01/data/tbs6.f'
```

次のコマンドを発行します。

```
> impdp hr PARFILE=tablespaces.par
```

TRANSPORTABLE

デフォルト : NEVER

用途

表モードのインポート（TABLES パラメータで指定）の実行時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。

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: インポート・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。

NEVER: インポート・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ロードするように指示します。これがデフォルトです。

制限事項

- 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・インポートでのみ有効です（表はパーティション化またはサブパーティション化されている必要はありません）。
- トランスポータブル・インポートを実行するスキーマでは、ソース・データベースで EXP_FULL_DATABASE ロール、ターゲット・データベースで IMP_FULL_DATABASE ロールが必要になります。
- TRANSPORTABLE パラメータをすべて使用するには、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。

例

次に、ネットワーク・リンク・インポート中に TRANSPORTABLE パラメータを使用した例を示します。

```
> impdp system TABLES=hr.sales TRANSPORTABLE=always
  DIRECTORY=dpump_dir1 NETWORK_LINK=dbs1 PARTITION_OPTIONS=departition
  TRANSPORT_DATAFILES=datafile_name
```

VERSION

デフォルト: COMPATIBLE

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。なお、これは、10.1 より前のバージョンの Oracle Database でデータ・ポンプ・インポートが使用可能ということではありません。データ・ポンプ・インポートは、Oracle Database 10*g* 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・システムのバージョンより古い互換バージョンの 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。指定したバージョン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、指定したバージョン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、データベースのバージョンに対応します。
- *version_string*: 特定のデータベース・バージョン（11.1.0 など）。Oracle Database 11*g* の場合、9.2.0 以上の値を指定する必要があります。

参照: 「データベース・バージョンが異なる場合のデータ移動」（1-13 ページ）

例

次に、VERSION 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、2-21 ページの「FULL」を参照してください。

```
> impdp hr DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp TABLES=employees
VERSION=LATEST
```

オリジナルのインポート・ユーティリティのパラメータへのデータ・ポンプ・インポート・パラメータのマップ方法

表 3-2 は、データ・ポンプ・インポート・パラメータをオリジナルのインポート・パラメータにできるかぎり正確にマップしたものです。機能の設計変更で、オリジナルのインポート・パラメータが不要になったため、対応するデータ・ポンプ・コマンドがない場合もあります。また、表に示すとおり、パラメータ名が同じ場合もありますが、機能は多少異なります。

表 3-2 オリジナルのインポート・パラメータと、それらに対応するデータ・ポンプ・インポートのパラメータ

オリジナルのインポート・パラメータ	対応するデータ・ポンプ・インポート・パラメータ
BUFFER	BUFFER に相当するパラメータは不要になりました。
CHARSET	CHARSET に相当するパラメータは不要になりました。
COMMIT	COMMIT に相当するパラメータはサポートされなくなりました。
COMPILE	COMPILE に相当するパラメータはサポートされなくなりました。

表 3-2 オリジナルのインポート・パラメータと、それらに対応するデータ・ポンプ・インポートのパラメータ（続き）

オリジナルのインポート・パラメータ	対応するデータ・ポンプ・インポート・パラメータ
CONSTRAINTS	EXCLUDE=CONSTRAINT
DATAFILES	TRANSPORT_DATAFILES
DESTROY	REUSE_DATAFILES
FEEDBACK	STATUS
FILE	DUMPFILE
FILESIZE	不要。ダンプ・ファイル・セットに含まれています。
FROMUSER	SCHEMAS
FULL	FULL
GRANTS	EXCLUDE=GRANT
HELP	HELP
IGNORE	TABLE_EXISTS_ACTION
INDEXES	EXCLUDE=INDEX
INDEXFILE	SQLFILE（INCLUDE INDEX も指定）
LOG	LOGFILE
PARFILE	PARFILE
RECORDLENGTH	RECORDLENGTH に相当するパラメータは不要になりました。
RESUMABLE	RESUMABLE に相当するパラメータは不要になりました。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	RESUMABLE_NAME に相当するパラメータは不要になりました。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	RESUMABLE_TIMEOUT に相当するパラメータは不要になりました。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	CONTENT=METADATA_ONLY
ROWS=Y	CONTENT=ALL
SHOW	SQLFILE
SKIP_UNUSABLE_INDEXES	SKIP_UNUSABLE_INDEXES
STATISTICS	オリジナルのインポート・ユーティリティの STATISTICS パラメータには、ALWAYS、SAFE、RECALCULATE および NONE の 4 つの値を使用していました。 オリジナルのインポート・ユーティリティの STATISTICS=NONE に等しいデータ・ポンプ・パラメータは、EXCLUDE=STATISTICS です。 ソース表に統計がある場合、それらはデフォルトでインポートされるため、STATISTICS=ALWAYS|SAFE|RECALCULATE に相当するデータ・ポンプ・インポート・パラメータは不要になりました。

表 3-2 オリジナルのインポート・パラメータと、それらに対応するデータ・ポンプ・インポートのパラメータ（続き）

オリジナルのインポート・パラメータ	対応するデータ・ポンプ・インポート・パラメータ
STREAMS_CONFIGURATION	STREAMS_CONFIGURATION
STREAMS_INSTANTIATION	STREAMS_INSTANTIATION に相当するパラメータは不要になりました。
TABLES	TABLES
TABLESPACES	TABLESPACES
TOID_NOVALIDATE	TOID_NOVALIDATE に相当するコマンドは不要になりました。型比較に、OID は使用されなくなりました。
TOUSER	REMAP_SCHEMA
TRANSPORT_TABLESPACE	メタデータはダンプ・ファイル・セットに格納されるため、TRANSPORT_TABLESPACE に相当するパラメータは不要になりました。
TRANSPORT_TABLESPACE	
TTS_OWNERS	情報はダンプ・ファイル・セットに格納されるため、TTS_OWNERS に相当するパラメータは不要になりました。
USERID	USERID に相当するパラメータは不要になりました。この情報は、インポート・ユーティリティの起動時に、ユーザー名とパスワードで指定します。
VOLSIZE	テープはサポートされないため、VOLSIZE に相当するパラメータは不要になりました。

インポート・ユーティリティ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。

> **注意：** データ・ポンプ・インポートの対話方式コマンド・モードは、オリジナルのインポート・ユーティリティの対話方式モードとは異なります。このモードでは、入力のためのプロンプトが表示されます。オリジナルのインポート・ユーティリティの対話方式モードについては、20-6 ページの「対話方式モード」を参照してください。

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場合に有効です。

表 3-3 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・インポート・プロンプトから実行できる操作を示します。

表 3-3 データ・ポンプ・イン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
対話方式コマンド・モードを終了する。	CONTINUE_CLIENT（3-44 ページ）
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（3-44 ページ）
使用可能なコマンドの概要を表示する。	HELP（3-45 ページ）
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（3-45 ページ）
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、Enterprise Edition のみで使用可能です。	PARALLEL（3-45 ページ）
接続している停止ジョブを再開する。	START_JOB（3-46 ページ）
現行のジョブの詳細な状態を表示する。	STATUS（3-46 ページ）
現行のジョブを停止する。	STOP_JOB（3-47 ページ）

次の項では、データ・ポンプ・イン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について説明します。

CONTINUE_CLIENT

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
```

EXIT_CLIENT

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 では、ジョブが実行されたままになるため、ジョブがまだ「実行中」または「停止」状態になっている場合は、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

```
Import> EXIT_CLIENT
```

HELP

用途

対話方式コマンド・モードで使用可能なデータ・ポンプ・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
```

KILL_JOB

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、ジョブのプロセス構造が即時に停止し、マスター表およびダンプ・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

```
Import> KILL_JOB
```

PARALLEL

用途

現行のジョブに対してアクティブなワーカー・プロセスまたは PQ スレーブ（あるいはその両方）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。整数値を小さくすると、ワーカー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はされません。

> **参照：** 並列度の詳細は、3-22 ページの「PARALLEL」を参照してください。

制限事項

- PARALLEL は、Enterprise Edition のみで使用可能です。

例

```
Import> PARALLEL=10
```

START_JOB

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=y]
```

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・セットおよびマスター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 オプションは、以前一部の DDL 文が失敗したために再開に失敗したジョブを再開できます。失敗する文はスキップされ、ジョブは次の項目から再開されます。

SQLFILE ジョブもトランスポータブル表領域モード・インポートも再開できません。

例

```
Import> START_JOB
```

STATUS

用途

現行の操作の説明とともにジョブの状態を累積的に表示します。ジョブの完了率も返されます。

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
```

STOP_JOB

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・ユーティリティを終了します。

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マスター表およびダンプ・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。マスター・プロセスは、ワーカー・プロセスで現行のタスクが終了するまで待機はしません。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

```
Import> STOP_JOB=IMMEDIATE
```

データ・ポンプ・インポートの使用例

この項では、データ・ポンプ・インポートの使用例を示します。

- データのみ表モード・インポートの実行
- スキーマ・モード・インポートの実行
- ネットワーク・モード・インポートの実行

これらの例を正しく使用するために役立つ情報については、3-7 ページの「インポート・パラメータの使用例」を参照してください。

データのみ表モード・インポートの実行

例 3-1 に、employees 表のデータのみ表モード・インポートの実行方法を示します。例 2-1 で作成されたダンプ・ファイルを使用します。

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=y
```

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。

スキーマ・モード・インポートの実行

例 3-2 に、例 2-4 で作成したダンプ・ファイル・セットのスキーマ・モード・インポートを示します。

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 パラメータは、インポートに、すでに存在する場合は表を削除し、ダンプ・ファイルの内容を使用してその表を再作成してロードするように指定します。

ネットワーク・モード・インポートの実行

例 3-3 では、ソースが、NETWORK_LINK パラメータで指定されたデータベースであるネットワーク・モード・インポートを実行します。

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 IMP_FULL_DATABASE ロールおよびソース・データベースの EXP_FULL_DATABASE ロールが必要です。

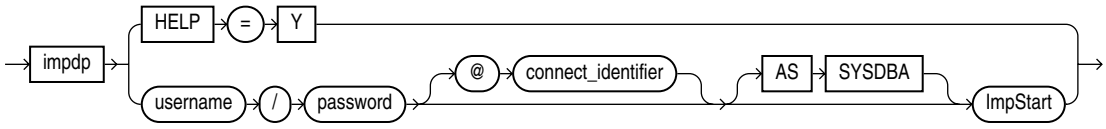
REMAP_SCHEMA は、ソース・スキーマにあるすべてのオブジェクトをターゲット・スキーマにロードします。

> **参照：** データベース・リンクの詳細は、3-20 ページの「NETWORK_LINK」を参照してください。

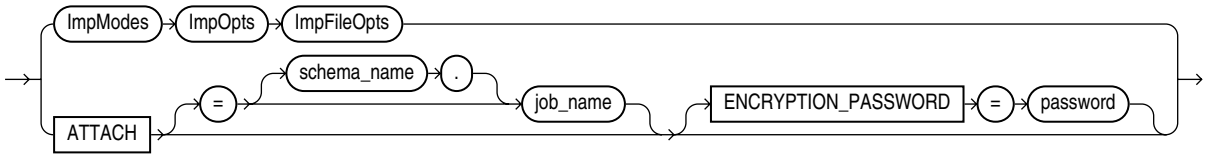
データ・ポンプ・インポートの構文図

この項では、データ・ポンプ・インポートの構文図を示します。これらの構文図では、標準SQL 構文の表記法を使用します。SQL 構文の表記については、『Oracle Database SQL 言語リファレンス』を参照してください。

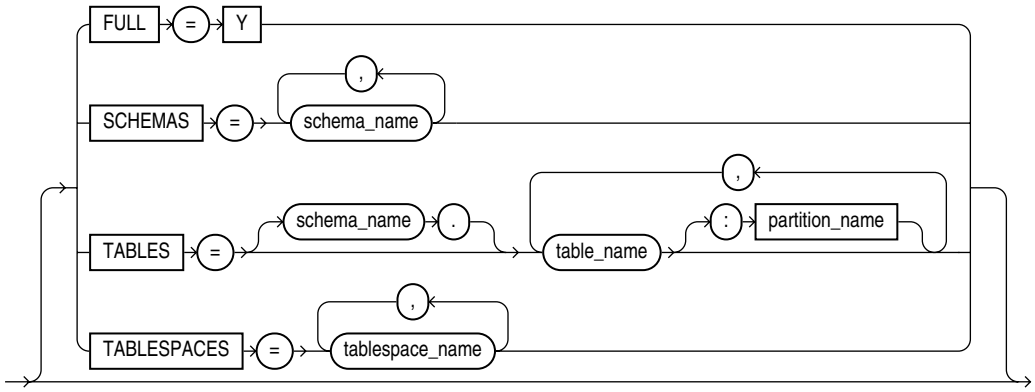
ImpInit



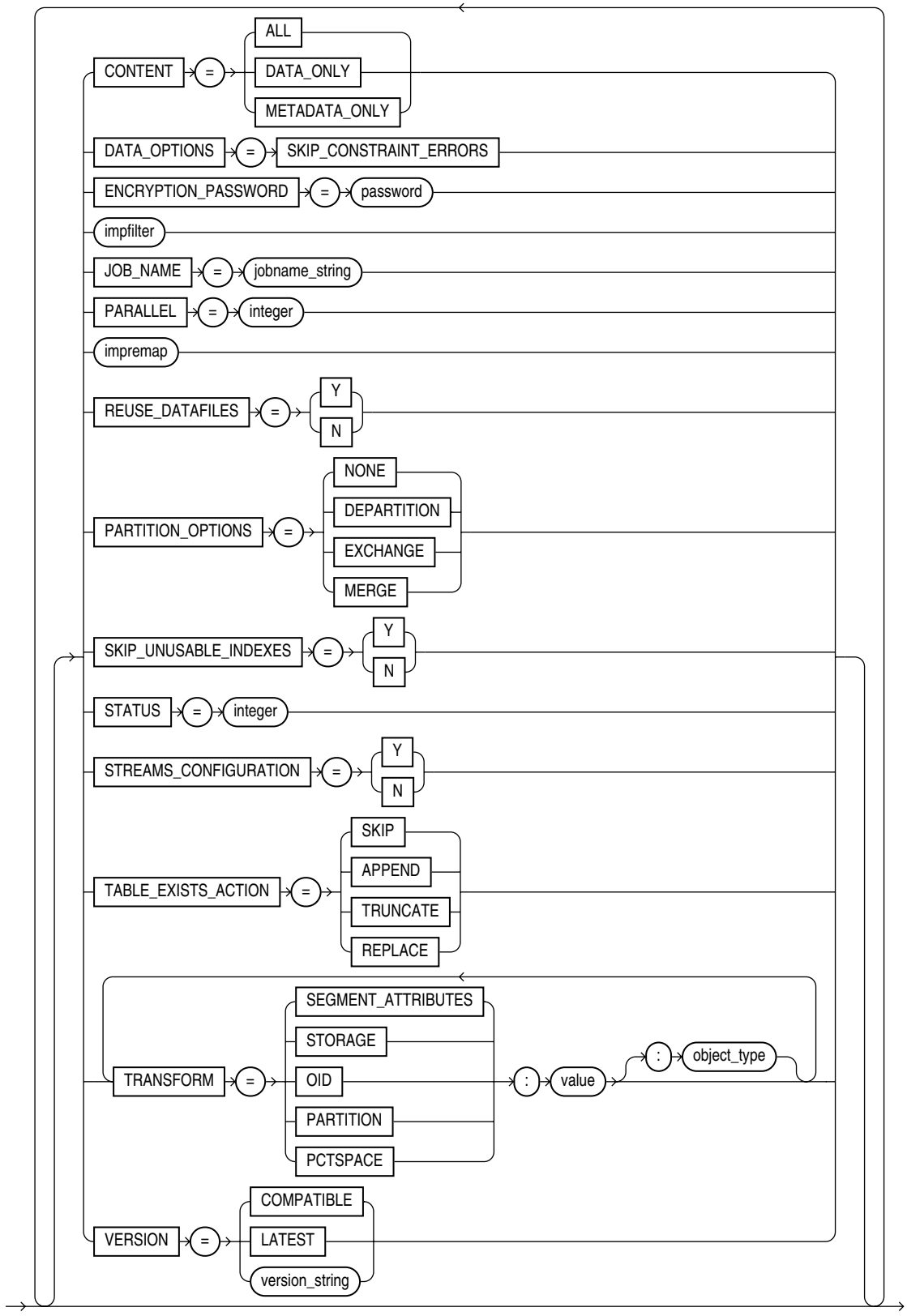
ImpStart



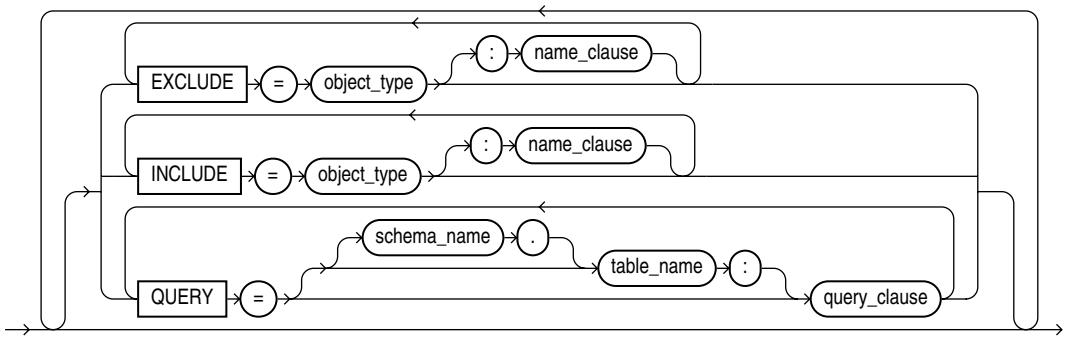
ImpModes



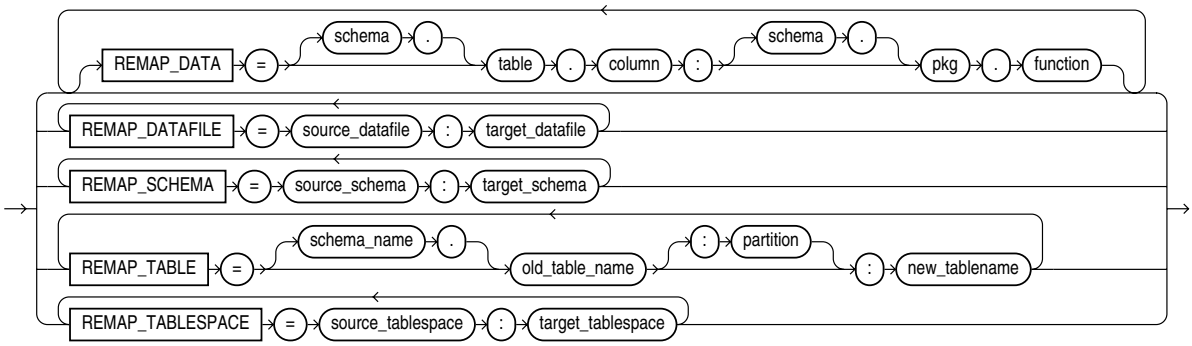
ImpOpts


CONTENT
=
ALL
DATA_ONLY
METADATA_ONLY
DATA_OPTIONS
=
SKIP_CONSTRAINT_ERRORS
ENCRYPTION_PASSWORD
=
password
impfilter
JOB_NAME
=
jobname_string
PARALLEL
=
integer
impremap
REUSE_DATAFILES
=
Y
N
PARTITION_OPTIONS
=
NONE
DEPARTITION
EXCHANGE
MERGE
SKIP_UNUSABLE_INDEXES
=
Y
N
STATUS
=
integer
STREAMS_CONFIGURATION
=
Y
N
TABLE_EXISTS_ACTION
=
SKIP
APPEND
TRUNCATE
REPLACE
TRANSFORM
=
SEGMENT_ATTRIBUTES
STORAGE
OID
PARTITION
PCTSPACE
:
value
:
object_type
VERSION
=
COMPATIBLE
LATEST
version_string

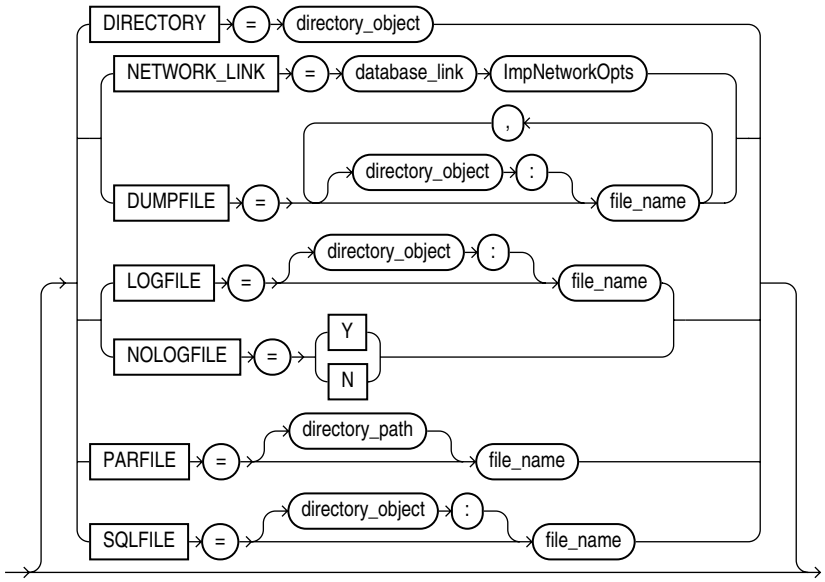
ImpFilter



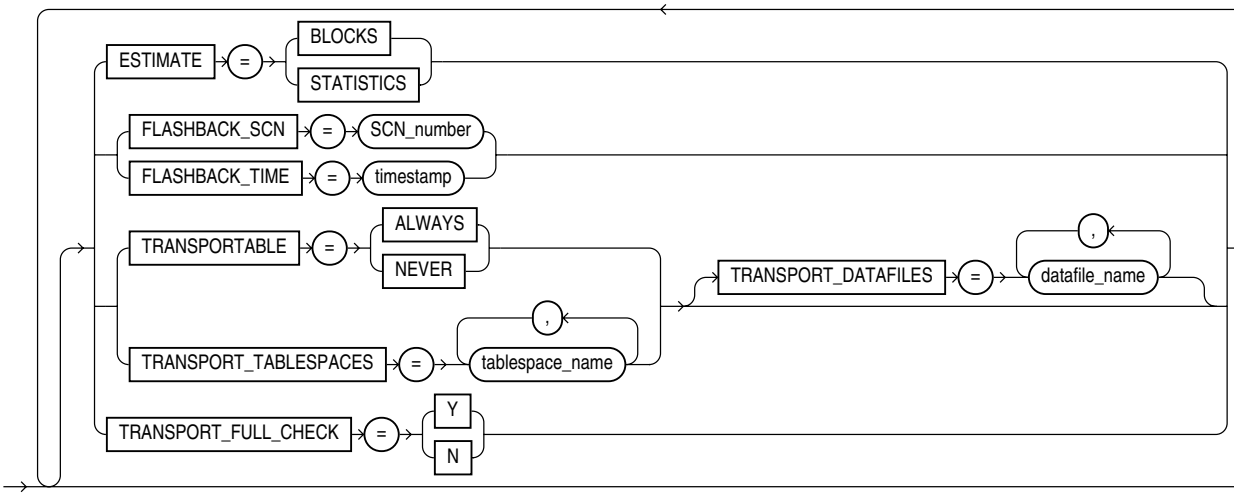
ImpRemap



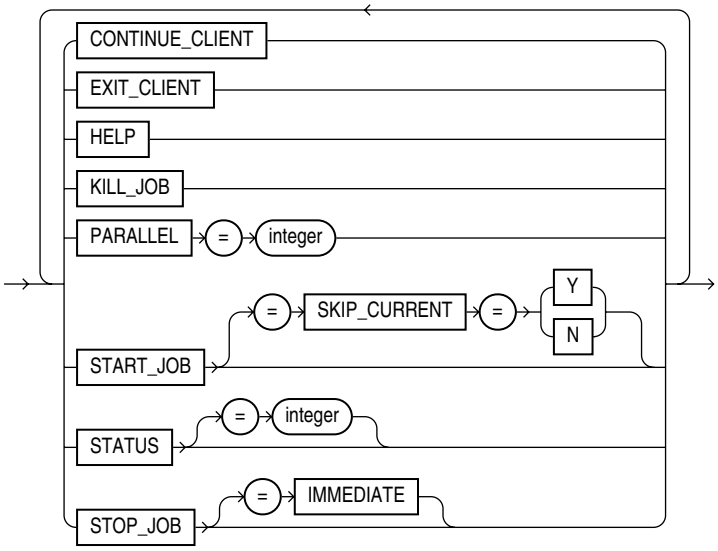
ImpFileOpts



ImpNetworkOpts



ImpDynOpts



4

データ・ポンプ・ユーティリティのパフォーマンス

データ・ポンプ・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。サイトに、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、パフォーマンスが大幅に向上します。この章では、パフォーマンス向上の理由を簡単に説明し、エクスポートおよびインポート操作のパフォーマンスを向上させるための具体的な手順を示します。

この章の内容は、次のとおりです。

- 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・ユーティリティのデータ・パフォーマンスの改善点
- パフォーマンスのチューニング
- 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

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。

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・ユーティリティのデータ・パフォーマンスの改善点

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。
- データ・ポンプでは、索引の作成およびパッケージ本体のロードにパラレル処理を使用します。
- ダンプ・ファイルは、直接サーバーで読取りおよび書込みが行われるため、データをクライアントに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。これは、表領域内部の Oracle Database のデータ・ファイルに格納されている形式と同様の形式です。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 のパラレル問合せおよびパラレル DML 機能を効率的に使用できます。
- 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。

パフォーマンスのチューニング

データ・ポンプ・テクノロジでは、すべての使用可能なリソースを最大限に使用して、スループットの最大化およびジョブ経過時間の最小化が行われます。これを実現するには、システムで CPU、メモリー、I/O 間でバランスがと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・セット内のダンプ・ファイルの書込みおよび読取りはパラレルで行われるため、最大のパフォーマンスを得るにはそれらのダンプ・ファイルを個別のディスクに常駐させる必要があります。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。これを実行するには、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。（データ・ポンプ固有のチューニング・パラメータは PARALLEL パラメータのみです。）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。

参照：

- エクスポート・ユーティリティの PARALLEL パラメータの詳細は、2-27 ページの「PARALLEL」を参照してください。
- インポート・ユーティリティの PARALLEL パラメータの詳細は、3-22 ページの「PARALLEL」を参照してください。

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。

`PARALLEL` パラメータは、Oracle Database 11*g* の Enterprise Edition でのみ有効です。

圧縮および暗号化によるパフォーマンスへの影響

圧縮および暗号化に関連するデータ・ポンプ・パラメータを使用すると、エクスポート操作とインポート操作のパフォーマンスが低下する場合があります。これは、RAW データの変換を実行するために、追加の CPU リソースを必要とするためです。

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。

Streams 環境でのバッファ・キャッシュ・サイズの設定

Oracle Data Pump は、Streams 機能を使用してプロセス間の通信を行います。`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 10M にすることをお薦めします。

参照：『Oracle Streams 概要および管理』

5
データ・ポンプ API

データ・ポンプ API（DBMS_DATAPUMP）は、あるサイトのデータおよびメタデータのすべてまたは一部をデータベース間で移動するための高速メカニズムを提供します。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・ユーティリティはデータ・ポンプ API に基づいています。

この章では、データ・ポンプ API の機能を詳細に説明します。この章の内容は、次のとおりです。

- データ・ポンプ API のクライアント・インタフェースの動作
- データ・ポンプ API を使用する場合の基本手順
- データ・ポンプ API の使用例

参照：

- DBMS_DATAPUMP パッケージで使用可能なプロシージャの詳細は、『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』を参照してください。
- データ・ポンプの概念の詳細は、第 1 章「Oracle Data Pump の概要」を参照してください。

データ・ポンプ API のクライアント・インタフェースの動作

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 ファンクションまたは 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。この機能によって、DBA は帰宅前にジョブを開始して、自宅でその進捗状況を確認できます。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。

ジョブの状態

次に、各フェーズに関連付けられたジョブの状態を示します。

- 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終了し、データ・ポンプ・プロセスを終了中
- 完了 : ジョブの完了時
- 停止保留 : 手順に従ったジョブの停止が要求された場合
- 停止処理中 : 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に 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（関連付けられたデータ・ポンプ・プロセスが存在しない）ジョブに対してマスター表が存在する状態

「アイドル」状態のジョブに対して DBMS_DATAPUMP.START_JOB を実行すると、「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切断すると、そのジョブはデータベースから完全に削除されます。

ジョブが異常終了した場合またはジョブを実行しているインスタンスが停止した場合、「実行中」または「アイドル」状態のジョブは「未実行」状態になります。ユーザーは、その状態からジョブを再開できます。

マスター制御プロセスは、「定義中」、「アイドル」、「実行中」、「停止処理中」、「停止保留」、「完了処理中」の状態でアクティブです。また、一時的に「停止」および「完了」の状態にもなります。ジョブのマスター表は、「未定義」状態を除いてすべての状態で存在します。ワーカー・プロセスは「実行中」および「停止保留」状態、つまりインポート・ジョブに対する「定義中」状態でのみアクティブです。

ジョブの状態が「実行中」の場合に接続を切断しても、そのジョブは停止しません。実行中のジョブにはいつでも再接続して、ジョブに関する状態情報を再度取得できます。

DBMS_DATAPUMP.DETACH プロシージャが実行されると、明示的に切断されることがあります。または、データ・ポンプ API セッションが停止した場合や、データ・ポンプ API がデータ・ポンプ・ジョブと通信できない場合、もしくは DBMS_DATAPUMP.STOP_JOB プロシージャが実行された場合は、暗黙的に切断されることがあります。

「未実行」状態は、実行中のジョブのコンテキストの外部にマスター表が存在することを示します。この状態は、（後で再開するために）ジョブが停止された場合またはジョブが異常終了した場合に発生します。また、この状態は、ジョブの開始時に行われるジョブの状態の移行中、およびマスター表を削除する前に行うジョブの終了時に、一時的に発生する場合があります。「未実行」状態は、DBA_DATAPUMP_JOBS ビューおよび USER_DATAPUMP_JOBS ビューでのみ表示されます。GET_STATUS プロシージャから返されることはありません。

表 5-1 に、DBMS_DATAPUMP プロシージャを実行できる有効なジョブの状態を示します。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 5-1 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、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にのみ有効）	ダンプ・ファイル・セット、ログ・ファイル、または SQL_FILE の出力用のファイルを指定する。
ATTACH	定義中、実行中、アイドル、停止、完了、完了処理中、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する。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または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	すべての状態	ユーザー・セッションをジョブから切断する。
GET_DUMPFILE_INFO	すべての状態	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、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[1]	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。
STOP_JOB	定義中、実行中、アイドル、停止保留	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。

[1] ENCRYPTION_PASSWORD パラメータは、「アイドル」状態および「定義中」状態のときに入力できます。

データ・ポンプ API を使用する場合の基本手順

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。次の手順では、データ・ポンプ API の使用に関連する基本操作を示します。通常操作を行う順序でこれらの手順を示します。

1. DBMS_DATAPUMP.OPEN プロシージャを実行して、データ・ポンプ・ジョブとのそのインフラストラクチャを作成します。
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. オプションで、ジョブを完了まで監視できます。
5. オプションで、ジョブとの接続を切断し、後で再接続します。
6. オプションで、ジョブを停止します。
7. オプションで、ジョブを再開します。

これらの手順の概要は、次の項で示す使用例で説明します。

> **参照：** DBMS_DATAPUMP パッケージの詳細は、『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』を参照してください。

データ・ポンプ API の使用例

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。

- 例 5-1「簡単なスキーマ・エクスポートの実行」
- 例 5-2「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」
- 例 5-3「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 5-1 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 5-2 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 5-1（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 5-3 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 5-1 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できます。

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, we will monitor
-- the job 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;
/
```

第 II 部

SQL*Loader

第 II 部では、SQL*Loader ユーティリティについて説明します。

第 6 章「SQL*Loader の概念」

この章では、SQL*Loader の機能について説明します。また、データ・ロードの概念（オブジェクト・サポートも含む）についても説明します。さらに、SQL*Loader への入力、データベースの事前準備および SQL*Loader からの出力についても説明します。

第 7 章「SQL*Loader コマンドライン・リファレンス」

この章では、SQL*Loader で使用するコマンドラインの構文について説明します。コマンドライン引数、SQL*Loader のメッセージを抑制する方法、バインド配列のサイズ指定などについて説明します。

第 8 章「SQL*Loader 制御ファイル・リファレンス」

この章では、SQL*Loader の設定に使用する制御ファイルの構文、およびデータを Oracle の形式にマップする方法を SQL*Loader に記述する方法について説明します。詳細な構文図を示すとともに、データ・ファイル、表と列、データの位置、ロードするデータの型と形式などの指定に関する情報についても説明します。

第 9 章「SQL*Loader フィールド・リスト・リファレンス」

この章では、SQL*Loader 制御ファイルのフィールド・リストのセクションについて説明します。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。

第 10 章「オブジェクト、LOB およびコレクションのロード」

この章では、様々な形式での列オブジェクトのロード方法について説明します。オブジェクト表、`REF` 列、LOB およびコレクションのロード方法についても説明します。

第 11 章「従来型パス・ロードおよびダイレクト・パス・ロード」

この章では、従来型パス・ロードとダイレクト・パス・ロードの違いを説明します。ダイレクト・パス・ロードは、大量のデータを従来より高速にロードするための高パフォーマンス・オプションです。

6

SQL*Loader の概念

この章では、SQL*Loader による Oracle Database へのデータのロードについて、基本的な概念を説明します。この章の内容は、次のとおりです。

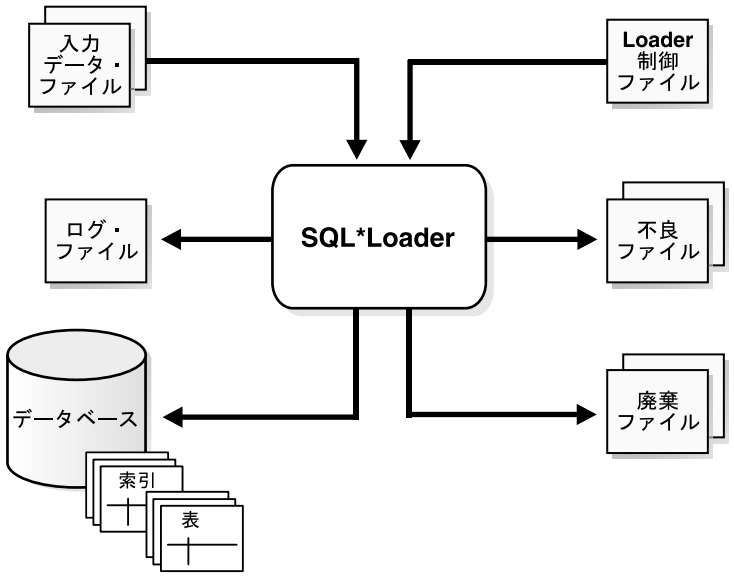
SQL*Loader の機能

SQL*Loader を使用して、外部ファイルのデータを Oracle Database の表にロードします。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。

- ネットワークを介したデータのロード（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）。
- 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。
- データのキャラクタ・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（SDF）を使用した、LOB およびコレクションのロード。
- 従来型パス・ロードまたはダイレクト・パス・ロードの使用。従来型パス・ロードでは高い柔軟性を、ダイレクト・パス・ロードでは優れたロード・パフォーマンスを提供します。詳細は、第 11 章を参照してください。

一般的な SQL*Loader セッションでは、SQL*Loader の動作を制御する制御ファイルと 1 つ以上のデータ・ファイルが入力用に使用されます。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、不良ファイルで、廃棄ファイルに出力される場合もあります。図 6-1 に、SQL*Loader セッションの流れの例を示します。

図 6-1 SQL*Loader の概要



SQL*Loader のパラメータ

SQL*Loader は、sqlldr コマンドを指定すると起動します。また、オプションで、セッション特性を確立するパラメータを指定した場合も起動します。

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、次の方法でパラメータを指定すると効率的です。

- パラメータは、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 パラメータを使用して、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。
- 一部のパラメータは、OPTIONS 句を使用して、SQL*Loader の制御ファイル内に指定することもできます。

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。

参照：

- SQL*Loader のパラメータの詳細は、第 7 章を参照してください。
- 「PARFILE（パラメータ・ファイル）」（7-8 ページ）
- 「OPTIONS 句」（8-3 ページ）

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。

制御ファイルには、大きく分けて 3 つのセクションがあります。

第 1 セクションには、セッション全体の情報が記述されます。たとえば、次のような情報です。

- バインドサイズ、行、スキップ・レコードなどのようなグローバル・オプション
- 入力データの配置先を指定する
- INFILE 句
- ロードされるデータ

第 2 セクションは、1 つ以上の INTO TABLE ブロックで構成されています。それぞれのブロックには、表名、その表の列などの、データがロードされる表についての情報が含まれています。

第 3 セクションはオプションで、このセクションがある場合は、入力データが記述されます。

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）。
- 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。
- 先頭にハイフンを 2 つ（--）続けて入力することによって、コメントを挿入できます。ハイフンから行の終わりまでがコメントになります。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。

参照： 制御ファイルの構文およびセマンティクスの詳細は、第 8 章を参照してください。

入力データおよびデータ・ファイル

制御ファイルに指定された 1 つ以上のファイルなどから、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。

注意： 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。この形式は柔軟性はありませんが、その結果、可変長またはストリーム形式よりも高いパフォーマンスを得ることができます。また、固定形式は簡単に指定できます。次に例を示します。

```
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 *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。

例 6-1 に、固定レコード形式で解釈されるようにデータ・ファイルを指定する制御ファイルを示します。この例では、5 つの物理レコードがあります。ピリオド（.）が空白を示すと想定すると、第 1 の物理レコードは [001,...cd,.] で、11 バイト（シングルバイト・キャラクタ・セットと想定）です。第 2 のレコードは [0002,fghi,\n] で、改行文字（11 バイト目）が続きます。改行文字は、固定レコード形式では必要ありません。

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。詳細は、8-17 ページの「文字長セマンティクス」を参照してください。

例 6-1 固定レコード形式でのデータのロード

```
load data
infile 'example.dat'  "fix 11"
into table example
fields terminated by ',' optionally enclosed by '"'
(col1, col2)

example.dat:
001,   cd, 0002,fghi,
00003,lmn,
1, "pqrs",
0005,uvwx,
```

可変レコード形式

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。指定しない場合、長さは 5 バイトとみなされます。*n* に、40 より大きい値を指定すると、エラーになります。

例 6-2 に、ファイル名が example.dat でレコード長フィールドが 3 バイト長の可変レコード形式の入力データに対する制御ファイルの指定を示します。example.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（すなわち 9）バイト長、2 つ目のレコードは 010（すなわち 1 バイトの改行を含めて 10）バイト長、3 つ目は 012（同様に 1 バイトの改行を含む）バイト長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。ここでは、シングルバイト・キャラクタ・セットであるとします。

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。詳細は、8-17 ページの「文字長セマンティクス」を参照してください。

例 6-2 可変レコード形式でのデータのロード

```
load data
infile 'example.dat'  "var 3"
into table example
fields terminated by ',' optionally enclosed by '"'
(col1 char(5),
 col2 char(7))

example.dat:
009hello,cd,010world,im,
012my,name is,
```

ストリーム・レコード形式

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、パフォーマンスに影響する場合があります。ストリーム・レコード形式として指定するには、次のように指定します。

```
INFILE datafile_name ["str terminator_string"]
```

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、*X'hex_string'* で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列（*'char_string'*）はバックスラッシュを使用すると指定できます。次に例を示します。

- \n LF
- \t 水平タブ
- \f 改ページ
- \v 垂直タブ
- \r 改行

セッションに対して NLS_LANG パラメータに指定されているキャラクタ・セットが、データ・ファイルのキャラクタ・セットと異なる場合、文字列はデータ・ファイルのキャラクタ・セットに変換されます。これは、SQL*Loader でデフォルトのレコード終了記号が確認される前に行われます。

16 進文字列はデータ・ファイルのキャラクタ・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 LF 文字 \n が使用されます。

Windows NT では、*terminator_string* を指定しない場合、¥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がレコード終了記号として使用されます。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、terminator_string を指定する必要があります。

例 6-3 に、terminator_string が文字列 '|\n' で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。

例 6-3 ストリーム・レコード形式でのデータのロード

```
load data
infile 'example.dat'  "str '|\n'"
into table example
fields terminated by ',' optionally enclosed by '"'
(col1 char(5),
 col2 char(7))

example.dat:
hello,world,|
james,bond,|
```

論理レコード

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が、複数の物理レコードが 1 つの論理レコードに結合される場合もあります。

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

参照 :

- 「物理レコードからの論理レコードの作成」(8-21 ページ)
- 「事例 4: 結合された物理レコードのロード」(事例へのアクセス方法については、6-12 ページの「SQL*Loader の事例」を参照)

データ・フィールド

論理レコードが作成されると、フィールドが設定されます。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。
- 特殊なデータ・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。

- Length-Value データ型が使用できます。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。

  参照：

  - 「データ・フィールドの位置指定」（9-2 ページ）
  - 「デリミタの指定」（9-18 ページ）

LOBFILE および SDF

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）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、従業員名、従業員 ID および従業員の履歴をロードする場合に LOBFILE を使用するとします。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場合に LOBFILE を使用するとします。`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。

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、制御ファイルごとに指定されます。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 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。

参照：

- 「LOBFILE からの LOB データのロード」（10-16 ページ）
- 「セカンダリ・データ・ファイル（SDF）」（10-23 ページ）

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 つの手順があります。

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析し、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。
2. Oracle Database 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 制御ファイルで定義されているフィールドのデータ型が、データベースの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。

廃棄レコードおよび拒否レコード

入力ファイルから読み込まれたすべてのレコードが、データベースに挿入されるわけではありません。挿入されないレコードは、不良ファイルまたは廃棄ファイルに書き込まれます。

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。ファイルの名前はデータ・ファイルと同じで、拡張子は .bad になります。次に、その理由を示します。

SQL*Loader による拒否

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。

Oracle Database による拒否

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、Oracle Database に送られ、行として表に挿入されます。Oracle Database 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、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。

参照 :

- 「不良ファイルの指定」(8-10 ページ)
- 「事例 4: 結合された物理レコードのロード」(事例へのアクセス方法については、6-12 ページの「SQL*Loader の事例」を参照)

廃棄ファイル

SQL*Loader の実行によって、廃棄ファイルをコールするファイルが作成されることがあります。ファイルが作成されるのは必要な場合のみで、廃棄ファイルを使用可能にすることを指定してある場合にかぎります。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。

したがって、廃棄ファイルには、データベースのどの表にも挿入されなかったレコードが格納されます。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。

参照 :

- 「事例 4: 結合された物理レコードのロード」(事例へのアクセス方法については、6-12 ページの「SQL*Loader の事例」を参照)
- 「廃棄ファイルの指定」(8-11 ページ)

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。ログ・ファイルを作成できない場合、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。

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、次の方法があります。

- 従来型パス・ロード
- ダイレクト・パス・ロード
- 外部表ロード

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にコピーされます。バインド配列が一杯になるか、または最終レコードが読み込まれた時点で配列の挿入が実行されます。

参照：

- 「データのロード方法」（11-2 ページ）
- 「バインド配列および従来型パス・ロード」（8-33 ページ）

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。

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。

参照： 「ダイレクト・パス・ロード」（11-5 ページ）

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。

参照： 「パラレル・データ・ロード・モデル」（11-23 ページ）

外部表ロード

外部表ロードでは、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・ファイルの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。

- 外部表ロードでは、パラレルでデータ・ファイルをロードできます。データ・ファイルが大きい場合、パラレルでファイルをロードできます。
- 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。

注意： Windows NT では、名前付きパイプを使用する外部表ロードはサポートされていません。

参照：

- 第 12 章「外部表の概要」
- 第 13 章「ORACLE_LOADER アクセス・ドライバ」

外部表および SQL*Loader の選択

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、状況によって、より適切な方法を選択する必要があります。

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

オブジェクト、コレクションおよび LOB のロード

オブジェクト、コレクションおよび LOB のバルク・ロードに SQL*Loader を使用できます。オブジェクトの概念およびオブジェクト・サポートの Oracle 実装の詳細は、『Oracle Database 概要』および『Oracle Database 管理者ガイド』を参照してください。

サポートされるオブジェクト型

SQL*Loader では、次の 2 つの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。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。

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。

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。

参照：

- 「列オブジェクトのロード」（10-2 ページ）
- 「オブジェクト表のロード」（10-9 ページ）

サポートされるコレクション型

SQL*Loader では、次の 2 つの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。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。

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、最大数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。

参照： SQL*Loader 制御ファイルの DDL 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、10-21 ページの「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。

サポートされる LOB 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 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キャラクタ・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 は列データ型で、NCLOB 以外は、オブジェクトの属性データ型です。LOB には、実際の値、NULL または「値なし（空）」を指定できます。

参照： SQL*Loader 制御ファイルの DDL を使用してこれらの LOB 型をロードする方法の詳細は、10-13 ページの「LOB のロード」を参照してください。

パーティション・オブジェクトのサポート

SQL*Loader では、データベース内のパーティション・オブジェクトのロードをサポートしています。Oracle Database 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、2000 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

アプリケーション開発 : ダイレクト・パス・ロード API

アプリケーション開発のために、ダイレクト・パス・ロード API が提供されています。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。

SQL*Loader の事例

SQL*Loader の機能が様々な事例で示されています。事例は、ユーザー `scott` が、デモンストレーション用の Oracle Database の `emp` 表および `dept` 表を所有している場合を想定して作成してあります。（事例の中では、さらに列が追加されていることもあります。）

事例は、簡単なものから複雑なものの順に 1 ～ 11 の番号が付けられています。

次に、事例の概要を示します。

- 事例 1: 可変長データのロード : ストリーム形式のレコードをロードします。ここで扱うレコードのフィールドは、カンマで区切るか、または引用符で囲みます。データは制御ファイルの終わりに入っています。
- 事例 2: 固定形式フィールドのロード : 別のデータ・ファイルからデータをロードします。
- 事例 3: 自由区分形式ファイルのロード : フィールドがデリミタ付きで順序番号が付いている、ストリーム形式のレコードのデータをロードします。データは制御ファイルの終わりに入っています。
- 事例 4: 結合された物理レコードのロード : 複数の物理レコードを結合して、データベースの 1 行に対応する論理レコードを構成します。
- 事例 5: 複数表へのデータのロード : 1 回の実行で、データを複数の表にロードします。
- 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 : ダイレクト・パス・ロード方法を使用してデータをロードします。
- 事例 7: 書式化されたレポートからのデータの抽出 : 書式化されたレポートからデータを抽出します。
- 事例 8: パーティション化された表のロード : パーティション表をロードします。
- 事例 9: LOBFILE のロード（CLOB）: `resume` という `CLOB` 列を `emp` 表に追加し、`FILLER` フィールド（`res_file`）を使用して、複数の LOBFILE を `emp` 表にロードします。
- 事例 10: REF フィールドおよび VARRAY のロード : OID を主キーとして利用するカスタマ表をロードして、注文した商品を `VARRAY` に格納します。カスタマ表および `VARRAY` の注文商品を参照する注文表をロードします。
- 事例 11: Unicode キャラクタ・セットのデータのロード : Unicode キャラクタ・セットの UTF16 データを、リトル・エンディアンのバイト順序でロードします。この事例は、文字長セマンティクスを使用します。

事例用ファイル

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 のインストール時にインストールされます。ファイルは、$ORACLE_HOME/rdbms/demo ディレクトリにあります。

事例用のサンプル・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。

表 6-1 に、各事例に関連のあるファイルを示します。

表 6-1 事例用ファイルおよび関連ファイル

事例	.ctl	.dat	.sql
1	ulcase1.ctl	なし	ulcase1.sql
2	ulcase2.ctl	ulcase2.dat	なし
3	ulcase3.ctl	なし	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	なし	ulcase10.sql
11	ulcase11.ctl	ulcase11.dat	ulcase11.sql

事例の実行

通常、事例は次の手順で実行します（カレント・ディレクトリは、事例ファイルの格納場所である $ORACLE_HOME/rdbms/demo にしてください）。

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、次のように入力します。

   ```
   SQL> @ulcase1
   ```

   事例用の表が準備および移入されると、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、次のように入力し、SQL*Loader を起動して事例を実行します。

   ```
   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。

実行する事例に固有な注意事項がないかどうか、制御ファイルの内容を確認してください。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。

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。

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。次のように行います。

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、emp 表がロードされた場合、次のように入力します。

   ```
   SQL> SELECT * FROM emp;
   ```

   emp 表の各行の内容が表示されます。

7

SQL*Loader コマンドライン・リファレンス

この章では、SQL*Loader を起動するために使用するコマンドライン・パラメータについて説明します。この章の内容は、次のとおりです。

SQL*Loader の起動

SQL*Loader を起動する際に、セッションの特性を確立するためにパラメータを指定します。必要に応じて、パラメータはカンマで区切ることができます。

パラメータは、キーワードまたは位置のいずれかで指定できます。キーワードで指定するとは、パラメータの名前と値を指定するということです。次の例では、CONTROL パラメータに ulcase1.ctl という制御ファイルの名前が指定されています。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。

```
> sqlldr CONTROL=ulcase1.ctl
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
```

キーワードでの指定が使用されると、その後、位置では指定できなくなります。たとえば、次のコマンドラインでは、ulcase1.log の位置が正しくても、エラーが発生します。

```
> sqlldr scott CONTROL=ulcase1.ctl ulcase1.log
```

パラメータを指定しないで SQL*Loader を起動すると、指定可能なパラメータとそのデフォルト値を示すヘルプ画面が表示されます。

> **参照：** すべてのコマンドライン・パラメータの詳細は、7-2 ページの「コマンドライン・パラメータ」を参照してください。

パラメータ指定の代替方法

コマンドラインの長さが、システムのコマンドラインの最大長を超える場合は、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。このパラメータ・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。

> **参照：**
>
> - 「OPTIONS 句」（8-3 ページ）
> - 「PARFILE（パラメータ・ファイル）」（7-8 ページ）

コマンドライン・パラメータ

この項では、SQL*Loader の各コマンドライン・パラメータについて説明します。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、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、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。

BAD（不良ファイル）

デフォルト：拡張子が .bad のデータ・ファイル名

SQL*Loader によって作成される不良ファイルの名前を指定します。このファイルには、挿入中にエラーの原因となったレコード、または形式が不適切なレコードが格納されます。ファイル名を指定しない場合、デフォルトが使用されます。拒否されたレコードがない場合、不良ファイルが自動的に作成されることはありません。

コマンドラインで指定された不良ファイル名は、制御ファイルの最初の INFILE 文に関連する不良ファイル名になります。つまり、制御ファイル中で指定した不良ファイル名よりも、コマンドラインで指定した不良ファイル名の方が優先されます。

> **参照：** 不良ファイルの形式の詳細は、8-10 ページの「不良ファイルの指定」を参照してください。

BINDSIZE（最大サイズ）

デフォルト：このパラメータのデフォルト値を参照するには、7-2 ページの「SQL*Loader の起動」で説明したとおり、任意のパラメータを指定しないで SQL*Loader を起動します。

バインド配列の最大サイズ（バイト単位）を指定します。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。

> **参照：**
>
> - 「バインド配列および従来型パス・ロード」（8-33 ページ）
> - 「READSIZE（読取りバッファ・サイズ）」（7-8 ページ）

COLUMNARRAYROWS

デフォルト：このパラメータのデフォルト値を参照するには、7-2 ページの「SQL*Loader の起動」で説明したとおり、任意のパラメータを指定しないで SQL*Loader を起動します。

行数を指定してダイレクト・パス列配列に割り当てます。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。

> **参照：**
>
> - 「CONCATENATE を使用した論理レコードの作成」（8-21 ページ）
> - 「列配列の行数およびストリーム・バッファ・サイズの指定」（11-16 ページ）

CONTROL（制御ファイル）

デフォルト：なし

データのロード方法を記述する制御ファイルの名前を指定します。ファイルの拡張子またはファイル・タイプを指定していない場合は、デフォルトで .ctl になります。省略すると、SQL*Loader からファイル名の入力を要求されます。

SQL*Loader 制御ファイル名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、オペレーティング・システム上でファイル・システム・パスの中にバックスラッシュが使用されている場合は、複数のエスケープ文字を使用するか、または引用符でパスを囲む必要があります。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。

> **参照：** SQL*Loader 制御ファイルの詳細は、第 8 章を参照してください。

DATA（データ・ファイル）

デフォルト : 拡張子が .dat の制御ファイル名

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。ファイルの拡張子またはファイル・タイプを指定していない場合は、デフォルトで .dat になります。

コマンドラインでデータ・ファイルを指定し、INFILE で制御ファイル内のデータ・ファイルを指定する場合は、コマンドラインで指定されたデータが最初に処理されます。制御ファイル内で指定された最初のデータ・ファイルは無視されます。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。

ファイル処理オプションを指定する場合は、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。

DATE_CACHE

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。

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。このキャッシュが有効なのは、テキスト形式から日付書式に変換する場合と比べて、日付の検索コストが非常に低くなるためです。データ・ファイルに同じ日付が繰り返して存在する場合は、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。

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、ダイレクト・パス・ロードでのみ使用できます。デフォルトでは、この機能は使用可能です。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。

参照 : 「日付キャッシュの値の指定」（11-16 ページ）

DIRECT（データ・パス）

デフォルト : false

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。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。

参照 : 第 11 章「従来型パス・ロードおよびダイレクト・パス・ロード」

DISCARD（ファイル名）

デフォルト：拡張子が .dsc のデータ・ファイル名

SQL*Loader で作成する廃棄ファイル（オプション）を指定します。このファイルには、表に挿入されず、拒否もされなかったレコードが保存されます。

コマンドラインで指定された廃棄ファイル名は、制御ファイルの最初の INFILE 文に関連する廃棄ファイル名となります。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。

> **参照：** 廃棄ファイルの形式の詳細は、6-8 ページの「廃棄レコードおよび拒否レコード」を参照してください。

DISCARDMAX（整数）

デフォルト：ALL

廃棄レコードの最大数を指定します。廃棄レコード件数がここで指定した数に達すると、ロードは中止されます。最初の廃棄レコードで中止するには、1 を指定します。

ERRORS（エラーの許容最大数）

デフォルト：このパラメータのデフォルト値を参照するには、7-2 ページの「SQL*Loader の起動」で説明したとおり、任意のパラメータを指定しないで SQL*Loader を起動します。

挿入エラーの許容最大数を指定します。発生したエラーの数が ERRORS に指定された値を超えると、ロード処理が中止されます。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。

単一の表をロードする場合、エラーの数がこの上限を超えると、ロード処理が中止されます。ただし、その前に挿入されたデータはコミットされます。

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。

EXTERNAL_TABLE

デフォルト：NOT_USED

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。EXTERNAL_TABLE には、次の 3 つの値を指定できます。

- 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。)

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、DELETE ANY DIRECTORY 権限も必要です。

注意： 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、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。

参照：

- 事例へのアクセス方法については、6-12 ページの「SQL*Loader の事例」を参照してください。
- 第 12 章「外部表の概要」
- 第 13 章「ORACLE_LOADER アクセス・ドライバ」

EXTERNAL_TABLE の使用上の制限

EXTERNAL_TABLE 修飾子を使用する場合、次の制限が適用されます。

- 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。

FILE（ロード先の表領域ファイル）

デフォルト : なし

FILE は、エクステントを割り当てるデータベース・ファイルを指定します。このパラメータは、ダイレクト・パスのパラレル・ロードでのみ使用します。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。

参照 : 「パラレル・データ・ロード・モデル」(11-23 ページ)

LOAD（ロードするレコード数）

デフォルト : すべてのレコードがロードされます。

LOAD は、指定した件数のレコードをスキップした後に、ロードする論理レコード件数の最大数を指定します。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。

LOG（ログ・ファイル）

デフォルト : 拡張子が .log の制御ファイル名

LOG は、SQL*Loader によって作成されるログ・ファイルを指定します。このファイルには、ロード処理に関するログ情報が保存されます。

MULTITHREADING

デフォルト : 複数 CPU システムでは true で、単一 CPU システムでは false。

このパラメータは、ダイレクト・パス・ロードでのみ使用できます。

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。マルチスレッドを使用可能にすることによって、サーバー・システムでのストリームのロードと並行して、クライアント・システムでストリームを作成できます。

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。

参照 : 「複数 CPU システムのダイレクト・パス・ロードの最適化」
(11-17 ページ)

PARALLEL（パラレル・ロード）

デフォルト：false

ダイレクト・ロード時に複数の同時セッションによって同じ表にデータをロードできるかどうかを指定します。

参照： 「パラレル・データ・ロード・モデル」（11-23 ページ）

PARFILE（パラメータ・ファイル）

デフォルト：なし

コマンドラインで頻繁に使用するパラメータを記述したファイルを指定します。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・ファイルを指定した後、SQL*Loader によってパスワードを入力するように要求されます。次に例を示します。

```
sqlldr PARFILE=daily_report.par
Password: password
```

注意： 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。

READSIZE（読取りバッファ・サイズ）

デフォルト：このパラメータのデフォルト値を参照するには、7-2 ページの「SQL*Loader の起動」で説明したとおり、任意のパラメータを指定しないで SQL*Loader を起動します。

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。

デフォルトを使用しない場合は、READSIZE パラメータを使用して読取りバッファのサイズ（バイト単位）を指定できます。許容最大サイズは、プラットフォームによって異なります。

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 に 1000000 を設定すると、コミットを実行する前に、SQL*Loader によって、外部のデータ・ファイルから 1,000,000 バイト単位で読取りを実行できます。

注意： READSIZE に BINDSIZE より小さい値を指定した場合、READSIZE の値は増加します。

READSIZE パラメータは、LOB に影響しません。LOB 読取りバッファのサイズは、64KB に固定されています。

詳細は、7-3 ページの「BINDSIZE（最大サイズ）」を参照してください。

RESUMABLE

デフォルト : false

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。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。

参照 :

- 『Oracle Database 概要』
- 『Oracle Database 管理者ガイド』

RESUMABLE_NAME

デフォルト : 'User USERNAME (USERID), Session SESSIONID, Instance INSTANCEID'

再開可能な文を指定します。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 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。

RESUMABLE_TIMEOUT

デフォルト : 7200 秒（2 時間）

エラー修正に必要な時間を指定します。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 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。

ROWS（1 回にコミットする行数）

デフォルト : このパラメータのデフォルト値を参照するには、7-2 ページの「SQL*Loader の起動」で説明したとおり、任意のパラメータを指定しないで SQL*Loader を起動します。

ROWS に低い値を指定し、表圧縮を使用してデータを圧縮しようとすると、圧縮比が低下するため注意してください。データの圧縮には、高い値を指定するかデフォルト値を使用することをお薦めします。

従来型パス・ロードの場合のみ : ROWS でバインド配列の行数を指定します。詳細は、8-33 ページの「バインド配列および従来型パス・ロード」を参照してください。

ダイレクト・パス・ロードの場合のみ : ROWS でデータ・ファイルからデータのセーブ前に読み込む行数を指定します。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。詳細は、11-11 ページの「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。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。

注意 : 索引構成表（IOT）にデータがロードされる場合、あるいは VARRAY、XML 列または LOB が含まれる表にデータがロードされる場合、ROWS パラメータはダイレクト・パス・ロードで無視されます。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。

SILENT（フィードバック・モード）

SQL*Loader を開始すると、使用されている SQL*Loader のバージョンに関する情報が画面に表示され、ログ・ファイルに書き込まれます。また、SQL*Loader 実行中には次の例のようなフィードバック・メッセージも画面に表示されます。

```
Commit point reached - logical record count 20
```

SQL*Loader で次のようなデータ・エラー・メッセージが表示される場合もあります。

```
Record 4: Rejected - Error on table EMP
ORA-00001: unique constraint <name> violated
```

1 つ以上の値を持つ SILENT を指定して、これらのメッセージを抑止できます。

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。

```
SILENT=(HEADER, FEEDBACK)
```

適切な値を使用して、次のいずれかの処理を抑止します（複数も可）。

- 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: 画面に通常表示される「commit point reached」フィードバック・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。
- 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。
- 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。

SKIP（スキップされるレコード）

デフォルト : レコードは 1 件もスキップされません。

ファイルの先頭から何件の論理レコードをロード対象外とするかを指定します。

SKIP を指定すると、なんらかの理由で中断されたロードが継続されます。このパラメータは、従来型ロードおよび単一表のダイレクト・ロードで使用できます。複数の表のダイレクト・ロードに関しては、各表に対して同じ件数のレコードをロードする場合のみ使用できます。複数の表にそれぞれ異なる件数のレコードをダイレクト・ロードする場合は、使用できません。

WHEN 句を使用し、セカンダリ・データもロード対象となる場合、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。

参照 : 「中断されたロード」（8-19 ページ）

SKIP_INDEX_MAINTENANCE

デフォルト: false

ダイレクト・パス・ロードの索引メンテナンスを停止します。これは、従来型パス・ロードには適用できません。このオプションを指定すると、索引キーが付けられた索引パーティションには、索引キーのかわりに索引使用禁止が設定されます。これは、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です。ロードの影響を受けない索引セグメントについては、ロード前の索引使用禁止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。
- 索引を持つオブジェクトのパラレル・ロードを実行できます（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）。
- ロードによって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）作成します。

SKIP_UNUSABLE_INDEXES

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。デフォルトのデータベースの設定は、TRUE です。

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 構成パラメータのデータベース設定を使用します。初期化パラメータ・ファイルで SKIP_UNUSABLE_INDEXES のデータベース設定を指定しない場合、デフォルトのデータベース設定は TRUE になります。

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。これによってロード開始前の状態が使用禁止の索引を含む表をロードできます。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の索引はメンテナンスされず、ロード完了時も使用禁止状態のままです。

注意: 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。

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。

STREAMSIZE

デフォルト : このパラメータのデフォルト値を参照するには、7-2 ページの「SQL*Loader の起動」で説明したとおり、任意のパラメータを指定しないで SQL*Loader を起動します。

ダイレクト・パス・ストリームに対して、サイズをバイト単位で指定します。

参照 : 「列配列の行数およびストリーム・バッファ・サイズの指定」(11-16 ページ)

USERID（ユーザー名 / パスワード）

デフォルト : なし

各ユーザーの Oracle ユーザー名とパスワードを指定します。省略すると、システムから入力を要求されます。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。

また、ユーザー SYS として接続する場合は、接続文字列に AS SYSDBA を指定する必要があります。

注意 : 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、接続文字列全体を一重引用符で囲むか、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。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必要がある場合もあります。

システムの特殊文字および予約文字の詳細は、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。

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。プラットフォームによっては、SQL*Loader の実行結果はログ・ファイルに記録されるのみでなく、プロセス終了コードにも通知されます。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。表 7-1 に、それぞれの結果に対応する終了コードを示します。

表 7-1 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
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_FAIL

UNIX の場合、終了コードは次のようになります。

```
EX_SUCC 0
EX_FAIL 1
EX_WARN 2
EX_FTL  3
```

Windows NT の場合、終了コードは次のようになります。

```
EX_SUCC 0
EX_FAIL 1
EX_WARN 2
EX_FTL  4
```

SQL*Loader が 0（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 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。

8

SQL*Loader 制御ファイル・リファレンス

この章では、SQL*Loader 制御ファイルについて説明します。この章の内容は、次のとおりです。

制御ファイルの内容

SQL*Loader 制御ファイルは、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 構文図については、付録 A 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）

例 8-1 に、サンプル制御ファイルを示します。

例 8-1 サンプル制御ファイル

```
1    -- This is a sample 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. 制御ファイルにコメントを入力します。詳細は、8-3 ページの「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、付録 A 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、8-7 ページの「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、8-10 ページの「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、8-11 ページの「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、8-25 ページの「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、8-25 ページの「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。詳細は、8-24 ページの「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、8-27 ページの「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、相対位置に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、8-28 ページの「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。制御ファイルのこの部分の詳細は、第 9 章を参照してください。

制御ファイルのコメント

コメントはファイル中のコマンド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。次に例を示します。

```
--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。

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。OPTIONS 句は、制御ファイルを、通常使用するオプションの組合せで起動する場合に使用します。OPTIONS 句は、LOAD DATA 文の前に置きます。

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詳細は、第 7 章を参照してください。

```
BINDSIZE = n
COLUMNARRAYROWS = n
DATE_CACHE = n
DIRECT = {TRUE | FALSE}
ERRORS = n
EXTERNAL_TABEL = {NOT_USED | GENERATE_ONLY | EXECUTE}
FILE
LOAD = n
MULTITHREADING = {TRUE | FALSE}
PARALLEL = {TRUE | FALSE}
READSIZE = n
RESUMABLE = {TRUE | FALSE}
RESUMABLE_NAME = 'text string'
RESUMABLE_TIMEOUT = n
```

```
ROWS = n
SILENT = {HEADER | FEEDBACK | ERRORS | DISCARDS | PARTITIONS | ALL}
SKIP = n
SKIP_INDEX_MAINTENANCE = {TRUE | FALSE}
SKIP_UNUSABLE_INDEXES = {TRUE | FALSE}
STREAMSIZE = n
```

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) )
```

注意： 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。

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。この項では、次の項目について説明します。

- SQL および SQL*Loader の予約語と競合するファイル名
- SQL 文字列の指定
- オペレーティング・システムに関する考慮点

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に大 / 小文字の区別が必要な場合に使用する必要があります。

参照：『Oracle Database SQL 言語リファレンス』

SQL 文字列の指定

SQL 文字列を指定する場合は、二重引用符で囲みます。SQL 文字列を使用すると、SQL 演算子をデータフィールドに適用できます。

参照：「フィールドへの SQL 演算子の適用」(9-38 ページ)

オペレーティング・システムに関する考慮点

次の項では、ご使用のオペレーティング・システムによって異なる仕様について説明します。

完全パスの指定

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。多くの場合、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）二重引用符の前にエスケープ文字のバックスラッシュ「\」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。

次に例を示します。

```
"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
```

> **注意：** 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。

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）。SQL*Loader 制御ファイルを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。

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・システムで実行している Oracle Database のバージョンで、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。

エスケープ文字が使用できない場合

現在使用しているバージョンの Oracle Database では、移植不能文字列に対してエスケープ文字を使用できない場合があります。エスケープ文字が禁止されている場合、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（ただし、移植不能文字列以外の文字列では使用可能）。この場合、次のようなパス名を指定できます。

```
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。これは、二重引用符についても同様です。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。

XMLType 表の識別

Oracle Database 10*g* 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。この句を使用して、正しい SQL 文を構築するために XMLType 表を識別します。例 8-2 では、スキーマベースの XMLType 表にデータをロードするために、SQL*Loader の制御ファイルで XMLTYPE 句を使用する方法を示します。

例 8-2 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";
```

データを表 xdb_tab5 にロードするために使用する制御ファイルは、次のようになります。登録された XML スキーマ xdb_user.xsd を使用して、XMLType データがロードされます。XMLTYPE 句を使用して、この表を XMLType 表として識別します。ダイレクト・パスまたは従来型モードのいずれかを使用して、表にデータをロードできます。

```
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>
```

データ・ファイルの指定

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。

注意： コマンドラインからデータ・ファイルを指定する場合は、7-2 ページの「コマンドライン・パラメータ」で説明している DATA パラメータを使用します。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、デフォルトで制御ファイル名の拡張子を .dat にしたものが採用されます。

ロードす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。指定の詳細は、8-9 ページの「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。

注意： 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。

LOBFILE の詳細は、10-16 ページの「LOBFILE からの LOB データのロード」を参照してください。

SDF の詳細は、10-23 ページの「セカンダリ・データ・ファイル（SDF）」を参照してください。

INFILE の構文は次のとおりです。

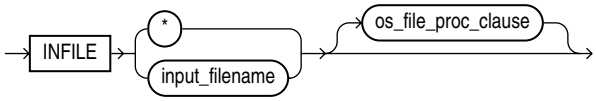


表 8-1 では、INFILE キーワードのパラメータについて説明します。

表 8-1 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。 ファイル名中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。詳細は、8-4 ページの「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。
*	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。制御ファイルおよびデータ・ファイルの両方にデータがある場合は、読み込むデータをまずアスタリスクで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。詳細は、8-9 ページの「データ・ファイル形式およびバッファリングの指定」を参照してください。

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'
  ```

> **注意：** 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。

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。次に、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ている制御ファイルの例を示します。

```
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。

BEGINDATA による制御ファイルのデータの識別

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。

最初のデータ・レコードの前に、BEGINDATA 文を指定します。構文は次のとおりです。

```
BEGINDATA
data
```

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、BEGINDATA の後に続けて記述しないでください。

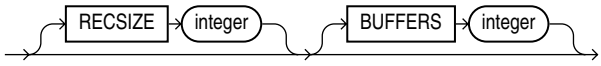
参照：

- INFILE の使用例については、8-7 ページの「データ・ファイルの指定」を参照してください。
- 「事例 1: 可変長データのロード」（事例へのアクセス方法については、6-12 ページの「SQL*Loader の事例」を参照）

データ・ファイル形式およびバッファリングの指定

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"
```

注意： 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。

参照： Windows システムでの *os_file_proc_clause* の使用方法については、『Oracle Database プラットフォーム・ガイド for Microsoft Windows』を参照してください。

不良ファイルの指定

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した場合は、次の規則に従って作成されます。

- 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。

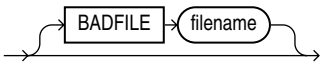
> **注意：** 一部のシステムでは、同じ名前のファイルがすでに存在する場合でも、新しいバージョンのファイルが作成される場合があります。

不良ファイルの名前を指定する場合は、BADFILE 句の後にファイル名を続けます。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。7-2 ページの「コマンドライン・パラメータ」で説明するとおり、BAD パラメータを使用して、コマンドラインから不良ファイルを指定することもできます。

コマンドラインから指定した不良ファイル名は、制御ファイル中の最初に記述される INFILE 句に対して指定されたものとみなされます。この場合、前述の句の中で不良ファイル名が指定されていても、コマンドラインから指定した不良ファイル名が優先されます。

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。したがって、データを修正した後そのまま再ロードすることができます。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。

不良ファイルの構文は次のとおりです。



BADFILE 句の後に、不良ファイルのファイル名を指定します。

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。ファイル名中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。

不良ファイル名指定の例

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、次のとおり入力します。

```
BADFILE sample
```

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、次のいずれかの行を入力します。

```
BADFILE bad0001.rej
BADFILE '/REJECT_DIR/bad0001.rej'
```

LOBFILE および SDF を使用した不良ファイルの処理方法

LOBFILE および SDF のデータは、拒否された行がある場合、不良ファイルには書き込まれません。LOB のロード中にエラーが発生した場合、その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（ゼロ）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。

拒否レコードの条件

レコードは、次の理由で拒否されます。

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. レコードが不適切にフォーマットされて、SQL*Loader がフィールドの境界を検索できない場合。
3. レコードが制約に違反するか、または一意の索引を非一意にしようとする場合。

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。

前述のリストの 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込みできません。

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、事例 4 を参照してください。（事例へのアクセス方法については、6-12 ページの「SQL*Loader の事例」を参照してください。）

廃棄ファイルの指定

ロード条件を満たさないレコードがある場合、SQL*Loader を実行すると、廃棄ファイルが生成されることがあります。このファイルに出力された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。）
- 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。

制御ファイル内から廃棄ファイルを作成するには、DISCARDFILE *filename*、DISCARDS、DISCARDMAX のいずれかを指定します。

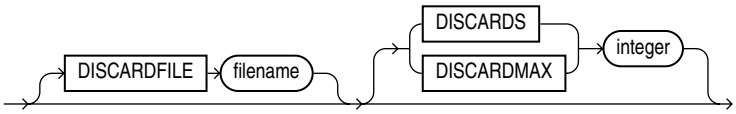
コマンドラインから廃棄ファイルを作成するには、DISCARD または DISCARDMAX のいずれかを指定します。

このように、名前を指定して廃棄ファイルを直接指定することも、また、廃棄数の最大値を指定することで間接的に廃棄ファイルの生成を指示することもできます。

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。

制御ファイルでの廃棄ファイルの指定

廃棄ファイルの名前を指定する場合は、DISCARDFILE 句の後にファイル名を続けます。



DISCARDFILE 句の後には、廃棄ファイルのファイル名を指定します。

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。ファイル名中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。

コマンドラインからの廃棄ファイルの指定

コマンドラインから廃棄ファイルを指定する方法の詳細は 7-5 ページの「DISCARD（ファイル名）」を参照してください。

コマンドラインで指定したファイル名で、制御ファイルに指定した廃棄ファイルが上書きされます。

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。

- 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、次のとおり入力します。

```
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、次のとおり入力します。

```
DISCARDFILE notappl.may
```

- 廃棄ファイル forget.me にフルパスを指定するには、次のとおり入力します。

```
DISCARDFILE  '/discard_dir/forget.me'
```

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: 書式化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。（事例へのアクセス方法については、6-12 ページの「SQL*Loader の事例」を参照してください。）

LOBFILE および SDF を使用した廃棄ファイルの処理方法

LOBFILE および SDF のデータは、廃棄された行がある場合も、廃棄ファイルには書き込まれません。

廃棄レコード数の上限付け

DISCARDS または DISCARDMAX キーワードに *integer* を指定して、各データ・ファイルに対して廃棄されるレコード数を制限できます。

廃棄レコード数がこの数に達すると、そのデータ・ファイルの処理は終了します。次のデータ・ファイルがあれば、その処理が開始されます。

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とファイル拡張子（またはファイル・タイプ）で廃棄ファイルが作成されます。

異なる文字コード体系の処理

SQL*Loader では、（キャラクタ・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。

参照：『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）・キャラクタ・セット

マルチバイト・キャラクタ・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。

Unicode キャラクタ・セット

SQL*Loader では、Unicode キャラクタ・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・キャラクタ・セットであり、単一のキャラクタ・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。

注意： 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、キャラクタ・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 ～ 0x007f は、シングルバイト ASCII コードの 0x00 ～ 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 ～ 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。

参照：

- 「事例 11: Unicode キャラクタ・セットのデータのロード」（事例へのアクセス方法については、6-12 ページの「SQL*Loader の事例」を参照）
- Unicode 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。

データベース・キャラクタ・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・キャラクタ・セットが使用されます。シングルバイト・キャラクタ・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キャラクタ・セットのみがデータベース・キャラクタ・セットとしてサポートされます。マルチバイト固定幅キャラクタ・セット（たとえば、AL16UTF16）は、データベース・キャラクタ・セットとしてサポートされません。

SQL の NCHAR データ型（NCHAR、NVARCHAR および NCLOB）に格納されたデータ用のデータベースでは、代替キャラクタ・セットを使用できます。代替キャラクタ・セットは、データベース各国語キャラクタ・セットと呼ばれます。Unicode キャラクタ・セットのみが、データベース各国語キャラクタ・セットとしてサポートされます。

データ・ファイル・キャラクタ・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キャラクタ・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・キャラクタ・セットは、データベース・キャラクタ・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キャラクタ・セットがサポートされます（データベース・キャラクタ・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キャラクタ・セット（AL16UYF16、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 エンコーディングのみが、データベース・キャラクタ・セットとしてではなく、データベース・各国語キャラクタ・セットとしてのみサポートされます。

データ・ファイルのキャラクタ・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。

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キャラクタ・セットは、NLS_LANG パラメータで定義されたセッション・キャラクタ・セットです。入力データ・ファイルで使用されるキャラクタ・セットは、CHARACTERSET パラメータで指定できます。

SQL*Loader には、データ・ファイル・キャラクタ・セット、データベース・キャラクタ・セットおよびデータベース各国語キャラクタ・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・キャラクタ・セットまたはデータベース各国語キャラクタ・セットに自動的に変換する機能もあります。

データのキャラクタ・セット変換が必要な場合、ターゲットのキャラクタ・セットは、ソースのデータ・ファイル・キャラクタ・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・キャラクタ・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・キャラクタ・セットの変換が必要な場合、変換される文字のソース・キャラクタ・セットより多くのバイトがターゲット・キャラクタ・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。

参照：

- データベースの文字列セマンティクスの詳細は、『Oracle Database 概要』を参照してください。
- 「文字長セマンティクス」（8-17 ページ）
- 『Oracle Database グローバリゼーション・サポート・ガイド』

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

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・キャラクタ・セットとは異なるキャラクタ・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で、1 行もロードされないまま異常終了する可能性があります。
- 主キー・ベース 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、クライアントのキャラクタ・セットをデータベース・キャラクタ・セットに設定します。

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・ファイルのキャラクタ・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キャラクタ・セットは、NLS_LANG パラメータで定義されたセッション・キャラクタ・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・ファイルのキャラクタ・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、キャラクタ・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キャラクタ・セットの名前である必要があります。

Unicode の UTF-16 エンコーディングに関しては、AL16UTF16 ではなく UTF16 という名前を使用します。AL16UTF16 は、UTF-16 でエンコーディングされたデータに対して Oracle がサポートするキャラクタ・セット名であり、ビッグ・エンディアン・バイト順序の UTF-16 データのみに使用されます。ただし、データ・ファイルを作成するシステムのバイト順序を使用してデータが設定できるため、データ・ファイルのデータはビッグ・エンディアンまたはリトル・エンディアンのいずれにもなります。したがって、異なるキャラクタ・セット名（UTF16）が使用されます。キャラクタ・セット名 AL16UTF16 もサポートされます。ただし、リトル・エンディアン・バイト順序のデータ・ファイルに対して AL16UTF16 を指定すると、SQL*Loader では、警告メッセージが発行され、データ・ファイルがビッグ・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・ファイルは同じキャラクタ・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 パラメータで設定されたキャラクタ・セットは、制御ファイルのデータ（INFILE で指定されている）には適用されません。NLS_LANG パラメータで指定されているセッション・キャラクタ・セット以外のキャラクタ・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。

参照：

- 「バイト順序」（9-29 ページ）
- サポートされているキャラクタ・セットの名前の詳細は、『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・キャラクタ・セット」（8-16 ページ）
- 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 キャラクタ・セットのデータのロード」を参照してください。（事例へのアクセス方法については、6-12 ページの「SQL*Loader の事例」を参照してください。）

制御ファイル・キャラクタ・セット

SQL*Loader 制御ファイル自体のキャラクタ・セットは、NLS_LANG パラメータで指定されているセッション・キャラクタ・セットであることが前提です。制御ファイル・キャラクタ・セットがデータ・ファイル・キャラクタ・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・キャラクタ・セットからデータ・ファイル・キャラクタ・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'"
```

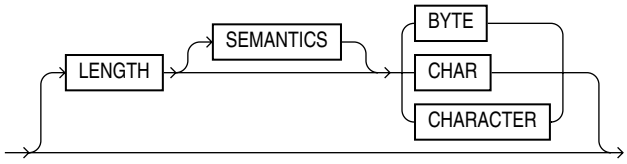
BEGINDATA 文の後に指定するデータも、NLS_LANG パラメータで指定されたセッション・キャラクタ・セットであることが前提です。

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・キャラクタ・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・キャラクタ・セットが使用されている場合、異なることもあります。

キャラクタ・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。

文字長セマンティクス

バイト長セマンティクスは、UTF-16 キャラクタ・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、LOBFILE データ・ファイルおよび SDF に関してのみではなく、プライマリ・データ・ファイルに関する構文の指定にも適用されます。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。

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。

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。

参照：「SKIP（スキップされるレコード）」（7-10 ページ）

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。

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。

- 領域エラーによって中断されたロード
- 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
- 致命的エラーによって中断されたロード
- [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

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込み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。

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。

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。

[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。SKIP パラメータの値は、すべての表で同じになります。

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。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、11-10 ページの「使用禁止状態（Index Unusable）のままの索引」を参照してください。

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。

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。

参照： 「SKIP（スキップされるレコード）」（7-10 ページ）

物理レコードからの論理レコードの作成

Oracle9*i* では、64KB を超えるユーザー定義のレコード・サイズがサポートされています（7-8 ページの「READSIZE（読取りバッファ・サイズ）」参照）。これによって、論理レコードを複数の物理レコードに細分化する必要がなくなります。ただし、細分化する必要が完全になくなるわけではありません。複数の物理レコードを結合して 1 つの論理レコードに戻すには、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。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 に指定された整数値によって、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。

参照：

- 「COLUMNARRAYROWS」（7-3 ページ）
- 「列配列の行数およびストリーム・バッファ・サイズの指定」（11-16 ページ）

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 バイト目にシャープ記号（#）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。

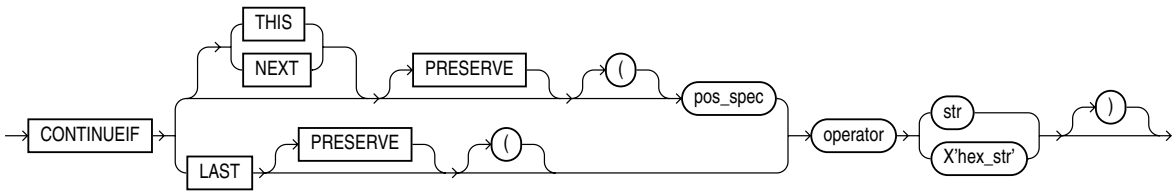


表 8-2 では、CONTINUEIF 句のパラメータについて説明します。

表 8-2 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（X' hex_string'）または文字列（char_string）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 パラメータを使用しない場合、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。

CONTIUEIF 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 8-3 ～例 8-6 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 8-3 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....
%%ddddddddddd..
%%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....
ddddddddddd..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 8-4 PRESERVE パラメータを使用した CONTINUEIF THIS

例 8-3 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....
%%ddddddddddd..%%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 8-5 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....
..ddddddddddd..
%%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 8-3 と同じ結果）。

```
aaaaaaaa....bbbbbbbb....cccccccc....
ddddddddddd..eeeeeeeeee..ffffffffff..
```

例 8-6 PRESERVE パラメータを使用した CONTINUEIF NEXT

例 8-5 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....
..ddddddddddd..%%eeeeeeeeee..%%ffffffffff..
```

参照： 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。（事例へのアクセス方法については、6-12 ページの「SQL*Loader の事例」を参照してください。）

表への論理レコードのロード

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

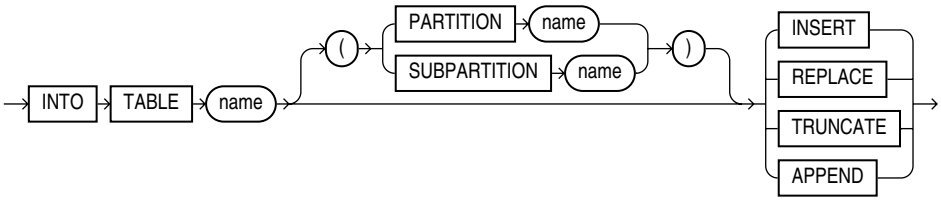
LOAD DATA 文の INTO TABLE 句を使用して、表、フィールドおよびデータ型を識別できます。この句で、データ・ファイル中のレコードとデータベース中の表の関係を定義します。フィールドやデータ型の指定については後述します。

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。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。

構文は次のようになります。



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）。

> **注意：** 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。

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。

空の表へのデータのロード

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 これは、SQL*Loader のデフォルトの方法です。ロードする前に表を空にする必要があります。表に行が存在する場合はエラーが返され、ロードが終了します。「事例 1: 可変長データのロード」に例があります。（事例へのアクセス方法については、6-12 ページの「SQL*Loader の事例」を参照してください。）

空でない表へのデータのロード

ロード先の表にすでにデータが存在する場合は、3 つのオプションがあります。

- APPEND
- REPLACE
- TRUNCATE

> **注意：** 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 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。(事例へのアクセス方法については、6-12 ページの「SQL*Loader の事例」を参照してください。)

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。(事例へのアクセス方法については、6-12 ページの「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。削除のカスケード化の詳細は、『Oracle Database 概要』のデータ整合性に関する説明を参照してください。

注意: 表のエクステントを再利用しない SQL の TRUNCATE 文とは対照的に、SQL*Loader の REPLACE では、表のエクステントを再利用します。

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。

TRUNCATE TRUNCATE オプションを使用すると、SQL の TRUNCATE TABLE 文が実行されます。これにより、短時間で効率的に表またはクラスタから行を削除して、最大限の処理パフォーマンスを実現します。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。

参照: この項で説明されている SQL 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)。

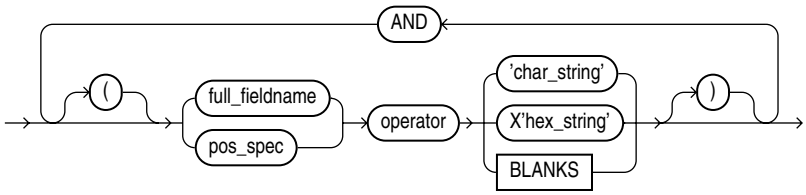
OPTIONS パラメータの構文は次のとおりです。

参照: 「パラレル・ダイレクト・パス・ロードのパラメータ」(11-25 ページ)

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。次に例を示します。

```
WHEN (deptno = '10') AND (job = 'SALES')
```

参照 :

- SQL*Loader による NULLIF、DEFAULTIF および WHEN 句の評価方法については、9-25 ページの「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」（事例へのアクセス方法については、6-12 ページの「SQL*Loader の事例」）を参照してください。

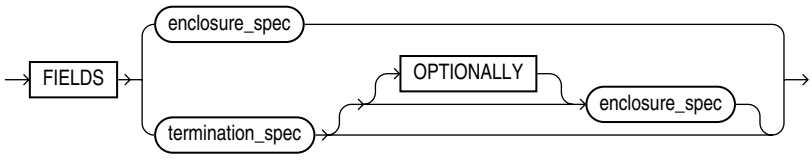
LOBFILE および SDF での WHEN 句の使用

LOBFILE または SDF のレコードが廃棄されると、SQL*Loader によってその LOBFILE または SDF 内の対応するデータがスキップされます。

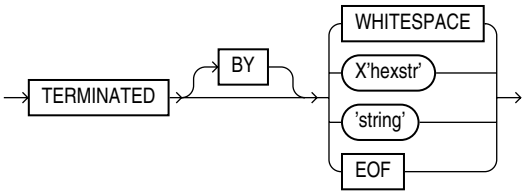
デフォルトのデータ・デリミタ（区切り記号）の指定

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デリミタ（区切り記号）を指定することができます。fields_spec、termination_spec および enclosure_spec 句の構文は、次のとおりです。

fields_spec



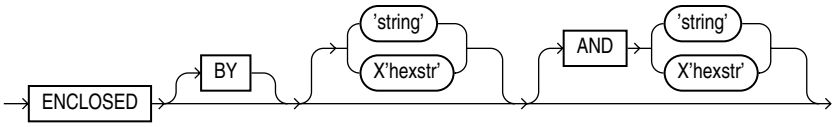
termination_spec



注意： 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。

---

enclosure_spec



注意： 囲み文字列には、1 つ以上の文字が含まれます。

特定の列に対して別のデリミタを使用する場合は、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。（事例へのアクセス方法については、6-12 ページの「SQL*Loader の事例」を参照してください。）

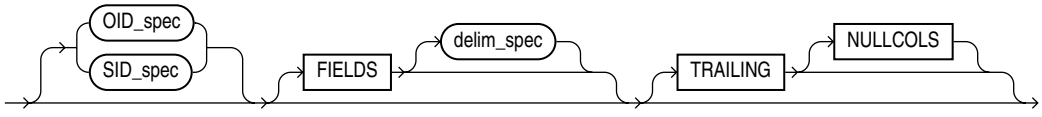
参照：

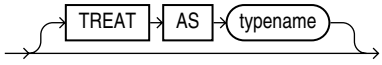
- 構文の詳細は、9-18 ページの「デリミタの指定」を参照してください。
- 「LOBFILE からの LOB データのロード」（10-16 ページ）

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

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。

> **参照：** TRAILING NULLCOLS の使用例は、「事例 7: 書式化されたレポートからのデータの抽出」（事例へのアクセス方法については、6-12 ページの「SQL*Loader の事例」）を参照してください。

索引オプション

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

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。

> **参照：** 「SORTED INDEXES 句」（11-14 ページ）

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、SINGLEROW を指定すると、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）。

複数の INTO TABLE 句を使用するメリット

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。

注意： 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、このようなスキャン時の動作を利用して INTO TABLE 句を指定する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。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。

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。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。2 番目の 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、8-31 ページの「異なる入力レコード形式の区別」および 8-33 ページの「複数表へのデータのロード」を参照してください。

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

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 POSITION 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。

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。

> **注意：** 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)
```

> **参照：** オブジェクト型のロードの詳細は、10-2 ページの「列オブジェクトのロード」を参照してください。

複数表へのデータのロード

複数の INTO TABLE パラメータで POSITION 句を指定すると、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。（事例へのアクセス方法については、6-12 ページの「SQL*Loader の事例」を参照してください。）

要約

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。

バインド配列および従来型パス・ロード

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。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 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。ダイレクト・パス・ロードでは、Oracle SQL インタフェースではなくダイレクト・パス API が使用されるため、バインド配列サイズを決定する必要はありません。

> **参照：** 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。

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。

BINDSIZE パラメータおよび ROWS パラメータについては、7-2 ページの「コマンドライン・パラメータ」を参照してください。

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。

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。

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（7-3 ページの「BINDSIZE（最大サイズ）」を参照）または制御ファイル中の OPTIONS 句（8-3 ページの「OPTIONS 句」を参照）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掛け合されます。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（7-9 ページの「ROWS（1 回にコミットする行数）」を参照）で指定されていても、制御ファイル中の OPTIONS 句（8-3 ページの「OPTIONS 句」を参照）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。

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、9-6 ページの「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
- TIME WITH TIME ZONE
- VARCHAR
- VARCHARC
- VARGRAPHIC
- VARRAW
- VARRAWC

これらのデータ型の最大長の詳細は、9-6 ページの「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。

注意： 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

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。

> **注意：** 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。

フィールド・バッファ・サイズの計算

表 8-3 ～表 8-6 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、「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、9-6 ページの「SQL*Loader のデータ型」を参照してください。

表 8-3 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 言語の INT データ型に相当するサイズ
INTEGER(*N*)	*N* バイト
SMALLINT	C 言語の SHORT INT データ型に相当するサイズ
FLOAT	C 言語の FLOAT データ型に相当するサイズ
DOUBLE	C 言語の DOUBLE データ型に相当するサイズ
BYTEINT	C 言語の 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 8-4 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
(PACKED) 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 8-5 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	DATATYPE 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 8-6 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	4KB	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

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。このデフォルト・サイズによっては、メモリーを大量に使用することがあります。特に、デフォルト・サイズにバインド配列の行数を掛け合せると、使用するメモリーは非常に大きくなります。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。次の例について考えてみます。

```
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」がデータ・ファイル・キャラクタ・セット中の文字のバイト単位での最大値です)。

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。

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。

注意： 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。

9

SQL*Loader フィールド・リスト・リファレンス

この章では、SQL*Loader 制御ファイルのフィールド・リストについて説明します。この章の内容は、次のとおりです。

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、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。

例 9-1 に、第 8 章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 9-1 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、9-44 ページの「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、9-2 ページの「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、フィールド用のデータ型です。9-6 ページの「データ・フィールドのデータ型の指定」および 9-15 ページの「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、9-25 ページの「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 を使用して、フィールドを空白と比較しています。詳細は、9-24 ページの「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、9-19 ページの「TERMINATED フィールド」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、9-19 ページの「ENCLOSED フィールド」を参照してください。

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（columnspec）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、絶対位置で指定することも、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。

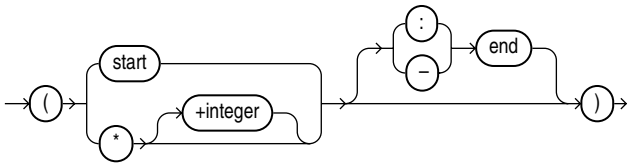


表 9-1 では、位置指定句のパラメータについて説明します。

表 9-1 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。SQL*Loader の拡張 SQL 文字列機能を使用して、書式化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。

このようなエラーは、データにタブが含まれているときに発生します。紙に出力した場合、各タブの幅は数列分に拡がります。ただし、データ・ファイルでは、それぞれのタブは 1 文字のままです。そのため、SQL*Loader は、データ・ファイルの読取り時に、誤った POSITION 指定を参照することになります。

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。

参照： 「デリミタの指定」（9-18 ページ）

複数表のロードでの POSITION の使用

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、8-30 ページの「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 をリセットする必要があります。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。

POSITION を使用した例

```
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 ～ 20 を占める文字データで、その次の列 empno は位置 22 ～ 26 を占める数値型データです。列 allow は empno の終了位置直後の位置（27）から +2 の位置、つまり位置 29 から始まり、スラッシュが検出されるまで継続するデータとなります。

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。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（columnspec）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、9-42 ページの「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（columnspec）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、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。次に、例を示します。

```
LOAD DATA
INFILE *
INTO TABLE BFILE1O_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 が指定および適用される場合、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。

注意： 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。

FILLER フィールド指定の例を次に示します。

```
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)
```

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。

各フィールドに対して指定できるデータ型は 1 つのみです。データ型を指定しない場合は、CHAR が想定されます。

SQL*Loader データ型から Oracle データ型への変換処理および SQL*Loader の各データ型の詳細は、9-6 ページの「SQL*Loader のデータ型」を参照してください。

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 のデータ型

SQL*Loader 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分類されます。さらに、これら 2 つのグループは、LENGTH-VALUE データ型および VALUE データ型に分類されます。

移植可能なデータ型および移植不能なデータ型は、データ型のプラットフォーム依存性によって分類されます。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マ（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。バイト順序スキーマ、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。

移植不能なデータ型

移植不能なデータ型は、VALUE データ型および LENGTH-VALUE データ型に分類されます。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
- (PACKED) 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の構文の詳細は、A-7 ページの「datatype_spec」の構文図を参照してください。

INTEGER(*n*)

データは、フルワード 2 進整数（*n* は、1、2、4 または 8 のオプションで提供された長さ）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、特定のプラットフォームでの C 言語の LONG INT のサイズに基づいて決まります。

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、移植不能です。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 を使用して、正しい結果で INTEGER データにアクセスできます。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 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。

参照： 「異なるプラットフォーム間でのデータのロード」（9-28 ページ）

SMALLINT

データはハーフワードの 2 進整数で表現されます。このフィールド長には、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。デフォルトでは、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 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。詳細は、9-29 ページの「バイト順序」を参照してください。

注意： このデータ型のフィールド長は、C 言語の SHORT INT データ型と同じです。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。

FLOAT

データは単精度浮動小数点 2 進数で表現されます。POSITION 句で *end* を指定すると *end* は無視されます。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。（C 言語のデータ型は FLOAT に相当する長さです。）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。

FLOAT は、FLOAT の表現に互換性があり、その長さが同じシステム間のみで、正しい結果の出るロードができます。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。詳細は、9-29 ページの「バイト順序」を参照してください。

DOUBLE

データは倍精度浮動小数点 2 進数で表現されます。POSITION 句で *end* を指定すると *end* は無視されます。このフィールド長には、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。（C 言語のデータ型 DOUBLE または LONG FLOAT に相当する長さです。）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。

DOUBLE は、DOUBLE の表現に互換性があり、その長さが同じシステム間のみで、正しい結果でロードできます。バイト順序が 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。詳細は、9-29 ページの「バイト順序」を参照してください。

BYTEINT

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。（C 言語のデータ型 UNSIGNED CHAR に相当する長さです。）

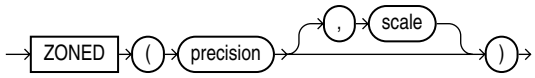
このデータ型の構文の例は次のとおりです。

```
(column1 position(1) BYTEINT,
column2 BYTEINT,
...
)
```

ZONED

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（COBOL の SIGN TRAILING フィールドに相当）で表され、最終バイトに符号が入ります。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。

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。次の例では、位置 32 から始まる 8 桁の整数を表します。

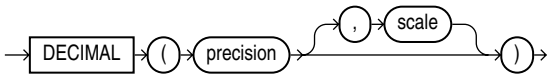
```
sal    POSITION(32)    ZONED(8),
```

Oracle Database では、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、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 では「ESA/390 操作の原理」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・ファイルのキャラクタ・セット・エンコーディングによって異なります。詳細は、8-15 ページの「CHARACTERSET パラメータ」を参照してください。

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。)

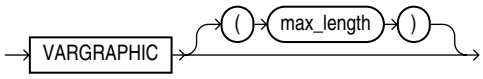
VARGRAPHIC

このデータは、可変長のダブルバイト・キャラクタ・セット（DBCS）です。length サブフィールドおよび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・キャラクタ・セットは、Oracle Database ではサポートされていないため、SQL*Loader を使用してシングルバイトとして読み込み、RAW データとしてロードします。RAW データ型と同様、VARGRAPHIC フィールドは変更されずに指定の列に格納されます。

> **注意：** 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 言語の SHORT INT 型に相当する長さ）が取られます。詳細は、9-7 ページの「SMALLINT」を参照してください。

VARGRAPHIC は、SHORT INT の長さが同じバイト数のシステム間でのみ、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。詳細は、9-29 ページの「バイト順序」を参照してください。

VARGRAPHIC データ型の構文は次のとおりです。



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（maximum_length）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 文の前に位置指定が指定されている場合、（pos_spec を使用して）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。

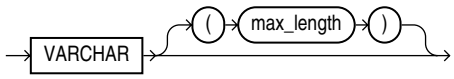
VARCHAR

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。詳細は、8-17 ページの「文字長セマンティクス」を参照してください。

VARCHAR フィールドは、SHORT データ・フィールド 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、または VARCHAR フィールドに UTF-16 キャラクタ・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、UTF-16 キャラクタ・セットに対してのみ問題となります。詳細は、9-29 ページの「バイト順序」を参照してください。

> **注意：** 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 言語の SHORT INT 型に相当する長さ）が取られます。詳細は、9-7 ページの「SMALLINT」を参照してください。

VARCHAR データ型の構文は次のとおりです。



制御ファイルに指定する最大長（maximum_length）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、キャラクタ・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。詳細は、8-17 ページの「文字長セマンティクス」を参照してください。

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。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。

VARRAW

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。

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。詳細は、9-29 ページの「バイト順序」を参照してください。

LONG VARRAW

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 サブフィールドを持つ 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。

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。詳細は、9-29 ページの「バイト順序」を参照してください。

移植可能なデータ型

移植可能なデータ型は、VALUE データ型および LENGTH-VALUE データ型に分類されます。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 および ZONED）
- RAW

移植可能な LENGTH-VALUE データ型は次のとおりです。

- VARCHARC
- VARRAWC

これらのデータ型の構文の詳細は、A-7 ページの「datatype_spec」の構文図を参照してください。

文字データ型には、CHAR 型、DATE 型および数値型 EXTERNAL 型があります。これらのフィールドにはデリミタを使用できます。また、制御ファイルにフィールド長（または最大長）を指定することができます。

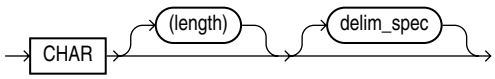
CHAR

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）。データ長については、次のことにも注意してください。

- 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。



参照： 「デリミタの指定」(9-18 ページ)

日時データ型および期間データ型

日時および期間の両方ともフィールドで構成されています。これらのフィールドの値によってデータ型の値が決定されます。

日時データ型には次のものがあります。

- DATE
- TIME
- TIME WITH TIME ZONE
- TIMESTAMP
- TIMESTAMP WITH TIME ZONE
- TIMESTAMP WITH LOCAL TIME ZONE

日時データ型の値は、Datetimes と呼ばれる場合があります。次に説明する日時データ型（DATE を除く）では、オプションで fractional_second_precision の値を指定できます。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は、値を 0 ～ 9 に指定できます。デフォルトは 6 です。

期間データ型には次のものがあります。

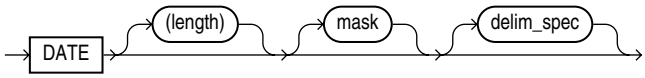
- INTERVAL YEAR TO MONTH
- INTERVAL DAY TO SECOND

期間データ型の値は、Intervals と呼ばれる場合があります。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。指定できる値は 0 ～ 9 で、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は、値を 0 ～ 9 に指定できます。デフォルトは 6 です。

参照： fractional_second_precision、year_precision および day_precision を使用する、日時データ型と期間データ型の指定の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

DATE 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。DATE フィールドの構文は次のとおりです。



次に例を示します。

```
LOAD DATA
INTO TABLE dates (col_a POSITION (1:15) DATE "DD-Mon-YYYY")
BEGINDATA
1-Jan-2002
1-Apr-2002 28-Feb-2002
```

デリミタがない場合、空白は無視され、日付は左から右に構文解析されます（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）。

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。詳細は、8-17 ページの「文字長セマンティクス」を参照してください。

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, 1991" 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。

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、常に長さを指定してください。

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。これらはいずれも、マスクから求められる長さより優先されます。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。

DATE 型のフィールドでは、デリミタも使用できます。詳細は、9-18 ページの「デリミタの指定」を参照してください。

TIME TIME データ型には、時、分、秒の値が格納されます。次のように指定します。

```
TIME [(fractional_second_precision)]
```

TIME WITH TIME ZONE TIME WITH TIME ZONE データ型は、タイムゾーンの置換えを含む TIME データ型の変形です。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。次のように指定します。

```
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。

TIMESTAMP TIMESTAMP データ型は、DATE データ型の拡張です。DATE データ型の年、月、日に加えて、TIME データ型の時、分、秒の値を格納します。次のように指定します。

```
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。

TIMESTAMP WITH TIME ZONE TIMESTAMP WITH TIME ZONE データ型は、タイムゾーンの置換えを含む TIMESTAMP データ型の変形です。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。次のように指定します。

```
TIMESTAMP [(fractional_second_precision)] WITH TIME ZONE
```

TIMESTAMP WITH LOCAL TIME ZONE TIMESTAMP WITH LOCAL TIME ZONE データ型は、タイムゾーンの置換えを含む TIMESTAMP データ型の変形です。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。次のように指定します。

```
TIMESTAMP [(fractional_second_precision)] WITH LOCAL TIME ZONE
```

INTERVAL YEAR TO MONTH INTERVAL YEAR TO MONTH データ型は、YEAR および MONTH 日時フィールドを使用して一定期間を格納します。次のように指定します。

```
INTERVAL YEAR [(year_precision)] TO MONTH
```

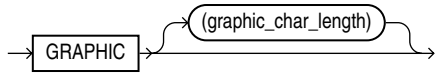
INTERVAL DAY TO SECOND INTERVAL DAY TO SECOND データ型は、DAY および SECOND 日時フィールドを使用して一定期間を格納します。次のように指定します。

```
INTERVAL DAY [(day_precision)] TO SECOND [(fractional_second_precision)]
```

GRAPHIC

このデータは、ダブルバイト・キャラクタ・セット（DBCS）形式の文字列データです。ダブルバイト・キャラクタ・セットは Oracle Database ではサポートされていないため、SQL*Loader を使用してシングルバイトとして読み込みます。RAW データ型と同様、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 データ型の構文は次のとおりです。



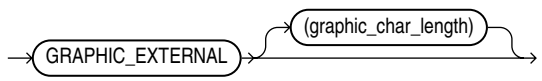
GRAPHIC 型および GRAPHIC EXTERNAL 型では、POSITION*(start:end)* を指定すると、論理レコードにおけるフィールドの正確な位置が決まります。

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。

GRAPHIC EXTERNAL

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。

GRAPHIC EXTERNAL データ型の構文は次のとおりです。



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* の値は、DBCS のデータ長を指定します（9-14 ページの「GRAPHIC」を参照）。

たとえば、[ ] をシフトイン / シフトアウト文字とし、# を任意のダブルバイト文字とします。

[####] を表現する場合は、POSITION(1:4) GRAPHIC または POSITION(1) GRAPHIC(2) と指定します。

[####] を表現する場合は、POSITION(1:6) GRAPHIC EXTERNAL または POSITION(1) GRAPHIC EXTERNAL(2) と指定します。

数値型 EXTERNAL

数値型 EXTERNAL データ型は、数値データ型（INTEGER、FLOAT、DECIMAL および ZONED）に EXTERNAL、オプションのデータ長およびデリミタ指定を指定したものです。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。詳細は、8-17 ページの「文字長セマンティクス」を参照してください。

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、9-11 ページの「CHAR」を参照してください。

数値型 EXTERNAL データ型の構文は、A-7 ページの「datatype_spec」を参照してください。

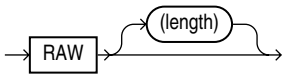
> **注意：** このデータは、バイナリ表現ではなく、文字形式の数字になります。したがって、これらのデータ型の処理方法は、DEFAULTIF を使用する場合を除き、CHAR と同じです。デフォルトを NULL にする場合は CHAR を使用します。デフォルトを 0（ゼロ）にする場合は EXTERNAL を使用します。詳細は、9-25 ページの「WHEN、NULLIF および DEFAULTIF 句の使用」を参照してください。

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。

RAW

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 によるデータ変換は行われません。CHAR 列にロードした場合は、Oracle Database によって、16 進数にデータ変換されます。DATE 型や数値型の列にはロードできません。

RAW データ型の構文は次のとおりです。



ここで、length には制御ファイルに指定されたバイト数を指定します。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・リソースの許容範囲内であれば自由に指定できます。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。

VARCHARC

VARCHARC データ型は、文字の length サブフィールドおよびその後に続く文字列 VALUE サブフィールドで構成されます。

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。

次に例を示します。

- 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。

詳細は、8-17 ページの「文字長セマンティクス」を参照してください。

VARRAWC

VARRAWC データ型は、RAW 文字列の VALUE サブフィールドで構成されています。

次に例を示します。

- 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。
- 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 VARRAWC になります。

システム固有のデータ型フィールド長の競合

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 Datatype
----------------------- --------- ----- ---- ---- ---------
COLUMN1                      1:6     4             INTEGER
```

LENGTH-VALUE データ型のフィールド長

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
```

データ型の変換

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・ファイルのキャラクタ・セットとデータベースのキャラクタ・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。

注意： 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。したがって、SQL から Oracle エラー ORA-01461 が返され、SQL*Loader はその行を拒否してロードを続行します。

入力ファイル中のデータ型は、Oracle 表における列のデータ型に一致している必要はありません。データ型が一致していない場合、Oracle Database サーバーで自動的に変換が行われます。ただし、変換が正常に実行されたか、エラーは発生していないかについては実行後に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 型のデータベース列にロードしたとします。この場合、その文字フィールドの値が有効な数値となっているかどうかを必ず確認してください。

注意： 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。

日時データ型および期間データ型のデータ型変換

表 9-2 に、Oracle Database データ型と 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ている変換およびサポートされていない変換を示します。

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。ただし、前述のとおり、SQL*Loader には、NUMBER、CHAR、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。ただし、Oracle Database で変換可能なデータは、これらのデータ型やその他のデータ型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。行全体を見ると、Oracle Database データ型の CHAR、VARCHAR2、DATE、TIMESTAMP および TIMESTAMP WITH TIME ZONE データ型に対してデータ型変換がサポートされていることを確認できます。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 データ型に対しては変換がサポートされていません。

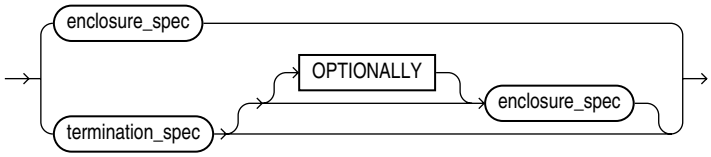
表 9-2 日時データ型および期間データ型のデータ型変換

SQL*Loader のデータ型	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）
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）

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、特定の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。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。データ型指定の後にデリミタを指定して、フィールドをどのように区切るかを指定します。

次の構文に示すとおり、デリミタ付きデータは、終了デリミタまたは囲みデリミタで区切られます。



デリミタの指定には、TERMINATED BY 句または ENCLOSED BY 句（あるいはその両方）も使用できます。両方とも指定する場合は TERMINATED BY 句を先に指定してください。enclosure_spec および termination_spec の構文については、9-19 ページの「終了および囲みの指定に関する構文」を参照してください。

TERMINATED フィールド

TERMINATED フィールドには、フィールドの開始位置から最初のデリミタ文字までのデータが読み込まれます（デリミタ文字自体は読み込まれません）。終了デリミタが最初の列位置にあれば、そのフィールドは NULL となります。

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、次に空白以外の文字が現れるまで連続する空白文字は読み込まれません。したがって、フィールド値の間に入る空白は、いくつあってもかまいません。

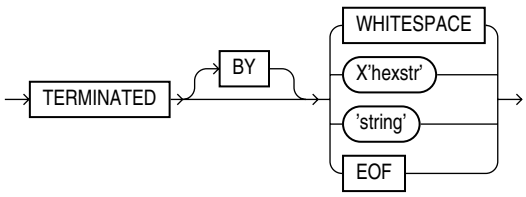
ENCLOSED フィールド

ENCLOSED フィールドの読取りでは、空白以外の文字が検出されるまで、空白文字はスキップされます。このとき、現れた空白以外の文字がデリミタの場合は、次のデリミタまでのデータが読み込まれます。現れた空白以外の文字がデリミタでない場合は、エラーとなります。

デリミタ文字が 2 つ続けて現れた場合は、1 つのデリミタ文字のみがデータ値の一部として扱われます。たとえば 'DON''T' は、DON'T として格納されます。ただし、フィールドに 2 つのデリミタのみ含まれている場合は NULL 値となります。

終了および囲みの指定に関する構文

次の図に、termination_spec および enclosure_spec の構文を示します。



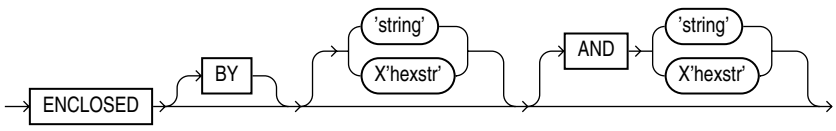


表 9-3 では、デリミタの指定に使用する、終了および囲みの指定に関する構文について説明します。

表 9-3 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'	デリミタには 1 文字を指定しますが、ここでは文字コード体系における *X'hexstr'* によって指定される値で、文字を指定します。たとえば、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。

各指定方法によるデリミタの指定例およびそれぞれの場合の実際の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)
```

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。

デリミタ付きデータの最大長

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)
```

デリミタを使用した後続の空白のロード

後続の空白は、PRESERVE BLANKS を指定しないかぎり、デリミタなしのデータ型ではロードされません。たとえば、データ・フィールド長が 9 文字で、DANIEL*bbb* という値のデータがあるとします。ここでの *bbb* は 3 つの空白を示します。このとき、CHAR(9) と宣言されていると、Oracle Database には「DANIEL」がロードされます。

この例で後続の空白も必要な場合は、CHAR(9) TERMINATED BY ':' と宣言し、さらにデータ・ファイルにコロンを追加してフィールドを DANIEL*bbb*: とします。このフィールドは、後続の空白とともに、"DANIEL " としてロードされます。TERMINATED BY 句を使用しないで PRESERVE BLANKS を指定し、同じ結果を得ることもできます。

参照 :

- 「空白の切捨て」(9-32 ページ)
- 「空白の切捨てに対する PRESERVE BLANKS オプションの影響」(9-37 ページ)

文字データ型フィールド長の競合

CHAR 型、DATE 型、数値型 EXTERNAL 型の文字データ型の場合は、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

前述のデータ型のフィールドに対して開始位置と終了位置を指定すると、そのフィールド長は指定された開始 / 終了位置から求められます。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、次のように、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。

```
POSITION(1:10) CHAR(15)
```

この場合、このフィールド長は 15 になります。

デリミタ付きフィールド

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が計算できない場合は、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` が指定されている場合は、残りのフィールドには NULL 値が設定されます。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。

日付フィールド・マスク

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, 1991」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, 1992」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。

> **参照：** `DATE` フィールドの詳細は、9-12 ページの「日時データ型および期間データ型」を参照してください。

フィールド条件の指定

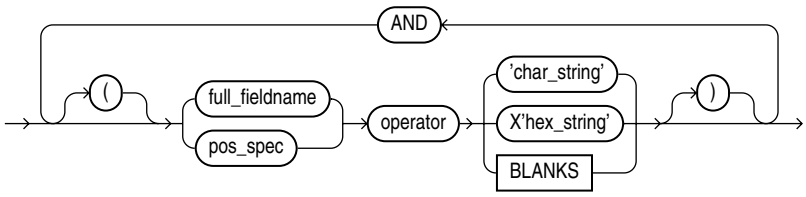
フィールド条件とは、論理レコード中のフィールドに関して、それが真か偽かを評価する条件を記述したものです。フィールド条件は、`WHEN` 句、`NULLIF` 句および `DEFAULTIF` 句で使用します。

注意： 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、常に、その句は FALSE と評価されます。この機能を例 9-5 に示します。

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。

> **注意：** フィールド条件は、SDF のフィールドに基づくことができません。

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。

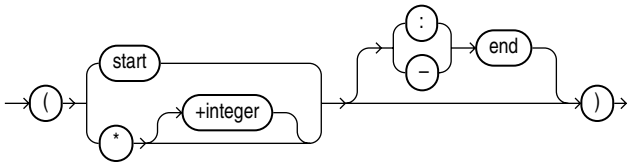


表 9-4 では、フィールド条件句のパラメータについて説明します。位置指定パラメータの詳細は、表 9-1 を参照してください。

表 9-4 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および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。

表 9-4 フィールド条件句のパラメータ（続き）

パラメータ	説明
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・キャラクタ・セットを使用する場合に必要となります。

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。

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。BLANKS 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 2 つ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・キャラクタ・セットには複数の空白が存在することもあります。このようなキャラクタ・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・キャラクタ・セットの詳細は、8-13 ページの「マルチバイト（アジア系言語）・キャラクタ・セット」を参照してください。

フィールドと文字列の比較

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、次のように空白が埋め込まれます。

```
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。

WHEN、NULLIF および DEFAULTIF 句の使用

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。空白およびタブの切捨ての詳細は、9-32 ページの「空白の切捨て」を参照してください。
- 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 の処理手順によっても結果に影響します。SQL*Loader は次の手順を順番に実行します。ただし、すべての手順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、設定作業の残りの手順は無視されます。たとえば、手順 5 でフィールドが設定された場合、SQL*Loader は手順 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 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
   - （PACKED）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、手順 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。

注意： これらの手順に示したとおり、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。手順 1 では、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には、この機能を利用するためにデータを変更できるかどうかを検討してください。手順 1 での NULL 値の検出は、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、またはすべての空白が含まれ、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）。

WHEN、NULLIF および DEFAULTIF 句の使用

例 9-2 から例 9-5 に、異なる条件で WHEN 句、NULLIF 句および DEFAULTIF 句を使用した場合の結果を示します。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 9-2 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、次のものが含まれます。

```
aname...
```

例 9-2 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 9-3 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、次のものが含まれます。

```
aname...
```

例 9-3 では、行の col1 は再度 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 9-4 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、次のものが含まれます。

```
.....123
```

例 9-4 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）の 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 と比較されます。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 9-5 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、次のものが含まれます。

```
.....123
```

例 9-5 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）の 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、SQL*Loader では、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。

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。このようにして作成したデータ・ファイルは、システム固有のデータ型を使用して作成したデータ・ファイルよりサイズが大きくなる場合があります。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、9-29 ページの「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。

バイト順序

> **注意：** この項の説明は、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上で入力データを作成する場合のみに適用されます。それ以外の場合は、次の項に進んでください。

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、Sun SPARC Solaris システム上では、SQL*Loader でビッグ・エンディアン・バイト順序が使用されます。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 キャラクタ・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 でサポートされるすべてのキャラクタ・セットでは、UTF16 を除いて、一度に 1 バイト書き込まれます。そのため、UTF8 などのマルチバイト・キャラクタ・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 キャラクタ・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キャラクタ・セットのデータは移植可能です。

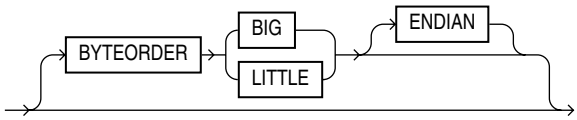
データ・ファイルのバイト順序が問題となるのは、バイト順序依存データを含むデータ・ファイルが、SQL*Loader が実行されているシステムとは異なるバイト順序のシステムで作成される場合のみです。SQL*Loader でデータのバイト順序を認識できる場合、必要に応じてバイトを入れ替えて、データが正常にターゲット・データベースにロードされることを確認します。バイト・スワップとは、ビッグ・エンディアン形式のデータをリトル・エンディアン形式に変換すること、またはその逆の変換を意味します。

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。

> **参照：** SQL*Loader でのバイト・スワップの処理例については、「事例 11: Unicode キャラクタ・セットのデータのロード」を参照してください。（事例へのアクセス方法については、6-12 ページの「SQL*Loader の事例」を参照してください。）

バイト順序の指定

入力データ・ファイルのデータのバイト順序を指定するには、SQL*Loader 制御ファイルの次の構文を使用します。



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・ファイルに対して異なるバイト順序を指定することができます。ただし、INFILE パラメータの前の BYTEORDER 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 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 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 キャラクタ・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 以外のキャラクタ・セットのデータ・ファイルの文字データ
  - ZONED データ型または（PACKED）DECIMAL データ型

バイト順序マーク（BOM）の使用

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。キャラクタ・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 キャラクタ・セットが使用され、BOM も使用されている場合は、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、バイト順序依存の Unicode キャラクタ・セット（UTF16）が使用され、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 キャラクタ・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・ファイルでキャラクタ・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 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 キャラクタ・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。

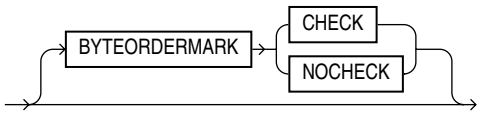
つまり、LOBFILE および SDF に対するバイト順序インジケータの優先順位は次のようになります。

- 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

> **注意：** データ・ファイルのキャラクタ・セットが Unicode キャラクタ・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。

BOM の確認の抑止

Unicode キャラクタ・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 の確認もできます。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、SQL*Loader で BOM が確認されます。これは、Unicode キャラクタ・セットのデータ・ファイルのデフォルトの動作です。ただし、この仕様は説明のために制御ファイルで使用される場合があります。Unicode 以外のキャラクタ・セットが使用されているデータ・ファイルに BYTEORDERMARK CHECK を指定すると、エラーが発生します。

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。
- LOBFILE データ・ファイルおよび SDF のみではなくプライマリ・データ・ファイルに関する構文の指定にも適用されます。
- 異なるデータ・ファイルに異なる BYTEORDERMARK 値を指定することができます。ただし、INFILE パラメータの前の BYTEORDERMARK 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。
- LOBFILE または SDF で Unicode 以外のキャラクタ・セットが使用された場合に、値 CHECK が無視される以外は、プライマリ・データ・ファイルの BYTEORDERMARK 指定は LOBFILE および SDF のデフォルトとしても使用されます。LOBFILE および SDF のデフォルト設定は、LOBFILE または SDF 仕様部で BYTEORDERMARK を指定して、上書きできます。

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。

参照：

- NULLIF 句を使用して、すべてが空白のフィールドを NULL としてロードする方法の例については、「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。（事例へのアクセス方法については、6-12 ページの「SQL*Loader の事例」を参照してください。）
- 「空白の切捨て」（9-32 ページ）
- 「空白の切捨てに対する PRESERVE BLANKS オプションの影響」（9-37 ページ）

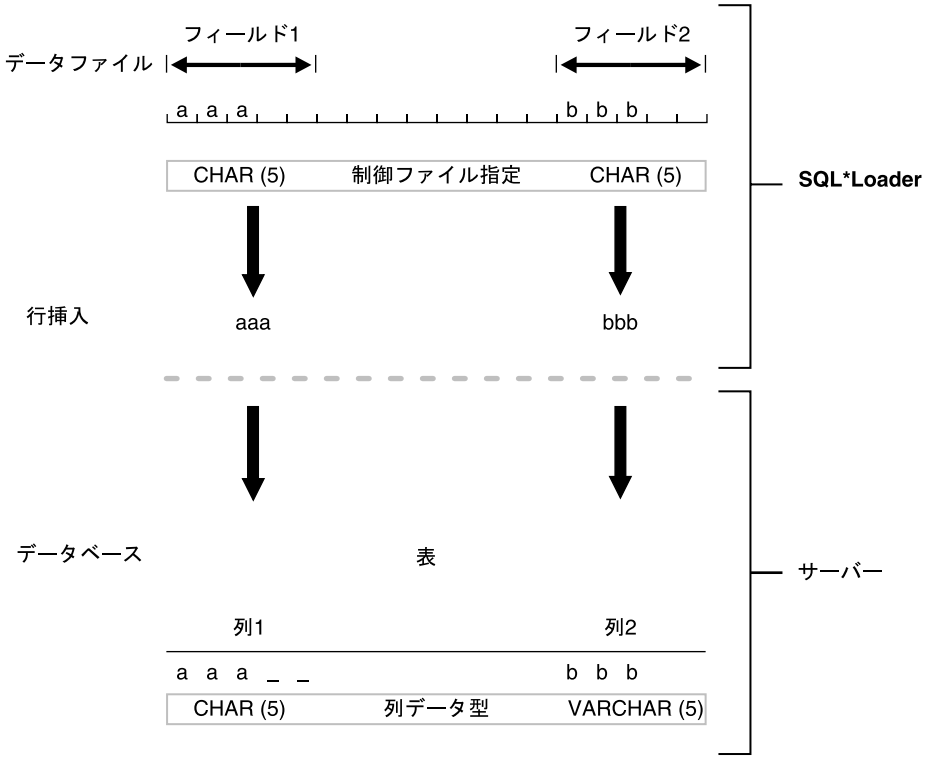
空白の切捨て

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 9-1 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、NVARCHAR、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。その後、図 9-1 に示すとおり、フィールド 1 とフィールド 2 は 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 9-1 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 9-5 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については、9-37 ページの「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 9-5 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[1]	後続の空白[1]
サイズ指定あり	__aa__	__aa	あり	なし
終了デリミタ	__aa__,	__aa__	あり	あり[2]
囲みデリミタ	"__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__	なし	あり
前のフィールドが空白で区切られている場合	__aa__	aa[3]	なし	[3]

1 空白のみのフィールドが切り捨てられた場合、値は NULL になります。

2 空白で終了するフィールドを除きます。

3 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。

- CHAR データ型
- 日時データ型および期間のデータ型
- 数値型 EXTERNAL データ型
  - INTEGER EXTERNAL
  - FLOAT EXTERNAL
  - (PACKED) DECIMAL EXTERNAL
  - ZONED (10 進) EXTERNAL

> **注意：** 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。

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

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を持つ位置指定は、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

フィールドのサイズを事前に決定するには、フィールドの開始位置と終了位置を指定するか、フィールド長を指定します。それぞれの指定例を示します。

```
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。

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 '|'
```

フィールドの相対的な位置指定

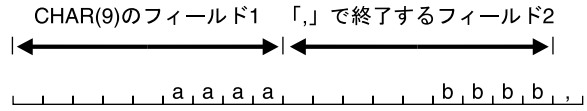
この項では、次の条件で SQL*Loader を使用してフィールドの開始位置を決定する方法について説明します。

- フィールドの開始位置が指定されていない場合
- 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。図 9-2 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。

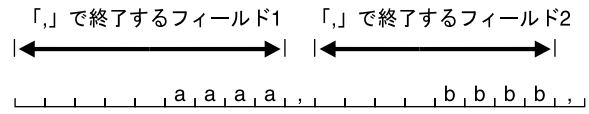
図 9-2 固定長フィールドの後の相対的な位置指定



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。この例を図 9-3 に示します。

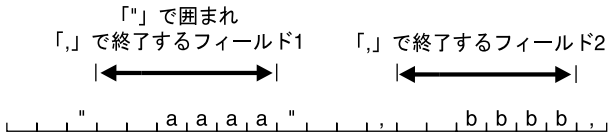
図 9-3 デリミタ付きフィールドの後の相対的な位置指定



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。この例を図 9-4 に示します。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 9-4 囲みデリミタの後の相対的な位置指定



先頭の空白

図 9-4 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。

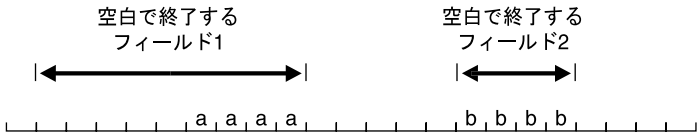
- 前のフィールドが空白で区切られて（終了して）いて、現行のフィールドの開始位置が指定されていない場合。
- フィールドに対してオプションの囲みデリミタが指定されているにもかかわらず、その囲みデリミタが使用されていない場合。

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。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 9-5 に示します。

図 9-5 空白で区切られたフィールド



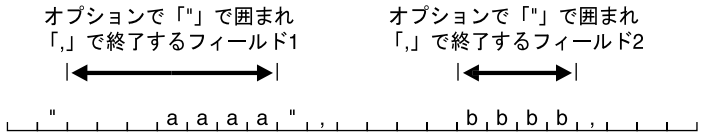
このようなケースは、前述の例のとおり前のフィールドが TERMINATED BY WHITESPACE 句で明示的に指定された場合に発生します。グローバルに FIELDS TERMINATED BY WHITESPACE 句が指定された場合も、このケースに相当します。

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 9-6 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 9-6 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。

注意： 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 9-6 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。

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。

囲まれたフィールドの切捨て

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 9-6 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。

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、CHAR、DATE または数値型 EXTERNAL フィールドのいずれか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
```

[NO] PRESERVE BLANKS とデリミタ句の併用

デリミタ句が指定されていると、PRESERVE BLANKS オプションに次のような影響があります。

- 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 POSITION 句の指定がない場合は、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。

参照：「空白の切捨て」(9-32 ページ)

フィールドへの SQL 演算子の適用

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。式の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

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、次の制御ファイルの指定例に示すとおり、2 つの名前を完全に同じ記述に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
Phil Grant
Jason Taylor
```

前述の例では、次のことに注意してください。

- 表の作成時、(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" のすべてが同じ列を指すため、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。

次の要件および制限は SQL 文字列を使用している場合に適用されます。

- 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組合せを保持しています。また、二重引用符を使用すると、バックスラッシュ（エスケープ）文字も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。

  - `:CHILD.\"HEIGHT_%TILE"`
  - `: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、バインド変数を正確に指定するように求めるためのものであり、ロードが異常終了することはありません。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列オブジェクト属性フルネームとしてではなく、1 つの識別子として解釈されるため、引用符ルールは変更されました。
- 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 SQL 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、FILLER 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。これには、列オブジェクトの属性である `VARRAY`、ネストした表または LOB 列も含まれます。
- 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・データを返さない場合があります。

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、次の例の説明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)  "UPPER(:field4.attr3)",
  attr2  CHAR(2),
  attr3  CHAR(3)  ":field4.attr1 + 1"
 ),
 field5  EXPRESSION "MYFUNC(:FIELD4, SYSDATE)"
)
BEGINDATA
ABCDEF1234511  ,:field1500YYabc
abcDEF67890    ,:field2600ZZghl
```

この例に関する注意事項

- 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、8-4 ページの「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。フィールド `FIELD4.ATTR1` は、入力レコードの値 500 および 600 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。

- `field5` 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。

フィールド指定での SQL 演算子の共通使用

次のタスクでは、共通して 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)"
```

SQL 演算子の組合せ

複数の演算子を次の例のように組み合せることができます。

```
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' )"
```

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')
```

DATE フィールド・データ型を使用する場合、日付マスクなしの SQL 文字列は使用できないことに注意してください。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、単純な解決策としては、CHAR データ型を使用します。

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。次に例を示します。

```
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、書式化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。（事例へのアクセス方法については、6-12 ページの「SQL*Loader の事例」を参照してください。）

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起動します。

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。

参照 :

- 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 データベース型の使用方法の詳細は、『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』を参照してください。

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。

- CONSTANT パラメータ
- EXPRESSION パラメータ
- RECNUM パラメータ
- SYSDATE パラメータ
- SEQUENCE パラメータ

ファイルを使用しないデータのロード

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。

列への定数値の設定

これは、生成するデータとしては最も単純な形式です。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

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。

注意： 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。

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

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"
```

列へのデータ・ファイルのレコード番号の設定

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

列名および RECNUM の組合せを指定すると、列指定は完結します。

```
column_name RECNUM
```

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ものと同じ、現在のシステム日付が列に設定されます。詳細は、『Oracle Database SQL 言語リファレンス』の「DATE データ型」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を指定すると、列指定は完結します。

```
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。

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

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。

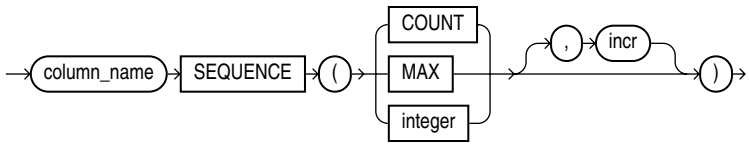


表 9-6 では、列指定に使用されるパラメータについて説明します。

表 9-6 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。（事例へのアクセス方法については、6-12 ページの「SQL*Loader の事例」を参照してください。）

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

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。すべての INTO TABLE 句で増分値は 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。

10

オブジェクト、LOB およびコレクションのロード

この章の内容は、次のとおりです。

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。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。

注意： 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。

そのため、フィールドを参照する句を使用する場合は、フルネームで指定する必要があります。たとえば、フィールド fld1 が COLUMN OBJECT に指定されており、そこにフィールド fld2 が含まれる場合、NULLIF などの句で fld2 を指定する場合は、フィールドのフルネーム 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・レコード形式への列オブジェクトのロード
- 可変レコード形式への列オブジェクトのロード
- ネスト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のロード
- オブジェクトに対する NULL 値の指定
- 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

ストリーム・レコード形式への列オブジェクトのロード

例 10-1 に、事前にサイズが決まっているフィールドにデータがある例を示します。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 10-1 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 'sample.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（sample.dat）

```
101 Mathematics  Johny Quest       30   1024
237 Physics      Albert Einstein   65   0000
```

注意

1. この列オブジェクトの型指定は、ネストした列オブジェクトの記述にも繰り返し使用できます。

可変レコード形式への列オブジェクトのロード

例 10-2 に、デリミタ・フィールドにデータがある例を示します。

例 10-2 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,
```

注意

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 の 32 乗 -1 で、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。

ネストした列オブジェクトのロード

例 10-3 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 10-3 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,
```

注意

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。

導出サブタイプを使用した列オブジェクトのロード

例 10-4 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 10-4 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,
```

注意

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を返します。

オブジェクトに対する NULL 値の指定

非スカラー・データ型で NULL 値を指定する場合、スカラー・データ型で指定するよりも複雑です。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

オブジェクト列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 10-5 に、この方法を示します。

例 10-5 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 NULLIF dept_mgr.name=BLANKS,
1    age        POSITION(35:37)    INTEGER EXTERNAL NULLIF dept_mgr.age=BLANKS,
1    emp_id     POSITION(40:46)    INTEGER EXTERNAL 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
```

注意

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。

アトミック NULL の指定

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 10-5 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 10-6 に示します。

例 10-6 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,
```

注意

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。

参照：「FILLER フィールドの指定」（9-5 ページ）

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

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。この相違点を例 10-7 に示します。

例 10-7 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,
```

注意

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 10-8 に示す定義について考えてみます。

例 10-8 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 10-7 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 10-9 に、この方法を示します。

例 10-9 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,
```

注意

1. この場合、従業員列オブジェクトが、SQL 式を使用してロードされます。この式によって、正しい数の引数が含まれているユーザー定義コンストラクタが起動されます。名前フィールドの `Johny Q.` および `Albert Einstein` は、両方とも小文字でロードされます。また、各行の従業員列オブジェクトの従業員識別子は、`employee_ids` 順序番号から値を取ります。

例 10-9 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.
```

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 10-10 に、主キー OID を使用したオブジェクト表のロード例を示します。

例 10-10 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、システム生成実 OID を含むデータ・ファイルにマップされた、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 10-11 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 10-11 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,
```

注意

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。前述のとおり、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。この概念を例 10-12 に示します。

例 10-12 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,
```

注意

1. TREAT AS 句で指定すると、SQL*Loader では、実際の宣言型が employee_type であるオブジェクト表が、hourly_emps_type 型であると宣言されたかのように処理されます。
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、この属性は、オブジェクト表の宣言型の属性ではないためエラーを返します。

REF 列のロード

SQL*Loader は、システム生成 OID の REF 列、主キー 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。これらの各列については、次の項で説明するとおり、表名を指定する方法が重要です。

REF 句における表名の指定

> **注意：** この項で説明する内容は、SQL*Loader および Oracle Database のバージョンがいずれも 11*g* リリース 1（11.1）の環境にのみ適用されます。SQL*Loader、Oracle Database またはその両方が以前のバージョンの環境には適用されません。

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 句で指定する表名が、定数として、または FILLER フィールドを使用して指定されるかどうかにかかわらず、大 / 小文字を区別して認識されます。このため、次のような状況が発生します。

- 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、定数が 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。

システム生成 OID 型の REF 列

システム生成 REF 列をロードする場合、SQL*Loader では、REF 列を構築する実 OID が残りのデータとともにデータ・ファイル内にあるとみなされます。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。適切な構文については、A-5 ページの「ref_spec」を参照してください。例 10-13 に、システム生成 OID 型の REF 列のロード例を示します。

例 10-13 システム生成 REF 列のロード

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,
```

注意

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。

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。適切な構文については、A-5 ページの「ref_spec」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 10-14 に、主キー REF 列のロード例を示します。

例 10-14 主キー REF 列のロード

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,
```

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。このような REF 列をロードするための構文は、システム生成 OID 型の REF 列または主キー REF 列にロードする場合と同じです。例 10-13「システム生成 REF 列のロード」および例 10-14「主キー REF 列のロード」を参照してください。

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、別の REF はオブジェクト表 Y を指す）をロードする場合、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。次に例を示します。

```
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'
)
```

この 2 つの方法のいずれも使用しない場合、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。

- コレクション内の有効範囲なし主キー REF は、SQL*Loader ではサポートされていません。
- この REF 列にシステム生成の REF をロードする場合は、10-11 ページの「システム生成 OID 型の REF 列」で説明されてい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、10-12 ページの「主キー REF 列」で説明されている制限も適用されます。

注意： 主キーが使用可能な有効範囲なし REF 列の場合、SQL*Loader では、(REF ディレクティブまたはデータ行のいずれかから) 最初の有効な解析済オブジェクト表が取得され、オブジェクト表の OID 型を使用してその単一表へのロードで参照可能な REF 型が決定されます。

LOB のロード

LOB は、ラージ・オブジェクト型です。SQL*Loader では、次の LOB 型がサポートされています。

- BLOB: 非構造化バイナリ・データを含む内部 LOB。
- CLOB: 文字データを含む内部 LOB。
- NCLOB: 各国語キャラクタ・セットの文字を含む内部 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 は列データ型で、NCLOB 以外は、オブジェクトの属性データ型です。LOB は実際の値を持ち、その値は NULL でも「値なし（空）」でもかまいません。LOB に格納される長さ 0 のフィールドがある場合は、SQL*Loader によって空の LOB が作成されます。（他のデータ型の場合は、このデータ型とは異なり、長さ 0 の文字列に対して、列は SQL*Loader によって NULL に設定されます。）つまり、NULL 値を LOB 列にロードする方法は、NULL 句を使用する方法のみです。

XML 列は、SYS.XMLTYPE 型であると宣言された列です。SQL*Loader は、XML 列を CLOB と同様に処理します。次の項で説明するプライマリ・データ・ファイルまたは LOBFILES からの LOB データのロード方法は、XML 列のロードに適用できます。

注意： LOB フィールドに SQL 文字列は指定できません。これは、LOBFILE_spec を指定する場合も同じです。

LOB は非常に大きなデータであるため、SQL*Loader では、LOB データをプライマリ・データ・ファイル（残りのデータを持つインライン）または LOBFILE のいずれかからロードできます。この項では、次の項目について説明します。

- プライマリ・データ・ファイルからの LOB データのロード
- LOBFILE 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 および NCLOB）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

これらの各形式については、次の項で説明します。

事前に決められたサイズのフィールドの LOB データ

例 10-15 に示すように、これは LOB データをロードする場合、最も高速で、概念的に単純な形式です。

注意： ロードする LOB データは、サイズが均等ではないため、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。

この形式で LOB をロードするには、ロード時のデータ型として CHAR または RAW を使用する必要があります。

例 10-15 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）

```
Johny Quest      Johny Quest
             500 Oracle Parkway
             jquest@us.oracle.com ...
```

注意

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。

デリミタ付きフィールドの LOB データ

この形式で、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 で区切り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・ファイルのキャラクタ・セットに注意してください。データ・ファイルのキャラクタ・セットが制御ファイルのキャラクタ・セットと異なる場合は、デリミタを 16 進文字列の表記法で指定できます（*X'hexadecimal string'*）。デリミタを実際に 16 進文字列で指定する場合は、入力データ・ファイルのキャラクタ・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 進文字列で指定しない場合、デリミタは、クライアント（制御ファイル）のキャラクタ・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・ファイルのキャラクタ・セットに変換されます。

次のこと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。

> **注意：** 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、9-11 ページの「CHAR」を参照してください。

例 10-16 にデリミタ付きフィールドへの LOB データのロードの例を示します。

例 10-16 デリミタ付きフィールドの LOB データのロード

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）

```
Johny Quest,<startlob>        Johny Quest
                          500 Oracle Parkway
                          jquest@us.oracle.com ...   <endlob>
2  |Speed Racer, .......
```

注意

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、8-17 ページの「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'）。

Length-Value Pair フィールドの LOB データ

VARCHAR、VARCHARC または VARRAW データ型を使用して、Length-Value Pair フィールドで編成された LOB データをロードできます。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、ロード前に各 LOB の長さの確認が必要です）。例 10-17 に、Length-Value Pair フィールドの LOB データのロード例を示します。

例 10-17 Length-Value Pair フィールドへの LOB データのロード

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）

```
  Johny Quest,479                 Johny Quest
                               500 Oracle Parkway
                               jquest@us.oracle.com
                                      ... <endrec>
3    Speed Racer,000<endrec>
```

注意

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、8-17 ページの「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。

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）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。

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ためには、SQL*Loader データ型の VARRAW、VARCHAR または VARCHARC 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、10-17 ページの「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、A-6 ページの「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。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込みます。

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。

ファイル当たり 1 つの LOB 例 10-18 では、各 LOBFILE は、それぞれ 1 つの LOB のソースです。この方法で編成された LOB データをロードするには、列名またはフィールド名の後に LOBFILE データ型を指定します。

例 10-18 LOBFILE 当たり 1 つの 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  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
     ...
```

注意

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。

事前に決められたサイズの LOB 例 10-19 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 10-19 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
     ...
```

注意

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、8-17 ページの「文字長セマンティクス」を参照してください。

デリミタ付きフィールドの LOB 例 10-20 では、LOBFILE がデリミタ付きフィールドである場合の、LOB データの例を示します。この形式では、サイズの異なる LOB を同じ列にロードしても、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 で区切り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 10-20 デリミタ付きフィールドの 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 '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>
```

注意

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 のロード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（LOBFILE 内の LOB には順序が不要）内での LOB の相対的な位置指定に関して、多少柔軟に対応できます。

Length-Value Pair で指定した LOB 例 10-21 では、LOBFILE の各 LOB の先頭でデータ長が定義されています。VARCHAR、VARCHARC または VARRAW データ型を使用して、この方法で編成された LOB データをロードできます。

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 10-21 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

```
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
```

注意

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、8-17 ページの「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。

LOBFILE から LOB をロードする場合の考慮点

LOBFILE から LOB をロードする場合は、次のことに注意してください。

- 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、そのレコードの LOB は、空の LOB になります。XML 列の場合、LOB のロード中に失敗すると 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。ただし、最大長を指定すると、メモリー使用量の最適化のヒントに使用されます。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。このような場合のロード・パフォーマンスを最適化するには、『Oracle Database SecureFiles およびラージ・オブジェクト開発者ガイド』の一時 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。

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。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 としてロードされるファイルは、ロード時に存在している必要はなく、後で作成できます。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。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。

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。いずれの引数も文字列定数として指定するか、他のフィールドを介して動的にロードできます。詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

次の 2 つの BFILE のロード例を示します。例 10-22 では、1 つのファイル名のみを動的に指定する方法を、例 10-23 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 10-22 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

```
LOAD DATA
INFILE sample.dat
INTO TABLE planets
FIELDS TERMINATED BY ','
   (pl_id    CHAR(3),
   pl_name   CHAR(20),
   fname     FILLER CHAR(30),
1  pl_pict   BFILE(CONSTANT "scott_dir1", fname))
```

データ・ファイル（sample.dat）

```
1,Mercury,mercury.jpeg,
2,Venus,venus.jpeg,
3,Earth,earth.jpeg,
```

注意

1. ディレクトリ名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。

例 10-23 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

```
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),
1  dname     FILLER CHAR(20),
   pl_pict   BFILE(dname, fname) )
```

データ・ファイル（sample.dat）

```
1, Mercury, mercury.jpeg, scott_dir1,
2, Venus, venus.jpeg, scott_dir1,
3, Earth, earth.jpeg, scott_dir2,
```

注意

1. 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。SDF の詳細は、10-23 ページの「セカンダリ・データ・ファイル（SDF）」を参照してください。

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、A-9 ページの「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。詳細は、10-2 ページの「列オブジェクトのロード」を参照してください。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、SDF を指定できます。
- 同じ SDF のファイル上にないかぎり、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 10-24 に、VARRAY およびネストした表のロード例を示します。

例 10-24 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
```

注意

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。

セカンダリ・データ・ファイル（SDF）

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、制御ファイルごとに指定されます。SDF 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。

注意： 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・ファイルについては、各 SDF に対して次の指定ができます。

- 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 のキャラクタ・セットの指定（8-13 ページの「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。

参照：

- 「READSIZE（読取りバッファ・サイズ）」（7-8 ページ）
- 「OPTIONS 句」（8-3 ページ）
- 「sdf_spec」（A-9 ページ）

動的および静的 SDF 指定

SDF を静的に指定（実際のファイル名を指定）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、SDF の EOF に到達するとファイルはクローズされ、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。

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。

このようなデータ・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込みます。

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。SID 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。

例 10-25 に、ユーザー定義 SID を使用した親表のロード方法を示します。

例 10-25 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,|
```

注意

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 10-26 に、ユーザー定義 SID を使用した子表（ネストした表の記憶表）のロード方法を示します。

例 10-26 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,
```

注意

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

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 ～ 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。

11

従来型パス・ロードおよびダイレクト・パス・ロード

この章では、SQL*Loader の従来型パス・ロードとダイレクト・パス・ロードの方法について説明します。この章の内容は、次のとおりです。

ダイレクト・パス・ロードを使用した例は、「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。その他の事例では、従来型パス・ロードを使用しています。(事例のアクセス方法については、6-12 ページの「SQL*Loader の事例」を参照してください。)

データのロード方法

SQL*Loader でデータをロードするには、次の 2 つの方法があります。

- 従来型パス・ロード
- ダイレクト・パス・ロード

従来型パス・ロードでは、Oracle Database の表に対して（1 つ以上の）SQL INSERT 文が実行されます。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、データ・ブロックを直接データ・ファイルに書き込むため、Oracle Database のオーバーヘッドが大幅に削減されます。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。この章では、ダイレクト・パス・ロード固有の問題（制限、セキュリティ、バックアップなど）について説明します。

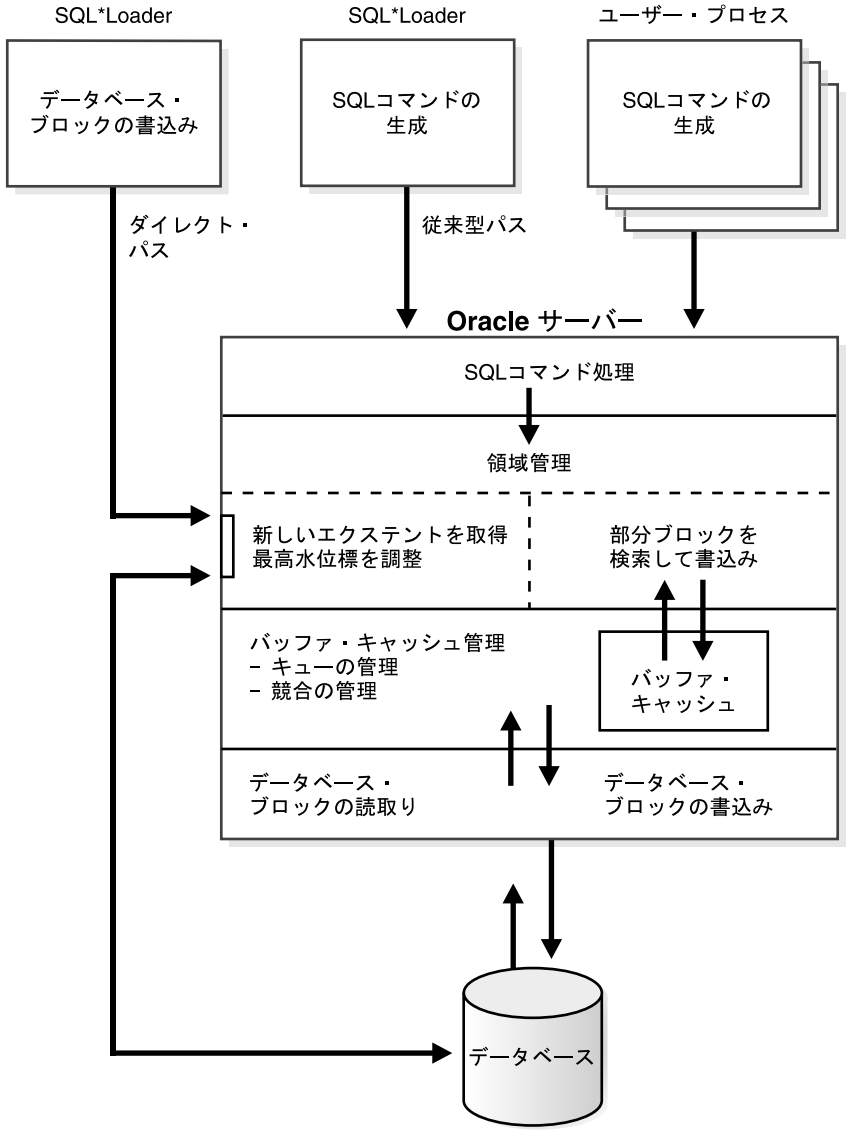
データをロードする表はデータベース中に存在している必要があります。SQL*Loader では、表は作成されません。すでにデータが含まれているか、または空である既存の表にロードされます。

ロードを実行するには、次の権限が必要です。

- ロードする表についての INSERT 権限。
- ロードする表にすでにデータが存在するために、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、その表についての DELETE 権限。

図 11-1 に、従来型パス・ロードおよびダイレクト・パス・ロードでのデータベースへの書込み方法を示します。

図 11-1 SQL*Loader ダイレクト・パスおよび従来型パスでのデータベースの書込み



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。

従来型パス・ロード

従来型パス・ロード（デフォルト）では、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。この方法は、すべての Oracle のツール製品および Applications で使用されます。

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。このため、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 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、このアクション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。

参照：「中断された従来型パス・ロード」（8-19 ページ）

単一パーティションの従来型パス・ロード

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。

従来型パスを使用する場合

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・ロードを使用する必要がある場合もあります。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。

  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。

  ダイレクト・パス・ロードでは、クラスタ表に対するロードはサポートされていません。

- 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。

  ダイレクト・パス・ロードでは、既存の索引を新しい索引キーとマージするために、既存の索引をコピーします。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。

  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間は使用禁止になります。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 エラーが発生した場合。
  - 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。

ダイレクト・パス・ロード

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 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。

参照：「中断されたダイレクト・パス・ロード」（8-19 ページ）

ダイレクト・パス・ロード時のデータ変換

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。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）は使用されません。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（初期化パラメータ・ファイルに NLS パラメータを設定することと同じ）か、または適切な環境変数を設定します。たとえば、フィールドに日付書式を指定するには、例 11-1 に示すとおり SQL*Loader の制御ファイルに日付書式を設定するか、または例 11-2 に示すとおり環境変数 NLS_DATE_FORMAT を設定します。

例 11-1 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 11-2 環境変数 NLS_DATE_FORMAT 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 csh の場合：

```
%setenv NLS_DATE_FORMAT='YYYYMMDD'
```

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。

注意： 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、ロード全体がロードバックされます。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。

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

パーティション表の単一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でき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・ロードでは、データベース処理は最小限に抑えられますが、ロードの開始時と終了時に、それぞれロードの初期化と終了処理のために Oracle Database に対するコールが数回実行されます。また、ロードの初期化中に必要な DML ロックがかけられ、ロード終了時に解除されます。ロード中には、索引キーが作成されてソートに使用されたり、必要に応じて新しいエクステントを取得するために領域管理ルーチンを使用することによって、データ・セーブポイントの上限（最高水位標）を調整するなどの動作も発生します。上限の調整については、11-11 ページの「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。

ダイレクト・パス・ロードのメリット

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 は SQL INSERT 文を実行しないため、Oracle Database の処理負荷が減ります。
- 表と索引をロード開始時にロックするように Oracle に要求し、ロード完了時にロック解除を要求します。これに対し、従来型パス・ロードでは、行配列ごとに Oracle を 1 回コールして、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・ロードを使用するプロセスは、Oracle 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。これによって、他の Oracle 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。
- ロードする表が空の場合、事前ソート・オプションを使用すると索引作成のソート段階とマージ段階を省略できます。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

  詳細は、11-12 ページの「インスタンス・リカバリおよびダイレクト・パス・ロード」を参照してください。

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。

- 表がクラスタ化されていないこと。
- ロードする表に未処理のアクティブ・トランザクションがないこと。

  この条件が満たされているかどうかを確認するには、MONITOR TABLE という OEM 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。
- Oracle9*i* より前のバージョンのデータベースでは、クライアントとサーバーが同じバージョンである場合のみ、SQL*Loader のダイレクト・パス・ロードを実行できます。また、Oracle9*i* のデータのダイレクト・パス・ロードは、以前のバージョンのデータベースに対しては実行できません。たとえば、ダイレクト・パス・ロードを使用して、Oracle9*i* リリース 1（9.0.1）のデータベースから Oracle8*i* リリース 8.1.7 のデータベースに、データはロードできません。

  Oracle9*i* 以上では、クライアントとサーバーのバージョンが異なる場合にも、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 Oracle9*i* リリース 1（9.0.1）以上で、クライアントのバージョンは、サーバーのバージョン以下である必要があります。たとえば、Oracle9*i* リリース 1（9.0.1）のデータベースから Oracle9*i* リリース 2（9.2）のデータベースへのダイレクト・パス・ロードを実行できます。たとえば、ダイレクト・パス・ロードを使用して、Oracle9*i* リリース 1（9.0.1）のデータベースから Oracle8*i* リリース 8.1.7 のデータベースに、データはロードできません。
- ダイレクト・パス・モードでロードする表に、VPD ポリシーが INSERT でアクティブにされていないこと。

次の機能は、ダイレクト・パス・ロードでは使用できません。

- 親表を子表とともにロード
- BFILE 列のロード
- ロード時の CREATE SEQUENCE の使用。これは、ダイレクト・パスでは INSERT 文が生成されないため、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。

単一パーティションのダイレクト・パス・ロードでの制限

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（Disable）であること。
- トリガーが使用禁止（Disable）であること。

ダイレクト・パスを使用する場合

前述の制限のいずれにも該当しない場合は、次のようなときにダイレクト・パス・ロードを使用してください。

- 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。詳細は、11-23 ページの「パラレル・データ・ロード・モデル」を参照してください。

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。ロード中には NOT NULL 制約が施行されます。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、一意制約が施行されます。

一意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、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。詳細は、11-19 ページの「ダイレクト・ロード、整合性制約およびトリガー」を参照してください。

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。

シノニムへのロード

ダイレクト・パス・ロードでは、表のシノニムにデータをロードできます。ただし、そのためには、そのシノニムが表を直接指している必要があります。シノニムがビューのシノニムであるか、他のシノニム用のシノニムである場合は、データはロードできません。

ダイレクト・パス・ロードの使用

この項では、SQL*Loader のダイレクト・パス・ロードの使用方法を説明します。

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には、セットアップ・スクリプトの `catldr.sql` を実行し、必要なビューを作成します。このスクリプトは、ダイレクト・ロードを行う予定のデータベースそれぞれに対して 1 回のみ実行します。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。

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、次の形式で、コマンドラインまたはパラメータ・ファイル（使用している場合）の `DIRECT` パラメータに `true` を設定します。

```
DIRECT=true
```

参照：

- ダイレクト・パス・ロードのパフォーマンスの最適化に使用可能なパラメータの詳細は、11-13 ページの「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 CPU システムまたは分散システムでダイレクト・パス・ロードを使用する場合は、11-17 ページの「複数 CPU システムのダイレクト・パス・ロードの最適化」を参照してください。

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。詳細は、8-29 ページの「SINGLEROW オプション」を参照してください。

注意： 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、マージも行われません。この場合は、索引にキーが直接付加されます。詳細は、11-13 ページの「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。

参照： 索引のサイズを計算する方法および記憶域パラメータを設定する方法の詳細は、『Oracle Database 管理者ガイド』を参照してください。

一時セグメント記憶域要件

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、キー記憶域が次のように定義されます。

```
key_storage = (number_of_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。詳細は、11-14 ページの「高速索引付けのためのデータの事前ソート」を参照してください。

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。

データ・セーブを使用したデータ損失の防止

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。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、まず索引を削除してからロードを続行します。ロードの終了後、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、11-11 ページの「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。

> **注意：** 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。）

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。

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。

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メモリー内で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。

> **参照：** リカバリの詳細は、『Oracle Database 管理者ガイド』を参照してください。

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. RECOVER コマンドを使用して、表領域をリカバリします。

> **参照：** Recovery Manager の RECOVER コマンドの詳細は、『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。

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、11-10 ページの「使用禁止状態（Index Unusable）のままの索引」を参照してください。

LONG 型データ・フィールドのロード

SQL*Loader の最大バッファ・サイズよりも長いデータをダイレクト・パスでロードするには、LOB を使用します。LOB に大きい STREAMSIZE 値を使用すると、パフォーマンスが向上します。

> **参照：**
>
> - 「LOB のロード」（10-13 ページ）
> - 「列配列の行数およびストリーム・バッファ・サイズの指定」（11-16 ページ）

次の項で説明するように、PIECED パラメータを使用すると、最大バッファ・サイズより長いデータもロードできますが、LOB を使用することをお薦めします。

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。

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。

ダイレクト・パス・ロードのパフォーマンスの最適化

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、次の方法があります。

- 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

領域を最小化するには、次の方法があります。

- 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。

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。

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、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。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、11-10 ページの「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。したがって、一時記憶域は不要であり、時間も短縮できます。

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。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。（事例へのアクセス方法については、6-12 ページの「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。

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。

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
...              ...
```

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、手順 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。

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。

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 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。アーカイブの詳細は、『Oracle Database 管理者ガイド』を参照してください。

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。

注意： 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));
```

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。

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。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な値は、使用しているシステム、入力データ型および Oracle の列データ型によって異なります。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。

これらのパラメータのデフォルト値のリストは、7-2 ページの「SQL*Loader の起動」で説明したとおり、パラメータを指定しないで SQL*Loader を起動すると参照できます。

注意： 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。詳細は、11-17 ページの「複数 CPU システムのダイレクト・パス・ロードの最適化」を参照してください。

日付キャッシュの値の指定

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。
- ほぼすべての入力データ値が一意の場合、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。

注意： 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。

参照： 「DATE_CACHE」（7-4 ページ）

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、11-16 ページの「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。

> **参照：** 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。

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、8-29 ページの「SINGLEROW オプション」を参照してください。

ダイレクト・ロード、整合性制約およびトリガー

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。このセクションでは、これらの機能に関するダイレクト・パス・ロードの使用について説明します。

整合性制約

整合性制約には、ダイレクト・パス・ロード時に自動的に使用禁止になるものがあります。また、使用禁止にならないものもあります。制約の詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』の制約によるデータ整合性のメンテナンスを参照してください。

使用可能な制約

ダイレクト・パス・ロード時でも使用可能な制約は、次のとおりです。

- NOT NULL 制約
- 一意制約
- 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、11-10 ページの「使用禁止状態（Index Unusable）のままの索引」を参照してください。

使用禁止の制約

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。

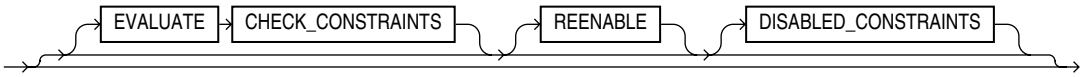
- CHECK 制約
- 参照制約（外部キー）

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
```

制約を使用可能に戻す方法

REENABLE 句を指定しておくと、ロードの完了時に、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 句を使用する場合は、指定する表がすでに存在していて、その表への挿入が可能である必要があります。ここで指定する表には、整合性制約のいずれかに違反したすべての行の ROWID 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。この例外表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。

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ます。また、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。

注意： 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;
```

注意： 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。

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。

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。

準備

いずれの方法の場合も、表に対して事前に行うべき作業があります。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（VARCHAR2）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。

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。

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）。

ストアド・プロシージャの使用

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. 次のようにして、挿入トリガーと同じ処理を行うストアド・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。

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、SQL*Loader では使用禁止のトリガーおよび制約が使用可能に戻されます。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。

適切なロックを獲得できなかったためにダイレクト・ロードが終了した場合は、ログ・ファイルを確認してください。ログ・ファイルには、使用禁止になっていたトリガーおよび制約と、それらを使用可能に戻そうと試みた履歴が記録されます。SQL*Loader によって使用可能に戻せなかったトリガーや制約は、『Oracle Database SQL 言語リファレンス』で説明されている `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。

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。

パラレル・データ・ロード・モデル

この項では、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 つの基本モデルについて説明します。

- 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

従来型パスによる同時ロード

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、11-23 ページの「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。

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。）

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。

パラレル・ダイレクト・パス・ロードの制限

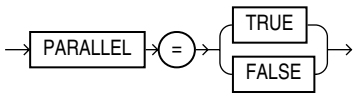
パラレル・ダイレクト・パス・ロードには次の制限があります。

- 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。
- 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。
- 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、個別のロードによって整合性がとられないためです）。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。
- ロードする表に、LOB 列または LOB として格納される列（VARRAY など）を指定できません。

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）。

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・セッションを起動するには、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。

注意： 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。

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、SQL*Loader 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。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。

次に例を示します。

```
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。

FILE パラメータの使用 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。

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。

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。

主キー制約および一意制約

主キーおよび一意制約が有効の場合、表に索引が作成されます。その後、その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。

参照：『Oracle Database パフォーマンス・チューニング・ガイド』

一般的なパフォーマンス改善のヒント

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、SQL*Loader 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、SQL*Loader 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、キャラクタ・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・キャラクタ・セットを使用します。
  - キャラクタ・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 つのキャラクタ・セットがサポートされています。
    * クライアント・キャラクタ・セット（クライアント sqlldr プロセスの NLS_LANG）
    * データ・ファイル・キャラクタ・セット（通常、クライアント・キャラクタ・セットと同じ）

* データベース・キャラクタ・セット
* データベース・キャラクタ・セット

すべてのキャラクタ・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・キャラクタ・セットとデータベース・サーバー・キャラクタ・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。キャラクタ・セットが同じ場合、キャラクタ・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 INDEX 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。詳細は、8-21 ページの「CONCATENATE を使用した論理レコードの作成」を参照してください。

さらに、12-8 ページの「外部表使用時のパフォーマンスのヒント」で説明しているパフォーマンスのヒントは、SQL*Loader にも適用できます。

第 III 部

外部表

第 III 部では、外部表の使用方法について説明します。

第 12 章「外部表の概要」

この章では、外部表の基本概念について説明します。

第 13 章「ORACLE_LOADER アクセス・ドライバ」

この章では、ORACLE_LOADER アクセス・ドライバについて説明します。

第 14 章「ORACLE_DATAPUMP アクセス・ドライバ」

この章では、ORACLE_DATAPUMP アクセス・ドライバのパラメータ、サポートされるデータ型のロードおよびアンロードに関する情報などについて説明します。

12

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。この機能によって、データベースに表がある場合と同様に、外部ソースのデータにアクセスできます。

Oracle Database 10*g* より前のリリースでは、外部表は読取り専用でした。Oracle Database 10*g* からは、外部表は書込みも可能となりました。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、12-10 ページの「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_LOADER アクセス・ドライバを使用し、データ・ファイルがテキスト形式である場合に、外部表機能を使用するには、ご使用のプラットフォーム上のデータ・ファイルのファイル形式およびレコード形式の知識が必要です。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。

この章の内容は、次のとおりです。

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。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。ORACLE_LOADER 型および ORACLE_DATAPUMP 型の 2 種類の型が選択可能です。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 ドライバは、デフォルトです。このドライバでは、データのロードのみを実行できます。また、データはテキスト・データ・ファイルからロードする必要があります。外部表から内部表へのロードは、外部表にあるテキストのみで構成されるデータ・ファイルから読み取ることによって実行されます。
  - ORACLE_DATAPUMP アクセス・ドライバでは、ロードおよびアンロードを実行できます。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。
- DEFAULT DIRECTORY: 外部表によって読取りまたは書込みが行われるファイルのデフォルトの位置を指定します。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。詳細は、12-3 ページの「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。詳細は、12-3 ページの「アクセス・パラメータ」を参照してください。
- LOCATION: 外部表の位置を指定します。位置はディレクトリ・オブジェクトおよびファイル名のリストとして指定されます。ディレクトリ・オブジェクトが指定されない場合は、ファイルの位置として、デフォルトのディレクトリ・オブジェクトが使用されます。

次に、各属性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.
```

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。アクセス・パラメータはオプションです。

アクセス・パラメータ

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。

注意： 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。

参照：

- 第 13 章「ORACLE_LOADER アクセス・ドライバ」
- 第 14 章「ORACLE_DATAPUMP アクセス・ドライバ」

データ・ファイルおよび出力ファイルの位置

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。これは、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。

- サーバーでは、アクセス・ドライバによってロードされるファイルにアクセスする必要があります。
- 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。

アクセス・ドライバでは、ファイルに完全な詳細を指定できません。これは、ユーザーがアクセスできないファイルに、サーバーがアクセスする場合があり、ユーザーがデータを読み込むことができると、セキュリティに影響するためです。同様に、通常、ユーザーが削除権限を持たないファイルを上書きする可能性もあります。

かわりに、ファイルの読取り元および書込み元の位置として、ディレクトリ・オブジェクトを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';
```

ディレクトリ・オブジェクトは、DBA または CREATE ANY DIRECTORY 権限を持つすべてのユーザーが作成できます。

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;
```

ディレクトリ・オブジェクトの名前は、CREATE TABLE...ORGANIZATION EXTERNAL 文の次の位置に示すことができます。

- DEFAULT DIRECTORY 句。この句は、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。
- 外部表のすべてのデータ・ファイルをリストする LOCATION 句。ファイル名は、*directory:file* という書式です。*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。

- 出力ファイルに名前を付ける ACCESS PARAMETERS 句。ファイル名は、*directory:file* という書式です。*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。アクセス・パラメータの構文で、特定の出力ファイルを作成しないように指定できます。この構文が有効なのは、出力ファイルを必要としない場合、またはいずれのディレクトリ・オブジェクトにも書込みアクセスをしない場合です。

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・オブジェクトへの READ 権限または WRITE 権限は、Oracle Database によるファイルの読取りまたは書込みのみを意味します。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。

例 : ORACLE_LOADER を使用した外部表の作成およびロード

この項の手順では、ORACLE_LOADER アクセス・ドライバを使用した外部表の作成およびロードの例を示します。emp という従来の表と emp_load という外部表が定義されます。外部データは次に内部表にロードされます。

1. .dat ファイルが次のとおりであるとします。

```
56november, 15, 1980  baker            mary       alice     09/01/2004
87december, 20, 1970  roper            lisa       marie     01/01/1999
```

2. データ・ソースを含むデフォルトのディレクトリを設定して、それに対するアクセス権限を付与するには、次の SQL 文を実行します。

```
CREATE DIRECTORY ext_tab_dir AS '/usr/apps/datafiles';
GRANT READ ON DIRECTORY ext_tab_dir TO SCOTT;
```

3. emp という名前の従来の表を作成します。

```
CREATE TABLE emp (emp_no CHAR(6), last_name CHAR(25), first_name CHAR(20), middle_
initial CHAR(1), hire_date DATE, dob DATE);
```

4. emp_load という名前の外部表を作成します。

```
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.
```

5. emp_load 外部表から emp 表へデータをロードします。

```
SQL> INSERT INTO emp (emp_no,
  2                   first_name,
  3                   middle_initial,
  4                   last_name,
  5                   hire_date,
  6                   dob)
  7  (SELECT employee_number,
  8          employee_first_name,
  9          substr(employee_middle_name, 1, 1),
 10          employee_last_name,
 11          employee_hire_date,
 12          to_date(employee_dob,'month, dd, yyyy')
 13  FROM emp_load);

2 rows created.
```

6. .dat ファイル内の情報が emp 表にロードされたことを確認するために、次の選択操作を行います。

```
SQL> SELECT * FROM emp;

EMP_NO LAST_NAME                 FIRST_NAME           M HIRE_DATE DOB
------ ------------------------- -------------------- - --------- ---------
56     baker                     mary                 a 01-SEP-04 15-NOV-80
87     roper                     lisa                 m 01-JAN-99 20-DEC-70

2 rows selected.
```

この例に関する注意事項

- データ・ファイルの employee_number フィールドは、外部表の employee_number フィールドの文字列に変換されます。
- データ・ファイルには、表のいずれのフィールドにもロードされない employee_dob フィールドが含まれています。
- 外部表の employee_middle_name 列で使用されている substr 関数によって、emp 表の middle_initial の値が生成されます。
- info.dat 内の employee_hire_date の文字列は、外部表の定義で指定されている書式マスクを使用して、外部表へのアクセス時に DATE データ型に自動的に変換されます。
- employee_hire_date とは異なり、employee_dob に対する DATE データ型への変換は、SELECT 時に行われます。この変換は、外部表の定義には含まれていません。

> **参照：** 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 言語リファレンス』を参照してください。

外部表を使用したデータのロードおよびアンロード

外部表のコンテキストではデータのロードは、外部表からデータを読み込み、データベース内の表にロードすることを意味します。データのアンロードは、データベース内のデータを読み込み、外部表に挿入することを意味します。

> **注意：** データは、ORACLE_DATAPUMP アクセス・ドライバを使用してのみアンロードできます。

データのロード

データがロードされた場合、データ・ストリームは LOCATION 句および DEFAULT DIRECTORY 句で指定されたファイルから読み込まれます。INSERT 文では、外部データ・ソースからデータが処理される Oracle SQL エンジンへのデータの流れが発生します。外部ソースからのデータはアクセス・ドライバにより解析され、外部表インタフェースに提供される際に、外部表現から Oracle 内部データ型に変換されます。

ORACLE_DATAPUMP アクセス・ドライバを使用したデータのアンロード

データをアンロードするには、ORACLE_DATAPUMP アクセス・ドライバを使用します。アンロードされるデータ・ストリームには独自の形式が使用され、アンロードされるすべての行に対するすべての列データが含まれます。

また、アンロード処理では、データ・ストリームの内容を記述するメタデータ・ストリームが作成されます。メタデータ・ストリーム内の情報はデータ・ストリームのロードに必要です。そのため、メタデータ・ストリームはデータ・ファイルに書き込まれ、データ・ストリームの前に置かれます。

列オブジェクトの処理

SQL 文を介して外部表にアクセスすると、外部表のフィールドは、通常の表の他のフィールドと同様に使用できます。特に、SQL の組込み関数、PL/SQL ファンクションまたは Java ファンクションの引数として使用できます。これによって、外部ソースのデータを操作できます。

外部表に列オブジェクトは含まれませんが、コンストラクタ・ファンクションを使用して外部表の属性から列オブジェクトを作成できます。たとえば、データベースの表が次のように定義されているとします。

```
CREATE TYPE student_type AS object (
student_no CHAR(5),
name CHAR(20))
/

CREATE TABLE roster (
  student student_type,
  grade CHAR(2));
```

また、次のように定義された外部表があるとします。

```
CREATE TABLE roster_data (
  student_no CHAR(5),
  name CHAR(20),
  grade CHAR(2))
  ORGANIZATION EXTERNAL (TYPE ORACLE_LOADER DEFAULT DIRECTORY ext_tab_dir
                         ACCESS PARAMETERS (FIELDS TERMINATED BY ',')
                         LOCATION ('info.dat'));
```

表 roster を roster_data からロードするには、次のとおり指定します。

```
INSERT INTO roster (student, grade)
  (SELECT student_type(student_no, name), grade FROM roster_data);
```

外部表の使用時のデータ型の変換

データを外部表から、または外部表に移動すると、次の 3 つの場所にある同一の列が、異なるデータ型を持つ可能性があります。

- データベース : データが外部表にアンロードされる際のソースであり、データが外部表からロードされるときのロード先です。
- 外部表 : データが外部表にアンロードされると、外部表内の列のデータ型と一致するよう、データベースからのデータは必要に応じて変換されます。また、SQL 演算子をソース・データに適用して、データを外部表に移動する前にデータ型を変更できます。同様に、外部表からデータベースにロードすると、外部表からのデータはデータベース内の列のデータ型と一致するよう自動的に変換されます。さらに、外部表から選択している SQL 文内の SQL 演算子を使用して、他の変換も実行できます。より高いパフォーマンスを得るには、外部表のデータ型を、データベースのデータ型と一致させます。
- データ・ファイル : データを外部表にアンロードすると、データ・ファイルのフィールドのデータ型は外部表のフィールドのデータ型と完全に一致します。ただし、外部表からデータをロードすると、データ・ファイルのデータ型が外部表のデータ型と一致しない場合があります。その場合、データ・ファイルからのデータは、外部表のデータ型と一致するよう変換されます。列の変換時にエラーが発生した場合、その列を含むレコードはロードされません。より高いパフォーマンスを得るには、データ・ファイルのデータ型を、外部表のデータ型と一致させます。

データ・ファイルと外部表の間で変換エラーが発生すると、エラーの発生した行は無視されます。変換エラーおよび制約違反を含む、外部表とデータベース内の列でエラーが発生すると、操作全体が正常に完了されないまま終了します。

データが外部表にアンロードされる際に、ソース表内の列のデータ型が外部表の列のデータ型と一致していない場合、データが変換されます。変換エラーが発生した場合、その時点までに処理されたすべての行がデータ・ファイルに含まれず、データ・ファイルの読取りができなくなる場合があります。変換エラーによる操作の異常終了を回避するには、外部表の列のデータ型とデータベースの列のデータ型を一致させます。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。たとえば、ソース表に LONG 列がある場合、外部表の対応する列は CLOB である必要があり、外部表を移入するために使用した SELECT 副問合せは、列をロードするために TO_LOB 演算子を使用する必要があります。次に例を示します。

```
CREATE TABLE LONG_TAB_XT (LONG_COL CLOB) ORGANIZATION EXTERNAL...SELECT TO_LOB(LONG_
COL) FROM LONG_TAB;
```

外部表へのパラレル・アクセス

データ・ファイルでのパラレル処理を外部表でサポートするには、外部表の作成時に PARALLEL 句を使用します。アクセス・ドライバによって、パラレル・アクセス方法がわずかに異なります。

ORACLE_LOADER を使用したパラレル・アクセス

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・キャラクタ・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。

ORACLE_DATAPUMP を使用したパラレル・アクセス

データのアンロードに ORACLE_DATAPUMP アクセス・ドライバを使用する場合、PARALLEL 句または PARALLEL ヒントが指定され、外部表に複数の位置が指定されていると、データはパラレルでアンロードされます。

パラレル処理では、それぞれ固有のファイルに書込みを行います。したがって、LOCATION 句には、並列度と同じファイル数を指定する必要があります。指定した並列度よりファイル数が少ない場合、並列度は指定したファイル数に制限されます。指定した並列度よりファイル数が多い場合、超過したファイルは使用されません。

データのアンロードに加え、ORACLE_DATAPUMP アクセス・ドライバでは、データのロードも実行できます。パラレル処理では、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。これは、ORACLE_DATAPUMP アクセス・ドライバがデータをアンロードする場合、新しいデータ・チャンクの最初のダンプ・ファイルにオフセットを定期的に記録し、アンロードが完了した際に、ファイルにその情報を書き込むためです。非パラレル・ロードでは、一度にファイルにアクセスできる処理は 1 つのみのため、ファイルのオフセットは無視されます。パラレル・ロードでは、パラレル処理間でファイル・オフセットが分散され、ファイルまたはファイル・セットで同時に複数の処理が行われます。

外部表使用時のパフォーマンスのヒント

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。ORACLE_LOADER アクセス・ドライバでは、バッファ・サイズの指定にアクセス・パラメータの READSIZE 句を使用できます。

ORACLE_LOADER アクセス・ドライバに固有のパフォーマンスのヒント

この項では、ORACLE_LOADER アクセス・ドライバに固有のパフォーマンスに関する情報について説明します。

パフォーマンスは、日付キャッシュ機能を使用して向上させることができる場合があります。日付キャッシュを使用して、ロード中に予測される一意の日付の数を指定する、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と、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、13-9 ページの「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・キャラクタ・セットは、最も速く処理される。
- 固定幅キャラクタ・セットは、可変幅キャラクタ・セットより速く処理される。
- 可変幅キャラクタ・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。
- キャラクタ・セットを変換するより、データ・ファイルのキャラクタ・セットをデータベースのキャラクタ・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できる。
- 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。

参照： 「外部表および SQL*Loader の選択」（6-10 ページ）

外部表の制限事項

この項では、外部表機能で行われない処理および外部表処理上の制限事項について説明します。

- 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。
- 外部表には、データベースに格納されているデータは記述されません。
- 外部表には、外部ソースでのデータの格納方法は記述されません。これは、アクセス・パラメータの機能です。
- 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。予約語を識別子として使用する場合、二重引用符で囲む必要があります。

参照：

- 「ORACLE_LOADER アクセス・ドライバの予約語」（13-27 ページ）
- 「ORACLE_DATAPUMP アクセス・ドライバの予約語」（14-15 ページ）

ORACLE_DATAPUMP アクセス・ドライバに固有の制限

ORACLE_DATAPUMP アクセス・ドライバには、前述の制限の他に次の制限があります。

- 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・キャラクタ・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が、必ずしもバイト順序マークがデータ・ファイル内に存在するということ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。
- 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。詳細は、12-11 ページの「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。

SQL*Loader と外部表との処理内容の違い

この項では、外部表を使用したデータのロード方法（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方法の重要な違いについて説明します。ここで示す情報は、ORACLE_DATAPUMP アクセス・ドライバには適用されません。

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。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。

構文およびデータ型

次の操作は、外部表ロードではサポートされていません。

- 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

BOM

SQL*Loader では、プライマリ・データ・ファイルに Unicode キャラクタ・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。

デフォルトのキャラクタ・セット、日付マスク、小数点区切り

データ・ファイルのフィールドでは、クライアントの NLS 環境変数によって、デフォルトのキャラクタ・セット、日付マスクおよび小数点区切りが決定されます。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、デフォルトのキャラクタ・セット、日付マスクおよび小数点区切りが決定されます。

バックスラッシュ・エスケープ文字の使用

SQL*Loader では、次のようにバックスラッシュ（\）エスケープ文字を使用して、一重引用符を一重引用符として使用できます。

```
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。解決策としては、次のように分離文字列に二重引用符を使用します。

```
TERMINATED BY ',' ENCLOSED BY "'"
```

13

ORACLE_LOADER アクセス・ドライバ

この章では、デフォルトの外部表アクセス・ドライバである ORACLE_LOADER のアクセス・パラメータについて説明します。アクセス・パラメータは、外部表の作成時に指定します。

この章で説明する情報を使用するには、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およびレコード形式（キャラクタ・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。

この章の内容は、次のとおりです。

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。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。

参照：「EXTERNAL_TABLE」（7-5 ページ）

注意：

- 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、その説明を参照してください。
- この章では、外部表の場合の CREATE TABLE...ORGANIZATION EXTERNAL 文の例をデータ・ファイルの内容のサンプルとともに示します。これらの内容は、CREATE TABLE 文の一部ではなく、完全な例を示します。
- 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。予約語を識別子として使用する場合、二重引用符で囲む必要があります。詳細は、13-27 ページの「ORACLE_LOADER アクセス・ドライバの予約語」を参照してください。

---

access_parameters 句

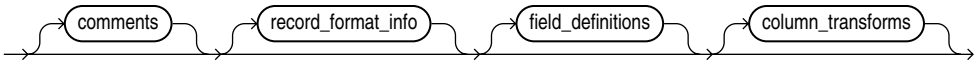
access_parameters 句には、コメント、レコード形式およびフィールド形式の情報が含まれています。

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。

前述のとおり、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。

access_parameters 句の構文は次のとおりです。



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、次の例のように、アクセス・パラメータより前に位置する必要があります。

```
--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）、データのキャラクタ・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、13-2 ページの「record_format_info 句」を参照してください。

field_definitions

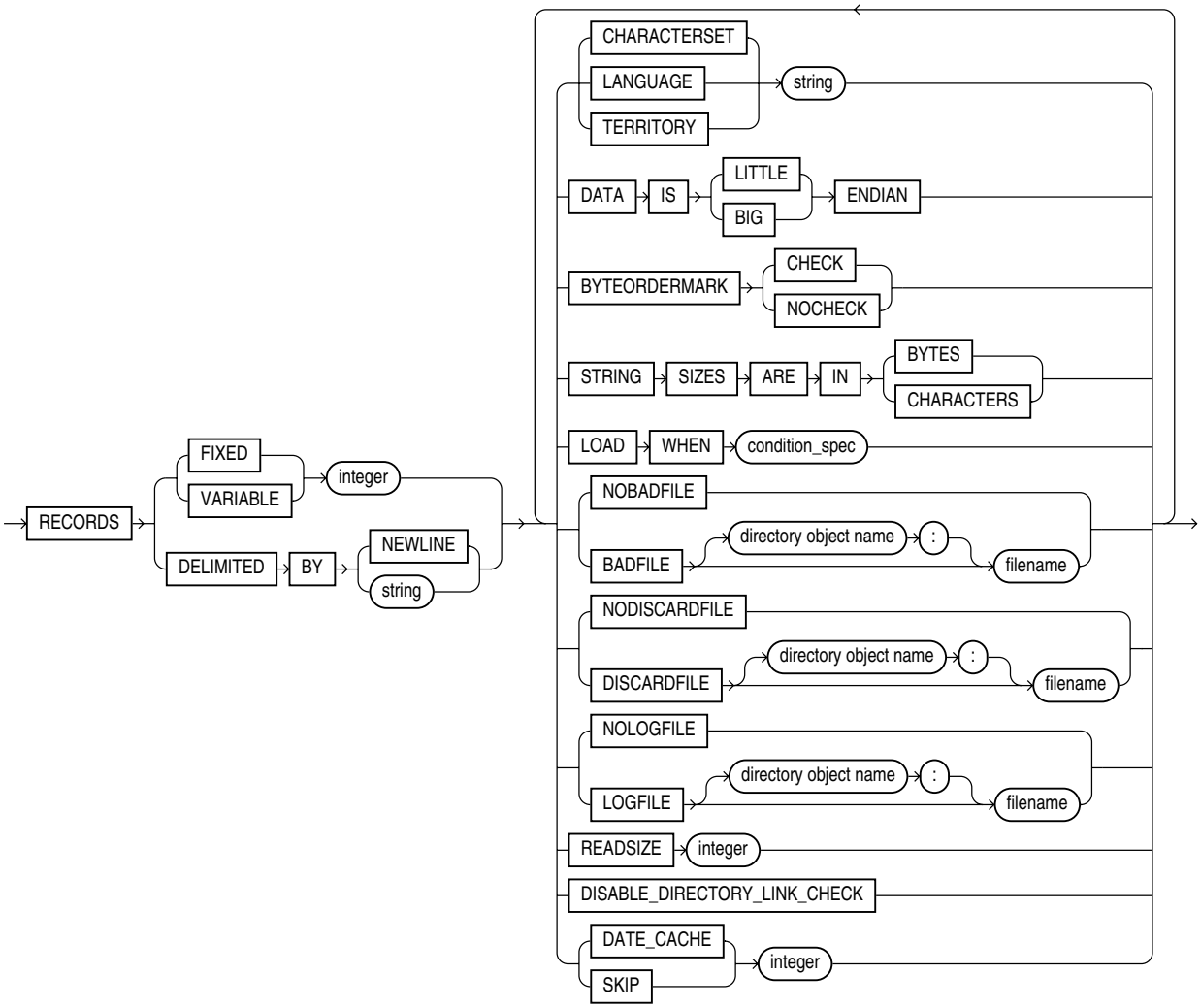
field_definitions 句を使用して、データ・ファイルのフィールドを指定します。データ・ファイルのフィールドが外部表の列と同じ名前の場合、フィールドのデータはその列に使用されます。構文の詳細は、13-11 ページの「field_definitions 句」を参照してください。

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、13-25 ページの「column_transforms 句」を参照してください。

record_format_info 句

record_format_info 句には、レコード（形式など）、データのキャラクタ・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。record_format_info 句はオプションです。句を指定しない場合、デフォルトの値は RECORDS DELIMITED BY NEWLINE です。record_format_info 句の構文は次のとおりです。



FIXED

FIXED 句を使用して、すべてのレコードをバイト単位の固定長として識別します。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。

次に、FIXED レコードの使用例を示します。データ・ファイルの各レコードの終わりに 1 バイトの改行文字があるとします。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 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
```

VARIABLE

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。

次に、VARIABLE レコードの使用例を示します。データ・ファイルの各レコードの終わりに 1 バイトの改行文字があるとします。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 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,
```

DELIMITED BY

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 を指定する場合、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 プラットフォームでは、終了文字は「\n」です。Windows NT では、終了文字は「\r\n」です。

DELIMITED BY *string* を指定する場合、*string* は、テキストまたは一連の 16 進数のいずれかになります。16 進数の場合は、OX または X で開始して引用符で囲みます。テキストの場合は、データ・ファイルのキャラクタ・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。詳細は、13-9 ページの「string」を参照してください。

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・パラメータのキャラクタ・セットがデータ・ファイルのキャラクタ・セットとは異なる場合
- デリミタ文字列中にデータ・ファイルのキャラクタ・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 進文字列ではキャラクタ・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。

注意： 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。

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
```

CHARACTERSET

CHARACTERSET *string* 句を使用して、データ・ファイルのキャラクタ・セットを識別します。キャラクタ・セットを指定しない場合、データベースのデフォルトのキャラクタ・セットが使用されます。詳細は、13-9 ページの「string」を参照してください。

注意： 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キャラクタ・セットに影響しません。

参照： Oracle でサポートされるキャラクタ・セットのリストは、『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。

LANGUAGE

LANGUAGE 句を使用すると、データのロケール依存情報を得るための言語名（FRENCH など）を指定できます。次に、言語名から得られる情報の例を示します。

- 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
- SQL 句 ORDER BY が指定されたときの、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）
- 肯定応答および否定応答の文字列（YES、NO など）

参照： 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。

TERRITORIES

TERRITORIES 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。

参照： 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。

DATA IS...ENDIAN

DATA IS...ENDIAN 句を使用して、バイト順序がデータ・ファイルを生成したプラットフォームによって異なるデータのエンディアンを指定します。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 キャラクタ・セットの文字データ型
- RECORDS DELIMITED BY *string* によって指定する UTF16 キャラクタ・セット文字列

リトル・エンディアン・データを生成する一般的なプラットフォームは Windows NT です。ビッグ・エンディアン・プラットフォームには、Sun Solaris および IBM MVS があります。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ります。このマークは、DATA IS...ENDIAN 句に優先します。

BYTEORDERMARK（CHECK | NOCHECK）

BYTEORDERMARK 句を使用して、データ・ファイルにバイト順序マーク（BOM）があるかを確認するかどうかを指定しますこの句は、キャラクタ・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 が存在するかどうかが確認されます。これは、Unicode キャラクタ・セットのデータ・ファイルのデフォルトの動作です。

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。
- BYTEORDERMARK 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。

参照： 「バイト順序」（9-29 ページ）

STRING SIZES ARE IN

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・キャラクタ・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。

LOAD WHEN

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。これは、ファイル中のロードできないレコードを、エラーなしで現行のレコード定義に解析できない場合に有効です。

詳細は、13-9 ページの「condition_spec」を参照してください。

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)
```

BADFILE | NOBADFILE

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は外部表の列のデータ型に変換できないため、不良ファイルにレコードが書き込まれます。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。

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 を指定する場合は、ファイル名を指定する必要があります。指定しない場合は、エラーが返されます。

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、拡張子 .bad が付けられます。

詳細は、13-10 ページの「[directory object name:] filename」を参照してください。

DISCARDFILE | NODISCARDFILE

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 を指定する場合は、ファイル名を指定する必要があります。指定しない場合は、エラーが返されます。

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、拡張子 .dsc が付けられます。

詳細は、13-10 ページの「[directory object name:] filename」を参照してください。

LOG FILE | NOLOGFILE

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 を使用してログ・ファイルの作成を回避できます。

LOGFILE を指定する場合は、ファイル名を指定する必要があります。指定しない場合は、エラーが返されます。

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、拡張子 .log が付けられます。

詳細は、13-10 ページの「[directory object name:] filename」を参照してください。

SKIP

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。SKIP は、データにパラレルにアクセスしない場合にのみ指定できます。

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。

DISABLE_DIRECTORY_LINK_CHECK

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。

注意： 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。

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。

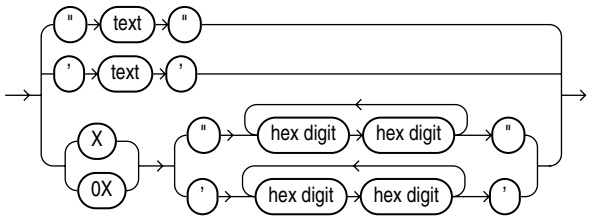
日付キャッシュ機能は、デフォルトで使用可能です。デフォルトの日付キャッシュ・サイズは 1000 要素です。デフォルトのサイズを使用し、1000 を超える一意の入力値をロードすると、日付キャッシュ機能は、この表に対して自動的に使用禁止となります。ただし、デフォルトを変更して 0 以外の日付キャッシュ・サイズを指定し、キャッシュ量がこのサイズを超えた場合、キャッシュは使用禁止になりません。

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。

参照 : 「日付キャッシュの値の指定」(11-16 ページ)

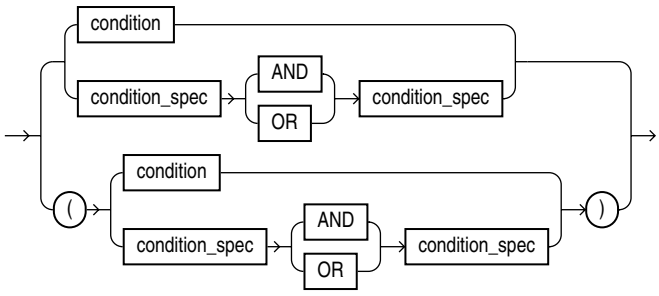
string

string は、引用符で囲まれた一連の文字または 16 進数字です。string が文字列である場合、この文字列は、データ・ファイルのキャラクタ・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・ファイルのキャラクタ・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、他のキャラクタ・セットの翻訳が実行されないことを意味します。string の構文は次のとおりです。



condition_spec

condition_spec は、真または偽のいずれかに評価される式です。ブール演算子によって結合される 1 つ以上の条件を指定します。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)。カッコを使用して、ブール演算子を評価するデフォルトの順序を変更できます。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
```

参照：「condition」（13-10 ページ）

[directory object name:] filename

この句を使用して、出力ファイル（BADFILE、DISCARDFILE または LOGFILE）の名前を指定します。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。

filename パラメータは、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です。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。UNIX および Windows NT 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 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。

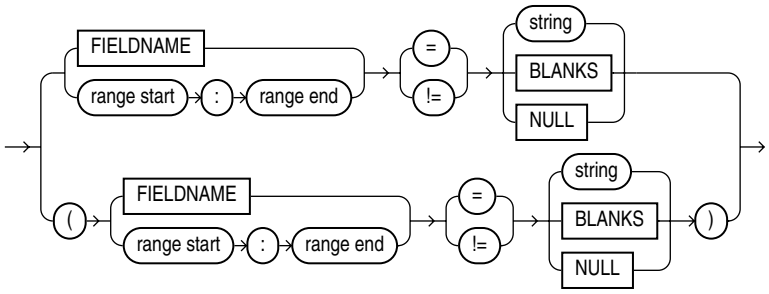
% 文字が検出され、前述の文字以外の文字がその後に続く場合、エラーが返されます。

%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。

BADFILE（または DISCARDFILE か LOGFILE）を指定する場合は、ファイル名を指定する必要があります。指定しない場合は、エラーが返されます。ただし、BADFILE （または DISCARDFILE か LOGFILE）を指定しない場合、アクセス・ドライバでは、表の名前に _%p を付いたものがファイル名として使用されます。ファイルに対して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。

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。文字列は、比較のターゲットとして指定すると、データ・ファイルのキャラクタ・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、データ型変換はフィールド値および文字列のいずれでも実行されません。condition の構文は次のとおりです。



range start : range end

この句を使用してレコードのバイト範囲または文字範囲を記述して条件を指定します。STRING SIZES ARE 句に使用する値で、range がバイトを示すか、文字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、可変幅キャラクタ・セットのデータに対してより固定幅キャラクタ・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。

注意： データ・ファイルには、バイナリ・データ（VARCHAR などの 2 進数を持つデータ型を含む）および文字データ（可変幅キャラクタ・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。

フィールドが NULL の場合、そのフィールドを NULL 以外の値と比較すると、FALSE が返されます。

次に、condition の使用例を示します。

```
empid != BLANKS
10:13 = 0x'00000830'
PRODUCT_COUNT = "MISSING"
```

field_definitions 句

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、レコード内で検索する方法を指定します。

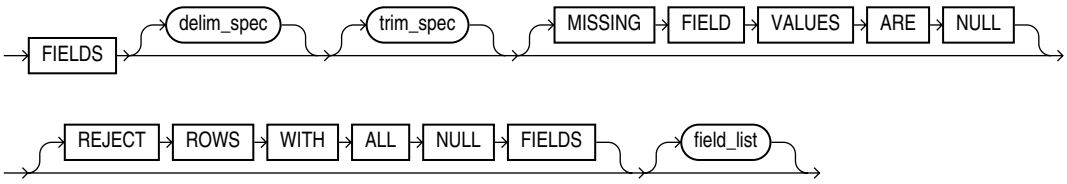
field_definitions 句を指定しない場合は、次のようになります。

- フィールドは、「,」で区切られる。
- フィールドは、文字型である。
- 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、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 LOCATION ('info.dat'));

Alvin,Tolliver,1976
Kenneth,Baer,1963
```

field_definitions 句の構文は次のとおりです。



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、13-12 ページの「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、13-14 ページの「trim_spec」を参照してください。

MISSING FIELD VALUES ARE NULL

MISSING FIELD VALUES ARE NULL は、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、データ値が欠落しているフィールドが NULL に設定されることを示します。構文の詳細は、13-15 ページの「MISSING FIELD VALUES ARE NULL」を参照してください。

REJECT ROWS WITH ALL NULL FIELDS

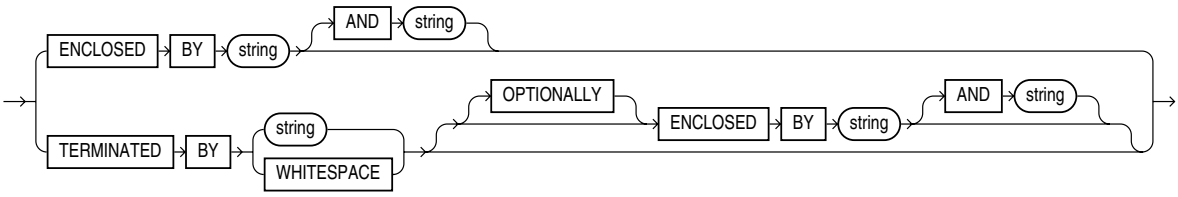
REJECT ROWS WITH ALL NULL FIELDS は、行内で参照されるすべてのフィールドが NULL の場合、その行が外部表にロードされないことを示します。このパラメータを指定しない場合、デフォルト値が使用され、すべてのフィールドが NULL の行が外部表にロードされます。このパラメータの設定は、「reject rows with all null fields」または「rows with all null fields are accepted」としてログ・ファイルに書き込まれます。

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、13-15 ページの「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。構文は次のとおりです。



ENCLOSED BY を指定すると、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、正しく書式化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、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の文字がフィールド部分とみなされます。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。

> **参照：** アクセス・ドライバのデフォルトの切捨て動作の詳細は、表 9-5 を参照してください。LTRIM および RTRIM を使用すると、この動作を変更できます。

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。

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、次のような方法で、データ・ファイル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、固定幅キャラクタ・セットのデータは、可変幅のキャラクタ・セットよりも速く検索できます。

注意： 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

次に、終了デリミタが使用されている外部表の例を示します。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。

```
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
```

例 : 囲みデリミタおよび終了デリミタを含む外部表

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)
```

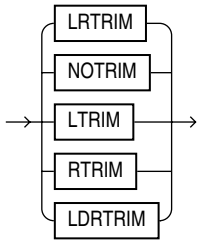
例 : オプションの囲みデリミタを含む外部表

次に、オプションの囲みデリミタを使用する外部表の例を示します。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、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 ','
                                            OPTIONALLY ENCLOSED BY '(' and ')'
                                            LRTRIM)
                         LOCATION ('info.dat'));

Alvin ,   Tolliver , 1976
(Kenneth),  (Baer), (1963)
( Mary ), Dube ,    (1973)
```

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。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、LF、改行など）が含まれます。trim_spec 句の構文は次のとおりです。



フィールドから文字を切り捨てない場合は、NOTRIM を使用します。

フィールドから文字を切り捨てる場合は、LRTRIM、LTRIM および RTRIM を使用します。LRTRIM を使用すると、先頭と後続の空白の両方が切り捨てられます。先頭の空白を切り捨てるには、LTRIM を使用します。後続の空白を切り捨てるには、RTRIM を使用します。

SQL*Loader の切捨て機能との互換性を保つには、LDRTRIM を使用します。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る。

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。

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
```

MISSING FIELD VALUES ARE NULL

MISSING FIELD VALUES ARE NULL は、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、データ値が欠落しているフィールドが NULL に設定されることを示します。MISSING FIELD VALUES ARE NULL を指定せず、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、行は拒否されます。

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
```

field_list

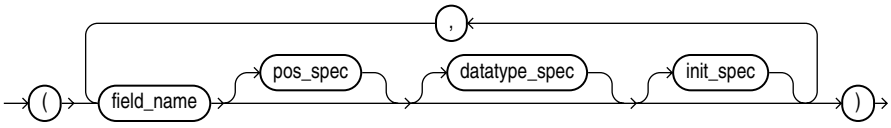
field_list 句を使用して、データ・ファイルのフィールドおよびそのデータ型を識別します。field_list 句では、次のように評価します。

- 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。

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
```

field_list 句の構文は次のとおりです。



field_name

field_name は、データ・ファイルのフィールド名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

pos_spec 句を使用して、レコード内の列の位置を指定します。構文の詳細は、13-16 ページの「pos_spec 句」を参照してください。

datatype_spec

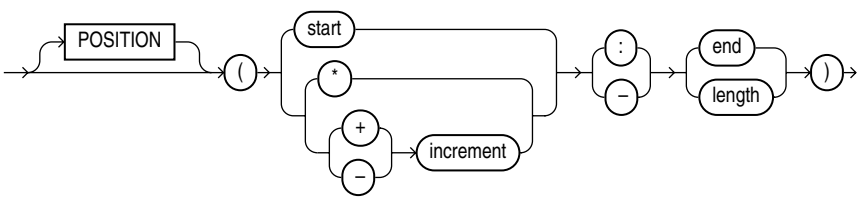
datatype_spec 句を使用して、フィールドのデータ型を指定します。datatype_spec が指定されない場合、アクセス・ドライバは、データ型は CHAR(255) であると想定します。構文の詳細は、13-18 ページの「datatype_spec 句」を参照してください。

init_spec

init_spec 句で、フィールドが NULL になる、またはデフォルトの値を設定されるタイミングを指定します。構文の詳細は、13-24 ページの「init_spec 句」を参照してください。

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。可変幅キャラクタ・セットで文字位置を使用すると、固定幅キャラクタ・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

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または文字数です。前のフィールド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します。

*****

* パラメータで、フィールドが前のフィールドの直後のバイトまたは文字から始まることを指定します。これは、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有効です。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。

increment

increment パラメータを使用して、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のバイト数または固定の文字数で設定します。**-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。**+increment* を使用して、開始位置を現在の位置の後に移動します。

end

end パラメータを使用して、フィールドの終了バイトをレコード内の絶対バイトまたは絶対文字オフセットで指定します。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。

length

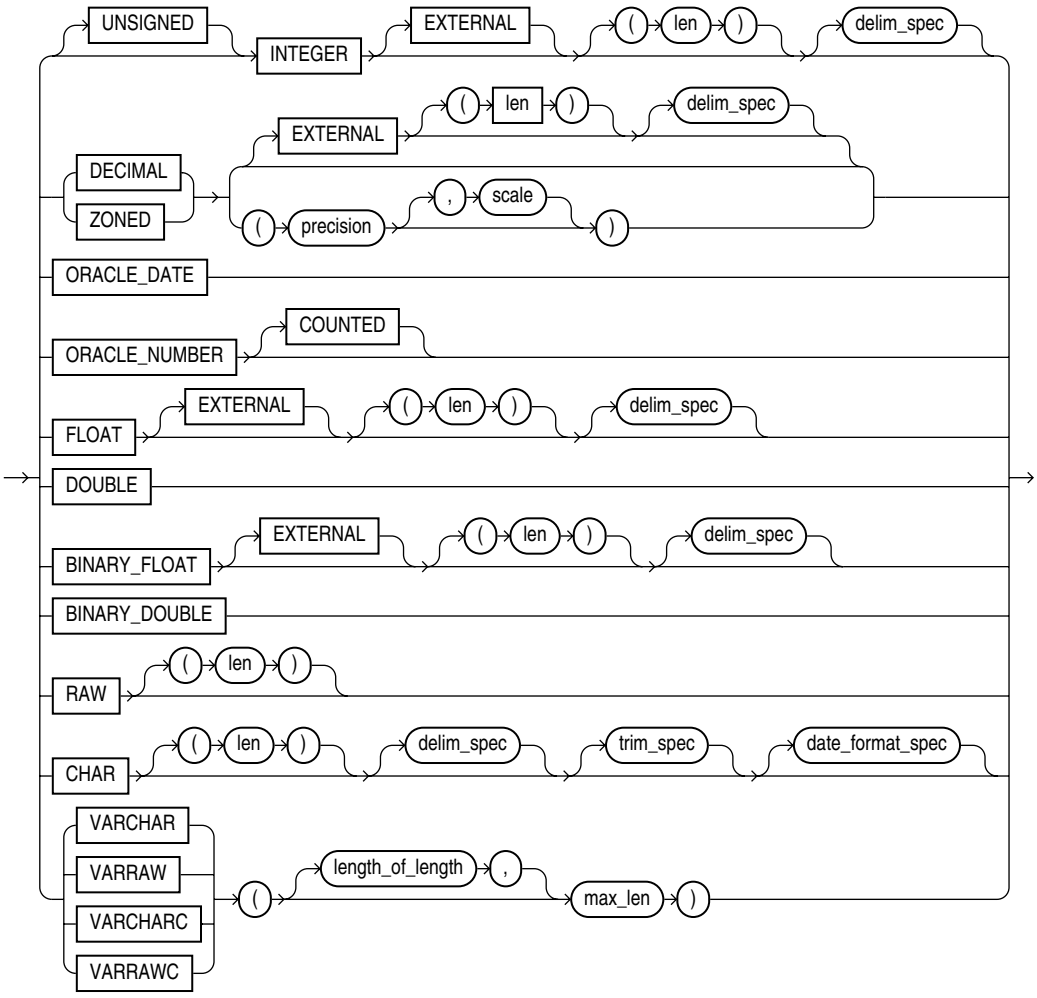
length パラメータで、フィールドの終了位置を開始位置からの固定のバイト数または文字数で指定します。開始位置を * で指定すると、固定長フィールドに有効です。

次に、pos_spec 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。

```
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
     (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
```

datatype_spec 句

データ型がデフォルトと異なる場合、datatype_spec 句を使用して、データ・ファイルのフィールドのデータ型を指定します。フィールドのデータ型は、外部表内の対応する列のデータ型と異なる場合があります。アクセス・ドライバで必要な変更が行われます。
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 句が真の場合、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。

参照：

- NULLIF および DEFAULTIF の詳細は、13-24 ページの「init_spec 句」を参照してください。
- データ型の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

[UNSIGNED] INTEGER [EXTERNAL] [(len)]

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。

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。

使用中のキャラクタ・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。

ORACLE_DATE

ORACLE_DATE は、Oracle バイナリ・データ・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。

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。

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 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。これらの形式は、プラットフォーム上の C プログラムで DOUBLE および FLOAT に対してデフォルトで使用されるデータ型と同じです。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。

注意： 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。

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。

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。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。

BINARY_DOUBLE

BINARY_DOUBLE は、64 ビット倍精度の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 データ型の注意に関する情報を参照してください。

BINARY_FLOAT

BINARY_FLOAT は、32 ビット単精度の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 データ型の注意に関する情報を参照してください。

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。

CHAR

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 SIZESAREIN 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。

date_format_spec 句を使用して、指定された形式の日付または時刻がフィールドに含むことを指定します。

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
```

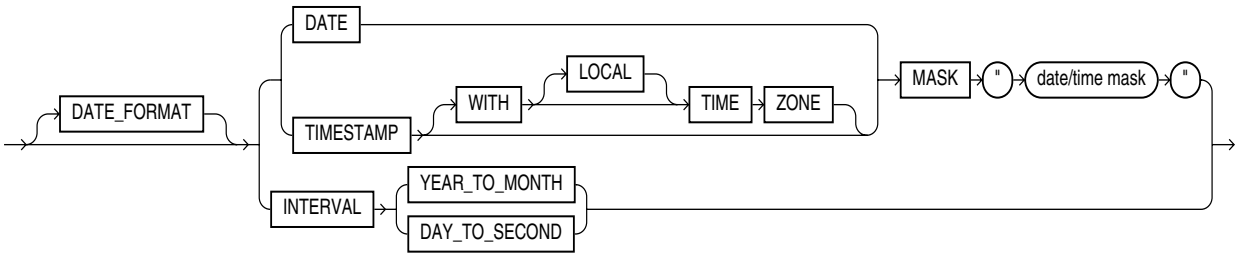
```
 9    (TYPE ORACLE_LOADER
10     DEFAULT DIRECTORY def_dir1
11     ACCESS PARAMETERS
12       (RECORDS DELIMITED BY NEWLINE
13        FIELDS (employee_number      CHAR(2),
14                employee_dob         CHAR(20),
15                employee_last_name   CHAR(18),
16                employee_first_name  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.
```

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。

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 言語リファレンス』を参照してください。

date_format_spec 句の構文は次のとおりです。



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。

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。

- DATE データ型の NLS_DATE_FORMAT
- TIMESTAMP データ型の NLS_TIMESTAMP_FORMAT
- TIMESTAMP WITH TIME ZONE データ型の NLS_TIMESTAMP_TZ_FORMAT

次のこと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。

TIMESTAMP TIMESTAMP 句を使用して、書式化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。

INTERVAL INTERVAL 句を使用して、フィールドに書式化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-99
01-DEC-99 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、01/01/1999）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      01/01/1999 02:44:55
PM01-DEC-99 02.03.00.678573 AM -08:00
```

VARCHAR および VARRAW

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、13-7 ページの「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 句で指定するエンディアンと同じエンディアンです。

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。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。

VARCHARC および VARRAWC

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、13-7 ページの「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・データが後に続くキャラクタ・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。

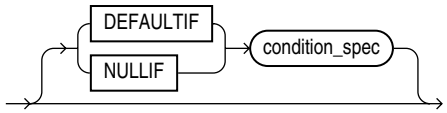
max_len フィールドを使用して、データ・ファイルのフィールドのインスタンスの最大サイズを指定します。VARRAWC の場合は、*max_len* はバイト数です。VARCHARC フィールドの場合は、*max_len* は、STRING SIZES ARE IN 句の設定に応じて、文字数またはバイト数のいずれかになります。

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
```

init_spec 句

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。

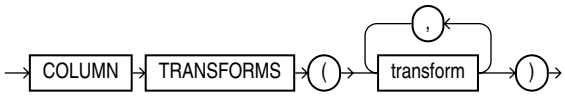


NULLIF 句および DEFAULTIF 句は、フィールドに各 1 回のみ指定できます。これらの句を使用して、次の処理を実行できます。

- NULLIF *condition_spec* を指定し、真と評価された場合、フィールドは NULL に設定されます。
- DEFAULTIF *condition_spec* を指定し、真と評価された場合、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 句および 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 句が偽と評価された場合のみに評価されます。

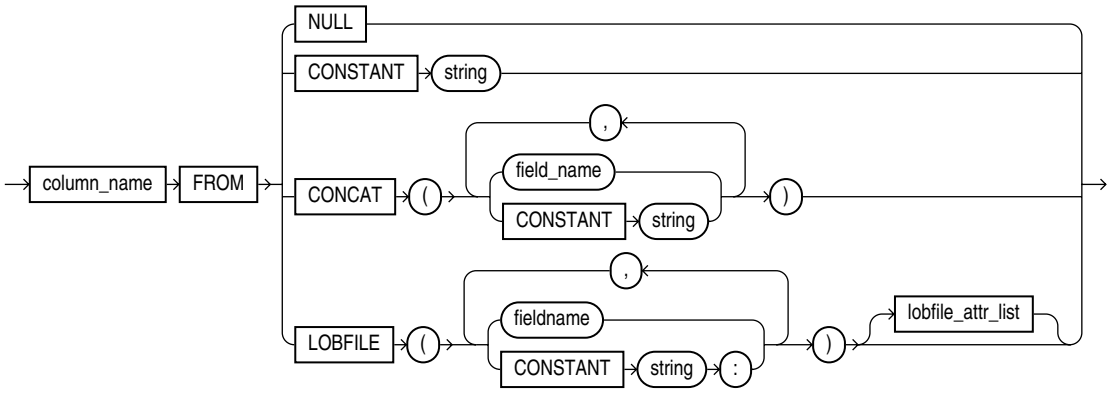
column_transforms 句

オプションの column_transforms 句は、データ・ファイルの列に直接マップしない外部表の列をどのようにロードするか、その記述を変換します。column_transforms 句の構文は次のとおりです。



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。構文は次のようになります。



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します。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）の連結に設定します。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します。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

column_name は、ロードする外部表の列を一意に識別します。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。

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。

CONSTANT

CONSTANT 変換では、レコード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キャラクタ・セットは、データベースのキャラクタ・セットです。

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。

LOBFILE

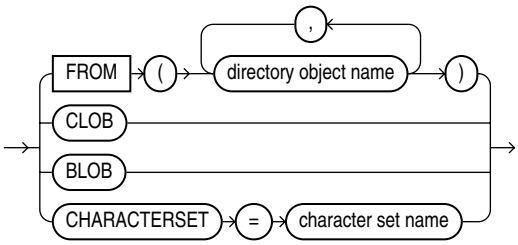
LOBFILE 変換は、外部表の列の値として内容を使用するファイルを識別します。すべての LOBFILE は、*directory object:filename* 形式のオプションのディレクトリ・オブジェクトおよびファイル名で識別されます。LOBFILE 変換には次の規則が適用されます。

- 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。
- 定数文字列が指定された場合、その文字列を使用して表の各行の LOBFILE を検索します。
- フィールド名が指定された場合、データ・ファイル内のそのフィールドの値を使用して LOBFILE を検索します。
- ディレクトリ・オブジェクトまたはファイル名のいずれかに対してフィールド名が指定され、そのフィールドの値が NULL の場合、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。

lobfile_attr_list

lobfile_attr_list は、LOBFILE の追加属性をリストします。構文は次のようになります。



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 RAW データではなく、文字データであることを示します。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キャラクタ・セットに変換する必要があります。

CHARACTERSET 属性は、LOBFILE 内のデータのキャラクタ・セット名を含みます。

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 に対してキャラクタ・セットが指定されない場合、データ・ファイルのキャラクタ・セットであるとみなされます。

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。予約語を識別子として使用する場合、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
- 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

14

ORACLE_DATAPUMP アクセス・ドライバ

この章では、ORACLE_DATAPUMP アクセス・ドライバについて説明します。この章の内容は、次のとおりです。

この章で説明する情報を使用するには、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 の知識が必要です。

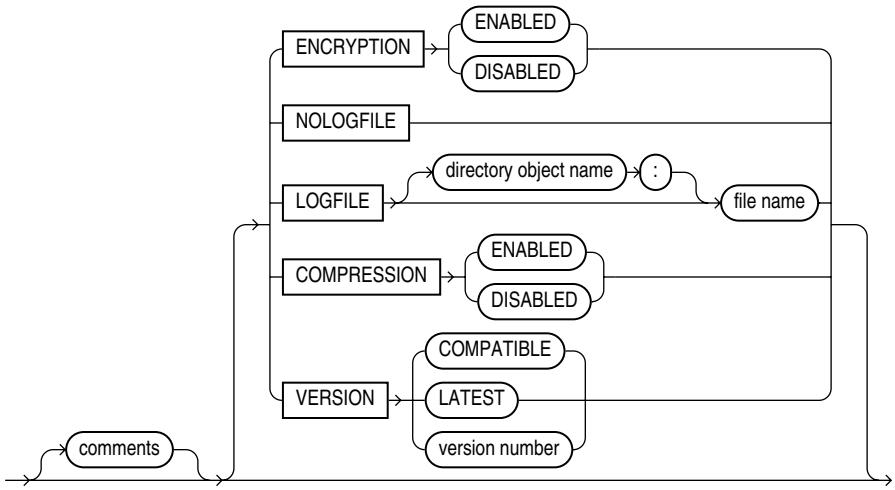
注意：

- 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、その説明を参照してください。
- 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。予約語を識別子として使用する場合、二重引用符で囲む必要があります。詳細は、14-15 ページの「ORACLE_DATAPUMP アクセス・ドライバの予約語」を参照してください。

---

access_parameters 句

外部表の作成時に、access_parameters 句内の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。



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、次の例のように、アクセス・パラメータより前に位置する必要があります。

```
--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。

COMPRESSION

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圧縮するかどうかを指定します。

構文および説明

```
COMPRESSION=[ENABLED | DISABLED]
```

ENABLED を指定すると、アップロード操作全体でデータが圧縮されます。

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 table deptXTec3
 ORGANIZATION EXTERNAL (TYPE ORACLE_DATAPUMP DEFAULT DIRECTORY def_dir1
 ACCESS PARAMETERS (COMPRESSION ENABLED) LOCATION ('dept.dmp'));
```

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'));
```

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。パラレル・ロードまたはパラレル・アンロード時に一意のファイル名を作成するためのワイルドカードの使用の詳細は、14-4 ページの「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'));
```

LOGFILE のファイル名

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。

VERSION 句

VERSION 句は、ダンプ・ファイルを読み込む Oracle Database の最小バージョンを指定するために使用します。バージョン 10.2 を指定すると、10.2 と 11.1 の両方のデータベースでダンプ・ファイルを読み込むことができます。バージョン 11.1 を指定すると、11.1 のデータベースのみでダンプ・ファイルを読み込むことができます。

デフォルト値は COMPATIBLE です。

SQL ENCRYPT 句の使用による影響

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・ファイルの暗号化列とダンプ・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・ファイルのバージョンは 10.2 以上にする必要があります。それ以外の場合は、エラーが返されます。

> **参照：** ENCRYPT 句を CREATE TABLE 文で使用する方法の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

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・ドライバを使用してファイルにデータを移入できます。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。ファイルにデータが移入された後は、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・ファイルとしてファイルを使用できます。

次の手順ではサンプル・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. 手順 10 および手順 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 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

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 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。ダンプ・ファイル内に情報が存在するため、複数のパラレル I/O サーバー・プロセスで、同じファイル内の異なる部分を読み取ることができます。そのため、存在するダンプ・ファイルが 1 つのみの場合でも、並列度を上げてファイルの読取り速度を上げることができます。

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。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、キャラクタ・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
```

サポートされるデータ型

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、データベースのキャラクタ・セットとデータベースの各国語キャラクタ・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
- 日付 / 時間
- 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
```

参照： 「サポートされないデータ型」（14-10 ページ）

サポートされないデータ型

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。文字データ型（LONG 以外）、RAW データ型、すべての数値データ型、すべての日付、タイムスタンプおよび期間データ型をサポートしています。

この項では、ORACLE_DATAPUMP アクセス・ドライバを使用して、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する方法を説明します。サポートされないデータ型の一部を次に示します。

- 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。

次の手順では、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
```

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。次の手順は、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
```

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。次の手順では、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.
```

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。

次の手順は、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
```

ORACLE_DATAPUMP 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。予約語を識別子として使用する場合、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

第 IV 部

その他のユーティリティ

第IV部には、次の章が含まれます。

第 15 章「ADRCI: ADR コマンド・インタプリタ」

この章では、自動診断リポジトリ・コマンド・インタプリタ（ADRCI）について説明します。これは、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。

第 16 章「DBVERIFY: オフライン・データベース検査ユーティリティ」

この章では、オフライン・データベース検査ユーティリティである DBVERIFY の使用方法について説明します。

第 17 章「DBNEWID ユーティリティ」

この章では、DBNEWID ユーティリティを使用してデータベースの名前または ID（あるいはその両方）を変更する方法について説明します。

第 18 章「REDO ログ・ファイル分析のための LogMiner の使用」

この章では、SQL インタフェース経由での REDO ログの問合せが可能な Oracle LogMiner ユーティリティについて説明します。

第 19 章「メタデータ API の使用」

この章では、データベース・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が可能なメタデータ API について説明します。

第 20 章「オリジナルのエクスポートおよびインポート」

この章では、オリジナルのエクスポートおよびインポート・ユーティリティについて説明します。

第 21 章「Enterprise Manager Configuration Assistant（EMCA）」

この章では、Enterprise Manager Configuration Assistant（EMCA）コマンドライン・インタフェースを使用して Database Control を構成する方法について説明します。

15

ADRCI: ADR コマンド・インタプリタ

ADR コマンド・インタプリタ（ADRCI）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。

この章の内容は、次のとおりです。

> **参照：** 診断データの管理の詳細は、『Oracle Database 管理者ガイド』を参照してください。

ADR コマンド・インタプリタ（ADRCI）

ADRCI は、Oracle Database リリース 11*g* に導入された故障診断機能インフラストラクチャに使用されるコマンドライン・ツールです。ADRCI では、次の操作を実行できます。

- 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。

ADRCI には豊富なコマンド・セットが用意されており、対話方式モードでまたはスクリプト内で使用できます。また、ADRCI では、SQL*Plus が SQL および PL/SQL コマンドのスクリプトを実行するのと同じ方法で、ADRCI コマンドのスクリプトを実行できます。

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。

注意： 診断データを管理する場合に最も簡単で推奨される方法は、Oracle Enterprise Manager 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。

サポート・ワークベンチの詳細は、『Oracle Database 管理者ガイド』を参照してください。

定義

次に、ADRCI および Oracle Database の故障診断機能インフラストラクチャに関連して使用される用語の定義を示します。

自動診断リポジトリ（ADR）

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。ADR は、複数のインスタンスおよび複数の製品間で統一されたディレクトリ構造を使用しています。リリース 11*g* 以上では、データベース、自動ストレージ管理（ASM）およびその他のオラクル社の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（「ADR ホーム」を参照）。たとえば、共有記憶域と ASM を使用する Oracle Real Application Clusters 環境では、各データベース・インスタンスおよび各 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 内で追跡されます。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（「問題キー」を参照）。

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。問題が複数回発生する場合は、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。インシデントのフラッド制御の詳細は、『Oracle Database 管理者ガイド』を参照してください。

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修正（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、自動診断リポジトリ（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 を使用して物理パッケージを作成します。

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。この手順は、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。

> **参照：** 相関診断データの詳細は、『Oracle Database 管理者ガイド』を参照してください。

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、ASM を使用する Real Application Clusters 環境では、各データベース・インスタンスと各 ASM インスタンスに ADR ホームがあります。すべての ADR 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。（詳細は、「ADR ベース」を参照してください。）

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。詳細は、15-4 ページの「ホームパス」を参照してください。

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 Real Application Clusters（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。詳細は、『Oracle Database 管理者ガイド』を参照してください。

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 Real Application Clusters 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。

参照：

- ADR およびそのディレクトリの構造および場所の詳細は、『Oracle Database 管理者ガイド』を参照してください。
- 「ADRCI コマンドを使用する前の ADRCI ホームパスの設定」（15-7 ページ）
- 「SET HOMEPATH」（15-37 ページ）
- 「SHOW HOMES」（15-42 ページ）

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
- ヘルプの利用
- バッチ・モードでの ADRCI の使用方法

対話方式モードでの ADRCI の使用方法

対話方式モードでは、個々のコマンドを 1 つずつ入力するように要求されます。入力した各コマンドの後にはその出力が表示されます。

対話方式モードで ADRCI を使用するには、次の手順に従います。

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。

   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され、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. 次のコマンドのいずれかを入力して、ADRCI を終了します。

   ```
   EXIT
   QUIT
   ```

ヘルプの利用

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、次の手順に従います。

1. 対話方式モードで ADRCI を起動します。

   詳細は、15-5 ページの「対話方式モードでの ADRCI の使用方法」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。

   ```
   HELP
   ```

特定の ADRCI コマンドのヘルプを利用するには、次の手順に従います。

1. 対話方式モードで ADRCI を起動します。

   詳細は、15-5 ページの「対話方式モードでの ADRCI の使用方法」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。

   ```
   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、次の手順に従います。

- 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)
----------------------------------------------------------------
```

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、一連の ADRCI コマンドを一度に実行できます。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 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。

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 15-1 バッチ操作用の ADRCI のコマンドライン・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には、次の手順に従います。

- 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 スクリプトを実行するには、次の手順に従います。

- 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、次のようにセミコロン（;）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。

```
SET HOMEPATH diag/rdbms/orcl/orcl; SHOW ALERT -term
```

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します。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 Real Application Clusters（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つ場合は、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。

参照：

- 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」（15-4 ページ）
- 「ADR ホーム」（15-3 ページ）
- 「ホームパス」（15-4 ページ）
- 「SET HOMEPATH」（15-37 ページ）
- 「SHOW HOMES」（15-42 ページ）

アラート・ログの表示

リリース 11*g* 以上の Oracle Database では、アラート・ログが XML 形式のファイルおよびテキスト・ファイルの両方で書き込まれます。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 タグを削除した 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 EDITOR 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、次の手順に従います。

1. 対話方式モードで ADRCI を起動します。

   詳細は、15-4 ページの「ADRCI の起動とヘルプの利用」を参照してください。

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、15-4 ページの「ホームパス」および 15-7 ページの「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 タグを削除したアラート・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
```

参照：

- 「SHOW ALERT」（15-37 ページ）
- 「SET EDITOR」（15-36 ページ）
- 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。

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
```

参照：

- 「SHOW TRACEFILE」（15-48 ページ）
- 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。

インシデントの表示

ADRCI の SHOW INCIDENT コマンドでは、未解決の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。複数のカレント ADR ホームが存在するように ADRCI ホームパスが設定されている場合、レポートにはすべてのカレント ADR ホームからのインシデントが含まれます。

すべての未解決のインシデントに関するレポートを表示するには、次の手順に従います。

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。

   詳細は、15-4 ページの「ADRCI の起動とヘルプの利用」および 15-4 ページの「ホームパス」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。

   ```
   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                   2007-06-18 21:35:49.322161 -07:00
3807              ORA 600 [4137]            2007-06-18 21:35:47.862114 -07:00
3805              ORA 600 [4136]            2007-06-18 21:35:25.012579 -07:00
3804              ORA 1578                  2007-06-18 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。

```
SHOW INCIDENT -MODE DETAIL -P "INCIDENT_ID=1681"
```

これは、インシデント 1681 のみを対象とする詳細なインシデント・レポートを表示します。

参照：「ADRCI コマンド・リファレンス」（15-14 ページ）

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
- インシデント・パッケージの作成

インシデントのパッケージ化

インシデントのパッケージ化プロセスは次の 3 つの手順で構成されています。

手順 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、手順 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。

手順 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、この手順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。

手順 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

次のように指定します。

- *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
```

インシデント・パッケージの作成

次の各項では、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 コマンドについて説明します。

- 論理インシデント・パッケージの作成
- 論理インシデント・パッケージへの診断情報の追加
- 物理インシデント・パッケージの生成

参照： 「インシデントのパッケージ化」（15-11 ページ）

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。

インシデントに基づいてパッケージを作成するには、次の手順に従います。

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。

   詳細は、15-4 ページの「ADRCI の起動とヘルプの利用」および 15-4 ページの「ホームパス」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
```

これは空のパッケージを作成します。IPS ADD INCIDENT または IPS ADD FILE の各コマンドを使用して、生成する前にパッケージに診断データを追加する必要があります。

```
IPS CREATE PACKAGE PROBLEM problem_ID
```

これは、パッケージを作成して、指定した問題 ID（問題 ID は整数）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。

注意： 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、15-28 ページの「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"
```

これは、パッケージを作成し、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。

注意： 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、15-28 ページの「IPS SET CONFIGURATION」を参照してください。

また、ADRCI によって、時間などの基準においてすでに追加されているインシデントと密接に相関付けられている他のインシデントを追加する場合もあります。

問題キーは、一重引用符（'）または（空白や引用符が含まれている場合には）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'
```

これは、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。これは、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、2007 年 7 月 24 日～ 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 '2007-07-24 00:00:00 -07:00' to '2007-07-30 23.59.59 -07:00'
```

参照： 「IPS CREATE PACKAGE」（15-21 ページ）

論理インシデント・パッケージへの診断情報の追加

既存の論理パッケージ（パッケージ）に次の診断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
- ADR 内の名前付きファイル

既存のパッケージにインシデントを追加するには、次の手順に従います。

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。

   詳細は、15-4 ページの「ADRCI の起動とヘルプの利用」および 15-4 ページの「ホームパス」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 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、次の手順に従います。

- 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。

  参照： 「ADRCI コマンド・リファレンス」（15-14 ページ）

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。

物理インシデント・パッケージを生成するには、次の手順に従います。

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。

   詳細は、15-4 ページの「ADRCI の起動とヘルプの利用」および 15-4 ページの「ホームパス」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 GENERATE PACKAGE package_number IN path
   ```

   これは、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。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
   ```

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。

増分物理インシデント・パッケージを生成するには、次の手順に従います。

- 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
  ```

  参照：

  - 「ADRCI コマンド・リファレンス」（15-14 ページ）
  - 「インシデントのパッケージ化」（15-11 ページ）

ADRCI コマンド・リファレンス

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。

表 15-2 では、ADRCI コマンド・セットを一覧表示します。

表 15-2 ADRCI コマンドのリスト

コマンド	説明
CREATE REPORT	指定したレポート・タイプおよび ID のレポートを作成します。
ECHO	入力文字列をエコー処理します。
EXIT	カレントの ADRCI セッションを終了します。
HOST	ADRCI からオペレーティング・システムのコマンドを実行します。
IPS	IPS ユーティリティを起動します。ADRCI 内で使用可能な IPS コマンドについては、表 15-3 を参照してください。
QUIT	カレントの ADRCI セッションを終了します。
RUN	ADRCI スクリプトを実行します。
SET BASE	カレント ADRCI セッションの ADR ベースを設定します。
SET BROWSER	今後使用する目的で確保されているコマンドです。
SET CONTROL	ADR の内容の削除ポリシーを設定します。
SET ECHO	コマンド出力を切り替えます。
SET EDITOR	トレース・ログおよびアラート・ログの内容を表示するデフォルト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。
SET TERMOUT	ターミナル出力を切り替えます。
SHOW ALERT	アラート・ログ・メッセージを表示します。
SHOW BASE	カレント ADR ベースを表示します。
SHOW CONTROL	カレントの削除ポリシーを含む ADR 情報を表示します。
SHOW HM_RUN	状態モニターの実行情報を表示します。
SHOW HOMEPATH	カレントのホームパスを表示します。
SHOW HOMES	カレントの ADR ホームを表示します。
SHOW INCDIR	指定したインシデントに対して作成されたトレース・ファイルを表示します。
SHOW INCIDENT	インシデントのリストを出力します。
SHOW PROBLEM	問題のリストを出力します。
SHOW REPORT	指定したレポート・タイプおよび ID のレポートを表示します。
SHOW TRACEFILE	修飾されたトレース・ファイル名を表示します。
SPOOL	出力をファイルへ送信します。

注意： 特に指定されないかぎり、次に示すコマンドは複数のカレント ADR ホームで動作します。

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。

> **注意：** 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。

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
```

> **注意：** 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、15-7 ページの「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。

ECHO

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

次の例は、文字列 Hello, world! を出力します。

```
echo "Hello, world!"

echo 'Hello, world!'
```

EXIT

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。

HOST

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"
```

IPS

用途
インシデント・パッケージング・サービス（IPS）のコマンド・セットを呼び出します。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。

> **参照：** 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。

IPS のコマンド・セットには次のコマンドが含まれます。

表 15-3 IPS のコマンド・セット

コマンド	説明
IPS ADD	インシデント、問題または問題キーをパッケージに追加します。
IPS ADD FILE	ファイルを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に対する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 へファイルをコピーします。

表 15-3 IPS のコマンド・セット（続き）

コマンド	説明
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。
IPS CREATE PACKAGE	新しい（論理）パッケージを作成します。
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	指定したパッケージの内容の ZIP ファイルをターゲット・ディレクトリに生成します。
IPS GET MANIFEST	パッケージの ZIP ファイルからマニフェストを取得して表示します。
IPS GET METADATA	パッケージの ZIP ファイルからメタデータを抽出して表示します。
IPS PACK	インシデント、問題または問題キーから直接物理パッケージ（ZIP ファイル）を作成します。
IPS REMOVE	既存のパッケージからインシデントを削除します。
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 構成パラメータの値を表示します。
IPS SHOW FILES	パッケージ内のファイルを一覧表示します。
IPS SHOW INCIDENTS	パッケージ内のインシデントを一覧表示します。
IPS UNPACK FILE	パッケージの ZIP ファイルを指定したパスに解凍します。

注意： 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、15-7 ページの「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 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
```

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 inc_id | problem prob_id | problemkey pr_key |
        seconds secs | time start_time to end_time}          package pkg_
id
```

表 15-4 に、IPS ADD の引数を示します。

表 15-4 IPS ADD 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に追加します。
problem *prob_id*	問題 ID が *prob_id* 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。（**注意：**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、15-28 ページの「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。（**注意：**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 *pkg_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、2007 年 5 月 1 日の午前 10 時～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 '2007-05-01 10:00:00.00 -07:00' to '2007-05-01 23:00:00.00 -07:00'
```

IPS ADD FILE

用途

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 *pkg_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。

pkg_id はパッケージ 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
```

> **参照：** <ADR_HOME> ディレクトリ構文の詳細は、15-18 ページの「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 ADD NEW INCIDENTS

用途

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
```

> **注意：** 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。15-28 ページの「IPS SET CONFIGURATION」を参照してください。

IPS COPY IN FILE

用途

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。

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 *pkgid*
[incident *incid*]

外部ファイル *filename*（フルパス名で指定）を ADR にコピーします。ファイルは既存のパッケージ *pkgid* に関連付けますが、オプションでインシデント *incid* にも関連付けることもあります。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
```

参照 :

- <ADR_HOME> 変数については、15-18 ページの「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一覧表示については、15-31 ページの「IPS SHOW FILES」を参照してください。

IPS COPY OUT FILE

用途

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。

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、/trace/mydb1_ora_13579.trc というファイルを ADR ホームからローカル・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
```

参照 :

- <ADR_HOME> ディレクトリ構文の詳細は、15-18 ページの「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一覧表示については、15-31 ページの「IPS SHOW FILES」を参照してください。

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 *inc_id* | problem *prob_id* | 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 15-5 に、IPS CREATE PACKAGE の引数を示します。

表 15-5 IPS CREATE 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に追加します。
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。（**注意：**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、15-28 ページの「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。（**注意：**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。
package *pkg_id*	インシデントを ID が *pkg_id* のパッケージに追加します。

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
```

出力例：

```
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～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 '2007-05-01 10:00:00.00 -07:00' to '2007-05-01 23:00:00.00
-07:00'
```

出力例：

```
Created package 6 based on time range 2007-05-01 10:00:00.00 -07:00 to 2007-05-01
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
```

出力例 :

```
Created package 7 based on problem id 3, correlation level typical
```

注意 : 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、15-28 ページの「IPS SET CONFIGURATION」を参照してください。

参照 : 「インシデント・パッケージの作成」

IPS DELETE PACKAGE

用途

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` *`pkg_id`*

`pkg_id` は削除するパッケージです。

例

```
ips delete package 12
```

IPS FINALIZE

用途

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` *`pkg_id`*

`pkg_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
```

参照 : 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。

IPS GENERATE PACKAGE

用途

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。これがデフォルトの動作となります。

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
```

参照：「物理インシデント・パッケージの生成」（15-14 ページ）

IPS GET MANIFEST

用途

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
```

IPS GET METADATA

用途

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。

> **参照：** 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。

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 inc_id | problem prob_id | 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 15-6 に、IPS PACK の引数を示します。

表 15-6 IPS PACK 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に追加します。
problem *prob_id*	問題 ID が *prob_id* 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。（**注意：**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、15-28 ページの「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。（**注意：**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。

表 15-6 IPS PACK コマンドの引数（続き）

引数	説明
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。
[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
```

> **参照：** 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。

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 pkg_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 15-7 では、IPS REMOVE の引数を示します。

表 15-7 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 *pkg_id*	ID が *pkg_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
```

> **参照：** パッケージのメタデータについては、15-24 ページの「IPS GET MANIFEST」を参照してください。

IPS REMOVE FILE

用途

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 *pkg_id*

file_name はパッケージ *pkg_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 1 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 RECORD
************************************************************

------------------------------------------------------------
FILE INFORMATION:
   FILE_LOCATION                  <ADR_HOME>/trace
   FILE_NAME                      orcl_ora_13579.trc
   LAST_SEQUENCE                  0
   EXCLUDE                        1
.
.
.
```

参照：

- パッケージのメタデータについては、15-24 ページの「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、15-18 ページの「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」（15-31 ページ）

IPS SET CONFIGURATION

用途

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 value
```

parameter_id は変更するパラメータ ID で、*value* は新しい値です。構成パラメータおよびその ID のリストに対しては、「IPS SHOW CONFIGURATION」を使用します。

例

```
ips set configuration 3 10
```

IPS SHOW CONFIGURATION

用途

IPS 構成パラメータおよびその値のリストを表示します。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]
```

IPS SHOW CONFIGURATION は、構成パラメータごとに次の情報を表示します。

- PARAMETER ID
- NAME
- DESCRIPTION
- UNIT（パラメータが使用する単位、日数や時間数など）
- VALUE
- DEFAULT VALUE

オプションで、*parameter_id* 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
```

出力例：

```
IPS CONFIGURATION PARAMETER
**********************************************************
----------------------------------------------------------
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

--------------------------------------------------------
********************************************************
IPS CONFIGURATION PARAMETER
********************************************************
--------------------------------------------------------
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

--------------------------------------------------------
********************************************************
IPS CONFIGURATION PARAMETER
********************************************************
--------------------------------------------------------
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

--------------------------------------------------------
********************************************************
IPS CONFIGURATION PARAMETER
********************************************************
--------------------------------------------------------
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

--------------------------------------------------------
********************************************************
IPS CONFIGURATION PARAMETER
********************************************************
--------------------------------------------------------
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

--------------------------------------------------------
********************************************************
IPS CONFIGURATION PARAMETER
********************************************************
--------------------------------------------------------
```

```
PARAMETER INFORMATION:
   PARAMETER_ID      6
   NAME              DEFAULT_CORRELATION_LEVEL
   DESCRIPTION       Default correlation level for packages
   UNIT              Number
   VALUE             2
   DEFAULT_VALUE     2

------------------------------------------------------------
```

例

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 15-8 に、IPS 構成パラメータの詳細を示します。

表 15-8 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 での相関）: 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 での相関）: 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 での相関）: インシデント・ダンプ、および最初に選択されたインシ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。

参照： 「IPS SET CONFIGURATION」（15-28 ページ）

IPS SHOW FILES

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 *pkg_id*

pkg_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 3 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 3
```

出力例 :

```
************************************************************
FILE RECORD
************************************************************

------------------------------------------------------------
FILE INFORMATION:
   FILE_LOCATION         <ADR_HOME>/incident/incdir_4!/nick/oracle/log/diag/rdbms/
   FILE_NAME             orcl_ora_13579_i4_2.trc
   LAST_SEQUENCE         0
   EXCLUDE               0

------------------------------------------------------------
************************************************************
FILE RECORD
************************************************************

------------------------------------------------------------
FILE INFORMATION:
   FILE_LOCATION         <ADR_HOME>/incident/incdir_4!/nick/oracle/log/diag/rdbms/
   FILE_NAME             orcl_ora_13579_i4.trc
   LAST_SEQUENCE         0
   EXCLUDE               0

------------------------------------------------------------
************************************************************
FILE RECORD
************************************************************

------------------------------------------------------------
FILE INFORMATION:
   FILE_LOCATION         <ADR_HOME>/incident/incdir_4!/nick/oracle/log/diag/rdbms/
   FILE_NAME             orcl_ora_13579_i4_sql_2.trc
   LAST_SEQUENCE         0
   EXCLUDE               0

------------------------------------------------------------
```

IPS SHOW INCIDENTS

用途

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 *pkg_id*

pkg_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 3 のインシデントを表示します。

```
ips show incidents package 3
```

出力例:

```
************************************************************
Main INCIDENTS
************************************************************

------------------------------------------------------------
************************************************************
INCIDENT RECORD
************************************************************

------------------------------------------------------------
INCIDENT INFORMATION:
   INCIDENT_ID                   3827
   PROBLEM_ID                    3
   EXCLUDE                       0

------------------------------------------------------------
************************************************************
Correlated INCIDENTS
************************************************************

------------------------------------------------------------
************************************************************
INCIDENT RECORD
************************************************************

------------------------------------------------------------
INCIDENT INFORMATION:
   INCIDENT_ID                   3633
   PROBLEM_ID                    2
   EXCLUDE                       0

------------------------------------------------------------
************************************************************
INCIDENT RECORD
************************************************************

------------------------------------------------------------
INCIDENT INFORMATION:
   INCIDENT_ID                   3634
   PROBLEM_ID                    3
   EXCLUDE                       0

------------------------------------------------------------
```

IPS UNPACK FILE

用途

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
```

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}]]]
```

表 15-9 に、PURGE のフラグを示します。

表 15-9 PURGE コマンドのフラグ

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}]	削除する診断データの種類（アラート・ログ・メッセージ、インシデント・データ、トレース・ファイル（ダンプを含む）、コア・ファイルまたは状態モニターの実行データとレポート）を指定します。

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
```

> **注意：** 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、15-7 ページの「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。

QUIT

「EXIT」を参照してください。

RUN

用途

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
```

script_name は、実行する ADRCI コマンドを含むファイルです。フルパス名が指定されていないかぎり、ADRCI はカレント・ディレクトリで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が指定されている場合、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 や @ と似ていますが、スクリプト内で使用される場合に、@@ はカレント・ディレクトリではなくスクリプトをコールするパスを使用して *script_name* を検索する点が異なります。

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

```
run my_script

@my_script
```

SET BASE

用途

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 /net/sttttd1/scratch/steve/view_storage/steve_v1/log
```

参照： 「ADR ベース」（15-4 ページ）

SET BROWSER

用途

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。

注意： 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
```

参照：

- レポートの表示の詳細は、15-47 ページの「SHOW REPORT」を参照してください。
- スプールの詳細は、15-49 ページの「SPOOL」を参照してください。

SET CONTROL

用途

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。詳細は、15-40 ページの「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。

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)
```

SET ECHO

用途

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
```

参照： スプールの詳細は、15-49 ページの「SPOOL」を参照してください。

SET EDITOR

用途

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
```

参照： 「SHOW REPORT」（15-47 ページ）

SET HOMEPATH

用途

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 homepath homepath_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。ディレクトリ名の diag は省略可能です。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/aime3/aime3  diag/rdbms/aime3/aime32
```

参照： 「ホームパス」（15-4 ページ）

SET TERMOUT

用途

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。

参照： スプールの詳細は、「SPOOL」を参照してください。

例

```
set termout on
```

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 15-10 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 15-11 に、述語文字列で使用できるフィールドを表示します。
-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 15-11 SHOW ALERT のアラート・フィールド

フィールド	型
ORIGINATING_TIMESTAMP	timestamp
NORMALIZED_TIMESTAMP	timestamp
ORGANIZATION_ID	text(65)
COMPONENT_ID	text(65)
HOST_ID	text(65)
HOST_ADDRESS	text(17)
MESSAGE_TYPE	number
MESSAGE_LEVEL	number
MESSAGE_ID	text(65)
MESSAGE_GROUP	text(65)
CLIENT_ID	text(65)
MODULE_ID	text(65)
PROCESS_ID	text(33)
THREAD_ID	text(65)
USER_ID	text(65)
INSTANCE_ID	text(65)
DETAILED_LOCATION	text(161)
UPSTREAM_COMP_ID	text(101)
DOWNSTREAM_COMP_ID	text(101)

表 15-11 SHOW ALERT のアラート・フィールド（続き）

フィールド	型
EXECUTION_CONTEXT_ID	text(101)
EXECUTION_CONTEXT_SEQUENCE	number
ERROR_INSTANCE_ID	number
ERROR_INSTANCE_SEQUENCE	number
MESSAGE_TEXT	text(2049)
MESSAGE_ARGUMENTS	text(129)
SUPPLEMENTAL_ATTRIBUTES	text(129)
SUPPLEMENTAL_DETAILS	text(129)
PROBLEM_KEY	text(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/rdbms/
2: diag/tnslsnr/sta00339/listener
Q: to quit

Please select option:
1
```

参照： 「SET EDITOR」（15-36 ページ）

SHOW BASE

用途

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、クライアントと 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
```

出力例:

```
ADR base is "/scratch/nick/rdbms/log"
```

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
```

SHOW CONTROL

用途

削除ポリシーを含む、自動診断リポジトリ（ADR）に関する情報を表示します。

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。
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。

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]
```

[*-p 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 15-12 に、使用可能なフィールドのリストを示します。

表 15-12 状態モニターの実行に使用するフィールド

フィールド	型
RUN_ID	number
RUN_NAME	text(31)
CHECK_NAME	text(31)
NAME_ID	number
MODE	number
START_TIME	timestamp
RESUME_TIME	timestamp
END_TIME	timestamp
MODIFIED_TIME	timestamp
TIMEOUT	number
FLAGS	number
STATUS	number
SRC_INCIDENT_ID	number
NUM_INCIDENTS	number
ERR_NUMBER	number
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"
```

> **参照：** 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。

SHOW HOMEPATH

用途

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
```

出力例 :

```
ADR Homes:
diag/diagtool/user_nick/host_3075434791_11
diag/rdbms/db1/db1
diag/rdbms/db2/db2
```

> **参照 :** ホームパスの設定方法については、「SET HOMEPATH」を参照してください。

SHOW HOMES

用途

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
```

出力例 :

```
ADR Homes:
diag/diagtool/user_nick/host_3075434791_11
diag/rdbms/db1/db1
diag/rdbms/db2/db2
```

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* ～ *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
```

出力例:

```
ADR Home = /ade/sfogel_emdb/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
```

出力例:

```
ADR Home = /ade/sfogel_emdb/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 ～ 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
```

出力例:

```
ADR Home = /ade/sfogel_emdb/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
```

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 15-13 に、SHOW INCIDENT のフラグを示します。

表 15-13 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 15-14 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 15-14 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、結果は昇順で示されます。

表 15-14 SHOW INCIDENT のインシデント・フィールド

フィールド	型
INCIDENT_ID	number
PROBLEM_KEY	text(550)
PROBLEM_ID	number
CREATE_TIME	timestamp
CLOSE_TIME	timestamp
STATUS	number
FLAGS	number
FLOOD_CONTROLLED	number（ADRCI によりテキスト・ステータスにデコードされる）
ERROR_FACILITY	text(10)
ERROR_NUMBER	number
ERROR_ARG1	text(64)
ERROR_ARG2	text(64)
ERROR_ARG3	text(64)

表 15-14 SHOW INCIDENT のインシデント・フィールド（続き）

フィールド	型
ERROR_ARG4	text(64)
ERROR_ARG5	text(64)
ERROR_ARG6	text(64)
ERROR_ARG7	text(64)
ERROR_ARG8	text(64)
SIGNALLING_COMPONENT	text(64)
SIGNALLING_SUBCOMPONENT	text(64)
SUSPECT_COMPONENT	text(64)
SUSPECT_SUBCOMPONENT	text(64)
ECID	text(64)
IMPACT	number

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
```

出力例:

```
ADR Home = /ade/sfogel_emdb/oracle/log/diag/rdbms/emdb/emdb:
*******************************************************************************
INCIDENT_ID          PROBLEM_KEY                                       CREATE_TIME
-------------------- ------------------------------------------------- ----------------------------
3808                 ORA 603                                           2007-06-18 21:35:49.322161 -07:00
3807                 ORA 600 [4137]                                    2007-06-18 21:35:47.862114 -07:00
3806                 ORA 603                                           2007-06-18 21:35:26.666485 -07:00
3805                 ORA 600 [4136]                                    2007-06-18 21:35:25.012579 -07:00
3804                 ORA 1578                                          2007-06-18 21:35:08.483156 -07:00
3713                 ORA 600 [4136]                                    2007-06-18 21:35:44.754442 -07:00
3633                 ORA 600 [4136]                                    2007-06-18 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"
```

出力例:

```
ADR Home = /ade/sfogel_emdb/oracle/log/diag/rdbms/emdb/emdb:
*******************************************************************************

*********************************************************
INCIDENT INFO RECORD 1
*********************************************************
   INCIDENT_ID                   3805
   STATUS                        closed
   CREATE_TIME                   2007-06-18 21:35:25.012579 -07:00
   PROBLEM_ID                    2
   CLOSE_TIME                    2007-06-18 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                 2007-06-18 21:35:25.012579 -07:00
   LAST_INCIDENT                 3713
   LASTINC_TIME                  2007-06-18 21:35:44.754442 -07:00
   IMPACT1                       0
   IMPACT2                       0
   IMPACT3                       0
   IMPACT4                       0
   KEY_NAME                      Client ProcId
   KEY_VALUE                     oracle@stadh43 (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
```

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]]

表 15-15 に、SHOW PROBLEM のフラグを示します。

表 15-15 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 15-16 に、述語文字列で使用できるフィールドを表示します。
[-last *num*| -all]	最新の *num* 個の問題（-last）を示すか、またはすべての問題（*-all*）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 15-16 SHOW PROBLEM の問題フィールド

フィールド	型
PROBLEM_ID	number
PROBLEM_KEY	text(550)
FIRST_INCIDENT	number
FIRSTINC_TIME	timestamp
LAST_INCIDENT	number
LASTINC_TIME	timestamp
IMPACT1	number
IMPACT2	number
IMPACT3	number
IMPACT4	number
SERVICE_REQUEST	text(64)
BUG_NUMBER	text(64)

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"
```

SHOW REPORT

用途

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。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、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
```

参照：

- 「CREATE REPORT」（15-16 ページ）
- 「SHOW HM_RUN」（15-41 ページ）

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 15-18 に、SHOW TRACEFILE の引数を示します。

表 15-17 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 15-18 SHOW TRACEFILE コマンドのフラグ

フラグ	説明
-rt | -t	タイムスタンプに従ってトレース・ファイル名を順序付けます。-t はタイムスタンプでファイル名を昇順でソートし、-rt はファイル名を逆順でソートします。ファイル名の順序付けは、ファイルのディレクトリに対してのみ相対的に行われます。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
```

SPOOL

用途

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
```

ADRCI のトラブルシューティング

一般的な ADRCI エラー・メッセージの一部と、その考えられる原因および回避策を次に示します。

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、15-4 ページの「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。

16

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
- DBVERIFY を使用したセグメントの検査

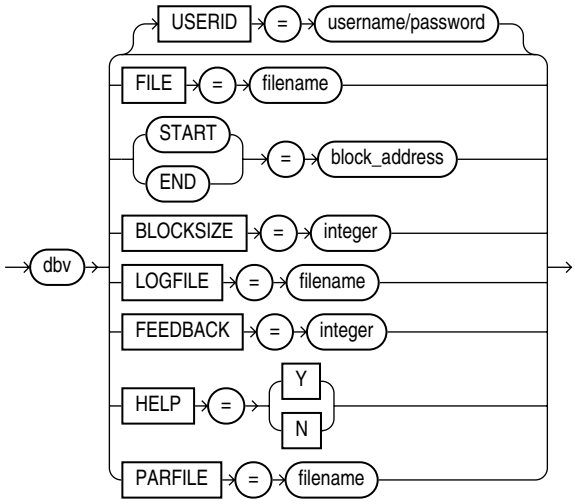
DBVERIFY を使用した単一データ・ファイルのディスク・ブロックの検査

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。

注意： 検査するファイルが、自動ストレージ管理（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 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。

構文

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。



パラメータ

パラメータの詳細は次のとおりです。

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 ASM ファイルである場合にのみ必要です。
FILE	検査するデータベース・ファイル名。
START	検査する最初のブロック・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、ファイル内の最初のブロックが DBVERIFY によってデフォルト設定されます。
END	検査する最後のブロック。END を指定しないと、ファイル内の最後のブロックが DBVERIFY によって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、BLOCKSIZE を指定しないと DBV-00103 のエラーが発生します。
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。

パラメータ	説明
FEEDBACK	進捗画面が端末に表示され、DBVERIFY の実行中に検証されたページ数 *n* が 1 つのピリオド（.）で示されます。*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・ファイル名を指定します。フラット・ファイルの DBVERIFY パラメータには、複数の値を格納できます。これによって、パラメータ・ファイルをカスタマイズして、異なるタイプのデータ・ファイルの処理、およびデータ・ファイルに固有の整合性チェックの実行ができます。

単一データ・ファイルに対する DBVERIFY の出力例

次に、ファイル t_db1.dbf の検査の例を示します。フィードバック・パラメータに 100 が指定されているため、100 ページの処理が行われるたびにピリオドが 1 つ表示されます。出力結果の一部も示します。

```
% dbv FILE=t_db1.dbf FEEDBACK=100
.
.
.
DBVERIFY - Verification starting : FILE = t_db1.dbf

.....................................................................


DBVERIFY - Verification complete

Total Pages Examined         : 9216
Total Pages Processed (Data) : 2044
Total Pages Failing   (Data) : 0
Total Pages Processed (Index): 733
Total Pages Failing   (Index): 0
Total Pages Empty            : 5686
Total Pages Marked Corrupt   : 0

Total Pages Influx           : 0
```

注意：

- Pages = ブロック。
- Total Pages Examined = ファイルのブロック数。
- Total Pages Processed = 検査されたブロック数（書式化されたブロック）。
- 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 がブロック型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ります（INFLUX）。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。

DBVERIFY を使用したセグメントの検査

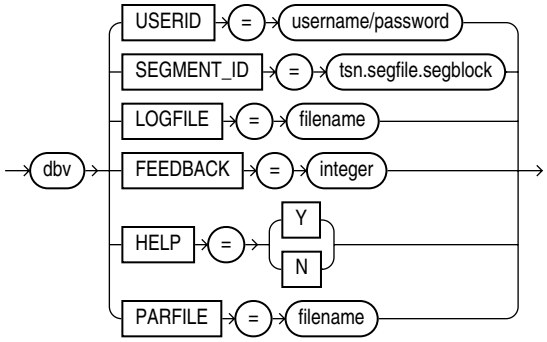
このモードでは、DBVERIFY を使用して表セグメントまたは索引セグメントの検査を指定できます。確認中のセグメントに行連鎖ポインタがあることを確認します。

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。

構文

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。



パラメータ

パラメータの詳細は次のとおりです。

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。
SEGMENT_ID	検査するセグメントを指定します。このパラメータは、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 から得られます。関係がある列は、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。
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	進捗画面が端末に表示され、DBVERIFY の実行中に検証されたページ数 *n* が 1 つのピリオド（.）で示されます。*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・ファイル名を指定します。フラット・ファイルの DBVERIFY パラメータには、複数の値を格納できます。これによって、パラメータ・ファイルをカスタマイズして、異なるタイプのデータ・ファイルの処理、およびデータ・ファイルに固有の整合性チェックの実行ができます。

検査対象のセグメントに対する DBVERIFY の出力例

次の出力例は、SEGMENT_ID 1.2.67 を検査する DBVERIFY 操作で表示されるものです。

```
DBVERIFY - Verification starting : SEGMENT_ID = 1.2.67

DBVERIFY - Verification complete

Total Pages Examined         : 8
Total Pages Processed (Data) : 0
Total Pages Failing   (Data) : 0
Total Pages Processed (Index): 1
Total Pages Failing   (Index): 0
Total Pages Processed (Other): 2
Total Pages Processed (Seg)  : 1
Total Pages Failing   (Seg)  : 0
Total Pages Empty            : 4
Total Pages Marked Corrupt   : 0
Total Pages Influx           : 0
Highest block SCN            : 7358 (0.7358)
```

17

DBNEWID ユーティリティ

DBNEWID ユーティリティは、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。

この章の内容は、次のとおりです。

DBNEWID ユーティリティとは

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）を指定することができました。ただし、データベースに新しい識別子（DBID）を指定することはできません。DBID は、データベース用の内部的な一意の識別子です。Recovery Manager 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 Recovery Manager 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

DBID および DBNAME の変更による影響

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。これは、データがすでにデータ・ファイルに存在する以外は、データベースの作成と同様です。DBID を変更すると、変更前に作成されたバックアップおよびアーカイブ・ログは使用できません。これは、バックアップやアーカイブ・ログにまだ元の DBID があるためです。元の DBID は最新の DBID とは一致しません。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あります。これによって、オンライン REDO ログが再作成され、この順序番号が 1 にリセットされます（詳細は、『Oracle Database 管理者ガイド』を参照）。このため、DBID の変更直後に、データベース全体のバックアップを取る必要があります。

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への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。取得プロセスの詳細は、『Oracle Streams 概要および管理』を参照してください。

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* 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・データベース名の変更は、SQL の ALTER DATABASE 文を使用してのみ行えます。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。

次の例では、ドメイン us.oracle.com のデータベース名を sales に変更します。

```
ALTER DATABASE RENAME GLOBAL_NAME TO sales.us.oracle.com
```

これは、DBNEWID を使用してデータベース名の変更を完了した後に行います。

参照： グローバル・データベース名の詳細は、『Oracle Database 管理者ガイド』を参照してください。

データベースの DBID および DBNAME の変更

この項の内容は、次のとおりです。

DBID およびデータベース名の変更

この項では、データベースの DBID を変更する方法について説明します。オプションで、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。次に例を示します。

```
SHUTDOWN IMMEDIATE
STARTUP MOUNT
```

3. コマンドラインで DBNEWID 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。次に例を示します。

```
% nid TARGET=SYS

DBNEWID: Release 11.1.0.6.0 - Production on Tue Aug 21 05:25:22 2007

Copyright (c) 1982, 2007, Oracle. All rights reserved.

Password: password
```

DBID に加えてデータベース名も変更するには、コマンドラインで DBNAME パラメータも指定します。パスワードを入力するように要求されます。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

DBNEWID: Release 11.1.0.6.0 - Production on Tue Aug 21 05:35:18 2007

Copyright (c) 1982, 2007, Oracle. All rights reserved.

Password: password
```

DBNEWID ユーティリティを使用して、データ・ファイルおよび制御ファイルへの入出力を試行する前に、これらの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（ログ・ファイルを指定した場合、確認は求められません）、NORMAL モードでオフラインにされたデータ・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 が変更されます。この例のように指定した場合は、DBNAME も変更されます。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。

```
.
.
.
Connected to database PROD (DBID=86997811)
.
.
.
Control Files in database:
    /oracle/TEST_DB/data/cf1.f
```

```
    /oracle/TEST_DB/data/cf2.f

The following datafiles are offline clean:
    /oracle/TEST_DB/data/tbs_61.f (23)
    /oracle/TEST_DB/data/tbs_62.f (24)
    /oracle/TEST_DB/data/temp3.f (3)
These files must be writable by this utility.

The following datafiles are read-only:
    /oracle/TEST_DB/data/tbs_51.f (15)
    /oracle/TEST_DB/data/tbs_52.f (16)
    /oracle/TEST_DB/data/tbs_53.f (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.f - modified
    Control File /oracle/TEST_DB/data/cf2.f - modified
    Datafile /oracle/TEST_DB/data/tbs_01.f - dbid changed, wrote new name
    Datafile /oracle/TEST_DB/data/tbs_ax1.f - dbid changed, wrote new name
    Datafile /oracle/TEST_DB/data/tbs_02.f - dbid changed, wrote new name
    Datafile /oracle/TEST_DB/data/tbs_11.f - dbid changed, wrote new name
    Datafile /oracle/TEST_DB/data/tbs_12.f - dbid changed, wrote new name
    Datafile /oracle/TEST_DB/data/temp1.f - dbid changed, wrote new name
    Control File /oracle/TEST_DB/data/cf1.f - dbid changed, wrote new name
    Control File /oracle/TEST_DB/data/cf2.f - 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で、DBNEWID ユーティリティ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することができます。

```
.
.
.
Connected to database PROD (DBID=86997811)
.
.
.
Control Files in database:
    /oracle/TEST_DB/data/cf1.f
    /oracle/TEST_DB/data/cf2.f

The following datafiles are offline clean:
    /oracle/TEST_DB/data/tbs_61.f (23)
    /oracle/TEST_DB/data/tbs_62.f (24)
    /oracle/TEST_DB/data/temp3.f (3)
These files must be writable by this utility.

The following datafiles are read-only:
    /oracle/TEST_DB/data/tbs_51.f (15)
    /oracle/TEST_DB/data/tbs_52.f (16)
    /oracle/TEST_DB/data/tbs_53.f (22)
These files must be writable by this utility.
```

```
The following datafiles are offline immediate:
    /oracle/TEST_DB/data/tbs_71.f (25)
    /oracle/TEST_DB/data/tbs_72.f (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。次に例を示します。

```
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。次に例を示します。

```
ALTER DATABASE OPEN RESETLOGS;
```

データベースの新しいバックアップを取ります。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。

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 を変更するには、17-3 ページの「DBID およびデータベース名の変更」の手順に従ってください。ただし、手順 3 では、オプションのデータベース名（DBNAME）を指定しません。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.f
    /oracle/TEST_DB/data/cf2.f

The following datafiles are offline clean:
    /oracle/TEST_DB/data/tbs_61.f (23)
    /oracle/TEST_DB/data/tbs_62.f (24)
    /oracle/TEST_DB/data/temp3.f (3)
These files must be writable by this utility.

The following datafiles are read-only:
    /oracle/TEST_DB/data/tbs_51.f (15)
    /oracle/TEST_DB/data/tbs_52.f (16)
    /oracle/TEST_DB/data/tbs_53.f (22)
These files must be writable by this utility.

Changing database ID from 86997811 to 4004383693
    Control File /oracle/TEST_DB/data/cf1.f - modified
    Control File /oracle/TEST_DB/data/cf2.f - modified
    Datafile /oracle/TEST_DB/data/tbs_01.f - dbid changed
    Datafile /oracle/TEST_DB/data/tbs_ax1.f - dbid changed
    Datafile /oracle/TEST_DB/data/tbs_02.f - dbid changed
    Datafile /oracle/TEST_DB/data/tbs_11.f - dbid changed
    Datafile /oracle/TEST_DB/data/tbs_12.f - dbid changed
    Datafile /oracle/TEST_DB/data/temp1.f - dbid changed
    Control File /oracle/TEST_DB/data/cf1.f - dbid changed
    Control File /oracle/TEST_DB/data/cf2.f - 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.
```

データベース名のみの変更

この項では、DBID を変更せずにデータベース名を変更する方法について説明します。

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。次に例を示します。

   ```
   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: Release 11.1.0.6.0 - Production on Tue Aug 21 05:40:21 2007

   Copyright (c) 1982, 2007, Oracle. All rights reserved.

   Password: password
   ```

   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、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.f
       /oracle/TEST_DB/data/cf2.f

   The following datafiles are offline clean:
       /oracle/TEST_DB/data/tbs_61.f (23)
       /oracle/TEST_DB/data/tbs_62.f (24)
       /oracle/TEST_DB/data/temp3.f (3)
   These files must be writable by this utility.

   The following datafiles are read-only:
       /oracle/TEST_DB/data/tbs_51.f (15)
       /oracle/TEST_DB/data/tbs_52.f (16)
       /oracle/TEST_DB/data/tbs_53.f (22)
   These files must be writable by this utility.

   Changing database name from PROD to TEST_DB
       Control File /oracle/TEST_DB/data/cf1.f - modified
       Control File /oracle/TEST_DB/data/cf2.f - modified
       Datafile /oracle/TEST_DB/data/tbs_01.f - wrote new name
       Datafile /oracle/TEST_DB/data/tbs_ax1.f - wrote new name
       Datafile /oracle/TEST_DB/data/tbs_02.f - wrote new name
       Datafile /oracle/TEST_DB/data/tbs_11.f - wrote new name
   ```

```
Datafile /oracle/TEST_DB/data/tbs_12.f - wrote new name
Datafile /oracle/TEST_DB/data/temp1.f - wrote new name
Control File /oracle/TEST_DB/data/cf1.f - wrote new name
Control File /oracle/TEST_DB/data/cf2.f - 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、17-3 ページの「DBID およびデータベース名の変更」の手順 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。

> **注意：** 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。次に例を示します。

```
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。これまでのすべてのバックアップは使用可能です。

DBNEWID のトラブルシューティング

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。そのため、DBNEWID ユーティリティの実行前に最新のバックアップを使用可能にしておくことが重要です。

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 がリリース 1（10.1）以上の Oracle Database に対して実行された場合は、操作の要約がアラート・ファイルに記録されます。たとえば、データベース名およびデータベース ID の変更では、次のような内容が記録されます。

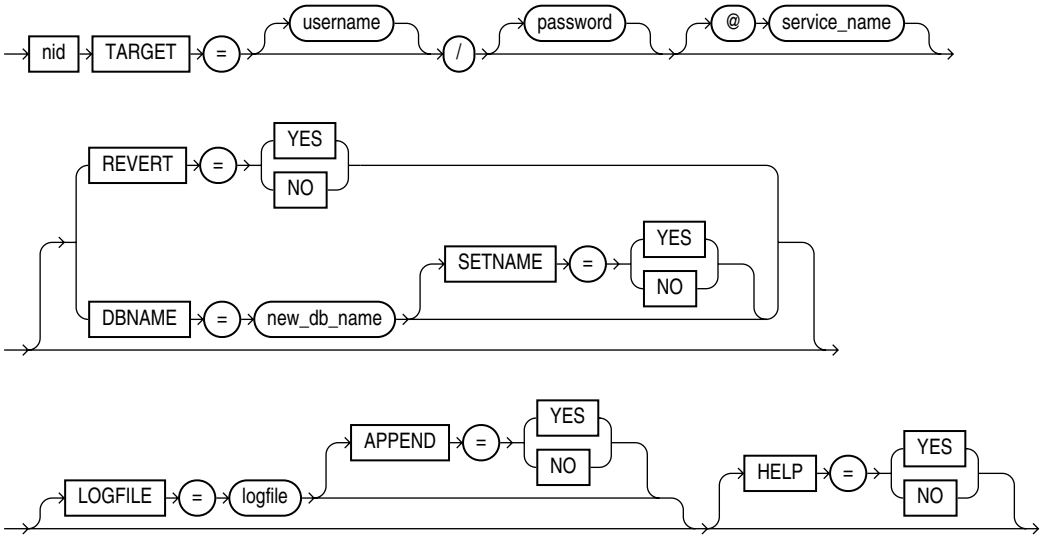
```
*** DBNEWID utility started ***
DBID will be changed from 86997811 to new DBID of 1250452230 for
database PROD
DBNAME will be changed from PROD to new DBNAME of TEST_DB
Starting datafile conversion
Setting recovery target incarnation to 1
Datafile conversion complete
Database name changed to TEST_DB.
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 ***
```

DBNEWID ユーティリティの構文

次の図に、DBNEWID ユーティリティの構文を示します。



パラメータ

表 17-1 に、DBNEWID ユーティリティの構文のパラメータを示します。

表 17-1 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。
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。
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。

制限事項および使用上の注意

DBNEWID ユーティリティには次の制限事項があります。

- 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（詳細は、『Oracle Database 管理者ガイド』を参照）。
- 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、17-2 ページの「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。

Oracle Database 10*g* 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 1（10.1）より前の Oracle Database に対し実行する場合は、次の制限事項がさらに適用されます。

- `nid` 実行可能ファイルを実行するにはデータ・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。

18

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。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。

この章の内容は、次のとおりです。

この章では、コマンドラインから LogMiner を使用する方法について説明します。LogMiner には、Oracle LogMiner ビューア Graphical User Interface からもアクセスできます。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。

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、18-13 ページの「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。例は、18-63 ページの「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。例は、18-64 ページの「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。データベース監査の詳細は、『Oracle Database 管理者ガイド』を参照してください。）

LogMiner の概要

次の各項では、LogMiner の概要を説明します。

- LogMiner の構成
- LogMiner 操作の指定および分析するデータの取得

この章の後半では、これらの概念と関連トピックを詳しく説明します。

LogMiner の構成

LogMiner 構成には、よく理解しておく必要がある次の 4 つの基本オブジェクトがあります。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、分析対象のデータが含まれる REDO ログ・ファイルです。

- **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**は、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するために使用されます。

  LogMiner は、内部オブジェクト ID とデータ型をオブジェクト名と外部データ書式に変換するためにもディクショナリを使用します。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 文について考えます。

```
INSERT INTO HR.JOBS(JOB_ID, JOB_TITLE, MIN_SALARY, MAX_SALARY)
 VALUES('IT_WT','Technical Writer', 4000, 11000);
```

  ディクショナリがない場合、LogMiner は次のように表示します。

```
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。

サンプル構成

図 18-1 にサンプル LogMiner 構成を示します。この図では、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は Oracle9*i* を実行しており、サンフランシスコのデータベースでは Oracle Database 10*g* を実行しています。

図 18-1 サンプル LogMiner データベースの構成

図 18-1 は、有効な LogMiner 構成の一例です。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。他のデータ・ディクショナリ・オプションの詳細は、18-5 ページの「LogMiner ディクショナリ・オプション」を参照してください。

要件

次に、ソース・データベース、マイニング・データベース、データ・ディクショナリ、および LogMiner がマイニングする REDO ログ・ファイルの要件を示します。

- 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・データベースでは、ソース・データベースと同一またはそれ以上のバージョンの Oracle Database ソフトウェアを稼働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キャラクタ・セット（またはキャラクタ・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 が分析する 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により生成される必要があります。
- すべての REDO ログ・ファイルに対する条件を次に示します。
  - 同じソース・データベースで生成される必要があります。
  - 同じデータベース RESETLOGS SCN と関連付けられている必要があります。
  - リリース 8.0 以上の Oracle Database である必要があります。ただし、リリース 1（9.0.1）で導入された LogMiner 機能の一部は、Oracle9*i* 以上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。詳細は、18-66 ページの「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」を参照してください。

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、また分析対象の 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。

注意： 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・ロギングの詳細は、18-25 ページの「サプリメンタル・ロギング」を参照してください。

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、それぞれ次のように使用します。

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（手順 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（手順 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。（このビューを問い合せるには、SELECT ANY TRANSACTION 権限が必要です。）

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。

LogMiner PL/SQL パッケージの使用および V$LOGMNR_CONTENTS ビューの問合せでは、EXECUTE_CATALOG_ROLE ロールが付与されている必要があります。

注意： 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。この制限事項は、CONTINUOUS_MINE オプションを使用して、ソース・データベースでアーカイブ・ログをマイニングする場合にも適用されます。有効または無効なスレッドがない場合にのみ、Oracle RAC データベースで CONTINUOUS_MINE オプションを使用する必要があります。

参照： LogMiner の使用例は、18-35 ページの「一般的な LogMiner セッションの手順」を参照してください。

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
- REDO ログ・ファイル・オプション

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

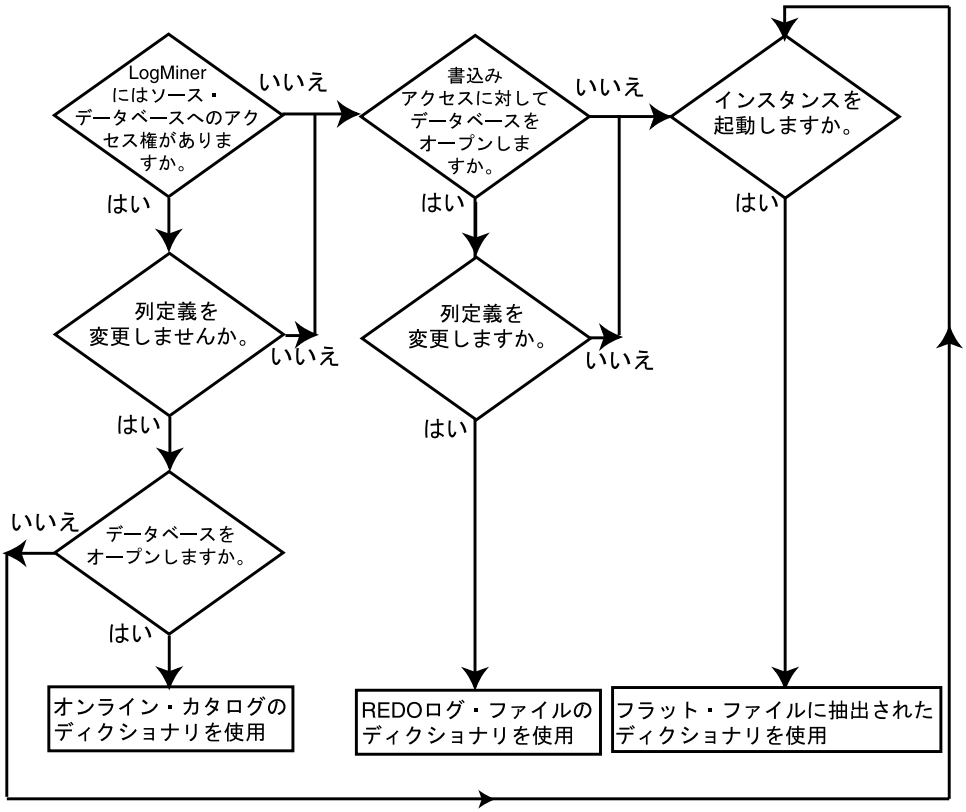
  REDO ログ・ファイルが作成されたソース・データベースにアクセスできない場合、または分析する表の列定義が変更される可能性がある場合には、このオプションの使用をお薦めします。

- 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・ファイルからディクショナリを抽出することをお薦めします。

図 18-2 に、状況に合わせた LogMiner ディクショナリの選択方法を示します。

図 18-2 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、次のように、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。

```
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。

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);
```

> **参照：** ARCHIVELOG モードの詳細は『Oracle Database バックアップおよびリカバリ基礎』を、DBMS_LOGMNR_D.BUILD プロシージャの詳細は、『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』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・ファイル名を（必要であればその間の他のログも）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、そのための特別な手順は必要ありません。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。

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。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。

ディクショナリの作成中に DDL 操作が発生しないように注意してください。

ディクショナリをフラット・ファイルに抽出する手順は、次のとおりです。手順 1 と 2 は準備のための手順です。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、DBMS_LOGMNR_D.BUILD プロシージャが使用するディレクトリ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。ディレクトリを指定するには、初期化パラメータ・ファイルで UTL_FILE_DIR 初期化パラメータを設定します。

   参照： 初期化パラメータファイル（init.ora）の詳細は、

   『Oracle Database リファレンス』を、DBMS_LOGMNR_D.BUILD プロシージャの詳細は、『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』を参照してください。

   たとえば、ディクショナリ・ファイルを置くディレクトリとして /oracle/database を使用するように UTL_FILE_DIR を設定するには、次の内容を初期化パラメータ・ファイルに記述します。

   ```
   UTL_FILE_DIR = /oracle/database
   ```

   初期化パラメータ・ファイルに対する変更を有効にするには、データベースの停止と再起動が必要であることに注意してください。

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。

   ```
   STARTUP
   ```

3. DBMS_LOGMNR_D.BUILD PL/SQL プロシージャを実行します。ディクショナリのファイル名およびこのファイルのディレクトリ・パス名を指定します。このプロシージャにより、ディクショナリ・ファイルが作成されます。たとえば、/oracle/database で dictionary.ora ファイルを作成するには、次のとおり入力します。

   ```
   EXECUTE DBMS_LOGMNR_D.BUILD('dictionary.ora', -
      '/oracle/database/', -
       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。

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。それには、DBMS_LOGMNR.START_LOGMNR プロシージャで LogMiner を起動するときに、CONTINUOUS_MINE オプションを使用し、時間範囲または SCN 範囲を指定します。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。

注意： CONTINUOUS_MINE オプションでは、データベースがマウントされ、アーカイブが使用可能である必要があります。

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。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( -
   STARTTIME => '01-Jan-2003 08:30:00', -
   ENDTIME => '01-Jan-2003 08:45:00', -
   OPTIONS => DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG + -
   DBMS_LOGMNR.CONTINUOUS_MINE);
```

（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。）

LogMiner 起動時に、DBMS_LOGMNR.ADD_LOGFILE を使用して 1 つの REDO ログ・ファイルのみを指定し、CONTINUOUS_MINE オプションを指定しても、分析する REDO ログ・ファイルのリストを LogMiner に自動的に作成させることができます。ただし、最初に説明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。

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。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: 18-6 ページの「オンライン・カタログの使用」を参照
- DICT_FROM_REDO_LOGS: 18-37 ページの「LogMiner の起動」を参照
- CONTINUOUS_MINE: 18-8 ページの「REDO ログ・ファイル・オプション」を参照
- COMMITTED_DATA_ONLY: 18-19 ページの「コミット済トランザクションのみの表示」を参照
- SKIP_CORRUPTION: 18-21 ページの「REDO 破損のスキップ」を参照
- NO_SQL_DELIMITER: 18-23 ページの「再実行のために再構築された SQL 文の書式設定」を参照
- PRINT_PRETTY_SQL: 18-23 ページの「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」を参照
- NO_ROWID_IN_STMT: 18-23 ページの「再実行のために再構築された SQL 文の書式設定」を参照
- DDL_DICT_TRACKING: 18-30 ページの「LogMiner ディクショナリでの DDL 文の追跡」を参照

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$LOGMNR_CONTENTS ビューは、18-12 ページの「V$LOGMNR_CONTENTS ビューへの移入方法」で説明するとおり、この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。

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、SELECT ANY TRANSACTION 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。

注意： LogMiner は透過的データ暗号化（TDE）をサポートしているため、LogMiner データ・ディクショナリに対象オブジェクトに必要なメタデータが含まれ、適切なマスター・キーが Oracle Wallet にある場合は、V$LOGMNR_CONTENTS に、暗号化された列（更新中の暗号化された列を含む）が含まれる表で実行された DML 操作が表示されます。Oracle Wallet はオープンされている必要があります。オープンされていない場合は、V$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。

V$LOGMNR_CONTENTS の問合せの例

たとえば、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';
```

次の出力が生成されます。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。その理由は、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 の出力にはそれが反映されません。そのため、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$LOGMNR_CONTENTS 固定ビューは、他のビューと異なり、表に格納されたデータの選択的表示ではありません。ユーザーが REDO ログ・ファイルに対して要求したデータのリレーショナルな表示です。LogMiner がこのビューに移入するのは、このビューに対する問合せに応答する場合のみです。V$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。次に例を示します。

- 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。

> **参照：** DBMS_LOGMNR.START_LOGMNR に対する COMMITTED_DATA_ONLY オプションの詳細は、18-19 ページの「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。

注意： 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。

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を返します。

MINE_VALUE 関数によって返される NULL 値の意味

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;
```

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 関数には、次の使用規則が適用されます。

- 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呼び出す必要があります。
- LONG、LONG RAW、CLOB、BLOB、NCLOB、ADT または COLLECTION の各データ型はサポートしていません。

参照： MINE_VALUE 関数および COLUMN_PRESENT 関数を含む DBMS_LOGMNR パッケージの詳細は、『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』を参照してください。

XMLType 列および XMLType 表に基づいた V$LOGMNR_CONTENTS の問合せ

データが CLOB として格納されている場合、LogMiner は XMLType データをサポートします。LOB データとは異なり、XML 文書では空の文書を挿入し、文書の一部分を使用して追加していくことはできません。したがって、XML 文書は作成してから、それ全体を挿入する必要があります。

この要件を満たすために、LogMiner では、XML データを表すためのバインド変数を使用した DML 文に対して、V$LOGMNR_CONTENTS の SQL_REDO が表示されます。この後に、XML 文書の一部を含む 1 つ以上の行が示されます。

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、手順 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。CLOB として格納される 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。

次に、CLOB として格納される XMLType 表に対する変更のマイニングの例を示します。

```
select operation, status, object_id, sql_redo from v$logmnr_contents
where  seg_owner = 'SCOTT' and table_name = 'XML_TYPE_CLOB_TAB';

OPERATION     STATUS   OBJECT_ID                          SQL_REDO

INSERT          2      300A9394B0F7B2D0E040578CF5025CC3   insert into "SCOTT"."XML_TYPE_CLOB_TAB"
                                                            values(EMPTY_CLOB())

XML DOC BEGIN   5      300A9394B0F7B2D0E040578CF5025CC3   insert into "SCOTT"."XML_TYPE_CLOB_TAB"
                                                            values (XMLType(:1)

XML DOC WRITE   5      300A9394B0F7B2D0E040578CF5025CC3   XML Data

XML DOC WRITE   5      300A9394B0F7B2D0E040578CF5025CC3   XML Data

XML DOC WRITE   5      300A9394B0F7B2D0E040578CF5025CC3   XML Data

XML DOC END     5
```

全体的なパターンは、XMLType 列に非常に類似しています。ただし、いくつかの主な違いがあります。1 つは、OBJECT_ID 列が移入されている点です。2 つ目の相違点は、最初の挿入にもかかわらず、INVALID_SQL の状態が 2 である点です。このことは、このレコードが行われる変更に対するプレースフォルダとして REDO で発生するにもかかわらず、この変更に対して生成された SQL が適用できないことを示しています。XML DOC BEGIN 操作の SQL_REDO には、作成された XML 文書とともに使用する際に、行に対して行われた変更が反映されます。

XML 文書が表外の列として格納されていない場合、その文書に対して 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 INSERT 文に含まれます。

```
OPERATION   STATUS  OBJECT_ID                               SQL_REDO

INSERT      2       300AD8CECBA75ACAE040578CF502640C        insert into "SCOTT"."XML_TYPE_CLOB_TAB"
                                                               values (EMPTY_CLOB())

INSERT      0       300AD8CECBA75ACAE040578CF502640C        insert into "SCOTT"."XML_TYPE_CLOB_TAB"
                                                               values (XMLType(
                                                               '<?xml version="1.0"?>
                                                               <PO pono="1">
                                                               <PNAME>Po_99</PNAME>
                                                               <CUSTNAME>
                                                               Dave Davids
                                                               </CUSTNAME>
                                                               </PO>'))
```

XMLType データを使用する LogMiner の使用上の制限

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。

XMLType データを作成するための PL/SQL プロシージャの例

この項の例では、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します。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。

> **注意：** 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'
```

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。

- コミット済トランザクションのみの表示
- REDO 破損のスキップ
- 時刻によるデータのフィルタ処理
- SCN によるデータのフィルタ処理

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。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。

> **注意：** 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。

> **参照：** COMMITTED_DATA_ONLY オプションの使用例の詳細は、18-39 ページの「LogMiner の使用例」を参照してください。

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$LOGMINER_CONTENTS view. This example shows corruptions are -- in
the redo log files.
--
SELECT rbasqn, rbablk, rbabyte, operation, status, info
   FROM V$LOGMNR_CONTENTS;

ERROR at line 3:
ORA-00368: checksum error in redo log block
ORA-00353: log corruption near block 6 change 73528 time 11/06/2002 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$LOGMINER_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
```

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-1998 08:30:00', -
   ENDTIME => '01-Jan-1998 08:45:00'-
   OPTIONS => DBMS_LOGMNR.CONTINUOUS_MINE);
```

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。

参照：

- 時刻によるデータのフィルタ処理の例の詳細は、18-39 ページの「LogMiner の使用例」を参照してください。
- 開始時刻および終了時刻を指定し、その指定時刻が LogMiner REDO ログ・ファイル・リストで検出されなかった場合に発生する状態、およびこれらのパラメータと CONTINUOUS_MINE オプションとの相互作用の詳細は、『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』を参照してください。

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、次の例に示すように、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。

```
EXECUTE DBMS_LOGMNR.START_LOGMNR(-
   STARTSCN => 621047, -
   ENDSCN   => 625695, -
   OPTIONS  => DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG + -
               DBMS_LOGMNR.CONTINUOUS_MINE);
```

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。

参照：

- SCN によるデータのフィルタ処理の例の詳細は、18-39 ページの「LogMiner の使用例」を参照してください。
- 開始 SCN および終了 SCN の値を指定し、指定した SCN が LogMiner REDO ログ・ファイル・リストで検出されなかった場合に発生する状態、およびこれらのパラメータと CONTINUOUS_MINE オプションとの相互作用の詳細は、『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』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

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。

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

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。

> **参照：** PRINT_PRETTY_SQL オプションの使用例の詳細は、18-39 ページの「LogMiner の使用例」を参照してください。

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 DDL 文が Oracle によって内部的に実行される場合があることに注意してください。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。

> **注意：** 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。例は、18-46 ページの「例 4: REDO ログ・ファイル内の LogMiner ディクショナリの使用」の手順 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、V$LOGMNR_CONTENTS の INFO 列に値 USER_DDL が含まれる場合にのみ、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。

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。

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 ～ 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、手順 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 1 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 2 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、手順 2 と 3 を繰り返します。

サプリメンタル・ロギング

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」（18-26 ページ）
- 「表レベルのサプリメンタル・ロギング」（18-28 ページ）

参照： 「サプリメンタル・ロギング設定に関するビューの問合せ」（18-34 ページ）

データベース・レベルのサプリメンタル・ロギング

データベース・レベルのサプリメンタル・ロギングには、次の項で説明するとおり、最小サプリメンタル・ロギングと識別キー・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。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;
```

注意： Oracle Database のリリース 1（9.0.1）では、最小サプリメンタル・ロギングが LogMiner のデフォルト動作でした。リリース 2（9.2）以上では、デフォルトでサプリメンタル・ロギングは行われません。サプリメンタル・ロギングは、明示的に有効にする必要があります。

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。

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）、Oracle Database 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による場合があります。

  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;
```

注意： 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、18-23 ページの「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。
- 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;
```

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。

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。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・サプリメンタル・ロギングの両方が表レベルで有効になります。

```
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、18-26 ページの「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。

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。)

  注意： 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。

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注意事項

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。

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。次に例を示します。

```
EXECUTE DBMS_LOGMNR.START_LOGMNR(OPTIONS => -
   DBMS_LOGMNR.DDL_DICT_TRACKING + DBMS_LOGMNR.DICT_FROM_REDO_LOGS);
```

例の詳細は、18-54 ページの「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。

注意： 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。

注意： 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。

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次の相互作用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。

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、次のログ・ファイルが読み込まれます。
   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$LOGMNR_PARAMETERS ビューの REQUIRED_START_DATE 列または REQUIRED_START_SCN 列を問い合せます。LogMiner が読取りを開始する位置に関係なく、要求した範囲内の行のみが V$LOGMINER_CONTENTS ビューから返されます。

ビューでの LogMiner 操作情報へのアクセス

LogMiner 操作情報（REDO データではなく）は、次の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、18-33 ページの「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、18-34 ページの「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。

  参照： これらのビューの内容の詳細は、『Oracle Database リファレンス』を参照してください。

V$LOGMNR_LOGS の問合せ

V$LOGMNR_LOGS ビューを問い合せて、LogMiner による分析のために 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・ファイルを判別することができます。このビューでは、REDO ログ・ファイルごとに 1 行が含まれます。各 REDO ログ・ファイルに関する重要な情報（ファイル名、順序番号、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。この 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。

  例 : スレッド番号 1、順序番号 100 ～ 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 へのコール時に指定可能な DDL_DICT_TRACKING オプションおよび CONTINUOUS_MINE 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$LOGMNR_LOGS ビューの STATUS 列を問い合せることによって、リストから欠落している REDO ログ・ファイルを判別できます。その後、欠落している REDO ログ・ファイルを検索して、手動で追加すると、再度 DBMS_LOGMNR.START_LOGMNR をコールできます。
- DBMS_LOGMNR.START_LOGMNR へのコール時に指定可能なその他のオプションでは、LogMiner REDO ログ・ファイル・リストからファイルが欠落する場合がありますが、ファイルの欠落は、可能なかぎり回避する必要があります。V$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 列の欠落している SCN 範囲を示す文字列（「欠落 SCN 100 ～ 200」など）

サプリメンタル・ロギング設定に関するビューの問合せ

次のリストに示すように、多数のビューを問い合せてサプリメンタル・ロギングの現在の設定を確認できます。

- 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 の ALTER DATABASE ADD SUPPLEMENTAL LOG DATA 文が発行されたため、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_TYPES 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。

一般的な LogMiner セッションの手順

この項では、一般的 LogMiner セッションの手順について説明します。各手順の詳細は、該当する後述の項を参照してください。

1. サプリメンタル・ロギングの有効化
2. LogMiner ディクショナリの抽出（オンライン・カタログを使用する予定でない場合）
3. 分析する REDO ログ・ファイルの指定
4. LogMiner の起動
5. V$LOGMNR_CONTENTS の問合せ
6. LogMiner セッションの終了

LogMiner を実行するには、DBMS_LOGMNR PL/SQL パッケージを使用します。また、オンライン・カタログを使用するのではなく、LogMiner ディクショナリを抽出した場合は、DBMS_LOGMNR_D パッケージも使用します。

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。

LogMiner PL/SQL パッケージは、SYS スキーマが所有します。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。次に例を示します。

```
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。

参照：

- これらの LogMiner パッケージの構文とパラメータの詳細は、『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』を参照してください。
- PL/SQL プロシージャの実行方法の詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。

サプリメンタル・ロギングの有効化

使用する種類のサプリメンタル・ロギングを有効にします。最低限、次のように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。

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;
```

詳細は、18-25 ページの「サプリメンタル・ロギング」を参照してください。

LogMiner ディクショナリの抽出

LogMiner を使用するには、次のいずれかの方法でディクショナリを含める必要があります。

- 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。詳細は、18-6 ページの「オンライン・カタログの使用」を参照してください。
- 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。詳細は、18-7 ページの「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。
- 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。詳細は、18-7 ページの「フラット・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。

分析する REDO ログ・ファイルの指定

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。この指定を行うには、次の手順で示すとおり、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。

注意： REDO ログ・ファイルを生成するデータベース・インスタンスでマイニングを行う場合は、LogMiner の起動時に CONTINUOUS_MINE オプションおよび次のいずれかのパラメータを指定する必要があります。

- STARTSCN パラメータ
- STARTTIME パラメータ

詳細は、18-8 ページの「REDO ログ・ファイル・オプション」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 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');
   ```

LogMiner の起動

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 を起動する必要があります。次の手順を実行します。

1. DBMS_LOGMNR.START_LOGMNR プロシージャを実行して、LogMiner を起動します。

   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。この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。したがって、内部オブジェクト ID が返され、データはバイナリ・データとして提供されます。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。例は、18-7 ページの「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。

注意： 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 に対する現在のコールでディクショナリを指定しない場合は、前回のコールで指定したディクショナリ仕様を保持します。

DICT_FROM_ONLINE_CATALOG オプションの詳細は、18-6 ページの「オンライン・カタログの使用」を参照してください。

2. オプションで、時刻または SCN によって問合せをフィルタ処理できます。18-22 ページの「時刻によるデータのフィルタ処理」または 18-22 ページの「SCN によるデータのフィルタ処理」を参照してください。

3. 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することもできます。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 オプションの詳細は、『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』を参照してください。

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。この機能は、たとえば、V$LOGMNR_CONTENTS の問合せから必要な結果を取得できず、他のオプションを指定して LogMiner を再起動する場合に有効です。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。

V$LOGMNR_CONTENTS の問合せ

この時点で、LogMiner は起動され、V$LOGMNR_CONTENTS ビューに対して問合せを実行できます。この操作例は、18-19 ページの「V$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。

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、起動を実行した 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。

LogMiner の使用例

この項では、次の各一般的カテゴリでの LogMiner の使用例を示します。

- 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
- REDO ログ・ファイルのリストを明示的に指定しないマイニングの例
- 使用例

注意： 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;
```

詳細は、18-25 ページの「サプリメンタル・ロギング」を参照してください。

18-61 ページの「例 2: 指定した SCN 範囲での REDO ログ・ファイルのマイニング」と 18-63 ページの「使用例」以外のすべての例では、NLS_DATE_FORMAT パラメータが次のように設定されているとします。

```
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が表示されるため、この手順はオプションです。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

次の例では、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。この項は、次に示す例で構成されています。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。

- 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
- 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
- 例 3: 再構築された SQL の書式設定
- 例 4: REDO ログ・ファイル内の LogMiner ディクショナリの使用
- 例 5: 内部ディクショナリでの DDL 文の追跡
- 例 6: 時間範囲による出力のフィルタ処理

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。

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

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（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 データベースでないものとします）。

手順 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。
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。

```
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
```

手順 2 分析する REDO ログ・ファイルのリストを指定します。
手順 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);
```

手順 3 LogMiner を起動します。
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);
```

手順 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-2003
                  ('306','Nandini','Shastry',       13:34:43', 'dd-mon-yyyy
hh24:mi:ss')
                  'NSHASTRY', '1234567890',         and "JOB_ID" = 'HR_REP' and
                  TO_DATE('10-jan-2003 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-2003
                  ('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-2003 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;
```

手順 5 LogMiner セッションを終了します。

```
SQL> EXECUTE DBMS_LOGMNR.END_LOGMNR();
```

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

最初の例（18-39 ページの「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」）で示したとおり、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。

手順 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。

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。

```
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
```

手順 2 分析する REDO ログ・ファイルのリストを指定します。

手順 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);
```

手順 3 LogMiner を起動します。

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);
```

手順 4 V$LOGMNR_CONTENTS ビューを問い合せます。

18-39 ページの「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」に示したとおり、トランザクション 1.11.1476 は 1.1.1484 より前に開始されていますが、トランザクション 1.1.1484 より前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-2003
                    ('306','Nandini','Shastry',        13:34:43', 'dd-mon-yyyy
hh24:mi:ss')
                    'NSHASTRY', '1234567890',          and "JOB_ID" = 'HR_REP' and
                    TO_DATE('10-jan-2003 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-2003
                    ('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-2003 13:41:03',    = '50' and ROWID =
'AAAHSkAABAAAY6rAAP';
                    'dd-mon-yyyy hh24:mi:ss'),
                    'SH_CLERK','110000', '.05',
                    '105','50');

HR      1.11.1476   commit;
```

手順 5 LogMiner セッションを終了します。

```
EXECUTE DBMS_LOGMNR.END_LOGMNR();
```

例 3: 再構築された SQL の書式設定

18-42 ページの「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」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。

手順 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。

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。

```
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
```

手順 2 分析する REDO ログ・ファイルのリストを指定します。

手順 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);
```

手順 3 LogMiner を起動します。

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。

手順 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-2003 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-2003 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;
```

手順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-2003 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-2003 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;
```

手順 6 LogMiner セッションを終了します。

```
EXECUTE DBMS_LOGMNR.END_LOGMNR();
```

例 4: REDO ログ・ファイル内の LogMiner ディクショナリの使用

この例では、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示します。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。

手順 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。

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。

```
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
```

手順 2 ディクショナリを含む REDO ログ・ファイルを検索します。

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。

1. 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
```

2. データ・ディクショナリ抽出の先頭（前の手順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
```

3. 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');
```

4. 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。

```
SQL> SELECT FILENAME AS name, LOW_TIME, HIGH_TIME FROM V$LOGMNR_LOGS;
 NAME                                LOW_TIME             HIGH_TIME
-----------------------------------  -------------------  -------------------
/usr/data/db1arch_1_207_482701534.dbf   10-jan-2003 12:01:34  10-jan-2003 13:32:46

/usr/data/db1arch_1_208_482701534.dbf   10-jan-2003 13:32:46  10-jan-2003 15:57:03

Missing logfile(s) for thread number 1, 10-jan-2003 15:57:03  10-jan-2003 15:59:53
sequence number(s) 209 to 209

/usr/data/db1arch_1_210_482701534.dbf   10-jan-2003 15:59:53  10-jan-2003 16:07:41
```

手順 3 LogMiner を起動します。

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);
```

手順 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-2003 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;
```

手順 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。

> **注意：** 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-2003 14:01:03', 'dd-mon-yyyy hh24:mi:ss'),
    "MTIME" = TO_DATE('13-jan-2003 14:01:03', 'dd-mon-yyyy hh24:mi:ss'),
    "STIME" = TO_DATE('13-jan-2003 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-2003 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-2003 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;
```

手順 6 LogMiner セッションを終了します。

```
EXECUTE DBMS_LOGMNR.END_LOGMNR();
```

例 5: 内部ディクショナリでの DDL 文の追跡

この例では、DBMS_LOGMNR.DDL_DICT_TRACKING オプションを使用して、REDO ログ・ファイル内で検出された DDL 文で LogMiner 内部ディクショナリを更新する方法を示します。

手順 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。

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。

```
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
```

手順 2 REDO ログ・ファイル内のディクショナリを検索します。

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。

1. 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
   ```

2. 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
   ```

手順 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
```

手順 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');
```

手順 5 LogMiner を起動します。

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);
```

手順 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-2003 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-2003 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-2003 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;
```

手順 7 LogMiner セッションを終了します。

```
EXECUTE DBMS_LOGMNR.END_LOGMNR();
```

例 6: 時間範囲による出力のフィルタ処理

前の 2 つの例では、問合せにタイムスタンプ・ベースの条件（TIMESTAMP > '10-jan-2003 15:59:53'）を指定して、行をフィルタ処理しました。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。

手順 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 2003 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-2003 14:00:00');
```

手順 2 V$LOGMNR_LOGS を問い合せて、REDO ログ・ファイルのリストを確認します。
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-2003 14:02:35  23683584
/usr/orcl/arch1_311_482932022.dbf    13-jan-2003 14:56:35  2564096
/usr/orcl/arch1_312_482932022.dbf    13-jan-2003 15:10:43  23683584
/usr/orcl/arch1_313_482932022.dbf    13-jan-2003 15:17:52  23683584
/usr/orcl/arch1_314_482932022.dbf    13-jan-2003 15:23:10  23683584
/usr/orcl/arch1_315_482932022.dbf    13-jan-2003 15:43:22  23683584
/usr/orcl/arch1_316_482932022.dbf    13-jan-2003 16:03:10  23683584
/usr/orcl/arch1_317_482932022.dbf    13-jan-2003 16:33:43  23683584
/usr/orcl/arch1_318_482932022.dbf    13-jan-2003 17:23:10  23683584
```

手順 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-2003 15:00:00'` および `TIMESTAMP < '13-jan-2003 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-2003 15:00:00', -
   ENDTIME   => '13-jan-2003 16:00:00', -
   OPTIONS   => DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG + -
                DBMS_LOGMNR.COMMITTED_DATA_ONLY + -
                DBMS_LOGMNR.PRINT_PRETTY_SQL);
```

手順 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-2003 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-2003 15:29:34   1.17.2376      insert into "OE"."PRODUCT_TRACKING"
                                        values
                                        "PRODUCT_ID" = 3399,
                                        "MODIFIED_TIME" = TO_DATE('13-jan-2003 15:29:34',
                                        'dd-mon-yyyy hh24:mi:ss'),
                                        "OLD_LIST_PRICE" = 815,
                                        "OLD_WARRANTY_PERIOD" = TO_YMINTERVAL('+02-00');

13-jan-2003 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-2003 15:52:43   1.15.1756      insert into "OE"."PRODUCT_TRACKING"
                                        values
                                        "PRODUCT_ID" = 1768,
                                        "MODIFIED_TIME" = TO_DATE('13-jan-2003 16:52:43',
                                        'dd-mon-yyyy hh24:mi:ss'),
                                        "OLD_LIST_PRICE" = 715,
                                        "OLD_WARRANTY_PERIOD" = TO_YMINTERVAL('+02-00');
```

手順 5 LogMiner セッションを終了します。

```
EXECUTE DBMS_LOGMNR.END_LOGMNR();
```

REDO ログ・ファイルのリストを明示的に指定しないマイニングの例

前述の例では、マイニングする REDO ログ・ファイルを明示的に指定しました。この例では、REDO ログ・ファイルを生成したデータベースと同じデータベースでマイニングする場合、分析する時間（または SCN）範囲を指定するのみで適切な REDO ログ・ファイルのリストをマイニングできる方法を示します。REDO ログ・ファイルを明示的に指定せずにマイニングするには、DBMS_LOGMNR.START_LOGMNR プロシージャに対して DBMS_LOGMNR.CONTINUOUS_MINE オプションを使用し、分析する時間範囲または SCN 範囲のいずれかを指定します。

この項は、次の例で構成されています。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。

- 例 1: 指定した時間範囲での REDO ログ・ファイルのマイニング
- 例 2: 指定した SCN 範囲での REDO ログ・ファイルのマイニング
- 例 3: 問合せに将来の値を含める CONTINUOUS_MINE オプションの使用

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。

例 1: 指定した時間範囲での REDO ログ・ファイルのマイニング

この例は、18-46 ページの「例 4: REDO ログ・ファイル内の LogMiner ディクショナリの使用」と同様ですが、REDO ログ・ファイルを明示的に指定しない点が異なります。この例では、REDO ログ・ファイルに対して抽出されたデータ・ディクショナリを使用するものとします。

手順 1 データ・ディクショナリの先頭を含む REDO ログ・ファイルのタイムスタンプを判別します。

```
SELECT NAME, FIRST_TIME FROM V$ARCHIVED_LOG
    WHERE SEQUENCE# = (SELECT MAX(SEQUENCE#) FROM V$ARCHIVED_LOG
    WHERE DICTIONARY_BEGIN = 'YES');

NAME                                           FIRST_TIME
---------------------------------------------  --------------------
/usr/oracle/data/db1arch_1_207_482701534.dbf  10-jan-2003 12:01:34
```

手順 2 その時点までに生成されたすべての REDO ログ・ファイルを表示します。

この手順は必須ではありませんが、手順 4 に示すとおり、CONTINUOUS_MINE オプションが予期したとおりに機能することを示すために含めています。

```
SELECT FILENAME name FROM V$LOGMNR_LOGS
   WHERE LOW_TIME > '10-jan-2003 12:01:34';

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
```

手順 3 LogMiner を起動します。

使用するディクショナリおよび COMMITTED_DATA_ONLY、PRINT_PRETTY_SQL、CONTINUOUS_MINE の各オプションを指定して、LogMiner を起動します。

```
EXECUTE DBMS_LOGMNR.START_LOGMNR(-
   STARTTIME => '10-jan-2003 12:01:34', -
     ENDTIME => SYSDATE, -
     OPTIONS => DBMS_LOGMNR.DICT_FROM_REDO_LOGS + -
                DBMS_LOGMNR.COMMITTED_DATA_ONLY + -
                DBMS_LOGMNR.PRINT_PRETTY_SQL + -
                    DBMS_LOGMNR.CONTINUOUS_MINE);
```

手順 4 V$LOGMNR_LOGS ビューを問い合せます。

この手順は、予期したとおり CONTINUOUS_MINE オプションを指定した DBMS_LOGMNR.START_LOGMNR プロシージャに、この時点までに生成されたすべての REDO ログ・ファイルが含まれることを示します。(この手順の出力を手順 2 の出力と比較してください。)

```
SELECT FILENAME name FROM V$LOGMNR_LOGS;

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
```

手順 5 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。)

問合せで返される再構築 SQL 文は、すべて正しく変換されていることに注意してください。

```
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-2003 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-2003 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-2003 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;
```

手順 6 LogMiner セッションを終了します。

```
EXECUTE DBMS_LOGMNR.END_LOGMNR();
```

例 2: 指定した SCN 範囲での REDO ログ・ファイルのマイニング

この例では、分析する SCN 範囲を指定する方法、およびその範囲を満たす REDO ログ・ファイルをマイニングする方法を示します。LogMiner を使用して、データ・ファイルで影響がまだ永続的になっていないすべてのコミット済 DML 文を表示できます。

この例では（他の例と異なり）、NLS_DATE_FORMAT パラメータを設定していないことに注意してください。

手順 1 最後のチェックポイントが実行された SCN を判別します。

```
SELECT CHECKPOINT_CHANGE#, CURRENT_SCN FROM V$DATABASE;

CHECKPOINT_CHANGE#  CURRENT_SCN
------------------  ---------------
          56453576         56454208
```

手順 2 LogMiner を起動し、CONTINUOUS_MINE オプションを指定します。

```
EXECUTE DBMS_LOGMNR.START_LOGMNR(-
   STARTSCN => 56453576, -
   ENDSCN   => 56454208, -
   OPTIONS  => DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG + -
               DBMS_LOGMNR.COMMITTED_DATA_ONLY + -
               DBMS_LOGMNR.PRINT_PRETTY_SQL + -
               DBMS_LOGMNR.CONTINUOUS_MINE);
```

手順 3 LogMiner によって追加されたアーカイブ済 REDO ログ・ファイルのリストを表示します。

```
SELECT FILENAME name, LOW_SCN, NEXT_SCN FROM V$LOGMNR_LOGS;

NAME                                          LOW_SCN   NEXT_SCN
--------------------------------------------  --------  --------
/usr/oracle/data/db1arch_1_215_482701534.dbf   56316771  56453579
```

LogMiner によって追加された REDO ログ・ファイルには、SCN 範囲全体が含まれないことに注意してください。CONTINUOUS_MINE オプションを指定した場合に DBMS_LOGMNR.START_LOGMNR プロシージャをコールすると、アーカイブ済 REDO ログ・ファイルのみが LogMiner によって追加されます。LogMiner は、問合せ実行中必要に応じて、オンライン REDO ログ・ファイルに含まれている残りの SCN 範囲を自動的に追加します。追加された REDO ログ・ファイルが生成された最新のアーカイブ済 REDO ログ・ファイルであるかどうかを確認するには、次の問合せを使用します。

```
SELECT NAME FROM V$ARCHIVED_LOG
   WHERE SEQUENCE# = (SELECT MAX(SEQUENCE#) FROM V$ARCHIVED_LOG);

NAME
-------------------------------------------
/usr/oracle/data/db1arch_1_215_482701534.dbf
```

手順 4 ユーザー表に対して行われた変更に対する V$LOGMNR_CONTENTS ビューを問い合せます。

次の問合せは、トランザクション 1.6.1911 と関連付けられた SET TRANSACTION READ WRITE 文および COMMIT 文は返しません。これらの文には、セグメント所有者（SEG_OWNER）が関連付けられていないためです。

デフォルトの NLS_DATE_FORMAT である DD-MON-RR が、DATE 型の MODIFIED_TIME 列の表示に使用されていることに注意してください。

```
SELECT SCN, (XIDUSN || '.' || XIDSLT || '.' ||  XIDSQN) as XID, SQL_REDO
    FROM V$LOGMNR_CONTENTS
    WHERE SEG_OWNER NOT IN ('SYS', 'SYSTEM');


SCN        XID        SQL_REDO
---------- ---------- -------------
56454198   1.6.1911   update "OE"."PRODUCT_INFORMATION"
                        set
                          "WARRANTY_PERIOD" = TO_YMINTERVAL('+05-00')
                        where
                          "PRODUCT_ID" = 2430 and
                          "WARRANTY_PERIOD" = TO_YMINTERVAL('+02-00') and
                          ROWID = 'AAAHTKAABAAAY9AAAC';

56454199   1.6.1911   insert into "OE"."PRODUCT_TRACKING"
                        values
                          "PRODUCT_ID" = 2430,
                          "MODIFIED_TIME" = TO_DATE('17-JAN-03', 'DD-MON-RR'),
                          "OLD_LIST_PRICE" = 175,
                          "OLD_WARRANTY_PERIOD" = TO_YMINTERVAL('+02-00');

56454204   1.6.1911    update "OE"."PRODUCT_INFORMATION"
                         set
                           "WARRANTY_PERIOD" = TO_YMINTERVAL('+05-00')
                         where
                           "PRODUCT_ID" = 2302 and
                           "WARRANTY_PERIOD" = TO_YMINTERVAL('+02-00') and
                           ROWID = 'AAAHTKAABAAAY9QAAA';
56454206   1.6.1911    insert into "OE"."PRODUCT_TRACKING"
                         values
                           "PRODUCT_ID" = 2302,
                           "MODIFIED_TIME" = TO_DATE('17-JAN-03', 'DD-MON-RR'),
                           "OLD_LIST_PRICE" = 150,
                           "OLD_WARRANTY_PERIOD" = TO_YMINTERVAL('+02-00');
```

手順 5 LogMiner セッションを終了します。

```
EXECUTE DBMS_LOGMNR.END_LOGMNR();
```

例 3: 問合せに将来の値を含める CONTINUOUS_MINE オプションの使用

所定の時刻になるまでまたは SCN に到達するまで問合せが終了しないように指定するには、DBMS_LOGMNR.START_LOGMNR プロシージャへのコールで、CONTINUOUS_MINE オプションを使用し、ENDTIME オプションまたは ENDSCAN オプションのいずれかを、将来の時刻またはまだ到達していない SCN 値に設定します。

この例では、現在から 5 時間後までの間に表 hr.employees に対して行われたすべての変更を監視するとします。また、オンライン・カタログ内のディクショナリを使用するとします。

手順 1 LogMiner を起動します。

```
EXECUTE DBMS_LOGMNR.START_LOGMNR(-
   STARTTIME => SYSDATE, -
   ENDTIME   => SYSDATE + 5/24, -
   OPTIONS   => DBMS_LOGMNR.CONTINUOUS_MINE  + -
                DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG);
```

手順 2 V$LOGMNR_CONTENTS ビューを問い合せます。

この選択操作は、分析する時間範囲（現在から 5 時間）後に生成される最初の REDO ログ・ファイル・レコードが検出されるまで完了しません。[Ctrl] キーを押しながら [C] キーを入力すると、完了前に選択操作を中止できます。

この例では、SET ARRAYSIZE 文を指定して、行が REDO ログ・ファイルに入力されたとおりに表示されるように設定しています。SET ARRAYSIZE 文を指定していない場合、SQL 内部バッファが一杯になるまで行は返されません。

```
SET ARRAYSIZE 1;
SELECT USERNAME AS usr, SQL_REDO FROM V$LOGMNR_CONTENTS
   WHERE  SEG_OWNER = 'HR' AND TABLE_NAME = 'EMPLOYEES';
```

手順 3 LogMiner セッションを終了します。

```
EXECUTE DBMS_LOGMNR.END_LOGMNR();
```

使用例

この項では、LogMiner の一般的な使用例を示します。次の例を示します。

- 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
- 使用例 2: LogMiner を使用した表アクセス統計の計算

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

この例では、joedevo というユーザーによって特定の時間範囲にデータベースに対して行われた変更をすべて表示する方法を示します。データベースに接続した後、次の手順を実行します。

1. 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、または LogMiner 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。LogMiner ディクショナリを作成する例は、18-36 ページの「LogMiner ディクショナリの抽出」を参照してください。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。

2. 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);
```

3. 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'));
```

4. 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;
```

5. 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( );
```

使用例 2: LogMiner を使用した表アクセス統計の計算

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。すでに LogMiner ディクショナリを作成済で、検索する REDO ログ・ファイルを追加してあるとします（前述の例と同様）。次の手順を実行します。

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-2003 08:30:00','DD-MON-YYYY HH:MI:SS'), -
   ENDTIME => TO_DATE('21-Jan-2003 08:45:00','DD-MON-YYYY HH:MI:SS'), -
   DICTFILENAME => '/usr/local/dict.ora');
```

2. 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、従来どおり名前に $ を含むシステム表がフィルタ処理で除外されます。)

```
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
```

   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、指定した 2 週間の間に cust.account 表が最も多く変更され、hr.employees 表と sys.donor 表は同じ期間の変更回数が最小でした。

4. 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( );
```

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

次の項では、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・ファイルのバージョンを示します。

- サポートされるデータ型と表記憶域属性
- サポートされないデータ型と表記憶域属性
- 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型と表記憶域属性

LogMiner は、次のデータ型と表記憶域属性をサポートします。

- CHAR
- NCHAR
- VARCHAR2 および VARCHAR
- NVARCHAR2
- NUMBER
- DATE
- TIMESTAMP
- TIMESTAMP WITH TIME ZONE
- TIMESTAMP WITH LOCAL TIME ZONE
- INTERVAL YEAR TO MONTH
- INTERVAL DAY TO SECOND

- RAW
- CLOB
- NCLOB
- BLOB
- LONG
- LONG RAW
- BINARY_FLOAT
- BINARY_DOUBLE
- 索引構成表（IOT）（オーバーフローまたは LOB 列がある索引構成表を含む）
- 関数ベースの索引
- CLOB 形式で格納される場合の XMLTYPE データ

マルチバイトの CLOB は、互換性が 10.1 以上に設定されているデータベースによって生成された REDO ログの場合のみサポートされます。

LOB データ型および LONG データ型は、互換性が 9.2.0.0 以上に設定されているデータベースによって生成された REDO ログの場合のみサポートされます。

オーバーフロー・セグメントまたは LOB 列を含まない索引構成表は、互換性が 10.0.0.0 以上に設定されているデータベースによって生成された REDO ログの場合のみサポートされます。オーバーフロー・セグメントまたは LOB 列を含む索引構成表は、互換性が 10.2.0.0 以上に設定されているデータベースによって生成された REDO ログの場合のみサポートされます。

サポートされないデータ型と表記憶域属性

LogMiner は、次のデータ型と表記憶域属性をサポートしません。

- BFILE データ型
- 単純およびネストされた抽象データ型（ADT）
- コレクション（ネストされた表および VARRAY）
- オブジェクト参照
- 表圧縮が使用されている表
- SecureFiles

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

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バージョンではなく、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、Oracle9*i* 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。

参照： 18-35 ページの「一般的な LogMiner セッションの手順」および 18-25 ページの「サプリメンタル・ロギング」

19

メタデータ API の使用

この章では、次の作業を行うためのメタデータ Application Programming Interface（API）について説明します。

- 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

この章の内容は、次のとおりです。

メタデータ API を使用する理由

Oracle Database の長期の使用に伴い、ディレクトリからメタデータを抽出し、そのメタデータに対し列の追加やデータ型の変更などを行い、さらに DDL に変換して元のまたは異なるデータベースのオブジェクトを再作成するために、ユーザーが独自のコードを開発する機会が多くなりました。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。

メタデータ API を使用すると、メタデータを抽出するためにコードを記述して管理する必要がなくなります。メタデータ API は、ディクショナリ・メタデータの抽出、操作および再送信に対する集中的な機能を提供します。また、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。

メタデータ 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。メタデータ API は、データ・ディクショナリ・ビューと同様、データベースのインストール時に `catproc.sql` を実行し、SQL スクリプトを起動してインストールします。インストール後は、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。

メタデータ API は、異なる Oracle のバージョン間で上位互換性があるため、データベースのバージョンを変更しても、ソース・コードの変更は必要ありません。1 つのバージョンで取得した XML 文書を、同じまたはそれ以上のバージョンの送信インタフェースで処理できます。たとえば、Oracle9*i* で取得した XML 文書は、Oracle Database 10*g* に送信できます。

メタデータ API の概要

メタデータ 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。

メタデータ API は、XML（Extensible Markup Language）および XSLT（Extensible Stylesheet Language Transformation）を使用します。XML が解析や変換が容易な汎用形式であるため、メタデータ API は、オブジェクト・メタデータを XML として表現します。メタデータ 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。

メタデータ API を使用して、メタデータのフェッチの際（または再送信の際）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メタデータ API の使用
- 取得したオブジェクトを再作成するためのメタデータ API の使用
- 異なるオブジェクト型のコレクションの取得

オブジェクトのメタデータを取得するためのメタデータ API の使用

メタデータ 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。

注意： 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、定義者権限パッケージ）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（または一般的に 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、実行者権限パッケージ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使用している SQL スクリプトおよびカスタマイズされたコードのかわりに、メタデータ 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手順

メタデータを取得する場合、メタデータ API のプロシージャを含む DBMS_METADATA PL/SQL パッケージを使用します。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。

参照：

- プログラム・インタフェースで使用する DBMS_METADATA プロシージャの詳細は、表 19-1 を参照してください。
- ブラウザ・インタフェースで使用する DBMS_METADATA プロシージャの詳細は、表 19-2 を参照してください。
- DBMS_METADATA パッケージの詳細は、『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』を参照してください。

例 19-1 では、メタデータ API 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。get_table_md というファンクションを作成するメタデータ API 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。

例 19-1 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 を作成するメタデータ API 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 19-2 に示すとおり、ブラウザ・インタフェースを使用しても同じ結果を得られます。

例 19-2 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;
```

結果は、例 19-1 の手順 5 と同様になります。

複数のオブジェクトの取得

例 19-1 では、存在するオブジェクトが 1 つのみであるため、FETCH_CLOB プロシージャは 1 度のみのコールでしたが、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;
```

例 19-3 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 19-3 複数のオブジェクトの取得

```
-- Because not all objects can be returned, they are stored in a table and queried at
the end.

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;
/
-- Execute the procedure.

EXECUTE get_tables_md;

-- See what was retrieved.

SET LONG 9000000
SET PAGES 0
SELECT * FROM my_metadata;
```

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 DLL 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 19-4 に、SET_TRANSFORM_PARAM プロシージャの使用方法を示します。

例 19-4 変換の条件指定

```
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;
/
```

SQL 文（SELECT get_table_md FROM DUAL;）を実行すると、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。メタデータ API では、複数の変換を指定できます。その場合、最初の出力が 2 番目の入力になり、2 番目の出力が 3 番目の入力になります。

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 19-5 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 19-5 XML 文書の変更

```
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;
/
```

SQL 文（SELECT remap_schema FROM DUAL;）を実行すると、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。

特定のメタデータ属性へのアクセス

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取得できますが、メタデータ 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 19-6 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 19-6 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

```
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
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 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);

  -- Set up the inner loop: fetch the INDEX objects.
  LOOP
    DDL := DBMS_METADATA.FETCH_CLOB(h2);

    -- When there are no more objects to be retrieved, FETCH_CLOB returns NULL.
    EXIT WHEN ddl IS NULL;

    -- Store the metadata in our table.
    INSERT INTO my_metadata(object_type, name, md)
      VALUES ('INDEX',NULL,ddl);
    COMMIT;
  END LOOP;
  DBMS_METADATA.CLOSE(h2);
 END LOOP;
 DBMS_METADATA.CLOSE(h1);
END;
/

-- Execute the procedure.

EXECUTE get_tables_and_indexes;

-- Perform a query to check what was retrieved.

SET LONG 9000000
SET PAGES 0
SELECT * FROM my_metadata;
```

取得したオブジェクトを再作成するためのメタデータ 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。

> **参照：** 送信インタフェースで使用する DBMS_METADATA プロシージャおよびファンクションの詳細は、19-20 ページの表 19-3 を参照してください。

例 19-7 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 19-7 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

```
-- Connect as a privileged user.

CONNECT system
Enter password: password

-- Create an invoker's rights package to hold the procedure
-- because access to objects in another schema requires the
-- SELECT_CATALOG_ROLE role.  In a definer's rights PL/SQL object
-- (such as a procedure or function), roles are disabled.

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;
/

-- Now try it: create a table in SCOTT...

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

-- ...and copy it to SYSTEM.
```

```
DROP TABLE my_example;
EXECUTE example_pkg.move_table('MY_EXAMPLE','SCOTT','SYSTEM');

-- Verify that it worked.

SELECT DBMS_METADATA.GET_DDL('TABLE','MY_EXAMPLE') FROM dual;
```

異なるオブジェクト型のコレクションの取得

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。メタデータ 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、いくつか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 19-8 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 19-8 異種オブジェクト型の取得方法

```
-- Create a table to store the retrieved objects.
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;
/
-- Execute the procedure.

EXECUTE get_schema_md;

-- See what was retrieved.

SET LONG 9000000
SET PAGESIZE 0
SELECT * FROM my_metadata;
```

この例では、次のことに注意してください。

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返りのフィルタ

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、メンバーの型の名前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 19-9 に、返されるオブジェクトに対して、より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 19-9 異種オブジェクト型の返りのフィルタ

```
-- Create a table to store the retrieved objects.
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;
/
-- Execute the procedure.

EXECUTE get_schema_md2;

-- See what was retrieved.

SET LONG 9000000
SET PAGESIZE 0
SELECT * FROM my_metadata;
```

メタデータ 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

この項では、メタデータ 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。

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。19-15 ページの「メタデータ 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、その例の目的は多くのプログラム・コールを示すことであり、その目的には、この方法が最適であるためです。
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. メタデータ 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。

   参照：『Oracle Database SecureFiles およびラージ・オブジェクト開発者ガイド』

メタデータ API の使用例

この項では、メタデータ API の使用例を示します。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。
- メタデータ 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。パスワードを入力するように要求されます。

   ```
   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
   ```

   この手順で行われる処理の詳細は、19-16 ページの「メタデータ API 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、手順 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、19-18 ページの「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。

メタデータ API の例で行われる処理

mddemo スクリプトを実行すると、次の手順が行われます。これらの手順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;
/
```

注意： 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。

GET_PAYROLL_TABLES プロシージャで生成される出力

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" ;
```

DBMS_METADATA プロシージャの要約

この項では、メタデータ 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。プロシージャの詳細は、『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』を参照してください。

表 19-1 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 19-1 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.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.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.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。

表 19-2 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、異種オブジェクト型はサポートされません。

表 19-2 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。

表 19-3 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 19-3 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

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。
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。

20

オリジナルのエクスポートおよびインポート

この章では、それぞれ exp および imp コマンドで起動する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・ユーティリティの使用方法について説明します。これらのユーティリティは、Oracle Database 10*g* から使用可能となる Oracle Data Pump Export および Import ユーティリティと区別するために、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・ユーティリティと呼びます。データ・ポンプのエクスポートおよびインポート・ユーティリティは、それぞれ expdp および impdp コマンドで起動します。

注意： オリジナルのエクスポート・ユーティリティは、Oracle Database 11*g* からは一般的な使用としてサポートされません。オリジナルのエクスポート・ユーティリティの使用が 11*g* でサポートされるのは、10*g* リリース 2（10.2）以前の下位のデータベース・バージョンに XMLType データを移行する場合のみです。したがって、次のようにオリジナルのエクスポートおよびインポート・ユーティリティが必要な場合を除いて、新しいデータ・ポンプ・エクスポートおよびインポート・ユーティリティの使用をお薦めします。

- 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場合。
- オリジナルのインポート・ユーティリティ（imp）を使用したインポートを予定しているファイルをエクスポートする場合。たとえば、Oracle Database 10*g* からデータをエクスポートして、そのデータを旧リリースのデータベースにインポートする場合です。

参照： 第 I 部「Oracle Data Pump」

この章の内容は、次のとおりです。

- インポート・セッションの例
- 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プラットフォーム間のデータベースの移動
- 警告、エラーおよび完了メッセージ
- 終了コードによる結果の検査と表示
- ネットワークに関する考慮点
- キャラクタ・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
- マテリアライズド・ビューおよびスナップショット
- トランスポータブル表領域
- 読取り専用表領域
- 表領域を削除する方法
- 表領域を再編成する方法
- 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
- エクスポートおよびインポートでのインスタンス親和性の使用
- データベースの断片化を解消する方法
- エクスポートおよびインポートでの記憶域パラメータの使用
- エクスポート固有の情報
- インポート固有の情報
- 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
- リリースおよびバージョンが異なるエクスポート・ユーティリティの使用方法

エクスポート・ユーティリティおよびインポート・ユーティリティとは

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と、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上にある Oracle Database 間で、データ・オブジェクトの転送が簡単にできます。

Oracle Database に対してエクスポートを実行すると、まずオブジェクト（表など）が抽出され、次にそれに関連するオブジェクト（索引、コメント、権限など）が抽出されます。抽出されたデータは、エクスポート・ファイルに書き込まれます。インポート・ユーティリティは、ダンプ・ファイルからオブジェクトの定義および表データを読み込みます。

エクスポート・ファイルは、通常、ディスクまたはテープにあるバイナリ形式の Oracle ダンプ・ファイルです。ダンプ・ファイルは、FTP を使用して別サイトに転送、または物理的に移送（テープの場合）できます。そのため、インポート・ユーティリティでエクスポート・ファイルを使用することで、ネットワークで接続されていないシステム上のデータベース間でデータを転送できます。また、標準のバックアップ手順以外のバックアップとしても使用できます。

エクスポート・ダンプ・ファイルは、Oracle のインポート・ユーティリティ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。

また、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。この場合は、インポート・ユーティリティの SHOW パラメータを使用します。詳細は、20-35 ページの「SHOW」を参照してください。

ASCII 固定形式ファイルまたは区切りファイルからデータをロードするには、SQL*Loader ユーティリティを使用します。

参照：

- 「リリースおよびバージョンが異なるエクスポート・ユーティリティの使用方法」（20-76 ページ）
- SQL*Loader ユーティリティの詳細は、第 II 部を参照してください。
- 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、オフライン・インスタンス化などの Oracle Advanced Replication の機能については、『Oracle Database アドバンスト・レプリケーション』を参照してください。

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前に

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前に、次のことを行う必要があります。

- catexp.sql または catalog.sql スクリプトの実行
- エクスポート・ファイルの書込み先であるディスクまたはテープに十分な記憶域があることの確認
- 必要なアクセス権限を所有していることの確認

catexp.sql または catalog.sql の実行

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には、データベースを作成するかまたは Oracle Database 10*g* へ移行した後で、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・ディレクト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

エクスポート操作に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・ファイルの書込み先であるディスク上またはテープ上に、十分な記憶領域があることを確認してください。十分な領域がない場合は、書込み失敗というエラーでエクスポートの処理が中止されます。

表サイズを使用して、必要な最大容量を見積もることができます。表サイズは、Oracle データ・ディクショナリの USER_SEGMENTS ビューで参照できます。次の問合せを行うと、すべての表に関するディスクの使用状況が表示されます。

```
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）列、VARRAY 列またはパーティション表に格納されているデータに使用されているディスク領域は含まれません。

> **参照：** 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。

エクスポートおよびインポート操作のアクセス権限の確認

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には、Oracle Database に対する 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する表をエクスポートする場合は、EXP_FULL_DATABASE ロールを使用可能にしておく必要があります。このロールは、すべてのデータベース管理者（DBA）に付与されています。

EXP_FULL_DATABASE ロールに含まれるシステム権限がない場合、別のユーザーのスキーマに格納されているオブジェクトをエクスポートすることはできません。シノニムを作成しても、別のユーザーのスキーマの表はエクスポートできません。

多数のシステム・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。

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。

エクスポート・ユーティリティおよびインポート・ユーティリティの起動

次のいずれかの方法を使用して、エクスポート・ユーティリティおよびインポート・ユーティリティの起動およびパラメータの指定ができます。

- コマンドライン
- 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。20-17 ページの「エクスポート・パラメータ」および 20-29 ページの「インポート・パラメータ」を参照してください。

注意： LOG_USER および PRIV_USER などのインポート・ジョブの属性は、エクスポート操作時の値に関係なく、インポートを実行するユーザーに常に設定されます。たとえば、ユーザー scott としてエクスポートを実行した後に、ユーザー system としてインポートすると、ダンプ・ファイル・セットのジョブはすべて LOG_USER および PRIV_USER = SYSTEM で作成されます。

この制限に対処するには、ジョブの所有者としてジョブをエクスポートおよびインポートする必要があります。

SYSDBA でのエクスポート・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・ユーティリティまたはインポート・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

コマンドライン

次の構文を使用して、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（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）。

```
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。この例では、使用してエクスポート・ユーティリティを起動する exp のかわりに、imp を使用してインポート・ユーティリティを起動できます。

パラメータ・ファイル

この項の情報は、エクスポート・ユーティリティおよびインポート・ユーティリティの両方に該当しますが、例ではエクスポート・コマンドの exp を使用します。

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。パラメータを 1 つのファイルに格納することによって、パラメータを簡単に変更または再利用できるため、エクスポート・ユーティリティの起動方法としてパラメータを 1 つのファイルに格納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。次に例を示します。

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
```

> **注意：** 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。

シャープ（#）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。シャープ（#）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、エクスポート・ユーティリティを次のコマンドで起動するとします。

```
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。

参照：

- エクスポート・パラメータの詳細は、20-17 ページの「エクスポート・パラメータ」を参照してください。
- 「インポート・パラメータ」（20-29 ページ）
- 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、20-54 ページの「Oracle Net を使用したエクスポートおよびインポート」を参照してください。

対話方式モード

プロンプトで各パラメータの値を入力する場合は、コマンドラインで exp または imp のいずれか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。対話方式のインタフェースを使用する場合は、Oracle Enterprise Manager のエクスポートまたはインポート・ウィザードの使用をお薦めします。

エクスポート・ユーティリティの対話方式を使用する際の制限事項

対話方式を使用する場合は、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、エクスポート対象とするすべてのユーザー名を入力するプロンプトが表示されます。ユーザー名のリストの入力後、[Enter] を押して現行のエクスポート・セッションを開始します。
- 表モードでは、スキーマの接頭辞を指定しないと、エクスポート実行者のスキーマ、または現行のセッション中に最後にエクスポートされた表が格納されているスキーマがデフォルトの値になります。

  たとえば、特権ユーザーである beth が表モードでエクスポートを実行している場合、他のユーザーのスキーマが指定されるまでは、すべての表は beth のスキーマにあると判断されます。他のユーザーのスキーマに属する表をエクスポートできるのは、特権ユーザー（EXP_FULL_DATABASE ロールを持つユーザー）のみです。
- 「エクスポートする表」という入力要求のプロンプトに対して表を指定しないと、エクスポート・ユーティリティは終了します。

オンライン・ヘルプの利用

エクスポート・ユーティリティおよびインポート・ユーティリティでは、オンライン・ヘルプが提供されます。エクスポート・ユーティリティのヘルプを起動するにはコマンドラインに exp help=y を入力します。インポート・ユーティリティのヘルプを起動するには imp help=y を入力します。

オブジェクトをスキーマにインポートする方法

表 20-1 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 20-1 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）
表データ	INSERT TABLE（オブジェクト）

表 20-1 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（続き）

オブジェクト	必要な権限（該当する場合の権限タイプ）
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）

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 20-2 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 20-2 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 With Admin Option および SYSTEM 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。エクスポート・ファイルが全体エクスポートの場合にパラメータ 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

処理上の制限事項

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データを処理する場合、次の制限が適用されます。

- Enterprise JavaBeans（EJB）を使用して作成された Java クラス、リソースおよびプロシージャは、エクスポート・ファイルには書き込まれません。
- RELY キーワードを使用して変更された制約は、エクスポートされると RELY 属性が失われます。
- 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。

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・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。エクスポート・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。

たとえば、emp 表に dept 表に対する参照整合性制約が指定されて、emp 表が最初にインポートされた場合、この制約が使用可能になっていると、まだ dept にインポートされていない部門を参照するすべての emp 行が拒否されます。

データが既存の表にインポートされた場合でも、インポートの順序によっては参照整合性のエラーが発生することがあります。前述の状況で、emp 表がすでに存在していて、参照整合性制約が使用可能になっている場合、多くの行が拒否されることがあります。

その表自体への参照整合性制約がある場合にも、同様の状況が発生します。たとえば、emp 表において scott の管理者が drake で、drake の行がまだロードされていない場合、インポートが終了した場合には有効になるとしても、scott の行はロードされません。

注意： このような理由から、既存の表にインポートする場合は、参照制約を使用禁止にすることをお薦めします。インポートの完了後、再度制約を使用可能にできます。

既存の表へのインポート

この項では、既存の表にデータをインポートする場合に考慮すべき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

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
- 互換性がないデータ型への列のデータ型の変更（たとえば、LONG 型から NUMBER 型への変更）
- 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

注意： 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。

参照制約を使用禁止にする方法

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・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。

手動によるインポートの順序付け

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。

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、次の手順に従います。

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。

注意： 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。SYS 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、SYS トリガーはインポートの後に再作成することをお薦めします。

エクスポート・モードおよびインポート・モード

エクスポート・ユーティリティおよびインポート・ユーティリティには、次の 4 種類の操作モードがあります。

- 全体エクスポートおよび全体インポート : 全データベースのエクスポートおよびインポートを実行します。EXP_FULL_DATABASE および 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）をエクスポートおよび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。エクスポート・ユーティリティでこのモードを指定するには OWNER パラメータを使用します。インポート・ユーティリティで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エクスポートおよび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。表にスキーマ名が指定されていない場合、エクスポート・ユーティリティではエクスポート実行者のスキーマ名がデフォルト値として採用されます。TABLES パラメータを使用してこのモードを指定します。

表 20-3 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。

注意 : 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: Nonexistent type メソッドの使用方法が正しくありません。

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。

従来型パス・エクスポートまたはダイレクト・パス・エクスポートは、表領域モード以外のモードで使用できます。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の詳細は、20-61 ページの「従来型パス・エクスポートおよびダイレクト・パス・エクスポート」を参照してください。

参照 :

- 『Oracle Database 管理者ガイド』
- トランスポータブル表領域の機能の詳細は、『Oracle Database 概要』を参照してください。

表 20-3 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

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[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
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

表 20-3 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（続き）

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
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
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

表 20-3 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（続き）

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
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[2]	あり[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

[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。

[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。

[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。SYS 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、SYS トリガーは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のエクスポートは、次のように実行できます。

- **表レベル・エクスポート :** 指定された表の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をエクスポートします。

どのモードの場合も、パーティション・データは、インポート時にパーティション単位またはサブパーティション単位で選択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

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は、索引およびその他の表に依存するオブジェクトとともに全体的にエクスポートされます。パーティション表の場合は、すべてのパーティションおよびサブパーティションもエクスポートされます。ダイレクト・パス・エクスポートおよび従来型パス・エクスポートの両方とも、この点は同じです。表レベル・エクスポートは、いずれのエクスポート・モードでも実行できます。

パーティション・レベル・エクスポート

パーティション・レベル・エクスポートでは、表の 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指定して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・エクスポートおよびパーティション・レベル・エクスポートの指定方法の詳細は、20-26 ページの「TABLES」を参照してください。

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のインポートは、次のように実行できます。

- 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。

既存の表にデータをロードするときは、パラメータ IGNORE=y を指定します。詳細は、20-32 ページの「IGNORE」を参照してください。

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。

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マ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。

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。

注意： 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。

エクスポート・パラメータ

この項では、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

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されるバッファのサイズをバイト単位で指定します。これにより、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最大行数が決まります。バッファ・サイズの計算には、次の計算式を使用してください。

```
buffer_size = rows_in_array * maximum_row_size
```

0（ゼロ）を指定すると、1 回に 1 行のみフェッチされます。

LOB、LONG、BFILE、REF、ROWID、LOGICAL ROWID または DATE 型の列が含まれる表は、1 回に 1 行ずつフェッチされます。

注意： BUFFER パラメータを使用できるのは、従来型パス・エクスポートの場合のみです。ダイレクト・パス・エクスポートの場合は、このパラメータの指定による影響はありません。ダイレクト・パス・エクスポートでは、エクスポート・ファイルへの書込みに使用するバッファのサイズは、RECORDLENGTH パラメータで指定します。

バッファ・サイズの計算

ここでは、バッファ・サイズの計算方法の例を示します。

次の表を作成します。

```
CREATE TABLE sample (name varchar(30), weight number);
```

name 列の最大サイズは 30 で、インジケータ用に 2 バイトをプラスします。weight 列の最大サイズは 22（Oracle の NUMBER データ型の内部表現のサイズ）で、インジケータ用に 2 バイトをプラスします。

したがって、行の最大サイズは 56（30+2+22+2）になります。

100 行単位で配列の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。

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・ユーティリティによる、表データの初期エクステントの管理方法を指定します。

デフォルトの COMPRESS=y を指定すると、インポート時に表データを 1 つの初期エクステントに整理統合するためのフラグが付きます。エクステント・サイズが大きい場合（たとえば、PCTINCREASE パラメータが指定されている場合）、データの格納に必要以上の領域が割り当てられます。

COMPRESS=n を指定すると、エクスポート・ユーティリティは、初期エクステントのサイズおよび第 2 エクステントのサイズが指定されている現行の記憶域パラメータを使用します。パラメータの値は、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になります。たとえば、表が大きくなった場合、および PCTINCREASE パラメータに 0（ゼロ）以外の値が指定されている場合は、第 2 エクステントのサイズが変更されることがあります。

COMPRESS パラメータはビットマップ表領域では機能しません。

注意： 実際に整理統合が実行されるのはインポート時ですが、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。記憶域パラメータなどのデータ定義は、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生成されるためです。したがって、エクスポート時に COMPRESS=y を指定した場合、そのデータは整理統合形式でのみインポートできます。

注意： 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のサイズおよび第 2 エクステントのサイズの値が使用されます。

CONSISTENT

デフォルト : n

エクスポート・ユーティリティによって読み込まれたデータのある時点における一貫性を維持し、exp コマンドの実行中に変更されないようにするために、SET TRANSACTION READ ONLY 文を使用するかどうかを指定します。エクスポート開始後に、他のアプリケーションによってそのデータベースが更新されることがわかっている場合は、CONSISTENT=y を指定してください。

CONSISTENT=n を使用すると、通常、各表は 1 つのトランザクションでエクスポートされます。ただし、表の内部にネストした表がある場合は、外部表および各内部表は別のトランザクションでエクスポートされます。パーティション表の場合は、パーティションごとに別のトランザクションでエクスポートされます。

したがって、ネストした表やパーティション表が別のアプリケーションによって更新中の場合、エクスポートされるデータが一貫性を維持できないことがあります。このような危険性をできるだけ低くするために、これらの表のエクスポートは、更新中でないときに実行してください。

表 20-4 に、2 人のユーザーによるイベントの順序を示します。user1 が表のパーティションをエクスポートし、user2 が同じ表のデータを更新するとします。

表 20-4 2 人のユーザーによる更新時のイベントの順序

時系列	user1	user2
1	TAB: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	TAB:P2 を更新 TAB:P1 を更新 トランザクションをコミット
3	TAB:P1 のエクスポートを終了	アクティビティなし
4	TAB: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 を指定すると、user2 によって実行された更新はエクスポート・ファイルには書き込まれません。

エクスポートで CONSISTENT=n を指定すると、TAB:P1 に対する更新はエクスポート・ファイルには書き込まれません。ただし、TAB:P2 に対する更新は TAB:P2 のエクスポート開始前にコミットされているため、更新がエクスポート・ファイルに書き込まれます。その結果、user2 のトランザクションは部分的にのみエクスポート・ファイルに書き込まれるため、エクスポート・ファイルではデータの一貫性を維持できません。

CONSISTENT=y を指定しているときに更新量が多いと、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、ロールバック・セグメントをスキャンしてコミットされていないトランザクションを探すため、各表のエクスポートにかかる時間が長くなります。

CONSISTENT=y を指定する場合は、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とき、または AS SYSDBA を使用しているとき（あるいはその両方）に実行するエクスポートでは、
- CONSISTENT=y はサポートされません。
- メタデータのエクスポートには、再帰的 SQL 内の SYS スキーマを使用する必要があります。この場合、CONSISTENT=y を指定しても無視されます。CONSISTENT=y を選択したエクスポートの処理中は、メタデータを変更しないことをお薦めします。
- エクスポートに必要な時間および領域を最小にするには、一貫性が要求される表をまとめてエクスポートし、残りの表は別途エクスポートします。たとえば、CONSISTENT=y を指定して emp 表および dept 表をまとめてエクスポートした後、残りの表をエクスポートします。
- 「スナップショットが古すぎます」というエラーは、ロールバック領域を使い果たし、コミットされたトランザクションの領域が新しいトランザクションのために再利用されたときに発生します。ロールバック・セグメント領域を再利用することによって、最小の領域でデータベースの整合性を維持できますが、読取り一貫性のためのイメージを維持する時間が制限されます。

  コミット済トランザクションが上書きされた後で、データベースの読取り一貫性ビューを維持するために、上書きによって消去された情報が必要になった場合、「スナップショットが古すぎます」という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できるだけ短くします（エクスポートするオブジェクト数を制限し、可能な場合はデータベースのトランザクション率を低くします）。また、ロールバック・セグメントをできるだけ大きく設定しておきます。

注意： 将来のリリースの Oracle Database では、ロールバック・セグメントは廃止されます。かわりに、自動 UNDO 管理を使用することをお薦めします。

参照： 「OBJECT_CONSISTENT」（20-23 ページ）

CONSTRAINTS

デフォルト：y

表制約をエクスポートするかどうかを指定します。

DIRECT

デフォルト：n

ダイレクト・パス・エクスポートの使用を指定します。

DIRECT=y を指定すると、エクスポート・ユーティリティが、（バッファを調べ）SQL コマンド処理レイヤーをバイパスしてデータを直接読み込み、データを抽出します。この方法は、従来型パス・エクスポートに比べて非常に高速です。

セキュリティおよびパフォーマンスに関する考慮点を含むダイレクト・パス・エクスポートの詳細は、20-61 ページの「ダイレクト・パス・エクスポートの起動」を参照してください。

FEEDBACK

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設定できません。

FILE

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、別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、複数ファイルのエクスポートをサポートしているため、複数のファイル名を指定できます。次に例を示します。

```
exp scott/tiger FILE = dat1.dmp, dat2.dmp, dat3.dmp FILESIZE=2048
```

FILESIZE に指定した最大値までエクスポートが実行されると、現行のファイルへの書込みは中止され、FILE パラメータで次のファイル名として指定した名前のエクスポート・ファイルがオープンされます。エクスポートが完了するまで、または FILESIZE の最大値に再度到達するまでエクスポートが続行されます。指定したエクスポート・ファイル名が十分でないためにエクスポートを完了できない場合は、ファイル名を追加するためのプロンプトが表示されます。

FILESIZE

デフォルト : データは、表 20-5 に示す最大サイズに到達するまで、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・ユーティリティによって、それぞれのダンプ・ファイルに指定したバイト数のみ書き込まれます。

エクスポートで書き込まれるデータの量が FILESIZE に指定した最大値を超えると、次のエクスポート・ファイルの名前が FILE パラメータで指定されるか（20-20 ページの「FILE」を参照）、または FILE パラメータで指定したすべての名前が使用されている場合は、新しいエクスポート・ファイル名を指定するためのプロンプトが表示されます。FILESIZE の値を指定しない場合（0 の指定は、FILESIZE の指定なしを意味する）は、FILE パラメータで指定したファイルの数にかかわらず、1 つのファイルのみに書き込まれます。

> **注意：** 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場合は、エクスポートが終了するため、十分なディスク領域を確保した後で再度エクスポートする必要があります。

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 20-5 に、ご使用のオペレーティング・システムおよび Oracle Database のリリースによって異なるダンプ・ファイルの最大サイズを示します。

表 20-5 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
すべて	8.1.5 より前	2GB
32 ビット・システム	8.1.5	2GB
64 ビット・システム	8.1.5 以上	無制限
32 ビット・ファイルを扱う 32 ビット・システム	すべて	2GB
64 ビット・ファイルを扱う 32 ビット・システム	8.1.6 以上	無制限

ファイルに格納可能な最大値は、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。また、エクスポートで指定するファイル・サイズが、インポートを実行するシステムでサポートされていることも確認してください。

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）。

FLASHBACK_SCN

デフォルト : なし

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、フラッシュバックを使用可能にします。エクスポート・オペレーションは、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。

> **参照 :** フラッシュバックの使用方法の詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。

次に、SCN の指定例を示します。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
```

FLASHBACK_TIME

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・ユーティリティによって、指定されたタイムスタンプに一番近い SCN が検索されます。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・オペレーションは、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間を指定できます。次のいずれかの方法で指定できます。

```
> exp FILE=exp.dmp FLASHBACK_TIME="TIMESTAMP '2002-05-01 11:00:00'"

> exp FILE=exp.dmp FLASHBACK_TIME="TO_TIMESTAMP('12-02-2001 14:35:00', 'DD-MM-YYYY HH24:MI:SS')"
```

また、下位互換性を維持するために、次の例に示す古い形式も使用可能です。

```
> exp FILE=exp.dmp FLASHBACK_TIME="'2002-05-01 11:00:00'"
```

> **参照 :**
>
> - フラッシュバックの使用方法の詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。
> - DBMS_FLASHBACK パッケージの詳細は、『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』を参照してください。

FULL

デフォルト : n

エクスポートが、全データベース・モードのエクスポートであること（データベース全体のエクスポート）を示します。全データベース・モードでエクスポートするには、FULL=y を指定します。このモードでのエクスポートには、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。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。SYS 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、SYS トリガーは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・ファイルは、他のデータベースに含まれると破損します。エクスポートされたデータベースが同じシステムにある場合、そのデータ・ファイルはインポート先のデータベースで再利用されるため、特に注意が必要です。
  - 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。

GRANTS

デフォルト : y

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、エクスポート・モードが全データベース・モードかユーザー・モードかによって異なります。全データベース・モードでは、表に対するすべての権限がエクスポートされます。一方、ユーザー・モードでは、表の所有者が付与した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。

HELP

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。ヘルプを起動するには、コマンドラインで exp help=y を入力します。

INDEXES

デフォルト : y

索引をエクスポートするかどうかを指定します。

LOG

デフォルト : なし

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。

このパラメータを指定すると、メッセージはログ・ファイルに記録されるとともに端末画面に表示されます。

OBJECT_CONSISTENT

デフォルト : n

エクスポート・ユーティリティによって読み込まれたデータのある時点における一貫性を維持し、エクスポート中に変更されないようにするために、SET TRANSACTION READ ONLY 文を使用するかどうかを指定します。OBJECT_CONSISTENT を y に設定した場合、オブジェクトがパーティション化されている場合でも、読取り専用トランザクションに各オブジェクトをエクスポートできます。これに対し、CONSISTENT パラメータを使用する場合、読取り専用トランザクションは 1 つのみです。

参照 : 「CONSISTENT」（20-18 ページ）

OWNER

デフォルト : なし

ユーザー・モード・エクスポートでエクスポートすることを示します。エクスポートの対象となるオブジェクトを所有するユーザー名のリストを表示します。データベース管理者（DBA）ユーザーがエクスポートを起動している場合は、複数のユーザーがリストされる場合があります。

ユーザー・モードのエクスポートは、1 人以上のデータベース・ユーザーのバックアップに使用可能です。たとえば、削除されたユーザーの表を、DBA が一定の期間バックアップを取る場合などに有効です。ユーザー・モードは、ユーザーが自分自身のデータのバックアップを取る場合や、ある所有者のオブジェクトを別の所有者に移す場合にも適しています。

PARFILE

デフォルト : なし

エクスポート・パラメータのリストが格納されているファイルのファイル名を指定します。パラメータ・ファイルの使用方法の詳細は、20-5 ページの「エクスポート・ユーティリティおよびインポート・ユーティリティの起動」を参照してください。

QUERY

デフォルト : なし

表モード・エクスポートの実行時に、一連の表から行のサブセットを選択できるようにします。QUERY 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（または表パーティション）に適用される SQL の SELECT 文の WHERE 句を含む文字列です。

たとえば、ユーザー scott が、職種が SALESMAN で、給与が 1600 より少ない従業員のみをエクスポートする場合は、次のように指定します（この例は UNIX ベースの場合）。

```
exp scott/tiger TABLES=emp QUERY=\"WHERE job=\'SALESMAN\' and sal \<1600\"
```

> **注意：** 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、すべての文字列 WHERE job=\'SALESMAN\' および sal\<1600 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・システムの予約文字も、エスケープ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、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。

この問合せの実行時、エクスポート・ユーティリティによって、次のように SQL の 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 パラメータでリストされたすべての表（または表パーティション）に適用されます。たとえば、次の文では、問合せと一致する emp および bonus の両方の行がアンロードされます。

```
exp scott/tiger TABLES=emp,bonus QUERY=\"WHERE job=\'SALESMAN\' and sal\<1600\"
```

また、エクスポート・ユーティリティによって次の 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;
```

QUERY 句で指定された列が表にない場合は、エラー・メッセージが表示されて、行はエクスポートされません。

QUERY パラメータ使用時の制限事項

- QUERY パラメータは、全データベース・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では指定できません。
- QUERY パラメータは、すべての指定した表に適用される必要があります。
- 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、内部にネストした表を含む表では指定できません。
- 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。

RECORDLENGTH

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。

ファイル・レコードの長さをバイト単位で指定します。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は、ご使用のプラットフォーム固有のバッファ・サイズのデフォルト値が採用されます。

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）。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。

> **注意：** このパラメータは、エクスポートの I/O バッファのサイズ指定に使用できます。

RESUMABLE

デフォルト: n

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。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=y に設定する必要があります。

参照:

- 『Oracle Database 概要』
- 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。

RESUMABLE_NAME

デフォルト: 'User USERNAME (USERID), Session SESSIONID, Instance INSTANCEID'

再開可能な文を指定します。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。

RESUMABLE_TIMEOUT

デフォルト: 7200 秒（2 時間）

エラー修正に必要な時間を指定します。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。

ROWS

デフォルト: y

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

デフォルト: ESTIMATE

エクスポートされたデータのインポート時に生成されるデータベース・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。インポート・パラメータの詳細は、20-36 ページの「STATISTICS」および 20-68 ページの「統計情報のインポート」を参照してください。

エクスポート・ユーティリティによって、ANALYZE 文および計算済統計情報がエクスポート・ファイルに書き込まれ、統計情報が再生成される場合もあります。

ただし、システムによって生成された名前を持つ列が表に含まれる場合、計算済オプティマイザ統計は、エクスポート時に使用されません。

次の場合は、エクスポート時に、計算済オプティマイザ統計に問題ありのフラグが付きます。

- エクスポート中に、行エラーが発生した場合
- クライアント・キャラクタ・セットまたは NCHAR キャラクタ・セットが、サーバー・キャラクタ・セットまたはサーバーの NCHAR と一致しない場合

- QUERY 句を指定した場合
- 一部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る場合

注意： ROWS=n を指定しても、計算済統計情報は、エクスポート・ファイルから削除されません。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。

参照：『Oracle Database 概要』

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。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リストとして指定します。表名を指定するときに、次の項目を指定できます。

- *schemaname* には、表またはパーティションのエクスポート元のユーザーのスキーマ名を指定します。スキーマ名を指定しない場合、エクスポート実行者のスキーマがデフォルトで使用されます。ORDSYS、MDSYS、CTXSYS、LBACSYS および 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* には、エクスポートされる表名を指定します。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と表名の 0 個以上の文字を一致させます。関連するスキーマにある、指定されたパターンと一致するすべての表は、それぞれの表名がパラメータで明示的に指定された場合と同様に、選択されてエクスポートされます。
- *partition_name* は、エクスポートがパーティション・レベル・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、表の 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指定して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・エクスポートの例は、20-44 ページの「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

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シャープ（#）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、表名にシャープ（#）記号を使用すると、シャープ（#）記号より右側の文字はコメントとして解釈されます。

  たとえば、パラメータ・ファイルに次のコマンドラインが記述されている場合、emp# の右側は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、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)
```

> **注意：** 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。

TABLESPACES

デフォルト : なし

TABLESPACES パラメータを使用して、表領域のすべての表がエクスポート・ダンプ・ファイルへエクスポートされるように指定します。これには、表領域のリストに含まれるすべての表およびパーティションを持つすべての表が含まれます。索引は、格納されている場所にかかわらず、表とともにエクスポートされます。

TABLESPACES を使用して表領域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 を TRANSPORT_TABLESPACE=y と組み合せて使用すると、表領域の一部のリストをデータベースからエクスポート・ファイルへエクスポートするように指定できます。

TRANSPORT_TABLESPACE

デフォルト : n

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。

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。

注意： 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。

参照：

- 「トランスポータブル表領域」（20-57 ページ）
- 『Oracle Database 管理者ガイド』
- 『Oracle Database 概要』

TRIGGERS

デフォルト：y

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

デフォルト：n

y を設定すると、エクスポート・ユーティリティによって、リカバリ・セット（リカバリの対象となる表領域のセット）にリカバリ・セット外のオブジェクト（またはその逆）に対する依存関係（特に IN ポインタ）がないことが確認されます。

USERID（ユーザー名 / パスワード）

デフォルト：なし

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。パスワードを指定しないと、入力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。

参照：

- 『Oracle Database Heterogeneous Connectivity 管理者ガイド』
- 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。

VOLSIZE

デフォルト：なし

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。

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）。

インポート・パラメータ

この項では、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

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 が含まれている表および LONG、BFILE、REF、ROWID、UROWID または TIMESTAMP の列が含まれている表は、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。

注意： 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。

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 または UROWID の列が含まれている表は、配列単位では挿入できません。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。

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。

参照： 「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」（20-73 ページ）

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。

DATAFILES

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。

参照 : 「TRANSPORT_TABLESPACE」（20-40 ページ）

DESTROY

デフォルト : n

データベースを構成している既存のデータ・ファイルを再利用するかどうかを指定します。DESTROY=y を指定して SQL の CREATE TABLESPACE 文の datafile 句に REUSE オプションを付けることによって、元のデータベースのデータ・ファイルの内容を削除した後でこれらのファイルを再利用します。

エクスポート・ファイルには、各表領域で使用されるデータ・ファイル名が格納されています。DESTROY=y を指定し、（テストや他の目的で）同一システム上に 2 番目のデータベースを作成しようとすると、表領域の作成時に、元のデータベースのデータ・ファイルが上書きされます。このような場合、デフォルトの DESTROY=n を指定すると、表領域作成時にすでにデータ・ファイルがある場合は、エラーが返されます。また、元のデータベースへインポートする必要がある場合は、IGNORE=y を指定し、既存のデータ・ファイルを置換せずに追加します。

注意 : 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。

FEEDBACK

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。

FILE

デフォルト : expdat.dmp

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、複数ファイルのエクスポートをサポートしているため（次の FILESIZE パラメータの説明を参照）、複数のインポート・ファイル名が必要な場合もあります。次に例を示します。

```
imp scott/tiger IGNORE=y FILE = dat1.dmp, dat2.dmp, dat3.dmp FILESIZE=2048
```

ユーザー自身がエクスポートしたファイルでなくても指定できますが、そのファイルに対する読取り権限が必要です。他のユーザーがエクスポートしたエクスポート・ファイルの場合は、IMP_FULL_DATABASE ロールが必要です。

FILESIZE

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・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。エクスポート 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。インポート・ユーティリティでは、エクスポートの最大ダンプ・ファイル・サイズを指定するために、インポート FILESIZE パラメータを使用する必要があります。

> **注意：** 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）。

ダンプ・ファイルの最大サイズの詳細は、表 20-5 を参照してください。

FROMUSER

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（20-39 ページの「TOUSER」を参照）。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。

> **注意：** 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。

FULL

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。

GRANTS

デフォルト : y

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)。

> **注意 :** 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。

HELP

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。ヘルプを起動するには、コマンドラインで imp HELP=y を入力します。

IGNORE

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。

注意： 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。

INDEXES

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。

INDEXFILE

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、次の手順を実行します。

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。

   （この手順でデータベース・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。

LOG

デフォルト : なし

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。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。

PARFILE

デフォルト : なし

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。パラメータ・ファイルの使用方法の詳細は、20-5 ページの「パラメータ・ファイル」を参照してください。

RECORDLENGTH

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。

ファイル・レコードの長さをバイト単位で指定します。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 のデフォルト値が採用されます。

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。

RESUMABLE

デフォルト : n

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。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=y に設定する必要があります。

参照 :

- 『Oracle Database 概要』
- 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。

RESUMABLE_NAME

デフォルト : 'User USERNAME (USERID), Session SESSIONID, Instance INSTANCEID'

再開可能な文を指定します。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。

RESUMABLE_TIMEOUT

デフォルト : 7200 秒（2 時間）

エラー修正に必要な時間を指定します。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。

ROWS

デフォルト : y

表のデータ行をインポートするかどうかを指定します。

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。

SHOW

デフォルト : n

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。

SKIP_UNUSABLE_INDEXES

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

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。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・ユーティリティのコマンドラインで SKIP_UNUSABLE_INDEXES の値を指定した場合、この値は初期化パラメータ・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。

注意 : 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。

参照 : 『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。

STATISTICS

デフォルト：ALWAYS

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。

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。

参照：

- 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」（20-68 ページ）

STREAMS_CONFIGURATION

デフォルト：y

エクスポート・ダンプ・ファイル内に存在する一般的な Streams メタデータをインポートするかどうかを指定します。

参照：『Oracle Streams レプリケーション管理者ガイド』

STREAMS_INSTANTIATION

デフォルト：n

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。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。

参照：『Oracle Streams レプリケーション管理者ガイド』

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（20-12 ページの表 20-3 を参照）。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、20-51 ページの「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。

注意： 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

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シャープ（#）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、表名にシャープ（#）記号を使用すると、シャープ（#）記号より右側の文字はコメントとして解釈されます。

  たとえば、パラメータ・ファイルに次のコマンドラインが記述されている場合、emp# の右側は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、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)
  ```

  > **注意：** 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（$）やシャープ（#）（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。

TABLESPACES

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、20-40 ページの「TRANSPORT_TABLESPACE」を参照してください。

TOID_NOVALIDATE

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。

構文は次のようになります。

```
TOID_NOVALIDATE=([schemaname.]typename [, ...])
```

次に例を示します。

```
imp scott/tiger TABLES=jobs TOID_NOVALIDATE=typ1
imp scott/tiger 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"
[...]
```

> **注意：** 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。

TOUSER

デフォルト : なし

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。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。次に例を示します。

```
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、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。

> **参照：** FROMUSER および TOUSER を使用する場合の制限の詳細は、20-31 ページの「FROMUSER」を参照してください。

TRANSPORT_TABLESPACE

デフォルト : n

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。

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。

注意： 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。

TTS_OWNERS

デフォルト : なし

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。

詳細は、20-40 ページの「TRANSPORT_TABLESPACE」を参照してください。

USERID（ユーザー名 / パスワード）

デフォルト : なし

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。

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。

> **参照：**
>
> - 『Oracle Database Heterogeneous Connectivity 管理者ガイド』
> - 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。

VOLSIZE

デフォルト : なし

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。

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）。

エクスポート・セッションの例

この項では、次のタイプのエクスポート・セッションの例を示します。

- 全データベース・モードでのエクスポート・セッションの例
- ユーザー・モードでのエクスポート・セッションの例
- 表モードでのエクスポート・セッションの例
- パーティション・レベル・エクスポートでのエクスポート・セッションの例

各例で、コマンドライン方式およびパラメータ・ファイル方式の両方の使用方法を示します。一部の例では、例の出力を含めるには長すぎた部分を示すために縦の省略記号を使用しています。

全データベース・モードでのエクスポート・セッションの例

全データベース・モードでエクスポートを実行できるのは、DBA ロールまたは EXP_FULL_DATABASE ロールを持つユーザーのみです。この例では、すべての GRANTS（付与されている権限）およびすべてのデータとともにデータベース全体をファイル dba.dmp に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=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、データベースが完全にエクスポートされると出力されます。最終の完了メッセージは、エクスポートが警告もなく正常に完了した場合に返されます。

ユーザー・モードでのエクスポート・セッションの例

ユーザー・モードのエクスポートは、1 人以上のデータベース・ユーザーのバックアップに使用可能です。たとえば、削除されたユーザーの表を、DBA が一定の期間バックアップを取る場合などに有効です。ユーザー・モードは、ユーザーが自分自身のデータのバックアップを取る場合や、ある所有者のオブジェクトを別の所有者に移す場合にも適しています。次の例では、ユーザー scott が自分が所有する表をエクスポートします。

パラメータ・ファイル方式

```
> exp scott/tiger 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/tiger 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.
```

表モードでのエクスポート・セッションの例

表モードでは、表データまたは表定義のみをエクスポートできます。(エクスポートする行がない場合は、CREATE TABLE 文がエクスポート・ファイルに書き込まれます。このとき、権限付与および索引が指定されると、これらもエクスポート・ファイルに書き込まれます。)

EXP_FULL_DATABASE ロールを持つユーザーは、表モードで TABLES=schemaname.tablename を指定することによって、どのユーザーのスキーマに属する表でもエクスポートできます。

schemaname を指定しない場合、エクスポート実行者のスキーマ名がデフォルト値として採用されます。次の例では、a 表および c 表に対して、SYSTEM スキーマがデフォルトとして採用されます。

```
> exp TABLES=(a, scott.b, c, mary.d)
```

EXP_FULL_DATABASE ロールを持つユーザーは、他のユーザーが所有する依存オブジェクトのエクスポートも実行できます。権限を持たないユーザーは、そのユーザー自身が所有し、指定した表の依存オブジェクトのみをエクスポートできます。

表モードのエクスポートには、クラスタ定義がありません。このため、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、表モードは、表の非クラスタ化に使用できます。

例 1: DBA による 2 人のユーザーの表のエクスポート

この例では、DBA が 2 人のユーザーの表を指定して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.
```

例 2: ユーザーによる自分が所有する表のエクスポート

この例では、ユーザー blake が自分が所有している表の中から選択した表をエクスポートします。

パラメータ・ファイル方式

```
> exp blake/paper 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/paper 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.
```

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 およびユーザー 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.
```

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・エクスポートでは、エクスポートの対象を表のパーティションおよびサブパーティション単位で指定できます。

例 1: パーティションを指定しない表のエクスポート

従業員名にパーティションが指定されている emp 表があるとします。m および z の 2 つのパーティションがあります。次の例に示すように、パーティションを指定しないでエクスポートを実行すると、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/tiger 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/tiger 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.
```

例 2: パーティションを指定した表のエクスポート

従業員名にパーティションが指定されている emp 表があるとします。m および z の 2 つのパーティションがあります。次の例に示すように、パーティションを指定して表をエクスポートすると、指定し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/tiger 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/tiger 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.
```

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 および z で構成されているパーティション表です。emp は、コンポジット・メソッドでパーティション化されます。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。次の例に示すように、コンポジット・パーティション m をエクスポートする場合、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。サブパーティション（sp4）を指定して表をエクスポートすると、指定したサブ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/tiger 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/tiger 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.
```

インポート・セッションの例

この項では、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。ここでは、次の 4 つのインポート・セッションの例を示します。

- 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

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。

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。

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.
```

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

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/tiger 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/tiger 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。

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/tiger 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/tiger 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.
```

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、次の手順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、これらの手順の例を示します。

```
> exp scott/tiger 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/tiger
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/tiger 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/tiger
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;
```

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.
```

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プラットフォーム間のデータベースの移動

ハードウェア・プラットフォーム間で既存の Oracle Database を移動させるために Oracle でサポートされている方法は、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方法のみです。これには UNIX システムと NT システム間および異なるプラットフォームで実行されている 2 つの NT システム間の移動が含まれます。

次の手順では、プラットフォーム間のデータベースの移動方法の概要を示します。

1. DBA ユーザーとして次の SQL 問合せを発行して、すべての表領域の正確な名前を取得します。この情報は後の手順で必要となります。

   ```
   SQL> SELECT tablespace_name FROM dba_tablespaces;
   ```

2. DBA ユーザーとして、ソース・データベースからの全体エクスポートを次のように実行します。

   ```
   > exp FULL=y FILE=expdat.dmp
   ```

   参照： 「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」（20-22 ページ）

3. ダンプ・ファイルをターゲット・データベース・サーバーに移動します。FTP を使用する場合は、ファイルの破損を防止するために、（FTP プロンプトで `binary` を入力して）必ずバイナリ形式でコピーします。
4. ターゲット・サーバーでデータベースを作成します。

   参照： データベースの作成方法の詳細は、『Oracle Database 管理者ガイド』を参照してください。

5. ダンプ・ファイルのインポート前に、手順 1 で取得した情報を使用し、表領域を作成する必要があります。表領域を作成しなかった場合、対応するデータ・ファイルがソース・データベースと同じファイル構造で作成され、ターゲット・システムのファイル構造と互換性が得られない場合があります。
6. `IGNORE` パラメータを使用可能にし、DBA ユーザーとして全体インポートを実行します。

   ```
   > imp FULL=y IGNORE=y FILE=expdat.dmp
   ```

   `IGNORE=y` を使用すると、インポート時の作成エラーは無視され、インポートが完了します。
7. 新しいデータベースの全体バックアップを実行します。

警告、エラーおよび完了メッセージ

この項では、エクスポート・ユーティリティおよびインポート・ユーティリティによって発行される異なるタイプのメッセージについて説明します。また、そのメッセージのログ・ファイルへの保存方法についても説明します。

ログ・ファイル

すべてのエクスポート・メッセージおよびインポート・メッセージは、ログ・ファイルに保存できます。この場合の保存方法は 2 つあります。1 つ目は、`LOG` パラメータを使用する方法です。2 つ目は、システムでサポートされている場合に限定されますが、出力をファイルにリダイレクトする方法です。リダイレクト先のファイルには、正常にアンロードおよびロードされた場合はその内容が、エラーが発生した場合はそのエラーに関する詳細情報が記録されます。

警告メッセージ

リカバリ可能なエラーの場合、エクスポートおよびインポート処理は続行されます。たとえば、表のエクスポート中にエラーが発生した場合は、エラー・メッセージが表示され（またはログが記録され）、次の表にスキップして処理が続けられます。リカバリ可能なエラーは、警告と呼ばれます。

また、インポートおよびエクスポート操作で、無効なオブジェクトが検出された場合にも警告が発行されます。

たとえば、表モード・エクスポートで、存在しない表を指定した場合、エクスポート・ユーティリティは他の表をすべてエクスポートします。その後、警告が発行されて処理が正常に終了されます。

リカバリ不能エラー・メッセージ

エラーの中にはリカバリ不能なものもあり、このようなエラーが発生するとエクスポート・セッションまたはインポート・セッションは終了します。これらのエラーは、内部的な問題が原因であるか、またはメモリーなどのリソースが使用できないか、リソースを使い果たしたことが原因で発生します。たとえば、`catexp.sql` スクリプトが実行されていない場合、エクスポート・ユーティリティによって、次のようなリカバリ不能なエラー・メッセージが発行されます。

```
EXP-00024: Export views not installed, please notify your DBA
```

完了メッセージ

エクスポートまたはインポートがエラーを返さないで完了した場合、そのことを示すメッセージが表示されます。次に、例を示します。

```
Export terminated successfully without warnings
```

リカバリ可能エラーが 1 つ以上発生しても、ジョブがそのまま続行され、処理が完了した場合は、次のようなメッセージが表示されます。

```
Export terminated successfully with warnings
```

リカバリ不能なエラーが発生した場合、ジョブは即時終了し、そのことを示すメッセージが表示されます。次に、例を示します。

```
Export terminated unsuccessfully
```

終了コードによる結果の検査と表示

エクスポート・ユーティリティおよびインポート・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 20-6 に、それぞれの結果に対応する終了コードを示します。

表 20-6 エクスポートおよびインポート時の終了コード

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。 イン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。 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エクスポートを終了します。 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
```

ネットワークに関する考慮点

この項では、ネットワークを介したエクスポートおよびインポートの実行時に考慮すべき点について説明します。

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。

Oracle Net を使用したエクスポートおよびインポート

Oracle Net を使用すると、ネットワークを介してエクスポートおよびイン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て、リモート Oracle Database のデータをローカル・エクスポート・ファイルに書込みできます。また、インポート・ユーティリティをローカルで実行して、リモート Oracle Database のデータの読取りができます。

Oracle Net でエクスポート・ユーティリティまたはインポート・ユーティリティを使用するには、`exp` コマンドまたは `imp` 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。

参照 :

- 『Oracle Database Net Services 管理者ガイド』
- 『Oracle Database Heterogeneous Connectivity 管理者ガイド』

キャラクタ・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

この項では、ユーザー・データおよびデータ定義言語（DDL）のキャラクタ・セット変換に関連するエクスポート・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・サーバーのキャラクタ・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。（キャラクタ・セットは、データベース作成時に指定されます。）ソース・データベースのキャラクタ・セットが、インポート・データベースのキャラクタ・セットと異なる場合、自動的にデータをインポート・サーバーのキャラクタ・セットに変換するためのキャラクタ・セット変換が実行されます。

変換によるキャラクタ・セットのソート順への影響

エクスポート・キャラクタ・セットのソート順が、インポート・キャラクタ・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 キャラクタ・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 キャラクタ・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 キャラクタ・セット・ベースのデータベースにインポートされると、EBCDIC キャラクタ・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。

参照：『Oracle Database グローバリゼーション・サポート・ガイド』

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 回のキャラクタ・セット変換が必要です。

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キャラクタ・セットで書き出されます。NLS_LANG の値が、データベースのキャラクタ・セットと異なる場合は、キャラクタ・セット変換が実行されます。
2. エクスポート・ファイルのキャラクタ・セットが、インポート先ユーザー・セッション用のキャラクタ・セットと異なる場合、ユーザー・セッションのキャラクタ・セットに変換されます。シングルバイト・キャラクタ・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・キャラクタ・セットの場合は、インポート・ファイルのキャラクタ・セットがエクスポート・ファイルのキャラクタ・セットと同じである必要があります。
3. ターゲット・データベースのキャラクタ・セットが、インポート・ユーザー・セッション用のキャラクタ・セットと異なる場合、3 回目のキャラクタ・セット変換が実行される場合があります。

キャラクタ・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、同一のキャラクタ・セットを使用するようにしてください。

シングルバイト・キャラクタ・セットとエクスポートおよびインポート

8 ビット・キャラクタ・セットのエクスポート・ファイルをインポートすると、8 ビット文字の一部が消去されることがあります（同等の 7 バイトに変換されます）。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・キャラクタ・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・キャラクタ・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・ファイルのキャラクタ・セットを設定できます。

マルチバイト・キャラクタ・セットおよびエクスポートとインポート

ターゲット・キャラクタ・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・キャラクタ・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・キャラクタ・セットはソース・キャラクタ・セットのスーパーセットであるか、ソース・キャラクタ・セットと同等である必要があります。

参照：『Oracle Database グローバリゼーション・サポート・ガイド』

注意： インポート・クライアントとインポート・サーバーでキャラクタ・セット幅が異なる場合、変換によってデータが長くなると、データが切り捨てられることがあります。データが切り捨てられると、インポート・ユーティリティによって警告メッセージが表示されます。

マテリアライズド・ビューおよびスナップショット

注意： 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・システムには、マスター表、オプションのスナップショット・ログおよびスナップショット自体の 3 つのオブジェクトがあり、相互に関連しています。表（マスター表、スナップショット・ログ表定義およびスナップショット表）は、個別にエクスポートできます。スナップショット・ログは、対応付けられたマスター表をエクスポートしないかぎり、エクスポートできません。スナップショットをエクスポートできるのは、全データベース・エクスポートおよびユーザー・モード・エクスポートの場合のみです。表モードではエクスポートできません。

参照： インポート固有の移行と互換性、およびスナップショットとスナップショット・ログの詳細は、『Oracle Database アドバンスト・レプリケーション』を参照してください。

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。

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。

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。新しくインポートしたスナップショットには、時刻 A に実行した最後のリフレッシュが記録されていますが、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。

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 文で明示的に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。

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。

注意： 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・ファイルおよびメタデータのエクスポート・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、フラットまたはバイナリ・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

エクスポート操作でのこれらのパラメータの使用方法の詳細は、20-27 ページの「TABLESPACES」および 20-27 ページの「TRANSPORT_TABLESPACE」を参照してください。

インポート操作でのこれらのパラメータの使用方法の詳細は、20-38 ページの「TABLESPACES」および 20-40 ページの「TRANSPORT_TABLESPACE」を参照してください。

参照：

- 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。
- トランスポータブル表領域の詳細は、『Oracle Database 概要』を参照してください。

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。読取り専用機能が必要な場合は、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。

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。

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 IOT 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、次の手順を実行します。

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）。

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・ポリシーを使用可能にして、表をエクスポートおよびインポートできます。その場合は、次のことに注意してください。

- 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・ポリシーが使用可能な表をエクスポートするための正しい権限が付与されていない場合は、読取り権限のある行のみがエクスポートされます。

  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。したがって、セキュリティ上の理由から、そのような表をエクスポートおよびインポートするユーザーは、DBA である必要があります。

- SELECT 文でファイングレイン・アクセス・コントロールが使用可能な場合、ファイングレイン・アクセスにより問合せがリライトされるため、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。
- ユーザー SYS、EXP_FULL_DATABASE ロールを使用可能なユーザーまたは EXEMPT ACCESS POLICY を付与されたユーザーのみがファイングレイン・アクセス・コントロールを持つ表に対してダイレクト・パス・エクスポートを実行できます。

参照： ファイングレイン・アクセス・コントロールの詳細は、『Oracle Database アドバンスト・アプリケーション開発者ガイド』を参照してください。

エクスポートおよびインポート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。

参照：

- 『Oracle Database 管理者ガイド』
- 『Oracle Database リファレンス』
- 『Oracle Database アップグレード・ガイド』

データベースの断片化を解消する方法

断片化とは、多数の小さな空き領域が散在しているデータベースの状態のことです。断片化しているデータベースを再編成すると、空き領域をより大きな連続したブロックとして使用できるようになります。次のように全データベース・エクスポートおよびインポートを実行することで、データベースの断片化を解消できます。

1. データベース全体のバックアップを取るために、全データベース・エクスポート（FULL=y）を実行します。
2. すべてのユーザーがログオフしてから、Oracle Database を停止します。
3. データベースを削除します。データベース削除の詳細は、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。
4. CREATE DATABASE 文を使用して、データベースを再作成します。
5. データベース全体をリストアするために、全データベース・インポート（FULL=y）を実行します。

参照： データベース作成の詳細は、『Oracle Database 管理者ガイド』を参照してください。

エクスポートおよびインポートでの記憶域パラメータの使用

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、20-58 ページの「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 データと LOB 索引は、格納している表と同じ表領域に常駐することはできません。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。

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。

エクスポート・パラメータ COMPRESS

エクスポート時のデフォルトによって、初期エクステントにインポートされる表のすべてのデータを整理統合するように、記憶域パラメータが調整されます。初期エクステントのサイズを元のまま保つには、エクスポート時にエクステントが整理統合されないように COMPRESS=n を指定します。詳細は、20-17 ページの「COMPRESS」を参照してください。

エクスポート固有の情報

この項では、オリジナルのエクスポート・ユーティリティ固有の事項について説明します。この章の内容は、次のとおりです。

- 従来型パス・エクスポートおよびダイレクト・パス・エクスポート
- ダイレクト・パス・エクスポートの起動
- 読取り専用データベースからのエクスポート
- 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

従来型パス・エクスポートおよびダイレクト・パス・エクスポート

エクスポートでは次の 2 つの方式で表データをエクスポートできます。

- 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 の SELECT 文によって、データベースの表からデータが抽出されます。データはディスクからバッファ・キャッシュに読み込まれ、行は評価バッファに転送されます。式の評価が終了すると、そのデータはエクスポートを実行するクライアントへ転送され、そこでエクスポート・ファイルに書き込まれます。

ダイレクト・パス・エクスポートでは、データがディスクからバッファ・キャッシュに読み込まれ、行がエクスポート・クライアントに直接転送されるため、従来型パス・エクスポートに比べて非常に高速です。評価バッファ（SQL コマンド処理レイヤー）はバイパスします。データは、すでにエクスポート・ユーティリティが要求する形式になっているため、不要なデータ変換をする必要がありません。データはエクスポート・クライアントに転送され、このクライアントでエクスポート・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。デフォルトは DIRECT=n です。この場合、従来型パスで表データが抽出されます。この項の後半では、次の項目について説明します。

- ダイレクト・パス・エクスポートのセキュリティに関する考慮点
- ダイレクト・パス・エクスポートのパフォーマンスの考慮点
- ダイレクト・パス・エクスポートの制限事項

注意： ダイレクト・パスで表をエクスポートする場合は、他のトランザクションが同じ表を更新しておらず、ロールバック・セグメントが十分なサイズがあることを確認してください。他のトランザクションが同じ表を更新していたり、ロールバック・セグメントのサイズが不十分な場合は、次のエラーが返される場合があります。

ORA-01555: スナップショットが古すぎます : ロールバック・セグメント *string*、名前 *string* が小さすぎます。

エクスポートは正常に実行されず、終了します。

ダイレクト・パス・エクスポートのセキュリティに関する考慮点

Oracle 仮想プライベート・データベース（VPD）および Oracle Label Security は、ダイレクト・パス・エクスポート中には施行されません。

次のユーザーは、データベースからデータを抽出するために使用するエクスポート・モード、アプリケーションまたはユーティリティにかかわらず、VPD および Oracle Label Security を施行する必要はありません。

- データベース・ユーザー SYS
- 直接またはデータベース・ロールを介して、EXEMPT ACCESS POLICY 権限を付与されたデータベース・ユーザー

EXEMPT ACCESS POLICY 権限を付与されたユーザーは、VPD および Oracle Label Security を施行する必要はありません。これは強力な権限であるため、慎重に管理する必要があります。この権限は、SELECT、INSERT、UPDATE および DELETE などの従来のオブジェクト権限の施行には影響しません。ユーザーが EXEMPT ACCESS POLICY 権限を付与されていても、これらの権限は施行されます。

参照：

- 「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」（20-59 ページ）
- 『Oracle Database アドバンスト・アプリケーション開発者ガイド』

ダイレクト・パス・エクスポートのパフォーマンスの考慮点

ダイレクト・パス・エクスポートの起動時に、RECORDLENGTH パラメータの値を大きくすると、パフォーマンスが向上する場合があります。実際のパフォーマンス向上の度合いは、次の要因によって異なります。

- DB_BLOCK_SIZE
- 表の列の型
- I/O レイアウト（エクスポート・ファイルの転送先ドライブは、データベース・ファイルが常駐するディスク・ドライブとは別にする。）

RECORDLENGTH の値は、次のように設定することをお薦めします。

- ファイル・システムの I/O ブロック・サイズの倍数であること。
- DB_BLOCK_SIZE の倍数であること。

ダイレクト・パス・エクスポートで作成された。エクスポート・ファイルのインポートには、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・ファイルと同様のインポート時間が必要です。

ダイレクト・パス・エクスポートの制限事項

ダイレクト・パス・モードを使用する場合は、次の制限事項に注意する必要があります。

- 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・ファイルのいずれかを使用する必要があります。対話方式でダイレクト・パス・エクスポートを起動することはできません。
- エクスポート・ユーティリティの BUFFER パラメータを使用できるのは、従来型パス・エクスポートの場合のみです。ダイレクト・パス・エクスポートでは、エクスポート・ファイルへの書込みに使用するバッファのサイズは、RECORDLENGTH パラメータで指定します。
- 表領域モード（TRANSPORT_TABLESPACES=Y）でエクスポートする場合は、ダイレクト・パスは使用できません。
- 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・キャラクタ・セットと等しい場合にのみデータをエクスポートできます。NLS_LANG が設定されていない場合、またはデータベース・キャラクタ・セットとは異なる場合は、警告が表示され、エクスポートは中断されます。NLS_LANG 環境変数のデフォルト値は AMERICAN_AMERICA.US7ASCII です。

読取り専用データベースからのエクスポート

ソース・データベースからメタデータを抽出するには、エクスポートで順序付けする句（ソート操作）を含む問合せを使用します。これらの問合せが成功するには、エクスポートを実行しているユーザーがソート・セグメントを割り当てることが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割り当てるには、ユーザーの一時表領域が、ローカル管理一時表領域を示すように設定する必要があります。

> **参照：** この環境の設定の詳細は、『Oracle Data Guard 概要および管理』を参照してください。

データベース・オブジェクトのエクスポートに関する考慮点

次の項からは、特定のデータベース・オブジェクトをエクスポートするときに考慮すべき点について説明します。

順序のエクスポート

エクスポート中にトランザクションが順序番号にアクセスすると、順序番号はスキップされる可能性があります。順序番号がスキップされないようにするには、エクスポート中に順序番号にアクセスしないようにします。

順序番号がスキップされる可能性があるのは、キャッシュされている順序番号が使用中の場合のみです。順序番号のキャッシュが割り当てられている場合は、現行のデータベースでこれらの順序番号を使用できます。エクスポートされる値は、その次（キャッシュされている値の後）の順序番号です。キャッシュされても使用されていない順序番号は、順序のインポート時に失われます。

LONG データ型および LOB データ型のエクスポート

エクスポート時、LONG データ型は、セクション単位でフェッチされます。ただし、各行のすべてのデータ（LONG データ型を含む）を保持できるだけのメモリーが使用可能である必要があります。

LONG 列の長さは、最大 2GB です。

LOB 列のすべてのデータを、同時にメモリーに置いておく必要はありません。LOB データのロードおよびアンロードはセクション単位で行われます。

注意： 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 列に比べ、より少ない制限事項が適用されます。また、LOB 機能はすべてのリリースで拡張されていますが、LONG 機能はいくつかのリリースでは拡張されていません。

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内容は、エクスポート・ファイルにはエクスポートされません。全データベース・モード・エクスポートおよびユーザー・モード・エクスポートの場合、ライブラリの仕様（名前、位置）のみがエクスポートされます。データベースを新しい場所に移動する場合、ライブラリの実行可能ファイルを移動し、ライブラリの仕様を更新する必要があります。

オフライン・ローカル管理表領域のエクスポート

エクスポート中の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場合、表領域の定義を完全にエクスポートできず、エラー・メッセージが表示されます。データはインポートできますが、インポートの前に、まずオフライン・ローカル管理表領域を作成し、不完全な表領域を参照する DDL コマンドでエラーが発生しないようにする必要があります。

ディレクトリ別名のエクスポート

ディレクトリ別名の定義は、全データベース・モード・エクスポートの場合のみエクスポートされます。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、その新しい場所に対応する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・エクスポートや表モード・エクスポートの場合はエクスポートされません。したがって、ディレクトリ別名を使用する前に、ターゲット・システムにディレクトリ別名が作成されていることを確認する必要があります。

BFILE 列および属性のエクスポート

エクスポート・ファイルには、BFILE 列またはその属性から参照される外部ファイルの内容は格納されません。ファイルの名前およびディレクトリ別名のみが、エクスポート時にコピーされ、インポート時にリストアされます。旧ディレクトリからではファイルにアクセスできない場所にデータベースを移動する場合、DBA は、指定されたファイルが格納されているディレクトリを、アクセス可能な新しい場所へ移動する必要があります。

外部表のエクスポート

外部表の内容は、エクスポート・ファイルにはエクスポートされません。全データベース・モード・エクスポートおよびユーザー・モード・エクスポートの場合、表の仕様（名前、位置）のみがエクスポートされます。データベースを新しい場所に移動する場合、外部データを手動で移動して表の仕様を更新する必要があります。

オブジェクト型定義のエクスポート

どのエクスポート・モードでも、エクスポートされる表で使用されているオブジェクト型定義に関する情報はエクスポートされます。オブジェクト名、オブジェクト識別子、オブジェクト構成などの情報は、ターゲット・システムでのオブジェクト型とエクスポート・ファイルに格納されているオブジェクト・インスタンスに整合性があることを検証するために必要となります。これによって、インポート時に、表に必要なオブジェクト型が同一のオブジェクト識別子で作成されます。

ただし、表モード、ユーザー・モードおよび表領域モードで、オブジェクト型に対する実行権がないユーザーがエクスポートを実行している場合は、表に必要なすべてのオブジェクト型定義が、エクスポート・ファイルに保持されるとはかぎりません。この場合、同じオブジェクト型の識別子および同じオブジェクトの構成を持つ型の存在を検証するために必要な情報のみが、インポートのターゲット・システムに書き込まれます。

DBA に協力を求めて型定義を作成するか、DBA が実行した全データベース・モードまたはユーザー・モードのエクスポートから型定義をインポートすることによって、ターゲット・システムに適切な型定義が確実に存在するようにしてください。

すべてのオブジェクト型定義を保持するには、定期的に全データベース・エクスポートを実行することが重要です。また、別のユーザーのスキーマに属するオブジェクト型定義を使用する場合は、DBA が、適切なユーザー・グループのユーザー・モードで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 が所有する table1 に blake のオブジェクト型である type1 が存在する場合、この表に必要な型定義を保持するには、DBA がユーザー・モードで blake および scott の両方を指定してエクスポートを実行する必要があります。

ネストした表のエクスポート

ネストした表については、外部表がエクスポートされる場合は、必ず内部表のデータもエクスポートされます。内部のネストした表を指定することはできますが、それらを個別にエクスポートすることはできません。

アドバンスト・キュー（AQ）表のエクスポート

キューは表に実装されています。キューのエクスポートおよびインポートは、その基礎となるキュー表および関連するディクショナリ表のエクスポートおよびインポートになります。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、表定義に関する情報とキュー・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の両方がエクスポートされるため、キュー表のインポート時に、インポートを実行したユーザーがアプリケーション・レベルでのデータの整合性をメンテナンスすることになります。

> **参照：**『Oracle Streams アドバンスト・キューイング・ユーザーズ・ガイド』

シノニムのエクスポート

シノニムおよびもう 1 つのオブジェクトとして使用される名前を参照する、コンパイル済のオブジェクトをエクスポートする場合は、注意が必要です。これらのオブジェクトをエクスポートおよびインポートすると、強制的に再コンパイルされ、オブジェクトの定義が変更される可能性があります。

次の例では、この問題について説明します。

```
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;
```

前述の例では、データベースがエクスポートされた場合、トリガーの emp に対する参照では、scott の表ではなく blake のビューが参照されます。このため、インポート時に t_emp トリガーを再確立しようとすると、エラーになります。

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

対応する DBMS_JAVA パッケージがないとき、または Java がロードされていないか、不適切にロードされているときに、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した場合、エラーが発生して、エクスポートは終了します。この場合に生成されるエラー・メッセージには EXP-00008、ORA-00904、ORA-29516 などがあります。

Java が使用可能な場合、エクスポート・ユーティリティを再実行する前に、DBMS_JAVA シノニムと DBMS_JAVA パッケージの両方が作成済であり、有効であることを確認します。

Java が使用可能でない場合、エクスポート・ユーティリティを再実行する前に Java 関連のオブジェクトを削除します。

インポート固有の情報

この項では、オリジナルのインポート・ユーティリティ固有の事項について説明します。この章の内容は、次のとおりです。

インポート操作中のエラー処理

この項では、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。「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 次の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。

- 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

参照：

- 『Oracle Database アドバンスト・アプリケーション開発者ガイド』
- 『Oracle Database 概要』

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。

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。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。

> **注意：** IGNORE=y を指定した場合、表の 1 つ以上の列に対して一意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。

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。

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。20-29 ページの「COMMIT」を参照してください。

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。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。

索引作成およびメンテナンスの制御

この項では、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。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。

索引作成およびメンテナンスの制御

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 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、次の手順を実行します。

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/tiger 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。

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。エクスポート・パラメータの詳細は、20-25 ページの「STATISTICS」を、インポート・パラメータの詳細は、20-36 ページの「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・データベース間のキャラクタ・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）。

注意： 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。

インポート操作のチューニングに関する考慮点

この項では、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。パフォーマンスを向上させる方法は、次のとおりです。

- システム・レベル・オプションの変更
- 初期化パラメータの変更
- インポート・オプションの変更
- 大量の LOB データの処理
- 大量の LONG データの処理

システム・レベル・オプションの変更

システム・レベル・オプションに関して次の点を考慮すると、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。

- 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。

  注意： 将来のリリースの Oracle Database では、ロールバック・セグメントは廃止されます。かわりに、自動 UNDO 管理を使用することをお薦めします。

- 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。

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次の点を考慮すると、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。

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次の点を考慮すると、パフォーマンスの向上につながる場合があります。20-29 ページの「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廃止されるため、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。

大量の LOB データの処理

大量の LOB データをインポートする場合は、次のことに注意してください。

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。

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。

参照： 「LONG 列のインポート」（20-74 ページ）

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

次の項では、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

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、20-39 ページの「TOID_NOVALIDATE」を参照してください。

> **注意：** 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。

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

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。

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。

REF データのインポート

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;
```

> **参照：** ANALYZE TABLE 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。

BFILE 列およびディレクトリ別名のインポート

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。

外部関数ライブラリのインポート

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。

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 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。

> **参照：** 「COMPILE」（20-29 ページ）

Java オブジェクトのインポート

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）。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。

外部表のインポート

インポート・ユーティリティでは、外部表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・ファイルの表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。

AQ 表のインポート

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。

> **参照：** 『Oracle Streams アドバンスト・キューイング・ユーザーズ・ガイド』

LONG 列のインポート

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。

注意： 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 列に比べ、より少ない制限事項が適用されます。また、LOB 機能はすべてのリリースで拡張されていますが、LONG 機能は、いくつかのリリースでは拡張されていません。

トリガーが存在する場合の LOB 列のインポート

Oracle Database 10*g* 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。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改善されています。

LOB が NULL ではなく、空である必要がある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 列および空の LOB 列を確認してください。

ビューのインポート

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 OPTION 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、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。

パーティション表のインポート

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_P*nnn* 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。

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。

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。

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

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、次の手順に従います。

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=schema`*n* `FILE=schema`*n*
   b. `imp FILE=schema`*n* `FROMUSER=schema`*n* `TOUSER=schema`*n* `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。

リリースおよびバージョンが異なるエクスポート・ユーティリティの使用方法

この項では、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 の使用に関連する互換性の問題について説明します。

リリースが異なる Oracle Database 間でデータを移動させる場合は、常に、次の基本的な規則が適用されます。

- 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）のバージョン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・データベースまたはターゲット・データベースの下位の方のバージョンと同じである必要があります。

  たとえば、上位バージョンのデータベースにインポートするエクスポート・ファイルを作成するには、ソース・データベ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。それに対し、下位バージョンのデータベースにインポートするエクスポート・ファイルを作成するには、ターゲット・データベ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 のエクスポート・ユーティリティを使用して、Oracle9*i* サーバーからエクスポートし、Oracle8 のエクスポート・ファイルを作成できます。詳細は、20-77 ページの「Oracle9i データベースからの Oracle8 のエクスポート・ファイルの作成」を参照してください。

リリースおよびバージョン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上の制限

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、次の制限が適用されます。

- 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 2（9.2）のエクスポート・ダンプ・ファイルは、リリース 1（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。
- ダイレクト・パスの場合も従来型パスの場合も、Oracle9*i* のエクスポート・ユーティリティを使用して作成されたエクスポート・ファイルは、旧リリースのインポート・ユーティリティとは互換性がないため、インポートに使用できるのは Oracle9*i* のインポート・ユーティリティのみです。下位互換性が問題となる場合は、Oracle9*i* データベースに対して下位のリリースまたはバージョンのエクスポート・ユーティリティを使用します。

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

表 20-7 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 20-7 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

Oracle9*i* データベースからの Oracle8 のエクスポート・ファイルの作成

Oracle9*i* データベースから Oracle8 のエクスポート・ファイルを作成する場合、特別な処理は必要ありません。ただし、Oracle9*i* データベースで Oracle8 のエクスポート・ユーティリティを使用した場合、次の機能はサポートされません。

- ダイレクト・パス・ロード（`DIRECT=y`）を指定した場合、オブジェクトおよび LOB を含む表の行はエクスポートされません。
- ディメンションはエクスポートされません。
- ファンクション索引およびドメイン索引はエクスポートされません。

- セカンダリ・オブジェクト（表、索引、順序など、ドメイン索引のサポートで作成されるもの）はエクスポートされません。
- ビュー、プロシージャ、ファンクション、パッケージ、型本体および Oracle9*i* の新機能への参照を含む型は、コンパイルされない場合があります。
- SQL の DDL が、SQL ではなくストアド・プロシージャとして実装されているオブジェクトはエクスポートされません。
- アクションが CALL 文であるトリガーは、エクスポートされません。
- 論理 ROWID 列、主キー参照またはユーザー定義 OID 列を含む表はエクスポートされません。
- 一時表はエクスポートされません。
- 索引構成表（IOT）は圧縮前の状態に戻ります。
- パーティション化された IOT のパーティション情報が消失します。
- 索引タイプおよび演算子はエクスポートされません。
- ローカル管理表領域、一時表領域および UNDO 表領域はエクスポートされません。
- Java ソース、Java クラスおよび Java リソースはエクスポートされません。
- VARRAY 列の、可変幅 CLOB、コレクション拡張および LOB ストレージ句、またはネストした表の拡張はエクスポートされません。
- ファイングレイン・アクセス・コントロール・ポリシーは保持されません。
- 外部表はエクスポートされません。

21

Enterprise Manager Configuration Assistant（EMCA）

この章では、Enterprise Manager Configuration Assistant（EMCA）について説明します。

この章の主な内容は、次のとおりです。

- EMCA による Database Control の構成
- EMCA によるソフトウェア・ライブラリの構成
- EMCA パラメータへの入力ファイルの使用方法
- Oracle Real Application Clusters による EMCA の使用方法
- EMCA で使用するポートの指定
- EMCA によるトラブルシューティングのヒント

> **参照：** その他の高度な構成については、『Oracle Enterprise Manager アドバンスト構成』を参照してください。

EMCA による Database Control の構成

Database Configuration Assistant（DBCA）を使用して Oracle Database 11*g* を構成する場合、DBCA の Graphical User Interface により、Database Control オプションを選択したり、データベースの他の項目を構成することができます。

オペレーティング・システムのコマンドラインを使用して Database Control を構成する場合は、Enterprise Manager Configuration Assistant（EMCA）を使用できます。

EMCA を使用して Database Control を構成する手順は次のとおりです。

1. 管理対象データベースの Oracle ホームおよびシステム識別子（SID）を指定するように、次の環境変数を設定します。
   - ORACLE_HOME
   - ORACLE_SID
2. ディレクトリを ORACLE_HOME/bin ディレクトリに変更します。
3. 表 21-1 に示されているオプションのコマンドライン引数とともに次のコマンドを入力して、EMCA を起動します。

   ```
   $PROMPT> ./emca
   ```

   コマンドラインに指定する引数に応じて、EMCA により Database Control の構成に必要な情報を入力するプロンプトが表示されます。

   たとえば、次のコマンドを入力して Database Control を構成すると、データベースの自動日次バックアップが実行されます。

   ```
   $PROMPT> ./emca -config dbcontrol db -backup
   ```

EMCA コマンドの形式は次のとおりです。

```
emca [operation] [mode] [flags] [parameters]
```

注意： ASM を使用して単一インスタンスのデータベースにデータベース・コンソールを構成する場合は、EMCA コマンドとともに追加のパラメータを渡す必要はありません。ASM インスタンスを自動的に検出するデータベース・コンソールを構成するには、次のコマンドを実行します。

```
emca -config dbcontrol db -repos create
```

注意： Database Control リポジトリはセキュア・モードです。つまり、Enterprise Manager のデータは暗号化されていることを意味します。暗号化キーのファイル位置は、データベースのタイプに従って次のようになります。

- 単一インスタンスのデータベース：

  ```
  ORACLE_HOME/<host>_<sid>/emkey.ora
  ```

- Oracle RAC データベース：

  ```
  ORACLE_HOME/<node>_<sid>/emkey.ora
  ```

このファイルを紛失すると暗号化されたデータが使用できなくなるため、このファイルのバックアップを作成することをお薦めします。

表 21-1 では、有効な実行操作とモードを説明し、オプションのパラメータを括弧内に示します。表 21-2 では、フラグとその動作を説明し、表 21-3 に、オプションのパラメータの詳細を示します。EMCA パラメータの形式は、[-parameterName parameterValue] です。コマンドラインで複数のパラメータを組み合せて使用できます。

表 21-1 EMCA のコマンドライン操作

コマンド	説明
emca -h | --h | -help | --help	EMCA ユーティリティのヘルプ・メッセージを表示するには、このオプションを使用します。表 21-1、表 21-2 および表 21-3 に示されているオプションと、指定可能な有効なパラメータが表示されます。
emca -version	EMCA に関連付けられているバージョン情報を出力します。
emca -config dbcontrol db [-repos (create | recreate)] [-cluster] [-silent] [-backup] [parameters]	データベースに対して Database Control を構成します。オプションには、Database Control リポジトリの作成（または再作成）、自動バックアップの構成、およびクラスタ・データベースにおけるこれらの操作の実行などがあります。
emca -config centralAgent (db | asm) [-cluster] [-silent] [parameters]	データベース・インスタンスまたは自動ストレージ管理（ASM）インスタンスに対して集中エージェント管理を構成します。オプションには、クラスタ環境におけるこの操作の実行があります。この操作を行うと、Oracle Enterprise Manager Grid Control コンソールによって集中管理が可能なデータベースが構成されます。このオプションを使用するには、ネットワーク・ホスト上に Enterprise Manager の Oracle Management Service コンポーネントをあらかじめインストールしておく必要があります。また、データベースを稼働させるホスト上に Oracle Management Agent をインストールすることも必要です。
emca -config all db [-repos (create | recreate)] [-cluster] [-silent] [-backup] [parameters]	データベースに対して Database Control および集中エージェント管理の両方を構成します。使用可能な構成オプションは、前述したオプションと同様です。
emca -deconfig dbcontrol db [-repos drop] [-cluster] [-silent] [parameters]	データベースに対して Database Control の構成を解除します。オプションには、Database Control リポジトリの削除、およびクラスタ・データベースにおけるこの操作の実行があります。たとえば、削除予定のデータベースから Database Control 構成を削除するために、このコマンドを使用する場合があります。このような場合、物理的にデータベースを削除する前に、Database Control 構成が削除されます。この操作では、実際のデータベースおよびそのデータ・ファイルは削除されません。
emca -deconfig centralAgent (db | asm) [-cluster] [-silent] [parameters]	データベース・インスタンスまたは ASM インスタンスに対して集中エージェント管理の構成を解除します。オプションには、クラスタ環境におけるこの操作の実行があります。たとえば、削除予定のデータベースから集中エージェント管理構成を削除するために、このコマンドを使用する場合があります。このような場合、物理的にデータベースを削除する前に、集中エージェント管理構成が削除されます。この操作では、実際のデータベースおよびそのデータ・ファイルは削除されません。
emca -deconfig all db [-repos drop] [-cluster] [-silent] [parameters]	データベースに対して Database Control および集中エージェント管理の両方の構成を解除します。使用可能な構成解除オプションは、前述したオプションと同様です。
emca -addInst (db | asm) [-silent] [parameters]	データベースまたは ASM 記憶域の新しいクラスタ・インスタンスに対して Enterprise Manager を構成します。詳細は、21-7 ページの「Oracle Real Application Clusters による EMCA の使用方法」を参照してください。
emca -deleteInst (db | asm) [-silent] [parameters]	クラスタ・データベースまたは ASM 記憶域に対して特定のインスタンスの Enterprise Manager の構成を解除します。詳細は、21-7 ページの「Oracle Real Application Clusters による EMCA の使用方法」を参照してください。

表 21-1 EMCA のコマンドライン操作（続き）

コマンド	説明
emca -reconfig ports [-cluster] [parameters]	Database Control ポートを明示的に再割り当てします。オプションには、クラスタ環境におけるこの操作の実行があります。詳細は、21-9 ページの「EMCA で使用するポートの指定」を参照してください。
emca -reconfig dbcontrol -cluster [-silent] [parameters]	クラスタ・データベースの Database Control デプロイメントを再構成します。このコマンドは -cluster オプションとともに使用する必要があります。詳細は、21-7 ページの「Oracle Real Application Clusters による EMCA の使用方法」を参照してください。
emca -displayConfig dbcontrol -cluster [-silent] [parameters]	クラスタ環境における Database Control の現在のデプロイメント構成の情報を表示します。このコマンドは -cluster オプションとともに使用する必要があります。詳細は、21-7 ページの「Oracle Real Application Clusters による EMCA の使用方法」を参照してください。
emca -upgrade (db | asm | db_asm) [-cluster] [-silent] [parameters]	旧バージョンの Enterprise Manager 構成を現行バージョンにアップグレードします。この操作は、データベース、ASM、またはデータベースと ASM インスタンスで同時に実行可能です。この操作により、実際のデータベース・インスタンスおよび ASM インスタンスがアップグレードされることも、Enterprise Manager ソフトウェアがアップグレードされることもありません。かわりに、指定したインスタンスの構成ファイルがアップグレードされ、現行バージョンの Enterprise Manager ソフトウェアと互換性がとれるようになります。EMCA は、すべての Oracle ホームのホスト上で、指定されたデータベースまたは ASM ターゲットのすべてのインスタンスのアップグレードを試行します（リスナー・ポートまたは Oracle ホームなどの特定のターゲット・プロパティが変更されている可能性があるため）。
emca -restore (db | asm | db_asm) [-cluster] [-silent] [parameters]	現行バージョンの Enterprise Manager 構成を旧バージョンにリストアします。これは -upgrade オプションの逆の操作で、-upgrade 操作による変更を元に戻します。また、-upgrade 操作とオプションは同じです。
emca -migrate -from dbcontrol -to centralAgent [-repos drop] [-cluster] [-silent] [parameters]	Database Control から集中エージェントに Enterprise Manager 構成を移行します。

表 21-2 EMCA のコマンドライン・フラグ

フラグ	説明
db	データベース（クラスタ・データベースを含む）に対する操作を実行します。自動ストレージ管理（ASM）を使用するデータベースにこのオプションを使用して、データ・ファイルを格納します。データベースが ASM を使用している場合、前述の（-upgrade と -restore を除く）すべての構成操作およびモードによってこのフラグが自動的に検出され、データベース・インスタンスおよび ASM インスタンスの両方に変更が適用されます。
asm	ASM のみのインスタンス（クラスタ ASM インスタンスを含む）に対して操作を実行します。

表 21-2 EMCA のコマンドライン・フラグ（続き）

フラグ	説明
db_asm	このフラグは、-upgrade および -restore モードでのみ使用可能です。データベース・インスタンスと ASM インスタンスの両方に対してアップグレードまたはリストア操作を実行します。データベース・インスタンスおよび ASM インスタンスは、別々にアップグレードまたはリストアされる場合があります（つまり、ASM インスタンスをアップグレードしても、そのインスタンスがサービスを提供するデータベース・インスタンスをアップグレードする必要はありません）。したがって、Enterprise Manager 構成は、データベースおよびそれに対応する ASM インスタンスに対して、別々にアップグレードまたはリストアできます。
-repos create	新しい Database Control 管理リポジトリを作成します。
-repos drop	現在の Database Control 管理リポジトリを削除します。
-repos recreate	現在の Database Control 管理リポジトリを削除してから、新しく再作成します。
-cluster	クラスタ・データベース・インスタンスまたは ASM インスタンスに対して操作を実行します。
-silent	その他の情報が求められることなく操作が実行されます。このモードを指定した場合、すべての必要なパラメータをコマンドラインに入力するか、または睦 espFile 引数を使用して入力ファイルに指定する必要があります。また、コマンドラインに emca -help を入力すると、使用可能なパラメータのリストを表示できます。
-backup	データベースに対して自動バックアップを構成します。EMCA では、自動日次バックアップ・オプションのプロンプトが表示されます。データベース・ファイルのバックアップには、デフォルトの Enterprise Manager 設定が使用されます。 **注意：**このオプションを使用すると、EMCA では自動バックアップ用のフラッシュバック・リカバリ領域の指定に、db_recovery_file_dest 初期化パラメータの値が使用されます。このパラメータが設定されていないと、EMCA ではエラーが発生します。これらの設定は、Database Control の「メンテナンス」ページを使用して後で変更できます。詳細は、Database Control のオンライン・ヘルプを参照してください。

表 21-3 EMCA のコマンドライン・パラメータ

パラメータ	説明
-respFile	構成操作の実行時に使用する EMCA のパラメータがリストされている入力ファイルのパスを指定します。詳細は、21-7 ページの「EMCA パラメータへの入力ファイルの使用方法」を参照してください。
-SID	データベース・システム識別子。
-PORT	データベースにサービスを提供しているリスナーのポート番号。
-ORACLE_HOME	データベースの Oracle ホーム（絶対パス）。
-ORACLE_HOSTNAME	ローカル・ホスト名。
-LISTENER_OH	リスナーの実行元である Oracle ホーム。リスナーが、データベースが稼働しているもの以外の Oracle ホームから実行されている場合は、パラメータ LISTENER_OH を指定する必要があります。
-HOST_USER	ホスト・マシンのユーザー名（自動バックアップ用）。
-HOST_USER_PWD	ホスト・マシンのユーザー・パスワード（自動バックアップ用）。

表 21-3 EMCA のコマンドライン・パラメータ（続き）

パラメータ	説明
-BACKUP_SCHEDULE	HH:MM 形式でのスケジュール（自動日次バックアップ用）。
-EMAIL_ADDRESS	通知用の電子メールアドレス。
-MAIL_SERVER_NAME	通知用の送信メール（SMTP）・サーバー。
-ASM_OH	自動ストレージ管理の Oracle ホーム。
-ASM_SID	ASM インスタンスのシステム識別子。
-ASM_PORT	ASM インスタンスにサービスを提供しているリスナーのポート番号。
-ASM_USER_ROLE	ASM インスタンスへの接続用ユーザー・ロール。
-ASM_USER_NAME	ASM インスタンスへの接続用ユーザー名。
-ASM_USER_PWD	ASM インスタンスへの接続用パスワード。
-DBSNMP_PWD	DBSNMP ユーザーのパスワード。
-SYSMAN_PWD	SYSMAN ユーザーのパスワード。
-SYS_PWD	SYS ユーザーのパスワード。
-SRC_OH	Enterprise Manager の構成をアップグレードまたはリストアするデータベースの Oracle ホーム。
-DBCONTROL_HTTP_PORT	Database Control コンソールを Web ブラウザで表示するときに使用するポートを指定するには、このパラメータを使用します。詳細は、21-9 ページの「EMCA で使用するポートの指定」を参照してください。
-AGENT_PORT	Database Control に管理エージェント・ポートを指定するには、このパラメータを使用します。詳細は、21-9 ページの「EMCA で使用するポートの指定」を参照してください。
-RMI_PORT	Database Control に RMI ポートを指定するには、このパラメータを使用します。詳細は、21-9 ページの「EMCA で使用するポートの指定」を参照してください。
-JMS_PORT	Database Control に JMS ポートを指定するには、このパラメータを使用します。詳細は、21-9 ページの「EMCA で使用するポートの指定」を参照してください。
-CLUSTER_NAME	クラスタ名（クラスタ・データベース用）。
-DB_UNIQUE_NAME	一意のデータベース名（クラスタ・データベース用）。
-SERVICE_NAME	データベース・サービス名（クラスタ・データベース用）。
-EM_NODE	Database Control コンソールが実行されるノード（クラスタ・データベース用）。詳細は、21-7 ページの「Oracle Real Application Clusters による EMCA の使用方法」を参照してください。
-EM_SID_LIST	エージェントのみの構成用 SID のカンマ区切りリスト。データを -EM_NODE にアップロードします。詳細は、21-7 ページの「Oracle Real Application Clusters による EMCA の使用方法」を参照してください。
-EM_SWLIB_STAGE_LOC	ソフトウェア・ライブラリの場所。
-PORTS_FILE	使用するポートを指定している静的ファイルへのパス。デフォルト値は次のとおりです。`${ORACLE_HOME}/install/staticports.ini`

EMCA によるソフトウェア・ライブラリの構成

Database Control 構成の一環として、ソフトウェア・ライブラリが構成され、プロビジョニング・アーカイブがデプロイされます。デフォルトでは、ソフトウェア・ライブラリは次の場所に構成されます。

```
ORACLE_HOME/EMStagePatches_<db_unique_name>
```

Oracle Database 11*g* のインストールと構成が完了した後は、EMCA の EM_SWLIB_STAGE_LOC コマンドライン・パラメータを使用して、ソフトウェア・ライブラリを構成することもできます。

Oracle Real Application Clusters（RAC）では、EM_SWLIB_STAGE_LOC パラメータに渡されているソフトウェア・ライブラリの場所または ORACLE_HOME が共有記憶域内にある場合のみ、ソフトウェア・ライブラリは構成され、プロビジョニング・アーカイブがデプロイされます。

EMCA パラメータへの入力ファイルの使用方法

EMCA を実行している場合、一連のプロンプトに応答するかわりに、-respFile 引数を使用して入力ファイルを指定できます。作成する入力ファイルは、次の例と同様の形式にする必要があります。

```
PORT=1521
SID=DB
DBSNMP_PWD=xpE234D
SYSMAN_PWD=KDOdk432
```

作成した EMCA 入力ファイルは、コマンドラインで次のように使用できます。

```
$PROMPT> ./emca -config dbcontrol db -respFile input_file_path
```

たとえば、Database Control を構成して日次バックアップを実行し、Database Control 管理リポジトリを作成するには、例 21-1 に示したものと同様の入力ファイルを作成し、オペレーティング・システムのプロンプトに次のコマンドを入力します。

```
$PROMPT> ./emca -config dbcontrol db -repos create -backup -respFile input_file_path
```

例 21-1 EMCA の入力ファイルのサンプル

```
PORT=1521
SID=DB
DBSNMP_PWD=dow3l224
SYSMAN_PWD=squN3243
HOST_USER=johnson
HOST_USER_PWD=diTf32of
SYS_PWD=qlKj4352
BACKUP_SCHEDULE=06:30
```

Oracle Real Application Clusters による EMCA の使用方法

Oracle Real Application Clusters（RAC）では、複数のホストにわたる高可用性データベース環境を実現できます。各クラスタは、複数のクラスタ・データベースで構成され、そのデータベースはそれぞれ複数のクラスタ・データベース・インスタンスで構成されます。クラスタ・データベースは、クラスタ内のインスタンスの 1 つが使用可能なかぎり使用できます。

Oracle RAC 環境では、すべての EMCA コマンドを使用できます（一部のコマンドはクラスタの設定時にのみ使用可能です）。クラスタ・データベースを使用していることを示すには、-cluster フラグを使用します。このフラグは、ほぼすべての EMCA 操作モードで使用できます。

EMCA を使用して Oracle RAC に対して Database Control を構成する場合は、クラスタ内の各インスタンスに対して Database Control を構成します。ただし、デフォルトでは、Database Control コンソールはローカル・ノードでのみ起動します。クラスタの他のすべてのノードでは、Enterprise Manager エージェントのみが起動します。これは、Database Control コンソールによって、データベースへの多数の接続がオープンされるためです。コンソールの 1 つのインスタンスがクラスタ内のすべてのホストで稼働している場合、32 ノード環境または 64 ノード環境で許可されているオープン接続の最大数を簡単に超えてしまうことがあります。

これを回避するため、Database Control コンソールはローカル・ノードでのみ起動します。他のすべてのノードでエージェントの起動と停止に使用するコマンドは、emctl start dbconsole と emctl stop dbconsole のみです。各リモート・エージェントによって、それぞれのデータがローカル・ノードで稼働しているコンソールにアップロードされるため、そのコンソールからクラスタ内のすべてのターゲットの監視および管理を実行できます。RAC データベースの各インスタンスには、次のサブディレクトリが作成されます。

```
$ORACLE_HOME/hostname1.domainname_SID1
$ORACLE_HOME/hostname2.domainname_SID2
.
.
$ORACLE_HOME/hostnamen.domainname_SIDn
```

<SID1> ～ <SIDn> は、データベース・システム識別子です。

ただし、Oracle 11*g* リリース 1 の Database Control で構成された Oracle 10*g* リリース 2 の Oracle RAC データベースをアップグレードする場合、10*g* リリース 2 の Database Control 構成は保持されます。Oracle10*g* リリース 2 の Database Control には、各 Oracle RAC ノードで稼働しているデータベース・コンソールが含まれています。このコンソールはそれぞれ個々のノードで起動されたままです。この構成を変更する場合は、次のコマンドを使用します。

```
emca -reconfig dbcontrol -cluster -EM_NODE <nodename> -EM_SID_LIST <SID list>
```

<nodename> はノードのパブリック名を示し、<SID list> はデータベース・システム識別子のカンマ区切りリストを示します。このコマンドにより、現行の Database Control 設定が再構成され、次の操作が実行されます。

1. <nodename> の Database Control コンソールを起動します（起動していなかった場合）。
2. <nodename> で稼働しているコンソールにデータがアップロードされるように、<SID list> 内のデータベース・インスタンスを監視するエージェントをリダイレクトします。<nodename> 上のデータベース・インスタンスを監視するエージェントは、ローカル・コンソールにもデータをアップロードします。コマンドラインで -EM_NODE または -EM_SID_LIST を渡さなかった場合、これらを要求するプロンプトが表示されます。

プロンプトが表示されたときに -EM_NODE を指定しない場合は、ローカル・ノードがデフォルトで選択されます。-EM_SID_LIST を指定しない場合は、すべてのデータベース・インスタンスがデフォルトで選択されます。

このコマンドを使用すると、複数のノードでコンソールが起動することがあります。たとえば、<node1, node2, node3, node4, node5, node6, node7, node8> およびデータベース・インスタンス <oradb1, oradb2, oradb3, oradb4, oradb5, oradb6, oradb7, oradb8> を持つ 8 ノード・クラスタでは、続けて次のコマンドを実行できます。

```
$PROMPT> emca -reconfig dbcontrol -cluster -EM_NODE node1 -EM_SID_LIST
oradb2,oradb3,oradb4
$PROMPT> emca -reconfig dbcontrol -cluster -EM_NODE node5 -EM_SID_LIST
oradb6,oradb7,oradb8
```

この使用例では、node1 に 1 つ、node5 にもう 1 つの 2 つの Database Control コンソールが稼働されています。これらのコンソールのいずれかから、クラスタ内のすべてのターゲットを管理および監視できます。

現在のクラスタ構成の詳細は、次のコマンドを実行して表示できます。

```
emca -displayConfig dbcontrol -cluster
```

前述のコマンドでは、クラスタ・データベースの一意のデータベース名を入力するプロンプトが表示されます。これにより、現在の構成が画面に出力され、コンソールが稼働されているノードと、各エージェントがアップロード中のコンソールが示されます。

データベースまたは ASM 記憶域の新しいクラスタ・インスタンスに対して Enterprise Manager を構成する場合、次のコマンドを使用します。

```
emca -addInst db
```

クラスタ・データベースでは、その他の通常の操作にデータベース・インスタンスの作成と削除があります。新規インスタンスを作成した後で、EMCA を実行し、そのインスタンスに対して emca -addInst db コマンドを使用すると、Database Control または集中エージェント管理を構成できます。EMCA を実行しても、実際のデータベース・インスタンスは作成されません。Enterprise Manager の構成のみを行い、残りのクラスタ・データベース・インスタンスと一貫性のとれた方法でインスタンスを管理できるようにします。新規インスタンスに対して Enterprise Manager を構成する場合、EMCA コマンドを実行するのはインスタンスを作成した後のみです。また、これらの構成設定が新規インスタンスに伝播されるように、クラスタに関連付けられたデータベース・インスタンスに対して Enterprise Manager を構成したクラスタ内のノードからコマンドを実行します。新規インスタンスが作成されたノードからは、このコマンドを実行しないでください。このオプションは Oracle RAC 環境でのみ使用可能なため、コマンドラインで -cluster オプションを使用する必要はありません。emca -addInst db コマンドを実行した後、ノードおよびデータベースに対して次の情報を入力します。

```
Node name: node2
Database Unique Name: EM102
Database SID: EM1022
```

特定のデータベース・インスタンス（通常はデータベース・インスタンスが削除される前）に対して Enterprise Manager の構成を解除するには、構成する場合と逆のコマンド emca -deleteInst db を使用します。EMCA を実行しても、データベース・インスタンスは削除されません。Enterprise Manager の構成のみ削除され、Enterprise Manager によるインスタンスの管理ができなくなります。EMCA コマンドを実行してから、実際のクラスタ・データベース・インスタンスを削除するようにします。また、データベース・インスタンスが削除されるノードからではなく、別のノードからコマンドを実行するようにしてください。このオプションは Oracle RAC 環境でのみ使用可能なため、コマンドラインで -cluster オプションを使用する必要はありません。

詳細は、EMCA のコマンドライン操作が説明されている表 21-1 を参照してください。

注意： emca -c を使用して Oracle RAC に対して Database Control を構成する場合は、すべてのクラスタ・ノード上の TNS_ADMIN をチェックします。各ノードに対して異なる TNS_ADMIN が設定されていると、リスナーはターゲットに対して適切に構成されません。その場合、すべてのクラスタ・ノード上で同一の TNS_ADMIN を設定した後に、emca -c コマンドを実行します。

EMCA で使用するポートの指定

初めて Oracle Databse 11*g* をインストールする場合や、EMCA によって Database Control を初めて構成する場合は、Database Control により一連のデフォルト・システム・ポートが使用されます。たとえば、11*g* リリース 1 のデフォルトでは、ポート 1158 を使用して、次に示す Database Control にアクセスします。

```
http://host.domain:1158/em
```

これは、Internet Assigned Numbers Authority（IANA）によって、Database Control に割り当てられているデフォルト・ポートです。同様に、IANA によって割り当てられているデフォルトの Database Control エージェント・ポートは 3938 です。

EMCA を使用して Database Control を初めて構成する場合に、デフォルト・ポート以外のポートを使用するには、次の EMCA コマンドライン引数を使用します。また、次のコマンドを使用することで、Database Control の構成後に明示的にポートを割り当てることができます。

```
emca -reconfig ports [-cluster]
```

> **注意：** Oracle Database 11*g* のインストールと構成が完了した後は、次の EMCA コマンドライン引数を使用して、Database Control を構成することもできます。

次のリストには、標準の Database Control ポート割当てを制御する EMCA コマンドライン引数をまとめています。

- -DBCONTROL_HTTP_PORT <port_number>

  このポート番号は Database Control コンソールの URL に使用されます。たとえば、このポートを 5570 に設定すると、次の URL を使用して Database Control コンソールを表示できます。

  `http://host.domain:5570/em`

- -RMI_PORT <port_number>

  このポート番号は、Database Control に必要な J2EE ソフトウェアの一部の Remote Method Invocation（RMI）システムによって使用されます。デフォルト・ポートは、データベース・コンソールに固有のポートを構成する場合に変更できます。デフォルト・ポート（1521）以外のポートを使用する場合は、-RMI_PORT または -JMS_PORT オプションを emca reconfig コマンドとともに使用します。

- -JMS_PORT <port_number>

  このポートは、Database Control に必要な J2EE ソフトウェアの一部の OC4J Java Message Service（JMS）によって使用されます。デフォルト・ポートは、データベース・コンソールに固有のポートを構成する場合に変更できます。デフォルト・ポート（1521）以外のポートを使用する場合は、-RMI_PORT または -JMS_PORT オプションを emca reconfig コマンドとともに使用します。

- -AGENT_PORT <port_number>

  このポートは、Database Control のデータベースの監視および管理を行う Database Control 管理エージェントによって使用されます。

EMCA によるトラブルシューティングのヒント

次の項では、EMCA を使用して Database Control を構成する際のトラブルシューティングのヒントをいくつか説明します。

- データベース・リスナー・ポートを変更した後の EMCA の使用方法
- 11g リリース 1 の Grid Control エージェントによるデータベースまたは ASM インスタンスのアップグレード
- データベース・ホスト名または IP アドレス変更時の EMCA の使用方法
- TNS 構成を変更した場合の EMCA の使用方法

データベース・リスナー・ポートを変更した後の EMCA の使用方法

Database Control を構成した後にデータベースのリスナー・ポートを変更すると、データベースの状態は停止と表示されます。新しいリスナー・インポートを使用するように Database Control を再構成するには、-config dbcontrol db [-cluster] コマンドライン引数を使用して EMCA コマンドを実行します。

11*g* リリース 1 の Grid Control エージェントによるデータベースまたは ASM インスタンスのアップグレード

Oracle Enterprise Manager（Database Control または Grid Control 集中エージェント）に対して構成された 10*g* リリース 2 のデータベースまたは ASM インスタンスを 11*g* リリース 1 にアップグレードすると、アップグレードされるインスタンスを参照している関連ホスト上の Enterprise Manager のターゲットはすべて自動的に更新されます。これは、インスタンスの Oracle ホーム、ポート、またはその他のターゲット関連プロパティの変更がアップグレードに含まれているためです。ただし、ホスト上のこれらのターゲットの一部が 11*g* リリース 1 の Grid Control エージェントによって管理されている場合は、アップグレード時に正常に更新されないものもあります。これらのターゲットを更新するには、アップグレードされるデータベース（または ASM）・ターゲットの「ホーム」ページで「監視構成」リンクをクリックします。そのページで、Oracle ホーム、リスナー・ポートなどの必須プロパティを適切な値に更新できます。

データベース・ホスト名または IP アドレス変更時の EMCA の使用方法

データベース・ホスト名（ドメイン名を含む）または IP アドレスの変更時には、データベース・コンソールの構成を解除してから、リポジトリの作成コマンドを使用して再作成します。次のコマンドを実行します。

```
emca -deconfig dbcontrol db -repos drop
emca -config dbcontrol db -repos create
```

または

```
emca -deconfig dbcontrol db
emca -config dbcontrol db -repos recreate
```

TNS 構成を変更した場合の EMCA の使用方法

TNS 構成を変更する場合は、環境変数を設定して次のコマンドを実行します。

```
emca -config dbcontrol db
```

第 V 部

付録

第 V 部には、次の付録が含まれます。

付録 A「SQL*Loader の構文図」

この付録では、SQL*Loader の構文図について説明します。

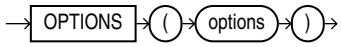
A

SQL*Loader の構文図

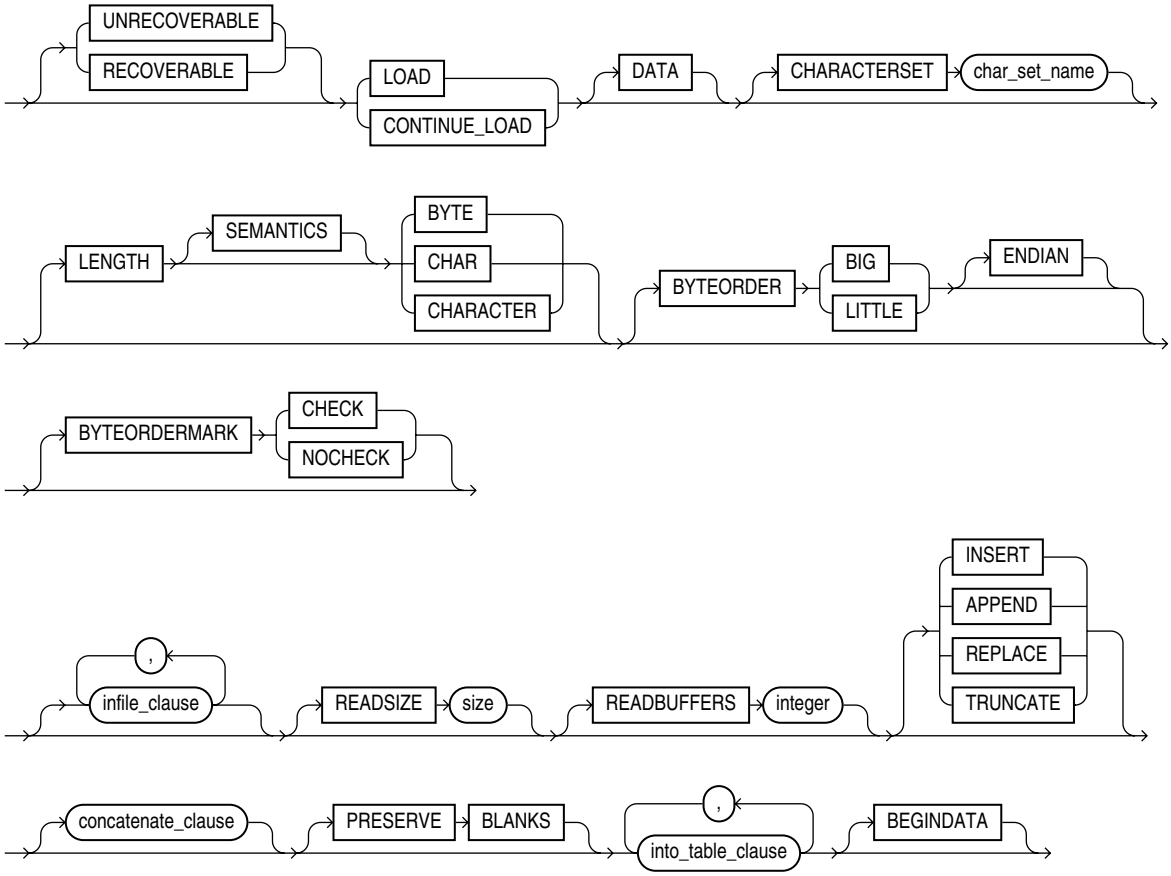
この付録では、SQL*Loader の構文を図形式（線路図または DDL 構文図ともいいます）で示します。ここで使用している構文の表記法の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

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。

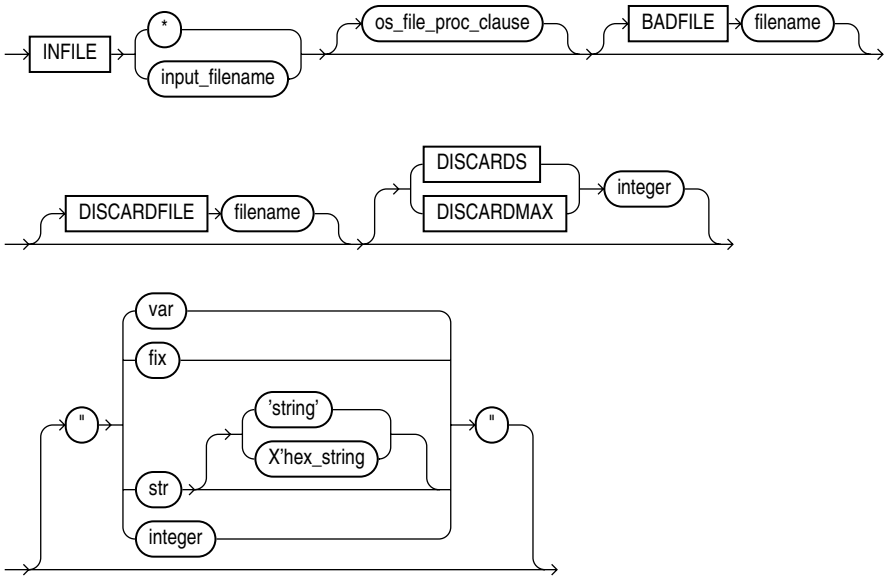
OPTIONS 句



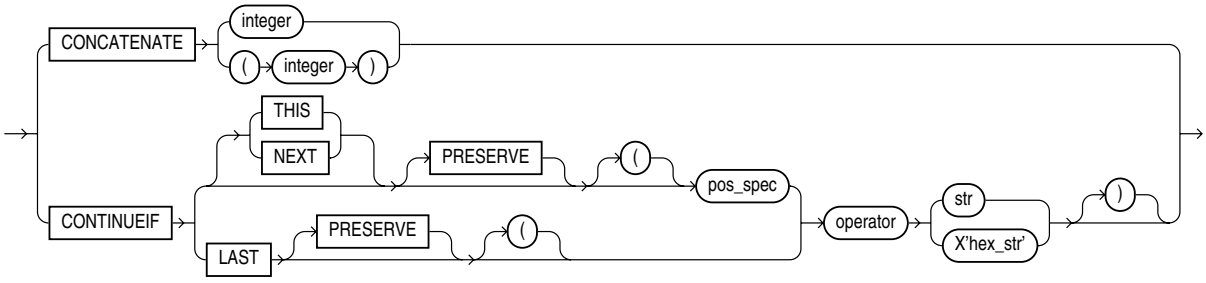
Load 文



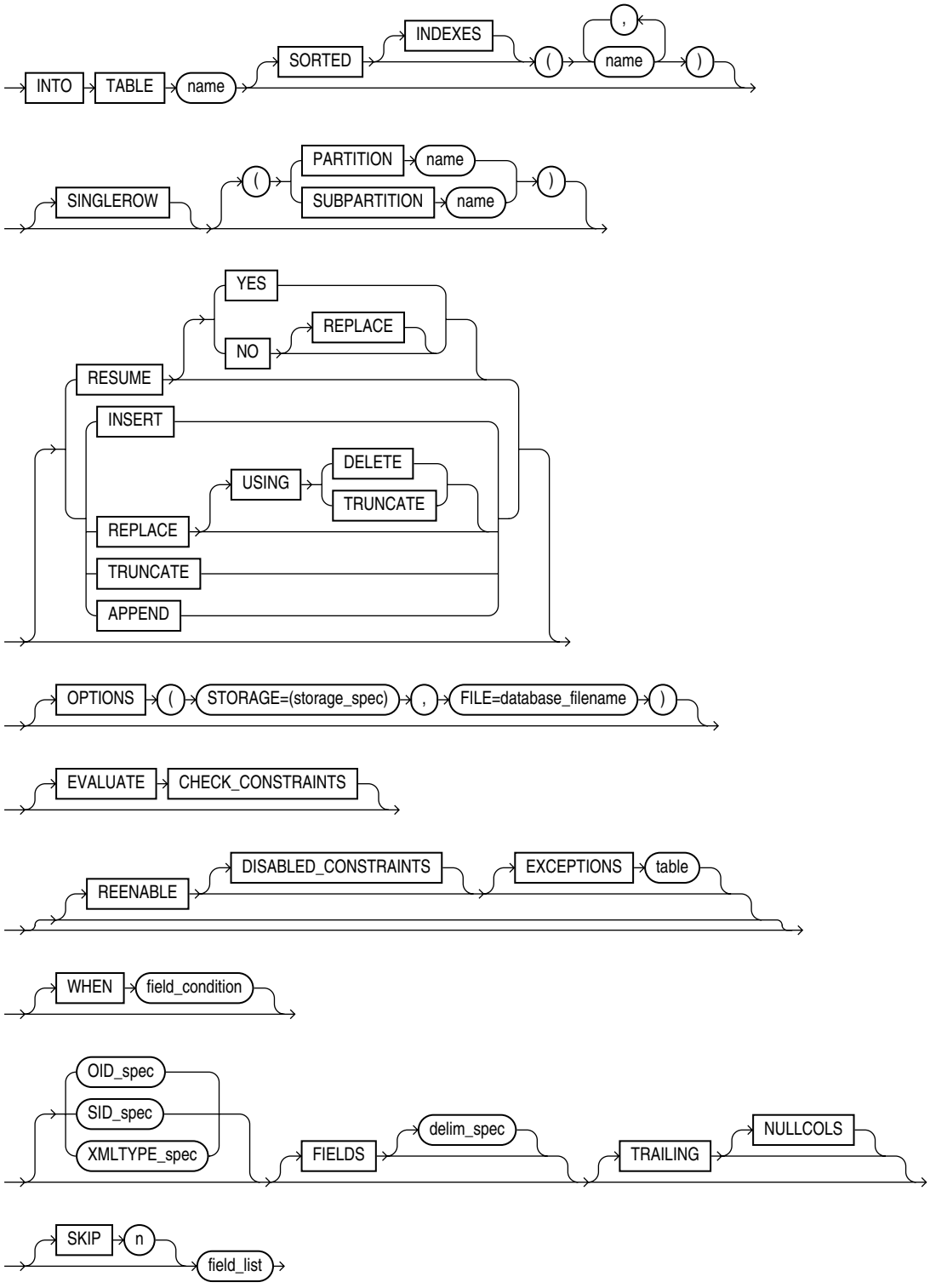
infile_clause



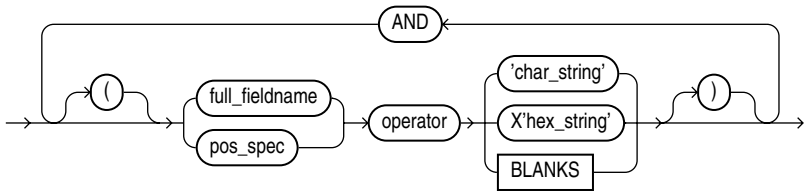
concatenate_clause



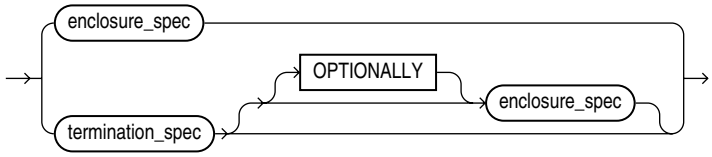
into_table_clause



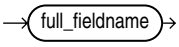
field_condition



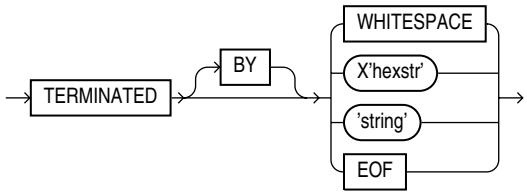
delim_spec



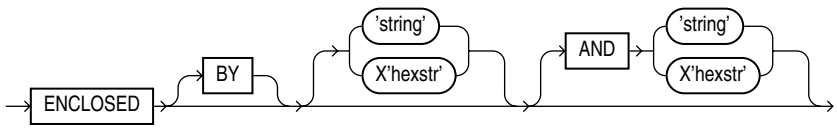
full_fieldname



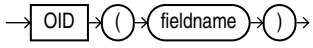
termination_spec



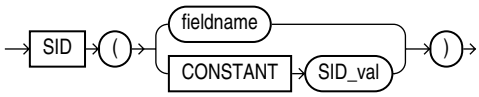
enclosure_spec



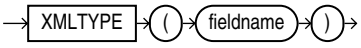
oid_spec



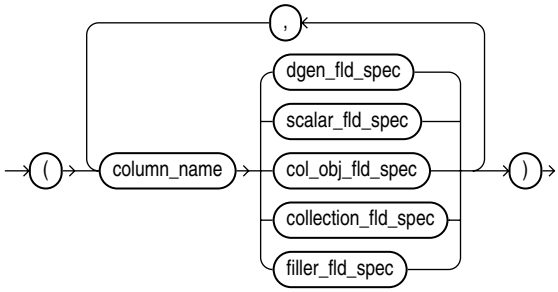
sid_spec



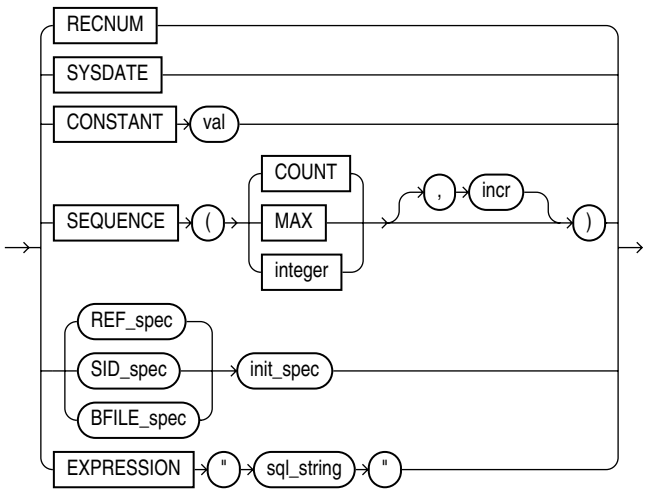
xmltype_spec



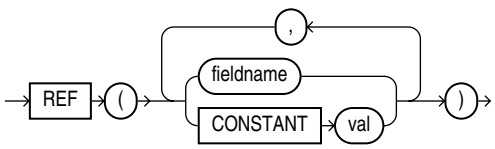
field_list



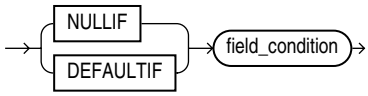
dgen_fld_spec



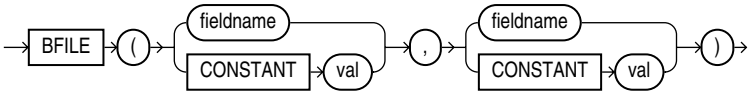
ref_spec



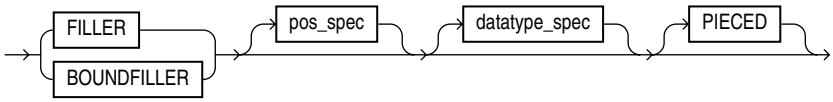
init_spec



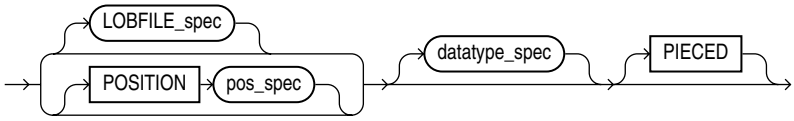
bfile_spec

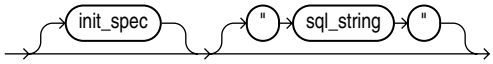


filler_fld_spec

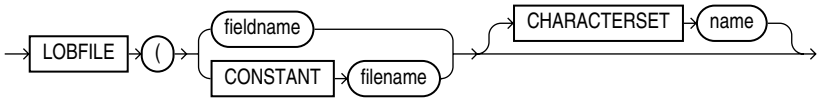


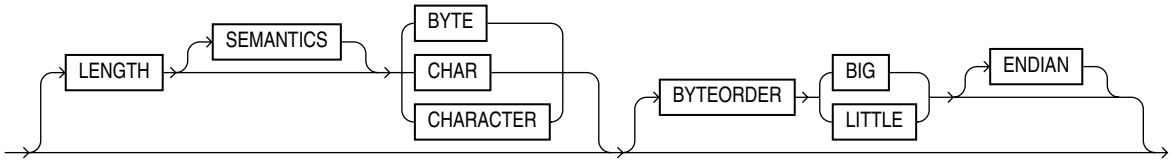
scalar_fld_spec

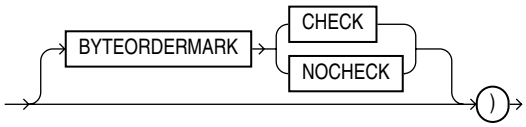




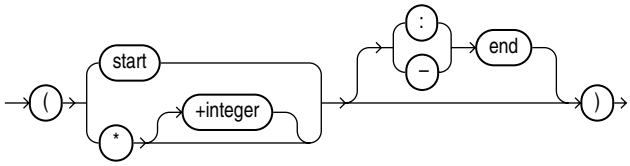
lobfile_spec



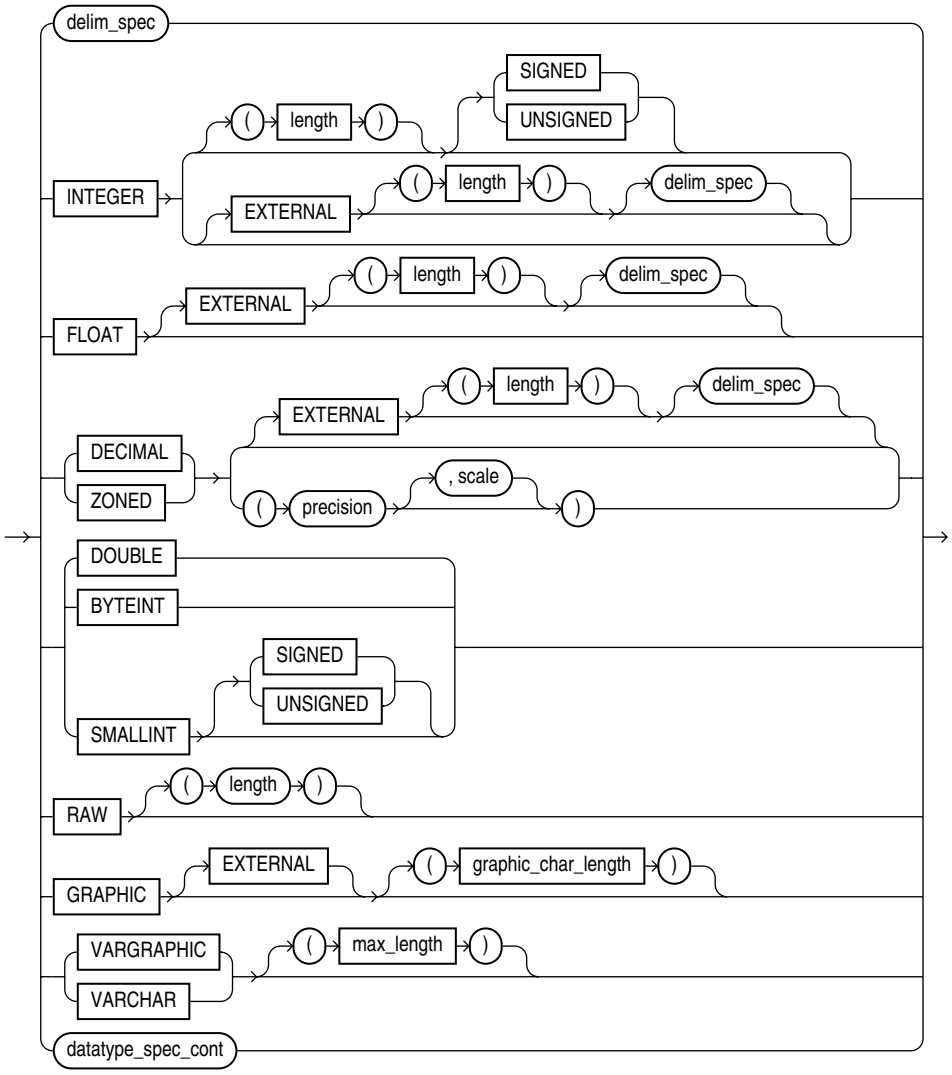




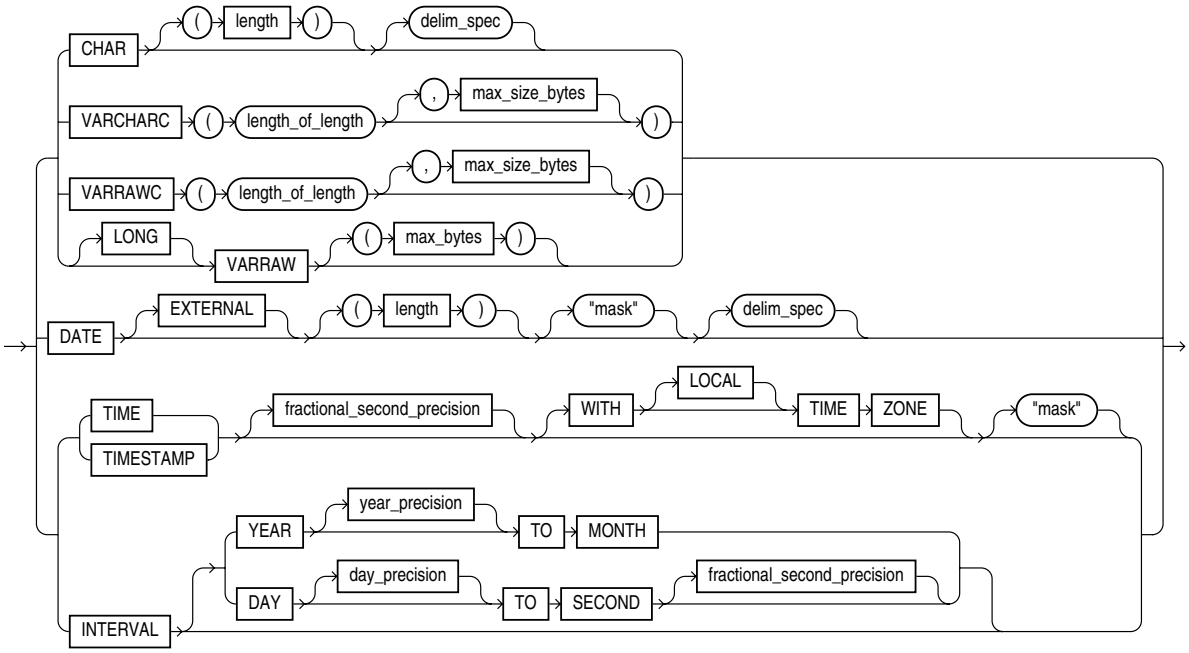
pos_spec



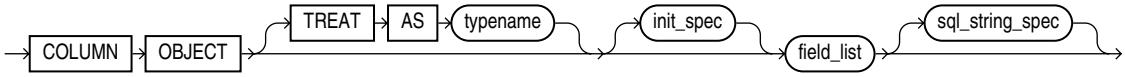
datatype_spec



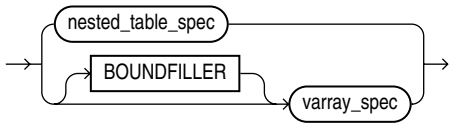
datatype_spec_cont



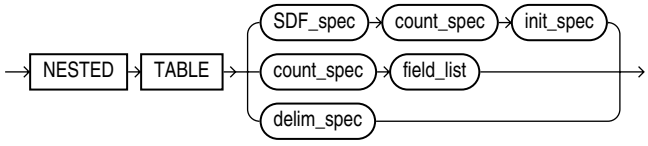
col_obj_fld_spec



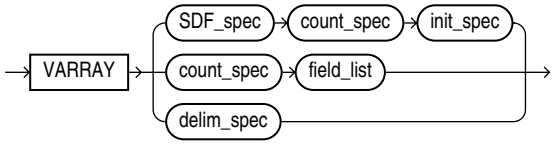
collection_fld_spec



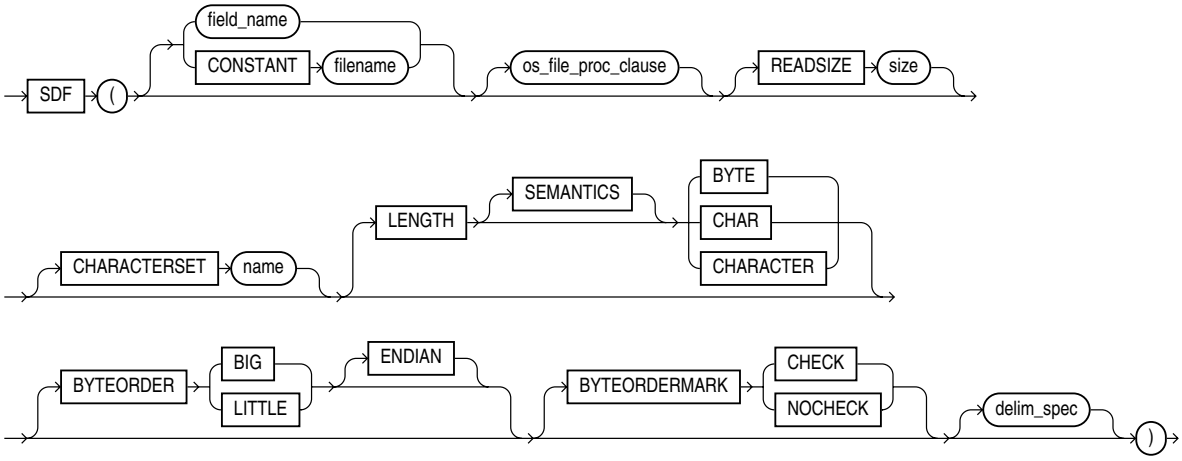
nested_table_spec



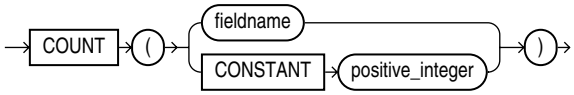
varray_spec



sdf_spec



count_spec



索引

数字

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

あ

い

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

て

と

な

に

ね

は

ひ

ふ

へ

ほ

ま

み

む

め

も

ゆ

よ

ら

り

れ

ろ