Oracle® Database

データベース概要

19c

F16110-06(原本部品番号:E96138-06)

2021年4月

タイトルおよび著作権情報

Oracle Databaseデータベース概要, 19c

F16110-06

Copyright © 1993, 2021, Oracle and/or its affiliates.

原本著者: Lance Ashdown、Donna Keesling、Tom Kyte、Joe McCormack

原本協力者: Drew Adams、Ashish Agrawal、Troy Anthony、Vikas Arora、Jagan Athraya、David Austin、Thomas Baby、Vladimir Barriere、Hermann Baer、Srinagesh Battula、Nigel Bayliss、Tammy Bednar、Virginia Beecher、Bjorn Bolltoft、Ashmita Bose、David Brower、Larry Carpenter、Lakshminaray Chidambaran、Deba Chatterjee、Shasank Chavan、Tim Chien、Gregg Christman、Bernard Clouse、Maria Colgan、Carol Colrain、Nelson Corcoran、Michael Coulter、Jonathan Creighton、Judith D'Addieco、Mark Dilman、Kurt Engeleiter、Bjørn Engsig、Marcus Fallon、Steve Fogel、Jonathan Giloni、Naveen Gopal、Bill Habeck 、Min-Hank Ho、Lijie Heng、Bill Hodak、Yong Hu、Pat Huey、Praveen Kumar Tupati Jaganath、Sanket Jain、Prakash Jashnani、Caroline Johnston、Shantanu Joshi、Jesse Kamp、Vikram Kapoor、Feroz Khan、Jonathan Klein、Andre Kruglikov、Sachin Kulkarni、Surinder Kumar、Paul Lane、Adam Lee、Allison Lee、Jaebock Lee、Sue Lee、Teck Hua Lee、Yunrui Li 、Ilya Listvinski、Bryn Llewellyn、Rich Long、Barb Lundhild、Neil Macnaughton、Vineet Marwah、Susan Mavris、Bob McGuirk、Joseph Meeks、Mughees Minhas、Sheila Moore、Valarie Moore、Gopal Mulagund、Charles Murray、Kevin Neel、Sue Pelski、Raymond Pfau、Gregory Pongracz、Vivek Raja、Ashish Ray、Bert Rich、Kathy Rich、Andy Rivenes、Scott Rotondo、Vivian Schupmann、Venkat Senaptai、Shrikanth Shankar、Prashanth Shanthaveerappa、Cathy Shea、Susan Shepard、Kam Shergill、Mike Skarpelos、Sachin Sonawane、James Spiller、Suresh Sridharan、Jim Stenoish、Janet Stern、Rich Strohm、Roy Swonger、Kamal Tbeileh、Juan Tellez、Ravi Thammaiah、Lawrence To、Tomohiro Ueda、Randy Urbano、Badhri Varanasi、Nick Wagner、Steve Wertheimer、Patrick Wheeler、Doug Williams、James Williams、Andrew Witkowski、Daniel Wong、Hailing Yu

目次

- タイトルおよび著作権情報
- はじめに
    - 対象読者
    - ドキュメントのアクセシビリティについて
    - 関連ドキュメント
    - 表記規則
- 『Oracle Database概要』のこのリリースでの変更点
    - Oracle Databaseリリース19cバージョン19.1での変更点
        - 新機能
        - サポート対象外となった機能
    - Oracle Databaseリリース18cバージョン18.1での変更点
        - 新機能
        - 非推奨となった機能
- 1 Oracle Databaseの概要
    - リレーショナル・データベースについて
        - データベース管理システム(DBMS)
        - リレーショナル・モデル
        - リレーショナル・データベース管理システム(RDBMS)
        - Oracle Databaseの歴史の概要
    - スキーマ・オブジェクト
        - 表
        - 索引
    - データ・アクセス
        - Structured Query Language(SQL)
        - PL/SQLとJava
    - トランザクション管理
        - トランザクション
        - データ同時実行性
        - データ整合性
    - Oracle Databaseアーキテクチャ
        - データベースおよびインスタンス
            - マルチテナント・アーキテクチャ
                - マルチテナント・アーキテクチャの利点
                - データベースの統合
                - アプリケーション・コンテナ
            - シャーディング・アーキテクチャ
        - データベース記憶域構造
            - 物理記憶域構造
            - 論理記憶域構造
        - データベース・インスタンス構造
            - Oracle Databaseプロセス
            - インスタンスのメモリー構造

- アプリケーションおよびネットワークのアーキテクチャ
  - アプリケーション・アーキテクチャ
  - Oracle Net Servicesのアーキテクチャ
- Oracle Databaseドキュメントのロードマップ
  - Oracle Databaseドキュメント: 初級グループ
  - Oracle Databaseドキュメント: 中級グループ
  - Oracle Databaseドキュメント: 上級グループ
- 第I部 Oracleリレーショナル・データ構造
  - 2 表と表クラスタ
    - スキーマ・オブジェクトの概要
      - スキーマ・オブジェクトのタイプ
      - スキーマ・オブジェクトの格納
      - スキーマ・オブジェクトの依存性
      - SYSスキーマとSYSTEMスキーマ
      - サンプル・スキーマ
    - 表の概要
      - 列
        - 仮想列
        - 非表示列
      - 行
      - 例: CREATE TABLE文とALTER TABLE文
      - Oracleデータ型
        - 文字データ型
          - VARCHAR2およびCHARデータ型
          - NCHARおよびNVARCHAR2データ型
        - 数値データ型
          - NUMBERデータ型
          - 浮動小数点数
        - 日時データ型
          - DATEデータ型
          - TIMESTAMPデータ型
        - ROWIDデータ型
          - ROWIDの使用
          - ROWID疑似列
        - 書式モデルとデータ型
      - 整合性制約
      - 表の記憶域
        - 表の編成
        - 行の格納
        - 行断片のROWID
        - NULL値の格納
      - 表圧縮
        - 基本表圧縮と高度な行圧縮
        - ハイブリッド列圧縮
          - ハイブリッド列圧縮のタイプ

- 圧縮ユニット
- DMLとハイブリッド列圧縮
- 表クラスタの概要
  - 索引付きクラスタの概要
  - ハッシュ・クラスタの概要
    - ハッシュ・クラスタの作成
    - ハッシュ・クラスタの問合せ
    - 様々なハッシュ・クラスタ
    - ハッシュ・クラスタの記憶域
- 属性クラスタ表の概要
  - 属性クラスタ表の利点
  - 結合属性クラスタ表
  - ゾーンを使用するI/Oの削減
    - ゾーン・マップ
    - ゾーンの目的
    - ゾーン・マップの仕組み：例
  - 線形順序付けを使用する属性クラスタ表
  - インターリーブされた順序付けを使用する属性クラスタ表
- 一時表の概要
  - 一時表の目的
  - 一時表でのセグメントの割当て
  - 一時表の作成
- 外部表の概要
  - 外部表の目的
  - オブジェクト・ストアのデータ
  - 外部表のアクセス・ドライバ
  - 外部表の作成
- ブロックチェーン表の概要
  - 行連鎖
  - 行内容
  - ブロックチェーン表のユーザー・インタフェース
- 不変表の概要
- オブジェクト表の概要

- 3 索引と索引構成表
  - 索引の概要
    - 索引の長所と短所
    - 索引の使用可能性と可視性
    - キーと列
    - コンポジット索引
    - 一意索引と非一意索引
    - 索引のタイプ
    - データベースで索引を管理する方法
    - 索引記憶域
  - Bツリー索引の概要
    - ブランチ・ブロックとリーフ・ブロック

- 索引スキャン
  - 全索引スキャン
  - 高速全索引スキャン
  - 索引レンジ・スキャン
  - 索引の一意スキャン
  - 索引スキップ・スキャン
  - 索引クラスタ化係数
- 逆キー索引
- 昇順索引と降順索引
- 索引の圧縮
  - 接頭辞圧縮
  - 拡張索引圧縮
- ビットマップ索引の概要
  - 例: 単一表でのビットマップ索引
  - ビットマップ結合索引
  - ビットマップ記憶域構造
- ファンクション索引の概要
  - ファンクション索引の使用方法
  - ファンクション索引による最適化
- アプリケーション・ドメイン索引の概要
- 索引構成表の概要
  - 索引構成表の特性
  - 行オーバーフロー領域付きの索引構成表
  - 索引構成表の2次索引
    - 論理ROWIDと物理推測
    - 索引構成表のビットマップ索引
- 4 パーティション、ビューおよびその他のスキーマ・オブジェクト
  - パーティションの概要
    - パーティションの特性
      - パーティション・キー
      - パーティション化計画
        - レンジ・パーティション化
        - 時間隔パーティション化
        - リスト・パーティション化
        - ハッシュ・パーティション化
        - 参照パーティション化
        - コンポジット・パーティション化
    - パーティション表
      - パーティション表のセグメント
      - パーティション表の圧縮
    - パーティション索引
      - ローカル・パーティション索引
        - ローカルの同一キーおよび非同一キー索引
        - ローカル・パーティション索引の記憶域
      - グローバル・パーティション索引

- パーティション表の部分索引
- シャード表の概要
  - シャード表
- ビューの概要
  - ビューの特性
    - ビューのデータ操作
    - ビュー内のデータへのアクセス方法
  - 更新可能な結合ビュー
  - オブジェクト・ビュー
- マテリアライズド・ビューの概要
  - マテリアライズド・ビューの特性
  - マテリアライズド・ビューのリフレッシュ方法
    - 完全リフレッシュ
    - 増分リフレッシュ
    - インプレース・リフレッシュおよびホーム外リフレッシュ
  - クエリー・リライト
- 順序の概要
  - 順序の特性
  - 順序への同時アクセス
- ディメンションの概要
  - ディメンションの階層構造
  - ディメンションの作成
- シノニムの概要

- 5 データ整合性
  - データ整合性の概要
    - データ整合性を保証する方法
    - 整合性制約の利点
  - 整合性制約のタイプ
    - NOT NULL整合性制約
    - 一意制約
    - 主キー制約
    - 外部キー制約
      - 自己参照型整合性制約
      - NULLと外部キー
      - 親キーの修正と外部キー
      - 索引と外部キー
    - チェック制約
  - 整合性制約の状態
    - 変更されたデータおよび既存のデータのチェック
    - データベースが制約の妥当性をチェックするタイミング
      - 遅延不可制約
      - 遅延可能制約
    - 制約チェックの例
      - 例: 親キー値が存在しない時点での外部キー列での値の挿入
      - 例: すべての外部キー値および親キー値の更新

- 6 データ・ディクショナリと動的パフォーマンス・ビュー
  - データ・ディクショナリの概要
    - データ・ディクショナリの内容
      - 接頭辞がDBA_のビュー
      - 接頭辞がALL_のビュー
      - 接頭辞がUSER_のビュー
      - DUAL表
    - データ・ディクショナリの記憶域
    - Oracle Databaseでのデータ・ディクショナリの使用方法
      - データ・ディクショナリ・ビューのパブリック・シノニム
      - データ・ディクショナリ・キャッシュ
      - 他のプログラムとデータ・ディクショナリ
  - 動的パフォーマンス・ビューの概要
    - 動的パフォーマンス・ビューの内容
    - 動的パフォーマンス・ビューの記憶域
  - データベース・オブジェクト・メタデータ
- 第II部 Oracleデータ・アクセス
  - 7 SQL
    - SQLの概要
      - SQLデータ・アクセス
      - SQL規格
    - SQL文の概要
      - データ定義言語(DDL)文
      - データ操作言語(DML)文
        - SELECT文
        - 結合
        - 副問合せ
      - トランザクション制御文
      - セッション制御文
      - システム制御文
      - 埋込みSQL文
    - オプティマイザの概要
      - オプティマイザの使用
      - オプティマイザ・コンポーネント
        - 問合せトランスフォーマ
        - エスティメータ
        - プラン・ジェネレータ
      - アクセス・パス
      - オプティマイザ統計
      - オプティマイザ・ヒント
    - SQL処理の概要
      - SQL処理の段階
        - SQLの解析
        - SQLの最適化
        - SQLの行ソース生成

- SQLの実行
- DML処理とDDL処理の違い
- 8 サーバー側プログラミング: PL/SQLとJava
  - サーバー側プログラミングの概要
  - PL/SQLの概要
    - PL/SQLサブプログラム
      - PL/SQLサブプログラムの利点
      - PL/SQLサブプログラムの作成
      - PL/SQLサブプログラムの実行
    - PL/SQLパッケージ
      - PL/SQLパッケージの利点
      - PL/SQLパッケージの作成
      - PL/SQLパッケージ・サブプログラムの実行
    - PL/SQL無名ブロック
    - PL/SQLの言語構造
    - PL/SQLのコレクションとレコード
      - コレクション
      - レコード
    - PL/SQLの実行方法
  - Oracle DatabaseにおけるJavaの概要
    - Java仮想マシン(JVM)の概要
      - Oracle JVMの概要
      - Oracle JVMの主要コンポーネント
    - Javaプログラミング環境
      - Javaストアド・プロシージャ
      - JavaおよびPL/SQLの統合
        - JDBCドライバ
        - SQLJ
  - トリガーの概要
    - トリガーの利点
    - トリガーのタイプ
    - トリガーのタイミング
    - トリガーの作成
      - 例: CREATE TRIGGER文
      - 例: 行レベル・トリガーの起動
    - トリガーの実行
    - トリガーの記憶域
- 第III部 Oracleトランザクションの管理
  - 9 データの同時実行性と整合性
    - データの同時実行性と整合性の概要
      - マルチバージョン読取り一貫性
        - 文レベルの読取り一貫性
        - トランザクション・レベルの読取り一貫性
        - 読取り一貫性とUNDOセグメント
          - 読取り一貫性: 例

- 読取り一貫性とInterested Transaction List
- 読取り一貫性および遅延挿入
- ロックのメカニズム
- ANSI/ISOトランザクション分離レベル
- Oracle Databaseのトランザクション分離レベルの概要
- コミット読取り分離レベル
- コミット読取り分離レベルでの読取り一貫性
- コミット読取りトランザクション内での書込みの競合
- シリアライズ可能分離レベル
- 読取り専用分離レベル
- Oracle Databaseのロック・メカニズムの概要
- ロック動作の概要
- ロックの使用
- ロック・モード
- ロック変換とロックの段階的拡大
- ロックの期間
- ロックとデッドロック
- 自動ロックの概要
- DMLロック
- 行ロック(TX)
- 行ロックと同時実行性
- 行ロックの記憶域
- 表ロック(TM)
- ロックと外部キー
- ロックと索引のない外部キー
- ロックと索引付き外部キー
- DDLロック
- 排他DDLロック
- 共有DDLロック
- ブレーク可能解析ロック
- システム・ロック
- ラッチ
- mutex
- 内部ロック
- 手動データ・ロックの概要
- ユーザー定義ロックの概要
- 10 トランザクション
- トランザクションの概要
- サンプル・トランザクション: 口座の借方と貸方
- トランザクションの構造
- トランザクションの開始
- トランザクションの終了
- 文レベルの原子性
- システム変更番号(SCN)
- トランザクション制御の概要

- トランザクション名
- アクティブなトランザクション
- セーブポイント
  - セーブポイントへのロールバック
  - トランザクションのエンキュー
- トランザクションのロールバック
- トランザクションのコミット
- トランザクション・ガードの概要
  - トランザクション・ガードの利点
  - トランザクション・ガードにより行われる処理
    - 失われたコミット・メッセージ
    - 論理トランザクションID
  - トランザクション・ガード: 例
- アプリケーション・コンティニュイティの概要
  - アプリケーション・コンティニュイティの利点
    - アプリケーション・コンティニュイティのユースケース
    - 計画的メンテナンスに対するアプリケーション・コンティニュイティ
  - アプリケーション・コンティニュイティ・アーキテクチャ
- 自律型トランザクションの概要
- 分散トランザクションの概要
  - 2フェーズ・コミット
  - インダウト・トランザクション

- 第IV部 Oracleデータベース記憶域構造
  - 11 物理記憶域構造
    - 物理記憶域構造の概要
      - データベース・ファイル格納のメカニズム
      - Oracle Automatic Storage Management(Oracle ASM)
        - Oracle ASMのストレージ・コンポーネント
        - Oracle ASMインスタンス
      - Oracle Managed FilesおよびユーザーS管理ファイル
    - データファイルの概要
      - データファイルの使用
      - 永続データファイルと一時データファイル
      - オンライン・データファイルとオフライン・データファイル
      - データファイルの構造
    - 制御ファイルの概要
      - 制御ファイルの使用
      - 多重制御ファイル
      - 制御ファイルの構造
    - オンラインREDOログの概要
      - オンラインREDOログの使用
      - Oracle Databaseでのオンライン REDOログの書込み方法
        - オンラインREDOログ・スイッチ
        - オンラインREDOログ・ファイルの複数コピー
        - アーカイブREDOログ・ファイル

- オンラインREDOログの構造
- 12 論理記憶域構造
  - 論理記憶域構造の概要
    - 論理記憶域階層
    - 論理領域管理
      - ローカル管理表領域
        - 自動セグメント領域管理
        - 手動セグメント領域管理
      - ディクショナリ管理表領域
  - データ・ブロックの概要
    - データ・ブロックとオペレーティング・システム・ブロック
      - データベース・ブロック・サイズ
      - 表領域ブロック・サイズ
    - データ・ブロックの形式
      - データ・ブロック・オーバーヘッド
      - 行形式
        - 行ヘッダー
        - 列データ
        - ROWID形式
    - データ・ブロックの圧縮
    - データ・ブロック内の領域管理
      - データ・ブロック内の空き領域の割合
      - データ・ブロック内の空き領域の最適化
        - 空き領域の拡大による最適化
        - 断片化領域の結合による最適化
      - 行の連鎖と移行
    - 索引ブロックの概要
      - 索引ブロックのタイプ
      - 索引エントリの記憶域
      - 索引ブロック内のスロットの再利用
      - 索引ブロックの結合
  - エクステントの概要
    - エクステントの割当て
    - エクステントの割当て解除
    - エクステントの記憶域パラメータ
  - セグメントの概要
    - ユーザー・セグメント
      - ユーザー・セグメントの作成
    - 一時セグメント
      - 問合せ用の一時セグメントの割当て
      - 一時表および索引用のセグメントの割当て
    - UNDOセグメント
      - UNDOセグメントとトランザクション
      - トランザクションのロールバック
      - 一時UNDOセグメント

- セグメント領域と最高水位標
- 表領域の概要
  - 永続表領域
    - SYSTEM表領域
    - SYSAUX表領域
    - UNDO表領域
      - 自動UNDO管理モード
      - 自動UNDO保存
    - シャドウ表領域
      - シャドウ表領域の目的
      - シャドウ表領域の機能
      - シャドウ表領域のユーザー・インタフェース
      - 例：書込み欠落保護の構成
  - 一時表領域
    - 共有およびローカル一時表領域
    - デフォルト一時表領域
      - デフォルト一時表領域の作成
      - 一時記憶域へのアクセス
  - 表領域モード
    - 読取り/書込み表領域と読取り専用表領域
    - オンライン表領域とオフライン表領域
  - 表領域ファイル・サイズ
- 第V部 Oracleインスタンス・アーキテクチャ
  - 13 Oracleデータベース・インスタンス
    - Oracleデータベース・インスタンスの概要
      - データベース・インスタンスの構造
      - データベース・インスタンス構成
      - 読取り/書込みインスタンスと読取り専用インスタンス
      - データベース・インスタンスの継続時間
      - データベース・インスタンスの識別
        - Oracleベース・ディレクトリ
        - Oracleホーム・ディレクトリ
        - Oracleシステム識別子(SID)
    - データベース・インスタンスの起動と停止の概要
      - インスタンスとデータベースの起動の概要
        - 管理者権限での接続
        - インスタンスの起動方法
        - データベースのマウント方法
        - データベースのオープン方法
          - 読取り専用モード
          - データベース・ファイルのチェック
      - データベースとインスタンスの停止の概要
        - 停止モード
        - データベースのクローズ方法
          - 正常停止時のデータベースのクローズ方法

- 異常停止時のデータベースのクローズ方法
- データベースのアンマウント方法
- インスタンスの停止方法
- チェックポイントの概要
  - チェックポイントの目的
  - Oracle Databaseでチェックポイントが開始される時期
- インスタンス・リカバリの概要
  - インスタンス・リカバリの目的
  - Oracle Databaseでインスタンス・リカバリが実行される時期
  - インスタンス・リカバリでのチェックポイントの重要性
  - インスタンス・リカバリのフェーズ
- パラメータ・ファイルの概要
  - 初期化パラメータ
    - 初期化パラメータの機能グループ
    - 基本的な初期化パラメータと拡張初期化パラメータ
  - サーバー・パラメータ・ファイル
  - テキスト形式の初期化パラメータ・ファイル
  - 初期化パラメータ値の変更
- 診断ファイルの概要
  - 自動診断リポジトリ
    - 問題とインシデント
    - ADRの構造
  - アラート・ログ
  - DDLログ
  - トレース・ファイル
    - トレース・ファイルのタイプ
    - トレース・ファイルの場所
    - トレース・ファイルのセグメント化
  - 診断ダンプ
    - トレース・ダンプおよびインシデント
- 14 メモリー・アーキテクチャ
  - Oracle Databaseのメモリー構造の概要
    - 基本的なメモリー構造
    - Oracle Databaseのメモリー管理
  - ユーザー・グローバル領域の概要
  - プログラム・グローバル領域(PGA)の概要
    - PGAの内容
      - プライベートSQL領域
      - SQL作業領域
    - 専用サーバー・モードおよび共有サーバー・モードにおけるPGAの使用
  - システム・グローバル領域(SGA)の概要
    - データベース・バッファ・キャッシュ
      - データベース・バッファ・キャッシュの目的
      - バッファ状態
      - バッファ・モード

- バッファI/O
  - バッファ置換アルゴリズム
  - バッファ書込み
  - バッファ読取り
  - バッファのタッチ・カウント
- バッファ・プール
- バッファと全表スキャン
  - 完全表スキャンのデフォルト・モード
  - パラレル問合せの実行
  - CACHE属性
  - KEEP属性
  - 強制全データベース・キャッシュ・モード
- REDOログ・バッファ
- 共有プール
  - ライブラリ・キャッシュ
    - 共有SQL領域
    - プログラム・ユニットとライブラリ・キャッシュ
    - 共有プール内のメモリーの割当てと再利用
  - データ・ディクショナリ・キャッシュ
  - サーバー結果キャッシュ
    - SQL問合せ結果キャッシュ
    - PL/SQLファンクション結果キャッシュ
  - 予約プール
- ラージ・プール
  - ラージ・プール・メモリー管理
  - 遅延挿入のためのラージ・プール・バッファ
- Javaプール
- 固定SGA
- オプションのパフォーマンス関連SGAサブ領域
  - インメモリー領域
  - memoptimizeプール
- ソフトウェア・コード領域の概要

- 15 プロセス・アーキテクチャ
  - プロセスの概要
    - プロセスのタイプ
    - マルチプロセスおよびマルチスレッド化されたOracle Databaseシステム
  - クライアント・プロセスの概要
    - クライアントおよびサーバー・プロセス
    - 接続とセッション
    - データベース操作
  - サーバー・プロセスの概要
    - 専用サーバー・プロセス
    - 共有サーバー・プロセス
    - Oracle Databaseによるサーバー・プロセスの作成方法
  - バックグラウンド・プロセスの概要

- 必須のバックグラウンド・プロセス
  - プロセス・モニター・プロセス(PMON)グループ
    - プロセス・モニター・プロセス(PMON)
    - クリーンアップ・メイン・プロセス(CLMN)
    - クリーンアップ・ヘルパー・プロセス(CLnn)
    - データベース・リソースの隔離
  - プロセス・マネージャ(PMAN)
  - リスナー登録プロセス(LREG)
  - システム・モニター・プロセス(SMON)
  - データベース・ライター・プロセス(DBW)
  - ログ・ライター・プロセス(LGWR)
    - LGWRおよびコミット
    - LGWRおよびアクセス不能ファイル
  - チェックポイント・プロセス(CKPT)
  - 管理性モニター・プロセス(MMONおよびMMNL)
  - リカバラ・プロセス(RECO)
- オプションのバックグラウンド・プロセス
  - アーカイバ・プロセス(ARCn)
  - ジョブ・キュー・プロセス(CJQ0およびJnnn)
  - フラッシュバック・データ・アーカイブ・プロセス(FBDA)
  - 領域管理コーディネータ・プロセス(SMCO)
- スレーブ・プロセス
  - I/Oスレーブ・プロセス
  - パラレル実行(PX)サーバー・プロセス
    - 問合せコーディネータ
    - プロデューサとコンシューマ
    - グラニュル
- 16 アプリケーションとOracle Net Servicesのアーキテクチャ
  - Oracleアプリケーション・アーキテクチャの概要
    - クライアント/サーバー・アーキテクチャの概要
      - 分散処理
      - クライアント/サーバー・アーキテクチャの利点
    - 複数層アーキテクチャの概要
      - クライアント
      - アプリケーション・サーバー
      - データベース・サーバー
      - サービス指向アーキテクチャ(SOA)
    - グリッド・アーキテクチャの概要
  - Oracle Net Servicesのアーキテクチャの概要
    - Oracle Net Servicesの動作
    - Oracle Net Listener
      - サービス名
      - サービス登録
    - 専用サーバー・アーキテクチャ
    - 共有サーバー・アーキテクチャ

- ディスパッチャのリクエスト・キューとレスポンス・キュー
- ディスパッチャ・プロセス(Dnnn)
- 共有サーバー・プロセス(Snnn)
- 共有サーバーの限定的運用
- データベース常駐接続プーリング
- プログラム・インタフェースの概要
- プログラム・インタフェースの構造
- プログラム・インタフェース・ドライバ
- オペレーティング・システムの通信ソフトウェア
- 第VI部 Oracle Databaseの管理およびアプリケーション開発
- 17 データベース管理者および開発者向けのトピック
- データベース・セキュリティの概要
- ユーザー・アカウント
- 権限
- ロール
- 権限分析
- ユーザー・プロファイル
- データベース認証
- 暗号化
- ネットワーク暗号化
- 透過的データ暗号化
- Oracle Data Redaction
- 方向
- Oracle Database Vault
- 仮想プライベート・データベース(VPD)
- Oracle Label Security(OLS)
- データ・アクセスの監視
- データベース監査
- 監査ポリシー
- 監査管理者ロール
- 統合監査証跡
- Enterprise Managerの監査サポート
- Oracle Audit Vault and Database Firewall。
- 高可用性の概要
- 高可用性と計画外停止時間
- サイト障害
- コンピュータ障害
- ストレージ障害
- データ破損
- 人為的エラー
- 高可用性と計画停止時間
- システムおよびデータベースの変更
- データ変更
- アプリケーション変更
- グリッド・コンピューティングの概要

- データベース・サーバー・グリッド
  - スケーラビリティ
  - フォルト・トレランス
  - サービス
- Oracle Flex Clusters
- データベース・ストレージ・グリッド
- データ・ウェアハウスとビジネス・インテリジェンスの概要
  - データ・ウェアハウスとOLTP
  - データ・ウェアハウスのアーキテクチャ
    - データ・ウェアハウスのアーキテクチャ(基本)
    - データ・ウェアハウスのアーキテクチャ(ステージング領域あり)
    - データ・ウェアハウスのアーキテクチャ(ステージング領域およびデータ・マートあり)
  - 抽出、変換、ロード(ETL)の概要
  - ビジネス・インテリジェンス
    - 分析SQL
    - 分析ビュー
    - OLAP
    - Oracle Advanced Analytics
      - Oracle Data Mining
      - Oracle R Enterprise
- Oracleにおける情報統合の概要
  - 統合されたアクセス
    - 分散SQL
    - データベース・リンク
  - 情報の共有
    - Oracle GoldenGate
    - Oracle Database Advanced Queuing(AQ)
      - メッセージのキューイングとデキューイング
      - Oracle Database Advanced Queuingの機能
- 18 データベース管理者の概念
  - データベース管理者の業務
  - データベース管理者向けのツール
    - Oracle Enterprise Manager
      - Oracle Enterprise Manager Cloud Control。
      - Oracle Enterprise Manager Database Express 12c
    - SQL*Plus
    - データベースのインストールおよび構成ツール
    - Oracle Netの構成および管理ツール
    - データの移動および分析ツール
      - SQL*Loader
      - Oracle Data Pumpエクスポートおよびインポート
      - Oracle LogMiner
      - ADRコマンド・インタプリタ(ADRCI)
  - データベース管理者のためのトピック
    - バックアップおよびリカバリ

- バックアップおよびリカバリ方法
- Recovery Managerのアーキテクチャ
- データベース・バックアップ
  - データベース全体のバックアップと部分バックアップ
  - 一貫性バックアップと非一貫性バックアップ
  - バックアップ・セットおよびイメージ・コピー
- データ修復
  - Oracle Flashbackテクノロジ
  - データ・リカバリ・アドバイザ
  - ブロック・メディア・リカバリ
  - データ・ファイル・リカバリ
- Zero Data Loss Recovery Appliance。
  - リカバリ・アプライアンスの利点
  - リカバリ・アプライアンス環境
- メモリー管理
  - 自動メモリー管理
  - SGAの共有メモリー管理
  - インスタンスPGAのメモリー管理
  - メモリー管理方法の概要
- リソース管理およびタスクのスケジューリング
  - データベース・リソース・マネージャ
  - Oracle Scheduler
- パフォーマンスおよびチューニング
  - データベースの自己監視
  - 自動ワークロード・リポジトリ(AWR)
  - 自動データベース・モニター(ADDM)
  - アクティブ・セッション履歴(ASH)
  - アプリケーションとSQLチューニング
    - EXPLAIN PLAN文
    - オプティマイザ統計アドバイザ
    - SQLチューニング・アドバイザ
    - SQLアクセス・アドバイザ
    - SQL計画管理
- 19 データベース開発者の概念
  - データベース開発者の業務
  - データベース開発者向けのツール
    - SQL Developer
    - Oracle Application Express
    - Oracle JDeveloper
    - Oracle Developer Tools for Visual Studio .NET
  - データベース開発者のためのトピック
    - アプリケーション設計およびチューニングの原則
    - クライアント側のデータベース・プログラミング
      - 埋込みSQL
        - Oracleプリコンパイラ

- SQLJ
- クライアント側のAPI
  - OCIおよびOCCI
  - ODBCおよびJDBC
- グローバリゼーション・サポート
  - グローバリゼーション・サポート環境
    - 文字セット
    - ロケール固有の設定
  - Oracle Globalization Development Kit
- 非構造化データ
  - Oracle DatabaseにおけるXMLの概要
  - Oracle DatabaseにおけるJSONの概要
    - JSONとは
    - JSONおよびXML
    - JSONに対するネイティブ・データベース・サポート
  - LOBの概要
    - 内部LOB
    - 外部LOB
    - SecureFiles
  - Oracle Textの概要
  - Oracle Spatial and Graphの概要

- 用語集
- 索引

はじめに

このマニュアルでは、オブジェクト・リレーショナル・データベース管理システムであるOracleデータベース・サーバーの、アーキテクチャや概念という側面の概要を説明します。

また、Oracleデータベース・サーバーがどのように機能しているのかを説明し、他のマニュアルで説明される多くの実用的な情報のための概念的な基盤を提供します。このマニュアルの情報は、すべてのオペレーティング・システム上で稼働するOracleデータベース・サーバーを対象にしています。

この章の内容は次のとおりです。

- 対象読者
- ドキュメントのアクセシビリティについて
- 関連ドキュメント
- 表記規則

対象読者

『*Oracle Database*概要』は、Oracle Databaseの新規ユーザーである技術者ユーザー(主にデータベース管理者とデータベース・アプリケーションの開発者)を対象にしています。このマニュアルの対象読者は、他のリレーショナル・データベース用アプリケーションの管理または開発の経験者です。

このマニュアルを使用するには、次の知識が必要です。

- リレーショナル・データベースの一般的な概念
- 「Oracle Databaseの概要」の概念と用語
- Oracleを実行しているオペレーティング・システムの環境

ドキュメントのアクセシビリティについて

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

Oracleサポートへのアクセス

サポートを購入したオラクル社のお客様は、My Oracle Supportを介して電子的なサポートにアクセスできます。詳細情報は(http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info)か、聴覚に障害のあるお客様は(http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs)を参照してください。

関連ドキュメント

このマニュアルは、次のマニュアルとともに読むことを前提にしています。

- 『*Oracle Database 2*日でデータベース管理者』
- 『*Oracle Database 2*日で開発者ガイド』

他の関連ドキュメントの詳細は、「Oracle Databaseドキュメントのロードマップ」を参照してください。

Oracle Databaseマニュアル・セットに含まれるマニュアルの多くでは、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データベースのサンプル・スキーマを使用しています。これらのスキーマがどのように作成されているか、およびその使

用方法については、『Oracle Databaseサンプル・スキーマ』を参照してください。

表記規則

このマニュアルでは、次の表記規則が使用されます。

規則	意味
太字	太字は、操作に関連付けられているグラフィカル・ユーザー・インタフェース要素、または本文中や用語集で定義されている用語を示します。
イタリック体	イタリックは、ユーザーが特定の値を指定するプレースホルダ変数を示します。
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。

『Oracle Database概要』のこのリリースでの変更点

この章の内容は次のとおりです。

- Oracle Databaseリリース19cバージョン19.1での変更点
- Oracle Databaseリリース18cバージョン18.1での変更点

Oracle Databaseリリース19cバージョン19.1での変更点

Oracle Database 19cバージョン19.1の『*Oracle Database*概要』には、次の変更点があります。

新機能

今回のリリースの主な新機能は次のとおりです。

- 遅延挿入

  アプリケーションは単一行挿入を、ラージ・プールに格納されている特殊なバッファにストリームできます。これらのバッファはデータベースにより非同期に書き込まれるため、この挿入は遅延されます。挿入は自動的にコミットされ、ロールバックできません。

  「遅延挿入のためのラージ・プール・バッファ」を参照してください。

- ハイブリッド・パーティション表

  表には内部パーティションと外部パーティションを含めることができます。内部パーティションは、データベース内の領域(セグメント)を消費します。外部パーティションは、外部表機能を使用してデータベースの外部に格納されます。たとえばsales表に、データ・ファイルに格納されている2つのパーティションと、テキスト・ファイルに格納されている3つのパーティションを含めることができます。

  「パーティション表のセグメント」を参照してください。

サポート対象外となった機能

次の機能は、このリリースではサポートされなくなりました。

- Oracle Multimedia

  Oracleデータベース19c以降では、Oracle Multimediaはサポートされなくなりました。選択を解除しないかぎり、アップグレードされたデータベースおよび新しく作成されたデータベースでは、dba_registryのOracle Multimediaコンポーネント(ORDIM)はVALIDとして表示されます。Oracle Locator (Spatial and Graphがインストールされていない場合にOracle Multimediaとともにインストールされる)は引き続きインストールおよびサポートされます。SecureFiles LOBにマルチメディア・コンテンツを格納し、イメージ処理および変換にサード・パーティ製品を使用することをお薦めします。

  「SecureFiles」を参照してください。

- Oracle Streams

  Oracleデータベース19c以降では、Oracle Streamsはサポートされなくなりました。

  「Oracle GoldenGate」を参照してください。

Oracle Databaseリリース18cバージョン18.1での変更点

Oracle Database 18cバージョン18.1の『*Oracle Database*概要』には、次の変更点があります。

新機能

今回のリリースの主な新機能は次のとおりです。

- memoptimizeプール

  Oracle Databaseでは、MEMOPTIMIZE FOR READ句で指定した表の高パフォーマンスの読取りをサポートしています。バッファは、memoptimizeプールと呼ばれる新しいSGA領域にキャッシュされます。

  memoptimizeプールを参照してください。

- プライベート一時表

  プライベート一時表は、トランザクションまたはセッションの終了時に自動的に削除される一時データベース・オブジェクトです。プライベート一時表はメモリー内に格納され、作成したセッションのみがそれを表示できます。プライベート一時表は一時表の有効範囲をセッションまたはトランザクションに制限するため、アプリケーションのコーディングをより柔軟に行えるようになり、コードのメンテナンスが簡単になり、すぐに使用できる機能の性能が向上します。

  一時表の概要を参照してください。

- シャドウ消失書込み保護

  書込み欠落は、記憶域にブロックが書き込まれたとデータベースで認識されているが、書込みが行われていなかった場合に発生します。シャドウ表領域を使用すると、スタンバイ・データベースを使用する必要なしに、カスケード・エラーが発生することなく、データベースが書込み欠落の大部分を把握できるようになります。

  シャドウ表領域を参照してください。

非推奨となった機能

次の機能は、このリリースでは非推奨であり、将来のリリースではサポートされなくなる可能性があります。

- Oracle Multimedia

  Oracleデータベース18c以降では、Oracle Multimediaは非推奨になりました。SecureFiles LOBにマルチメディア・コンテンツを格納し、イメージ処理および変換にサード・パーティ製品を使用することをお薦めします。

  「SecureFiles」を参照してください。

1 Oracle Databaseの概要

この章では、Oracle Databaseの概要について説明します。

この章のトピックは、次のとおりです:

- リレーショナル・データベースについて
- スキーマ・オブジェクト
- データ・アクセス
- トランザクション管理
- Oracle Databaseアーキテクチャ
- Oracle Databaseドキュメントのロードマップ

リレーショナル・データベースについて

すべての組織には、その要件を満たすために格納および管理する必要のある情報があります。たとえば、会社では従業員の人事管理レコードを収集し、管理する必要があります。この情報は、必要とする人が使用できるようにする必要があります。

情報システムとは、情報を格納および処理するための形式が定められたシステムです。情報システムは、フォルダを収納した一連の段ボール箱とフォルダの格納および取出し方法に関するルールとすることもできます。ただし、今日のほとんどの企業はデータベースを使用して情報システムを自動化しています。データベースとは、1つの単位として取り扱われる編成されたデータの集合です。データベースは、データベース・アプリケーションで使用するための関連情報の収集、格納、および取出しを目的としています。

データベース管理システム(DBMS)

データベース管理システム(DBMS)とは、データの格納、編成および取得を制御するソフトウェアです。

通常、DBMSの要素は次のとおりです。

- カーネル・コード

  このコードでは、DBMS用のメモリーと記憶域を管理します。

- メタデータのリポジトリ

  このリポジトリは、通常、データ・ディクショナリと呼ばれます。

- 問合せ言語

  この言語を使用してデータにアクセスできます。

データベース・アプリケーションとは、データにアクセスして操作するためにデータベースと対話するソフトウェア・プログラムです。

第一世代のデータベース管理システムには次のタイプがありました。

- 階層

  階層データベースは、データをツリー構造に編成したものです。ファイルシステムの構造と同様に、各親レコードが1つ以上の子レコードを持ちます。

- ネットワーク

  ネットワーク・データベースは、レコードが1対多の関係ではなく多対多の関係を持つ点を除いて、階層データベースと同

じです。

第一世代のデータベース管理システムでは、事前に定義された厳密な関係に基づいてデータが格納されていました。データ定義言語がなかったため、データ構造の変更は困難でした。また、これらのシステムには簡単な問合せ言語がなく、このことがアプリケーション開発の妨げになっていました。

リレーショナル・モデル

独創性に富んだ1970年の論文「A Relational Model of Data for Large Shared Data Banks」において、E. F. Codd氏は、数学的な集合理論に基づいてリレーショナル・モデルを定義しました。今日、最も普及しているデータベース・モデルはリレーショナル・モデルです。

リレーショナル・データベースとは、リレーショナル・モデルに準拠したデータベースです。リレーショナル・モデルには次の特徴があります。

- 構造

  詳細に定義されたオブジェクトによって、データベースのデータの格納またはアクセスが行われます。

- 操作

  明確に定義されたアクションによって、アプリケーションはデータベースのデータおよび構造を操作できます。

- 整合性規則

  整合性規則は、データベースのデータおよび構造に対する操作を制御します。

リレーショナル・データベースでは、データは一連の単純なリレーションに格納されます。リレーションは一連のタプルです。タプルは、順不同の一連の属性値です。

表は、行(タプル)と列(属性)の形式でのリレーションの2次元表現です。表の各行は同じ一連の列を持ちます。リレーショナル・データベースは、データをリレーション(表)に格納するデータベースです。たとえば、リレーショナル・データベースで会社の従業員に関する情報を従業員表、部門表、および給与表に格納できます。

関連項目:

Codd氏の論文の抜粋およびリンクは、『A Relational Model of Data for Large Shared Data Banks』を参照

リレーショナル・データベース管理システム(RDBMS)

リレーショナル・モデルは、**リレーショナル・データベース管理システム(RDBMS)**の基礎です。RDBMSではデータをデータベース内に移動して格納し、それを取り出してアプリケーションで操作できるようにします。

RDBMSでは、次のように操作のタイプが区別されます。

- 論理演算

  この場合、アプリケーションは、必要な内容を指定します。たとえば、従業員の名前を要求したり、従業員レコードを表に追加します。

- 物理演算

  この場合、RDBMSは、どのように実行すべきかを判断して演算を実行します。たとえば、アプリケーションが表に問合せを実行すると、データベースの索引を使用して要求した行が検索され、データがメモリーに読み込まれ、その他多くのス

テップが実行されてからユーザーに結果が戻されます。RDBMSがデータの格納と取得を行うため、物理演算はデータベース・アプリケーションに対して透過的です。

Oracle DatabaseはRDBMSです。ユーザー定義型、継承、ポリモフィズムなどのオブジェクト指向機能を実装したRDBMSは、オブジェクト・リレーショナル・データベース管理システム(ORDBMS)と呼ばれます。Oracle Databaseはリレーショナル・モデルを拡張したオブジェクト・リレーショナル・モデルであり、複雑なビジネス・モデルをリレーショナル・データベースに格納できます。

Oracle Databaseの歴史の概要

現行バージョンのOracle Databaseは、35年以上にわたる革新的な開発の成果です。

Oracle Databaseの発展における重要事項は次のとおりです。

- オラクル社の設立

  1977年、Larry Ellison、Bob Miner、およびEd Oatesはソフトウェア開発研究所でコンサルタント業を始め、Relational Software, Inc.(RSI)となりました。1983年、RSIはOracle Systems Corporationとなり、その後Oracle Corporationになりました。

- 最初の市販RDBMS

  1979年にRSIは、SQLベースの最初の市販RDBMSとしてOracle V2(バージョン2)を発表しました。これはリレーショナル・データベースの歴史において画期的な出来事でした。

- 移植性の高いOracle Databaseのバージョン

  1983年にリリースされたOracleバージョン3は、メインフレーム、ミニコンピュータ、パーソナル・コンピュータ上で動作する最初のリレーショナル・データベースでした。このデータベースはC言語で記述されており、複数のプラットフォームにデータベースを移植することが可能になりました。

- 同時実行性制御、データ配分、およびスケーラビリティの拡張

  バージョン4では、マルチバージョンの読取り一貫性が導入されました。1985年にリリースされたバージョン5では、クライアント/サーバー・コンピューティングおよび分散データベース・システムがサポートされました。バージョン6では、ディスクI/O、行ロック、スケーラビリティ、バックアップおよびリカバリに対する拡張が行われました。また、バージョン6では、SQLを手続き型に拡張した独自のPL/SQL言語の最初のバージョンが導入されました。

- PL/SQLストアド・プログラム・ユニット

  1992年にリリースされたOracle7では、PL/SQLストアド・プロシージャおよびトリガーが導入されました。

- オブジェクトとパーティション化

  Oracle8は、オブジェクト・リレーショナル・データベースとして1997年にリリースされ、多数の新しいデータ型をサポートしています。さらに、Oracle8で大規模な表のパーティション化がサポートされました。

- インターネット・コンピューティング

  1999年にリリースされたOracle8i Databaseには、インターネット・プロトコルのネイティブ・サポートおよびサーバー側でのJavaサポートが備えられていました。Oracle8iはインターネット・コンピューティング用に設計され、複数層環境へのデータベースのデプロイが可能になりました。

- Oracle Real Application Clusters(Oracle RAC)

  2001年のOracle9i Databaseでは、Oracle RACが導入され、複数のインスタンスによる単一データベースへの同時アクセスが可能になりました。さらに、Oracle XML Database(Oracle XML DB)ではXMLの格納および問合せの機能が導入されました。

- グリッド・コンピューティング

  2003年のOracle Database 10gでは、グリッド・コンピューティングが導入されました。このリリースでは、組織は低コストの商用サーバーに基づくグリッド・インフラストラクチャを構築することにより、コンピューティング資源を仮想化できるようになりました。これは、データベースの自己管理および自己チューニングを実現することを目標としていました。Oracle Automatic Storage Management(Oracle ASM)では、データベース・ストレージ管理を仮想化および簡素化することによって、この目標の実現を支援しました。

- 管理性、診断性、および可用性

  2007年にリリースされたOracle Database 11gでは、管理者および開発者が変化するビジネス要件に迅速に適応できるようにする多数の新機能が導入されました。適応のキーは、情報を統合し、できるかぎり自動化を採用することで、情報インフラストラクチャを簡素化することです。

- Cloudへの接続

  2013年にリリースされたOracle Database 12cは、Cloud用に設計され、新しいマルチテナント・アーキテクチャ、インメモリー列ストア、およびJSONドキュメントのサポートを備えています。Oracle Database 12cは、コストの削減およびユーザーのサービス・レベルの向上を継続して、DBAがITリソースを効率的に使用するのに役立ちます。

- 統合およびメモリー・パフォーマンス

  Oracle Database 18cでは、Microsoft Active Directoryなどのディレクトリ・サービスとの統合を簡素化しました。また、列データ・モデルだけでなく、高速行アクセスのためにメモリーを活用する機能も導入しました。

- 高度な安定性

  Oracle Database 19cは、Oracle Database 12c (リリース12.2)製品ファミリの長期サポート・バージョンです。このリリースでは、主に安定性に焦点が当てられています。また、Oracle Database 19cではJSONやActive Data Guardなどの機能に、小さいながらも重要な改良が行われています。

スキーマ・オブジェクト

RDBMSの特性の1つは、物理データ記憶域が論理データ構造から独立していることです。

Oracle Databaseにおいて、データベース・スキーマとは、論理データ構造またはスキーマ・オブジェクトの集合です。データベース・スキーマはデータベース・ユーザーによって所有され、そのユーザー名と同じ名前になります。

スキーマ・オブジェクトは、ユーザーが作成する構造であり、データベース内のデータを直接参照します。データベースは多くのタイプのスキーマ・オブジェクトをサポートしますが、最も重要なのは表と索引です。

スキーマ・オブジェクトは、データベース・オブジェクトの1種です。プロファイルやロールなど、一部のデータベース・オブジェクトは、スキーマに常駐しません。

関連項目:

スキーマ・オブジェクトのタイプ、記憶域および依存性についてさらに学習するには、「スキーマ・オブジェクトの概要」を参照

表

表は従業員などのエンティティを表します。

表は、employeesなどの表名と、列の集合で定義されます。通常は、表を作成するとき、各列に名前、データ型、および幅を

指定します。

表は行の集合です。列は、表として記述されるエンティティの属性を識別し、行は、エンティティのインスタンスを識別します。たとえば、従業員というエンティティの属性は、従業員IDや姓の列に対応します。行は特定の従業員を識別します。

オプションで、列に対して整合性制約と呼ばれる規則を指定できます。一例としてNOT NULLの整合性制約があります。この制約では、列のすべての行に値が含まれている必要があります。

関連項目:

- 列および行、データ型、表の記憶域および表の圧縮について学習するには、「表の概要」を参照
- 考えられる制約のタイプおよび状態について学習するには、「データ整合性」を参照

索引

索引は、表の1つ以上の列に対して作成可能なオプションのデータ構造です。索引によってデータ検索のパフォーマンスが向上する場合があります。

1つの要求を処理するとき、データベースでは、要求された行を効率よく見つけるために使用可能な索引を使用します。索引は、アプリケーションが特定の行または行範囲を頻繁に問い合せる場合に便利です。

索引は、データから論理的にも物理的にも独立しています。したがって、索引を削除または作成しても、表または他の索引に影響はありません。索引を削除しても、すべてのアプリケーションは機能し続けます。

関連項目:

索引の目的およびタイプについて学習するには、「索引の概要」を参照

データ・アクセス

DBMSの一般的な要件は、業界で受け入れられているデータ・アクセス言語の標準に準拠することです。

Structured Query Language(SQL)

SQLは、Oracle DatabaseなどのRDBMSとのインタフェースを提供する集合ベースの宣言型言語です。

Cなどの手続き型言語では、処理の実行方法を記述します。SQLは非手続き型であり、処理の実行内容を記述します。

SQLはリレーショナル・データベース用のANSI標準言語です。Oracle Database内のデータの操作はすべて、SQL文を使用して実行されます。たとえば、SQLを使用して表を作成し、表内のデータを問い合せ、変更します。

SQL文は非常に簡単な文ですが、強力なコンピュータ・プログラム、または命令と考えることができます。ユーザーは、求める結果(たとえば、従業員の名前)を指定し、その結果をどのようにして導出するかは指定しません。SQL文は、次のようなSQLテキストの文字列です。

```
SELECT first_name, last_name FROM employees;
```

SQL文を使用して次のタスクを実行できます。

- データの問合せ

- 表内の行の挿入、更新および削除
- オブジェクトの作成、置換、変更および削除
- データベースおよびそのオブジェクトへのアクセスの制御
- データベースの整合性と一貫性の保証

SQLでは、前述のタスクを1つの一貫した言語に統合します。Oracle SQLは、ANSI標準の実装の1つです。Oracle SQLでは、標準SQLを拡張した数多くの機能がサポートされています。

関連項目:

SQL標準およびSQL文の主要なタイプについてさらに学習するには、「SQL」を参照

PL/SQLとJava

PL/SQLは、Oracle SQLに対する手続きを拡張します。

PL/SQLはOracle Databaseに統合されており、Oracle DatabaseのすべてのSQL文、関数、およびデータ型を使用できます。PL/SQLを使用すると、SQLプログラムのフローの制御、変数の使用、およびエラー処理プロシージャの作成を行うことができます。

PL/SQLの主な利点は、アプリケーション・ロジックをデータベースそのものに格納できることです。PL/SQLプロシージャまたはファンクションは、一連のSQL文と他のPL/SQL構文で構成され、グループとしてまとめてデータベースに格納されるスキーマ・オブジェクトであり、特定の問題を解決したり、一連の関連タスクを実行するために1つの単位として実行されます。サーバー側プログラミングの主な利点は、組込み機能を任意の場所にデプロイできることです。

Oracle Databaseは、Javaで作成されたプログラム単位も格納できます。Javaストアド・プロシージャは、SQLにパブリッシュされ、一般的な用途向けにデータベースに格納されるJavaメソッドです。既存のPL/SQLプログラムをJavaから呼び出したり、JavaプログラムをPL/SQLから起動できます。

関連項目:

- 「サーバー側プログラミング: PL/SQLとJava」
- 「クライアント側のデータベース・プログラミング」

トランザクションの管理

Oracle Databaseはマルチユーザー・データベースとして設計されています。データベースは、複数のユーザーが互いのデータを破損することなく同時に作業できることを保証する必要があります。

トランザクション

トランザクションは、1つ以上のSQL文を含むアトミックな論理作業単位です。

RDBMSは、SQL文をグループ化して、すべての文がコミットされた(つまりデータベースに適用された)状態にするか、すべての文がロールバックされた(つまり取り消された)状態にできる必要があります。

トランザクションが必要な例として、普通預金口座から当座預金口座への振替があります。振替は独立した次の操作で構成されます。

1. 普通預金口座の減額
2. 当座預金口座の増額
3. トランザクション・ジャーナルへのトランザクションの記録

Oracle Databaseは、3つの操作すべてが1つの単位として成功したこと、または失敗したことを保証します。たとえば、ハードウェア障害によってトランザクション内の文の1つを実行できなかった場合は、その他の文をロールバックする必要があります。

トランザクションは、Oracle Databaseとファイルシステムとの明らかな違いを示す1つの機能です。複数のファイルを更新する不可分な操作を実行する場合、その途中でシステム障害が発生すると、ファイル間の整合性が失われます。対照的に、トランザクションは、Oracle Databaseを矛盾のない1つの状態から矛盾のない別の状態に移行します。トランザクションの基本的な原則は、100かゼロかであり、不可分の操作はその全体が成功するかその全体が失敗します。

関連項目:

トランザクションの定義、文レベルの原子性およびトランザクション制御について学習するには、「トランザクション」を参照

データ同時実行性

マルチユーザーRDBMSの要件の1つに、**データ同時実行性**(同じデータに複数のユーザーが同時にアクセスすること)の制御があります。

同時実行性が制御されていない場合、ユーザーがデータを不適切に変更し、データ整合性が損われることがあります。たとえば、あるユーザーが行を更新する一方で、別のユーザーがその行を同時に更新することがあります。

複数のユーザーが同じデータにアクセスした場合、同時実行性を管理する1つの方法は、ユーザーを待機させることです。ただし、DBMSの目標は待機時間をゼロまたはごくわずかまで減らすことです。データを変更するすべてのSQL文は、できるだけ干渉されない状態で処理する必要があります。破壊的な相互作用とは、不正確なデータの更新または基礎となるデータ構造の不正な変更を引き起こす相互作用であり、これを回避する必要があります。

Oracle Databaseは、ロックを使用してデータへの同時アクセスを制御します。ロックは、共有リソースにアクセスする複数のトランザクション間で、破壊的な相互作用が起きないようにするメカニズムです。ロックは、データ整合性を保証する場合に役立つ一方、データへの最大の同時アクセスを可能にします。

関連項目:

「Oracle Databaseのロック・メカニズムの概要」

データ整合性

Oracle Databaseでは、各ユーザーに対して、ユーザー自身のトランザクションによる変更および他のユーザーのトランザクションのコミットによる変更を反映した、整合性のあるデータのビューが表示される必要があります。

たとえば、データベースは、あるトランザクションが別の同時トランザクションによるコミットされていない変更に遭遇した場合に発生する内容を保証しない読取りの問題を防止する必要があります。

Oracle Databaseでは、文レベルの読取り一貫性を適用することにより、ある特定の時点で1つの問合せによって戻されたデータがコミット済であり、一貫していることを保証しています。トランザクション分離レベルに応じて、この時点は文またはトランザクションが開始されたときになります。この時点は、Oracle Flashback Query機能で明示的に指定できます。

データベースは、トランザクションのすべての問合せに対して、トランザクション・レベルの読取り一貫性と呼ばれる読取り一貫性を実現することもできます。この場合、トランザクション内の各文には同じ時点(トランザクション開始時)のデータが表示されます。

関連項目:

- 内容を保証しない読取りについてさらに学習するには、データの同時実行性と整合性を参照
- Oracle Flashback Queryについて学習するには、『Oracle Database開発ガイド』を参照

Oracle Databaseアーキテクチャ

データベース・サーバーは、情報管理のキーとなります。

一般的にサーバーでは、多数のユーザーが同時に同じデータへアクセスできるように、マルチユーザー環境において大量のデータが確実に管理されます。権限のないアクセスに対して保護機能を備えながら、障害のリカバリについても効率のよい解決方法を提供します。

データベースおよびインスタンス

Oracleデータベース・サーバーは、1つのデータベースと1つ以上の**データベース・インスタンス**(通常は単に**インスタンス**と呼ばれる)で構成されます。

インスタンスとデータベースは非常に密接に関連しているため、**Oracleデータベース**という用語が、インスタンスとデータベースの両方を示す場合があります。これらの用語の厳密な意味は次のとおりです。

- データベース

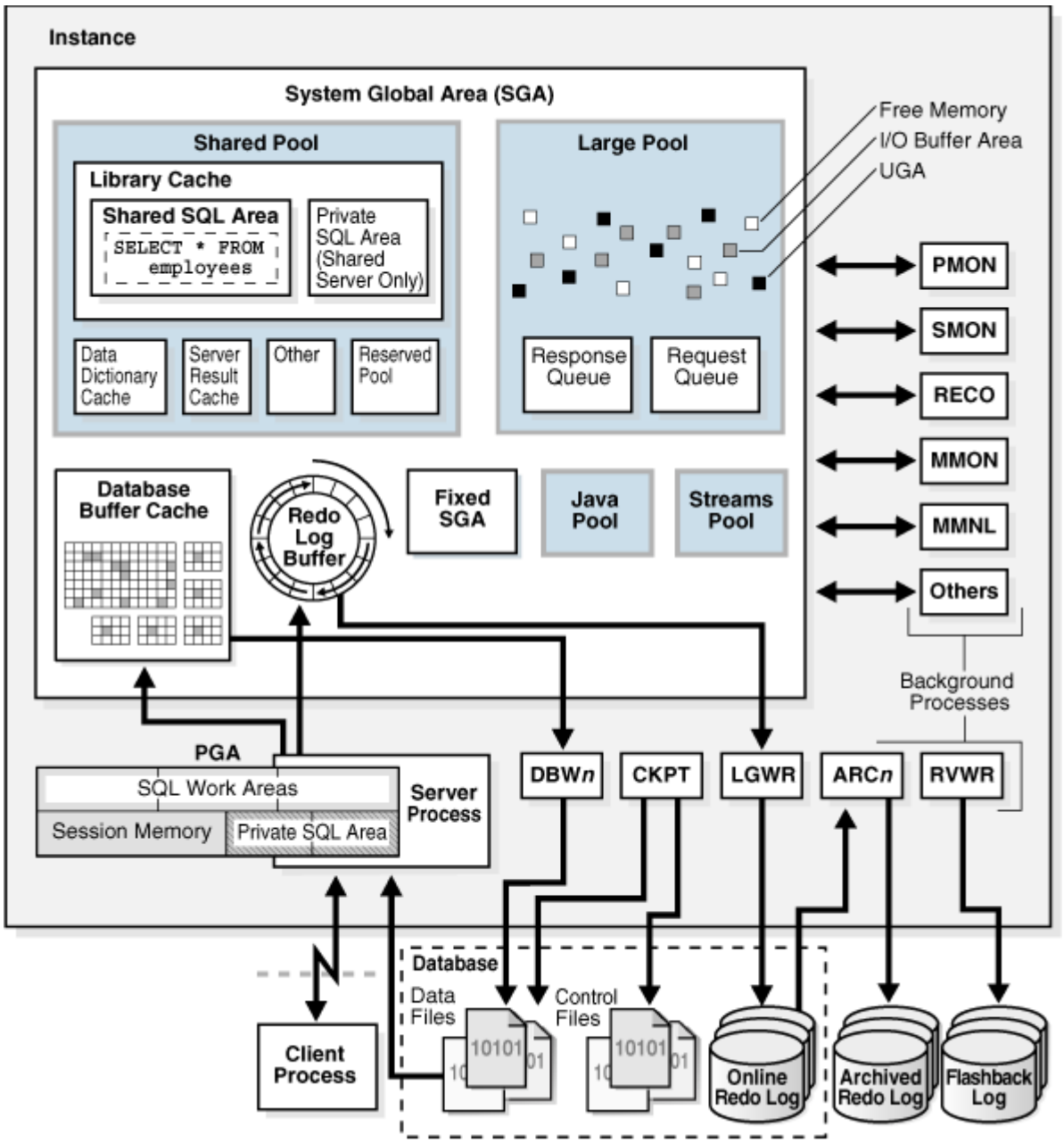
  データベースは、ディスク上に配置された、データを格納する一連のファイルです。これらのファイルはデータベース・インスタンスとは別に存在できます。

- データベース・インスタンス

  インスタンスは、データベース・ファイルを管理する一連のメモリー構造です。インスタンスは、システム・グローバル領域(SGA)と呼ばれる共有メモリー領域と、一連のバックグラウンド・プロセスで構成されます。インスタンスはデータベース・ファイルとは別に存在できます。

次の図に、データベースとそのインスタンスを示しています。インスタンスに対するそれぞれのユーザー接続に対して、クライアント・プロセスにより、アプリケーションが実行されます。各クライアント・プロセスは、独自のサーバー・プロセスに関連付けられています。サーバー・プールには、プログラム・グローバル領域(PGA)と呼ばれる独自のプライベート・セッション・メモリーがあります。

図1-1 Oracleインスタンスおよびデータベース



Oracleデータベースは厳密には一連の物理構造(ファイルとメモリーの構造)ですが、アプリケーションは、単一の物理データベース内にある複数の論理データベースと対話したり、複数の物理データベース間に分散された単一の論理データベースと対話したりできます。

この項では、次の項目について説明します。

- マルチテナント・アーキテクチャ
- シャーディング・アーキテクチャ

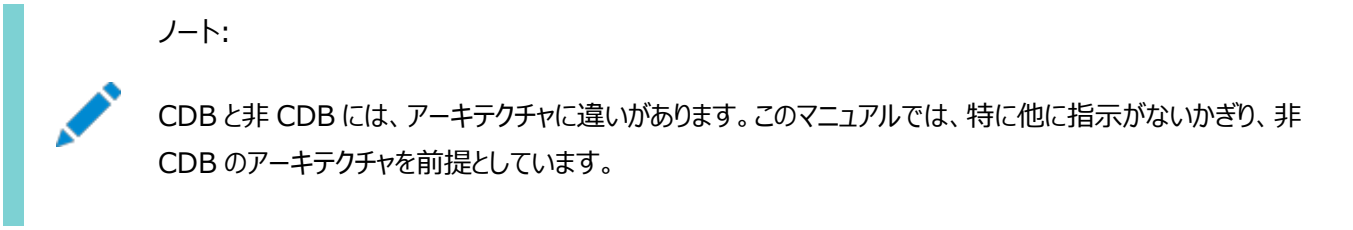
関連項目:

「Oracleデータベース・インスタンス」

マルチテナント・アーキテクチャ

マルチテナント・アーキテクチャを使用すると、Oracle Databaseをマルチテナント・コンテナ・データベース(CDB)にすることができます。

非CDBは、PDBを含めることができない従来のOracleデータベースです。CDBは、ユーザーが作成したプラガブル・データベースが0個、1個または多数含まれる単一の物理データベースです。プラガブル・データベース(PDB)は、Oracle Netクライアントに非CDBとして表示されるスキーマ、スキーマ・オブジェクトおよび非スキーマ・オブジェクトの移植可能な集合です。



ノート:

CDB と非 CDB には、アーキテクチャに違いがあります。このマニュアルでは、特に他に指示がないかぎり、非 CDB のアーキテクチャを前提としています。

マルチテナント・アーキテクチャの利点

マルチテナント・アーキテクチャによって、従来の非CDBアーキテクチャが引き起こす多くの問題が解決されます。

大企業では数百から数千ものデータベースを使用し、多くの場合、これらは複数の物理サーバー上の様々なプラットフォームで稼働している可能性があります。最近のサーバーは以前より高い負荷を処理できるようになりました。1つのデータベースで使用するサーバー・ハードウェアの容量はごく一部です。この方法では、ハードウェアと人事管理の両方が無駄になります。

マルチテナント・アーキテクチャでは、別々のコンピュータにある複数の物理データベースを1つのコンピュータ上の1つのデータベースに統合することにより、次の利点が得られます。

- ハードウェアのコスト削減
- より簡単かつ高速なデータおよびコードの移動
- 物理データベースの容易な管理および監視
- データおよびコードの分離
- 権限を付与されているPDBのみを管理するPDB管理者と、CDB全体を管理するCDB管理者との業務の分離

管理性には次のような利点があります。

- PDBの切断と接続によるデータおよびコードのアップグレードの簡略化
- PDBを本番CDBに接続する前に開発で使用することによる、テストの簡略化
- 個々のPDBを前のSCNまでフラッシュ・バックする機能
- PDBレベルでメモリーおよびI/Oのパフォーマンス制限を設定する機能
- 共通のアプリケーション・ルートに接続されたPDBのセットである、アプリケーション・コンテナ内のマスターアプリケーション定義をインストール、アップグレードおよび管理する機能

関連項目:

マルチテナント・アーキテクチャの利点についてさらに学習するには、Oracleマルチテナント管理者ガイドを参照してください。

データベースの統合

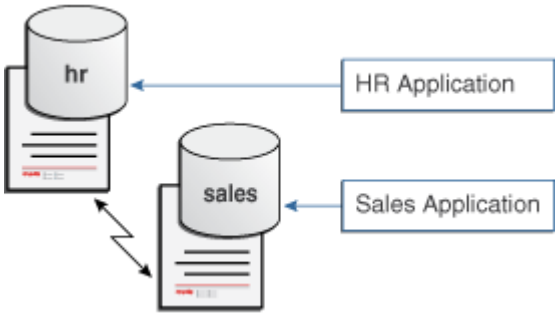
データベースの統合とは、1つ以上の非CDBのデータを1つのCDBに移動する一般プロセスです。

Oracle Database 12cから、データベースは、CDBか非CDBのいずれかとして作成する必要があります。従来の非CDBをPDBとしてCDBに接続できます。PDB/非CDB互換性保障とは、Oracle Netで接続しているクライアントから、PDBと非

CDBの動作が同じように見えることです。

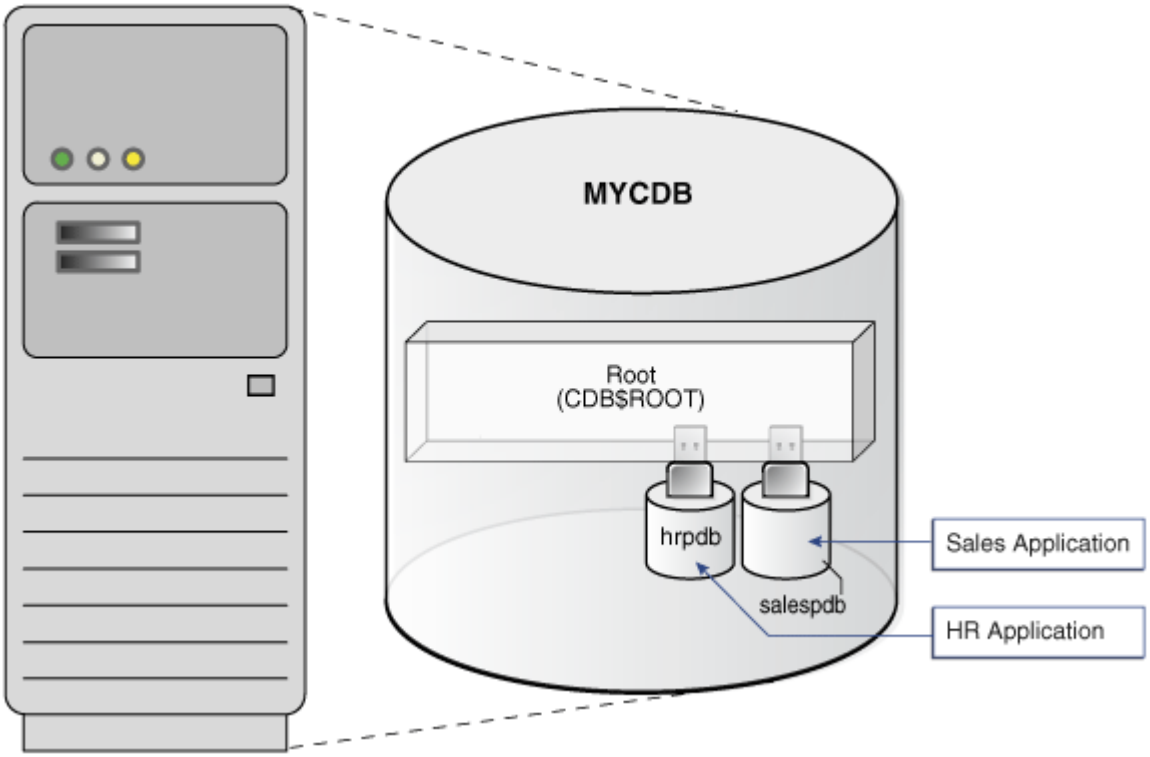
以下の図に、2つの別々の非CDB、hrとsalesを示します。各非CDBは、それぞれのメモリーとデータベース・ファイルのセットを持ち、それぞれのコンピュータに常駐します。各非CDBには、独自の専用ユーザー・アプリケーションがあります。

図1-2 非CDB



以下の図に、MYCDBという名前のCDBに統合後の同じデータを示します。

図1-3 非CDBのCDBへの統合



物理的に、MYCDBは1つのOracleデータベースです。MYCDBには、非CDBと同様に、1つのデータベース・インスタンス(ただし、Oracle Real Application Clustersでは複数のインスタンスが可能)と、1つのデータベース・ファイル・セットがあります。

MYCDBには2つのPDB、hrpdbとsalespdbが含まれます。図1-3に示すように、これらのPDBは、それぞれのアプリケーションには、データベース統合前と同様に表示されます。CDB内でCDB自身またはPDBを管理するために、CDB管理者は、すべてのPDBが属するスキーマ、スキーマ・オブジェクトおよび非スキーマ・オブジェクトの集合であるCDBルートに接続できます。

関連項目:

非CDBをCDBに統合する方法を学習するには、Oracleマルチテナント管理者ガイドを参照してください。

アプリケーション・コンテナ

Oracle Database 12cリリース2 (12.2)以降では、アプリケーション・コンテナはユーザーが任意に作成するコンテナであり、1

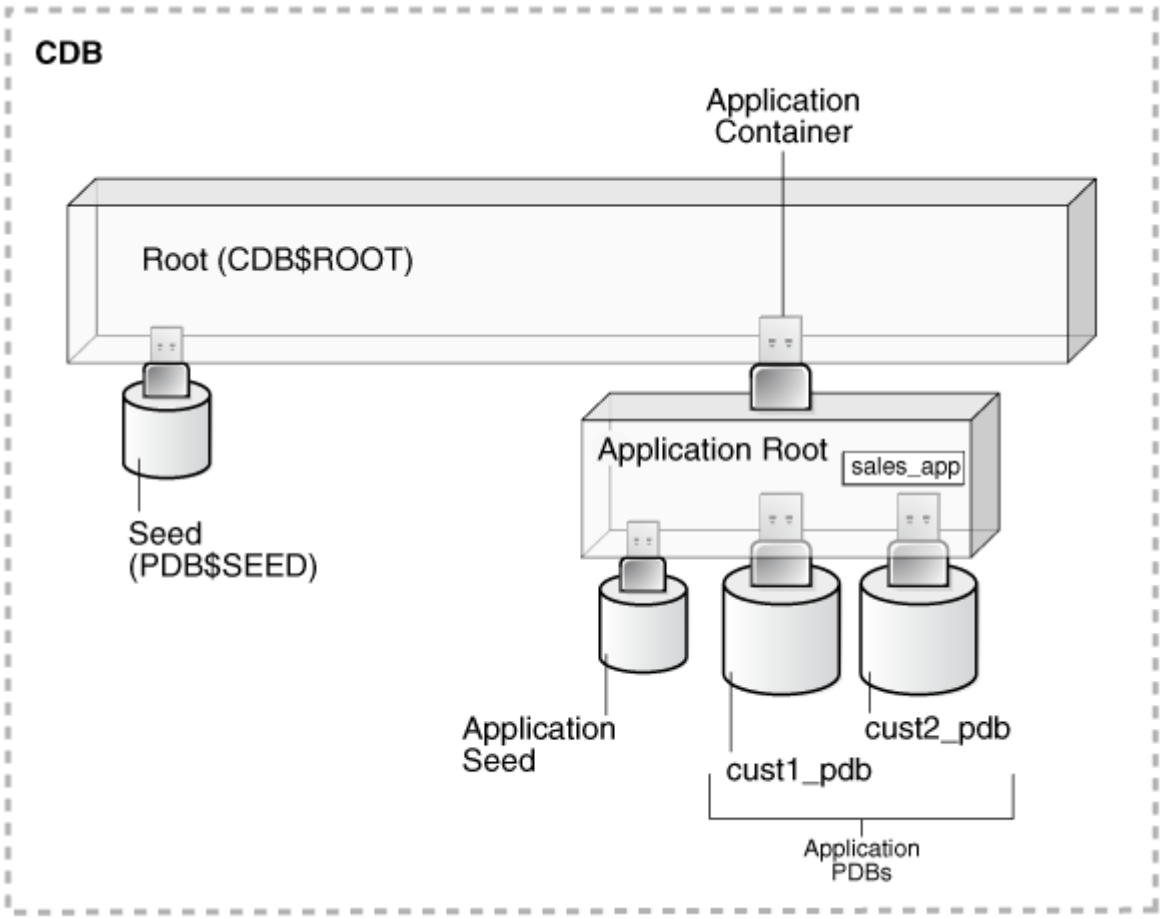
つ以上のアプリケーション・モデルのデータおよびメタデータが格納されます。

アプリケーション(アプリケーション・モデルとも呼ばれる)は、アプリケーション・ルートに格納される共通データおよびメタデータの名前付けおよびバージョニングされたセットです。たとえば、アプリケーション・モデルには一連のPDBに共通の表、ビュー、ユーザー・アカウント、PL/SQLパッケージの定義を含めることができます。

アプリケーション・コンテナはある意味で、CDB内のアプリケーション固有CDBとして機能します。アプリケーション・コンテナにはCDB自身のように複数のアプリケーションPDBが含まれ、これらのPDBはメタデータおよびデータを共有できます。

たとえば、SaaSデプロイメントでは、個々が別の顧客用で、アプリケーションのメタデータおよびデータを共有する複数のアプリケーションPDBを使用できます。たとえば次の図では、sales_appはアプリケーション・ルート内のアプリケーション・モデルです。cust1_pdbというアプリケーションPDBには顧客1用の販売データのみが含まれる一方、cust2_pdbというアプリケーションPDBには顧客2用の販売データのみが含まれます。接続、切断、クローニングおよびその他のPDBレベルの操作を、個々の顧客PDBで使用できます。

図1-4 SaaSのユースケース



関連項目:

アプリケーション・コンテナの概要については、Oracleマルチテナント管理者ガイドを参照してください。

シャーディング・アーキテクチャ

Oracle Shardingは、複数のOracleデータベース間におけるデータの水平パーティション化に基づくデータベース・スケーリング技術です。アプリケーションは、データベースのプールを単一の論理データベースとして認識します。

OLTPアプリケーションのシャーディングの主な利点には、垂直スケーラビリティ、障害の封じ込め、および地理的データ分散など

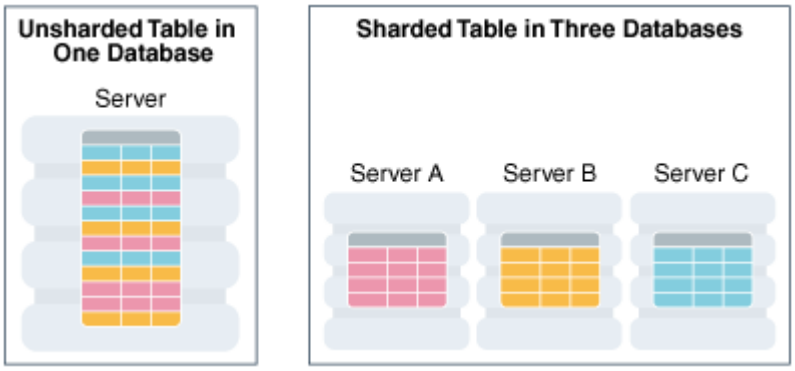
が挙げられます。シャーディングは、Oracle Cloudでのデプロイメントに適しています。シャーディングを実装するNoSQLデータ・ストアとは異なり、Oracle Shardingはエンタープライズ RDBMSの機能を犠牲にすることなく、シャーディングの利点を提供します。

シャーディング・アーキテクチャでは、各データベースは、独自のローカル・リソース(CPU、メモリー、フラッシュまたはディスク)を持つ専用サーバーでホストされます。このような構成の各データベースは**シャード**と呼ばれます。すべてのシャードが一体的に単一の論理データベースを形成し、これは**シャード・データベース**と呼ばれます。

水平パーティション化にはシャード間のデータベース表の分割が伴い、これによって各シャードには、列は同じですが行は異なるサブセットを持つ表が含まれます。この方法で分割された表は**シャード表**とも呼ばれています。

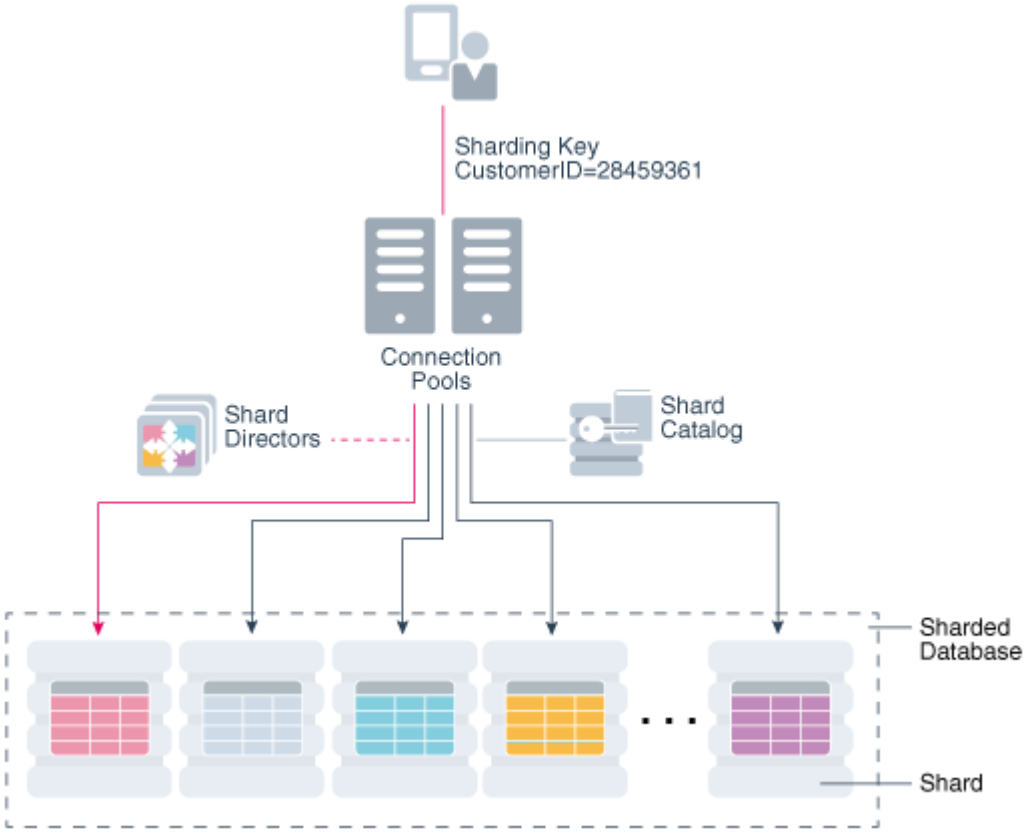
次の図は、3つのシャード間に水平にパーティション化されたシャード表を示しています。

図1-5 シャード間への表の水平パーティション化



ユースケースの1つは、複数の物理データベースへの顧客アカウント・データの分散です。たとえば、IDが28459361の顧客が自身のレコードを検索するとします。次の図に、考えられるアーキテクチャを示します。顧客のリクエストは、接続プールを介してルーティングされます。接続プールでは、シャード・ディレクタ(ネットワーク・リスナー)がリクエストをすべての顧客の行が含まれている適切なシャードに送信します。

図1-6 Oracle Shardingのアーキテクチャ



関連項目:

Oracle Shardingの概要については、Oracle Shardingの使用を参照してください。

データベース記憶域構造

データベースは、物理的な観点と論理的な観点の両方から検討できます。

物理データとは、オペレーティング・システム・レベルで表示可能なデータです。たとえば、Linuxのlsやpsなどのオペレーティング・システム・ユーティリティは、データベース・ファイルおよびプロセスをリスト表示できます。表などの論理データは、データベースに対してのみ意味を持ちます。Oracle Database内の表はSQL文でリスト表示できますが、オペレーティング・システム・ユーティリティでは表示できません。

データベースには物理構造と論理構造があります。物理構造と論理構造は分離されているため、論理記憶構造へのアクセスに影響を与えずに、データの物理記憶域を管理できます。たとえば、物理データベース・ファイルの名前を変更しても、このファイルにデータが格納されている表の名前は変更されません。

物理記憶域構造

物理データベース構造とは、データを格納するファイルです。

CREATE DATABASE文を実行すると、次のファイルが作成されます。

- データ・ファイル

  すべてのOracleデータベースには、すべてのデータベース・データを含む1つ以上の物理データファイルがあります。表や索引などの論理データベース構造のデータは、データファイルに物理的に格納されます。

- 制御ファイル

  すべてのOracleデータベースには、制御ファイルがあります。制御ファイルには、データベース名やデータベース・ファイルの名前と位置など、データベースの物理構造を指定したメタデータが含まれています。

- オンラインREDOログ・ファイル

  すべてのOracle Databaseには、2つ以上のオンラインREDOログ・ファイルのセットからなるオンラインREDOログがあります。オンラインREDOログは、データに加えられた変更をすべて記録するREDOエントリ(REDOログ・レコードとも呼ばれる)で構成されています。

Oracleデータベース・サーバーが機能するために重要なその他多くのファイルがあります。これらには、パラメータ・ファイルやネットワーキング・ファイルが含まれます。バックアップ・ファイルとアーカイブREDOログ・ファイルは、バックアップおよびリカバリに重要なオフライン・ファイルです。

関連項目:

「物理記憶域構造」

論理記憶域構造

論理記憶域構造によって、Oracle Databaseのディスク領域の使用をきめ細かく制御できます。

このトピックでは、論理記憶域構造について説明します。

- データ・ブロック

  最少レベルとして、Oracleデータベースのデータはデータ・ブロックに格納されます。1つのデータ・ブロックは、ディスク上の特定のバイト数に対応します。

- エクステント

  エクステントは、1回の割当てで取得される特定数の論理的に連続したデータ・ブロックで、特定タイプの情報の格納に使用されます。

- セグメント

  セグメントは、ユーザー・オブジェクト(表や索引)、UNDOデータ、または一時データに割り当てられるエクステントの集合です。

- 表領域

  データベースは、表領域と呼ばれる論理記憶域の単位に分割されます。表領域はセグメントの論理コンテナです。各表領域は1つ以上のデータ・ファイルで構成されます。

関連項目:

「論理記憶域構造」

データベース・インスタンス構造

Oracleデータベースは、メモリー構造とプロセスを使用してデータベースの管理やアクセスを行います。すべてのメモリー構造は、RDBMSを構成するコンピュータのメイン・メモリー内に存在します。

アプリケーションがOracleデータベースに接続すると、データベース・インスタンスに接続されます。インスタンスは、SGAのみでなくその他のメモリー領域も割り当て、バックグラウンド・プロセスの他のプロセスも起動して、アプリケーションに対応します。

Oracle Databaseプロセス

プロセスは一連のステップを実行できるオペレーティング・システムのメカニズムです。オペレーティング・システムによっては、「ジョブ」、「タスク」または「スレッド」という用語を使用します。

このトピックの目的では、スレッドはプロセスと同等です。Oracleデータベース・インスタンスには次のタイプのプロセスがあります。

- クライアント・プロセス

  これらのプロセスは、アプリケーション・プログラムまたはOracleツールのソフトウェア・コードを実行するために作成およびメンテナンスされます。ほとんどの環境ではクライアント・プロセス用に独立したコンピュータが使用されます。

- バックグラウンド・プロセス

  これらのプロセスは、各クライアント・プロセスごとに実行されている複数のOracle Databaseプログラムが処理する機能を1つにまとめます。バックグラウンド・プロセスは、非同期的にI/Oを実行して他のOracle Databaseプロセスを監視することによって、並列性を高め、パフォーマンスと信頼性を向上させます。

- サーバー・プロセス

  これらのプロセスは、クライアント・プロセスと通信し、Oracle Databaseと対話してリクエストに対応します。

Oracleプロセスには、サーバー・プロセスとバックグラウンド・プロセスがあります。ほとんどの環境では、Oracleプロセスとクライアン

ト・プロセスは別々のコンピュータ上で実行されます。

関連項目:

「プロセス・アーキテクチャ」

インスタンスのメモリー構造

Oracle Databaseでは、プログラム・コード、ユーザー間で共有されるデータ、および接続している各ユーザーのプライベート・データ領域のために、メモリー構造が作成および使用されます。

次のメモリー構造がデータベース・インスタンスに関連付けられています。

- システム・グローバル領域(SGA)

  SGAは、共有メモリー構造のグループであり、1つのデータベース・インスタンスに関するデータと制御情報が含まれています。SGAコンポーネントの例として、データベース・バッファ・キャッシュや共有SQL領域があります。Oracle Database 12cリリース1 (12.1.0.2)以降、SGAには、データを列形式でメモリーに移入できる、オプションのインメモリー列ストア(IM列ストア)を含めることができます。

- プログラム・グローバル領域(PGA)

  PGAは、サーバー・プロセスまたはバックグラウンド・プロセス用のデータや制御情報を含むメモリー領域です。PGAにアクセスできるのはプロセスのみです。それぞれのサーバー・プロセスおよびバックグラウンド・プロセスには専用のPGAがあります。

  関連項目:

  「メモリー・アーキテクチャ」

アプリケーションおよびネットワークのアーキテクチャ

特定のコンピュータ・システムまたはネットワークを最大限に活用できるように、Oracle Databaseではデータベース・サーバーとクライアント・プログラムの間で処理を分割できます。RDBMSを実行しているコンピュータはデータベース・サーバーの処理を実行し、アプリケーションを実行しているコンピュータはデータの解釈と表示を行います。

アプリケーション・アーキテクチャ

アプリケーション・アーキテクチャは、データベース・アプリケーションがOracleデータベースに接続されているコンピューティング環境です。データベースの最も一般的なアーキテクチャは、クライアント/サーバーと複数層の2種類です。

クライアント・サーバー・アーキテクチャ

クライアント/サーバー・アーキテクチャでは、クライアント・アプリケーションは、データベース・サーバー上で実行される操作に関するリクエストを開始します。サーバーは、Oracle Databaseソフトウェアを実行し、同時実行の共有データ・アクセスに必要な機能を処理します。サーバーは、クライアントから発行されたリクエストを受け取って処理します。

複数層アーキテクチャ

複数層アーキテクチャでは、1つ以上のアプリケーション・サーバーが操作の各部分を実行します。アプリケーション・サーバーには、アプリケーション・ロジックの大部分が含まれ、クライアントのデータ・アクセスを可能にし、いくつかの問合せ処理を実行します。こ

の方法で、データベースの負荷が軽減します。アプリケーション・サーバーは、クライアントと複数のデータベース間のインタフェースとして機能し、セキュリティ・レベルを高める役割を果します。

サービス指向アーキテクチャ(SOA)は、アプリケーション機能がサービスにカプセル化されている複数層アーキテクチャです。通常、SOAサービスはWebサービスとして実装されます。WebサービスはHTTP経由でアクセス可能なサービスであり、Webサービス記述言語(WSDL)やSOAPなどのXMLベースの規格に基づいています。Oracle Databaseは、従来の複数層またはSOA環境においてWebサービス・プロバイダとして機能できます。

Simple Oracle Document Access (SODA)は、データベースに格納されたデータへのアクセスを可能にするSOAのアダプタです。SODAは、リレーショナル・データベース機能やSQLおよびPL/SQLなどの言語に関する知識を使用しないスキーマレス・アプリケーション開発向けに設計されています。Oracle Databaseでは、ドキュメントがどのように格納されているかを把握しなくても、ドキュメント・コレクションの作成および格納や、それらのドキュメントの取得および問合せの実行を可能にします。RESTのためのSODAは、Representational State Transfer (REST)アーキテクチャ・スタイルを使用してSODAを実装します。

関連項目:

- 「複数層アーキテクチャの概要」
- データベースを使用したWebサービスの使用方法の詳細は、Oracle XML DB開発者ガイドを参照してください

Oracle Net Servicesのアーキテクチャ

Oracle Net Servicesは、データベースとネットワークの通信プロトコルとの間のインタフェースであり、分散処理と分散データベースの実現を支援します。

通信プロトコルにより、ネットワーク上でのデータの送受信方法が定義されます。Oracle Net Servicesは、TCP/IP、HTTP、FTPおよびWebDAVなど、主要なネットワーク・プロトコルによる通信をすべてサポートしています。

Oracle NetはOracle Net Servicesのコンポーネントであり、クライアント・アプリケーションからデータベース・サーバーへのネットワーク・セッションを確立して維持します。ネットワーク・セッションの確立後は、Oracle Netはクライアント・アプリケーションとデータベース・サーバーのためのデータ伝達手段として機能し、これらの間でメッセージを交換します。Oracle Netは、ネットワーク上の各コンピュータにあるため、これらのジョブを実行できます。

Net Servicesの重要なコンポーネントにOracle Net Listener (リスナーと呼ばれる)があり、プロセスとしてデータベースまたはネットワークの別の場所で実行されます。クライアント・アプリケーションは、リスナーに接続要求を送信し、リスナーはデータベースに対するこれらの要求のトラフィックを管理します。接続が確立されると、クライアントとデータベースは直接通信します。

クライアント要求を処理するようにOracle Databaseを構成する最も一般的な方法は、次のとおりです。

- 専用サーバー・アーキテクチャ

  各クライアント・プロセスは、専用サーバー・プロセスに接続します。サーバー・プロセスは、クライアントのセッションの継続中、その他のどのクライアントとも共有されません。それぞれの新規セッションに専用サーバー・プロセスが割り当てられます。

- 共有サーバー・アーキテクチャ

  データベースは複数のセッションに対して共有サーバー・プロセスのプールを使用します。クライアント・プロセスはディスパッチャと通信し、ディスパッチャは、クライアントごとに専用のサーバー・プロセスがなくても多数のクライアントが同じデータベース・インスタンスに接続できるようにするプロセスです。

関連項目:

- 「Oracle Net Servicesのアーキテクチャの概要」
- Oracle Netアーキテクチャについてさらに学習するには、『Oracle Database Net Services管理者ガイド』を参照してください。
- データベースでのWebDAVの使用方法の詳細は、Oracle XML DB開発者ガイドを参照してください

Oracle Databaseドキュメントのロードマップ

ユーザーが必要とする情報をできるだけ効率よく見つけられるように、マニュアル・セットは特定のアクセス・パスを使用して設計されています。

マニュアル・セットは、初級、中級、および上級の3つの層またはグループに分かれています。ユーザーは、初級グループのマニュアルから始め、中級グループのマニュアル(2日でシリーズ)に進み、最後に上級マニュアル(残りのマニュアルを含む)に進みます。

サポートされているOracle Databaseのリリースのドキュメントは、
https://docs.oracle.com/en/database/oracle/oracle-database/を参照してください。

Oracle Databaseドキュメント: 初級グループ

技術者でOracle Databaseの新規ユーザーである場合は、初級グループのマニュアルを1冊以上、最初から最後まで読むことから始めてください。このグループの各マニュアルは、2日で読めるように設計されています。

このマニュアルに加えて、初級グループには次の表に示すマニュアルが含まれます。

表1-1 初級グループ

手動	説明
Oracle Database 2 日でデータベース管理者	Oracle Database 管理者(DBA)には、Oracle Database の運用全般に対する責任があります。このタスクベースのクイック・スタートでは、DBA に Oracle Enterprise Manager Database Express (EM Express)を使用した日常のデータベース管理タスクの実行方法を指導します。このマニュアルでは、基本的なトラブルシューティングやパフォーマンス監視アクティビティの実行方法など、データベースの稼働を維持するために必要な、一般的なすべての管理タスクを実行する方法について説明しています。
『Oracle Database 2 日で開発者ガイド』	このタスクベースのクイック・スタートでは、SQL および PL/SQL で Oracle Database の基本機能を使用する方法について説明しています。

初級グループのマニュアルは密接に関連しており、そのために相互参照が多くなっています。たとえば、『Oracle Database概要』は、概念に基づいてタスクを実行する方法を学ぶことができるように、2日でシリーズのマニュアルを頻繁に参照しています。2日でシリーズのマニュアルでは、タスクの概念的な説明について、『*Oracle Database*概要』を頻繁に参照しています。

Oracle Databaseドキュメント: 中級グループ

初級グループの次のステップは、中級グループです。

中級グループのマニュアルは、初級グループの「2日で」シリーズのマニュアルに含まれる情報を前提とし、それと同じ「2日で」という言葉が先頭に付いています。「2日で」マニュアルでは、初級マニュアルよりも詳細にトピックを説明し、また、特に重要なトピックについて説明しています。マニュアルは、様々な読者を対象としています。

- データベース管理者

  Oracle Database 2日でパフォーマンス・チューニング・ガイドは、Oracle Diagnostics Pack、Oracle Tuning PackおよびOracle Enterprise Manager Cloud Control (Cloud Control)で用意されている機能を使用して、日常のデータベース・パフォーマンス・チューニング・タスクの実行方法を学習するクイック・スタート・ガイドです。

- データベース開発者

  Oracle Database 2日でJava開発者ガイドは、Javaアプリケーションの構築に使用されるすべてのJava製品を理解するのに役立ちます。このマニュアルでは、サンプルWebアプリケーションのOracle JDBC Thinドライバ、ユニバーサル接続プール(UCP)およびデータベースでのJava (OJVM)を使用する方法について説明します。

Oracle Databaseドキュメント: 上級グループ

上級グループのマニュアルは、特定のトピックについて、2日でシリーズのマニュアルよりも詳細な情報を必要とする専門家ユーザーを対象としています。

次の表に、上級グループの必須のリファレンス・マニュアルを示します。

表1-2 必須のリファレンス・マニュアル

手動	説明
Oracle Database SQL 言語リファレンス	Oracle Database の情報の管理に使用する構造化問合せ言語(SQL)の詳細な説明を提供します。
『Oracle Database リファレンス』	データベース初期化パラメータ、データ・ディクショナリ・ビュー、動的パフォーマンス・ビュー、待機イベントおよびバックグラウンド・プロセスについて説明します。
Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス	Oracle Database サーバーに付属する PL/SQL パッケージについて説明します。提供されたパッケージは、アプリケーションの作成や、独自のストアド・プロシージャを作成するアイデアのために使用できます。

上級ガイドは数が多いため、この項にすべてを示すことはできません。次の表に、熟練したOracle DBAの多くが使用するガイドを示します。

表1-3 DBA向け上級グループ

手動	説明
Oracle Database 管理者ガイド	データベースの作成と構成、データベースのメンテナンスと監視、スキーマ・オブジェクトの作成、ジョブのスケジューリング、問

手動	説明
	題の診断などのタスクを実行する方法について説明します。
『Oracle Database セキュリティ・ガイド』	デフォルトのデータベース機能を使用して Oracle Database のセキュリティを構成する方法について説明します。
Oracle Database パフォーマンス・チューニング・ガイド	このマニュアルでは、Oracle Database のツールを使用して、データベース・パフォーマンスを最適化する方法について説明します。また、データベースの作成でのパフォーマンスのベスト・プラクティスを説明するとともに、パフォーマンス関連の参照情報を示します。
Oracle Database SQL チューニング・ガイド	SQL 処理、オプティマイザ、実行計画、SQL 演算子、オプティマイザ統計、アプリケーションのトレースおよび SQL アドバイザについて説明します。
『Oracle Database バックアップおよびリカバリ・アドバンスト・ユーザーズ・ガイド』	Oracle データベースのバックアップ方法、リストア方法およびリカバリ方法、データベース・ファイルのバックアップに対するメンテナンスの実行方法、およびストレージ・システム間のデータの転送方法について説明します。
Oracle Real Application Clusters 管理およびデプロイメント・ガイド	Oracle RAC データベースのインストール方法、構成方法、管理方法およびトラブルシューティング方法について説明します。

次の表に、熟練したOracle開発者の多くが使用するガイドを示します。

表1-4 開発者向け上級グループ

手動	説明
Oracle Database 開発ガイド	Oracle Database 環境で実行するためのアプリケーションの開発方法または既存のアプリケーションの変換方法について説明します。このマニュアルでは、アプリケーション設計の基礎と、SQL および PL/SQL で開発するための必須の概念について説明しています。
Oracle Database PL/SQL 言語リファレンス	データ型、制御文、コレクション、トリガー、パッケージ、エラー処理を含む、PL/SQL 言語のあらゆる側面について説明します。
Oracle Database Java 開発者ガイド	Oracle Database での Java アプリケーションの開発、ロードおよび実行方法について説明します。

手動	説明
Oracle Database SecureFiles and Large Objects 開発者ガイド	ラージ・オブジェクト(LOB)、SecureFiles LOB およびデータベース・ファイル・システム(DBFS)を使用した新しいアプリケーションの開発方法について説明します。

特定のユーザーが必要とする他の上級ガイドは、そのユーザーの担当業務の分野によって異なります。

第I部 Oracleリレーショナル・データ構造

この部では、データ整合性の規則を含むデータベースの基本的なデータ構造、およびメタデータを格納する構造について説明します。

この部は、次の章で構成されています。

- 表と表クラスタ
- 索引と索引構成表
- パーティション、ビューおよびその他のスキーマ・オブジェクト
- データ整合性
- データ・ディクショナリと動的パフォーマンス・ビュー

2 表と表クラスタ

この章では、スキーマ・オブジェクトの概要、および最も一般的なスキーマ・オブジェクトである表について説明します。

この章の構成は、次のとおりです。

- スキーマ・オブジェクトの概要
- 表の概要
- 表クラスタの概要
- 属性クラスタ表の概要
- 一時表の概要
- 外部表の概要
- オブジェクト表の概要

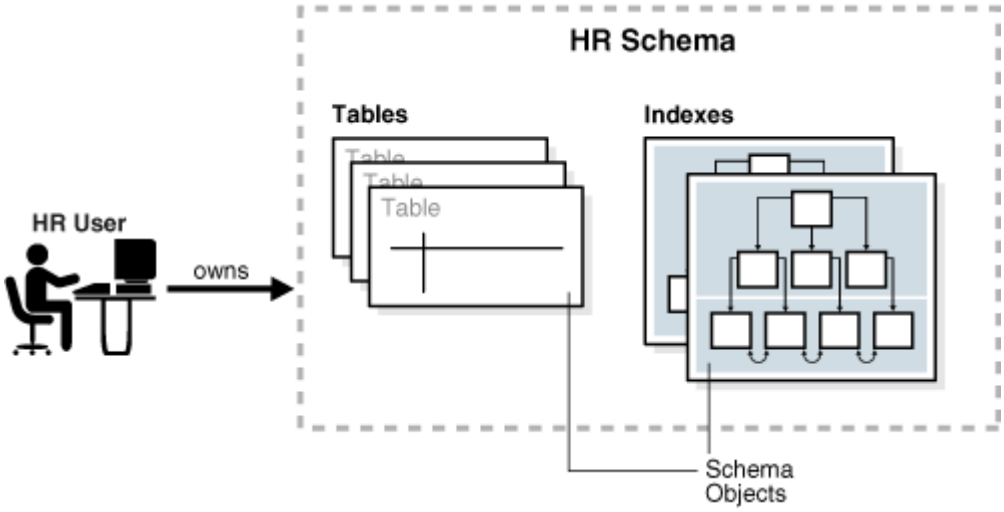
スキーマ・オブジェクトの概要

データベース・**スキーマ**は、スキーマ・オブジェクトと呼ばれるデータ構造の論理コンテナです。スキーマ・オブジェクトの例には、表や索引があります。SQLを使用して、スキーマ・オブジェクトを作成および操作できます。

データベース・ユーザー・アカウントには、パスワードおよび特定のデータベース権限があります。各ユーザー・アカウントは、ユーザーと同じ名前が付いた単一のスキーマを所有します。スキーマには、スキーマを所有するユーザーのデータが含まれています。たとえば、hrユーザー・アカウントはhrスキーマを所有し、このスキーマにemployees表などのスキーマ・オブジェクトが含まれます。通常、本番データベースでは、スキーマ所有者は人ではなく、データベース・アプリケーションを表します。

スキーマ内の特定のタイプの各スキーマ・オブジェクトには、一意の名前が付きます。たとえば、hr.employeesはhrスキーマ内のemployees表を指します。図2-1に、hrという名前のスキーマ所有者とhrスキーマ内のスキーマ・オブジェクトを示します。

図2-1 hrスキーマ



この項では、次の項目について説明します。

- スキーマ・オブジェクトのタイプ
- スキーマ・オブジェクトの格納
- スキーマ・オブジェクトの依存性
- SYSスキーマとSYSTEMスキーマ

- サンプル・スキーマ

関連項目:

ユーザーと権限についてさらに学習するには、「データベース・セキュリティの概要」を参照してください

スキーマ・オブジェクトのタイプ

Oracle SQLでは、他にも多数のタイプのスキーマ・オブジェクトを作成および操作できます。

次の表に、スキーマ・オブジェクトの主要なタイプを示します。

表2-1 スキーマ・オブジェクト

オブジェクト	説明	さらに学習するには
表	表では、データは行に格納されます。表は、リレーショナル・データベースで最も重要なスキーマ・オブジェクトです。	表の概要
索引	索引とは、表または表クラスタの索引付けされた各行のエントリを含むスキーマ・オブジェクトのことで、索引により行に直接かつ高速にアクセスできます。Oracle Database は、いくつかのタイプの索引をサポートしています。索引構成表は、データが索引の構造で格納される表です。	「索引と索引構成表」
パーティション	パーティションとは、大規模な表や索引の断片のことです。各パーティションには独自の名前があり、独自の記憶特性を持つ場合があります。	「パーティションの概要」
ビュー	ビューとは、1 つ以上の表または他のビューに含まれているデータの表現がカスタマイズされたものです。ビューは、ストアド・クエリーとみなすことができます。実際にはビューにデータは含まれていません。	ビューの概要
順序	順序とは、複数のユーザーが整数を生成して共有できるユーザー作成オブジェクトのことです。通常、順序を使用して主キーの値を生成します。	「順序の概要」

オブジェクト	説明	さらに学習するには
ディメンション	ディメンションは、列セットのペア間の親子関係を定義するもので、この列セットに含まれるすべての列は、同じ表の列である必要があります。ディメンションは、主に顧客、製品および時刻などのデータを分類するために使用されます。	「ディメンションの概要」
シノニム	シノニムとは、他のスキーマ・オブジェクトの別名のことです。シノニムは単なる別名であるため、データ・ディクショナリ内の定義以外に記憶域は必要ありません。	シノニムの概要
PL/SQL サブプログラムと PL/SQL パッケージ	PL/SQL は、SQL に対する Oracle の手続き型の拡張機能です。PL/SQL サブプログラムとは、一連のパラメータを使用して起動できる、名前付き PL/SQL ブロックのことです。PL/SQL パッケージとは、論理的に関連する PL/SQL タイプ、変数およびサブプログラムをグループ化するものです。	PL/SQL サブプログラム

データベースには他のタイプのオブジェクトも保存されており、SQL文で作成および操作できますが、スキーマには含まれていません。これらのオブジェクトには、データベース・ユーザー・アカウント、ロール、コンテキストおよびディクショナリ・オブジェクトが含まれています。

関連項目:

- スキーマ・オブジェクトの管理方法を学習するには、『Oracle Database管理者ガイド』を参照してください。
- スキーマ・オブジェクトとデータベース・オブジェクトの詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

スキーマ・オブジェクトの格納

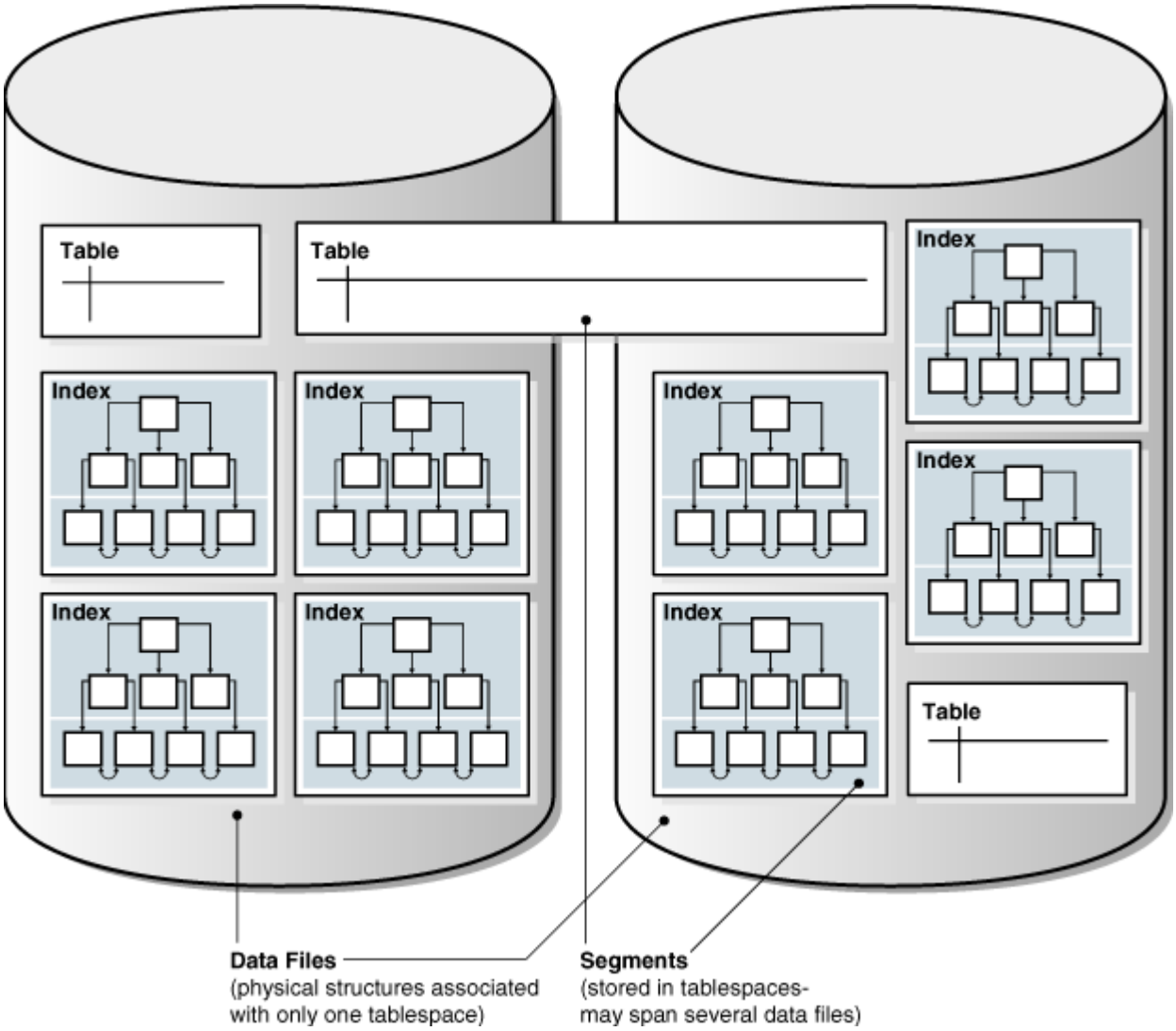
一部のスキーマ・オブジェクトでは、**セグメント**と呼ばれる論理記憶域構造のタイプでデータを格納します。たとえば、パーティション化されていないヒープ構成表や索引では、セグメントを作成します。

ビューや順序などの他のスキーマ・オブジェクトは、メタデータのみで構成されます。このトピックでは、セグメントを持つスキーマ・オブジェクトについてのみ説明します。

Oracle Databaseでは、スキーマ・オブジェクトは表領域内に論理的に格納されます。スキーマと表領域には関連はなく、表領域は異なるスキーマのオブジェクトを含むことができ、スキーマの各オブジェクトは異なる表領域に含めることができます。各オブジェクトのデータは、物理的には、1つ以上のデータファイルに格納されます。

次の図に、表と索引のセグメント、表領域およびデータファイルの可能な構成を示します。1つの表のデータ・セグメントが2つのデータファイル(両方とも同じ表領域の一部)にまたがっています。セグメントが複数の表領域にまたがることはできません。

図2-2 セグメント、表領域およびデータファイル



関連項目:

- 表領域とセグメントについて学習するには、「論理記憶域構造」を参照してください
- スキーマ・オブジェクトの記憶域を管理する方法を学習するには、『Oracle Database管理者ガイド』を参照してください

スキーマ・オブジェクトの依存性

スキーマ・オブジェクトには、他のオブジェクトを参照し、**スキーマ・オブジェクトの依存性**を作成するものがあります。

たとえば、ビューに表またはビューを参照する問合せが含まれている場合や、PL/SQLサブプログラムが他のサブプログラムを起動する場合があります。オブジェクトAの定義がオブジェクトBを参照している場合、AはBの依存オブジェクト、BはAの参照オブジェクトです。

Oracle Databaseには、依存オブジェクトが参照オブジェクトに関して常に最新であることを確認する自動メカニズムが用意されています。依存オブジェクトを作成すると、データベースは依存オブジェクトとその参照オブジェクトとの間の依存性を追跡します。参照オブジェクトに変更があり、依存オブジェクトがその影響を受ける可能性がある場合は、データベースにより依存オブジェクト

に無効のマークが付けられます。たとえば、ユーザーが表を削除した場合、その表に基づくビューは使用できなくなります。

無効な依存オブジェクトを使用できるようにするには、その依存オブジェクトが参照オブジェクトの新しい定義で再コンパイルされる必要があります。再コンパイルは、無効な依存オブジェクトが参照されると自動的に実行されます。

スキーマ・オブジェクトでどのように依存性を作成できるかを示す例として、次のサンプル・スクリプトでは、表test_tableを作成し、この表を問い合せるプロシージャを作成します。

```
CREATE TABLE test_table ( col1 INTEGER, col2 INTEGER );

CREATE OR REPLACE PROCEDURE test_proc
AS
BEGIN
 FOR x IN ( SELECT col1, col2 FROM test_table )
 LOOP
   -- process data
   NULL;
 END LOOP;
END;
/
```

次のようにプロシージャtest_procのステータスを問い合せると、プロシージャは有効であることが示されます。

```
SQL> SELECT OBJECT_NAME, STATUS FROM USER_OBJECTS WHERE OBJECT_NAME = 'TEST_PROC';

OBJECT_NAME STATUS
----------- -------
TEST_PROC   VALID
```

test_tableにcol3列を追加しても、プロシージャにはこの列に対する依存性がないため、その後もプロシージャは有効です。

```
SQL> ALTER TABLE test_table ADD col3 NUMBER;

Table altered.

SQL> SELECT OBJECT_NAME, STATUS FROM USER_OBJECTS WHERE OBJECT_NAME = 'TEST_PROC';

OBJECT_NAME STATUS
----------- -------
TEST_PROC   VALID
```

ただし、test_procプロシージャが依存しているcol1列のデータ型を変更すると、プロシージャは無効化されます。

```
SQL> ALTER TABLE test_table MODIFY col1 VARCHAR2(20);

Table altered.

SQL> SELECT OBJECT_NAME, STATUS FROM USER_OBJECTS WHERE OBJECT_NAME = 'TEST_PROC';

OBJECT_NAME STATUS
----------- -------
TEST_PROC   INVALID
```

次の例に示すように、プロシージャを実行または再コンパイルすると、プロシージャは再び有効になります。

```
SQL> EXECUTE test_proc

PL/SQL procedure successfully completed.

SQL> SELECT OBJECT_NAME, STATUS FROM USER_OBJECTS WHERE OBJECT_NAME = 'TEST_PROC';
```

```
OBJECT_NAME STATUS
----------- -------
TEST_PROC   VALID
```

関連項目:

スキーマ・オブジェクトの依存性を管理する方法を学習するには、『Oracle Database管理者ガイド』および『Oracle Database開発ガイド』を参照してください。

SYSスキーマとSYSTEMスキーマ

すべてのOracleデータベースには、デフォルトの管理アカウントが自動的に含まれています。

管理アカウントは、高度な権限が付与され、データベースの起動と停止、メモリーと記憶域の管理、データベース・ユーザーの作成と管理などのタスクの実行を認可されたDBA専用のアカウントです。

SYS管理アカウントは、データベースが作成されたときに自動的に作成されます。このアカウントでは、すべてのデータベース管理機能を実行できます。SYSスキーマには、データ・ディクショナリの実表とビューが格納されています。これらの実表とビューは、Oracle Databaseの操作にとって重要です。SYSスキーマ内の表はデータベースによってのみ操作され、ユーザーからは決して変更されません。

SYSTEM管理アカウントも、データベースが作成されたときに自動的に作成されます。SYSTEMスキーマには、管理情報を表示する追加の表とビュー、およびOracle Databaseの様々なオプションとツールで使用される内部の表とビューが格納されます。SYSTEMスキーマは、管理ユーザー以外のユーザーが必要とする表の格納には決して使用しないでください。

関連項目:

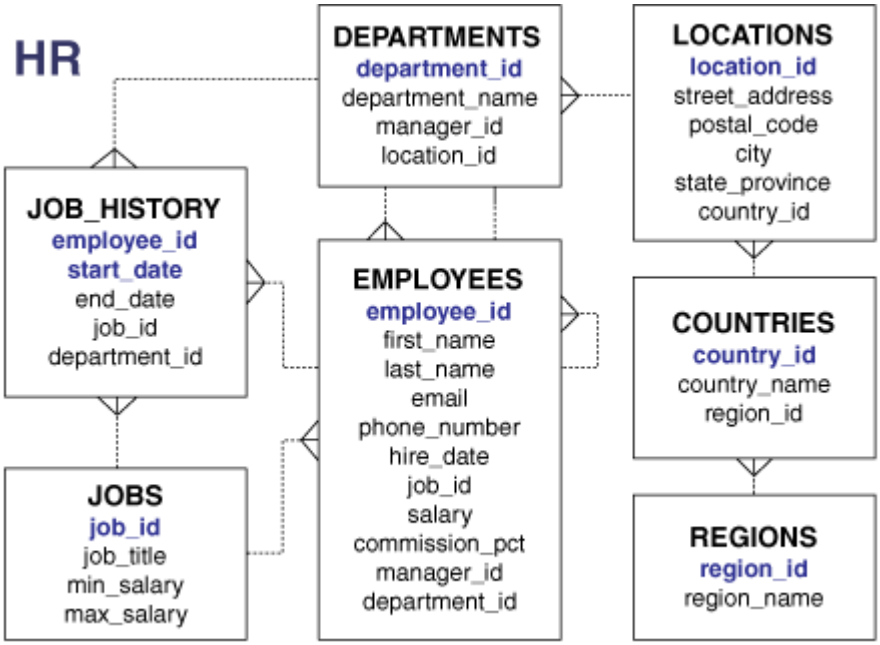
- ユーザー・アカウント
- 「管理者権限での接続」
- SYS、SYSTEMおよびその他の管理アカウントについて学習するには、『Oracle Database管理者ガイド』を参照してください。

サンプル・スキーマ

Oracleデータベースには、**サンプル・スキーマ**が含まれることがあり、サンプル・スキーマとは、Oracleドキュメントや教材で共通のデータベース・タスクを説明できるように相互にリンクされた一連のスキーマのことです。

hrサンプル・スキーマには、従業員、部門、勤務地、職歴などの情報が含まれています。次の図に、hr内の表のエンティティ関連ダイアグラムを示します。このマニュアルのほとんどの例では、このスキーマのオブジェクトを使用します。

図2-3 hrスキーマ



関連項目:

サンプル・スキーマのインストール方法を学習するには、『Oracle Databaseサンプル・スキーマ』を参照

表の概要

表は、Oracle Databaseにおけるデータ編成の基本単位です。

表には、記録が必要な情報に関してなんらかの意味を持つ**エンティティ**が記述されます。たとえば、従業員がエンティティになることがあります。

Oracle Databaseの表は、次の基本カテゴリに分類されます。

- リレーショナル表

  リレーショナル表は、単純な列で構成された最も一般的な表タイプです。例2-1に、リレーショナル表のCREATE TABLE文を示します。

- オブジェクト表

  列は、オブジェクト型のトップレベルの属性に相当します。「オブジェクト表の概要」を参照してください。

次の構成特性を持つリレーショナル表を作成できます。

- ヒープ構成表では、行は特定の順序で格納されません。デフォルトでは、CREATE TABLE文でヒープ構成表が作成されます。
- 索引構成表では、主キー値に従って行が整列されます。アプリケーションによっては、索引構成表によりパフォーマンスが強化され、ディスク領域をより効率的に使用できるものがあります。「索引構成表の概要」を参照してください。
- 外部表は読取り専用の表であり、これらのメタデータはデータベースに格納されますが、データはデータベースの外部に格納されます。「外部表の概要」を参照してください。

表は永続または一時のいずれかです。永続表の定義とデータは、セッションをまたいで存続します。一時表の定義も永続表の定義と同様に存続しますが、データはトランザクションまたはセッションの期間中にのみ存在します。一時表は、複数の操作の実行によって結果を作成する場合があるために、結果セットを一時的に保持する必要があるアプリケーションで役立ちます。

この項の内容は次のとおりです。

- 列
- 行
- 例: CREATE TABLE文とALTER TABLE文
- Oracleのデータ型
- 整合性制約
- 表の記憶域
- 表の圧縮

関連項目:

表の管理方法を学習するには、『Oracle Database管理者ガイド』を参照してください。

列

表の定義には、表名と列の集合が含まれています。

列は、表で示されるエンティティの属性を識別します。たとえば、employees表の列employee_idは、従業員エンティティの従業員ID属性を参照します。

通常は、表を作成するとき、各列に列名、データ型および幅を指定します。たとえば、employee_idのデータ型がNUMBER(6)である場合、この列に6桁以内の幅の数値データのみが格納されることを示します。幅は、DATEデータ型の場合のようにデータ型によって事前に決定されている場合があります。

仮想列

表には、通常の列とは異なり、ディスク領域を使用しない**仮想列**を含めることができます。

データベースによりユーザーの指定する一連の式または関数が計算され、必要に応じて仮想列から値が導出されます。たとえば、仮想列incomeが、salary列とcommission_pct列の関数になることがあります。

関連項目:

仮想列の管理方法を学習するには、『Oracle Database管理者ガイド』を参照してください。

非表示列

非表示列とは、列が明示的に名前で指定される場合のみ値が表示されるユーザー指定の列です。既存のアプリケーションに影響を与えずに、非表示列を表に追加し、必要に応じてその列を表示することができます。

一般に、非表示列により、オンライン・アプリケーションの移行や開発がしやすくなります。ユースケースが、3列の表をSELECT *文で問い合せるアプリケーションだとします。この表に4番目の列を追加すると、3列のデータを必要とするアプリケーションは破損します。4番目の非表示列を追加すると、アプリケーションは正常に機能します。そこで開発者は、4番目の列を処理できるようにアプリケーションを変更して、アプリケーションが起動すると、その列が表示されるようにすることができます。

次の例では、表productsを非表示列countを付けて作成し、その非表示列を表示します。

```
CREATE TABLE products ( prod_id INT, count INT INVISIBLE );
ALTER TABLE products MODIFY ( count VISIBLE );
```

関連項目:

- 非表示列の管理方法を学習するには、『Oracle Database管理者ガイド』を参照してください。
- 非表示列の詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

行

行は、表内のレコードに対応する列情報の集まりです。

たとえば、employees表の行では、特定の従業員の属性(従業員ID、姓、名など)が記述されます。表の作成後に、SQL文を使用して行の挿入、問合せ、削除および更新ができます。

例: CREATE TABLE文とALTER TABLE文

表を作成するためのOracle SQL文は、CREATE TABLEです。

例2-1 CREATE TABLE employees

次の例に、hrサンプル・スキーマに含まれるemployees表のCREATE TABLE文を示します。この文では、employee_idおよびfirst_nameなどの列を指定し、各列にNUMBERまたはDATEなどのデータ型を指定します。

```
CREATE TABLE employees
    ( employee_id    NUMBER(6)
    , first_name     VARCHAR2(20)
    , last_name      VARCHAR2(25)
        CONSTRAINT     emp_last_name_nn  NOT NULL
    , email          VARCHAR2(25)
      CONSTRAINT     emp_email_nn  NOT NULL
    , phone_number   VARCHAR2(20)
    , hire_date      DATE
      CONSTRAINT     emp_hire_date_nn  NOT NULL
    , job_id         VARCHAR2(10)
      CONSTRAINT     emp_job_nn  NOT NULL
    , salary         NUMBER(8,2)
    , commission_pct NUMBER(2,2)
    , manager_id     NUMBER(6)
    , department_id  NUMBER(4)
    , CONSTRAINT     emp_salary_min
                     CHECK (salary > 0)
    , CONSTRAINT     emp_email_uk
                     UNIQUE (email)
    ) ;
```

例2-2 ALTER TABLE employees

次の例に、employees表に整合性制約を追加するALTER TABLE文を示します。整合性制約はビジネス・ルールを適用し、無効な情報が表に入力されるのを防止します。

```
ALTER TABLE employees
ADD ( CONSTRAINT     emp_emp_id_pk
```

```
                      PRIMARY KEY (employee_id)
    , CONSTRAINT     emp_dept_fk
                      FOREIGN KEY (department_id)
                        REFERENCES departments
    , CONSTRAINT     emp_job_fk
                      FOREIGN KEY (job_id)
                        REFERENCES jobs (job_id)
    , CONSTRAINT     emp_manager_fk
                      FOREIGN KEY (manager_id)
                        REFERENCES employees
    ) ;
```

例2-3 employees表の行

次のサンプル出力は、8行および6列のhr.employees表を示しています。

```
EMPLOYEE_ID FIRST_NAME  LAST_NAME      SALARY COMMISSION_PCT DEPARTMENT_ID
----------- ----------- ------------- ------- -------------- -------------
        100 Steven      King            24000                          90
        101 Neena       Kochhar         17000                          90
        102 Lex         De Haan         17000                          90
        103 Alexander   Hunold           9000                          60
        107 Diana       Lorentz          4200                          60
        149 Eleni       Zlotkey         10500             .2           80
        174 Ellen       Abel            11000             .3           80
        178 Kimberely   Grant            7000            .15
```

前述の出力は、表、列および行に関する、次の重要な特徴のいくつかを示しています。

- 表の行には、1人の従業員の、名前、給与、部門などの属性が記述されます。たとえば、出力の最初の行には、Steven Kingという名前の従業員のレコードが表示されています。
- 列には、従業員の属性が記述されます。この例では、employee_id列が主キーであり、すべての従業員が一意の従業員IDで識別されることを意味します。2人の従業員が同じ従業員IDを持たないことが保証されます。
- 非キー列には、同じ値の行が含まれることがあります。この例では、従業員101と従業員102の給与の値が同じ17000です。
- 外部キー列は、同じ表または別の表の主キーや一意キーを参照します。この例では、department_idの90の値は、departments表のdepartment_id列に対応しています。
- フィールドは、行と列の交差部分です。フィールドには1つの値のみが含まれます。たとえば、従業員103の部門IDのフィールドには、値60が含まれています。
- フィールドには、値がないこともあります。この場合、フィールドにはNULL値が含まれることになります。従業員100のcommission_pct列の値はNULLですが、従業員149のこのフィールドの値は.2です。列にNULL値を入力できるのは、その列にNOT NULLまたは主キー整合性制約が定義されていない場合のみです。この場合、その列に値を持たない行は挿入できません。

関連項目:

CREATE TABLEの構文およびセマンティクスについては、『Oracle Database SQL言語リファレンス』を参照

Oracleデータ型

列は、それぞれ特定の記憶域形式、制約および値の有効範囲に関連付けられた**データ型**を持っています。値のデータ型により、固定された一連のプロパティが値に関連付けられます。

これらのプロパティに従い、Oracle Databaseではあるデータ型の値を別のデータ型の値と異なる方法で扱います。たとえば、NUMBERデータ型の値は乗算できますが、RAWデータ型の値は乗算できません。

表の作成時には、その各列のデータ型を指定する必要があります。その後、列に挿入した各値はその列をデータ型と想定します。

Oracle Databaseには、いくつかの組込みデータ型が用意されています。最も一般的に使用されるデータ型は、次のカテゴリに分類されます。

- 文字データ型
- 数値データ型
- 日時データ型
- ROWIDデータ型
- 書式モデルとデータ型

組込みデータ型のその他の重要なカテゴリには、RAW、ラージ・オブジェクト(LOB)およびコレクションがあります。PL/SQLには、定数と変数のデータ型として、BOOLEAN、参照型、コンポジット型(レコード)およびユーザー定義型があります。

関連項目:

- LOBの概要
- 組込みSQLデータ型について学習するには、『Oracle Database SQL言語リファレンス』を参照してください
- PL/SQLデータ型について学習するには、『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』を参照してください
- 組込みデータ型の使用方法を学習するには、『Oracle Database開発ガイド』を参照してください

文字データ型

文字データ型は、英数字データを文字列で格納します。最も一般的な文字データ型はVARCHAR2で、文字データを最も効率よく格納できます。

バイト値は、文字コード体系(通常、文字セットと呼ばれます)に対応します。データベースの文字セットは、データベースの作成時に設定されます。文字セットの例には、7ビットASCII、EBCDICおよびUnicode UTF-8があります。

文字データ型の長さセマンティクスは、バイト数または文字数で測定できます。文字列を一連のバイトとして取り扱うことは、バイト・セマンティクスと呼ばれます。これは、文字データ型のデフォルトです。文字列を一連の文字として取り扱うことは、キャラクタ・セマンティクスと呼ばれます。文字は、データベース文字セットのコード・ポイントです。

関連項目:

- 文字セット

- データ型の概要は、『Oracle Database 2日で開発者ガイド』を参照
- 文字データ型の選択方法を学習するには、『Oracle Database開発ガイド』を参照

VARCHAR2およびCHARデータ型

VARCHAR2データ型には、可変長の文字リテラルが格納されます。**リテラル**とは、固定データ値です。

たとえば、'LILA'、'St. George Island'および'101'はすべて文字リテラルで、5001は数値リテラルです。文字リテラルは一重引用符で囲み、これにより、データベースでスキーマ・オブジェクト名と区別されます。

ノート:

このマニュアルでは、テキスト・リテラル、文字リテラルおよび文字列という用語を同じ意味で使用します。

VARCHAR2の列を含む表の作成時には、文字列の最大長を指定します。例2-1では、last_name列にVARCHAR2(25)というデータ型が指定されており、これは、この列に最大25バイトの名前を格納できることを意味します。

それぞれの行で、列の値はOracle Databaseにより可変長のフィールドとして格納され、値が最大長を超えている場合は、データベースからエラーが戻されます。たとえば、シングルバイト文字セットで、ある行の last_name列に10文字を入力すると、その行の列には25文字ではなく10文字(10バイト)のみが格納されます。VARCHAR2を使用することにより、領域の消費量を削減できます。

VARCHAR2とは異なり、CHARは、固定長の文字列を格納します。CHAR列を含む表の作成時には、列の文字列の長さを指定する必要があります。デフォルトは1バイトです。データベースは、空白を使用して指定された長さまで値を埋め込みます。

Oracle Databaseは、非空白埋め比較方法を使用してVARCHAR2値を比較し、空白埋め比較方法を使用してCHAR値を比較します。

関連項目:

空白埋め比較方法と非空白埋め比較方法の詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

NCHARおよびNVARCHAR2データ型

NCHARとNVARCHAR2 データ型は、Unicode文字データを格納します。

Unicodeはユニバーサル・エンコードされた文字セットであり、1つの文字セットで任意の言語の情報を格納できます。NCHARは、各国語文字セットに対応する固定長の文字列を格納し、NVARCHAR2は、可変長の文字列を格納します。

データベースの作成時に各国語文字セットを指定します。NCHARデータ型とNVARCHAR2データ型の文字セットは、AL16UTF16またはUTF8のいずれかにする必要があります。どちらの文字セットでも、Unicodeエンコーディングが使用されます。

NCHARまたはNVARCHAR2の列で表を作成すると、最大サイズは、必ず文字長セマンティクスになります。文字長セマンティクスはNCHARまたはNVARCHAR2のデフォルトであり、その唯一の長さセマンティクスです。

関連項目:

Oracleのグローバリゼーション・サポート機能の詳細は、『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。

数値データ型

Oracle Databaseの数値データ型には、固定小数点数、浮動小数点数、ゼロおよび無限大が格納されます。いくつかの数値データ型には操作結果の未定義を示す値も格納され、これは、非数値つまりNaNと呼ばれます。

Oracle Databaseでは、数値データが可変長形式で格納されます。それぞれの値は科学表記法で格納され、指数を格納するために1バイトが使用されます。データベースでは、仮数(浮動小数点数の一部で、有効桁数を含む部分)を格納するために最大20バイトが使用されます。Oracle Databaseでは、先行ゼロと後続ゼロは格納されません。

NUMBERデータ型

NUMBERデータ型には、固定小数点数および浮動小数点数が格納されます。実質的には、どのような大きさの数値でも、データベースに格納可能です。Oracle Databaseが稼働するオペレーティング・システムであれば、相互にデータを移植することが可能です。数値データの格納が必要となるほとんどの場合で、NUMBERデータ型をお薦めします。

固定小数点数は、NUMBER(*p*, *s*)という形式で指定します。この*p*および*s*は、次の文字を示しています。

- 精度

  精度は、全体の桁数を指定します。精度を指定しない場合、値は丸められずに、アプリケーションから提供されたとおりに列に格納されます。

- スケール

  スケールは、小数点から最下位有効桁までの桁数を指定します。正のスケールは、小数点の右側から最下位有効桁まで(最下位有効桁を含む)の桁数を指定します。負のスケールは、小数点の左側から最下位有効桁まで(ただし、最下位有効桁を含まない)の桁数を指定します。NUMBER(6)のように、精度を指定してスケールを指定しない場合、スケールは0になります。

例2-1では、salary列はNUMBER(8,2)型であるため、精度は8でスケールは2です。したがって、100,000という給料は、データベースに100000.00という値で格納されます。

浮動小数点数

Oracle Databaseには、浮動小数点数専用にBINARY_FLOATおよびBINARY_DOUBLEという2つの数値データ型が用意されています。

これらのデータ型は、NUMBERデータ型の基本機能をすべてサポートしています。ただし、NUMBERは10進精度を使用しますが、BINARY_FLOATおよびBINARY_DOUBLEは2進精度を使用します(2進精度により、算術計算が高速になり、通常は記憶域必要量が減少します)。

BINARY_FLOATとBINARY_DOUBLEは、近似値データ型です。どちらにも、10進値の正確な表現ではなく近似値表現が格納されます。たとえば、BINARY_DOUBLEまたはBINARY_FLOATでは、値0.1を正確に表現できません。この2つは、通常は科学関係の計算に使用されます。両者の動作は、JavaとXMLSchemaのデータ型FLOATおよびDOUBLEに似ています。

関連項目:

精度、スケールおよびその他の数値データ型の特性について学習するには、『Oracle Database SQL言語リファレンス』を参照してください。

日時データ型

日時データ型は、DATEとTIMESTAMPです。Oracle Databaseでは、タイムスタンプに対する包括的なタイムゾーン・サポートが提供されます。

DATEデータ型

DATEデータ型には、日付と時刻が格納されます。日時は、文字または数値のデータ型で表現できますが、DATEには、関連付けられた特別なプロパティがあります。

データベースの内部では、日付が数字として格納されます。日付は、それぞれ世紀、年、月、日、時、分および秒に対応する7バイトの固定長フィールドに格納されます。

ノート:

日付では算術演算が全面的にサポートされているため、数値の場合とまったく同様に日付を加算または減算できます。

データベースでは、指定された書式モデルに従って日付が表示されます。書式モデルは、日時の書式を文字列で記述する文字リテラルです。標準の日付書式はDD-MON-RRであり、日付が01-JAN-11の形式で表示されます。

RRは、YY(年の最後の2桁)と似ていますが、戻り値の世紀は、指定された2桁の年と現在の年の最後の2桁に応じて異なります。1999年の日付がデータベースに01-JAN-11と表示されるとします。日付書式にRRを使用する場合、11は2011を指しますが、日付書式にYYを使用する場合、11は1911を指します。デフォルトの日付書式は、データベース・インスタンスとセッション・レベルの両方で変更できます。

Oracle Databaseでは、時刻はHH:MI:SSの24時間形式で格納されます。時刻部分に何も指定しない場合、デフォルトでは、日付フィールドの時刻部分は00:00:00 A.M.になります。時刻のみを入力した場合、日付部分のデフォルトは現在の月の1日になります。

関連項目:

- 世紀および日付の書式マスクの詳細は、『Oracle Database開発ガイド』を参照してください。
- 日時書式コードの詳細は、『Oracle Database SQL言語リファレンス』を参照してください。
- 日時データ型を使用した算術演算の実行方法を学習するには、『Oracle Database開発ガイド』を参照

TIMESTAMPデータ型

TIMESTAMPデータ型は、DATEデータ型の拡張機能です。

TIMESTAMPには、DATEデータ型に格納される情報に加え、小数秒が格納されます。TIMESTAMPデータ型は、イベントの順序を追跡する必要があるアプリケーションなどで、正確な時間を格納するために役立ちます。

DATETIMEデータ型のTIMESTAMP WITH TIME ZONEおよびTIMESTAMP WITH LOCAL TIME ZONEでは、タイム・ゾーンも認識されます。ユーザーがデータを選択すると、値はユーザー・セッションのタイム・ゾーンに調整されます。このデータ型は、複数の地理的地域にまたがって日付情報を収集および評価する場合に役立ちます。

関連項目:

タイムスタンプ列のデータの作成と入力を実行する構文の詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

ROWIDデータ型

データベースに格納されているすべての行にはアドレスがあります。Oracle Databaseでは、ROWIDデータ型にデータベース内の各行のアドレス(ROWID)が格納されます。

ROWIDは、次のカテゴリに分類されます。

- 物理ROWIDは、ヒープ構成表、表クラスタ、および表パーティションや索引パーティションに行のアドレスを格納します。
- 論理ROWIDは、索引構成表に行のアドレスを格納します。
- 外部ROWIDは、ゲートウェイを介してアクセスされるDB2表などの外部表の識別子です。これは、Oracle Databaseの標準ROWIDではありません。

ユニバーサルROWIDまたはUROWIDと呼ばれるデータ型は、すべてのタイプのROWIDをサポートしています。

ROWIDの使用

Oracle Databaseでは、ROWIDを内部的に使用して索引が構築されます。

最も一般的なタイプはBツリー索引で、この索引は、複数の範囲に分割され、順序付けられたキーのリストから構成されます。それぞれのキーは、高速アクセスのために、対応する行のアドレスを指すROWIDに結び付けられています。

エンド・ユーザーとアプリケーション開発者は、次のような重要な用途にROWIDを使用することもできます。

- ROWIDは特定の行にアクセスする最も高速な方法です。
- ROWIDは表の編成を把握する機能を提供します。
- ROWIDは表の中の行の一意識別子です。

また、ROWIDデータ型を使用して定義した列を持つ表を作成することもできます。たとえば、データ型ROWIDの列を持つ例外表を定義して、整合性制約に違反する行のROWIDを格納できます。ROWIDデータ型を使用して定義されている列の動作は、値を更新できるなど、表の他の列と同様です。

ROWID疑似列

Oracleデータベースのそれぞれの表は、ROWIDという名前の疑似列を持っています。

疑似列は表の列のように使用できますが、実際に表に格納されているわけではありません。疑似列から値を選択できますが、疑似列に対して値の挿入、更新、削除はできません。疑似列は、引数のないSQL関数にも似ています。引数のない関数は通常、結果セットの各行に同じ値を戻しますが、疑似列は通常、行ごとに異なる値を戻します。

ROWID疑似列の値は、各行のアドレスを表す文字列です。これらの文字列は、ROWIDデータ型です。SELECTまたはDESCRIBEを実行して表の構造をリストしても疑似列は表示されず、疑似列は領域も使用しません。ただし、それぞれの行のROWIDは、予約語ROWIDを列名として使用すると、SQL問合せで取得できます。

次の例では、ROWID擬似列を問い合せて、employees表にある従業員100の行のROWIDを表示します。

```
SQL> SELECT ROWID FROM employees WHERE employee_id = 100;

ROWID
```

```
------------------
AAAPecAAFAAAABSAAA
```

関連項目:

- 「ROWID形式」
- アドレスによる行の識別方法を学習するには、『Oracle Database開発ガイド』を参照してください。
- ROWIDデータ型について学習するには、『Oracle Database SQL言語リファレンス』を参照してください。

書式モデルとデータ型

書式モデルは、格納された日時または数値データの書式を文字列で記述する文字リテラルです。書式モデルによってデータベース内の値の内部表現が変更されることはありません。

文字列を日付または数値に変換する場合、データベースが文字列を解釈する方法は、書式モデルによって決まります。SQLでは、TO_CHAR関数とTO_DATE関数の引数として書式モデルを使用し、データベースから戻される値の書式またはデータベースに格納される値の書式を設定できます。

次の文は、部門80の従業員の給与を選択し、TO_CHAR関数を使用してそれらの給与を数値書式モデル'$99,990.99'で指定された書式の文字値に変換します。

```
SQL> SELECT last_name employee, TO_CHAR(salary, '$99,990.99') AS "SALARY"
  2  FROM   employees
  3  WHERE  department_id = 80 AND last_name = 'Russell';

EMPLOYEE                  SALARY
------------------------- -----------
Russell                    $14,000.00
```

次の例では、TO_DATE関数で書式マスク'YYYY MM DD'を使用して採用日を更新し、文字列'1998 05 20'をDATE値に変換します。

```
SQL> UPDATE employees
  2  SET hire_date = TO_DATE('1998 05 20','YYYY MM DD')
  3  WHERE last_name = 'Hunold';
```

関連項目:

書式モデルについてさらに学習するには、『Oracle Database SQL言語リファレンス』を参照してください。

整合性制約

整合性制約とは、表の1つ以上の列の値を制限する名前付きの規則のことです。

データ整合性の規則により、表に無効なデータが入力されないようにします。また、特定の依存性が存在する場合、制約によって表の削除を回避できます。

制約が使用可能である場合、データベースは、入力または更新の時点でのデータをチェックします。Oracle Databaseでは制約に準拠していないデータは入力されません。制約が使用禁止になっている場合、Oracle Databaseでは制約に準拠してい

ないデータがデータベースに入力される可能性があります。

例2-1のCREATE TABLE文には、last_name、email、hire_dateおよびjob_idの各列にNOT NULL制約が指定されています。これらの制約句により、列と制約の条件が識別されます。これらの制約は、指定された列にNULL値が含まれないことを保証します。たとえば、職務IDのない新しい従業員を挿入しようとすると、エラーが発生します。

制約は、表の作成時または作成後に作成できます。必要に応じて、一時的に制約を使用禁止にできます。データベースでは、制約はデータ・ディクショナリに格納されます。

関連項目:

- 整合性制約について学習するには、「データ整合性」を参照
- データ・ディクショナリについて学習するには、「データ・ディクショナリの概要」を参照
- SQL制約句について学習するには、『Oracle Database SQL言語リファレンス』を参照してください。

表の記憶域

Oracle Databaseは、表領域の**データ・セグメント**を使用して表データを保持します。

セグメントには、データ・ブロックから構成されるエクステントが含まれます。表のデータ・セグメント(表クラスタの場合はクラスタ用のデータ・セグメント)は、表所有者のデフォルト表領域、またはCREATE TABLE文で指定された表領域に配置されます。

関連項目:

セグメントのタイプと、セグメントの作成方法について学習するには、「ユーザー・セグメント」を参照

表の編成

デフォルトでは、表はヒープとして編成されます。つまり、データベースでは、ユーザーが指定する順序ではなく、最も収まりすいように行が配置されます。このため、ヒープ構成表は、順不同の行のコレクションとなります。

ノート:

ヒープ構成表では、異なる編成の原理が使用されます。

ユーザーが行を追加すると、データベースではデータ・セグメントの最初の使用可能な空き領域に行が格納されます。挿入された順序どおりに行が取得されることは保証されていません。

hr.departments表は、ヒープ構成表です。この表には、部門ID、部門名、管理者ID、所在地IDの列があります。行が挿入されると、データベースでは、それらの行が収まる位置に格納されます。表セグメント内のデータ・ブロックには、次の例に示すように、順不同の行が含まれることがあります。

```
50,Shipping,121,1500
120,Treasury,,1700
70,Public Relations,204,2700
30,Purchasing,114,1700
```

```
130,Corporate Tax,,1700
10,Administration,200,1700
110,Accounting,205,1700
```

1つの表内のすべての行は、列の順序が同一です。データベースでは、通常、CREATE TABLE文でリストされた順序で列が格納されますが、この順序は保証されません。たとえば、表にLONG型の列があると、Oracle Databaseでは、この列は必ず最後に行に格納されます。また、表に新しい列を追加すると、その新しい列は最後に格納されます。

表には仮想列を含めることができ、仮想列は通常の列とは異なり、ディスク上の領域を使用しません。データベースによりユーザーの指定する一連の式または関数が計算され、必要に応じて仮想列から値が導出されます。仮想列には、索引付け、統計の収集および整合性制約の作成が可能です。このため、仮想列は通常の列とほとんど同じです。

関連項目:

- 「索引構成表の概要」
- 仮想列について学習するには、『Oracle Database SQL言語リファレンス』を参照してください。

行の格納

データベースでは、行がデータ・ブロックに格納されます。列数が256未満のデータを含む表の各行は、1つ以上の行断片に格納されます。

Oracle Databaseは、可能なかぎり各行を1つの行断片として格納します。ただし、行データのすべてを1つのデータ・ブロックに挿入できない場合、または既存の行が更新されてそのデータ・ブロックの容量を超えた場合、データベースは複数の行断片を使用して行を格納します。

表クラスタ内の行には、クラスタ化されていない表の行と同じ情報が格納されます。また、表クラスタ内の行には、それらの行が属するクラスタ・キーを参照する情報も格納されます。

関連項目:

データ・ブロックのコンポーネントについて学習するには、「データ・ブロックの形式」を参照

行断片のROWID

ROWIDとは、実際には行の10バイトの物理アドレスのことです。

ヒープ構成表のすべての行にはその表で一意のROWIDがあり、これは行断片の物理アドレスに相当します。表クラスタの場合、同じデータ・ブロックにある異なる表の行のROWIDが同一の場合もあります。

Oracle Databaseでは、ROWIDを内部的に使用して索引が構築されます。たとえば、Bツリー索引のそれぞれのキーは、高速アクセスのために、対応する行のアドレスを指すROWIDに結び付けられています。物理ROWIDを使用すると、表の1つの行に最速でアクセスでき、データベースではわずか1回のI/Oで行を取得できます。

関連項目:

- ROWIDの構造について学習するには、「ROWID形式」を参照

- Bツリー索引のタイプおよび構造について学習するには、「Bツリー索引の概要」を参照

NULL値の格納

NULLは、列に値がないことを示します。NULLは、データがない、不明である、または適切でないことを示します。

データ値を持つ2つの列の間にはさまれたNULLはデータベースに格納されます。このような場合、列の長さ(ゼロ)を格納する1バイトのみが必要となります。新しい行のヘッダーが前行の残りの列がNULLであることを示すため、行の末尾にあるNULLには記憶域は不要です。たとえば、表の最後の3列がNULLであれば、その3列にはデータは格納されません。

関連項目:

NULL値についてさらに学習するには、『Oracle Database SQL言語リファレンス』を参照してください。

表の圧縮

データベースでは、**表の圧縮**を使用して、表に必要な記憶域を削減します。

圧縮によりディスク領域を節約し、データベース・バッファ・キャッシュでのメモリー使用量を減らすことができ、場合によっては問合せ実行時間も短縮できます。表の圧縮は、データベース・アプリケーションに対して透過的です。

基本表圧縮と高度な行圧縮

ディクショナリ・ベースの表圧縮は、ヒープ構成表に対する高い圧縮率を実現します。

Oracle Databaseでは、次のタイプのディクショナリ・ベースの表圧縮をサポートしています。

- 基本表圧縮

  このタイプの圧縮は、バルク・ロード操作向けのものです。データベースでは、従来型のDMLを使用して変更されたデータは圧縮されません。基本表圧縮を実行するには、ダイレクト・パスのINSERT操作、ALTER TABLE . . . MOVE操作または表のオンライン再定義を使用する必要があります。

- 高度な行圧縮

  このタイプの圧縮はOLTPアプリケーション向けで、すべてのSQL操作によって操作されたデータを圧縮します。データベースでは、競争力のある圧縮率を実現しながら、非圧縮表に対するDMLとほぼ同じ時間でアプリケーションがDMLを実行できます。

これらの圧縮タイプの場合、圧縮された行は行優先形式で格納されます。この形式では、ある行のすべての列がまとめて格納され、それ以降の行も同様に行単位ですべての列がまとめて格納されます。重複する値は、ブロックの先頭に格納されているシンボル表の短い参照で置き換えられます。このため、非圧縮データを再作成するためにデータベースに必要な情報は、データ・ブロック自体に格納されます。

圧縮されたデータ・ブロックは、通常のデータ・ブロックとほぼ同じです。通常のデータ・ブロックに対して動作するデータベース機能のほとんどは、圧縮ブロックに対しても動作します。

表領域、表、パーティションまたはサブパーティション・レベルで圧縮を宣言できます。表領域レベルで指定した場合、その表領域に作成されたすべての表はデフォルトで圧縮されます。

例2-4 表レベルの圧縮

次の文は、高度な行圧縮をorders表に適用します。

```
ALTER TABLE oe.orders ROW STORE COMPRESS ADVANCED;
```

例2-5 パーティション・レベルの圧縮

次の部分的なCREATE TABLE文の例では、1つのパーティションに高度な行圧縮を指定し、もう1つのパーティションに基本表圧縮を指定しています。

```
CREATE TABLE sales (
    prod_id     NUMBER     NOT NULL,
    cust_id     NUMBER     NOT NULL, ... )
 PCTFREE 5 NOLOGGING NOCOMPRESS
 PARTITION BY RANGE (time_id)
 ( partition sales_2013 VALUES LESS THAN(TO_DATE(...)) ROW STORE COMPRESS BASIC,
   partition sales_2014 VALUES LESS THAN (MAXVALUE) ROW STORE COMPRESS ADVANCED );
```

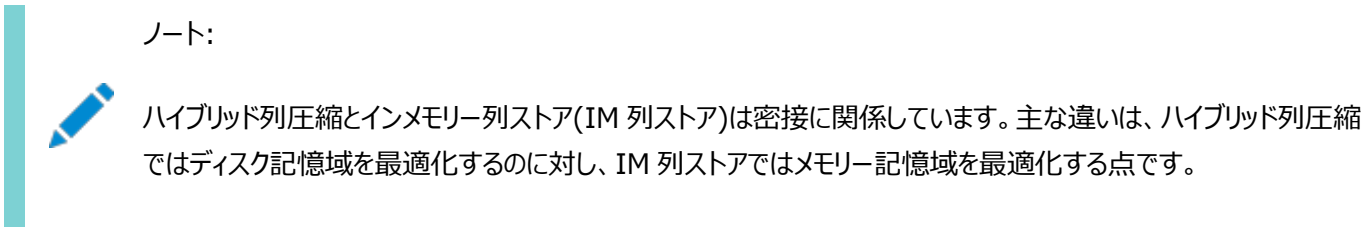
関連項目:

- 値が行に格納される方法を学習するには、「行形式」を参照
- 圧縮データ・ブロックの形式について学習するには、「データ・ブロックの圧縮」を参照してください
- SQL*Loaderを使用したダイレクト・パス・ロードについて学習するには、「SQL*Loader」を参照してください
- 表圧縮について学習するには、『Oracle Database管理者ガイド』および『Oracle Databaseパフォーマンス・チューニング・ガイド』を参照してください。

ハイブリッド列圧縮

ハイブリッド列圧縮では、行グループの同じ列がまとめてデータベースに格納されます。データ・ブロックのデータは行優先形式では格納されませんが、行および列の両方を組み合せた方法が使用されます。

同じデータ型で、類似した特性を持つ列データをまとめて格納することで、圧縮によって節減できる記憶域が大幅に増加します。データベースでは、任意のSQL操作によって処理されるデータを圧縮しますが、ダイレクト・パス・ロードでは圧縮レベルがより高くなります。データベース操作は圧縮オブジェクトに対して透過的に実行されるため、アプリケーションを変更する必要はありません。



ノート:

ハイブリッド列圧縮とインメモリー列ストア(IM 列ストア)は密接に関係しています。主な違いは、ハイブリッド列圧縮ではディスク記憶域を最適化するのに対し、IM 列ストアではメモリー記憶域を最適化する点です。

関連項目:

IM列ストアについてさらに学習するには、「インメモリー領域」を参照

ハイブリッド列圧縮のタイプ

基礎となるストレージがハイブリッド列圧縮をサポートしている場合は、必要に応じて異なるタイプの圧縮を指定できます。

圧縮オプションは次のとおりです。

- ウェアハウス圧縮

このタイプの圧縮は、記憶域の節減用に最適化されており、データ・ウェアハウス・アプリケーション向けです。

- アーカイブ圧縮

  このタイプの圧縮は、最大の圧縮レベル用に最適化されており、履歴データおよび変更されないデータ向けです。

ハイブリッド列圧縮は、データ・ウェアハウス用またはOracle Exadataストレージ上の意思決定支援アプリケーション用に最適化されています。Oracle Exadataでは、ハイブリッド列圧縮を使用して圧縮された表の問合せパフォーマンスが最大化され、Oracle Exadataストレージ・サーバーに統合されている処理能力、メモリーおよびインフィニバンド・ネットワーク帯域幅の利点を活用できます。

その他のOracleストレージ・システムでは、ハイブリッド列圧縮がサポートされ、Oracle Exadataストレージと同様の領域の節減は可能ですが、Oracle Exadataストレージと同じレベルの問合せパフォーマンスは得られません。これらのストレージ・システムでは、まれにしかアクセスしない古いデータのインデータベースでのアーカイブには、ハイブリッド列圧縮が理想的です。

圧縮ユニット

ハイブリッド列圧縮では、行の集合を格納するために、**圧縮ユニット**と呼ばれる論理的な構造が使用されます。

表にデータをロードする場合、データベースでは、行のグループは列形式で、それぞれの列の値としてまとめて圧縮されて格納されます。データベースでは、一連の行の列データが圧縮された後、そのデータが圧縮ユニットに収められます。

たとえば、ハイブリッド列圧縮をdaily_sales表に適用するとします。1日の終わりに、品目と販売数が表に移入され、品目IDと日付がコンポジット主キーになります。次の表にdaily_sales内の行のサブセットを示します。

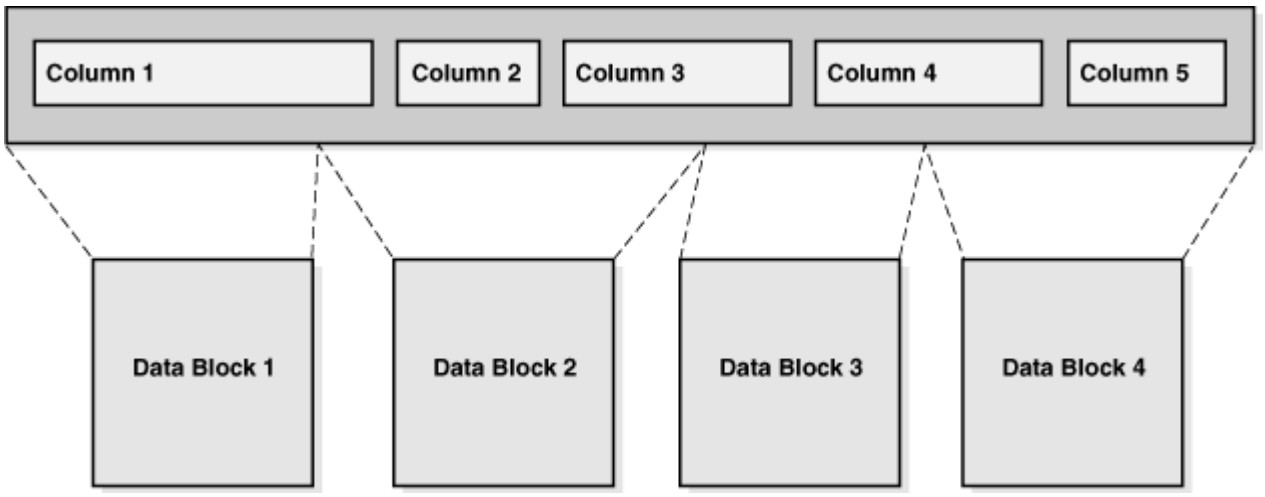
表2-2 サンプル表daily_sales

Item_ID	Date	Num_Sold	Shipped_From	Restock
1000	01-JUN-18	2	WAREHOUSE1	Y
1001	01-JUN-18	0	WAREHOUSE3	N
1002	01-JUN-18	1	WAREHOUSE3	N
1003	01-JUN-14	0	WAREHOUSE2	N
1004	01-JUN-18	2	WAREHOUSE1	N
1005	01-JUN-18	1	WAREHOUSE2	N

この行サブセットが1つの圧縮ユニットに格納されるとします。ハイブリッド列圧縮では、各列の値がまとめて格納され、その後で複数のアルゴリズムを使用してそれぞれの列が圧縮されます。データベースでは、列のデータ型、列内の実際の値のカーディナリティ、ユーザーの選択した圧縮レベルなどの様々な要因に基づいてアルゴリズムが選択されます。

次の図に示すように、各圧縮ユニットは、複数のデータ・ブロックにまたがることができます。特定の列の値は、複数のブロックにまたがる場合とまたがらない場合があります。

図2-4 圧縮ユニット



ハイブリッド列圧縮が領域節約につながらない場合、データベースではデータがDBMS_COMPRESSION.COMP_BLOCK形式で格納されます。この場合、データベースではハイブリッド列圧縮セグメントにあるブロックにOLTP圧縮が適用されます。

関連項目:

- 「行ロック(TX)」
- ハイブリッド列圧縮のライセンス要件について学習するには、『Oracle Databaseライセンス情報ユーザー・マニュアル』を参照
- ハイブリッド列圧縮の使用方法を学習するには、『Oracle Database管理者ガイド』を参照
- CREATE TABLEの構文およびセマンティクスについては、『Oracle Database SQL言語リファレンス』を参照
- DBMS_COMPRESSIONパッケージについて学習するには、『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』を参照

DMLとハイブリッド列圧縮

ハイブリッド列圧縮は、様々なタイプのDML操作の行のロックと密接に関連しています。

ダイレクト・パス・ロードと従来型挿入

ハイブリッド列圧縮を使用する表にデータをロードするときは、従来型挿入またはダイレクト・パス・ロードのいずれかを使用できます。ダイレクト・パス・ロードでは、表全体がロックされ、同時実行性が低下します。

Oracle Database 12cリリース2 (12.2)には、ハイブリッド列圧縮形式への従来型の配列挿入のサポートが追加されています。従来型の配列挿入の利点は次のとおりです。

- 挿入された行で行レベル・ロックが使用され、同時実行性が向上します。
- 自動データ最適化(ADO)およびヒート・マップが行レベル・ポリシーに対するハイブリッド列圧縮をサポートしています。したがって、DMLアクティビティがセグメントの他の部分で発生している場合でも、データベースでは対象のブロックについてハイブリッド列圧縮が使用されます。

アプリケーションで従来型の配列挿入が使用されると、Oracle Databaseでは次の条件に一致したときに行が圧縮ユニットに格納されます。

- 表がASSM表領域に格納されています。
- 互換性レベルは12.2.0.1以上です。
- LONG型の列がないことを含めて、表定義が既存のハイブリッド列圧縮の表制約を満たしており、行依存性もありません。

従来型挿入では、REDOとUNDOが生成されます。したがって、従来型のDML文によって作成された圧縮ユニットは、そのDMLとともにロールバックまたはコミットされます。データベースでは、従来型データ・ブロックに格納された行に対する場合と同様に、自動的に索引メンテナンスが実行されます。

更新と削除

デフォルトでは、圧縮ユニットのいずれかの行で更新または削除が発生した場合に、データベースによってそのユニットのすべての行がロックされます。この問題を回避するために、表で行レベルのロックを有効にすることができます。この場合は、更新または削除操作の影響を受ける行のみがデータベースによってロックされます。

関連項目:

- 「自動セグメント領域管理」
- 行ロック(TX)
- 従来型挿入の実行方法を学習するには、『Oracle Database管理者ガイド』を参照
- INSERT文について学習するには、『Oracle Database SQL言語リファレンス』を参照

表クラスタの概要

表クラスタとは、共通の列を共有し、関連するデータを同じブロックに格納する表のグループのことです。

表がクラスタ化されると、1つのデータ・ブロックに複数の表の行を含めることができます。たとえば、ブロックにemployees表とdepartments表のいずれか1つのみでなく、両方の表の行を格納できます。

クラスタ・キーとは、クラスタ化表で共有される1つ以上の列のことです。たとえば、employeesおよびdepartments表はdepartment_id列を共有しています。表クラスタの作成時と表クラスタに追加されるすべての表の作成時にクラスタ・キーを指定します。

クラスタ・キー値とは、特定の行のセットに対するクラスタ・キー列の値のことです。department_id=20などの同じクラスタ・キー値を含むすべてのデータは、物理的にまとめて格納されます。各クラスタ・キー値は、その値を持つ行がいくつかの表に含まれているとしても、クラスタとクラスタ索引にそれぞれ一度のみ格納されます。

たとえば、HR管理者が2つの書棚を使用しており、1つの書棚には従業員フォルダの箱があり、もう1つの書棚には部門フォルダの箱があるとします。ユーザーは、特定の部門に属する全従業員のフォルダを頻繁に問い合せます。管理者は、取り出しやすいようにすべての箱を1つの書棚に配置しなおします。管理者は、部門IDごとに箱を分割します。その結果、部門20に属する従業員のすべてのフォルダと部門20自体のフォルダが1つの箱に入り、部門100の従業員のフォルダと部門100のフォルダも別の箱に入り、他の部門のフォルダもそれぞれの箱に入ります。

複数の表が主に(変更ではなく)問合せの対象となっていて、これらの表のレコードが頻繁に同時に問い合されるか、または結合される場合は、表のクラスタ化を検討してください。表クラスタは、異なる表の関連した行を、同じデータ・ブロック内に格納するため、表クラスタを適切に使用すると、クラスタ化されていない表と比較して次のような利点があります。

- クラスタ化表を結合する場合のディスクI/Oが減少します。
- クラスタ化表を結合する場合のアクセス時間が改善されます。
- クラスタ・キー値が行ごとに繰り返し格納されないため、関連する表データと索引データの格納に必要な記憶域が少なくなります。

通常、表のクラスタ化は、次のような場合には適していません。

- 表が頻繁に更新される場合。
- 表の全表スキャンが頻繁に必要な場合。
- 表で切捨てが必要な場合。

索引付きクラスタの概要

索引クラスタとは、索引を使用してデータを検索する表クラスタのことです。**クラスタ索引**とは、クラスタ・キーのBツリー索引のことです。クラスタ索引は、クラスタ化表に行を挿入する前に作成しておく必要があります。

例2-6 表クラスタと関連する索引の作成

次の例に示すように、department_idという名前のクラスタ・キーを指定してクラスタemployees_departments_clusterを作成するとします。

```
CREATE CLUSTER employees_departments_cluster
   (department_id NUMBER(4))
SIZE 512;

CREATE INDEX idx_emp_dept_cluster
   ON CLUSTER employees_departments_cluster;
```

HASHKEYS句は指定されないため、employees_departments_clusterは索引付きクラスタになります。前述の例では、クラスタ・キーdepartment_idにidx_emp_dept_clusterという名前の索引を作成します。

例2-7 索引付きクラスタでの表の作成

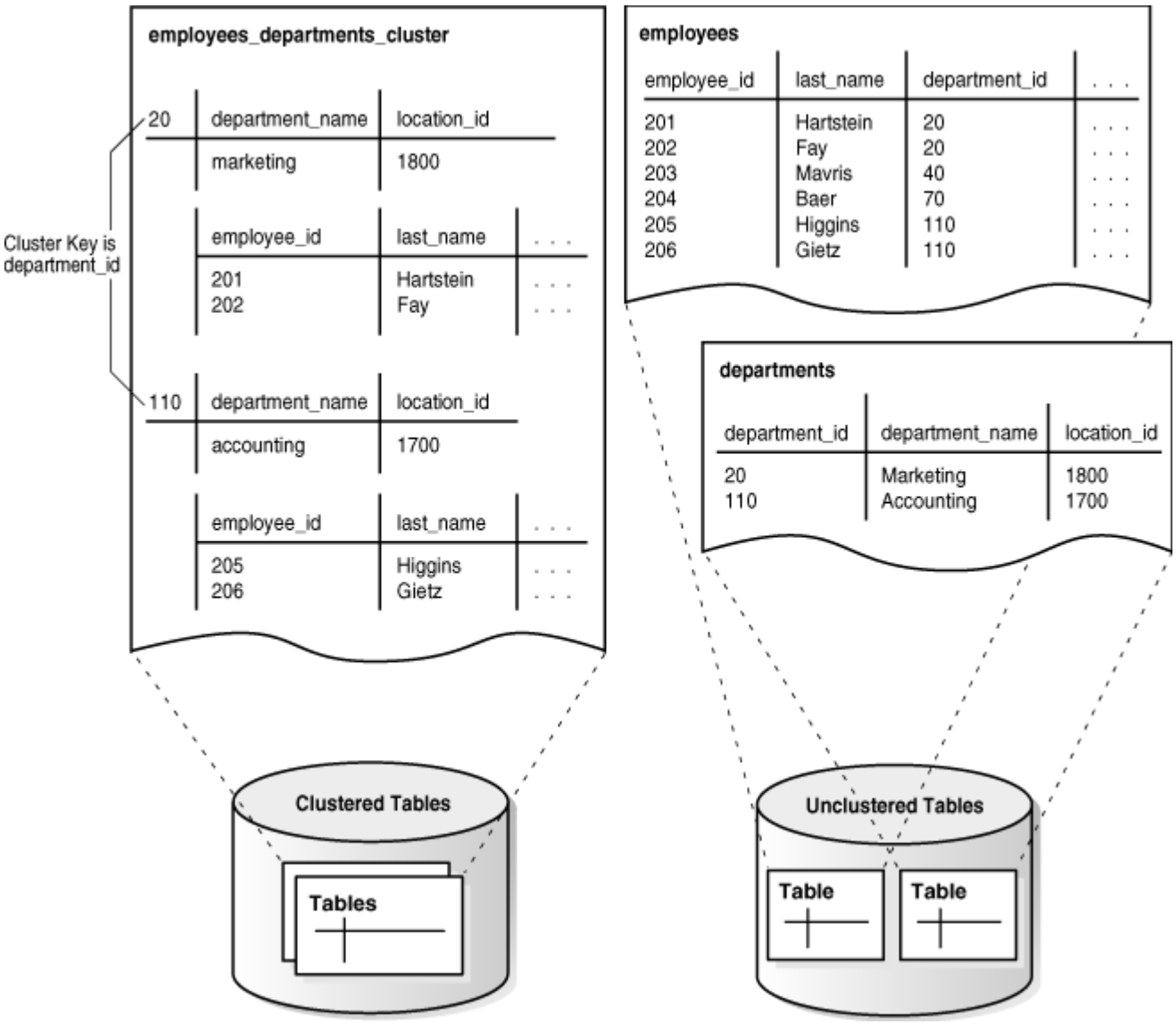
その後、次のように、このクラスタにemployees表とdepartments表を作成し、クラスタ・キーとしてdepartment_id列を指定します(省略記号は、列が指定される場所のマークです)。

```
CREATE TABLE employees ( ... )
   CLUSTER employees_departments_cluster (department_id);

CREATE TABLE departments ( ... )
   CLUSTER employees_departments_cluster (department_id);
```

employees表とdepartments表に行を追加するとします。データベースでは、employees表とdepartments表の各部門の行がすべて同じデータ・ブロックに物理的に格納されます。データベースでは、行がヒープに格納され、索引を使用してこれらの行が検索されます。

図2-5に、employeesとdepartmentsを含むemployees_departments_cluster表クラスタを示します。データベースでは、従業員20に属する従業員や部門110などの行がまとめて格納されます。表がクラスタ化されていない場合、データベースでは関連する行がまとめて格納されることは保証されません。

図2-5 クラスタ化表データ



Bツリー・クラスタ索引は、データを含むブロックのデータベース・ブロック・アドレス(DBA)にクラスタ・キー値を関連付けます。たとえば、キー20の索引エントリには、部門20の従業員のデータを含むブロックのアドレスが表示されます。

```
20, AADAAAA9d
```

クラスタ索引は、クラスタ化されていない表の索引と同様に個別に管理され、表クラスタとは別の表領域に存在することがあります。

関連項目:

- 索引の概要
- 索引付きクラスタの作成方法と管理方法を学習するには、『Oracle Database管理者ガイド』を参照してください。
- CREATE CLUSTERの構文およびセマンティクスは、『Oracle Database SQL言語リファレンス』を参照してください。

ハッシュ・クラスタの概要

ハッシュ・クラスタは、索引キーが**ハッシュ関数**に置き換わることを除いて、索引付きクラスタと似ています。別個のクラスタ索引は存在しません。ハッシュ・クラスタでは、データが索引になります。

索引付きの表または索引付きのクラスタでは、Oracle Databaseは、別個の索引に格納されたキー値を使用して、表の行を見つけます。索引付きの表または表クラスタを検索または格納するには、次のようにデータベースは、最低2回のI/Oを実行する

必要があります。

- 1回以上のI/Oによる、索引内でのキー値の検索または格納
- 表または表クラスタ内の行を読取り/書込みするためにさらに1回のI/O

Oracle Databaseでは、ハッシュ・クラスタ内で行を検索または格納するため、ハッシュ関数を行のクラスタ・キー値に適用します。結果として得られるハッシュ値はクラスタ内の1つのデータ・ブロックに対応し、データベースにより、発行された文による読取り/書込みがそのデータ・ブロックに対して実行されます。

ハッシングは、データ検索のパフォーマンスを改善するために表データを格納するオプションの方法です。ハッシュ・クラスタは、次の条件が満たされている場合に便利です。

- 表の問合せが、表の変更よりもはるかに頻繁に行われる場合。
- ハッシュ・キー列に対する問合せの条件が、多くの場合、WHERE department_id=20のような等価条件である場合。このような問合せでは、クラスタ・キー値がハッシュ化されます。ハッシュ・キー値は、行が格納されているディスク領域を直接指します。
- 当然、ハッシュ・キーの数が多くなれば、キー値も一緒に格納されるためデータのサイズが大きくなります。

ハッシュ・クラスタの作成

ハッシュ・クラスタを作成するときは、索引付きクラスタの場合と同じCREATE CLUSTER文を使用しますが、ハッシュ・キーを別途指定する必要があります。クラスタのハッシュ値の数はハッシュ・キーに応じて増減します。

クラスタ・キーは、索引付きクラスタのキーと同様に、単一の列、またはクラスタ内の表が共有するコンポジット・キーです。ハッシュ・キー値は、クラスタ・キー列に挿入される実際の値または推定値です。たとえば、クラスタ・キーがdepartment_idの場合は、ハッシュ・キー値が10、20、30のような値になります。

Oracle Databaseでは、無数にあるハッシュ・キー値を入力として受け付け、限定された数のバケットに選別するハッシュ関数が使用されます。各バケットにはハッシュ値という一意の数値IDが付与されます。各ハッシュ値は、ハッシュ・キー値(部門10、20、30など)に対応する行の格納ブロックのデータベース・ブロック・アドレスにマップされます。

次の例では、存在すると推測される部門数が100であるため、HASHKEYSが100に設定されています。

```
CREATE CLUSTER employees_departments_cluster
   (department_id NUMBER(4))
SIZE 8192 HASHKEYS 100;
```

employees_departments_clusterの作成後に、そのクラスタ内にemployees表とdepartments表を作成できます。次に、索引付きクラスタの場合と同様に、ハッシュ・クラスタにデータをロードできます。

関連項目:

- 「索引付きクラスタの概要」
- ハッシュ・クラスタの作成方法と管理方法を学習するには、『Oracle Database管理者ガイド』を参照してください。

ハッシュ・クラスタの問合せ

ハッシュ・クラスタの問合せで、ユーザーが入力するキー値をハッシュする方法は、データベースが決定します。

たとえば、ユーザーが次のような問合せを頻繁に実行し、p_idに様々な部門ID番号を入力するとします。

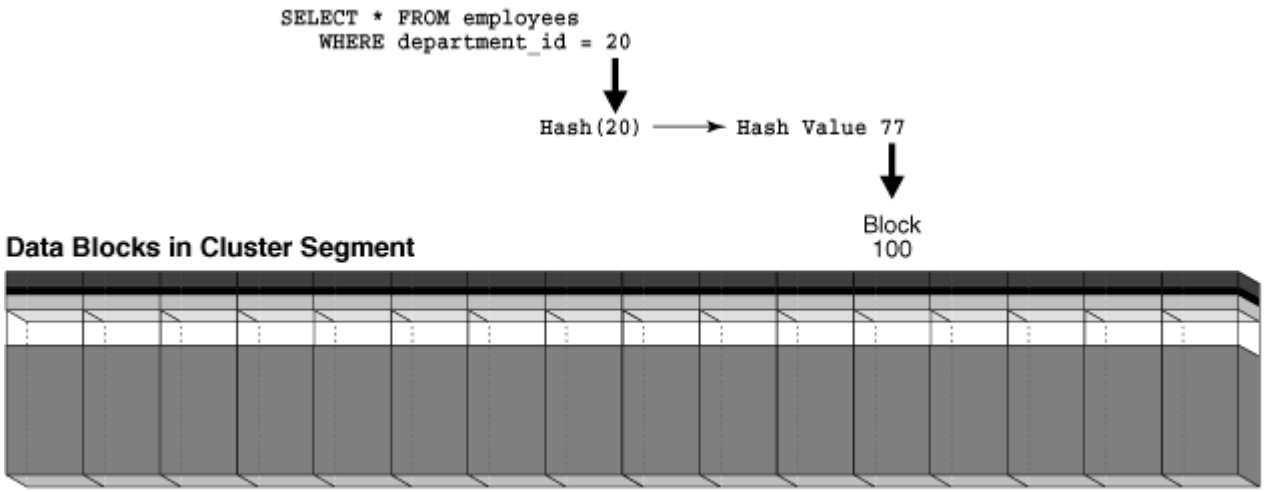
```
SELECT *
```

```
FROM   employees
WHERE  department_id = :p_id;

SELECT *
FROM   departments
WHERE  department_id = :p_id;

SELECT *
FROM   employees e, departments d
WHERE  e.department_id = d.department_id
AND    d.department_id = :p_id;
```

ユーザーがdepartment_id=20の従業員を問い合せた場合は、この値がバケット77にハッシュされます。department_id=10の従業員を問い合せた場合は、この値がバケット15にハッシュされます。データベースは、内部的に生成されるハッシュ値を使用して、問合せ対象の部署に対応する従業員行を格納したブロックを検索します。

次の図に、ハッシュ・クラスタ・セグメントを水平方向に並べたブロック行として示します。この図に示すように、1件の問合せにつき、1回のI/Oでデータを取得できます。

図2-6 ハッシュ・クラスタからのデータの取得



ハッシュ・クラスタの制限は、索引のないクラスタ・キーのレンジ・スキャンを使用できないことです。「ハッシュ・クラスタの作成」で作成したハッシュ・クラスタに、独立した索引が存在しないものとします。IDが20から100の部門の問合せは、20から100の間のすべての取り得る値をハッシュしていないため、ハッシュ・アルゴリズムを使用できません。索引が存在しないため、データベースでは全体スキャンを実行する必要があります。

関連項目:

「索引レンジ・スキャン」

様々なハッシュ・クラスタ

単一表ハッシュ・クラスタとは、同時に1つの表のみをサポートするために最適化されたハッシュ・クラスタのバージョンのことです。ハッシュ・キーと行との間には、1対1のマッピングがあります。

単一表ハッシュ・クラスタは、ユーザーが主キーを使用して表に迅速にアクセスする必要がある場合に便利です。たとえば、ユーザーがemployee_idを使用してemployees表の従業員レコードを頻繁に検索する場合などです。

ソート済のハッシュ・クラスタに、ハッシュ関数の各値に対応する行が格納されると、データベースはソートされた順序で効率的に行を戻すことが可能になります。データベースは最適化されたソートを内部的に実行します。ソートされた状態のデータを常に使

用するアプリケーションの場合は、この方式によりデータの取得が速くなる可能性があります。たとえば、アプリケーションがorders表のorder_date列で常にソートする場合が考えられます。

関連項目:

単一表ハッシュ・クラスタとソート済のハッシュ・クラスタの作成方法を学習するには、『Oracle Database管理者ガイド』を参照してください

ハッシュ・クラスタの記憶域

Oracle Databaseは、ハッシュ・クラスタのための領域を索引付きクラスタとは異なる方法で割り当てます。

「ハッシュ・クラスタの作成」の例では、HASHKEYSに存在する可能性のある部門数を指定し、SIZEに各部門に関連付けられたデータのサイズを指定しています。次の式に従って記憶域領域の値が計算されます。

```
HASHKEYS * SIZE / database_block_size
```

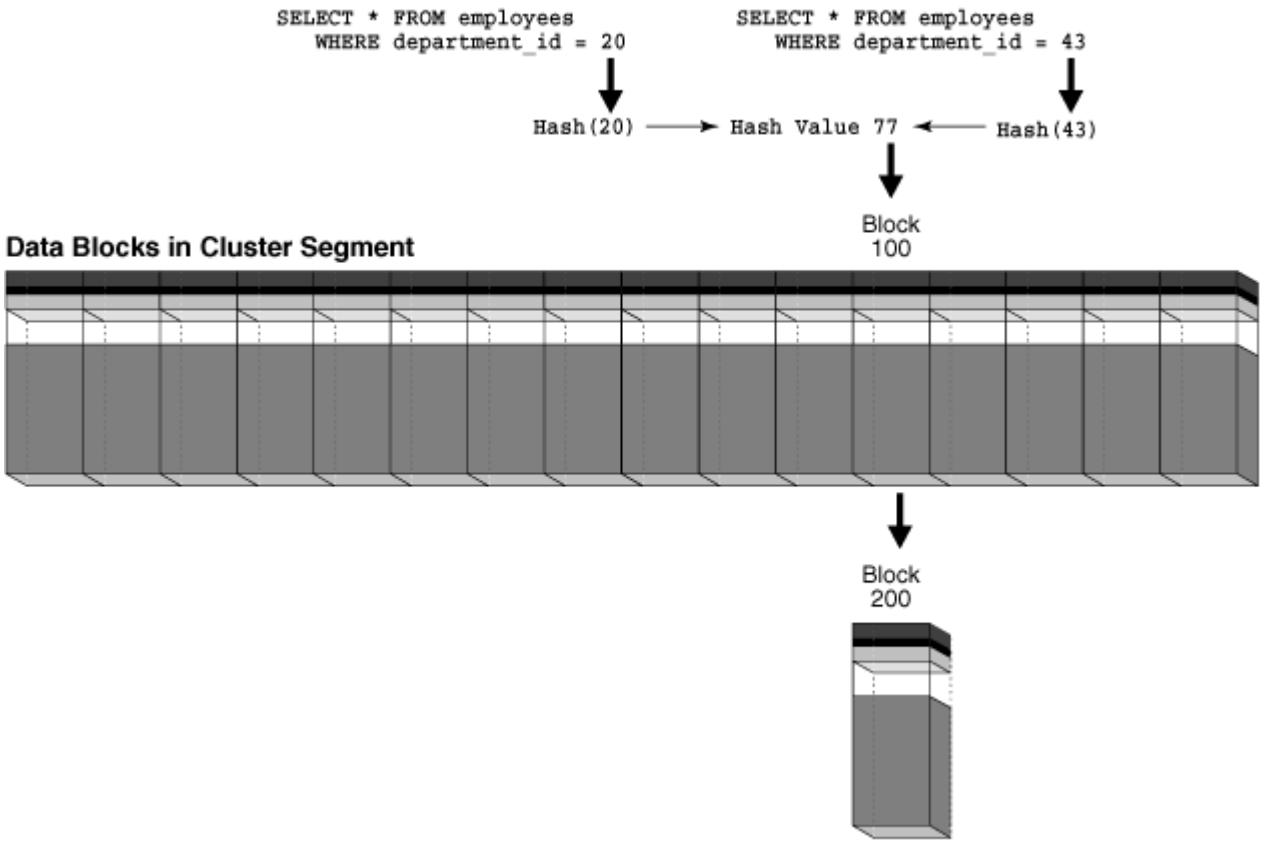
したがって、「ハッシュ・クラスタの作成」に示す例,のようにブロック・サイズが4096バイトの場合は、ハッシュ・クラスタに200以上のブロックが割り当てられます。

Oracle Databaseでは、クラスタに挿入可能なハッシュ・キー値の数に制限はありません。たとえば、HASHKEYSが100になっている場合でも、departments表に一意の部門を200個挿入することも可能です。ただし、ハッシュ値の数がハッシュ・キーの数よりも多くなると、ハッシュ・クラスタ取得の効率が悪くなります。

取得の問題について、図2-6でブロック100が部門20の行でまったく空きがない場合の例で説明します。ユーザーがdepartment_idが43の新しい部署をdepartments表に挿入します。部門の数がHASHKEYS値よりも大きいため、department_id 43はハッシュ値77にハッシュされますが、これは、department_id 20に使用されているハッシュ値と同じです。複数の入力値を同じ出力値にハッシュすることをハッシュ衝突と呼びます。

ユーザーが部門43のクラスタに行を挿入しようとしても、ブロック100はいっぱいであるため、行は格納されません。ブロック100は新しいオーバーフロー・ブロック(ここではブロック200とします)にリンクされ、その新しいブロックに挿入行が格納されます。つまり、ブロック100とブロック200の両方に、どちらの部門のデータも格納されている可能性があることになります。図2-7に示すように、部門20または部門43を問い合せると、ブロック100とその関連ブロックであるブロック200からデータが取得されるため、I/Oが2回必要になります。この問題を解決するには、別々のHASHKEYS値を持つクラスタを作成しなおす必要があります。

図2-7 ハッシュ衝突が発生している場合のハッシュ・クラスタからのデータの取得



関連項目:

ハッシュ・クラスタの領域を管理する方法を学習するには、『Oracle Database管理者ガイド』を参照してください。

属性クラスタ表の概要

属性クラスタ表は、ディスク上の近接するデータを、ユーザーが指定したクラスタリング・ディレクティブに基づいて格納するヒープ構成表です。ディレクティブでは、単一または複数の表で列を指定します。

ディレクティブは次のようになります。

- CLUSTERING ... BY LINEAR ORDERディレクティブは、指定された列に応じて表のデータを並べます。

  問合せにおいてクラスタリング句で指定された列の接頭辞を付与するときに、デフォルトのBY LINEAR ORDERクラスタリングを使用することを検討します。たとえば、sh.salesの問合せで、顧客IDまたは顧客IDと製品IDの両方を指定する場合、線形の列順序cust_id、prod_idを使用して、表のデータをクラスタ化できます。

- CLUSTERING ... BY INTERLEAVED ORDERディレクティブは、複数列のI/O削減を可能にするZオーダー関数に似ている特別なアルゴリズムを使用して、1つ以上の表のデータを並べます。

  問合せで様々な列の組合せを指定する際に、BY INTERLEAVED ORDERクラスタリングを使用することを検討します。たとえば、sh.salesの問合せで、様々なディメンションを様々な順序で指定する場合、これらのディメンションの列に応じて、sales表のデータをクラスタ化できます。

属性クラスタリングは、ダイレクト・パス挿入操作でのみ使用可能です。従来のDMLでは無視されます。

この項では、次の項目について説明します。

- 属性クラスタリングされた表の利点
- 結合属性クラスタ表

- ゾーンを使用するI/Oの削減
- 線形順序付けを使用する属性クラスタ表
- インターリーブされた順序付けを使用する属性クラスタ表

属性クラスタリングされた表の利点

属性クラスタ表の主な利点はI/Oの削減で、表スキャンのI/OコストとCPUコストを著しく削減できます。I/Oの削減は、ゾーンを使用して行うか、ディスク上のクラスタ値の物理的近接により物理I/Oを削減することで行われます。

属性クラスタ表には次の利点があります。

- スター・スキーマのディメンション列に基づいて、ファクト表をクラスタリングできます。

  スター・スキーマでは、大部分の問合せはディメンション表を修飾し、ファクト表を修飾しないため、ファクト表の列を基準にするクラスタリングは効果的ではありません。Oracle Databaseは、ディメンション表の列上でのクラスタリングをサポートしています。

- I/Oの削減は、次の複数のシナリオで生じます。
  - Oracle Exadata Storage Indexes、Oracle In-Memory最小/最大プルーニングまたはゾーン・マップで使用した場合
  - OLTPアプリケーションの接頭辞を付与する問合せで、線形順序で属性クラスタリングを使用する場合
  - BY INTERLEAVED ORDERクラスタリングのクラスタリング列のサブセットにおいて
- 属性クラスタリングではデータ圧縮が改善されるため、これにより、表スキャン・コストが間接的に削減されます。

  ディスク上で同じ値が互いに接近している場合、データベースでは、これらの値を容易に圧縮できます。

- Oracle Databaseでは、索引の記憶域コストおよび保守コストは発生しません。

関連項目:

属性クラスタ表の利点の詳細は、Oracle Databaseデータ・ウェアハウス・ガイドを参照してください

結合属性クラスタ表

結合された列に基づく属性クラスタリングは、**結合属性クラスタリング**と呼ばれます。表クラスタと異なり、結合属性クラスタ表は、表グループのデータを同じデータベース・ブロックには格納しません。

たとえば、ディメンション表productsと結合された属性クラスタ表salesを考えてみます。sales表にはsales表の行のみが含まれますが、行の順序はproducts表から結合された列の値に基づいています。データ移動操作、ダイレクト・パス挿入操作およびCREATE TABLE AS SELECT操作中に、適切な結合が実行されます。これに対し、salesおよびproductsが標準の表クラスタ内にある場合、データ・ブロックには両方の表の行が含まれます。

関連項目:

結合属性クラスタリングについてさらに学習するには、『*Oracle Databaseデータ・ウェアハウス・ガイド*』を参照してください。

ゾーンを使用するI/Oの削減

ゾーンは、関連する列の最小値および最大値を格納する、一連の隣接するデータ・ブロックです。

SQL文に、ゾーンに格納されている列の述部が含まれる場合、データベースは述部の値をゾーンに格納されている最小値および最大値と比較します。このようにして、データベースはSQL実行時に読み取るゾーンを判別します。

I/Oの削減とは、データベースの問合せを満たすデータを含まない、表ブロックまたは索引ブロックをスキップする機能です。この削減により、表スキャンのI/OコストとCPUコストが著しく削減されます。

ゾーン・マップ

ゾーン・マップは、データ・ブロックをゾーンに分割する、独立したアクセス構造です。Oracle Databaseでは、各ゾーン・マップを**マテリアライズド・ビュー**のタイプとして実装します。

表でCLUSTERINGが指定されるたびに、データベースの指定したクラスタリング列でゾーンが自動的に作成されます。ゾーン・マップは、列の最小値および最大値と、属性クラスタ表内の連続するデータ・ブロックを互いに関係付けます。属性クラスタ表では、ゾーン・マップを使用してI/Oの削減を実行します。

ゾーン・マップを使用しない属性クラスタ表を作成できます。属性クラスタ表なしでゾーン・マップを作成することもできます。たとえば、取引が時系列で並べられる株式取引表など、行が列セット上に自然に並べられる表にゾーン・マップを作成できます。DDL文を実行して、ゾーン・マップを作成、削除および管理できます。

関連項目:

- 「マテリアライズド・ビューの概要」
- ゾーン・マップについてさらに学習するには、『Oracle Databaseデータ・ウェアハウス・ガイド』を参照してください。

ゾーンの目的

ゾーンのゆるいアナロジーとして、書棚の小仕切り(これはデータ・ブロックと似ています)を使用するセールス・マネージャを考えてみます。

各小仕切りには、出荷日別に並べられた、顧客に販売したシャツを表す受領物(行)があります。このアナロジーで、ゾーン・マップは、索引カードのスタックに似ています。各カードは、小仕切り1-10など、小仕切りのゾーン(隣接する範囲)に相当します。ゾーンごとに、カードにはゾーンに格納されている受領物の最小および最大出荷日がリストされます。

特定日付にどのシャツが出荷されたかを知りたい場合、マネージャはリクエストされた日付を含む日付範囲が来るまでカードをめくり、小仕切りゾーンをノートにとってから、リクエストされた受領物についてこのゾーン内の小仕切りのみを検索します。この方法で、マネージャは受領物のために書棚のすべての小仕切りを探す必要がなくなります。

ゾーン・マップの仕組み: 例

この例は、述語が定数を含む問合せにおいてゾーン・マップがデータをプルーニングする方法を示しています。

次のlineitem表を作成するとします。

```
CREATE TABLE lineitem
  ( orderkey      NUMBER        ,
    shipdate      DATE          ,
    receiptdate   DATE          ,
    destination   VARCHAR2(50)  ,
    quantity      NUMBER        );
```

表lineitemには、各ブロックに2つの行がある4つのデータ・ブロックが含まれます。表2-3に、表の8つの行を示します。

表2-3 lineitem表のデータ・ブロック

Block	orderkey	shipdate	receiptdate	destination	quantity
1	1	1-1-2014	1-10-2014	San_Fran	100
1	2	1-2-2014	1-10-2014	San_Fran	200
2	3	1-3-2014	1-9-2014	San_Fran	100
2	4	1-5-2014	1-10-2014	San_Diego	100
3	5	1-10-2014	1-15-2014	San_Fran	100
3	6	1-12-2014	1-16-2014	San_Fran	200
4	7	1-13-2014	1-20-2014	San_Fran	100
4	8	1-15-2014	1-30-2014	San_Jose	100

CREATE MATERIALIZED ZONEMAP文を使用して、lineitem表にゾーン・マップを作成できます。各ゾーンには2つのブロックが含まれ、orderkey、shipdateおよびreceiptdate列の最小値と最大値を格納しています。表2-4は、ゾーン・マップを示しています。

表2-4 lineitem表のゾーン・マップ

Block Range	min orderkey	max orderkey	min shipdate	max shipdate	min receiptdate	max receiptdate
1-2	1	4	1-1-2014	1-5-2014	1-9-2014	1-10-2014
3-4	5	8	1-10-2014	1-15-2014	1-15-2014	1-30-2014

次の問合せを実行すると、データベースはゾーン・マップを読み取り、日付1-3-2014が最小日付と最大日付の間にあるため、ブロック1とブロック2のみをスキャンして、ブロック3とブロック4をスキップできます。

```
SELECT * FROM lineitem WHERE shipdate = '1-3-2014';
```

関連項目:

- ゾーン・マップの使用方法を学習するには、Oracle Databaseデータ・ウェアハウス・ガイドを参照してください
- CREATE MATERIALIZED ZONEMAP文の構文とセマンティクスの詳細は、Oracle Database SQL言語リファレンスを参照してください

線形順序付けを使用する属性クラスタ表

表の線形順序付けスキームでは、ユーザーが指定した属性に基づいて、行を特定順序で範囲に分割します。Oracle Databaseでは、主キーと外部キーの関係を介して接続された単一の表または複数の表で、線形順序付けをサポートしています。

たとえば、sales表ではcust_id列とprod_id列を範囲に分割してから、これらの範囲をディスク上で一緒にクラスタ化します。表にBY LINEAR ORDERディレクティブを指定すると、述部で接頭辞列またはディレクティブのすべての列を指定する際に、I/Oが著しく削減されることがあります。

salesの問合せでは通常、顧客IDまたは顧客IDと製品IDの組合せを指定するとします。属性クラスタ表を作成して、こうした問合せでI/O削減の利点を享受することができます。

```
CREATE TABLE sales
(
   prod_id     NOT NULL NUMBER
,  cust_id     NOT NULL NUMBER
,  amount_sold NUMBER(10,2) ...
)
CLUSTERING
  BY LINEAR ORDER (cust_id, prod_id)
  YES ON LOAD YES ON DATA MOVEMENT
  WITH MATERIALIZED ZONEMAP;
```

列cust_idおよびprod_idの両方、または接頭辞cust_idを付与する問合せでは、I/O削減が生じます。prod_idはBY LINEAR ORDER句の接頭辞であるため、prod_idのみを付与する問合せでは、著しいI/O削減は生じません。次の例では、表スキャン時にデータベースでI/Oを削減する方法を示します。

例2-8 cust_idのみを指定

アプリケーションは、次の問合せを発行します。

```
SELECT * FROM sales WHERE cust_id = 100;
```

sales表はBY LINEAR ORDERクラスタであるため、データベースではcust_id値100を含むゾーンのみを読み取る必要があります。

例2-9 prod_idとcust_idを指定

アプリケーションは、次の問合せを発行します。

```
SELECT * FROM sales WHERE cust_id = 100 AND prod_id = 2300;
```

sales表はBY LINEAR ORDERクラスタであるため、データベースではcust_id値100とprod_id値2300を含むゾーンのみを読み取る必要があります。

関連項目:

- 線形順序付けを使用する表をクラスタ化する方法を学習するには、『*Oracle Database*データ・ウェアハウス・ガイド』を参照してください。
- BY LINEAR ORDER文の構文とセマンティクスの詳細は、『*Oracle Database SQL*言語リファレンス』を参照してください。

インターリーブされた順序付けを使用する属性クラスタ表

インターリーブされた順序付けでは、Zオーダーとよく似た技術を使用します。

インターリーブされた順序付けにより、データベースではクラスタリング列の述部の任意のサブセットに基づいて、I/Oをプルーニングできます。インターリーブされた順序付けは、データ・ウェアハウスのディメンション階層では便利です。

線形順序付けを使用する属性クラスタ表と同様に、Oracle Databaseでは、主キーと外部キーの関係を介して接続された単一の表または複数の表で、インターリーブされた順序付けをサポートしています。属性クラスタ表以外の表の列は、外部キーでリンクして、属性クラスタ表に結合する必要があります。

大規模なデータ・ウェアハウスでは多くの場合、データをスター・スキーマで編成します。ディメンション表は親子階層を使用し、外部キーでファクト表に接続されます。インターリーブされた順序でファクト表をクラスタリングすることで、データベースでは表スキャン時に、特別な関数を使用してディメンション列の値をスキップできます。

例2-10 インターリーブされた順序付けの例

データ・ウェアハウスにsalesファクト表とその2つのディメンション表customersとproductsが含まれているとします。ほとんどの問合せでは、customers表階層(cust_state_province, cust_city)と製品階層(prod_category, prod_subcategory)に述部があります。次の例の文の一部に示すように、sales表にインターリーブされた順序付けを使用できます。

```
CREATE TABLE sales
(
   prod_id NUMBER NOT NULL
,  cust_id NUMBER NOT NULL
,  amount_sold NUMBER(10,2) ...
)
CLUSTERING sales
   JOIN products ON (sales.prod_id = products.prod_id)
   JOIN customers ON (sales.cust_id = customers.cust_id)
   BY INTERLEAVED ORDER
   (
     (  products.prod_category
     ,  products.prod_subcategory
     ),
     (  customers.cust_state_province
     ,  customers.cust_city
     )
   )
WITH MATERIALIZED ZONEMAP;
```

ノート:

BY INTERLEAVED ORDER 句で指定した列が実際のディメンション表に存在する必要はありませんが、主キーと外部キーの関係を介して接続されている必要があります。

アプリケーションで結合内のsales、productsおよびcustomers表を問い合せるとします。問合せでは、述部でcustomers.prod_categoryおよびcustomers_cust_state_province列を次のように指定します。

```
SELECT cust_city, prod_sub_category, SUM(amount_sold)
FROM   sales, products, customers
WHERE  sales.prod_id = products.prod_id
```

```
AND    sales.cust_id = customers.cust_id
AND    customers.prod_category = 'Boys'
AND    customers.cust_state_province = 'England - Norfolk'
GROUP BY cust_city, prod_sub_category;
```

前述の問合せでは、prod_categoryおよびcust_state_province列は、CREATE TABLEの例に示すクラスタリング定義の一部です。sales表のスキャン時に、データベースはゾーン・マップを参照して、このゾーン内のROWIDにのみアクセスすることができます。

関連項目:

- ディメンションの概要
- インターリーブされた順序付けを使用する表をクラスタ化する方法を学習するには、『*Oracle Database*データ・ウェアハウス・ガイド』を参照してください。
- BY INTERLEAVED ORDER文の構文とセマンティクスの詳細は、『*Oracle Database SQL*言語リファレンス』を参照してください。

一時表の概要

一時表には、トランザクションまたはセッションの期間中にのみ存在するデータが保持されます。

一時表内のデータはセッション専用です。各セッションで参照および変更できるのは、そのセッション自体のデータのみです。

グローバル一時表または**プライベート一時表**のいずれかを作成できます。次の表に、これらの間の主な違いを示しています。

表2-5 一時表の特徴

特徴	グローバル	プライベート
命名規則	永続表の場合と同じです。	ORA$PTT_を接頭辞として付ける必要があります
表定義の可視性	すべてのセッション	表を作成したセッションのみ
表定義の記憶域	ディスク	メモリーのみ
タイプ	トランザクション固有(ON COMMIT DELETE ROWS)またはセッション固有(ON COMMIT PRESERVE ROWS)	トランザクション固有(ON COMMIT DROP DEFINITION)またはセッション固有(ON COMMIT PRESERVE DEFINITION)

cursor-duration一時表と呼ばれる3番目のタイプの一時表が、特定のタイプの問合せのためにデータベースによって自動的に作成されます。

関連項目:

cursor-duration一時表についてさらに学習するには、Oracle Database SQLチューニング・ガイドを参照してください。

一時表の目的

一時表は、結果セットをバッファリングする必要があるアプリケーションで役立ちます。

たとえば、スケジューリング・アプリケーションでは、大学生が学期のオプション・コースのスケジュールを作成できます。グローバル一時表内の1行は、各スケジュールを表します。セッション中、スケジュール・データはプライベートです。学生がスケジュールを選択すると、アプリケーションは選択されたスケジュールの行を永続表に移します。セッションの終了時、データベースではグローバル一時表内にあったスケジュール・データが自動的に削除されます。

プライベート一時表は、動的レポート作成アプリケーションに役立ちます。たとえば、顧客リソース管理(CRM)アプリケーションが同じユーザーで無制限に接続し、複数のセッションが同時にアクティブになる可能性があります。各セッションは、新規トランザクションごとにORA$PTT_crmという名前のプライベート一時表を作成します。アプリケーションは、定義が異なればすべてのセッションで同じ表名を使用できます。データと定義は、そのセッションのみで表示できます。表の定義は、トランザクションが終了するか表を手動で削除するまで保持されます。

一時表でのセグメントの割当て

永続表と同様に、グローバル一時表は、データ・ディクショナリで静的に定義されている永続オブジェクトです。プライベート一時表の場合、メタデータはメモリー内にのみ存在しますが、ディスク上の一時表領域に格納できます。

グローバル一時表とプライベート一時表では、セッションで初めてデータが挿入される際にデータベースによって一時セグメントが割り当てられます。セッションでデータがロードされるまで、表は空のように見えます。トランザクション固有の一時表の場合、データベースではトランザクションの終了時に一時セグメントの割当てが解除されます。セッション固有の一時表の場合、データベースではセッションの終了時に一時セグメントの割当てが解除されます。

関連項目:

「一時セグメント」

一時表の作成

一時表は、CREATE ... TEMPORARY TABLE文で作成されます。

GLOBAL TEMPORARY TABLEまたはPRIVATE TEMPORARY TABLEのいずれかを指定します。どちらの場合も、ON COMMIT句では、表データをトランザクション固有(デフォルト)とセッション固有のどちらにするかを指定します。それぞれのPL/SQLストアド・プロシージャではなく、データベース自体に一時表を作成します。

プライベートではないグローバル一時表の索引は、CREATE INDEX文で作成できます。これらの索引も一時的です。その索引のデータは、一時表のデータと同じセッションまたはトランザクション・スコープを持ちます。また、グローバル一時表のビューまたはトリガーを作成することも可能です。

関連項目:

- ビューの概要
- 「トリガーの概要」
- 一時表の作成および管理方法を学習するには、Oracle Database管理者ガイドを参照してください
- CREATE ... TEMPORARY TABLEの構文およびセマンティクスについては、Oracle Database SQL言語リファレンスを

参照してください。

外部表の概要

外部表では、外部ソースのデータに対して、データベース内の表にあるデータのようにアクセスできます。

アクセス・ドライバが提供されている任意の形式のデータを使用できます。SQL (シリアルまたはパラレル)、PL/SQLおよびJavaを使用して外部データを問い合せることができます。

外部表の目的

外部表は、Oracle Databaseアプリケーションで非リレーショナル・データにアクセスする必要がある場合に役立ちます。

たとえば、SQLベースのアプリケーションで次の形式のレコードを含むテキスト・ファイルにアクセスする必要があるとします。

```
100,Steven,King,SKING,515.123.4567,17-JUN-03,AD_PRES,31944,150,90
101,Neena,Kochhar,NKOCHHAR,515.123.4568,21-SEP-05,AD_VP,17000,100,90
102,Lex,De Haan,LDEHAAN,515.123.4569,13-JAN-01,AD_VP,17000,100,90
```

外部表を作成し、外部表の定義で指定した場所にテキスト・ファイルをコピーすると、SQLを使用してテキスト・ファイル内のレコードを問い合せることができます。同様に、外部表を使用してJSONドキュメントまたはLOBへの読取り専用アクセスを提供できます。

データ・ウェアハウス環境では、抽出、変換およびロード(ETL)タスクを実行する場合に外部表が役立ちます。たとえば、外部表を使用して、データ・ロード・フェーズと変換フェーズをパイプライン処理できます。この手法により、データベース内で後続の処理に備えてデータをステージングする必要がなくなります。

外部表を仮想列または通常の列でパーティション化できます。一部のパーティションが内部で、一部が外部であるハイブリッド・パーティション表を作成することもできます。内部パーティションと同様に、外部パーティションでもパーティション・プルーニングやパーティション・ワイズ結合などのパフォーマンス拡張の利点を享受できます。たとえば、パーティション化された外部表を使用して、Hadoop Distributed File System (HDFS)または非SQLデータベースに格納されている大量の非リレーショナル・データを分析できます。

関連項目:

パーティション表

オブジェクト・ストアのデータ

外部表を使用して、オブジェクト・ストア内のデータにアクセスできます。

Oracleでは、オペレーティング・システム・ファイルやビッグ・データ・ソースに存在する外部データへのアクセスと、オブジェクト・ストア内の外部データへのアクセスをサポートしています。オブジェクト・ストレージはクラウドでは一般的であり、個々のオブジェクト(メタデータを含むあらゆるタイプの非構造化データ)を単純なコンテナにグループ化することで、管理のためのフラットなアーキテクチャを提供します。オブジェクト・ストレージは主にクラウドのデータ・ストレージ・アーキテクチャですが、オンプレミスのストレージ・ハードウェアとしても使用できます。

オブジェクト・ストアのデータにアクセスするには、DBMS_CLOUDパッケージを使用するか、外部表を手動で定義します。DBMS_CLOUDパッケージは追加機能を提供し、Oracle Autonomous Databaseと完全に互換性があるため、このパッケー

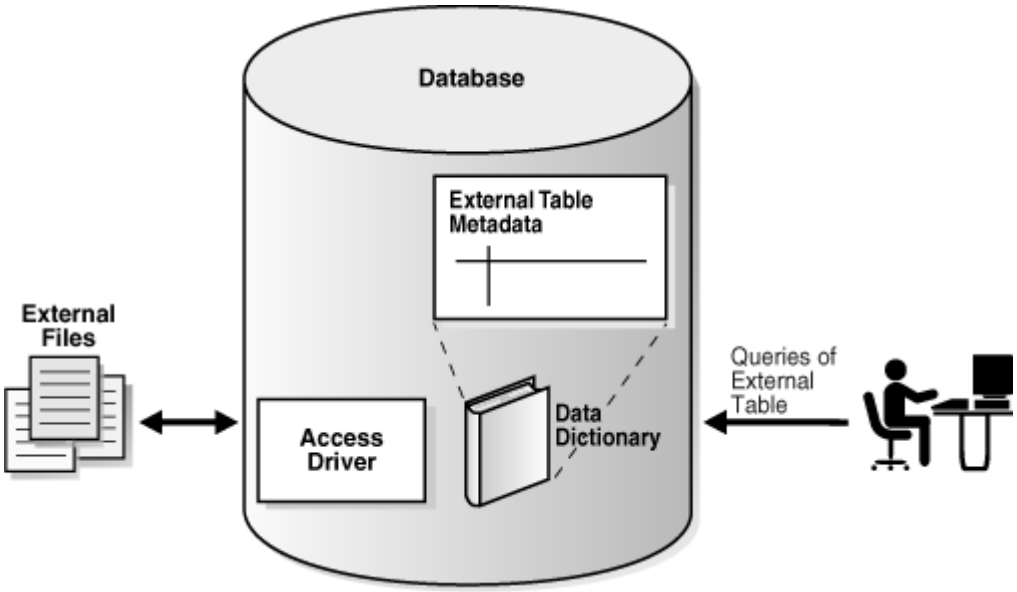
ジの使用をお薦めします。

外部表のアクセス・ドライバ

アクセス・ドライバとは、データベースの外部データを解釈するAPIのことです。アクセス・ドライバはデータベースの内部で動作し、データベースはアクセス・ドライバを使用して外部表のデータを読み取ります。外部表の定義に合うようにデータファイルのデータに必要な変換を行うのは、アクセス・ドライバおよび外部表レイヤーの役割です。

次の図に、外部データへのSQLアクセスを示します。

図2-8 外部表



Oracleでは、外部表のために次のアクセス・ドライバを提供しています。

- ORACLE_LOADER (デフォルト)

  SQL*Loaderでサポートされているほとんどの形式を使用して、外部ファイルにアクセスできます。ORACLE_LOADERドライバを使用して外部ファイルを作成または更新することや、外部ファイルへの追加を行うことはできません。

- ORACLE_DATAPUMP

  外部データをアンロードまたはロードできます。アンロード操作では、データベースからデータが読み取られ、1つ以上の外部ファイルで表される外部表にデータが挿入されます。外部ファイルが作成されると、データベースはデータの更新や外部ファイルへのデータの追加を実行できなくなります。ロード操作では、外部表が読み取られ、そのデータがデータベースにロードされます。

- ORACLE_HDFS

  Hadoop Distributed File System (HDFS)に格納されたデータを抽出できます。

- ORACLE_HIVE

  Apache Hiveデータベースに格納されたデータにアクセスできます。ソース・データは、HDFS、HBase、Cassandraまたはその他のシステムに格納できます。他のアクセス・ドライバとは異なり、ORACLE_HIVEでは外部のメタデータ・ストアから場所の情報が取得されるため、場所は指定できません。

- ORACLE_BIGDATA

  Apache Parquet、Apache Avro、Apache ORCおよびテキスト形式など、構造化形式と非構造化形式の両方で保存されたデータへの読取り専用アクセスを有効にします。このドライバを使用すると、テストや小規模なデータ・セットに有用なローカル・データを問い合せることもできます。

外部表の作成

内部的には、外部表の作成はデータ・ディクショナリでのメタデータの作成を意味します。通常の表とは異なり、外部表にはデータベースに格納されているデータは示されず、外部でのデータの格納方法も示されません。かわりに、外部表のメタデータに外部表レイヤーでのデータベースに対するデータの表示方法が記述されています。

CREATE TABLE ... ORGANIZATION EXTERNAL文は、2つの部分に分かれています。外部表の定義には、列型を記述します。この定義は、外部データをデータベースにロードすることなくSQL問合せを実行できるビューに似ています。文の2番目の部分は、外部データを列にマップします。

外部表は、CREATE TABLE AS SELECTでORACLE_DATAPUMPアクセス・ドライバを指定して作成される場合を除き、読取り専用です。外部表の制限は、索引付けされた列および列オブジェクトがサポートされないことです。

関連項目:

- 外部表について学習するには、『*Oracle Databaseユーティリティ*』を参照してください。
- 外部表、外部接続およびディレクトリ・オブジェクトの管理について学習するには、『*Oracle Database管理者ガイド*』を参照してください。
- 外部表の作成と問合せの詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください

ブロックチェーン表の概要

ブロックチェーン表は、集中管理ブロックチェーン・アプリケーション向けに設計された追加専用の表です。

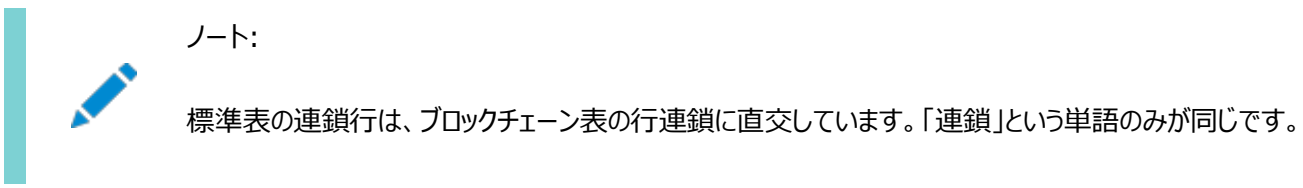
Oracle Blockchain Tableでは、ピアとはデータベースを信頼して改ざん不可能な元帳を維持するデータベース・ユーザーです。元帳は、アプリケーションによって定義および管理されるブロックチェーン表として実装されます。既存のアプリケーションは、新しいインフラストラクチャやプログラミング・モデルを必要とせずに、不正から保護することができます。トランザクションのスループットは標準の表よりも低くなりますが、ブロックチェーン表のパフォーマンスは、分散ブロックチェーンの場合よりも優れています。

ブロックチェーン表は、許可されたDMLがINSERTコマンドのみであるため、追加専用です。この表はUPDATE、DELETE、MERGE、TRUNCATEおよびダイレクト・パス・ロードを許可しません。データベース・トランザクションは、ブロックチェーン表と標準の表にまたがることができます。たとえば、1つのトランザクションで、標準の表と2つの異なるブロックチェーン表に行を挿入できます。

行連鎖

ブロックチェーン表では、**行連鎖**は、ハッシング・スキームとともにリンクされた一連の行です。

行連鎖は、データベース・インスタンスIDとチェーンIDの一意の組合せによって識別されます。ブロックチェーン表内の行は、正確に1つの行連鎖に属しています。単一の表は複数の行連鎖をサポートします。



ノート:

標準表の連鎖行は、ブロックチェーン表の行連鎖に直交しています。「連鎖」という単語のみが同じです。

連鎖のすべての行にはそれぞれ順序番号が付いています。データベースは、各連鎖の行でSHA2-512ハッシュ計算を使用して行の順序を決定します。挿入されたすべての行に対するハッシュは、連鎖に挿入されている行のハッシュ値、および挿入された

行の**行内容**から導出されます。

行内容

行内容は、行の列データと連鎖内の前の行のハッシュ値を含む、連続したバイトのシーケンスです。

ブロックチェーン表を作成すると、データベースでは非表示の複数の列が作成されます。たとえば、bank列とdeposit列を使用して、ブロックチェーン表bank_ledgerを作成できます。

```
CREATE BLOCKCHAIN TABLE bank_ledger (bank VARCHAR2 (128), deposit NUMBER)
  NO DROP UNTIL 31 DAYS IDLE
  NO DELETE UNTIL 31 DAYS AFTER INSERT
  HASHING USING "SHA2_512" VERSION "v1";
```

データベースでは、接頭辞ORABCTABのある非表示の列、ORABCTAB_INST_ID$、ORABCTAB_CHAIN_ID$、ORABCTAB_SEQ_NUM$などが自動的に作成されます。変更または管理できないこれらの非表示列によって、改ざん防止アルゴリズムを実装します。このアルゴリズムは、コミット時に一意の表レベル・ロックを特定の順序で取得することでデッドロックを回避します。

ノート:

ブロックチェーン表の行内容は、標準データ・ブロックに格納されます。このリリースの Oracle Database では、ブロックチェーン表で表クラスタはサポートされていません。

インスタンスID、チェーンIDおよびシーケンス番号によって、行が一意に識別されます。各行には、非表示の列ORABCTAB_HASH$に格納されるプラットフォームに依存しないSHA2-512ハッシュがあります。ハッシュは、挿入された行の内容、および連鎖内の前の行のハッシュに基づいています。

行の列値のデータ形式は、列のメタデータと内容のバイトで構成されます。列のメタデータは、表内の位置、データ型、NULLステータス、バイト長などの特性を記述する20バイトの構造です。列の内容は、1行の値を表すバイト・セットです。たとえば、値ChaseのASCII表現は43 68 61 73 65です。SQLでDUMP関数を使用すると、列のメタデータと内容の両方を取得できます。

ハッシュ計算の行内容には、複数の列からの列データ形式(連鎖内の前の行のハッシュ値、ユーザー定義列および非表示の固定列数)が含まれます。

ブロックチェーン表のユーザー・インタフェース

標準の表と同様に、ブロックチェーン表はSQLによって作成され、スカラー・データ型、LOBおよびパーティションをサポートします。ブロックチェーン表に対して索引およびトリガーを作成することもできます。

ブロックチェーン表を作成するには、CREATE BLOCKCHAIN TABLE文を使用します。ブロックチェーン表には、NO DROP UNTIL n DAYS IDLE句で指定された保存期間があります。この表はDROP TABLEを使用して削除できます。

Oracle Blockchain Tableでは、次のインタフェースをサポートしています。

- DBMS_BLOCKCHAIN_TABLEパッケージでは、表の行に対して様々な操作を実行できます。たとえば、以前に挿入した行の内容に署名を適用するには、SIGN_ROWプロシージャを使用します。行が改ざんされていないことを確認するには、VERIFY_ROWSを使用します。(NO DELETE句で指定した)保存期間が経過した後に行を削除するには、DELETE_ROWSを使用します。
- DBMS_TABLE_DATAパッケージは、列のバイト値を取得するプロシージャを提供します。ハッシュまたはユーザー署名の計

算対象となる行データの行内容を取得できます。

- DBA_BLOCKCHAIN_TABLESビューには、行の保存期間、表の削除が許可されるまでの非アクティブ期間、ハッシュ・アルゴリズムなどの表メタデータが表示されます。

ノート:

- ブロックチェーン表の管理方法を学習するには、Oracle Database 管理者ガイドを参照してください
- DBMS_BLOCKCHAIN_TABLE パッケージについて学習するには、『Oracle Database PL/SQL パッケージおよびタイプ・リファレンス』を参照してください
- DBMS_TABLE_DATA パッケージについて学習するには、『Oracle Database PL/SQL パッケージおよびタイプ・リファレンス』を参照してください
- DBA_BLOCKCHAIN_TABLES ビューについて学習するには、Oracle Database リファレンスを参照してください

不変表の概要

不変表は、内部関係者による不正なデータ変更や人為的エラーによる偶発的なデータ変更を防止する読取り専用の表です。

不正な変更は、内部関係者の資格証明へのアクセス権を持つ安全性の低い従業員や悪意のある従業員が試みる可能性があります。

新しい行は不変表に追加できますが、既存の行は変更できません。保存期間は、不変表と不変表内の行の両方に指定する必要があります。指定した行の保存期間が経過すると、行は使用されなくなります。不変表から削除できるのは、使用されなくなった行のみです。

不変表には、システム生成の非表示列が含まれます。この列は、ブロックチェーン表のものと同じです。行が挿入されると、ORABCTAB_CREATION_TIME$列とORABCTAB_USER_NUMBER$列にNULL以外の値が設定されます。残りのシステム生成の非表示列の値は、NULLに設定されます。

不変表を使用する場合、既存のアプリケーションを変更する必要はありません。

オブジェクト表の概要

オブジェクト表は、すべての行がオブジェクトを表す特殊な表です。

Oracleの**オブジェクト型**は、名前、属性およびメソッドを含むユーザー定義データ型です。オブジェクト型を使用すると、顧客や発注書などの実社会のエンティティをデータベース内のオブジェクトとしてモデル化できます。

オブジェクト型では論理構造が定義されますが、記憶域は作成されません。次の例では、department_typという名前のオブジェクト型を作成します。

```
CREATE TYPE department_typ AS OBJECT
   ( d_name      VARCHAR2(100),
     d_address   VARCHAR2(200) );
/
```

次の例では、オブジェクト型department_typのdepartments_obj_tという名前のオブジェクト表を作成し、その表に行を挿入します。departments_obj_t表の属性(列)は、オブジェクト型の定義から導出されます。

```
CREATE TABLE departments_obj_t OF department_typ;
INSERT INTO departments_obj_t VALUES ('hr', '10 Main St, Sometown, CA');
```

リレーショナル列と同様に、オブジェクト表に含めることができるのは、表と同一の宣言された型のオブジェクト・インスタンスなど、1種類の行のみです。デフォルトで、オブジェクト表のすべての行オブジェクトに関連する論理オブジェクト識別子(OID)があり、この識別子によりオブジェクト表の行オブジェクトが一意に識別されます。オブジェクト表のOID列は、非表示の列です。

関連項目:

- Oracle Databaseのオブジェクト・リレーショナル機能について学習するには、Oracle Databaseオブジェクト・リレーショナル開発者ガイドを参照してください
- CREATE TYPEの構文およびセマンティクスについては、Oracle Database SQL言語リファレンスを参照してください

3 索引と索引構成表

索引は、表の行へのアクセスを高速化できるスキーマ・オブジェクトです。索引構成表は、索引構造に格納された表です。

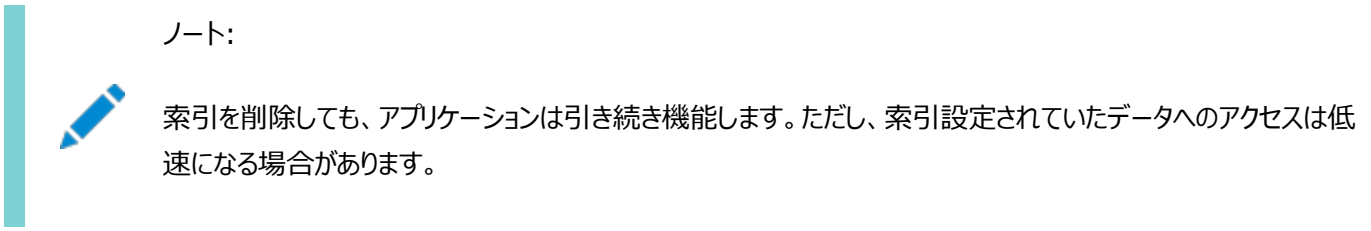
この章の構成は、次のとおりです。

- 索引の概要
- Bツリー索引の概要
- ビットマップ索引の概要
- ファンクション索引の概要
- アプリケーション・ドメイン索引の概要
- 索引構成表の概要

索引の概要

索引は、表または**表クラスタ**に関連するオプションの構造であり、索引によってデータ・アクセスを高速化できる場合があります。

索引は、関連するオブジェクトのデータから論理的にも物理的にも独立したスキーマ・オブジェクトです。そのため、索引を削除または作成しても、索引付けされた表に物理的な影響はありません。



ノート:

索引を削除しても、アプリケーションは引き続き機能します。ただし、索引設定されていたデータへのアクセスは低速になる場合があります。

たとえば、ある人事管理マネージャが複数の段ボール箱を棚に置いて管理しているとします。従業員情報を収めたフォルダが、それらの箱にランダムに収納されています。従業員Whalen(ID 200)のフォルダは、1つ目の箱の下から10フォルダ目であり、King(ID 100)のフォルダは3つ目の箱の一番下にあります。マネージャがフォルダを見つけるには、1つ目の箱の一番下から一番上まですべてのフォルダを確認し、目的のフォルダが見つかるまで次の箱で同じことを繰り返します。アクセスを高速化するために、マネージャは従業員全員のIDとフォルダの位置を順にリストした索引を作成できます。

```
ID 100: Box 3, position 1 (bottom)
ID 101: Box 7, position 8
ID 200: Box 1, position 10
.
.
.
```

同様に、マネージャは従業員の姓や部門IDなどの個別の索引を作成できます。

この項では、次の項目について説明します。

- 索引の長所と短所
- 索引の使用可能性と可視性
- キーと列
- コンポジット索引

- 一意索引と非一意索引
- 索引のタイプ
- データベースで索引を管理する方法
- 索引記憶域

索引の長所と短所

索引の有無によって、SQL文の表現を変更する必要はありません。

索引は、単一行のデータへの高速なアクセス・パスです。それは実行のスピードにのみ影響を与えます。索引の付いたデータ値では、索引はその値を含んでいる行の位置を直接示すポインタとして機能します。

索引が表の1つ以上の列に存在する場合、データベースで、ランダムに分散している行の小さなセットを表から取得できる場合があります。索引は、ディスクI/Oを削減するための様々な手段のうちの1つです。ヒープ構成表に索引がない場合、そのデータベースでは、値の検索に全表スキャンを実行する必要があります。たとえば、索引付けされていないhr.departments表の位置2700の問合せでは、データベースは、すべてのブロックのすべての行を検索する必要があります。データ量が増加すると、この方法では適切に対応できなくなります。

索引の短所は次のとおりです。

- 索引を手動で作成するには多くの場合、データ・モデル、アプリケーションおよびデータの配布に関する深い知識が必要です。
- データが変更されると、索引に関して以前に行った決定を再度行う必要があります。索引が役立たなくなるか、新しい索引が必要となる可能性があります。
- 索引はディスク領域を占有します。
- 索引付きデータでDMLが発生した場合にデータベースで索引を更新する必要があり、これはパフォーマンス・オーバーヘッドとなります。

ノート:

Oracle Database 19c から、Oracle Database はアプリケーション・ワークロードを常に監視し、索引を自動的に作成および管理できます。自動索引付けは、固定間隔で実行されるデータベース・タスクとして実装されます。

次の状況において、索引を作成することを検討してください。

- 索引付けされた列を頻繁に問い合せ、表内の合計行数に対して割合の小さい行数が戻される場合。
- 索引付けされた列に参照整合性制約が存在する場合。索引は、全表ロックを回避する手段ですが、親表の主キーを更新したり、親表に主キーをマージしたり、親表から主キーを削除する場合は必須です。
- 表に一意キー制約を適用し、索引およびすべての索引オプションを手動で指定する場合。

関連項目:

- 自動索引付けについてさらに学習するには、『Oracle Database管理者ガイド』を参照してください

- 様々なエディションおよびサービスでサポートされる機能の詳細は、『Oracle Databaseライセンス情報ユーザー・マニュアル』参照してください

索引の使用可能性と可視性

索引は、使用可能(デフォルト)または使用不可、参照用(デフォルト)または非参照用にできます。

これらのプロパティは次のように定義されます。

- 使用可能性

  使用禁止索引はオプティマイザによって無視され、DML操作によって管理されません。使用禁止索引により、バルク・ロードのパフォーマンスが向上します。索引を削除し、後から再作成するかわりに、索引を使用禁止にし、後から再構築できます。使用禁止索引と索引パーティションは、領域を使用しません。使用可能索引を使用禁止にすると、データベースによりその索引セグメントが削除されます。

- 可視性

  非参照用索引はDML操作により管理されますが、デフォルトではオプティマイザにより使用されません。索引を非表示にするのは、使用禁止または削除の代替手段です。非参照用索引は、索引を削除する前に削除をテストする場合や、アプリケーション全体に影響を与えることなく一時的に索引を使用する場合に特に役立ちます。

関連項目:

オプティマイザによって実行計画が選択される方法について学習するには、「オプティマイザの概要」を参照

キーと列

キーとは、索引を作成できる列または式の集合です。

索引とキーという用語は同じ意味で使用されることがありますが、これらの用語には違いがあります。索引は、データベース内に格納される構造体であり、ユーザーはSQL文を使用して管理します。一方、キーは厳密に論理的な概念です。

次の文は、サンプル表oe.ordersのcustomer_id列に索引を作成します。

```
CREATE INDEX ord_customer_ix ON orders (customer_id);
```

この文では、customer_id列が索引キーです。索引自体はord_customer_ixという名前です。

ノート:

主キーと一意キーには自動的に索引が作成されますが、外部キーにも索引を作成できます。

関連項目:

- 「データ整合性」
- CREATE INDEXの構文およびセマンティクスについては、『Oracle Database SQL言語リファレンス』を参照

コンポジット索引

コンポジット索引(**連結索引**とも呼ばれる)は、表の中の複数の列に対して作成される索引です。

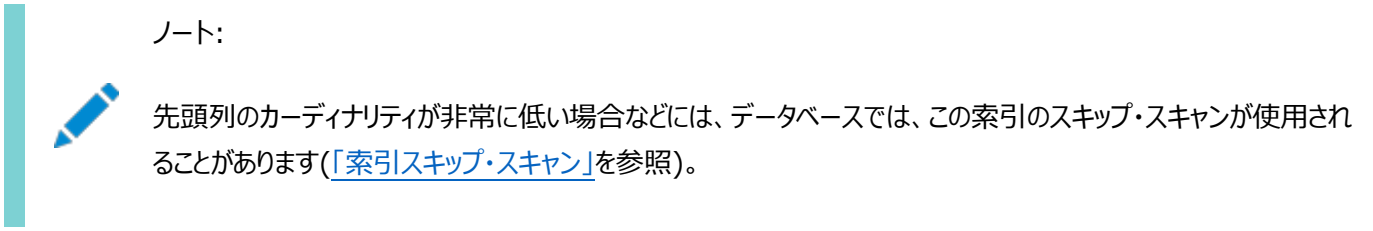
列をコンポジット索引で、データを取得する問合せに最も有用な順序に配置します。列は、表内で隣り合っている必要はありません。

SELECT文のWHERE句でコンポジット索引のすべての列または列の先頭部分を参照する場合には、コンポジット索引によってデータの検索速度を向上させることができます。そのため、定義で使用する列の順序は重要です。通常、最も頻繁にアクセスされる列が最初になります。

たとえば、アプリケーションが頻繁にemployees表のlast_name、job_idおよびsalary列への問合せを発行するとします。また、last_nameはカーディナリティが高く、表の行数に比べて個別値の数が多いとします。次のような列の順序で、索引を作成します。

```
CREATE INDEX employees_ix
   ON employees (last_name, job_id, salary);
```

3つの列すべて、last_name列のみ、またはlast_nameおよびjob_id列のみにアクセスする問合せは、この索引を使用します。この例では、last_name列にアクセスしない問合せは索引を使用しません。



ノート:

先頭列のカーディナリティが非常に低い場合などには、データベースでは、この索引のスキップ・スキャンが使用されることがあります(「索引スキップ・スキャン」を参照)。

次の条件のいずれかを満たす場合、複数の索引を同じ表で同じ列順に存在させることができます。

- 索引のタイプが異なります。

  たとえば、同じ列でビットマップ索引やBツリー索引を作成できます。

- 索引で異なるパーティション・スキーマが使用される。

  たとえば、ローカルでパーティション化された索引や、グローバルにパーティション化された索引を作成できます。

- 索引の一意性プロパティが異なります。

  たとえば、同じセットの列で一意と非一意の両方の索引を作成できます。

たとえば、パーティション化されていない索引、グローバル・パーティション索引、ローカルでパーティション化された索引を、同じ表の列のために同じ順序で存在させることができます。一度に表示できるのは、同じ順序で同数の列を持つ1つの索引のみです。

この機能により、既存の索引を削除し、別の属性で再作成する必要なく、アプリケーションを移行できます。また、索引キーが増え続け、新しいエントリが同じセットの索引ブロックに挿入されることになる場合、この機能はOLTPデータベースで役立ちます。このような「ホット・スポット」を軽減するには、パーティション化されていない索引からグローバル・パーティション索引に、索引を進化させることができます。

同じセットの列の索引が、タイプやパーティショニング・スキームで違いがない場合、これらの索引では異なる列順列を使用する必要があります。たとえば、次のSQL文では有効な列順列を指定しています。

```
CREATE INDEX employee_idx1 ON employees (last_name, job_id);
```

```
CREATE INDEX employee_idx2 ON employees (job_id, last_name);
```

関連項目:

同じ列のセットに複数の索引を作成する方法についてさらに学習するには、『Oracle Database SQLチューニング・ガイド』を参照

一意索引と非一意索引

索引には、一意索引と非一意索引の2種類があります。一意索引を使用すると、表の複数行のキー列に重複した値が入らないことが保証されます。

たとえば、アプリケーションでは、同じ従業員IDを持つ2人の従業員がいないことが必要になります。一意索引の場合、データ値ごとにrowidが1つ存在します。リーフ・ブロックのデータはキーのみによってソートされます。

非一意索引では、索引付けされた1つ以上の列に重複値が許可されます。たとえば、employees表のfirst_name列に、複数のMike値が含まれていてもかまいません。非一意索引の場合、ROWIDはソートされた順序でキーに含まれるため、非一意索引は索引キーとROWIDによってソートされます(昇順)。

Oracle Databaseでは、すべてのキー列がNULLの表の行には、索引は作成されません。ただし、ビットマップ索引の場合や、クラスタ・キーの列値がNULLの場合は例外です。

索引のタイプ

Oracle Databaseは、パフォーマンスの機能性を補足する複数の索引体系を提供しています。

Bツリー索引は標準索引タイプです。これらは高度に選択的な索引(各索引エントリに複数行が対応)および主キーに適しています。Bツリー索引を連結索引として使用すると、索引付けされた列でソートされたデータを取得できます。Bツリー索引には、次の表に示すサブタイプがあります。

表3-1 Bツリー索引のサブタイプ

Bツリー索引のサブタイプ	説明	さらに学習するには
索引構成表	索引構成表はデータ自体が索引であるため、ヒープ構成表とは異なります。	「索引構成表の概要」
逆キー索引	このタイプの索引では、索引キーのバイトの並びが逆になり、たとえば、103 は 301 として格納されます。バイトの並びを逆にすることにより、索引への挿入が多数のブロックに分散されます。	「逆キー索引」
降順索引	このタイプの索引では、特定の列(1 つまたは複数)のデータが降順で格納されます。	「昇順索引と降順索引」

Bツリー索引のサブタイプ	説明	さらに学習するには
B ツリークラスタ索引	このタイプの索引では、特定の列(1 つまたは複数)のデータが降順で格納されます。	「昇順索引と降順索引」

次の表に、Bツリー構造を使用しない索引のタイプを示します。

表3-2 Bツリー構造を使用しない索引

タイプ	説明	さらに学習するには
ビットマップ索引とビットマップ結合索引	ビットマップ索引では、索引エントリはビットマップを使用して複数の行を指します。対照的に、B ツリー索引エントリは単一の行を指します。ビットマップ結合索引は、2 つ以上の表を結合するためのビットマップ索引です。	「ビットマップ索引の概要」
ファンクションベース索引	このタイプの索引に含まれる列は、UPPER ファンクションなどのファンクションによって変換されるか、式に含まれます。B ツリー索引またはビットマップ索引はファンクション索引です。	「ファンクション索引の概要」
アプリケーション・ドメイン索引	ユーザーは、このタイプの索引をアプリケーション固有のドメインにあるデータ用に作成します。物理索引は従来の索引構造を使用する必要はなく、表として Oracle Database に格納することも、ファイルとして外部に格納することもできます。	「アプリケーション・ドメイン索引の概要」

関連項目:

- 索引の管理方法を学習するには、『Oracle Database管理者ガイド』を参照
- 様々な索引アクセス・パスについて学習するには、『Oracle Database SQLチューニング・ガイド』を参照

データベースで索引を管理する方法

作成された索引は、データベースによって自動的にメンテナンスされ、使用されます。

索引には、基礎となる表に対するデータ変更が自動的に反映されます。変更の例には、行の追加、更新および削除が含まれます。ユーザーの処理は不要です。

行が挿入されても、索引付きのデータ検索のパフォーマンスはほとんど一定です。ただし、表に対して数多くの索引が存在すると、データベースの索引も更新する必要があるため、DMLのパフォーマンスは低下します。

関連項目:

- 自動索引付けについてさらに学習するには、『Oracle Database管理者ガイド』を参照してください
- 様々なエディションおよびサービスでサポートされる機能の詳細は、『Oracle Databaseライセンス情報ユーザー・マ

ニュアル』参照してください

索引記憶域

Oracle Databaseでは、索引データは索引セグメントに格納されます

データ・ブロックで索引データのために使用できる領域は、データ・ブロック・サイズから、ブロック・オーバーヘッド、エントリ・オーバーヘッド、ROWID、および索引が作成される値1つ当たり1バイトの合計を引いたものです

索引セグメントの表領域は、所有者のデフォルト表領域またはCREATE INDEX文で明示的に指定された表領域です。管理を容易にするために、索引をその表とは別の表領域に格納できます。たとえば、索引のみを含む表領域は再構築できるため、これらの表領域をバックアップしないよう指定することによって、バックアップに必要な時間と記憶域を削減できます。

関連項目:

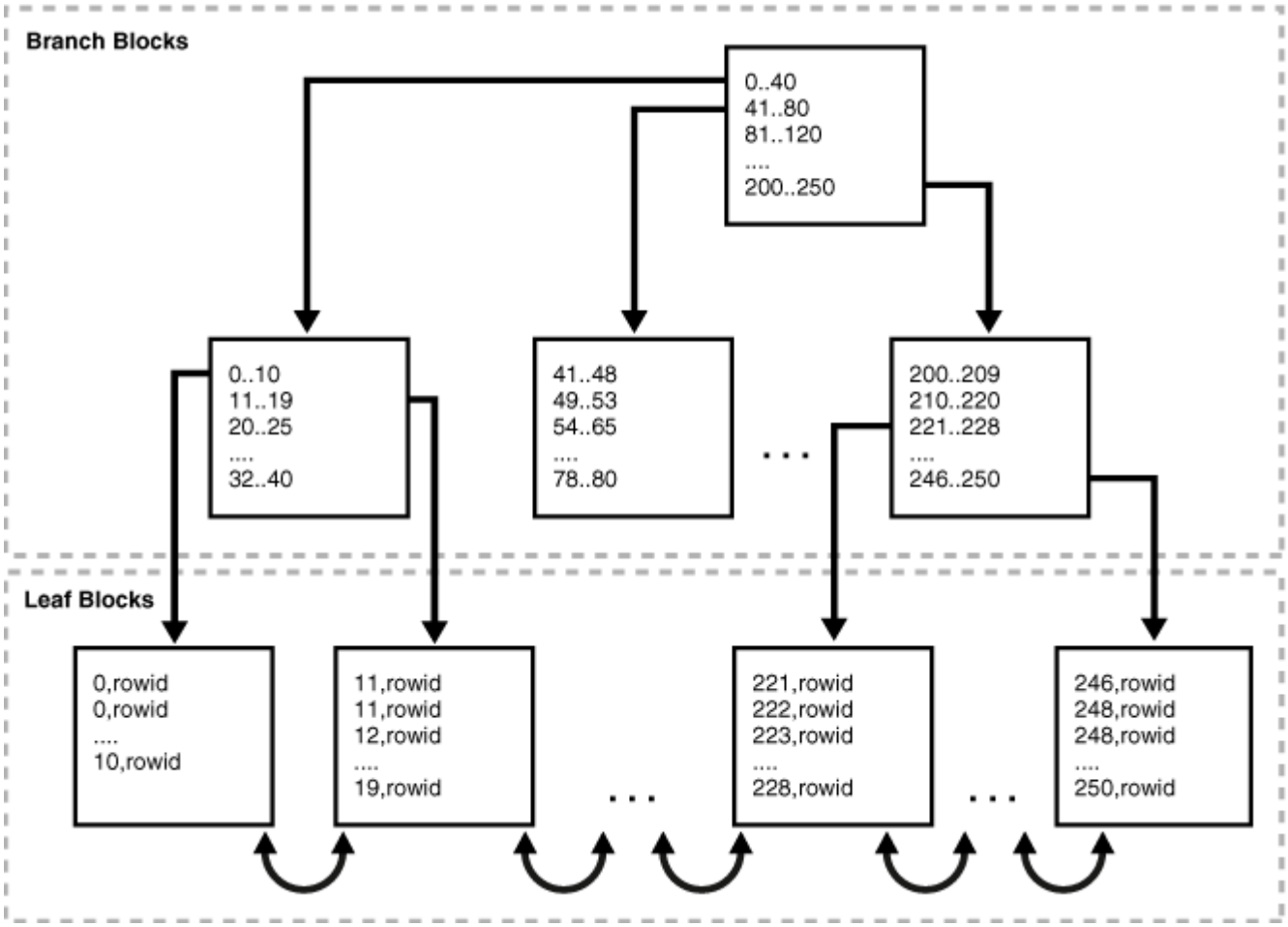
索引ブロックのタイプ(ルート、ブランチおよびリーフ)と、ブロック内に索引エントリが格納される方法について学習するには、「索引ブロックの概要」を参照

Bツリー索引の概要

Bツリー(バランス・ツリーの略)は、最も一般的なタイプのデータベース索引です。**Bツリー索引**は、複数の範囲に分割された順序付きの値リストです。Bツリーは、キーを行または行の範囲と関連付けることによって、完全一致や範囲検索など、広範囲の問合せに対して優れた検索パフォーマンスを提供します。

次の図に、Bツリー索引の構造を示します。この例では、employees表の外部キー列であるdepartment_id列の索引を示しています。

図3-1 Bツリー索引の内部構造



この項では、次の項目について説明します。

- ブランチ・ブロックとリーフ・ブロック
- 索引スキャン
- 逆キー索引
- 昇順索引と降順索引
- 索引の圧縮

ブランチ・ブロックとリーフ・ブロック

Bツリー索引には、検索用の**ブランチ・ブロック**と、キー値のソート用の**リーフ・ブロック**の2つのタイプのブロックがあります。Bツリー索引の上位ブランチ・ブロックには、下位レベルの索引ブロックを指す索引データが含まれます。

図3-1では、ルート・ブランチ・ブロックに0–40のエントリがあり、次のブランチ・レベルの左端のブロックを指しています。このブランチ・ブロックには、0–10や11–19などのエントリがあります。これらの各エントリは、それぞれの範囲に該当するキー値を含むリーフ・ブロックを指します。

すべてのリーフ・ブロックは自動的に同じ深さになるため、Bツリー索引はバランスが保たれます。したがって、索引のどの位置からでも、レコード検索にかかる時間はほぼ同じになります。索引の高さは、ルート・ブロックからリーフ・ブロックに達するまでに必要なブロックの数です。ブランチ・レベルは高さから1を引いた数です。図3-1では、索引の高さは3、ブランチ・レベルは2です。

ブランチ・ブロックには、2つのキーの分岐を決定する際に必要な、最小のキー接頭辞が格納されます。このテクニックを使用すると、データベースでは各ブランチ・ブロックにできるかぎり多くのデータを収納できます。ブランチ・ブロックには、キーを含む子ブロックへのポインタが含まれます。キーおよびポインタの数は、ブロックのサイズによって制限されます。

リーフ・ブロックには、すべての索引付きデータ値と、実際の行を検索するための対応するROWIDが含まれています。各エントリは(キー、ROWID)によってソートされます。リーフ・ブロック内では、キーとROWIDは兄弟関係にある左右のエントリにリンクされ

ます。リーフ・ブロック自体も二重にリンクされます。図3-1では、左端のリーフ・ブロック(0-10)が2番目のリーフ・ブロック(11-19)にリンクされます。

ノート:

文字データを持つ列の索引は、データベース文字セットにおける文字のバイナリ値を基盤にしています。

索引スキャン

索引スキャンの場合、データベースでは文に指定された索引付けされた列値を使用し、索引を読み込むことによって行を検索します。データベースでは、索引をスキャンして値を検索する場合、*n*回のI/Oでこの値を見つけます(*n*は、Bツリー索引の高さ)。これがOracle Database索引の基本原理です。

SQL文が索引付けされた列のみにアクセスする場合、データベースでは表ではなく索引から値を直接読み取ります。文が索引付けされた列に加えて索引付けされていない列にもアクセスする場合、データベースではROWIDを使用して表の中の行を検索します。通常、データベースでは索引ブロックと表ブロックを交互に読み取って表データを取得します。

関連項目:

索引スキャンの詳細は、『Oracle Database SQLチューニング・ガイド』を参照

全索引スキャン

全索引スキャンの場合、データベースでは索引全体を順に読み取ります。全索引スキャンは、SQL文の**述語**(WHERE句)が索引の列を参照している場合、および述語が指定されていない場合に使用できます。全スキャンでは、データが索引キーによって順序付けられるため、ソートが不要になります。

例3-1 全索引スキャン

アプリケーションにより、次の問合せが実行されるとします。

```
SELECT department_id, last_name, salary
FROM   employees
WHERE  salary > 5000
ORDER BY department_id, last_name;
```

この例では、department_id、last_nameおよびsalaryが索引のコンポジット・キーです。Oracle Databaseでは、ソートされた順序(部門IDおよび姓の順序)で索引を読み取り、salary属性でフィルタ処理して索引の全スキャンを実行します。これにより、データベースでは、問合せに含まれている列より多くの列を含むemployees表よりも小さいデータ・セットがスキャンされ、データのソートが不要になります。

全スキャンによって次のような索引エントリが読み取られます。

```
50,Atkinson,2800,rowid
60,Austin,4800,rowid
70,Baer,10000,rowid
80,Abel,11000,rowid
80,Ande,6400,rowid
110,Austin,7200,rowid
.
```

```
.
.
```

高速全索引スキャン

高速全索引スキャンは、データベースが特定の順序に従わずに索引ブロックを読み取る全索引スキャンで、表にアクセスすることなく、索引自体の中のデータにアクセスします。

高速全索引スキャンは、次に示す2つの条件が満たされている場合に全表スキャンの代替となるスキャンです。

- 問合せに必要なすべての列が索引に含まれていること。
- すべてNULLの行が問合せの結果セットに現れないこと。この結果を保証するためには、索引内の少なくとも1つの列に次のいずれかが存在する必要があります。
    - NOT NULL制約。
    - 問合せの結果セットにNULLが含まれないように列に適用された述語

例3-2 高速全索引スキャン

アプリケーションにより、ORDER BY句を含まない次の問合せが発行されるとします。

```
SELECT last_name, salary
FROM   employees;
```

last_name列にはNOT NULL制約があります。姓と給与が索引のコンポジット・キーである場合、高速全索引スキャンでは索引エントリを読み取って、要求された情報を取得できます。

```
Baida,2900,rowid
Atkinson,2800,rowid
Zlotkey,10500,rowid
Austin,7200,rowid
Baer,10000,rowid
Austin,4800,rowid
.
.
.
```

索引レンジ・スキャン

索引レンジ・スキャンは、条件に索引の1つ以上の先頭列が指定された索引の順序付きスキャンで、索引キーには0、1またはそれ以上の値を指定できます。

条件は、1つ以上の式および論理(ブール)演算子の組合せを指定したものです。これにより、TRUE、FALSEまたはUNKNOWNの値が戻されます。

データベースでは一般的に、索引レンジ・スキャンを選択的なデータへのアクセスに使用します。選択性は、問合せで選択される表の中の行の割合で、0は行がないことを表し、1はすべての行を表します。選択性は、問合せの述語(WHERE last_name LIKE 'A%'など)や述語の組合せに関連します。述語は、値が0に近付くほど選択性が高くなり、値が1に近付くほど選択性が低く(非選択性が高く)なります。

たとえば、姓がAで始まる従業員を問い合せるとします。last_name列に索引が作成されており、次のエントリがあるとします。

```
Abel,rowid
Ande,rowid
Atkinson,rowid
Austin,rowid
Austin,rowid
```

```
Baer,rowid
.
.
.
```

last_name列が述語に指定されており、それぞれの索引キーに複数のROWIDが考えられるため、データベースではレンジ・スキャンを使用できます。たとえば、Austinという姓の従業員が2人いた場合、2つのROWIDがキーAustinに関連付けられます。

索引レンジ・スキャンは、10から40のIDを持つ部門の問合せのように、両側に境界がある場合と、40より大きいIDの問合せのように片側のみに境界がある場合があります。索引をスキャンするには、データベースではリーフ・ブロックを後方または前方に移動します。たとえば、10から40のIDを求めるスキャンでは、最小キー値10以上を含む最初の索引リーフ・ブロックが検索されます。次に、40より大きい値が見つかるまで、スキャンはリンクされたリーフ・ノード・リストを水平に進みます。

索引の一意スキャン

索引レンジ・スキャンとは対照的に、**索引の一意スキャン**では、索引キーに関連付けられたROWIDは0個または1個である必要があります。

データベースでは、述語が等価演算子を使用して一意索引のキーのすべての列を参照する場合に、一意スキャンが実行されます。索引の一意スキャンでは、2つ目のレコードがないことがわかっているため、最初のレコードが見つかるとただちに処理が停止します。

例として、ユーザーが次の問合せを実行するとします。

```
SELECT *
FROM   employees
WHERE  employee_id = 5;
```

employee_id列が主キーで、次のようなエントリで索引が作成されているとします。

```
1,rowid
2,rowid
4,rowid
5,rowid
6,rowid
.
.
.
```

この場合、データベースでは、索引の一意スキャンを使用してIDが5の従業員のROWIDが検索されます。

索引スキップ・スキャン

索引スキップ・スキャンでは、コンポジット索引の論理副索引を使用します。個別の索引を検索する場合と同様に、データベースでは単一の索引をスキップします。

コンポジット索引の先頭列に含まれる個別値が少数であり、索引の先頭以外のキーに多数の個別値がある場合は、スキップ・スキャンが有効です。コンポジット索引の先頭列が問合せの述語に指定されていない場合、データベースでは、索引スキップ・スキャンを選択できます。

例3-3 コンポジット索引のスキップ・スキャン

sh.customers表の顧客に対して次の問合せを実行するとします。

```
SELECT * FROM sh.customers WHERE cust_email = 'Abbey@company.example.com';
```

customers表にはcust_genderという列があり、この列の値はMまたはFです。列(cust_gender、cust_email)にコンポジット

索引が存在するとします。次の例は、索引エントリの一部を示しています。

```
F,Wolf@company.example.com,rowid
F,Wolsey@company.example.com,rowid
F,Wood@company.example.com,rowid
F,Woodman@company.example.com,rowid
F,Yang@company.example.com,rowid
F,Zimmerman@company.example.com,rowid
M,Abbassi@company.example.com,rowid
M,Abbey@company.example.com,rowid
```

cust_genderはWHERE句に指定されていませんが、データベースでは、この索引のスキップ・スキャンを使用できます。

スキップ・スキャンでは、論理副索引の数は、先頭列の個別値の数によって決定されます。前述の例では、先頭列に2つの可能な値があります。データベースでは、索引を、キーFを持つ1つの副索引と、キーMを持つ2つ目の副索引に論理的に分割します。

電子メールがAbbey@company.example.comの顧客のレコードを検索する場合、最初に値Fを持つ副索引が検索され、次に値Mを持つ副索引が検索されます。その結果、データベースでは問合せが次のように処理されます。

```
SELECT * FROM sh.customers WHERE cust_gender = 'F'
  AND cust_email = 'Abbey@company.example.com'
UNION ALL
SELECT * FROM sh.customers WHERE cust_gender = 'M'
  AND cust_email = 'Abbey@company.example.com';
```

関連項目:

スキップ・スキャンについてさらに学習するには、『Oracle Database SQLチューニング・ガイド』を参照

索引クラスタ化係数

索引クラスタ化係数は、姓などの索引付きの値を対象に、行の順序性を測定した指標です。順序性がよいほど、クラスタ化係数は小さくなります。

クラスタ化係数は、索引を使用した表全体の読取りに必要な入出力回数のおおよその目安として使用できます。

- クラスタ化係数が高いほど、大規模な索引のレンジ・スキャン実行時の入出力回数が増えます。索引エントリはランダムな表ブロックを指しているため、データベースは索引が示すデータを取得するために、同じブロックの読取りが何回も必要になることがあります。
- クラスタ化係数が低いほど、大規模な索引のレンジ・スキャン実行時の入出力回数が減ります。同じ範囲の中にある索引キーは同じブロックを指していることが多いため、データベースは同じブロックを何度も読み取る必要がありません。

クラスタ化係数は、索引スキャンと深い関連性があります。これは、クラスタ係数で次の事項を判断できるからです。

- データベースが大規模なレンジ・スキャンに索引を使用するかどうか
- 索引キーに対する表の編成の度合い
- 索引キーに従って行を順序付ける必要がある場合に、索引構成表、パーティション化、表クラスタのいずれを使用するのがよいか

例3-4 クラスタ化係数

employees表が2つのデータ・ブロックに分かれているとします。表3-3は、2つのデータ・ブロック内の行を示しています(省略記号

は省略したデータを示します)。

表3-3 Employees表内にある2つのデータ・ブロックの内容

データ・ブロック1	データ・ブロック2
100 Steven **King** SKING ... 156 Janette **King** JKING ... 115 Alexander **Khoo** AKHOO ... . . . 116 Shelli **Baida** SBAIDA ... 204 Hermann **Baer** HBAER ... 105 David **Austin** DAUSTIN ... 130 Mozhe **Atkinson** MATKINSO ... 166 Sundar **Ande** SANDE ... 174 Ellen **Abel** EABEL ...	149 Eleni **Zlotkey** EZLOTKEY ... 200 Jennifer **Whalen** JWHALEN ... . . . 137 Renske **Ladwig** RLADWIG ... 173 Sundita **Kumar** SKUMAR ... 101 Neena **Kochar** NKOCHHAR ...

行は姓(太字の部分)の順序に従ってブロックに格納されています。たとえば、データ・ブロック1は、最後の行から先頭に向ってAbel、Andeと続き、先頭のSteven Kingまでアルファベット順に並んでいます。データ・ブロック2は、最後の行から先頭に向ってKochar、Kumarと続き、先頭のZlotkeyまでアルファベット順に並んでいます。

姓の列に索引が付けられているとします。それぞれの名前エントリが、ROWIDに相当します。索引エントリは概念的には次のようになります。

```
Abel,block1row1
Ande,block1row2
Atkinson,block1row3
Austin,block1row4
Baer,block1row5
.
.
.
```

社員IDの列に別の索引が付けられているとします。索引エントリは概念的には次のようになり、社員IDは、2つのブロック全体にわたり、ほぼランダムな場所に分散しています。

```
100,block1row50
101,block2row1
102,block1row9
103,block2row19
104,block2row39
105,block1row4
.
.
.
```

次の文では、ALL_INDEXESビューを問い合せて、次の2つの索引のクラスタ化係数を検索します。

```
SQL> SELECT INDEX_NAME, CLUSTERING_FACTOR
  2  FROM ALL_INDEXES
  3  WHERE INDEX_NAME IN ('EMP_NAME_IX','EMP_EMP_ID_PK');

INDEX_NAME           CLUSTERING_FACTOR
-------------------- -----------------
EMP_EMP_ID_PK                       19
EMP_NAME_IX                          2
```

EMP_NAME_IXのクラスタ化係数は低く、同じリーフ・ブロック内の隣接する索引エントリは、同じデータ・ブロック内の行を指す傾向があります。EMP_EMP_ID_PKのクラスタ化係数は高く、同じリーフ・ブロック内の隣接する索引エントリが、同じデータ・ブロック内の行を指すことはほとんどありません。

関連項目:

ALL_INDEXESについて学習するには、『Oracle Databaseリファレンス』を参照してください。

逆キー索引

逆キー索引はBツリー索引の一種であり、列の順序は保ちながら、各索引キーのバイトを物理的に逆にします。

たとえば、索引キーが20であり、このキーに対して標準Bツリー索引で格納される16進数の2バイトがC1, 15である場合、逆キー索引では、バイトが15, C1として格納されます。

キーを逆にすると、Bツリー索引の右側にあるリーフ・ブロックの競合の問題が解決されます。この問題は、複数のインスタンスが同じブロックを繰り返し変更するOracle Real Application Clusters(Oracle RAC)データベースで特に深刻です。たとえば、orders表では、オーダーの主キーは順次キーです。クラスタ内の1つのインスタンスが注文20を追加し、別のインスタンスがオーダー21を追加し、各インスタンスが索引の右側にある同じリーフ・ブロックにキーを書き込みます。

逆キー索引では、バイト順を逆にすることにより、挿入が索引のすべてのリーフ・キー全体に分散されます。たとえば、標準キー索引では隣接する20や21などのキーは、離れた別々のブロックに格納されます。したがって、順次キーの挿入のためのI/Oは、より等分に分散されます。

索引のデータは格納されるときに列キーでソートされないため、逆キー配列を使用すると、場合によっては索引レンジ・スキャン問合せを実行できなくなります。たとえば、20より大きいオーダーIDを求める問合せを発行した場合、データベースではこのIDを含むブロックから開始できず、リーフ・ブロックが水平に処理されます。

昇順索引と降順索引

昇順索引では、Oracle Databaseのデータは昇順に格納されます。デフォルトでは、文字データは値の各バイトに含まれるバイナリ値に従って順序付けられ、数値データは最小の数から最大の数へ、日付は古い値から新しい値への順になります。

昇順索引の例として、次のSQL文を考えてみます。

```
CREATE INDEX emp_deptid_ix ON hr.employees(department_id);
```

Oracle Databaseでは、hr.employees表をdepartment_id列でソートします。department_idおよび対応するROWID値を、0から始まる昇順で昇順索引にロードします。索引を使用するとき、Oracle Databaseでは、ソートしたdepartment_id値が検索され、関連付けられたROWIDを使用して要求されたdepartment_id値を持つ行が検索されます。

CREATE INDEX文にDESCキーワードを指定すると、降順索引を作成できます。この場合、索引は指定された列(1つまたは複数)にデータを降順に格納します。表3-3のemployees.department_id列の索引が降順である場合、250を含むリーフ・ブロックがツリーの左側となり、0のブロックが右側になります。降順索引でのデフォルトの検索は、最大値から最小値への順です。

降順索引は、問合せで一部の列を昇順、他の列を降順でソートする場合に有効です。たとえば、last_name列およびdepartment_id列に対して、次のようにコンポジット索引を作成したとします。

```
CREATE INDEX emp_name_dpt_ix ON hr.employees(last_name ASC, department_id DESC);
```

ユーザーがhr.employeesに対して、昇順(AからZへ)の姓と降順(大きいIDから小さいIDへ)の部門IDを問い合せる場合、データベースではこの索引を使用してデータを検索するため、データをソートする余分なステップが不要になります。

関連項目:

- 昇順および降順の索引検索についてさらに学習するには、Oracle Database SQLチューニング・ガイドを参照してく

ださい。

- CREATE INDEXのASCおよびDESCオプションの詳細は、『Oracle Database SQL言語リファレンス』を参照してください

索引の圧縮

索引の領域を削減するために、Oracle Databaseでは様々な圧縮アルゴリズムを使用できます。

接頭辞圧縮

Oracle Databaseでは、**接頭辞圧縮(キー圧縮**とも呼ばれます)を使用して、Bツリー索引または索引構成表の主キー列値の一部を圧縮できます。接頭辞圧縮によって、索引に使用される領域が大幅に減少します。

圧縮されていない索引エントリには、1つの要素があります。接頭辞圧縮を使用する索引エントリには、グループ化要素である接頭辞エントリと、一意要素またはほぼ一意要素である接尾辞エントリの2つの要素があります。データベースでは、圧縮するために、接頭辞エントリが索引ブロック内の接尾辞エントリ間で共有されます。

ノート:

一意要素を持つようにキーを定義しなければ、データベースによってグループ化要素に ROWID が追加され、一意要素が提供されます。

デフォルトでは、一意索引の接頭辞は最後のキー列を除くすべてのキー列で構成され、非一意索引の接頭辞はすべてのキー列で構成されます。次のように、oe.orders表の2つの列に、コンポジット、一意索引を作成するとします。

```
CREATE UNIQUE INDEX orders_mod_stat_ix ON orders ( order_mode, order_status );
```

前述の例では、索引キーはonline,0です。ROWIDはエントリのキー・データ部分に格納され、キー自体の一部ではありません。

ノート:

1 つの列に一意索引を作成した場合、共通の接頭辞が存在しないため、Oracle Database では接頭辞キー圧縮を使用できません。

一方、同じ列に非一意索引を作成するとします。

```
CREATE INDEX orders_mod_stat_ix ON orders ( order_mode, order_status );
```

また、order_mode列およびorder_status列で繰返し値が発生するとします。索引ブロックに、次の例に示すエントリがあるとします。

```
online,0,AAAPvCAAFAAAAFaAAa
online,0,AAAPvCAAFAAAAFaAAg
online,0,AAAPvCAAFAAAAFaAAl
online,2,AAAPvCAAFAAAAFaAAm
online,3,AAAPvCAAFAAAAFaAAq
online,3,AAAPvCAAFAAAAFaAAt
```

前述の例では、キー接頭辞は、online,0と同様にorder_modeとorder_statusの値を連結した値で構成されます。AAAPvCAAFAAAAFaAAaと同様に、接尾辞はROWIDにあります。ROWID自体がデータベース内で一意であるため、ROWIDによって索引エントリ全体が一意になります。

前述の例の索引がデフォルトの接頭辞圧縮によって作成された場合(COMPRESSキーワードで指定された)、online,0やonline,3のような重複するキー接頭辞が圧縮されます。その結果、データベースでは次のように圧縮されます。

```
online,0
AAAPvCAAFAAAAFaAAa
AAAPvCAAFAAAAFaAAg
AAAPvCAAFAAAAFaAAl
online,2
AAAPvCAAFAAAAFaAAm
online,3
AAAPvCAAFAAAAFaAAq
AAAPvCAAFAAAAFaAAt
```

接尾辞エントリ(ROWID)は、索引行の圧縮版を形成します。各接尾辞エントリは、接尾辞と同じ索引ブロックに格納されている接頭辞エントリを参照します。

別の方法として、接頭辞圧縮を使用する索引を作成するときに接頭辞の長さを指定できます。たとえば、COMPRESS 1と指定した場合、接頭辞はorder_modeとなり、接尾辞はorder_status,rowidとなります。索引ブロックの例の値では、索引によって接頭辞onlineの重複した発生が除外されており、これは、概念上次のように表すことができます。

```
online
0,AAAPvCAAFAAAAFaAAa
0,AAAPvCAAFAAAAFaAAg
0,AAAPvCAAFAAAAFaAAl
2,AAAPvCAAFAAAAFaAAm
3,AAAPvCAAFAAAAFaAAq
3,AAAPvCAAFAAAAFaAAt
```

索引では、特定の接頭辞がリーフ・ブロックごとに1回のみ格納されます。圧縮されるのは、Bツリー索引のリーフ・ブロックのキーのみです。ブランチ・ブロックの場合、キーの接尾辞は切り捨てることができますが、キーは圧縮されません。

関連項目:

- 圧縮型索引の使用方法を学習するには、『*Oracle Database*管理者ガイド』を参照してください。
- パーティション索引で接頭辞圧縮を使用する方法を学習するには、『*Oracle Database VLB*およびパーティショニング・ガイド』を参照してください。
- CREATE INDEXの*key_compression*句の詳細は、『*Oracle Database SQL*言語リファレンス』を参照してください

拡張索引圧縮

Oracle Database 12cリリース1 (12.1.0.2)以降、**拡張圧縮索引**により、ヒープ構成表でサポートされる索引の従来の接頭辞圧縮が改善されました。

拡張索引圧縮の利点

接頭辞圧縮は、サポートされる索引のタイプ、圧縮率および使い勝手に関して制限があります。すべてのブロックに固定重複キー排除を使用する接頭辞圧縮とは異なり、拡張索引圧縮ではブロックごとに適応重複キー排除を使用します。拡張索引圧縮の主な利点は次のとおりです。

- データベースは、列内レベル接頭辞、重複キー排除、ROWID圧縮などのアルゴリズムを使用して、各ブロックの最適な圧縮を自動的に選択します。接頭辞圧縮とは異なり、拡張索引圧縮ではユーザーがデータの特性を知る必要はありません。
- 高度圧縮は、非一意索引と一意索引の両方で動作します。接頭辞圧縮は一部の非一意索引で適切に動作しますが、先頭列に繰り返しが少ない索引では圧縮率が低くなります。
- 圧縮された索引は、圧縮されていない索引と同じように使用できます。索引は、同じアクセス・パス(一意キー検索、レンジ・スキャン、高速全スキャン)をサポートします。
- 索引は、親表または格納されている表領域から拡張圧縮を継承できます。

拡張索引圧縮の動作

拡張索引圧縮は、ブロック・レベルに作用し、各ブロックを最適に圧縮できます。データベースでは次の方法が使用されます。

- 索引作成時に、リーフ・ブロックが一杯になると、データベースでは自動的にブロックを最適レベルに圧縮します。
- DMLの結果として索引ブロックを再編成する際に、データベースで受信索引エントリのための十分な領域を作成できる場合、ブロック分割は行われません。ただし、拡張索引圧縮を使用しない場合、DML中にブロックが一杯になると、索引ブロックの分割が常に行われます。

拡張索引圧縮HIGH

Oracle Database 12cリリース2 (12.2)より前のリリースでは、拡張索引圧縮の唯一のフォームは低圧縮(COMPRESS ADVANCED LOW)でした。現在は高圧縮(COMPRESS ADVANCED HIGH)も指定できるようになり、これはデフォルトでもあります。HIGHオプションによる拡張索引圧縮には、次の利点があります。

- ほとんどの場合、圧縮率が高くなり、索引にアクセスする問合せのパフォーマンスも向上します
- 拡張低より複雑な圧縮アルゴリズムを採用します
- 特殊なディスク上形式である圧縮単位でデータを格納します

ノート:

HIGH 圧縮を適用すると、すべてのブロックが圧縮されます。LOW 圧縮を適用すると、データベースでは一部のブロックが非圧縮のままになることがあります。どの程度のブロックが非圧縮のままになっているかは、統計を使用して特定できます。

例3-5 拡張高圧縮を使用した索引の作成

この例はhr.employees表の索引に対する拡張索引圧縮を使用可能にします。

```
CREATE INDEX hr.emp_mndp_ix
  ON hr.employees(manager_id, department_id)
  COMPRESS ADVANCED;
```

次の問合せにより、圧縮のタイプが示されます。

```
SELECT COMPRESSION FROM DBA_INDEXES WHERE INDEX_NAME ='EMP_MNDP_IX';

COMPRESSION
-------------
ADVANCED HIGH
```

関連項目:

- 拡張索引圧縮を有効化する方法を学習するには、Oracle Database管理者ガイドを参照してください。
- CREATE INDEXの*key_compression*句の詳細は、Oracle Database SQL言語リファレンスを参照してください。
- ALL_INDEXESについて学習するには、『Oracle Databaseリファレンス』を参照してください。

ビットマップ索引の概要

ビットマップ索引では、各索引キーのビットマップがデータベースに格納されます。従来型のBツリー索引では、1つの索引エントリは単一の行を指します。ビットマップ索引では、各索引キーに複数の行へのポインタが格納されます。

ビットマップ索引は、主にデータ・ウェアハウス用または問合せが多数の列を非定型方式で参照する環境用に設計されています。ビットマップ索引が適している状況として、次のような状況があります。

- 索引付けされた列は、カーディナリティが低く、表の行数に比べて個別値の数が少なくなります。
- 索引付けされた表は、読取り専用であるか、DML文による重要な変更の対象外です。

データ・ウェアハウスの例として、sh.customer表にcust_gender列があるとすると、この列に指定される値は、MとFの2つのみです。特定の性別の顧客の数に対する問合せが一般的だとします。このような場合、customer.cust_gender列はビットマップ索引の候補になります。

ビットマップ内の各ビットは、使用可能なROWIDに対応しています。ビットが設定されている場合は、対応するROWIDを持つ行にはキー値が含まれます。マッピング機能がビット位置を実際のROWIDに変換するため、別の内部表現が使用されていても、ビットマップ索引はBツリー索引と同じ機能を提供します。

単一の行の索引付けされた列が更新された場合、データベースによって、更新された行にマッピングされた個別のビットではなく索引キー・エントリ(たとえば、MまたはF)がロックされます。キーは多数の行を指すため、索引付きデータに対するDMLは、通常、これらの行をすべてロックします。このため、ビットマップ索引は多くのOLTPアプリケーションに適していません。

関連項目:

- ビットマップ索引についてさらに学習するには、『Oracle Database SQLチューニング・ガイド』を参照
- データ・ウェアハウスでビットマップ索引を使用する方法を学習するには、『Oracle Databaseデータ・ウェアハウス・ガイド』を参照

例: 単一表でのビットマップ索引

次の例では、sh.customers表のいくつかの列がビットマップ索引の候補になっています。

次の問合せについて考えてみます。

```
SQL> SELECT cust_id, cust_last_name, cust_marital_status, cust_gender
  2  FROM   sh.customers
  3  WHERE  ROWNUM < 8 ORDER BY cust_id;

   CUST_ID CUST_LAST_ CUST_MAR C
---------- ---------- -------- -
         1 Kessel              M
         2 Koch                F
         3 Emmerson            M
```

```
         4 Hardy               M
         5 Gowen               M
         6 Charles    single   F
         7 Ingram     single   F

7 rows selected.
```

cust_marital_statusおよびcust_gender列のカーディナリティは低く、cust_idおよびcust_last_nameのカーディナリティは低くありません。このため、ビットマップ索引はcust_marital_statusおよびcust_genderに適切といえます。その他の列には、ビットマップ索引は効果がありません。この場合は、一意のBツリー索引を使用する方が、表示と検索が効率的になります。

表3-4に、前述の例に示したcust_gender列の出力に対するビットマップ索引を示します。これは、各性別に対応する2つのビットマップからなります。

表3-4 1つの列のサンプル・ビットマップ

値	行1	行2	行3	行4	行5	行6	行7
M	1	0	1	1	1	0	0
F	0	1	0	0	0	1	1

マッピング関数はビットマップの各ビットをcustomers表のROWIDに変換します。各ビットの値は、表内の対応する行の値に依存しています。たとえば、customers表の最初の行で性別がMであるため、M値のビットマップには、最初のビットとして1が含まれます。ビットマップcust_gender='M'の行2、6、および7のビットは、これらの行が値としてMを含まないため、0になります。

ノート:

B ツリー索引とは異なり、ビットマップ索引は、完全に NULL 値からなるキーを含む場合があります。NULL の索引作成は、集計関数 COUNT による問合せなど、一部の SQL 文に使用できます。

顧客の人口統計を調べているアナリストが、女性顧客のうち独身者または離婚者はどれくらいいるかを尋ねてきたとします。この質問は、次のSQL問合せで表現できます。

```
SELECT COUNT(*)
FROM   customers
WHERE  cust_gender = 'F'
AND    cust_marital_status IN ('single', 'divorced');
```

ビットマップ索引を使用すると、表3-5に示すように、結果のビットマップから1の値の数を数え、この問合せを効率よく処理できます。基準に該当する顧客を識別するときは、Oracle Databaseでは結果のビットマップを使用して表にアクセスできます。

表3-5 2つの列のサンプル・ビットマップ

値	行1	行2	行3	行4	行5	行6	行7
M	1	0	1	1	1	0	0
F	0	1	0	0	0	1	1

値	行1	行2	行3	行4	行5	行6	行7
single	0	0	0	0	0	1	1
divorced	0	0	0	0	0	0	0
single または divorced、および F	0	0	0	0	0	1	1

ビットマップ索引は、WHERE句に指定されたいくつかの条件に対応する索引を効率的にマージします。すべての条件ではなく一部の条件のみを満たす行は、表自体がアクセスされる前に除外されます。これによってレスポンス時間が短縮されます(多くの場合は大幅に改善されます)。

ビットマップ結合索引

ビットマップ結合索引は、2つ以上の表を**結合**するためのビットマップ索引です。

索引は、表列の値ごとに、対応する行のROWIDを索引付きの表に格納します。対照的に、標準ビットマップ索引は単一表に対して作成されます。

ビットマップ結合索引は、制限を事前に実行するため、結合するデータのボリュームを削減する効率的な方法です。ビットマップ結合索引が役に立つ場合の例として、ユーザーが特定の職種の従業員数を頻繁に問い合せるとします。一般的な問合せは次のようになります。

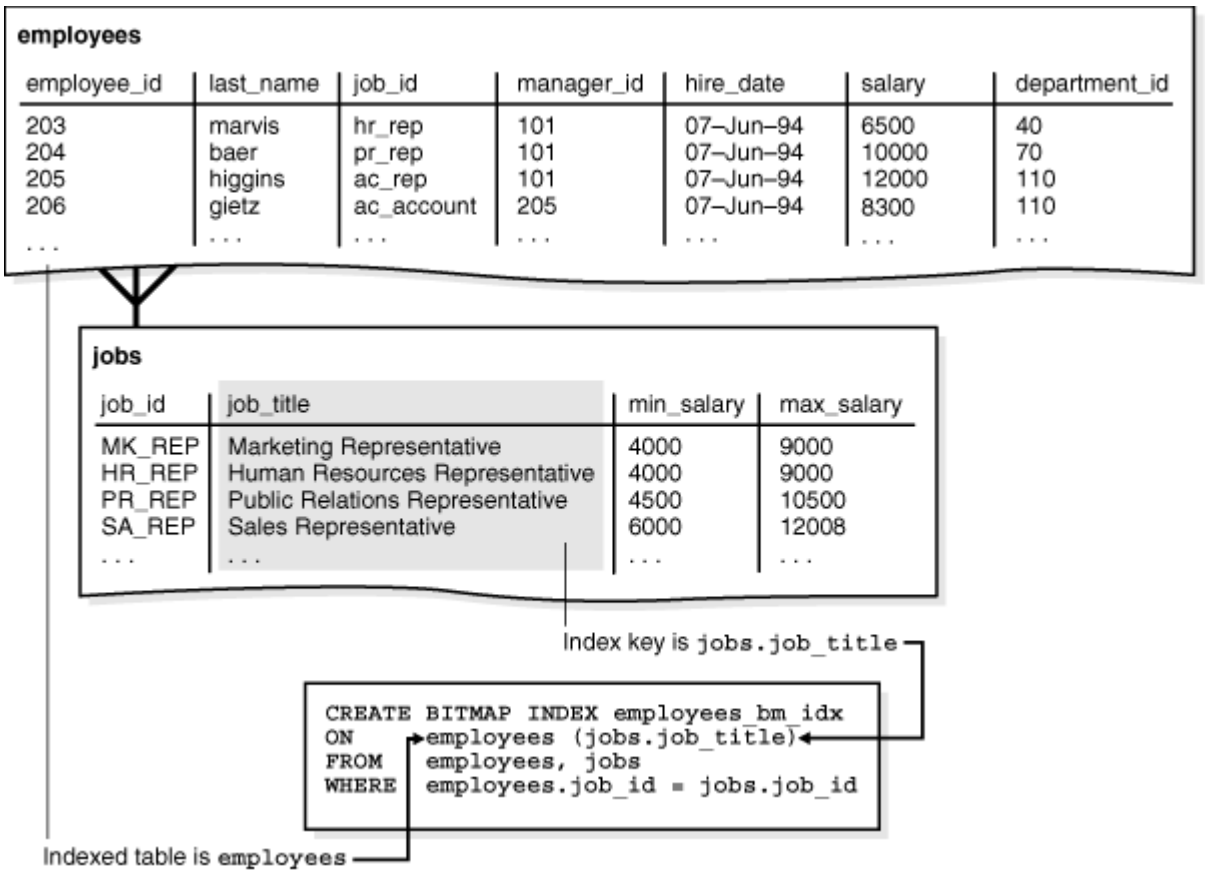
```
SELECT COUNT(*)
FROM   employees, jobs
WHERE  employees.job_id = jobs.job_id
AND    jobs.job_title = 'Accountant';
```

この問合せでは、通常、jobs.job_titleの索引を使用してAccountantの行を検索し、次にjob_idの行を検索し、employees.job_idの索引を使用して一致する行を検索します。表のスキャンではなく索引自体からデータを取り出すには、次のようにビットマップ結合索引を作成できます。

```
CREATE BITMAP INDEX employees_bm_idx
ON     employees (jobs.job_title)
FROM   employees, jobs
WHERE  employees.job_id = jobs.job_id;
```

次の図に示すように、索引キーはjobs.job_titleであり、索引付きの表はemployeesです。

図3-2 ビットマップ結合索引



概念的には、次の問合せ(サンプル出力を示しています)に示すSQL問合せのjobs.title列の索引はemployees_bm_idxです。索引のjob_titleキーは、employees表の行を指します。経理担当者の数を求める問合せでは、索引を使用すると、索引自体に要求された情報が含まれるため、employeesおよびjobs表へのアクセスを回避できます。

```
SELECT jobs.job_title AS "jobs.job_title", employees.rowid AS "employees.rowid"
FROM   employees, jobs
WHERE  employees.job_id = jobs.job_id
ORDER BY job_title;

jobs.job_title                     employees.rowid
---------------------------------- ------------------
Accountant                         AAAQNKAAFAAAABSAAL
Accountant                         AAAQNKAAFAAAABSAAN
Accountant                         AAAQNKAAFAAAABSAAM
Accountant                         AAAQNKAAFAAAABSAAJ
Accountant                         AAAQNKAAFAAAABSAAK
Accounting Manager                 AAAQNKAAFAAAABTAAH
Administration Assistant           AAAQNKAAFAAAABTAAC
Administration Vice President      AAAQNKAAFAAAABSAAC
Administration Vice President      AAAQNKAAFAAAABSAAB
.
.
.
```

データ・ウェアハウスでは、結合条件は、ディメンション表の主キー列とファクト表の外部キー列の間の等価結合(等価演算子を使用する)になります。ビットマップ結合索引は、格納に関して、事前に結合をマテリアライズするマテリアライズド結合ビューよりもはるかに効率的な場合があります。

関連項目:

ビットマップ結合索引の詳細は、Oracle Databaseデータ・ウェアハウス・ガイドを参照してください

ビットマップ記憶域構造

Oracle Databaseでは、Bツリー索引構造を使用して各索引付きキーのビットマップを格納します。

たとえば、jobs. job_titleがビットマップ索引のキー列である場合、1つのBツリーに索引データが格納されます。リーフ・ブロックには個々のビットマップが格納されます。

例3-6 ビットマップ記憶域の例

jobs. job_title列に一意の値であるShipping Clerk、Stock Clerk、およびその他いくつかの値があるとします。この索引のビットマップ索引エントリには、次のコンポーネントがあります。

- 索引キーとしての職種
- ROWIDの範囲として小さいROWIDと大きいROWID
- 範囲内の特定のROWIDのビットマップ

その結果、この索引の索引リーフ・ブロックには次のようなエントリが含まれる場合があります。

```
Shipping Clerk, AAAPzRAAFAAAABSABQ, AAAPzRAAFAAAABSABZ, 0010000100
Shipping Clerk, AAAPzRAAFAAAABSABa, AAAPzRAAFAAAABSABh, 010010
Stock Clerk, AAAPzRAAFAAAABSAAa, AAAPzRAAFAAAABSAAc, 1001001100
Stock Clerk, AAAPzRAAFAAAABSAAd, AAAPzRAAFAAAABSAAt, 0101001001
Stock Clerk, AAAPzRAAFAAAABSAAu, AAAPzRAAFAAAABSABz, 100001
.
.
.
```

ROWID範囲が異なるため、同じ職種が複数エントリに出現します。

セッションで、1人の従業員のジョブIDをShipping ClerkからStock Clerkに更新します。このような場合、セッションは古い値(Shipping Clerk)と新しい値(Stock Clerk)の索引キー・エントリに排他的にアクセスする必要があります。Oracle Databaseでは、UPDATEがコミットされるまで、これら2つのエントリで指し示された行がロックされます(ただし、Accountantまたはその他のキーで指し示された行はロックされません)。

ビットマップ索引のデータは1つのセグメント.に格納されます。Oracle Databaseでは、各ビットマップを1つ以上の断片に格納します。各断片は単一データ・ブロックの一部を占めます。

関連項目:

「ユーザー・セグメント」では、様々なセグメントのタイプと、セグメントの作成方法について説明します

ファンクション索引の概要

ファンクション索引では、1つ以上の列を含むファンクションや式の値が計算され、索引に格納されます。ファンクション索引はBツリー索引またはビットマップ索引のいずれかです。

索引付きファンクションは、算術式、あるいはSQL関数、ユーザー定義のPL/SQLファンクション、パッケージ・ファンクションまたはCコールアウトを含む式のいずれかです。たとえば、ファンクションでは2つの列の値を加算できます。

関連項目:

- ファンクション索引の作成方法を学習するには、『Oracle Database管理者ガイド』を参照
- ファンクション索引の使用方法の詳細は、『Oracle Database開発ガイド』を参照
- ファンクション索引の制約および使用上のノートは、『Oracle Database SQL言語リファレンス』を参照

ファンクション索引の使用方法

ファンクション索引は、WHERE句にファンクションを含む文を効率的に評価します。データベースでは、ファンクションが問合せに含まれている場合にのみ、ファンクション索引を使用します。ただし、INSERT文とUPDATE文の処理中には、データベースでは文を処理するために従来どおりファンクションを評価する必要があります。

例3-7 算術式に基づく索引

たとえば、次のようなファンクション索引を作成したとします。

```
CREATE INDEX emp_total_sal_idx
  ON employees (12 * salary * commission_pct, salary, commission_pct);
```

データベースでは、次(部分的なサンプル出力を示しています)に示すような問合せを処理するときに、この索引を使用できます。

```
SELECT   employee_id, last_name, first_name,
         12*salary*commission_pct AS "ANNUAL SAL"
FROM     employees
WHERE    (12 * salary * commission_pct) < 30000
ORDER BY "ANNUAL SAL" DESC;

EMPLOYEE_ID LAST_NAME                 FIRST_NAME           ANNUAL SAL
----------- ------------------------- -------------------- ----------
        159 Smith                     Lindsey                   28800
        151 Bernstein                 David                     28500
        152 Hall                      Peter                     27000
        160 Doran                     Louise                    27000
        175 Hutton                    Alyssa                    26400
        149 Zlotkey                   Eleni                     25200
        169 Bloom                     Harrison                  24000
```

例3-8 UPPERファンクションに基づく索引

SQLファンクションUPPER(*column_name*)またはLOWER(*column_name*)でファンクション索引を定義すると、大文字/小文字を区別しない検索が容易になります。たとえば、employeesのfirst_name列に大文字/小文字が含まれるとします。
hr.employees表に次のようなファンクション索引を作成します。

```
CREATE INDEX emp_fname_uppercase_idx
ON employees ( UPPER(first_name) );
```

emp_fname_uppercase_idx索引によって、次のような問合せが容易になります。

```
SELECT *
FROM   employees
WHERE  UPPER(first_name) = 'AUDREY';
```

例3-9 表の特定の行の索引付け

ファンクション索引は、表の特定の行のみに索引を付ける場合にも役に立ちます。たとえば、sh.customers表のcust_valid列には、値としてIまたはAがあるとします。Aの行のみに索引を付けるには、A以外のすべての行に対してNULL値を戻すファンク

ションを作成できます。次のような索引を作成できます。

```
CREATE INDEX cust_valid_idx
ON customers ( CASE cust_valid WHEN 'A' THEN 'A' END );
```

関連項目:

- 言語索引の詳細は、『*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。
- SQL関数についてさらに学習するには、『*Oracle Database SQL*言語リファレンス』を参照してください

ファンクション索引による最適化

WHERE句に式を含む問合せの場合、オプティマイザでは、ファンクション索引に対する索引レンジ・スキャンを使用できます。

範囲スキャンのアクセス・パスは、述語の選択性が高い場合、つまり選択される行が比較的少ない場合に特に効果を発揮します。例3-7では、式12*salary*commission_pctの索引が作成されていれば、オプティマイザは索引レンジ・スキャンを使用できます。

また、仮想列は、式から導出されるデータへのアクセスを高速化するために役立ちます。たとえば、仮想列annual_salを12*salary*commission_pctとして定義し、annual_salにファンクション索引を作成できます。

オプティマイザは、SQL文の式を解析し、文の式ツリーとファンクション索引を比較して、式のマッチングを実行します。この比較は大文字/小文字が区別され、ブランクは無視されます。

関連項目:

- 「オプティマイザの概要」
- 統計の収集についてさらに学習するには、Oracle Database SQLチューニング・ガイドを参照してください。
- 表への仮想列の追加方法を学習するには、Oracle Database管理者ガイドを参照してください。

アプリケーション・ドメイン索引の概要

アプリケーション・ドメイン索引は、アプリケーション固有のカスタマイズされた索引です。

拡張索引作成機能を使用すると、次のことが可能です。

- カスタマイズされた複合データ型(文書、空間データ、イメージ、ビデオ・クリップなど)の索引への対応(「非構造化データ」を参照)
- 特殊な索引作成方法の利用

アプリケーション固有の索引管理ルーチンを索引タイプ・スキーマ・オブジェクトとしてカプセル化し、オブジェクト型の表の列や属性のドメイン索引を定義できます。拡張索引作成機能により、アプリケーション固有の演算子を効率的に処理できます。

ドメイン索引の構造と内容は、 カートリッジと呼ばれるアプリケーション・ソフトウェアによって制御されます。データベースは、ドメイン索引の作成、メンテナンス、および検索のために、このアプリケーションと対話します。索引構造そのものは、索引構成表としてデータベースに格納するか、ファイルとして外部に格納できます。

関連項目:

Oracle Databaseの拡張性アーキテクチャ内のデータ・カートリッジの使用方法の詳細は、『*Oracle Databaseデータ・カートリッジ開発者ガイド*』を参照してください。

索引構成表の概要

索引構成表は、一種のBツリー索引構造に格納される表です。一方、**ヒープ構成表**では、行は収まる位置に挿入されます。

索引構成表では、行は表の主キーに定義された索引に格納されます。Bツリーの各索引エントリには、非キー列値も格納されます。したがって、索引はデータであり、データは索引です。アプリケーションは、ヒープ構成表と同じように、SQL文を使用して索引構成表を操作します。

たとえば、索引構成表について、人事管理マネージャが複数の段ボール箱を棚に置いて管理しているとします。それぞれの箱には1、2、3、4というように番号のラベルが付いていますが、箱は棚の中で順番に並んではいません。それぞれの箱には、次の箱が棚のどの位置にあるかを示すポインタがあります。

それぞれの箱には従業員レコードが入っているフォルダが収納されています。フォルダは従業員IDによってソートされています。従業員KingのIDは100で、それが最小のIDであるため、Kingのフォルダは箱1の一番下にあります。従業員101のフォルダは100の上、102は101の上というように、箱1がいっぱいになるまでフォルダが収められています。この並びにおける次のフォルダは、箱2の一番下になります。

この例の場合、フォルダを従業員IDで並べると、専用の索引をメンテナンスすることなく、フォルダを効率的に検索できます。ユーザーが従業員107、120、および122のレコードを要求していると想定します。1つのステップで索引を検索し、別のステップでフォルダを取り出すかわりに、マネージャは、フォルダを順番に検索し、個々のフォルダを見つかり次第取り出すことができます。

索引構成表は、主キーまたはそのキーの有効な接頭辞の使用により、表の行に対するアクセスを高速化します。リーフ・ブロックの行に非キー列が存在することにより、追加のデータ・ブロックI/Oが回避されます。たとえば、従業員100の給与は索引行自体に格納されます。さらに、行が主キーの順序で格納されるため、主キーまたは接頭辞によるレンジ・アクセスでは最小限のブロックI/Oが行われます。もう1つの利点は、個別の主キー索引の空間オーバーヘッドが回避されることです。

索引構成表は、関連したデータ断片をまとめて格納する必要がある場合や、データを特定の順序で物理的に格納する必要がある場合に役立ちます。通常、このタイプの表は情報取得、空間データおよびOLAPアプリケーションに使用されます。

関連項目:

- 「Oracle Spatial and Graphの概要」。
- 「OLAP」
- 索引構成表の管理方法を学習するには、『Oracle Database管理者ガイド』を参照してください。
- 索引構成表を使用してパフォーマンスを向上させる方法を学習するには、Oracle Database SQLチューニング・ガイドを参照してください。
- `CREATE TABLE ... ORGANIZATION INDEX`の構文およびセマンティクスについては、Oracle Database SQL言語リファレンス を参照してください

索引構成表の特性

データベース・システムは、Bツリー索引構造を操作することで、索引構成表に対するすべての操作を実行します。

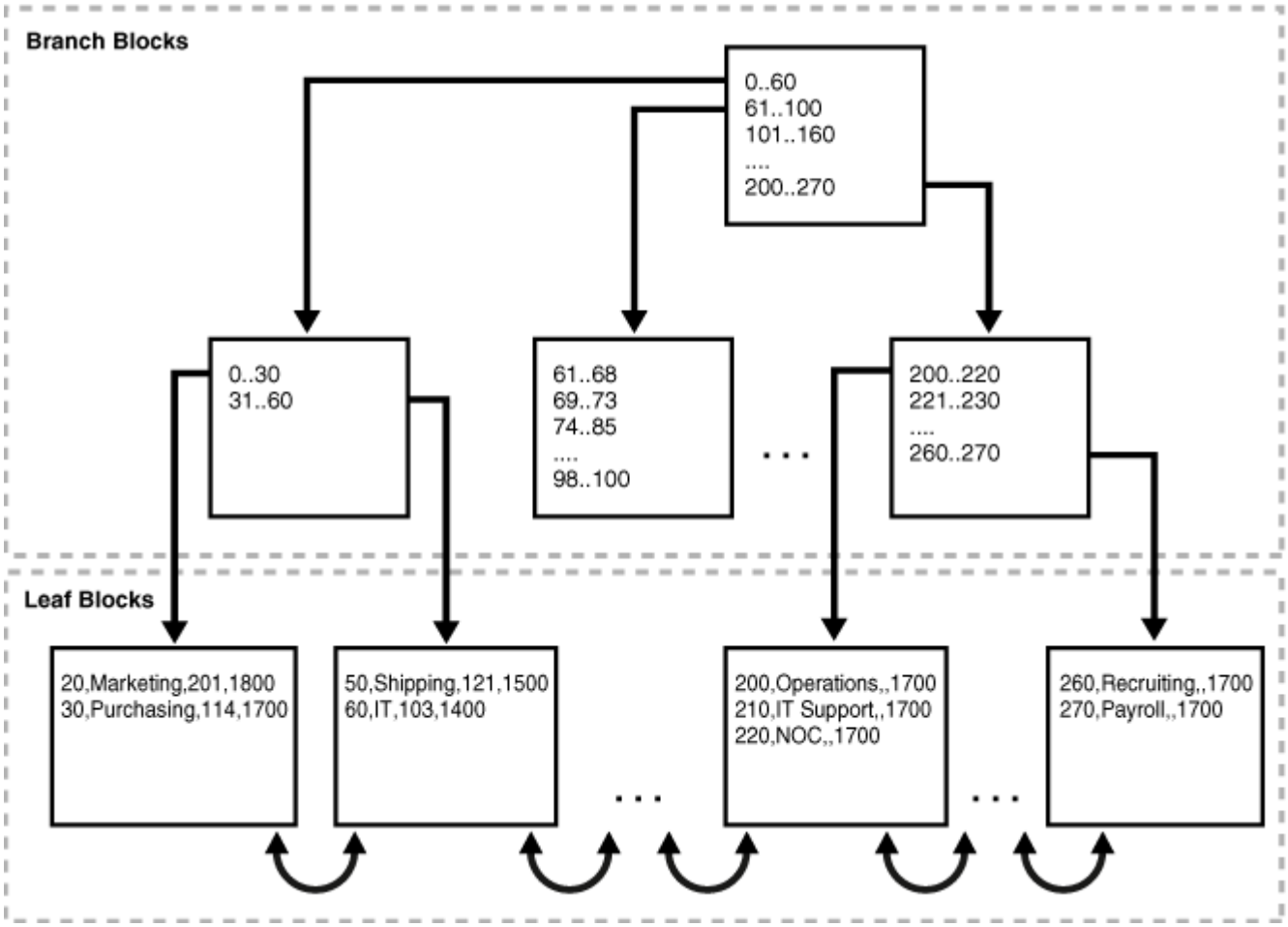
次の表は、索引構成表とヒープ構成表の違いをまとめたものです。

表3-6 ヒープ構成表と索引構成表の比較

ヒープ構成表	索引構成表
ROWID が行を一意に識別します。主キー制約はオプションで定義できます。	主キーが行を一意に識別します。主キー制約を定義する必要があります。
ROWID 擬似列の物理 ROWID で 2 次索引を作成できます。	ROWID 疑似列の論理 ROWID で 2 次索引を作成できます。
ROWID によって個々の行に直接アクセスできます。	個々の行には、主キーによって間接的にアクセスします。
順次全表スキャンは、特定の順序ですべての行を戻します。	全索引スキャンまたは高速全索引スキャンは、特定の順序ですべての行を戻します。
他の表とともに表クラスタに格納できます。	表クラスタには格納できません。
LONG データ型の列と LOB データ型の列を格納できます。	LOB 列は格納できますが、LONG 列は格納できません。
仮想列を格納できます(サポートされているのはリレーショナル・ヒープ表のみです)。	仮想列は格納できません。

図3-3に、索引構成表departmentsの構造を示します。リーフ・ブロックには表の行が主キーの順序に従って格納されています。たとえば、最初のリーフ・ブロックの最初の値は、部門ID 20、部門名Marketing、マネージャID 201、および場所ID 1800を示しています。

図3-3 索引構成表



例3-10 索引構成表のスキャン

索引構成表では、すべてのデータが同じ構造に格納され、ROWIDを格納する必要がありません。図3-3に示すように、索引構成表のリーフ・ブロック1には、主キーの順序に従った次のようなエントリが含まれます。

```
20,Marketing,201,1800
30,Purchasing,114,1700
```

索引構成表のリーフ・ブロック2には次のようなエントリが含まれます。

```
50,Shipping,121,1500
60,IT,103,1400
```

主キーの順序になっている索引構成表の行をスキャンすると、ブロックが次の順序で読み取られます。

1. ブロック1
2. ブロック2

例3-11 ヒープ構成表のスキャン

ヒープ構成表でのデータ・アクセスを索引構成表でのデータ・アクセスと比較するため、ヒープ構成表departmentsのセグメントのブロック1に、次のように行が格納されているとします。

```
50,Shipping,121,1500
20,Marketing,201,1800
```

ブロック2には同じ表の行が次のように格納されています。

```
30,Purchasing,114,1700
60,IT,103,1400
```

このヒープ構成表のBツリー索引リーフ・ブロックには、次のエントリが格納されています(最初の値は主キーであり、2つ目の値は

ROWIDです)。

```
20, AAAPeXAAFAAAAAyAAD
30, AAAPeXAAFAAAAAyAAA
50, AAAPeXAAFAAAAAyAAC
60, AAAPeXAAFAAAAAyAAB
```

主キーの順序になっている表の行をスキャンすると、表セグメント・ブロックが次の順序で読み取られます。

1. ブロック1
2. ブロック2
3. ブロック1
4. ブロック2

したがって、この例のブロックI/Oの数は、索引構成の例での数の倍になります。

関連項目:

- 「表の編成」

  (ヒープ構成表についてさらに学習するには)
- 「論理記憶域構造の概要」

  (セグメントとデータ・ブロックの関係についてさらに学習するには)

行オーバーフロー領域付きの索引構成表

索引構成表を作成するとき、別のセグメントを行オーバーフロー領域として指定できます。

索引構成表では、行全体を格納しているBツリー索引エントリは大きくなる場合があり、エントリを格納する別のセグメントが役立ちます。これに対し、Bツリー・エントリはキーとROWIDで構成されるため、通常は小さくなります。

行オーバーフロー領域を指定した場合、データベースでは索引構成表の行を次の部分に分割できます。

- 索引エントリ

  この部分には、主キー列すべての列値、行のオーバーフロー部分を指す物理ROWID、およびオプションで一部の非キー列が含まれます。この部分は索引セグメントに格納されます。
- オーバーフロー部分

  この部分には、残りの非キー列の列値が含まれます。この部分はオーバーフロー記憶域セグメントに格納されます。

関連項目:

- CREATE TABLEのOVERFLOW句を使用して行オーバーフロー領域を設定する方法を学習するには、Oracle Database管理者ガイドを参照してください。
- CREATE TABLE ... OVERFLOWの構文およびセマンティクスについては、Oracle Database SQL言語リファレンスを参照してください。

索引構成表の2次索引

2次索引は、索引構成表の索引です。

ある意味では、2次索引は索引の索引です。これは独立したスキーマ・オブジェクトであり、索引構成表とは別に格納されます。

Oracle Databaseでは、索引構成表に対して論理ROWIDと呼ばれる行識別子を使用します。論理ROWIDは、表の主キーをbase64でエンコードした表現です。論理ROWIDの長さは、主キーの長さに応じて異なります。

索引リーフ・ブロック内の行は、挿入によってブロック内またはブロック間で移動できます。索引構成表の行は、ヒープ構成の行のように移行しません。索引構成表の行には永続的な物理アドレスがないため、データベースでは主キーに基づいて論理ROWIDを使用します。

たとえば、departments表が索引構成表であるとします。location_id列には各部門のIDが格納されます。表には次のように行が格納され、最後の値が場所IDになります。

```
10,Administration,200,1700
20,Marketing,201,1800
30,Purchasing,114,1700
40,Human Resources,203,2400
```

location_id列の2次索引には、次のような索引エントリが含まれる場合があります。カンマに続く値が論理ROWIDです。

```
1700,*BAFAJqoCwR/+
1700,*BAFAJqoCwQv+
1800,*BAFAJqoCwRX+
2400,*BAFAJqoCwSn+
```

2次索引は、主キーでも主キーの接頭辞でもない列を使用して、索引構成表への高速で効率的なアクセスを可能にします。たとえば、IDが1700より大きい部門名の問合せでは、2次索引を使用してデータ・アクセスを高速化できます。

関連項目:

- ROWIDの使用およびROWID疑似列についてさらに学習するには、「ROWIDデータ型」を参照
- 「行の連鎖と移行」

  (行の移行の理由と、移行によって入出力回数が増える理由の詳細)
- 索引構成表に2次索引を作成する方法を学習するには、『Oracle Database管理者ガイド』を参照
- 索引構成表パーティションに2次索引を作成する方法について学習するには、『Oracle Database VLDBおよびパーティショニング・ガイド』を参照

論理ROWIDと物理推測

2次索引では論理ROWIDを使用して表の行を検索します。

論理ROWIDには、索引エントリの最初の作成時の物理ROWIDである**物理推測**が含まれます。Oracle Databaseは、物理推測を使用して、主キー検索をバイパスし、索引構成表のリーフ・ブロックを直接検証できます。行の物理位置が変化した場合、論理ROWIDは失効した物理推測が含まれていても引き続き有効です。

ヒープ構成表の場合、2次索引によるアクセスでは、行を含むデータ・ブロックをフェッチするために、2次索引のスキャンと付加的なI/Oが必要になります。索引構成表の場合、2次索引によるアクセスは、次のように物理推測を使用するかどうかと、その正確さに応じて異なります。

- 物理推測を使用しない場合、アクセスでは2次索引のスキャンとその後に主キー索引のスキャンという、2つの索引スキャンが行われます。
- 物理推測を使用する場合は、アクセスは物理推測の正確さに応じて異なります。
    - 正確な物理推測を使用する場合、アクセス時には、2次索引スキャンと追加のI/Oによって行を含むデータ・ブロックがフェッチされます。
    - 不正確な物理推測を使用する場合、アクセス時には2次索引スキャンとI/Oによって間違ったデータ・ブロック(物理推測が示すブロック)がフェッチされてから、主キー値による索引構成表の索引一意スキャンが実行されます。

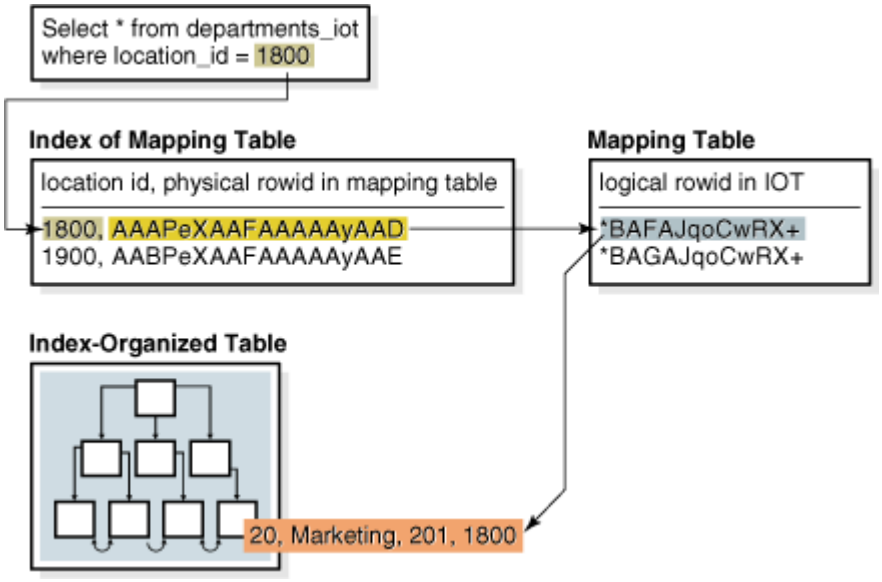
索引構成表のビットマップ索引

索引構成表の2次索引は、**ビットマップ索引**にできます。ビットマップ索引では、各索引キーのビットマップが格納されます。

索引構成表にビットマップ索引が存在する場合、すべてのビットマップ索引はヒープ構成のマッピング表を使用します。マッピング表には索引構成表の論理ROWIDが格納されます。マッピング表の各行には対応する索引構成表の行の論理ROWIDが格納されます。

データベースでは、検索キーを使用してビットマップ索引にアクセスします。キーが見つかると、ビットマップのエントリは物理ROWIDに変換されます。ヒープ構成表の場合、この物理ROWIDは実表へのアクセスに使用されます。索引構成表の場合、この物理ROWIDはマッピング表へのアクセスに使用され、それにより、索引構成表へのアクセスに使用される論理ROWIDが取得されます。次の図に、departments_iot表の問合せ用の索引アクセスを示します。

図3-4 索引構成表に対するビットマップ索引



ノート:

索引構成表で行を移動すると、その索引構成表に作成されたビットマップ索引の使用禁止状態が変化します。

関連項目:

物理ROWIDと論理ROWIDの相違点について学習するには、「行断片のROWID」を参照

4 パーティション、ビューおよびその他のスキーマ・オブジェクト

最も重要かつ一般的に使用されるスキーマ・オブジェクトは表と索引ですが、データベースでは他のタイプの多数のスキーマ・オブジェクトもサポートされています。この章では、最も一般的なスキーマ・オブジェクトについて説明します。

この章の構成は、次のとおりです。

- パーティションの概要
- シャード表の概要
- ビューの概要
- マテリアライズド・ビューの概要
- 順序の概要
- ディメンションの概要
- シノニムの概要

パーティションの概要

Oracleデータベースでは、**パーティション化**により、大規模な表や索引を**パーティション**という、より小さく管理しやすい部分に分割できます。各パーティションは独自の名前を持つ独立したオブジェクトであり、独自の記憶特性を持つ場合があります。

パーティション化の例として、ある人事管理マネージャが複数の従業員フォルダを含む1つの大きな箱を管理しているとします。各フォルダは従業員の雇用日を示しています。特定の月に雇用された従業員に対する問合せが頻繁に行われます。このような要求を満たす1つのアプローチとして、箱の中に分散しているフォルダの位置を特定する索引を、従業員の雇用日に対して作成する方法があります。一方、パーティション化計画ではより小さな箱を多数使用して、指定した月に雇用された従業員のフォルダをそれぞれの箱に格納します。

より小さな箱の使用には、いくつかの利点があります。6月に雇用された従業員のフォルダを取り出すよう依頼された場合、人事管理マネージャは6月の箱を取り出します。さらに、いずれかの小さな箱が破損したとしても、他の小さな箱は引き続き使用できます。また、マネージャは1つの重い箱を移動するのではなく、複数の小さな箱を移動すればよいため、オフィスの移転も容易になります。

アプリケーションから見ると、スキーマ・オブジェクトは1つのみ存在します。パーティション表にアクセスするためにSQL文を修正する必要はありません。パーティション化は様々なタイプのデータベース・アプリケーションで有効ですが、特に大量のデータを管理するアプリケーションで便利です。次のような利点があります。

- 可用性の向上

  いずれかのパーティションが使用できない場合も、オブジェクトの使用には関係ありません。問合せオプティマイザは参照されないパーティションを問合せ計画から自動的に削除するため、パーティションが使用できない場合にも、問合せは影響を受けません。

- スキーマ・オブジェクトの管理の簡素化

  パーティション・オブジェクトには、集中管理および個別管理のいずれも可能な要素があります。DDL文では、表全体または索引ではなくパーティションを操作できます。このため、索引または表の再作成など、リソース集中型のタスクを細分化できます。たとえば、一度に1つずつ表のパーティションを移動できます。問題が発生した場合は、表の移動ではなく、パーティションの移動のみを再実行します。また、パーティションを削除すると、DELETE文を多数実行する必要がなくなります。

- OLTPシステムの共有リソースの競合の軽減

  一部のOLTPシステムでは、パーティションによって共有リソースの競合を軽減できます。たとえば、DMLを1つのセグメントではなく、多数のセグメントに分散します。

- データ・ウェアハウスでの問合せパフォーマンスの向上

  データ・ウェアハウスでは、パーティションによって非定型問合せの処理を高速化できます。たとえば、100万行ある売上表を四半期ごとにパーティション化できます。

関連項目:

パーティション化の概要は、『Oracle Database VLDBおよびパーティショニング・ガイド』を参照してください。

パーティションの特性

表や索引の各パーティションは、列名、データ型、制約などの論理属性が同じである必要があります。

たとえば、1つの表内のすべてのパーティションは同じ列定義と制約定義を共有します。ただし、パーティションが属する表スペースなど、各パーティションの物理属性は異なる場合があります。

パーティション・キー

パーティション・キーは、パーティション表の各行が属するパーティションを決定する1つ以上の列のセットです。各行は、単一パーティションに明示的に割り当てられます。

たとえば、sales表で、レンジ・パーティションのキーとしてtime_id列を指定できます。Oracle Databaseにより、この列の日付が指定された範囲内にあるかどうかに基づいてパーティションに行が割り当てられます。Oracle Databaseでは、パーティション・キーを使用して適切なパーティションに対して挿入、更新および削除操作が自動的に行われます。

パーティション化計画

Oracle Partitioningは、データベースによる個々のパーティションへのデータの配置方法を制御する、複数のパーティション化計画を提供します。基本的な計画は、レンジ・パーティション化、リスト・パーティション化およびハッシュ・パーティション化です。

単一レベル・パーティション化では、リスト・パーティションのみまたはレンジ・パーティションのみなど、データ配分方法を1つのみ使用します。コンポジット・パーティション化では、1つのデータ配分方法で表をパーティション化し、別のデータ配分方法を使用して、各パーティションをさらに小さなサブパーティションに分割します。たとえば、channel_idにリスト・パーティションを使用し、time_idにレンジ・サブパーティションを使用できます。

例4-1 パーティション表のサンプル行セット

このパーティション化の例では、次の行を含むパーティション表salesを移入するとします。

```
  PROD_ID    CUST_ID TIME_ID   CHANNEL_ID   PROMO_ID QUANTITY_SOLD AMOUNT_SOLD
---------- ---------- --------- ---------- ---------- ------------- -----------
      116      11393 05-JUN-99          2        999             1       12.18
       40     100530 30-NOV-98          9         33             1       44.99
      118        133 06-JUN-01          2        999             1       17.12
      133       9450 01-DEC-00          2        999             1       31.28
       36       4523 27-JAN-99          3        999             1       53.89
      125       9417 04-FEB-98          3        999             1       16.86
       30        170 23-FEB-01          2        999             1         8.8
       24      11899 26-JUN-99          4        999             1       43.04
       35       2606 17-FEB-00          3        999             1       54.94
```

```
        45        9491 28-AUG-98           4          350                1        47.45
```

レンジ・パーティション化

レンジ・パーティション化では、Oracle Databaseはパーティション化キーの値の範囲に基づいて、行をパーティションにマップします。レンジ・パーティション化は、最も一般的なタイプのパーティション化であり、通常は日付とともに使用されます。

time_id列をパーティション・キーとして指定して次のSQL文を使用して、time_range_salesをパーティション表として作成するとします。

```
CREATE TABLE time_range_sales
   ( prod_id        NUMBER(6)
   , cust_id        NUMBER
   , time_id        DATE
   , channel_id     CHAR(1)
   , promo_id       NUMBER(6)
   , quantity_sold  NUMBER(3)
   , amount_sold    NUMBER(10,2)
   )
PARTITION BY RANGE (time_id)
 (PARTITION SALES_1998 VALUES LESS THAN (TO_DATE('01-JAN-1999','DD-MON-YYYY')),
  PARTITION SALES_1999 VALUES LESS THAN (TO_DATE('01-JAN-2000','DD-MON-YYYY')),
  PARTITION SALES_2000 VALUES LESS THAN (TO_DATE('01-JAN-2001','DD-MON-YYYY')),
  PARTITION SALES_2001 VALUES LESS THAN (MAXVALUE)
 );
```

その後、例4-1の行をtime_range_salesにロードします。このコードは、4つのパーティションへの行の配分を示します。Oracle Databaseでは、PARTITION BY RANGE句に指定されたルールに従って、time_id値に基づいて各行のパーティションが選択されます。レンジ・パーティション・キーの値により、指定されたパーティションについて非包含的上限が決定されます。

時間隔パーティション化

時間隔パーティション化は、レンジ・パーティション化の拡張機能です。

既存のレンジ・パーティションを超えるデータを挿入した場合、Oracle Databaseでは指定した時間間隔のパーティションが自動的に作成されます。たとえば、各月のデータを別のパーティションに格納する、販売履歴表を作成する場合があります。

時間隔パーティションでは、レンジ・パーティションを明示的に作成することを回避できます。時間隔パーティション化は、レンジ・パーティション化されており、新しいパーティションで固定間隔を使用するほぼすべての表で使用できます。レンジ・パーティションを様々な間隔で作成しない場合や、特定のパーティション属性を常に設定するわけではない場合は、時間隔パーティションの使用を検討してください。

時間隔でパーティション化する際は、少なくとも1つのレンジ・パーティションを指定してください。レンジ・パーティション化キーの値は、遷移ポイントと呼ばれるレンジ・パーティションの値の上限を決定します。データベースでは遷移ポイントを超える値を持つデータに対して、時間隔パーティションが自動的に作成されます。各時間隔パーティションの下限は、前のレンジまたは時間隔パーティションの上限(境界を含む)です。したがって、例4-2では、値01-JAN-2011はパーティションp2にあります。

データベースでは遷移ポイントを超えるデータに対して、時間隔パーティションを作成します。時間隔パーティション化は、指定された範囲または間隔のパーティションを作成するようデータベースに指示するレンジ・パーティション化の拡張機能です。表に挿入されたデータが既存のすべてのレンジ・パーティションを超えると、データベースによってパーティションが自動的に作成されます。例4-2では、p3パーティションに、パーティション化キーtime_idの値が01-JAN-2013と等しいか、それより大きい行が含まれています。

例4-2 時間隔パーティション化

様々な幅の4つのパーティションを持つ販売表を作成すると仮定します。2013年1月1日の遷移ポイントを超えると、データベー

スでは1か月の時間隔でパーティションが作成されることを指定します。パーティションp3の上位バウンドは遷移ポイントを表します。パーティションp3およびそれより下位のすべてのパーティションはレンジ・セクションにありますが、それより上位のすべてのパーティションは時間隔セクションに分類されます。

```
CREATE TABLE interval_sales
    ( prod_id        NUMBER(6)
    , cust_id        NUMBER
    , time_id        DATE
    , channel_id     CHAR(1)
    , promo_id       NUMBER(6)
    , quantity_sold  NUMBER(3)
    , amount_sold    NUMBER(10,2)
    )
  PARTITION BY RANGE (time_id)
  INTERVAL(NUMTOYMINTERVAL(1, 'MONTH'))
    ( PARTITION p0 VALUES LESS THAN (TO_DATE('1-1-2010', 'DD-MM-YYYY'))
    , PARTITION p1 VALUES LESS THAN (TO_DATE('1-1-2011', 'DD-MM-YYYY'))
    , PARTITION p2 VALUES LESS THAN (TO_DATE('1-7-2012', 'DD-MM-YYYY'))
    , PARTITION p3 VALUES LESS THAN (TO_DATE('1-1-2013', 'DD-MM-YYYY')) );
```

2014年10月10日に行われた販売を挿入します。

```
SQL> INSERT INTO interval_sales VALUES (39,7602,'10-OCT-14',9,null,1,11.79);

1 row created.
```

USER_TAB_PARTITIONSの問合せでは、販売日が遷移ポイントよりも後であるため、10月10日の販売用の新規パーティションがデータベースで作成されたことが示されます。

```
SQL> COL PNAME FORMAT a9
SQL> COL HIGH_VALUE FORMAT a40
SQL> SELECT PARTITION_NAME AS PNAME, HIGH_VALUE
  2  FROM USER_TAB_PARTITIONS WHERE TABLE_NAME = 'INTERVAL_SALES';

PNAME     HIGH_VALUE
--------- ----------------------------------------
P0        TO_DATE(' 2007-01-01 00:00:00', 'SYYYY-M
          M-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA
P1        TO_DATE(' 2008-01-01 00:00:00', 'SYYYY-M
          M-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA
P2        TO_DATE(' 2009-07-01 00:00:00', 'SYYYY-M
          M-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA
P3        TO_DATE(' 2010-01-01 00:00:00', 'SYYYY-M
          M-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA
SYS_P1598 TO_DATE(' 2014-11-01 00:00:00', 'SYYYY-M
          M-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA
```

関連項目:

時間隔パーティションについてさらに学習するには、『Oracle Database VLDBおよびパーティショニング・ガイド』を参照

リスト・パーティション化

リスト・パーティション化の場合、Oracle Databaseでは、各パーティションのパーティション・キーとして離散値のリストが使用されます。パーティション化キーは1つ以上の列で構成されます。

リスト・パーティション化を使用して、特定のパーティションに対する個々の行のマップ方法を制御できます。リストを使用すると、データの識別に使用されるキーの並び順が目的にそぐわない場合に、関連するデータのセットをグループ化して整理できます。

例4-3 リスト・パーティション化

channel_id列がパーティション・キーである次の文を使用して、リスト・パーティション表としてlist_salesを作成するとします。

```
CREATE TABLE list_sales
   ( prod_id        NUMBER(6)
   , cust_id        NUMBER
   , time_id        DATE
   , channel_id     CHAR(1)
   , promo_id       NUMBER(6)
   , quantity_sold  NUMBER(3)
   , amount_sold    NUMBER(10,2)
   )
PARTITION BY LIST (channel_id)
 ( PARTITION even_channels VALUES ('2','4'),
   PARTITION odd_channels VALUES ('3','9')
 );
```

その後、例4-1の行を表にロードします。このコードは、2つのパーティションへの行の配分を示します。データベースでは、PARTITION BY LIST句に指定されたルールに従って、channel_id値に基づいて各行のパーティションが選択されます。channel_idの値が2または4である行はEVEN_CHANNELSパーティションに格納され、channel_idの値が3または9である行はODD_CHANNELSパーティションに格納されます。

ハッシュ・パーティション化

ハッシュ・パーティション化では、データベースがユーザー指定のパーティション・キーに適用するハッシング・アルゴリズムに基づいて、行をパーティションにマップします。

行のマップ先は、データベースが行に適用する内部ハッシュ関数によって決定されます。パーティションの数が2の累乗である場合、ハッシング・アルゴリズムにより、すべてのパーティションに対して行がほぼ均等に分散されます。

ハッシュ・パーティション化は、大きな表を分割して管理しやすくする場合に役立ちます。1つの大きな表ではなく、複数のより小さい単位を管理します。1つのハッシュ・パーティションが失われても、残りのパーティションに影響はなく、個別にリカバリできます。また、ハッシュ・パーティション化は、更新時に競合が発生しやすいOLTPシステムにも役立ちます。たとえば、競合が発生しやすい1つのセグメントを複数の単位に分割し、各単位が個別に更新されるようにできます。

prod_id列をパーティション・キーとして指定して次の文を使用して、パーティション化されたhash_sales表を作成するとします。

```
CREATE TABLE hash_sales
   ( prod_id        NUMBER(6)
   , cust_id        NUMBER
   , time_id        DATE
   , channel_id     CHAR(1)
   , promo_id       NUMBER(6)
   , quantity_sold  NUMBER(3)
   , amount_sold    NUMBER(10,2)
   )
PARTITION BY HASH (prod_id)
PARTITIONS 2;
```

その後、例4-1の行を表にロードします。このコードは、2つのパーティションへの行の配分の例を示します。これらのパーティションの名前は、システムによって生成されます。

ユーザーが行を挿入すると、データベースではすべてのパーティションへのランダムかつ均等な行の配分が試行されます。行をどのパーティションに配置するかは指定できません。データベースはハッシュ関数を適用し、関数の出力結果に基づいて、どのパーティ

ションに行を格納するかを決定します。

関連項目:

- パーティションの作成方法を学習するには、『Oracle Database VLDBおよびパーティショニング・ガイド』を参照してください。
- CREATE TABLE ... PARTITION BYの例は、『Oracle Database SQL言語リファレンス』を参照してください。

参照パーティション化

参照パーティション化で、子表のパーティション化計画は、外部キーの親表との関係から単独で定義されます。親表内の各パーティションごとに、対応する1つのパーティションが子表内に存在します。親表では特定のパーティションに親レコードが格納され、子表では対応するパーティションに子レコードが格納されます。

たとえば、orders表はline_items表の親で、主キーおよび外部キーがorder_idで定義されています。表は参照別にパーティション化されます。たとえば、データベースがオーダー233をordersのパーティションQ3_2015に格納する場合、データベースはオーダー233のすべての明細項目をline_itemsのパーティションQ3_2015に格納します。パーティションQ4_2015がordersに追加される場合、データベースは自動的にQ4_2015をline_itemsに追加します。

参照パーティション化の利点は次のとおりです。

- 親表と子表の両方で同じパーティション計画を使用することにより、すべてのパーティション化キー列の重複を回避します。この計画によって非正規化の手動によるオーバーヘッドが軽減され、領域も節約できます。
- 親表に対するメンテナンス操作は、子表に対して自動的に行われます。たとえば、マスター表にパーティションを追加すると、データベースによって自動的にこの追加が子表に伝播されます。
- データベースでは自動的に親表および子表でパーティションのパーティション全体結合が使用されるため、パフォーマンスが向上します。

時間隔パーティション化などのすべての基本パーティション化計画とともに、参照パーティション化を使用できます。参照パーティション表をコンポジット・パーティション表として作成することもできます。

例4-4 参照パーティション表の作成

この例は、order_dateでレンジ・パーティション化された親表ordersを作成します。参照パーティションの子表order_itemsは、Q1_2015、Q2_2015、Q3_2015およびQ4_2015の4つのパーティションとともに作成されています。各パーティションには、それぞれの親パーティションの注文に対応するorder_items行が含まれています。

```
CREATE TABLE orders
    ( order_id           NUMBER(12),
      order_date         DATE,
      order_mode         VARCHAR2(8),
      customer_id        NUMBER(6),
      order_status       NUMBER(2),
      order_total        NUMBER(8,2),
      sales_rep_id       NUMBER(6),
      promotion_id       NUMBER(6),
      CONSTRAINT orders_pk PRIMARY KEY(order_id)
    )
  PARTITION BY RANGE(order_date)
    ( PARTITION Q1_2015 VALUES LESS THAN (TO_DATE('01-APR-2015','DD-MON-YYYY')),
      PARTITION Q2_2015 VALUES LESS THAN (TO_DATE('01-JUL-2015','DD-MON-YYYY')),
      PARTITION Q3_2015 VALUES LESS THAN (TO_DATE('01-OCT-2015','DD-MON-YYYY')),
```

```
      PARTITION Q4_2015 VALUES LESS THAN (TO_DATE('01-JAN-2006','DD-MON-YYYY'))
    );

CREATE TABLE order_items
    ( order_id           NUMBER(12) NOT NULL,
      line_item_id       NUMBER(3)  NOT NULL,
      product_id         NUMBER(6)  NOT NULL,
      unit_price         NUMBER(8,2),
      quantity           NUMBER(8),
      CONSTRAINT order_items_fk
      FOREIGN KEY(order_id) REFERENCES orders(order_id)
    )
    PARTITION BY REFERENCE(order_items_fk);
```

関連項目:

参照パーティション化の概要は、『Oracle Database VLDBおよびパーティショニング・ガイド』を参照

コンポジット・パーティション化

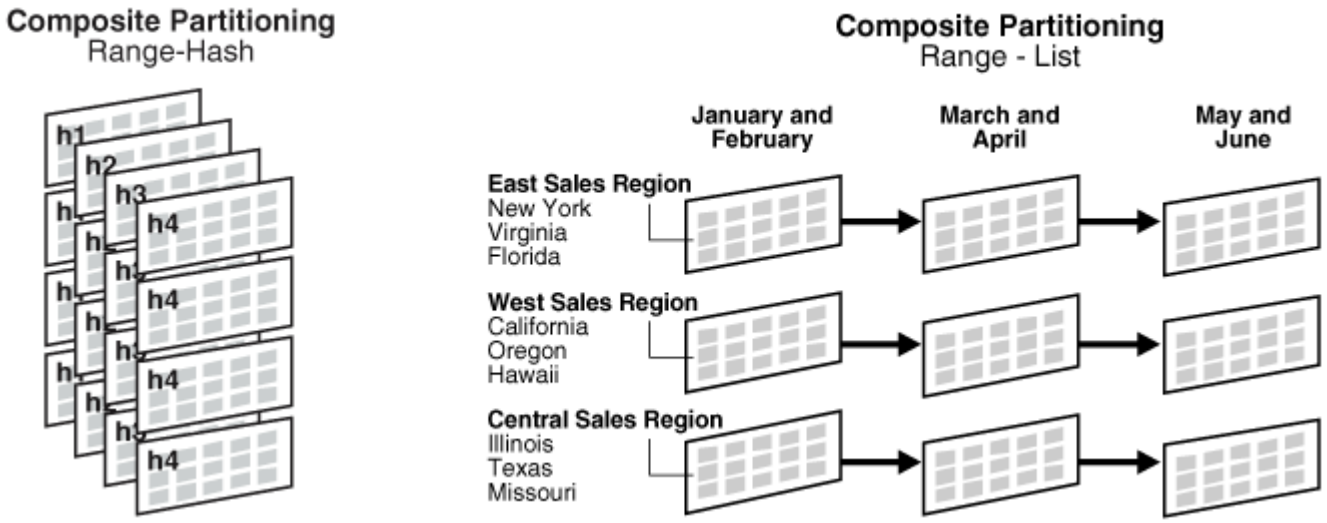
コンポジット・パーティション化では、1つのデータ配分方法で表をパーティション化し、別のデータ配分方法を使用して各パーティションをさらに小さなサブパーティションに再分割します。このように、コンポジット・パーティション化は基本的なデータ分配方法を組み合せたものです。1つのパーティションのすべてのサブパーティションは、データの論理的サブセットを表します。

コンポジット・パーティション化にはいくつかの利点があります。

- SQL文によっては、1つまたは2つのディメンションに対するパーティション・プルーニングによってパフォーマンスが向上する場合があります。
- 問合せで、いずれかのディメンションに対してパーシャルまたはフル・パーティション・ワイズ結合を使用できる場合があります。
- 1つの表のパラレル・バックアップおよびリカバリを実行できます。
- パーティション数が単一レベル・パーティション化より多いため、効果的なパラレル実行が可能になります。
- 履歴データをサポートするためのローリング・ウィンドウを実装でき、多くの文がパーティション・プルーニングまたはパーティション・ワイズ結合の恩恵を受ける場合には、別のディメンションでのパーティション化も可能です。
- パーティション化キーによる識別に基づいて、異なる方法でデータを格納できます。たとえば、特定の製品タイプのデータは読取り専用として圧縮形式で格納し、その他の製品タイプのデータは圧縮せずに保存することができます。

コンポジット・パーティション表のサブパーティション化計画としては、レンジ、リストおよびハッシュ・パーティション化が適しています。次の図に、レンジ - ハッシュおよびレンジ - リスト・コンポジット・パーティション化の図を示します。

図4-1 コンポジット・レンジ - リスト・パーティション化



データベースでは、コンポジット・パーティション化された表の各サブパーティションが個別のセグメントとして格納されます。このため、サブパーティションのプロパティは、表のプロパティや、そのサブパーティションが属するパーティションとは異なる場合があります。

関連項目:

コンポジット・パーティション化についてさらに学習するには、『Oracle Database VLDBおよびパーティショニング・ガイド』を参照

パーティション表

パーティション表は1つ以上のパーティションで構成されており、パーティションは個別に管理され、他のパーティションから独立して機能できます。

表はパーティション化または非パーティション化のいずれかの状態です。パーティション表が1つのパーティションのみで構成されている場合も、この表は、パーティションを追加できない非パーティション表とは異なります。

関連項目:

パーティション表の例は、「パーティションの特性」を参照

索引構成表の目的および特性について学習するには、「索引構成表の概要」を参照してください。索引構成表により、パーティション化による管理性、可用性およびパフォーマンスの向上の恩恵も受けられます。

パーティション表のセグメント

パーティション表は、1つ以上の表パーティション・セグメントで構成されています。

hash_productsという名前のパーティション表を作成した場合、この表に表セグメントは割り当てられません。かわりに、データベースでは独自のパーティション・セグメントに各表パーティションのデータを格納します。各表パーティション・セグメントには表データの断片が含まれます。

外部表がパーティション化されている場合、すべてのパーティションはデータベース外に存在します。ハイブリッド・パーティション表では、一部のパーティションはセグメントに格納され、それ以外は外部に格納されます。たとえば、sales表の一部のパーティションをデータ・ファイルに、それ以外をスプレッドシート内に格納できます。

関連項目:

- 「外部表の概要」
- 「セグメントの概要」

  オブジェクトとセグメントの関係について学習するには

パーティション表の圧縮

ヒープ構成表の一部またはすべてのパーティションは圧縮形式で格納できます。

圧縮すると領域を節約でき、問合せの実行を高速化できます。このため、圧縮は挿入および更新操作の量が少ないデータ・ウェアハウスなどの環境およびOLTP環境で役立つ場合があります。

表圧縮の属性は、表領域、表または表パーティションについて宣言できます。表領域レベルで宣言した場合、その表領域に作成される表はデフォルトで圧縮されます。表の圧縮属性を変更すると、変更はその表に格納される新規データにのみ適用されます。このため、1つの表またはパーティションには圧縮ブロックと非圧縮ブロックの両方を含むことができ、これにより、圧縮によってデータ・サイズが大きくならないことが保証されます。圧縮によってブロック・サイズが大きくなる場合、そのブロックに圧縮は適用されません。

関連項目:

- 基本表圧縮、高度な行圧縮、ハイブリッド列圧縮を含む表の圧縮のタイプについて学習するには、「表の圧縮」を参照
- データ・ウェアハウスでの表の圧縮について学習するには、『Oracle Databaseデータ・ウェアハウス・ガイド』を参照してください。

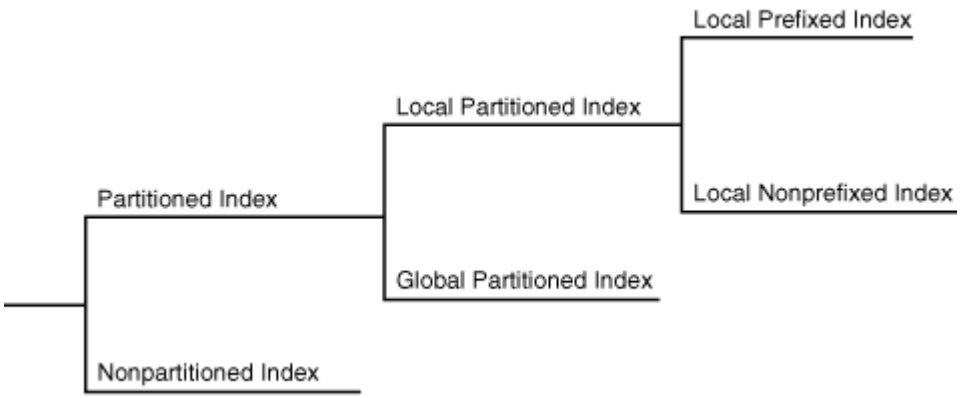
パーティション索引

パーティション索引とは、パーティション表のように、より小さく管理しやすい単位に分割された索引です。

グローバル索引は索引の作成対象の表から独立してパーティション化されますが、ローカル索引は表のパーティション化方法に自動的にリンクされます。パーティション表と同様、パーティション索引も管理性、可用性、パフォーマンスおよびスケーラビリティを改善します。

次の図に、索引パーティション化のオプションを示します。

図4-2 索引パーティション化のオプション



関連項目:

- 一意索引と非一意索引の相違点と、様々な索引タイプについて学習するには、「索引の概要」を参照
- パーティション索引と、使用するタイプの決定方法の詳細は、『Oracle Database VLDBおよびパーティショニング・ガイド』を参照

ローカル・パーティション索引

ローカル・パーティション索引では、その表と同じパーティション数および同じパーティション・バウンドで、同じ列に対して索引がパーティション化されます。

索引パーティションにあるすべてのキーが単一の表パーティションに格納された行のみを参照するように、各索引パーティションは、基礎となる表のパーティションと1対1で関連付けられます。この方法で、データベースは索引パーティションと関連する表パーティションを自動的に同期化し、表と索引の各ペアを独立させます。

ローカル・パーティション索引はデータ・ウェアハウス環境で共通のものです。ローカル索引には、次の利点があります。

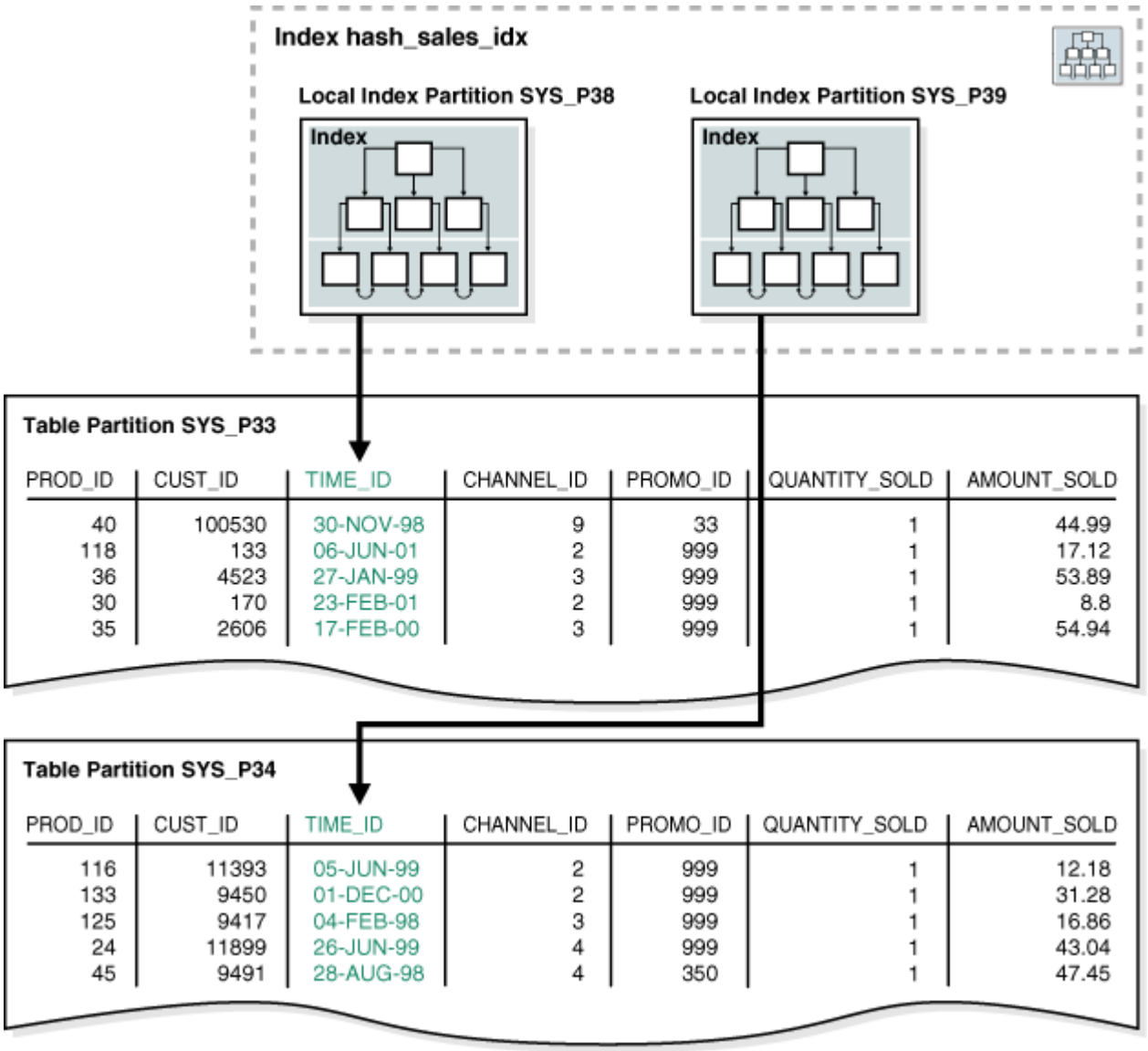
- パーティションのデータを無効または使用不可能にするアクションは、該当のパーティションにのみ反映されるため、可用性が向上します。
- パーティションのメンテナンスが簡単になります。表パーティションを移動した場合、またはデータがパーティションをエージ・アウトした場合、再作成またはメンテナンスが必要となるのは、関連するローカル索引パーティションのみです。グローバル索引では、すべての索引パーティションを再作成またはメンテナンスする必要があります。
- パーティションのポイント・イン・タイム・リカバリが発生した場合は、索引をリカバリ時間までリカバリできます(「データファイル・リカバリ」を参照)。索引全体を再作成する必要はありません。

ハッシュ・パーティション化の例は、prod_id列をパーティション・キーとして使用して、パーティション化されたhash_sales表を作成する文を示しています。次の例では、hash_sales表のtime_id列にローカル・パーティション索引を作成します。

```
CREATE INDEX hash_sales_idx ON hash_sales(time_id) LOCAL;
```

図4-3では、hash_products表に2つのパーティションがあり、そのため、hash_sales_idxにも2つのパーティションがあります。各索引パーティションは異なる表パーティションに関連付けられています。索引パーティションSYS_P38は表パーティションSYS_P33の行を索引付けするのに対し、索引パーティションSYS_P39は表パーティションSYS_P34の行を索引付けします。

図4-3 ローカル索引のパーティション



パーティションは、明示的にローカル索引に追加することはできません。基礎となる表にパーティションを追加した場合のみ、新しいパーティションがローカル索引に追加されます。同様に、パーティションを明示的にローカル索引から削除することもできません。かわりに、基礎となる表からパーティションを削除した場合にのみ、ローカル索引パーティションが削除されます。

他の索引の場合と同じように、ビットマップ索引もパーティション表に対して作成できます。ただし、ただ1つの制限として、ビットマップ索引はパーティション表に対してローカルであることが必要で、グローバル索引にはできません。グローバル・ビットマップ索引は、非パーティション表でのみサポートされます。

ローカルの同一キーおよび非同一キー索引

ローカル・パーティション索引は、同一キーまたは非同一キーのいずれかです。

索引のサブタイプは次のように定義されます。

- ローカル同一キー索引

  この索引の場合、パーティション・キーは索引定義の最初に配置されます。「レンジ・パーティション化」のtime_range_salesの例では、表がtime_idの範囲によってパーティション化されています。この表のローカル同一キー索引では、time_idがリストの最初の列になります。

- ローカル非同一キー索引

  この索引の場合、パーティション・キーは索引付けされた列のリストの最初にはなく、またリスト内に存在する必要もありません。「ローカル・パーティション索引」のhash_sales_idxの例では、パーティション・キーproduct_idが最初に置かれていないため、索引はローカル非同一です。

これらの両方のタイプの索引で、パーティション絞込み(パーティション・プルーニングとも呼ばれる)を活用でき、この機能は、オプティマイザがパーティションを対象外とすることによって、データ・アクセスを高速化する場合に使用されます。問合せでパーティションを除外できるかどうかは、問合せの述語によって決まります。ローカル同一キー索引を使用する問合せでは、常に索引のパーティション絞込みが可能ですが、ローカル非同一キー索引を使用している問合せでは、絞込みはできない場合があります。

関連項目:

同一キー索引および非同一キー索引の使用方法を学習するには、『Oracle Database VLDBおよびパーティショニング・ガイド』を参照してください

ローカル・パーティション索引の記憶域

表パーティションと同様に、ローカル索引パーティションは独自のセグメントに格納されます。各セグメントには、索引の総データの断片が含まれます。したがって、4つのパーティションから構成されるローカル索引は単一の索引セグメントではなく、4つの別々のセグメントに格納されます。

関連項目:

CREATE INDEX ... LOCALの例については、『*Oracle Database SQL言語リファレンス*』を参照してください。

グローバル・パーティション索引

グローバル・パーティション索引は、この索引の作成対象である基礎となる表とは別にパーティション化されたBツリー索引です。単一の索引パーティションは、任意またはすべての表パーティションを指す場合があり、その一方で、ローカル・パーティション索引では、索引パーティションと表パーティションの間に1対1のパリティが存在します。

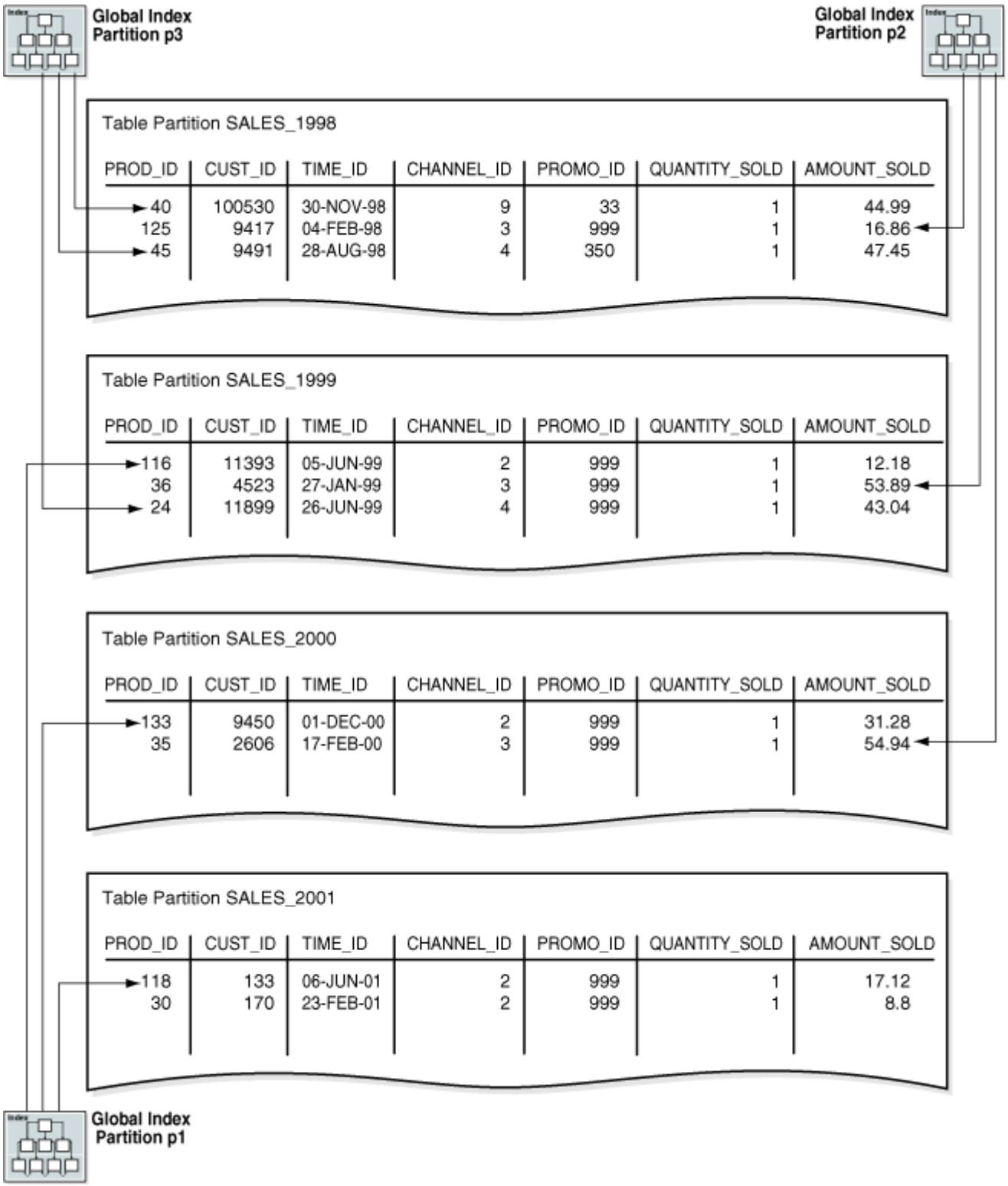
一般的に、グローバル索引は高速アクセス、データ整合性および可用性が重要となるOLTPアプリケーションに役立ちます。OLTPシステムでは、employees.department_id列など1つのキーによって表がパーティション化される場合がありますが、アプリケーションではemployee_idまたはjob_idなどの多数の異なるキーによるデータへのアクセスが必要になる可能性があります。このような使用例では、グローバル索引が便利な場合があります。

たとえば、「レンジ・パーティション化」のtime_range_sales表にグローバル・パーティション索引を作成するとします。この表では、1998年の売上を含む行が1つのパーティションに格納され、1999年の売上を含む行は別のパーティションに格納されるというように、行が格納されます。次の例では、channel_id列の範囲別にパーティション化されたグローバル索引を作成します。

```
CREATE INDEX time_channel_sales_idx ON time_range_sales (channel_id)
   GLOBAL PARTITION BY RANGE (channel_id)
      (PARTITION p1 VALUES LESS THAN (3),
       PARTITION p2 VALUES LESS THAN (4),
       PARTITION p3 VALUES LESS THAN (MAXVALUE));
```

図4-4に示すように、グローバル索引パーティションには、複数の表パーティションをそれぞれ指すエントリを含めることができます。索引パーティションp1はchannel_idが2の行を指し、索引パーティションp2はchannel_idが3の行を指し、索引パーティションp3はchannel_idが4または9の行を指しています。

図4-4 グローバル・パーティション索引



関連項目:

- グローバル・パーティション索引の管理方法を学習するには、Oracle Database VLDBおよびパーティショニング・ガイドを参照してください
- CREATE INDEXのGLOBAL PARTITION句について学習するには、Oracle Database SQL言語リファレンスを参照してください

パーティション表の部分索引

部分索引とは、関連付けられたパーティション表の索引付けプロパティと相互に関連付けられている索引のことです。

相関関係により、どの表パーティションに索引を付けるかを指定できます。部分索引には、次の利点があります。

- 索引付けされていない表パーティションでは、不要な索引記憶域が消費されません。
- ロードおよび問合せのパフォーマンスを改善できます。

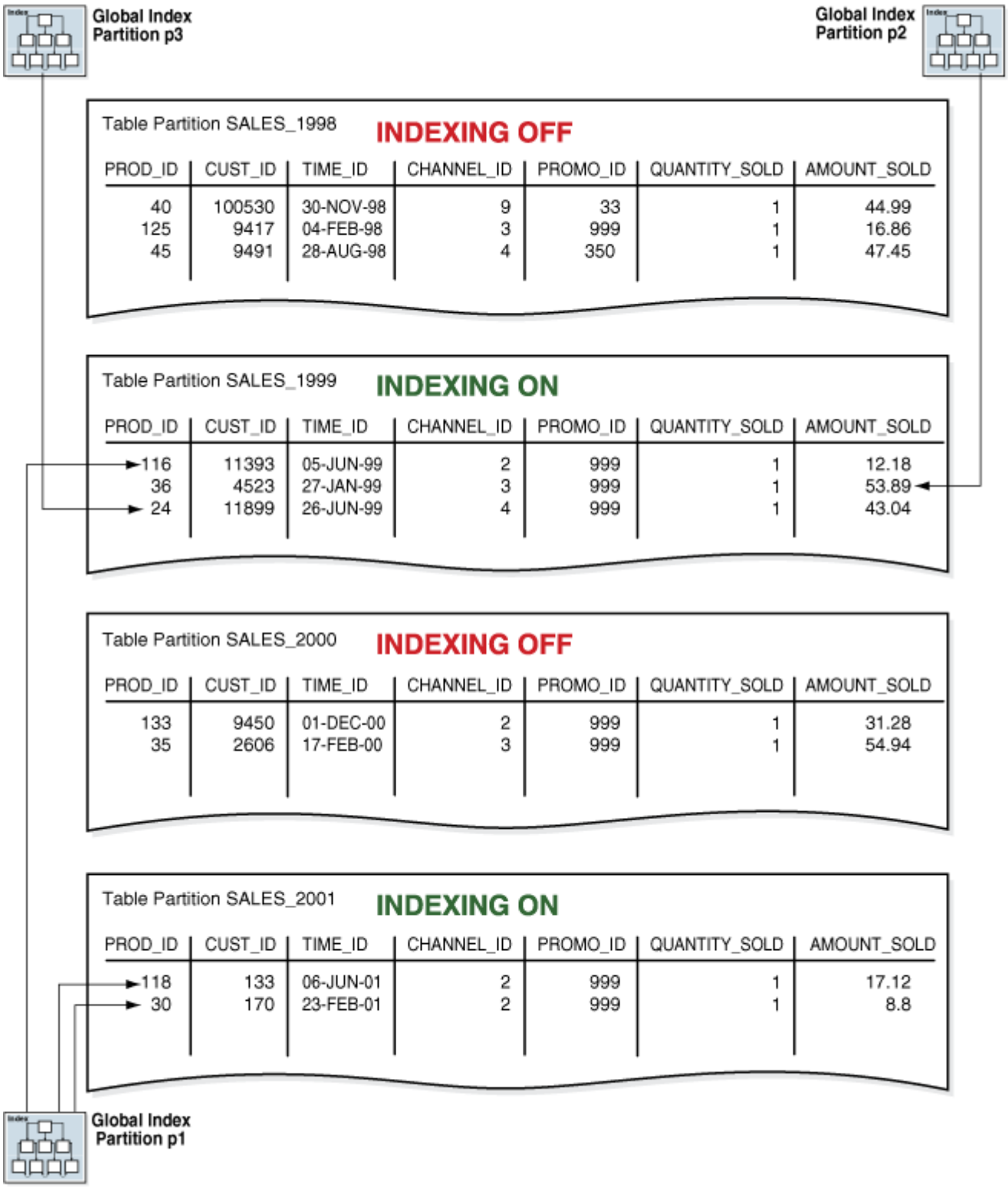
  Oracle Database 12cまでは、パーティションの交換操作には、関連付けられたグローバル索引を物理的に更新して、使用可能にしておく必要がありました。Oracle Database 12cからは、パーティションのメンテナンス操作に関係のあるパーティションが部分グローバル索引の一部でない場合、その索引はグローバル索引を維持しなくても、そのまま使用可能です。

- 索引作成時には一部の表パーティションにのみ索引を付け、後でその他のパーティションに索引を付けることで、索引の作成に必要なソート領域を削減できます。

表のパーティションごとに、索引付けを有効または無効にできます。部分ローカル索引では、索引付けが無効になっているすべての表パーティションに対して、使用可能な索引パーティションはありません。グローバル索引は、パーティション化されているかどうかにかかわらず、索引付けがオフになっているすべてのパーティションからデータを除外します。データベースでは、一意制約が強制される索引の部分索引はサポートされません。

図4-5に、グローバル索引が部分的である点を除いて、図4-4と同じグローバル索引を示します。表パーティションSALES_1998およびSALES_2000では、索引付けプロパティがOFFに設定されているため、部分グローバル索引によって索引は付けられません。

図4-5 部分グローバル・パーティション索引



シャード表の概要

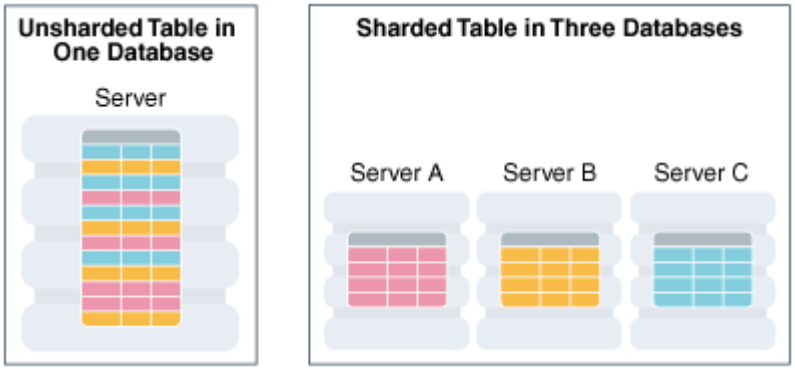
Oracleデータベースでは、シャーディングによって大規模な表を、複数のデータベースに格納可能なシャードと呼ばれる管理しやすい単位に分割できます。

各データベースは、独自のローカル・リソース(CPU、メモリー、フラッシュまたはディスク)を持つ専用サーバーでホストされます。このような構成の各データベースは**シャード**と呼ばれます。すべてのシャードが一体的に単一の論理データベースを形成し、これは**シャード・データベース**と呼ばれます。

水平パーティション化にはシャード間のデータベース表の分割が伴い、これによって各シャードには、列は同じですが行は異なるサブセットを持つ表が含まれます。この方法で分割された表は**シャード表**とも呼ばれています。

次の図は、3つのシャード間で水平パーティション化された表を示しています。

図4-6 シャード間への表の水平パーティション化



シャーディングはシェアードナッシング・ハードウェア・インフラストラクチャに基づいており、シャードはCPU、メモリー、ストレージ・デバイスなどの物理リソースを共有しないため、単一障害点が発生しません。シャードはソフトウェアの観点でも疎結合されています。クラスタウェアが実行されないためです。

通常、シャードは専用サーバーでホストされます。これらのサーバーには、市販のハードウェアまたはエンジニアド・システムを使用できます。シャードは、単一インスタンスまたはOracle RACデータベースで実行できます。それらは、オンプレミス、クラウド、またはオンプレミスとクラウドのハイブリッド構成に配置できます。

データベース管理者の視点からすると、SDBは、まとめて管理することも個別に管理することも可能な複数のデータベースで構成されています。ただし、アプリケーションの視点からはSDBは単一のデータベースのように見えます。シャードの数とそのシャード間におけるデータ分散は、データベース・アプリケーションに対して完全に透過的です。

シャーディングは、シャード・データベース・アーキテクチャに適したカスタムOLTPアプリケーションを対象としています。シャーディングを使用するアプリケーションには、シャーディング・キーを使用して主にデータにアクセスする適切に定義されたデータ・モデルおよびデータ分散方法(コンシステント・ハッシュ、範囲、リストまたはコンポジット)が必要です。シャーディング・キーの例として、customer_id、account_no、country_idなどがあります。

関連項目:

Oracle Shardingの使用

シャード表

シャード表は、複数のデータベース間でより小さい管理しやすい断片にパーティション化された表であり、シャードと呼ばれます。

Oracle ShardingはOracle Databaseのパーティション化機能に基づいて実装されています。Oracle Shardingは、シャード間の表パーティションの分散をサポートすることでパーティション化を拡張するため、事実上の分散パーティション化です。

パーティションはシャーディング・キーに基づいて表領域レベルでシャード間に分散されます。キーの例として顧客ID、アカウント番号、国IDなどがあります。

シャーディング・キーには次のデータ型がサポートされています。

- NUMBER
- INTEGER
- SMALLINT
- RAW
- (N)VARCHAR
- (N)VARCHAR2

- (N)CHAR
- DATE
- TIMESTAMP

シャード表の各パーティションが個別の表領域に存在し、各表領域が特定のシャードと関連付けられます。シャーディング方法に応じて、関連付けは自動的に確立されるか、管理者によって定義されることができます。

シャード表のパーティションが複数のシャード内に存在する場合でも、アプリケーションにとってその表は、単一データベース内のパーティション表とまったく同じように表示され、動作します。アプリケーションによって発行されたSQL文はシャードを参照する必要はなく、シャードの数およびその構成に依存することもありません。

例4-5 シャード表

表をパーティション化するためのいつものSQL構文によって、シャード間における表の行のパーティション化方法を指定します。たとえば、次のSQL文はシャーディング・キーcust_idに基づいて、シャード間で表を水平にパーティション化したシャード表を作成します。

```
CREATE SHARDED TABLE customers
( cust_id     NUMBER NOT NULL
, name        VARCHAR2(50)
, address     VARCHAR2(250)
, region      VARCHAR2(20)
, class       VARCHAR2(3)
, signup      DATE
CONSTRAINT cust_pk PRIMARY KEY(cust_id)
)
PARTITION BY CONSISTENT HASH (cust_id)
PARTITIONS AUTO
TABLESPACE SET ts1
;
```

前述の表はコンシステント・ハッシュ(スケーラブルな分散システムで一般的に使用される特殊なタイプのハッシュ・パーティション化)によりパーティション化されます。この手法は、シャード間で表領域を自動的に分散するため、データおよびワークロードが均等に分散されます。シャード表のグローバル索引はサポートされていませんが、ローカル索引はサポートされています。

表領域セット

Oracle Shardingは、**表領域セット**と呼ばれる単位として表領域を作成および管理します。PARTITIONS AUTO句は、パーティション数を自動的に判別することを指定します。このハッシュ・タイプによってシャード間のデータ移行でさらに柔軟性および効率性がもたらされ、これは弾力的な拡張性にとって重要です。

表領域はSDB内にデータを分散するための物理ユニットです。異なるシャードに存在する表領域にパーティションを自動的に作成することで、シャード間にパーティションを分散します。マルチシャード結合の数を最小にするために、関連する表の対応するパーティションは常に同じシャードに格納されます。シャード表の各パーティションが個別の表領域に格納されます。

ノート:

Oracle Managed Files のみが表領域セットでサポートされています。

個々の表領域を表領域セットと別個に削除または変更することはできません。

TABLESPACE SET はユーザー定義シャーディング・メソッドとともに使用することはできません。

関連トピック

- シャーディング・アーキテクチャ
- Oracle Shardingの使用

ビューの概要

ビューは、1つまたは複数の表の論理表現です。本質的には、ビューはストアド・クエリーです。

ビューは、実表と呼ばれる基礎となる表からデータを導出します。実表には、表または他のビューを指定できます。ビュー上で実行されるすべての操作は、実際にその実表に影響します。多くの場合、ビューは表と同じように使用できます。

ノート:

マテリアライズド・ビューは、標準のビューとは異なるデータ構造を使用します。

ビューを使用すると、様々なタイプのユーザーにあわせてデータの表現形式を変化させることができます。通常、ビューは次の目的で使用されます。

- 事前に定義された行や列の集合のみにアクセスを限定して、表のセキュリティ・レベルを強化します。

  たとえば、図4-7は、staffビューに実表employeesの列salaryまたはcommission_pctが含まれない仕組みを示しています。

- データの複雑さが隠されます

  たとえば、複数の表の関連した列または行を1つにまとめる結合によって、単一のビューを定義できます。ただし、ビューからは、この情報が複数の表から抽出されたものであるということはわかりません。1つの問合せで、表情報に対して複雑な計算を実行する場合もあります。これにより、ユーザーはビューに対して問い合せることができ、結合または計算の実行方法を知る必要がなくなります。

- 実表とは異なる視点からデータを提示します。

  たとえば、ビューが基礎にしている表に影響を与えずに、ビューの列名を変更できます。

- 実表の定義の変更からアプリケーションを分離します。

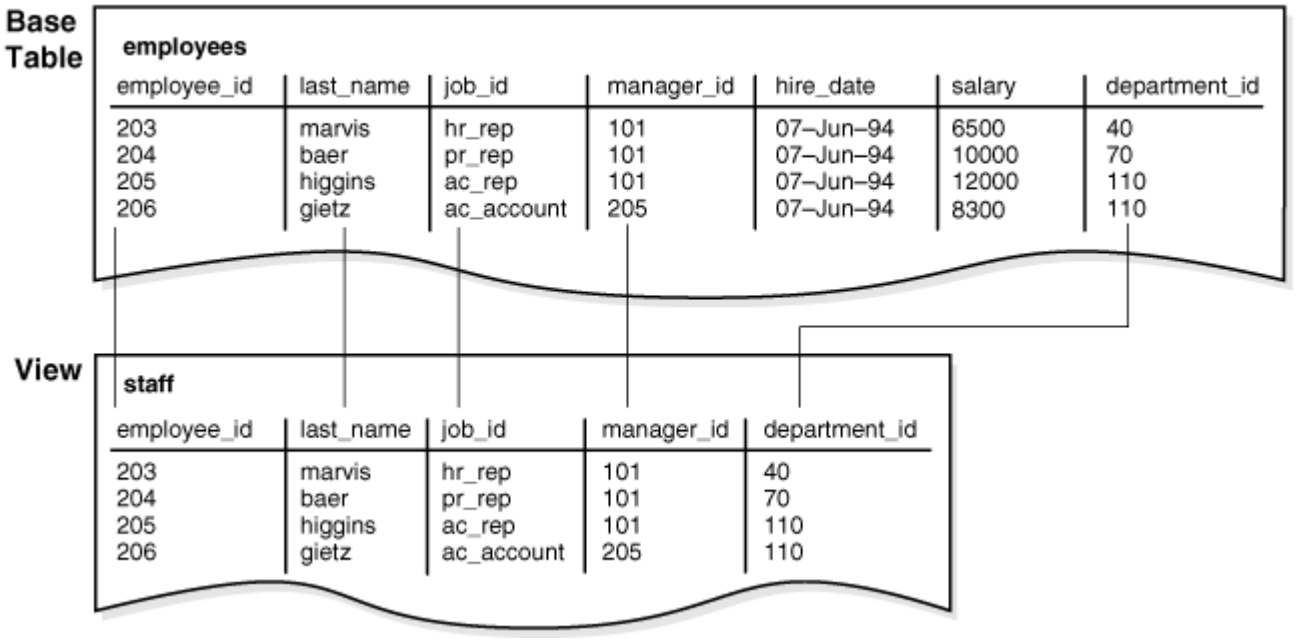
  たとえば、ビューを定義している問合せが表の4つの列の中の3つの列を参照している場合、たとえ5番目の列が追加されてもビューの定義は影響を受けないため、ビューを使用しているすべてのアプリケーションも影響を受けません。

ビューの使用例として、複数の列と膨大な行を持つhr.employees表について検討します。ユーザーがそのうちの5つの列または特定の行しか参照できないようにする場合は、次のようにビューを作成します。

```
CREATE VIEW staff AS
  SELECT employee_id, last_name, job_id, manager_id, department_id
  FROM   employees;
```

すべての副問合せと同様に、ビューを定義する問合せにFOR UPDATE句を含めることはできません。次の図に、staffという名前のビューを示します。ビューが、実表の5つの列のみを表示していることに注目してください。

図4-7 ビュー



関連項目:

- 「マテリアライズド・ビューの概要」
- ビューの管理方法を学習するには、『Oracle Database管理者ガイド』を参照してください
- CREATE VIEWの構文およびセマンティクスについては、『Oracle Database SQL言語リファレンス』を参照してください

ビューの特性

表とは異なり、ビューには記憶域は割り当てられず、データが格納されることもありません。ビューは、参照する実表からデータを抽出または導出する問合せによって定義されます。ビューは他のオブジェクトを基盤としているため、データ・ディクショナリ.内でビューを定義する問合せ用の記憶域以外には記憶域が必要ありません。

ビューは、データベースによって自動的に操作される参照先のオブジェクトに依存します。たとえば、ビューの実表を削除してから再作成した場合、データベースは新しい実表がビュー定義と一致するかどうかを確認します。

ビューのデータ操作

ビューは表から導出されるため、ビューと表には多数の類似点があります。ユーザーはビューを問い合せることができ、いくつかの制限付きでビューに対してDMLを実行できます。ビューに対して実行される操作は、そのビューの実表のデータに影響し、実表の整合性制約とトリガーの対象になります。

次の例では、hr.employees表のビューを作成します。

```
CREATE VIEW staff_dept_10 AS
SELECT employee_id, last_name, job_id,
       manager_id, department_id
FROM   employees
WHERE  department_id = 10
WITH CHECK OPTION CONSTRAINT staff_dept_10_cnst;
```

定義された問合せでは部門10の行のみを参照します。CHECK OPTIONは、ビューに対して発行されたINSERTおよびUPDATE文の結果が必ずビューで選択できる行になるよう、制約を持つビューを作成します。これにより、部門10の従業員の行は挿入できますが、部門30の行は挿入できません。

関連項目:

CREATE VIEW文の副問合せ制限について学習するには、『*Oracle Database SQL言語リファレンス*』を参照してください。

ビュー内のデータへのアクセス方法

Oracle Databaseでは、ビューの定義は、ビューを定義する問合せテキストとしてデータ・ディクショナリに保存されます。

SQL文でビューを参照すると、Oracle Databaseでは次のタスクが実行されます。

1. 可能な場合は常に、ビューに対する問合せとそのビューおよび基礎となるビューを定義する問合せをマージします。

   Oracle Databaseにより、マージされた問合せは、ビューを参照せずに問合せを発行したものとして最適化されます。そのため、Oracle Databaseでは、列がビュー定義で参照されても、またはビューに対するユーザーの問合せで参照されても、参照される実表の列に対する索引を使用できます。

   Oracle Databaseでは、ユーザーの問合せとビュー定義をマージできないこともあります。その場合、Oracle Databaseは参照された列の索引すべてを使用できるとはかぎりません。

2. 共有SQL領域内のマージ済の文を解析します。

   Oracle Databaseでは、新しい共有SQL領域内のビューを参照する文は、既存の共有SQL領域に同様の文が含まれていない場合にのみ解析されます。したがって、ビューを使用すると、共有SQLに関連付けられているメモリーの使用量を減らせるという利点があります。

3. SQL文を実行します。

次の例に、ビューが問合せを受ける際のデータ・アクセスを示します。employeesおよびdepartments表を基にemployees_viewを作成するとします。

```
CREATE VIEW employees_view AS
  SELECT employee_id, last_name, salary, location_id
  FROM   employees JOIN departments USING (department_id)
  WHERE  department_id = 10;
```

ユーザーはemployees_viewの次の問合せを実行します。

```
SELECT last_name
FROM   employees_view
WHERE  employee_id = 200;
```

Oracle Databaseはビューとユーザーの問合せをマージして、データ取得のために実行する次の問合せを構成します。

```
SELECT last_name
FROM   employees, departments
WHERE  employees.department_id = departments.department_id
AND    departments.department_id = 10
AND    employees.employee_id = 200;
```

関連項目:

- 共有SQL領域
- 「オプティマイザの概要」

- 問合せの最適化について学習するには、『Oracle Database SQLチューニング・ガイド』を参照

更新可能な結合ビュー

結合ビューは、FROM句に複数の表またはビューがあります。

次の例では、staff_dept_10_30ビューがemployees表とdepartments表を結合し、部門10または30の従業員のみを含めるように指定しています。

```
CREATE VIEW staff_dept_10_30 AS
SELECT employee_id, last_name, job_id, e.department_id
FROM   employees e, departments d
WHERE  e.department_id IN (10, 30)
AND    e.department_id = d.department_id;
```

更新可能な結合ビューは変更可能な結合ビューとも呼ばれ、2つ以上の実表またはビューを含み、DMLの各操作が可能です。更新可能なビューはSELECT文のトップレベルのFROM句に複数の表を含み、WITH READ ONLY句で制限されません。

本質的に更新可能であるためには、ビューは複数の条件を満たす必要があります。たとえば、結合ビューに対するINSERT、UPDATEまたはDELETE操作では、1回に変更可能な実表は1つのみであることが一般的なルールです。次のUSER_UPDATABLE_COLUMNSデータ・ディクショナリ・ビューの問合せは、staff_dept_10_30ビューが更新可能であることを示しています。

```
SQL> SELECT TABLE_NAME, COLUMN_NAME, UPDATABLE
  2  FROM   USER_UPDATABLE_COLUMNS
  3  WHERE  TABLE_NAME = 'STAFF_DEPT_10_30';

TABLE_NAME                     COLUMN_NAME                    UPD
------------------------------ ------------------------------ ---
STAFF_DEPT_10_30               EMPLOYEE_ID                    YES
STAFF_DEPT_10_30               LAST_NAME                      YES
STAFF_DEPT_10_30               JOB_ID                         YES
STAFF_DEPT_10_30               DEPARTMENT_ID                  YES
```

結合ビューの更新可能な列はすべて、キー保存表(その問合せの出力で、基礎となる表の各行が最大1回のみ表示される表)の列にマップする必要があります。staff_dept_10_30ビューでは、department_idがdepartments表の主キーであるため、employees表をキー保存表にして、employees表からの各行が結果セットに最高で1回のみ表示されます。departments表は、表内の各行が結果セットに複数回表示される可能性があるため、キー保存表ではありません。

関連項目:

結合ビューの更新方法を学習するには、Oracle Database管理者ガイドを参照してください。

オブジェクト・ビュー

ビューが仮想表であるのと同様に、**オブジェクト・ビュー**は仮想オブジェクト表です。ビュー内の各行はオブジェクトであり、**オブジェクト・タイプ**のインスタンスです。オブジェクト型はユーザー定義データ型です。

リレーショナル・データは、オブジェクト型として格納されているかのように取得、更新、挿入および削除できます。また、オブジェクト、REFおよびコレクション(ネストした表とVARRAY)などのオブジェクト・データ型である列を持つビューを定義できます。

リレーショナル・ビューと同様、オブジェクト・ビューには、データベース管理者がユーザーに対して表示するデータのみを表示できます。たとえば、オブジェクト・ビューにITプログラマについてのデータは表示し、給与に関する機密データは除外することが可能です。次の例では、employee_typeオブジェクトを作成してから、このオブジェクトを基にしたビューit_prog_viewを作成しています。

```
CREATE TYPE employee_type AS OBJECT
(
  employee_id  NUMBER (6),
  last_name    VARCHAR2 (25),
  job_id       VARCHAR2 (10)
);
/

CREATE VIEW it_prog_view OF employee_type
  WITH OBJECT IDENTIFIER (employee_id) AS
SELECT  e.employee_id, e.last_name, e.job_id
FROM    employees e
WHERE   job_id = 'IT_PROG';
```

オブジェクト・ビューは、ビュー内のデータをリレーショナル表から取得し、この表がオブジェクト表として定義されているものとしてデータにアクセスできるため、プロトタイプ化、およびオブジェクト指向アプリケーションへの移行に便利です。オブジェクト指向アプリケーションは、既存の表を別の物理構造に変換せずに実行できます。

関連項目:

- オブジェクト・タイプとオブジェクト・ビューについて学習するには、『*Oracle Databaseオブジェクト・リレーショナル開発者ガイド*』を参照してください。
- CREATE TYPE文について学習するには、『*Oracle Database SQL言語リファレンス*』を参照してください。

マテリアライズド・ビューの概要

マテリアライズド・ビューは、スキーマ・オブジェクトとして事前に格納またはマテリアライズされた問合せ結果です。問合せのFROM句では表、ビューまたはマテリアライズド・ビューの名前を指定できます。

マテリアライズド・ビューは、レプリケーションのマスター表およびデータ・ウェアハウスのファクト表として使用されることもよくあります。マテリアライズド・ビューは、データを集計、計算、レプリケートおよび分配します。これらは、次のような様々なコンピューティング環境に適しています。

- データ・ウェアハウスでは、マテリアライズド・ビューで、合計値や平均値など、集計関数から生成されたデータを計算し、格納できます。

  サマリーは集計ビューで、結合および集計操作を事前に計算して結果を表に格納することで、問合せ時間が短縮されます。マテリアライズド・ビューは、サマリーと同じものです。また、マテリアライズド・ビューを使用して、集計操作の有無に関係なく、結合を計算できます。

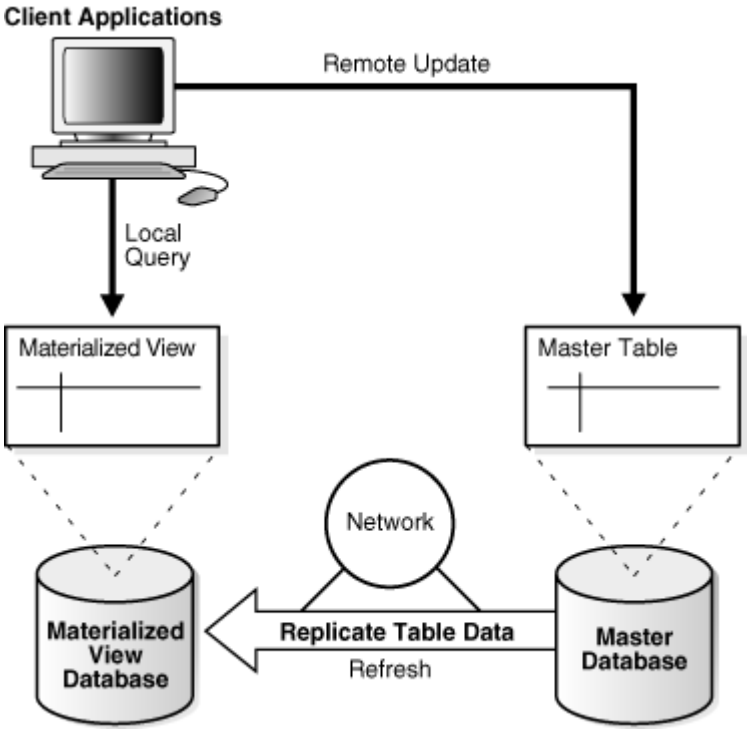
- XStreamおよびOracle GoldenGateの使用により実現されるマテリアライズド・ビュー・レプリケーションでは、ある特定の時点における表の完全または部分的なコピーがビューに格納されます。マテリアライズド・ビューは分散サイトにデータをレプリケートし、複数のサイトで実行される更新を同期化します。このレプリケーション形式は、データベースが常時ネットワークに接続されていない場合のフィールド・セールスなどの環境に適しています。

- モバイル・コンピューティング環境では、マテリアライズド・ビューで、中央のサーバーからモバイル・クライアントにデータのサブセットをダウンロードし、中央のサーバーから定期的にリフレッシュし、クライアントで実行された更新を中央のサーバー

に伝播できます。

レプリケーション環境では、マテリアライズド・ビューはマスター・データベースと呼ばれる別のデータベース内の表とデータを共有します。マテリアライズド・ビューに関連付けられたマスター・サイトの表がマスター表です。図4-8に、別のデータベース内のマスター表に基づく、1つのデータベース内のマテリアライズド・ビューを示します。マスター表を更新すると、マテリアライズド・ビューのデータベースにレプリケートされます。

図4-8 マテリアライズド・ビュー



関連項目:

- XStreamおよびOracle GoldenGateの使用によるレプリケーションについて学習するには、「情報の共有」を参照してください
- サマリーについてさらに学習するには、「データ・ウェアハウスのアーキテクチャ(基本)」を参照
- XStreamの概要は、『Oracle Database XStreamガイド』を参照してください
- Oracle GoldenGateについてさらに学習するには、http://www.oracle.com/technetwork/middleware/goldengate/documentation/index.htmlを参照してください
- CREATE MATERIALIZED VIEW文について学習するには、Oracle Database SQL言語リファレンスを参照してください

マテリアライズド・ビューの特性

マテリアライズド・ビューには、索引および非マテリアライズド・ビューと同じ特性がいくつかあります。

マテリアライズド・ビューは、次の点が索引に類似しています。

- 実際のデータを含み、記憶域を使用します。
- マスター表のデータに変更があった場合は、リフレッシュできます。

- クエリー・リライト操作に使用すると、SQLの実行効率が改善されます。
- その存在は、SQLアプリケーションとユーザーに対して透過的です。

マテリアライズド・ビューは他の表およびビューのデータを表す点において、非マテリアライズド・ビューに類似しています。索引とは異なり、ユーザーはSELECT文を使用してマテリアライズド・ビューを直接問合せできます。必要なリフレッシュのタイプによっては、DML文を使用してビューを更新することも可能です。

次の例では、shサンプル・スキーマの3つのマスター表に基づくマテリアライズド集計ビューを作成して、データを投入しています。

```
CREATE MATERIALIZED VIEW sales_mv AS
  SELECT t.calendar_year, p.prod_id, SUM(s.amount_sold) AS sum_sales
  FROM   times t, products p, sales s
  WHERE  t.time_id = s.time_id
  AND    p.prod_id = s.prod_id
  GROUP BY t.calendar_year, p.prod_id;
```

次の例では、sales_mvのマスター表であるsales表を削除してから、sales_mvを問い合せます。行はマスター表のデータとは別に格納(マテリアライズ)されているため、問合せによってデータが選択されます。

```
SQL> DROP TABLE sales;

Table dropped.

SQL> SELECT * FROM sales_mv WHERE ROWNUM < 4;

CALENDAR_YEAR    PROD_ID  SUM_SALES
------------- ---------- ----------
         1998         13  936197.53
         1998         26  567533.83
         1998         27  107968.24
```

マテリアライズド・ビューはパーティション化できます。マテリアライズド・ビューは、そのパーティション表および1つ以上の索引上で定義できます。

関連項目:

データ・ウェアハウスでマテリアライズド・ビューを使用する方法を学習するには、『Oracle Databaseデータ・ウェアハウス・ガイド』を参照してください。

マテリアライズド・ビューのリフレッシュ方法

データベースでは、実表の変更後にマテリアライズド・ビューをリフレッシュして、マテリアライズド・ビューのデータを管理します。リフレッシュ方法は、増分または完全リフレッシュです。

完全リフレッシュ

完全リフレッシュでは、マテリアライズド・ビューを定義する問合せが実行されます。マテリアライズド・ビューで事前作成表を参照するか、ユーザーが表をBUILD DEFERREDとして定義しないかぎり、完全リフレッシュは、マテリアライズド・ビューを最初に作成したときに実行されます。

完全リフレッシュは、特にデータベースで大量のデータを読み込んで処理する必要がある場合には、遅くなる可能性があります。マテリアライズド・ビューの作成後、いつでも完全リフレッシュを実行できます。

増分リフレッシュ

増分リフレッシュは、高速リフレッシュとも呼ばれ、既存データの変更のみを処理します。この方法では、マテリアライズド・ビューを最初から再作成する必要がなくなります。変更のみを処理するため、リフレッシュ時間が非常に速くなります。

マテリアライズド・ビューは、必要に応じて、または定期的な間隔で、リフレッシュできます。また、実表と同じデータベース内のマテリアライズド・ビューは、トランザクションによって実表の変更がコミットされるたびにリフレッシュするように構成できます。

高速リフレッシュは、次のいずれかの形式です。

- ログベースのリフレッシュ

  このタイプのリフレッシュでは、マテリアライズド・ビュー・ログまたはダイレクト・ローダー・ログに実表の変更が記録されます。マテリアライズド・ビュー・ログは、実表への変更を記録するスキーマ・オブジェクトであるため、実表で定義されているマテリアライズド・ビューを増分リフレッシュできます。マテリアライズド・ビュー・ログは、それぞれ1つの実表に関連付けられています。

- パーティション・チェンジ・トラッキング(PCT)リフレッシュ

  PCTリフレッシュは、実表がパーティション化されている場合のみ有効です。PCTリフレッシュでは、影響を受けるマテリアライズド・ビューのパーティションのすべてのデータ、または影響を受ける部分のデータを削除し、それらを再計算します。データベースでは、変更された実表パーティションを使用して、ビュー内で影響を受けるパーティションまたはデータの一部を識別します。パーティションの管理操作が実表で行われている場合、使用できる増分リフレッシュの方法はPCTリフレッシュのみです。

インプレース・リフレッシュおよびホーム外リフレッシュ

完全および増分方法の場合、データベースでは、インプレース(ビューに対して文を直接リフレッシュする)またはホーム外でマテリアライズド・ビューをリフレッシュできます。

ホーム外リフレッシュでは、1つ以上の外部表が作成され、そこでリフレッシュ文が実行されて、マテリアライズド・ビューまたは影響を受けるパーティションが外部表に切り替えられます。この方法により、リフレッシュ時、特にリフレッシュ文の終了まで長くかかる場合に、高可用性が実現します。

Oracle Database 12cでは、ホーム外フレッシュの1種である同期フレッシュが導入されました。同期リフレッシュでは、実表の内容は変更されませんが、かわりに同期リフレッシュ・パッケージのAPIを使用することで、これらの変更を実表とマテリアライズド・ビューに同時に適用して、一貫性を確保します。この方法により、一連の表とそれらで定義されているマテリアライズド・ビューを常に同期できます。データ・ウェアハウスでは、同期リフレッシュの方法が次の理由で適しています。

- 増分データのロードは、しっかり制御され、定期的に実行されます。
- 表およびそれらのマテリアライズド・ビューでは、多くの場合、同じようにパーティション化されるか、またはパーティションが機能依存性で関連付けられます。

関連項目:

マテリアライズド・ビューのリフレッシュ方法を学習するには、『Oracle Databaseデータ・ウェアハウス・ガイド』を参照してください。

クエリー・リライト

クエリー・リライトはマスター表について記述されたユーザー要求を、マテリアライズド・ビューを含む同等の要求に変換します。

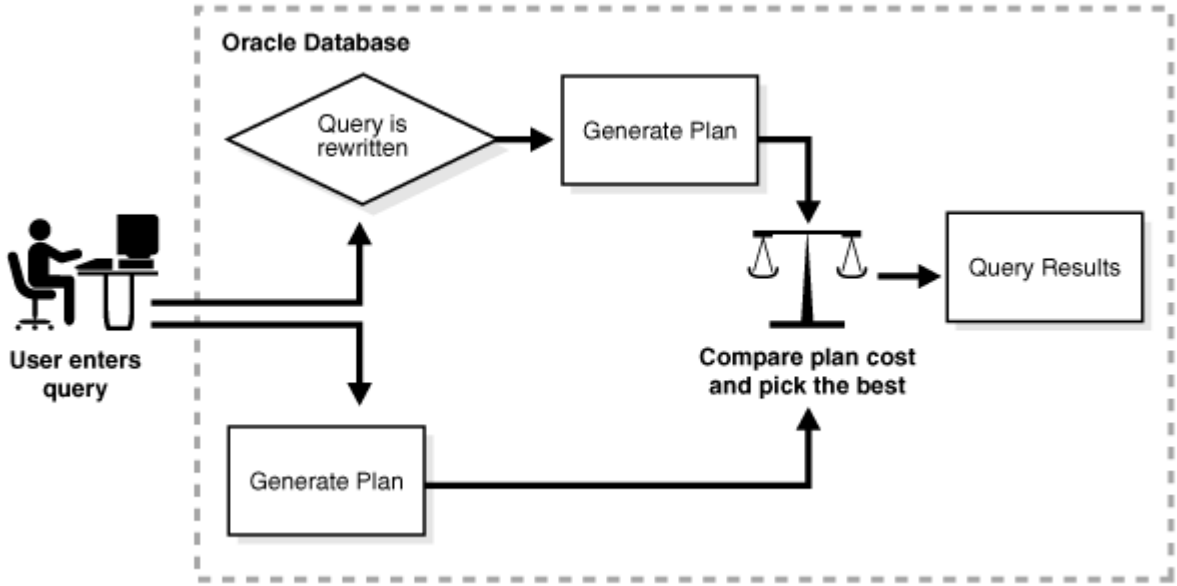
実表に大量のデータが含まれている場合には、集計および結合の計算は負荷が大きく、時間がかかります。マテリアライズド・

ビューには事前に計算された集計と結合が含まれているため、クエリー・リライトはマテリアライズド・ビューを使用してすぐに問合せに応答できます。

問合せトランスフォーマは、マテリアライズド・ビューを使用するために透過的にリクエストをリライトし、ユーザー操作およびSQL文によるマテリアライズド・ビューへの参照は必要ありません。クエリー・リライトは透過的であるため、アプリケーション・コードのSQLを無効にしなくても、マテリアライズド・ビューを追加または削除できます。

一般に、ディテール表ではなくマテリアライズド・ビューを使用するように問合せをリライトすると、応答のパフォーマンスが改善されます。次の図は、オリジナルまたはリライトされた問合せに対してデータベースが実行計画を生成し、最もコストの低い計画を選択する仕組みを示しています。

図4-9 クエリー・リライト



関連項目:

- 問合せ変換についてさらに学習するには、「オプティマイザの概要」を参照してください
- クエリー・リライトの使用方法を学習するには、『Oracle Databaseデータ・ウェアハウス・ガイド』を参照してください

順序の概要

順序とは、複数のユーザーが一意の整数を生成できるスキーマ・オブジェクトです。順序ジェネレータは、数値データ型に対してサロゲート・キーを生成するスケーラブルかつパフォーマンスの高い方法を提供します。

順序の特性

順序の定義には、順序の名前、順序が昇順か降順かなど、順序に関する一般的な情報を指定します。

順序の定義では、次についても指定します。

- 数値の増分値
- データベースが、メモリー内に生成された一連の順序番号をキャッシュするかどうか
- 制限に到達したときに、順序を循環させるかどうか

次の例では、サンプル・スキーマoeに順序customers_seqを作成します。customers表に行が追加されると、アプリケーションは

この順序を使用して、顧客ID番号を付与します。

```
CREATE SEQUENCE customers_seq
START WITH      1000
INCREMENT BY    1
NOCACHE
NOCYCLE;
```

最初のcustomers_seq.nextvalの参照では、1000が戻されます。2回目には1001が戻されます。以降の各参照では、前回の参照よりも1大きい値が戻されます。

関連項目:

- 順序の作成方法を示すチュートリアルは、『Oracle Database 2日で開発者ガイド』を参照
- SQL文で順序を参照する方法を学習するには、『Oracle Database管理者ガイド』を参照
- CREATE SEQUENCEの構文およびセマンティクスについては、『Oracle Database SQL言語リファレンス』を参照

順序への同時アクセス

同じ順序ジェネレータから、複数の表に対して数値を生成できます。

ジェネレータから自動的に主キーを作成して、複数の行または表にまたがるキーを統合できます。たとえば、順序はorders表およびcustomers表の主キーを生成できます。

順序ジェネレータは、ディスクI/Oのオーバーヘッドまたはトランザクション・ロックを発生させずに、一意の番号を生成するため、マルチ・ユーザー環境で便利です。たとえば、2人のユーザーがorders表に新しい行を同時に挿入するとします。順序を使用してorder_id列に一意の番号を生成すると、一方のユーザーは、他方のユーザーが注文番号を入力するのを待機する必要はありません。どちらのユーザーにも、順序により、正しい値が自動的に生成されるためです。

順序を参照する各ユーザーは、セッションで生成された最後の順序である現行の順序番号にアクセスできます。ユーザーは、文を発行して新しい順序番号を生成することも、セッションで最後に生成された現行の番号を使用することもできます。セッション内の文が順序番号を生成すると、その番号はこのセッションでのみ使用可能となります。最終的にロールバックされたトランザクションで生成されて使用された個々の順序番号はスキップされます。

警告:

アプリケーションが連続する番号を使用する場合、Oracle の順序は使用できません。独自に開発したコードを使用して、データベースのアクティビティをシリアライズする必要があります。

関連項目:

セッションが同時にデータにアクセスする方法を学習するには、「データの同時実行性と整合性」を参照

ディメンションの概要

データ・ウェアハウスには通常、ディメンションとファクトという2つの重要な要素があります。

ディメンションは、時刻、地理、製品、部門および販売チャネルなどのビジネス上の質問を指定する場合に使用される任意のカテゴリです。**ファクト**は、売上単位または利益などの特定のディメンション値のセットに関連付けられたイベントまたはエンティティです。

多次元要求の例には、次の処理が含まれます。

- 2013年と2014年について、州、国、地域というように地理ディメンションの集計レベルの昇順で全製品の総売上を表示します。
- 2013年と2014年における南アメリカの地域別経費を示す事業のクロス集計分析を作成します。可能な小計をすべて組み込みます。
- 自動車製品に関する2014年の販売収入に従って、アジアでの販売代理店の上位10社をリストし、そのコミッションのランキングを作成します。

多次元による多くの質問では、通常は時間、地理または予算にまたがって集計されたデータと、データ・セットの比較が必要になります。

ディメンションを作成すると、より広範囲でのクエリー・リライト機能の使用が許可されます。データベースでは、マテリアライズド・ビューを使用した透過的なクエリー・リライトによって、問合せのパフォーマンスを改善できます。

関連項目:

データ・ウェアハウスとOLTPデータベースの相違点について学習するには、「データ・ウェアハウスとビジネス・インテリジェンスの概要」を参照

ディメンションの階層構造

ディメンション表は、列または列セットのペア間の階層(親/子)関係を定義する論理構造です。

たとえば、1行内でcity列がstate列の値を示し、state列がcountry列の値を示すなど、ディメンションで指定できます。

顧客ディメンション内では、顧客は都市、州、国、準地域、地域までロールアップできます。通常、データ分析はディメンション階層の高いレベルから開始され、必要に応じて徐々に階層がドリルダウンされます。

子レベルの値は、それぞれが親レベルの1つの値にのみ対応付けられます。階層関係とは、階層内の1つのレベルから次のレベルに対する機能的な依存関係です。

ディメンションには、この論理構造に割り当てるデータ記憶域がありません。ディメンション情報がディメンション表に格納されるのに対して、ファクト情報はファクト表に格納されます。

関連項目:

- ディメンションについて学習するには、Oracle Databaseデータ・ウェアハウス・ガイドを参照してください。
- ディメンションの作成方法を学習するには、Oracle OLAPユーザーズ・ガイドを参照してください。

ディメンションの作成

ディメンションの作成では、CREATE DIMENSION SQL文を使用します。

この文では、次について指定します。

- 複数のLEVEL句。それぞれがディメンション内の1列または列の集合を識別します。
- 1つ以上のHIERARCHY句。隣接するレベル間の親子関係を指定します。
- オプションのATTRIBUTE句。それぞれが個々のレベルに対応付けられている他の列または列セットを識別します。

次の文は、サンプル・スキーマshにcustomers_dimディメンションを作成するために使用されました。

```
CREATE DIMENSION customers_dim
   LEVEL customer   IS (customers.cust_id)
   LEVEL city       IS (customers.cust_city)
   LEVEL state      IS (customers.cust_state_province)
   LEVEL country    IS (countries.country_id)
   LEVEL subregion  IS (countries.country_subregion)
   LEVEL region     IS (countries.country_region)
   HIERARCHY geog_rollup (
      customer      CHILD OF
      city          CHILD OF
      state         CHILD OF
      country       CHILD OF
      subregion     CHILD OF
      region
   JOIN KEY (customers.country_id) REFERENCES country )
   ATTRIBUTE customer DETERMINES
   (cust_first_name, cust_last_name, cust_gender,
    cust_marital_status, cust_year_of_birth,
    cust_income_level, cust_credit_limit)
   ATTRIBUTE country DETERMINES (countries.country_name);
```

ディメンション内の列は、同じ表のもの(非正規化)でも複数の表のもの(完全正規化または一部正規化)でもかまいません。たとえば、正規化されているtimeディメンションには、date表、month表およびyear表と、各date行をmonth行に、各month行をyear行に接続する結合条件を含めることができます。完全な非正規化のtimeディメンションでは、date、monthおよびyear列は同じ表に格納されます。正規化されているかどうかに関係なく、列相互の階層関係をCREATE DIMENSION文で指定する必要があります。

関連項目:

CREATE DIMENSIONの構文およびセマンティクスについては、『Oracle Database SQL言語リファレンス』を参照してください

シノニムの概要

シノニムとは、スキーマ・オブジェクトの別名です。たとえば、表またはビュー、順序、PL/SQLプログラム単位、ユーザー定義オブジェクト型または別のシノニムなどに対してシノニムを作成できます。シノニムは単なる別名であるため、データ・ディクショナリ内の定義以外に記憶域は必要ありません。

シノニムを使用すると、データベース・ユーザー側のSQL文を単純化できます。また、シノニムは、基礎となるスキーマ・オブジェクトのIDおよび場所を非表示にする場合にも便利です。基礎となるオブジェクトを改名または移動する必要がある場合、シノニムのみを再定義すればよいことになります。シノニムに基づくアプリケーションは、今後も変更なしに機能し続けることができます。

プライベート・シノニムとパブリック・シノニムの両方を作成できます。プライベート・シノニムは、特定のユーザーのスキーマに含まれており、そのユーザーが他のユーザーに対するシノニムの使用許可を制御します。パブリック・シノニムはPUBLICという名前の特別なユーザー・グループが所有し、すべてのデータベース・ユーザーがアクセスできます。

例4-6 パブリック・シノニム

データベース管理者がhr.employees表のpeopleという名前のパブリック・シノニムを作成するとします。その後、ユーザーがoeスキーマに接続して、シノニムが参照する表内の行数をカウントしています。

```
SQL> CREATE PUBLIC SYNONYM people FOR hr.employees;

Synonym created.

SQL> CONNECT oe
Enter password: password
Connected.

SQL> SELECT COUNT(*) FROM people;

  COUNT(*)
----------
       107
```

パブリック・シノニムではデータベースの統合が困難になるため、パブリック・シノニムは慎重に使用してください。次の例に示すように、データベース内に存在するパブリック・シノニムpeopleは1つのみであるため、別の管理者がパブリック・シノニムpeopleの作成を試行すると失敗します。パブリック・シノニムを多用すると、アプリケーション間でネームスペースに競合が発生します。

```
SQL> CREATE PUBLIC SYNONYM people FOR oe.customers;
CREATE PUBLIC SYNONYM people FOR oe.customers
                       *
ERROR at line 1:
ORA-00955: name is already used by an existing object

SQL> SELECT OWNER, SYNONYM_NAME, TABLE_OWNER, TABLE_NAME
  2  FROM DBA_SYNONYMS
  3  WHERE SYNONYM_NAME = 'PEOPLE';

OWNER      SYNONYM_NAME TABLE_OWNER TABLE_NAME
---------- ------------ ----------- ----------
PUBLIC     PEOPLE       HR          EMPLOYEES
```

シノニム自体は安全ではありません。シノニムに対してオブジェクト権限を付与すると、実際には基礎となるオブジェクトに権限を付与することになります。シノニムは、GRANT文のオブジェクトの別名としてのみ機能しています。

関連項目:

- シノニムの管理方法を学習するには、『*Oracle Database*管理者ガイド』を参照してください。
- CREATE SYNONYMの構文およびセマンティクスについては、『*Oracle Database SQL*言語リファレンス』を参照してください。

5 データ整合性

この章では、整合性制約によって、データベースに関連するビジネス・ルールがどのように規定され、表への無効なエントリがどのように防止されるかについて説明します。

この章の構成は、次のとおりです。

- データ整合性の概要
- 整合性制約のタイプ
- 整合性制約の状態

関連項目:

列の背景と整合性制約の必要性については、「表の概要」を参照

データ整合性の概要

データベース管理者やアプリケーション開発者が決定したビジネス・ルールに準拠し、**データ整合性**を維持するのは重要なことです。

ビジネス・ルールでは、常に真(true)となる必要がある、または常に偽(false)となる必要がある条件および関係を定義します。たとえば、各企業では、給与、従業員番号、在庫追跡などに関して独自のポリシーを定義します。

データ整合性を保証する方法

データベース・アプリケーションを設計する場合、開発者は、データベースに格納するデータの整合性を保証するためのいくつかのオプションを使用できます。

このオプションには次のものがあります。

- トリガーされるストアド・データベース・プロシージャを使用して、ビジネス・ルールを規定します
- ストアド・プロシージャを使用してデータへのアクセスを完全に制御します
- データベース・アプリケーションのコードでビジネス・ルールを規定します。
- 列またはオブジェクト・レベルで定義される、データベースの値を制限する規則である、Oracle Databaseの整合性制約を使用します。

関連項目:

- 「トリガーの概要」では、トリガーの目的およびタイプについて説明します
- 「サーバー側プログラミングの概要」では、ストアド・プロシージャの目的および特性について説明します

整合性制約の利点

整合性制約は、SQLを使用して作成および削除されるスキーマ・オブジェクトです。データ整合性を維持するために、可能な場

合は常に、整合性制約を使用するようにしてください。

データ整合性を維持する方法として、整合性制約には、次のような利点があります。

- 宣言の容易さ

  SQL文を使用して整合性制約を定義するため、追加のプログラミングなしに表を定義または変更できます。SQL文は記述が容易で、プログラミングのエラーを避けられます。

- 規則の集中化

  整合性制約は表に対して定義され、データ・ディクショナリに格納されます。したがって、すべてのアプリケーションから入力されるデータは、同じ整合性制約を遵守する必要があります。規則が表レベルで変化した場合も、アプリケーションを変更する必要はありません。また、データベースによってSQL文がチェックされる前に、アプリケーションがデータ・ディクショナリのメタデータを使用して、すぐにユーザーに違反を通知することもできます。

- データをロードする場合の柔軟性

  大量のデータをロードする場合は、パフォーマンスのオーバーヘッドを回避するために、整合性制約を一時的に無効にできます。データ・ロードが完了した後、整合性制約を再度有効にできます。

関連項目:

- 「データ・ディクショナリの概要」
- データ整合性を保つ方法を学習するには、Oracle Database 2日で開発者ガイドおよびOracle Database開発ガイドを参照してください
- 整合性制約の管理方法を学習するには、Oracle Database管理者ガイドを参照してください

整合性制約のタイプ

Oracle Databaseでは、表と列の両方のレベルで制約を適用できます。

列または属性の一部として指定される制約が、表内指定です。表定義の一部として指定される制約が、表外指定です。

キーとは、特定タイプの整合性制約の定義に含まれる列または列の集合です。キーによって、リレーショナル・データベースの表と列の間の関連が示されます。キーの中にある個々の値を、キー値と呼びます。

次の表に、制約のタイプを示します。表内指定であることが必要なNOT NULLを除き、どの制約も表内と表外のいずれかで指定できます。

表5-1 整合性制約のタイプ

制約タイプ	説明	関連項目
NOT NULL	指定した列での NULL を含む行の挿入または更新を許可または禁止します。	「NOT NULL 整合性制約」
一意キー	同じ列または列の組合せで、複数の行が同じ値を持つことを禁止しますが、一部の値が NULL であることを許可しま	「一意制約」

制約タイプ	説明	関連項目
	す。	
主キー	NOT NULL 制約と一意制約を結合します。同じ列または列の組合せで、複数の行が同じ値を持つことを禁止し、値が NULL であることを禁止します。	「主キー制約」
外部キー	列を外部キーに指定し、外部キーと、参照キーと呼ばれる主キーまたは一意キーの間の関係を確立します。	外部キー制約
チェック	指定した条件をデータベース値が満たしている必要があります。	チェック制約
REF	REF 列の値で許可されるデータ操作のタイプ、およびその操作の結果として依存データが受ける影響について指示します。オブジェクト・リレーショナル・データベースでは、REF と呼ばれる組込みデータ型によって、指定したオブジェクト型の行オブジェクトへの参照がカプセル化されます。REF 列に対する参照整合性制約によって、REF の行オブジェクトがあることが保証されます。	REF 制約について学習するには、『Oracle Database オブジェクト・リレーショナル開発者ガイド』を参照してください。

関連項目:

- 「表の概要」
- 制約のタイプについてさらに学習するには、『Oracle Database SQL言語リファレンス』を参照してください。

NOT NULL整合性制約

NOT NULL制約がある場合、表の列値にNULLを使用できません。**NULL**は、値がないことを示します。デフォルトでは、表のすべての列でNULLを使用できます。

NOT NULL制約は、値が必須である列のためのものです。たとえば、hr.employees表ではemail列に値が必要です。電子メール・アドレスなしで従業員の行を挿入しようとすると、次のエラーが発生します。

```
SQL> INSERT INTO hr.employees (employee_id, last_name) values (999, 'Smith');
.
.
.
ERROR at line 1:
```

```
ORA-01400: cannot insert NULL into ("HR"."EMPLOYEES"."EMAIL")
```

NOT NULL制約を持つ列は、行が1つも含まれていない表またはデフォルト値が指定された表にのみ追加できます。

関連項目:

- NOT NULL制約を表に追加する例については、Oracle Database 2日で開発者ガイドを参照してください。
- NOT NULL制約を使用する場合の制限については、Oracle Database SQL言語リファレンスを参照してください。
- NOT NULL制約をいつ使用するかを学習するには、Oracle Database開発ガイドを参照してください。

一意制約

一意キー制約では、列(または列の集合)のすべての値が一意である必要があります。一意キー制約が設定されていると、表の行では、列(**一意キー**)または列の集合(**コンポジット一意キー**)で重複する値を持つことはありません。

ノート:

キーという用語は、整合性制約で定義されている列のみを指します。データベースではキー列に対して索引を暗黙的に作成または再利用することで一意制約による規制を規定するため、一意キーという用語は一意キー制約または一意索引のシノニムとして誤用されることがあります。

一意キー制約は、任意の列に重複する値が格納されないようにする場合に適しています。一意制約は、(各表の行を一意に識別することが目的である)主キー制約とは異なり、通常は、一意であること以外の意味を持たない値が含まれています。一意キーの例には次のものがあります。

- 主キーが顧客番号である場合の、顧客電話番号
- 主キーが部門番号である場合の、部門名

例2-1に示すように、hr.employees表のemail列には一意キー制約が存在します。文の関連する部分は、次のとおりです。

```
CREATE TABLE employees    ( ...
    , email          VARCHAR2(25)
        CONSTRAINT   emp_email_nn  NOT NULL ...
    , CONSTRAINT     emp_email_uk  UNIQUE (email) ... );
```

次の例に示すように、emp_email_uk制約により、2人の従業員が同じ電子メール・アドレスを持つことはありません。

```
SQL> SELECT employee_id, last_name, email FROM employees WHERE email = 'PFAY';

EMPLOYEE_ID LAST_NAME                 EMAIL
----------- ------------------------- -------------------------
        202 Fay                       PFAY

SQL> INSERT INTO employees (employee_id, last_name, email, hire_date, job_id)
  1  VALUES (999,'Fay','PFAY',SYSDATE,'ST_CLERK');
.
.
.
ERROR at line 1:
```

```
ORA-00001: unique constraint (HR.EMP_EMAIL_UK) violated
```

NOT NULL制約も定義されている場合を除き、一意キー制約では、NULLは常に許可されます。このため、一意キー制約とNOT NULL制約の両方を持つ列が一般的に使用されます。これらの組合せにより、ユーザーは一意キーに必ず値を入力することになり、さらに新しい行データが既存の行データと競合することがなくなります。

ノート:

複数の列に対する一意キー制約の検索メカニズムにより、一部が NULL のコンポジット一意キー制約の非 NULL 列で同一の値は許されません。

例5-1 一意制約

```
SQL> SELECT employee_id, last_name, email FROM employees WHERE email = 'PFAY';

EMPLOYEE_ID LAST_NAME                 EMAIL
----------- ------------------------- -------------------------
        202 Fay                       PFAY

SQL> INSERT INTO employees (employee_id, last_name, email, hire_date, job_id)
  1  VALUES (999,'Fay','PFAY',SYSDATE,'ST_CLERK');
.
.
.
ERROR at line 1:
ORA-00001: unique constraint (HR.EMP_EMAIL_UK) violated
```

関連項目:

- 「一意索引と非一意索引」
- UNIQUE制約を表に追加する例については、『*Oracle Database 2日で開発者ガイド*』を参照してください。

主キー制約

主キー制約を設定すると、この制約の対象となる1つ以上の列グループの値によって、行が一意に識別されます。各表には1つの**主キー**を設定でき、この主キーによって行が識別され、行が重複しないことが保証されます。

主キーには、自然キーまたはサロゲート・キーを使用できます。自然キーは、表内の既存の属性から作成される、意味のある識別子です。たとえば、参照表の郵便番号を自然キーとして使用できます。これに対して、サロゲート・キーは、システムで生成され、値が増加する識別子で、この識別子によって表内での一意性が保たれます。通常、サロゲート・キーは順序によって生成されます。

Oracle Databaseでは、主キー制約の実装により、次のことが保証されます。

- 指定された列または列の集合の中に、2つの行が重複する値を持つことはありません。
- 主キー列では、NULLは許可されません。

主キーが必要となる典型的な状況は、従業員用の数値識別子です。各従業員は、一意のIDを持つ必要があります。従業員は、employees表の1つの行によってのみ記述する必要があります。

「一意制約」の例では、既存の従業員が従業員IDとして202を持っており、従業員IDが主キーであることが示されています。

次に示す例では、同じ従業員IDを持つ従業員、およびIDを持たない従業員を追加します。

```
SQL> INSERT INTO employees (employee_id, last_name, email, hire_date, job_id)
  1  VALUES (202,'Chan','JCHAN',SYSDATE,'ST_CLERK');
.
.
.
ERROR at line 1:
ORA-00001: unique constraint (HR.EMP_EMP_ID_PK) violated

SQL> INSERT INTO employees (last_name) VALUES ('Chan');
.
.
.
ERROR at line 1:
ORA-01400: cannot insert NULL into ("HR"."EMPLOYEES"."EMPLOYEE_ID")
```

データベースにより、索引を持つ主キー制約が規定されます。通常、列に対して作成された主キー制約によって、暗黙的に一意索引とNOT NULL制約が作成されます。この規則には、次の例外がある点に注意してください。

- 場合によっては、遅延可能制約を使用して主キーを作成する場合、生成された索引は一意ではありません。

ノート:

CREATE UNIQUE INDEX 文を使用すると、一意の索引を明示的に作成できます。

- 主キー制約の作成時に使用できる索引がある場合、制約ではこの索引を再利用し、索引を暗黙的に作成することはありません。

デフォルトでは、暗黙的に作成された索引の名前が、主キー制約の名前になります。索引にはユーザー定義名を指定することもできます。制約の作成に使用するCREATE TABLE文またはALTER TABLE文にENABLE句を組み込んで、索引の記憶域オプションを指定できます。

関連項目:

表に主キー制約を追加する方法を学習するには、Oracle Database 2日で開発者ガイドおよびOracle Database開発ガイドを参照してください

外部キー制約

2つの表に1つ以上の共通の列が含まれる場合、Oracle Databaseでは、参照整合性制約とも呼ばれる**外部キー制約**を介して2つの表の間の関係を規定できます。

外部キー制約では、制約が定義される列の各値に対して、他の表と列を指定するもう一方の列の値が一致する必要があります。参照整合性規則の例としては、既存の部門にのみ従事できる従業員があります。

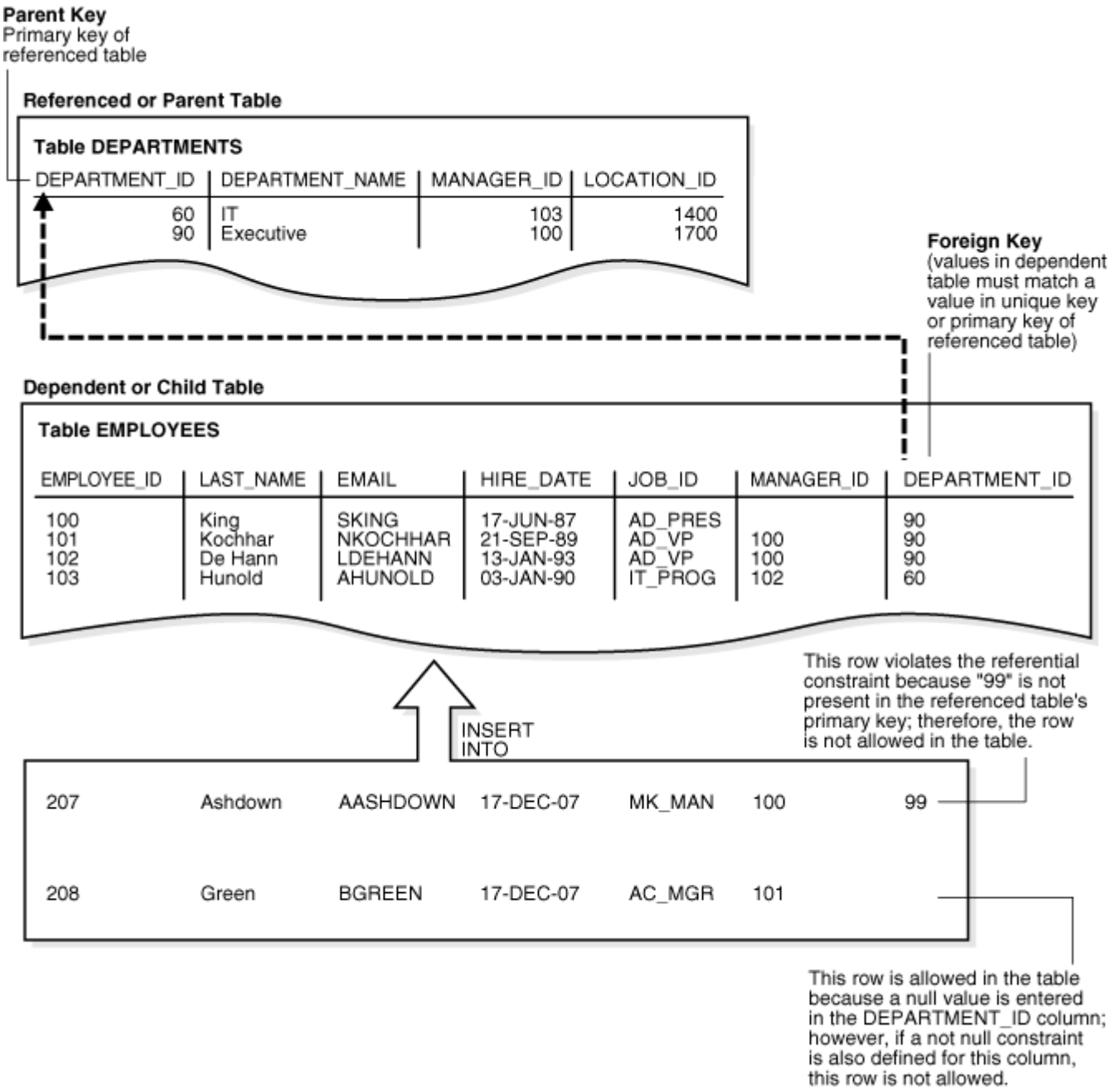
次の表に、参照整合性制約に関連する用語を示します。

表5-2 参照整合性制約の用語

用語	定義
外部キー	制約の定義に含まれている列または列の集合のうち、参照キーを参照するもの。たとえば、employees の department_id 列は外部キーであり、departments の department_id 列を参照します。 外部キーは、複数列として定義できます。ただし、コンポジット外部キーの列数とデータ型は、参照先のコンポジット主キーまたは一意キーと同じであることが必要です。 外部キーの値は、参照先の主キーまたは一意キーの値と一致するか、NULL であることが可能です。コンポジット外部キーのいずれかの列が NULL の場合、そのキーの NULL 以外の部分は、親キーの対応部分と一致している必要はありません。
参照キー	表の一意キーまたは主キーで、外部キーによって参照されるキー。たとえば、departments の department_id 列は、employees の department_id 列の参照キーです。
依存表または子表	外部キーを含む表。この表は、参照される一意キーまたは主キーにある値に依存しています。たとえば、employees 表は departments の子です。
参照表または親表	子表の外部キーが参照する表。この表の参照キーによって、子表に対する特定の挿入または更新が許可されるかどうかが決まります。たとえば、departments 表は employees の親です。

図5-1に、employees.department_id列の外部キーを示します。この列のすべての値が、departments.department_id列の値と一致することが保証されます。このため、employees.department_id列に間違った部門番号が存在することはありません。

図5-1 参照整合性制約



関連項目:

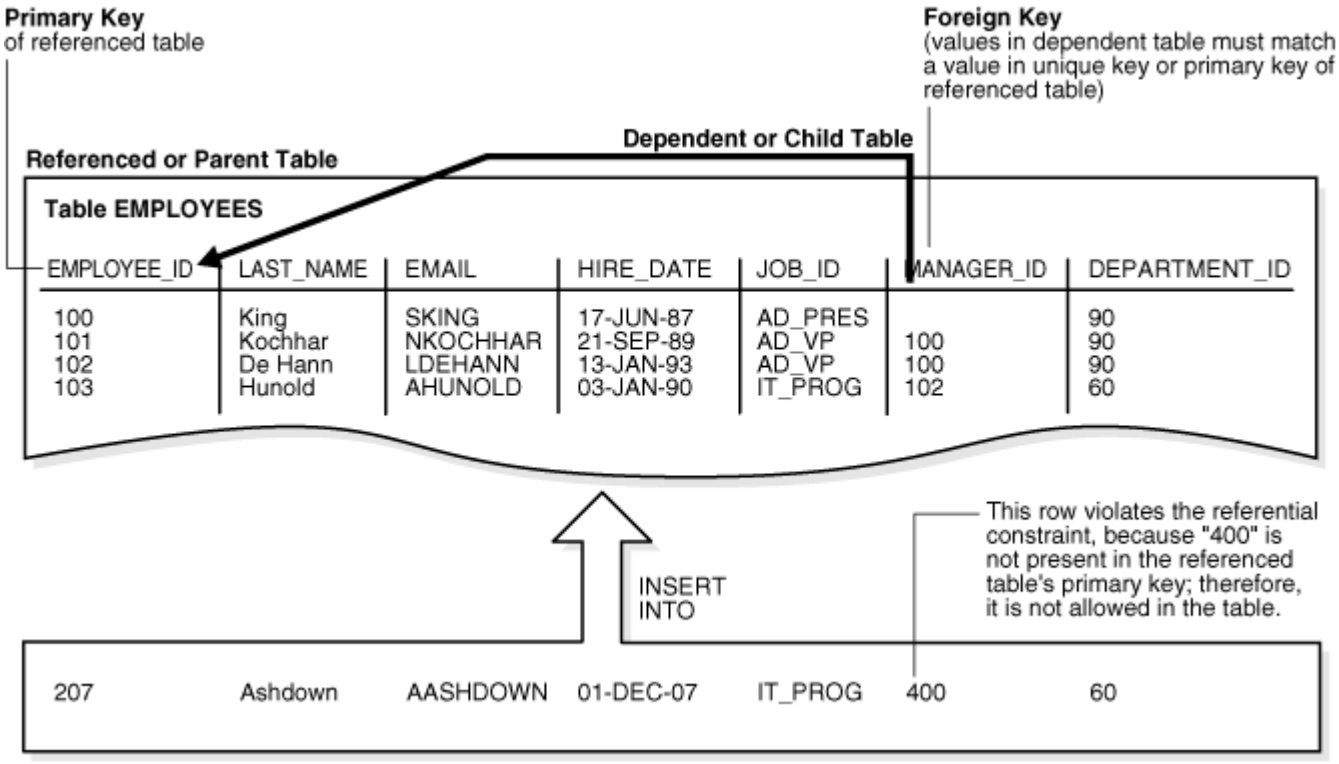
表に外部キー制約を追加する方法を学習するには、Oracle Database 2日で開発者ガイドおよびOracle Database開発ガイドを参照してください

自己参照型整合性制約

自己参照型整合性制約は、同じ表の親キーを参照する外部キーです。

次の図では、自己参照型制約によって、employees.manager_id列のすべての値が、employees.employee_id列の既存の値に対応するようになります。たとえば、従業員102の管理職がemployees表に存在する必要があります。この制約によって、間違った従業員番号がmanager_id列に存在する可能性がなくなります。

図5-2 単一表の参照制約



NULLと外部キー

リレーショナル・モデルでは、外部キーの値は、参照先の主キーまたは一意キーの値と一致するか、NULLであることが可能です。たとえば、hr.employeesの行で部門IDが指定されないことがあります。

コンポジット外部キーのいずれかの列がNULLの場合、そのキーのNULL以外の部分は、親キーの対応部分と一致している必要はありません。たとえば、reservations表のtable_id列およびdate列にはコンポジット外部キーが含まれるが、table_idはNULLであることがあります。

親キーの修正と外部キー

親キーを削除するとき、外部キーと親キーの間の関係が影響を及ぼします。たとえば、あるユーザーが部門のレコードの削除を試行すると、この部門の従業員のレコードに何が起こるかについて説明します。

親キーが変更された場合、参照整合性制約では、子表の依存行に対して実行される次のアクションを指定できます。

- 削除または更新に対するアクションなし

  通常、結果が参照整合性に違反する場合、ユーザーは参照キー値を変更できません。たとえば、employees.department_idがdepartmentsに対する外部キーであり、従業員が特定の部門に属す場合、この部門の行の削除を試行すると、制約に対する違反となります。

- 連鎖的な削除

  参照キー値を含む行が削除された場合、子表のうち依存している外部キー値を含むすべての行も削除されます(DELETE CASCADE)。たとえば、departmentsの行を削除すると、この部門のすべての従業員の行が削除されます。

- NULLを設定する削除

  参照キー値を含む行が削除された場合に、子表のうち依存している外部キー値を含むすべての行の値がNULLに設定されます(DELETE SET NULL)。たとえば、部門の行を削除すると、この部門のすべての従業員に関してdepartment_id列値がNULLに設定されます。

表5-3に、親表のキー値および子表の外部キー値に対する異なる参照アクションごとに可能なDML文の概要を示します。

表5-3 UPDATE NO ACTIONとDELETE NO ACTIONで許可されるDML文

DML文	親表に対して発行	子表に対して発行
INSERT	親キー値が一意であれば常に発行できます。	外部キーの値が親キーに存在するか、外部キーの一部またはすべてが NULL の場合にのみ発行できます。
UPDATE NO ACTION	文の実行後に、参照される親キー値のない行が子表内に残らない場合は発行できます。	文の実行後も新しい外部キー値によって参照キー値が参照される場合は発行できます。
DELETE NO ACTION	子表のどの行も親キー値を参照していない場合は発行できます。	常に発行可能
DELETE CASCADE	常に発行可能	常に発行可能
DELETE SET NULL	常に発行可能	常に発行可能

ノート:

Oracle Database の外部キー整合性制約でサポートされていない他の参照アクションは、データベース・トリガーを使用して規定できます。「トリガーの概要」を参照してください。

関連項目:

ON DELETE句について学習するには、『*Oracle Database SQL言語リファレンス*』を参照してください。

索引と外部キー

通常、外部キーには索引付けが必要です。唯一の例外は、対応する一意キーまたは主キーの更新や削除が発生しないことが確実な場合です。

子表で外部キーの索引付けを行うと、次の利点があります。

- 子表に対する完全表ロックが防止されます。かわりに、データベースは索引に対する行ロックを取得します。
- 子表の全表スキャンを行う必要がなくなります。例として、ユーザーがdepartments表から部門10のレコードを削除する場合を想定します。employees.department_idが索引付けされていない場合、データベースはemployeesをスキャンして、部門10に従業員が存在するかどうかを確認する必要があります。

関連項目:

- 「ロックと外部キー」では、索引付き外部キー列と索引なしの外部キー列のロック動作について説明します
- 「索引の概要」では、索引の目的および特性について説明します

チェック制約

列または列の集合に対する**チェック制約**では、すべての行について、指定した**条件**がTRUEまたはUNKNOWNであることが必要です。

DMLの結果で制約の条件がFALSEに評価される場合、そのSQL文はロールバックされます。チェック制約の主な利点は、特殊な整合性規則を規定できることです。たとえば、チェック制約を使用すると、hr.employees表に次の規則を規定できます。

- salary列の値は10000以下である必要があります。
- commission列の値は、salaryの値以下である必要があります。

次の例では、employeesに最大給与制約を作成し、最大値を超える給与を含む行を文で挿入すると、どのようなことが起こるかを示します。

```
SQL> ALTER TABLE employees ADD CONSTRAINT max_emp_sal CHECK (salary < 10001);
SQL> INSERT INTO employees (employee_id,last_name,email,hire_date,job_id,salary)
  1  VALUES (999,'Green','BGREEN',SYSDATE,'ST_CLERK',20000);
.
.
.
ERROR at line 1:
ORA-02290: check constraint (HR.MAX_EMP_SAL) violated
```

1つの列に、定義された列を参照する複数のチェック制約を設定できます。たとえば、salary列には、10000より大きい値を禁止する1つの制約と、500未満の値を禁止する別の制約を設定できます。

単一の列に対して複数のチェック制約が存在する場合は、その目的が競合しないように設計する必要があります。また、条件は特定の順序では評価されません。データベースでは、チェック条件が矛盾しないかどうかは検証されません。

関連項目:

チェック制約の制限について学習するには、Oracle Database SQL言語リファレンスを参照してください。

整合性制約の状態

制約定義の一部として、Oracle Databaseがいつどのようにして制約を規定するかを指定できます。これにより、制約状態が決定します。

変更されたデータおよび既存のデータのチェック

データベースでは、制約が既存のデータまたは将来のデータに適用されるかどうかを指定できます。制約が使用可能である場合、データベースは、入力または更新の時点での新規データをチェックします。制約に準拠しないデータをデータベースに入力することはできません。

たとえば、employees.department_idに対してNOT NULL制約を使用可能にすると、将来のすべての行が部門IDを持ちます。制約が使用禁止である場合、表には制約に違反する行を入力できます。

制約は次の検証モードのいずれかに設定できます。

- VALIDATE

  既存のデータは制約に準拠する必要があります。たとえば、employees.department_idに対してNOT NULL制約を使用可能にし、この制約をVALIDATEに設定すると、既存のすべての行が部門IDを持つことをチェックできます。

- NOVALIDATE

  既存のデータは制約に準拠する必要がありません。実際には、これは"信頼"モードです。たとえば、表にロードされたすべての売上に日付がある場合、日付列にNOT NULL制約を作成し、制約をNOVALIDATEに設定できます。適用されない制約は、通常、マテリアライズド・ビューおよびクエリー・リライトにのみ有効です。

  NOVALIDATEモードの制約の場合、RELYパラメータは、オプティマイザが制約を使用して結合情報を判別できることを示します。制約はデータの検証には使用されませんが、マテリアライズド・ビューのより高度なクエリー・リライトを可能にし、データ・ウェアハウス・ツールによってデータ・ディクショナリから制約情報を取得できます。デフォルトはNORELYです。これは、オプティマイザが実際には制約を認識しないことを意味します。

VALIDATEおよびNOVALIDATEの動作は、常に、制約が使用可能であるか使用禁止であるかどうかによって異なります。次の表に関係をまとめます。

表5-4 変更されたデータおよび既存のデータのチェック

変更されたデータ	既存のデータ	サマリー
ENABLE	VALIDATE	既存のデータと将来のデータは制約に準拠する必要があります。データが入力されている表に新しい制約を適用しようとすると、既存の行がその制約に違反する場合、エラーになります。
ENABLE	NOVALIDATE	データベースは制約をチェックしますが、制約はすべての行に関して TRUE である必要はありません。このため、既存の行が制約に違反している可能性はありますが、新しい行または変更された行は規則に準拠する必要があります。このモードは、整合性がすでに検証されている既存のデータを含むデータ・ウェアハウスで使用されます。
DISABLE	VALIDATE	データベースでは制約が使用禁止になり、索引が削除されて、制約の対象となる列への変更が禁止されます。
DISABLE	NOVALIDATE	制約はチェックされず、TRUE でなくてもかまいません。

関連項目:

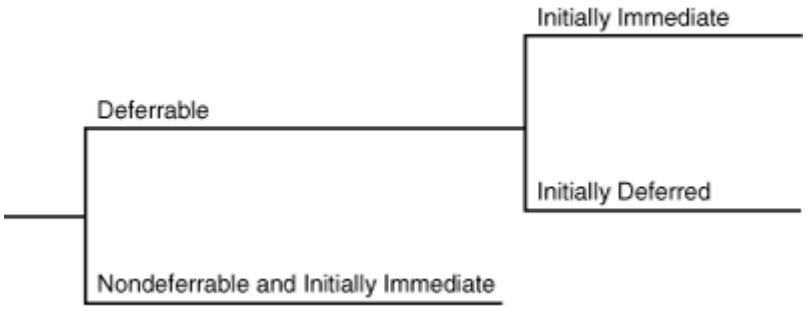
制約の状態について学習するには、『*Oracle Database SQL言語リファレンス*』を参照してください。

データベースが制約の妥当性をチェックするタイミング

あらゆる制約は、遅延不可状態(デフォルト)または遅延可能状態のいずれかにあります。この状態によって、Oracle Databaseが妥当性を確認するためにどの時点で制約をチェックするかが決まります。

次の図は、遅延可能制約のオプションを示します。

図5-3 遅延可能制約のオプション



遅延不可制約

遅延不可制約では、Oracle Databaseは、制約の妥当性チェックをトランザクションの最後まで遅延することはありません。かわりに、データベースでは各文の最後で制約をチェックします。制約に違反している場合、その文はロールバックします。

たとえば、employees.last_name列用に遅延不可のNOT NULL制約が存在します。セッションが姓のない行を挿入しようとすると、NOT NULL制約に違反しているため、データベースはただちに文をロールバックします。行は挿入されません。

遅延可能制約

遅延可能制約を使用すると、トランザクションでSET CONSTRAINT句を使用して、COMMIT文が発行されるまでこの制約のチェックを遅延させることができます。制約に違反している可能性があるデータベースを変更する場合、この設定により、すべての変更が完了するまで制約を事実上無効にすることができます。

データベースが遅延可能制約をチェックする時点に関して、デフォルト動作を設定できます。次のいずれかの属性を指定できます。

- INITIALLY IMMEDIATE

  データベースは、各文の実行直後に制約をチェックします。制約に違反している場合、データベースは文をロールバックします。

- INITIALLY DEFERRED

  データベースは、COMMITが発行された時点で制約をチェックします。制約に違反している場合、データベースはトランザクションをロールバックします。

employees.last_nameに対する遅延可能なNOT NULL制約がINITIALLY DEFERREDに設定されているとします。ユーザーは100個のINSERT文を含むトランザクションを作成し、その一部にはlast_nameとしてNULL値があります。ユーザーがコミットしようとすると、データベースは100個すべての文をロールバックします。ただし、この制約がINITIALLY IMMEDIATEに設定されている場合、データベースはトランザクションをロールバックしません。

制約によってアクションが発生する場合は、遅延制約か即時制約かに関係なく、このアクションはアクションを発生させた文の一部とみなされます。たとえば、departmentsの行を削除すると、削除された部門の行を参照するemployeesのすべての行が削除されます。この場合、employeesからの削除は、departmentsに対して実行されたDELETE文の一部とみなされます。

関連項目:

制約属性とそのデフォルト値の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

制約チェックの例

次の例では、Oracle Databaseが制約のチェックをどの時点で実行するかを示します。

この例では、次のような状況を想定します。

- employees表は、「自己参照型整合性制約」に示した構造を持っています。
- 自己参照型制約によって、manager_id列のエントリがemployee_id列の値に依存しています。

例: 親キー値が存在しない時点での外部キー列での値の挿入

この例では、employees表に最初の行を挿入する場合を考えます。現在は行が存在しないため、manager_id列の値がemployee_id列の既存の値を参照できない場合に行を入力する方法を考えなくてはなりません。

次の方法が考えられます。

- manager_id列にNOT NULL制約が定義されていない場合には、第1行のmanager_id列にnullを入力できます。

  外部キーにはnullが許されるため、Oracle Databaseではこの行を表に挿入します。

- 同じ値をemployee_id列およびmanager_id列に入力でき、これは、その従業員が管理職であることを示します。

  このことから、Oracle Databaseが文の実行後に制約チェックを実行することがわかります。親キーと外部キーに同じ値を指定した行の入力を許可するために、データベースは、まず新しい行を挿入してから、その新しい行のmanager_idに対応するemployee_idのある行がその表に含まれているかどうかをチェックします。

- SELECT文のネストを伴うINSERT文など、複数行を挿入するINSERT文により、相互に参照しあう行を挿入できます。

  たとえば、第1行には従業員IDとして200、管理職IDとして300が含まれており、第2行には従業員IDとして300、管理職IDとして200が含まれているとします。制約チェックはINSERT文の実行が完了するまで遅延されています。すべての行が挿入されてから、制約違反がないかどうかすべての行がチェックされます。

デフォルト値は、文の解析前にINSERT文の一部として組み込まれます。このため、デフォルトの列値はすべての整合性制約チェックの対象になります。

例: すべての外部キー値および親キー値の更新

この例では、自己参照型制約によって、employees のmanager_id列のエントリがemployee_id列の値に依存しています。

会社が買収された場合です。この買収に伴い、すべての従業員番号の現在の設定値に5000を加算して、新しい会社の従業員番号と調和させる必要があります。次の図に示すように、一部の従業員は管理職でもあります。

図5-4 更新前のemployees表

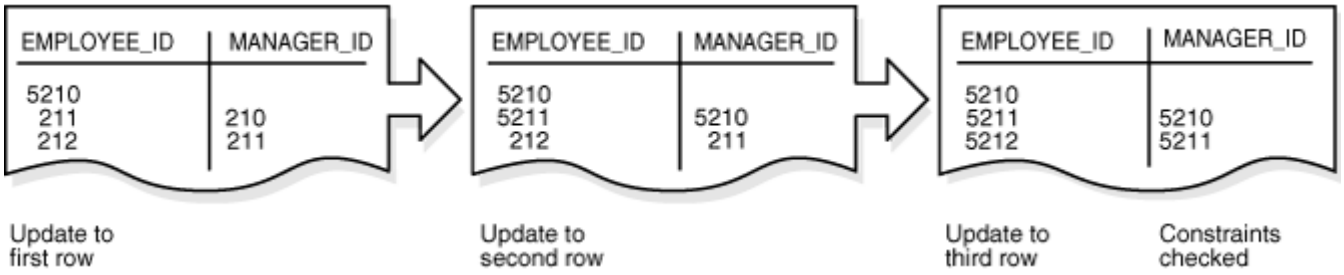
EMPLOYEE_ID	MANAGER_ID
210	
211	210
212	211

管理職番号は従業員番号でもあるため、管理職番号にも5000を加算する必要があります。次のSQL文を実行すると値を更新できます。

```
UPDATE employees SET employee_id = employee_id + 5000,
  manager_id = manager_id + 5000;
```

制約は、各manager_id値がemployee_id値と一致するかどうかを検証するように定義されていますが、データベースでは文の完了後に制約チェックが効率的に実行されるため、前述の文は有効です。図5-5に、データベースでは、制約をチェックする前に、SQL文全体のアクションが実行されることを示します。

図5-5 制約チェック



この項の例は、INSERT文とUPDATE文を実行した場合の制約チェックのメカニズムを示していますが、データベースでは同じメカニズムがすべてのタイプのDML文に使用されます。同じメカニズムが、自己参照型制約のみでなく、すべてのタイプの制約に使用されます。

ノート:

ビューまたはシノニムに対する操作は、実表に定義されている整合性制約に従います。

6 データ・ディクショナリと動的パフォーマンス・ビュー

各Oracle Databaseの読取り専用の参照表とビューの中核的なセットは、**データ・ディクショナリ**と呼ばれます。**動的パフォーマンス・ビュー**は、データベースがオープンされ、使用されている間に継続的に更新される特別なビューです。

この章の構成は、次のとおりです。

- データ・ディクショナリの概要
- 動的パフォーマンス・ビューの概要
- データベース・オブジェクト・メタデータ

データ・ディクショナリの概要

データ・ディクショナリはOracle Databaseの重要な部分であり、データベースに関する管理メタデータを提供する読取り専用の表の集合です。

データ・ディクショナリには次のような情報が含まれています。

- 列のデフォルト値、整合性制約情報など、データベース内の各スキーマ・オブジェクトの定義
- スキーマ・オブジェクトに割り当てられている領域、およびスキーマ・オブジェクトによって現在使用されている領域の容量
- Oracle Databaseユーザーの名前、ユーザーに付与された権限とロール、およびユーザーに関連する監査情報

データ・ディクショナリは、すべてのOracle Databaseでデータ管理の中心的な部分となります。たとえば、データベースでは次のアクションを実行します。

- ユーザー、スキーマ・オブジェクトおよび記憶域構造に関する情報を検索するために、データ・ディクショナリにアクセスします。
- DDL文が発行されるたびに、データ・ディクショナリを変更します

Oracle Databaseでは、データ・ディクショナリのデータは、他のデータと同様に表に格納されるため、ユーザーはSQLを使用してデータの問合せを実行できます。たとえば、ユーザーは自分の権限、自分のスキーマに存在する表、自分の表に存在する列、これらの列に対して索引が構築されているかどうかなどを判断するために、SELECT文を実行できます。

関連項目:

- 「スキーマ・オブジェクトの概要」
- 「ユーザー・アカウント」
- 「データ定義言語(DDL)文」

データ・ディクショナリの内容

データ・ディクショナリは、実表およびビューで構成されています。

これらのオブジェクトは次のように定義されます。

- 実表

  データベースに関する情報を格納します。これらの表に読取り/書込みできるのはOracle Databaseのみです。これら

の表は正規化され、データのほとんどは、暗号形式で格納されているため、ユーザーが直接アクセスすることはほとんどありません。

- ビュー

  実表にあるデータを、ユーザー名や表の名前などの実用的な情報にデコードし、結合とWHERE句を使用して情報を簡略化します。ビューには、データ・ディクショナリ内のすべてのオブジェクトの名前と説明が含まれています。すべてのユーザーがアクセスできるビューもいくつかありますが、その他のビューは管理者のみが使用するように設計されています。

一般的に、データ・ディクショナリはセットとしてグループ化されています。多くの場合、次の表で示すように、そのセットは、類似した情報が格納されている3つのビューで構成され、それぞれが接頭辞によって区別されます。適切なビューに問合せを発行すると、自分に関係のある情報にのみアクセスできます。

表6-1 データ・ディクショナリ・ビューのセット

接頭辞	ユーザー・アクセス	内容	ノート
DBA_	データベース管理者	すべてのオブジェクト	一部の DBA_ビューには、管理者にとって有用な情報を含む列が追加されています。
ALL_	すべてのユーザー	ユーザーが権限を持つオブジェクト	ユーザーが所有するオブジェクトが含まれています。これらのビューは有効化されている現在の一連のロールに従います。
USER_	すべてのユーザー	ユーザーによって所有されているオブジェクト	接頭辞が USER_のビューには、通常、列 OWNER は含まれません。この列は、問合せを発行するユーザーとして、USER_ビューに暗黙的に含まれています。

ビュー・セットには必ずしも3つのメンバーが含まれるわけではありません。たとえば、データ・ディクショナリにはDBA_LOCKビューはありますが、ALL_LOCKビューはありません。

システムによって提供されているDICTIONARYビューには、すべてのデータ・ディクショナリ・ビューの名前と短い説明が含まれています。次に、このビューに対する問合せと出力サンプルの一部を示します。

```
SQL> SELECT * FROM DICTIONARY
  2  ORDER BY TABLE_NAME;

TABLE_NAME                     COMMENTS
------------------------------ ----------------------------------------
ALL_ALL_TABLES                 Description of all object and relational
                               tables accessible to the user

ALL_APPLY                      Details about each apply process that
                               dequeues from the queue visible to the
                               current user
.
```

```
.
.
```

関連項目:

- 「ビューの概要」
- データ・ディクショナリ・ビューとその列の完全なリストは、『Oracle Databaseリファレンス』を参照してください

接頭辞がDBA_のビュー

接頭辞がDBA_のビューには、データベース全体の関連情報がすべて表示されます。DBA_ビューは、管理者のみが使用するように設計されています。

次の問合せの例では、データベース内のすべてのオブジェクトに関する情報が表示されます。

```
SELECT OWNER, OBJECT_NAME, OBJECT_TYPE
FROM   DBA_OBJECTS
ORDER BY OWNER, OBJECT_NAME;
```

関連項目:

管理権限の詳細は、『Oracle Database管理者ガイド』を参照

接頭辞がALL_のビュー

接頭辞がALL_のビューは、ユーザーのデータベース全体の概要を参照します。

これらのビューは、ユーザーが所有しているスキーマ・オブジェクトに加えて、権限とロールのPUBLICへの付与や明示的な付与によってそのユーザーがアクセスできるようになったスキーマ・オブジェクトに関する情報を戻します。

たとえば次の問合せは、アクセス権を持っているすべてのオブジェクトに関する情報を戻します。

```
SELECT OWNER, OBJECT_NAME, OBJECT_TYPE
FROM   ALL_OBJECTS
ORDER BY OWNER, OBJECT_NAME;
```

ALL_ビューは有効化されているロールの現在のセットに従うため、問合せの結果は、次に示す例のように、どのロールが有効化されているかによって異なります。

```
SQL> SET ROLE ALL;

Role set.

SQL> SELECT COUNT(*) FROM ALL_OBJECTS;

COUNT(*)
----------
68295

SQL> SET ROLE NONE;

Role set.
```

```
SQL> SELECT COUNT(*) FROM ALL_OBJECTS;

COUNT(*)
----------
53771
```

ストアド・プロシージャはロールがデフォルトで有効化されていないため、ここでALL_ビューを使用する場合、アプリケーション開発者はロールが及ぼす影響について認識しておく必要があります。

接頭辞がUSER_のビュー

通常のデータベース・ユーザーが最も頻繁に使用するのは、接頭辞がUSER_のビューです。

これらのビューには、次のような特長があります。

- ユーザーが作成したスキーマ・オブジェクトやユーザーによる権限付与に関するメタデータなど、ユーザーのプライベートなデータベース環境を参照します。
- ユーザーに関係する行のみを表示し、ALL_ビューにある情報のサブセットを戻します。
- 列OWNERが暗黙的に想定されることを除いて、他のビューと同一の列を持っています。
- 使用しやすいように短縮したPUBLICシノニムを持つことができます。

たとえば、次の問合せは、自分のスキーマに入っているすべてのオブジェクトを戻します。

```
SELECT OBJECT_NAME, OBJECT_TYPE
FROM   USER_OBJECTS
ORDER BY OBJECT_NAME;
```

DUAL表

DUAL表は、既知の結果を保証するために、Oracle Databaseとユーザー作成のプログラムによって参照されるデータ・ディクショナリ内の小さな表です。

DUAL表は、現在の日時など、値を1回のみ戻す必要がある場合に役立ちます。DUALにはすべてのデータベース・ユーザーがアクセスできます。

DUAL表にはDUMMYという1つの列と、値Xを格納する1つの行があります。次の例では、算術演算を行うためにDUAL表に対する問合せを発行しています。

```
SQL> SELECT ((3*4)+5)/3 FROM DUAL;

((3*4)+5)/3
-----------
 5.66666667
```

関連項目:

DUAL表の詳細は、『Oracle Database SQL言語リファレンス』を参照

データ・ディクショナリの記憶域

データ・ディクショナリの実表は、あらゆるOracle Databaseで最初に作成されるオブジェクトです。

データベースのデータ・ディクショナリ表およびビューは、すべてSYSTEM表領域に格納されます。SYSTEM表領域は、データベースのオープン中、常にオンラインであるため、データベースのオープン中、データ・ディクショナリは常に使用可能です。

関連項目:

SYSTEM表領域の詳細は、「SYSTEM表領域」を参照してください

Oracle Databaseでのデータ・ディクショナリの使用方法

データ・ディクショナリのすべての実表とユーザー・アクセス可能ビューは、Oracle Databaseユーザー・アカウントSYSが所有しています。

データベースの操作時に、Oracle Databaseはデータ・ディクショナリを読み取って、スキーマ・オブジェクトが存在すること、およびユーザーのアクセス権が適切に付与されていることを確認します。Oracle Databaseは、データベース構造、監査、権限付与およびデータの変更を反映するために、継続的にデータ・ディクショナリを更新します。

たとえば、ユーザーhrがinternsという表を作成すると、新しい表、列、セグメント、エクステントおよびhrがその表に対して持っている権限を反映するために、新しい行がデータ・ディクショナリに追加されます。この新しい情報は、次回ディクショナリ・ビューを問い合せるときに表示されます。

データ・ディクショナリの実表内のデータは、*Oracle Database*を機能させるために必要です。データ・ディクショナリの情報はOracle Databaseによってのみ書込みまたは変更される必要があります。したがって、データ整合性が損われることがあるため、Oracle Databaseのユーザーは SYSスキーマに含まれている行またはスキーマ・オブジェクトを変更できません。セキュリティ管理者は、この重要なアカウントを厳しく管理し続ける必要があります。

警告:

データ・ディクショナリ表のデータを変更したり操作すると、データベースの操作に永続的な悪影響を与えるおそれがあります。

関連項目:

「SYSスキーマとSYSTEMスキーマ」

データ・ディクショナリ・ビューのパブリック・シノニム

多くのデータ・ディクショナリ・ビューにユーザーが簡単にアクセスできるようにするため、Oracle Databaseはパブリック・**シノニム**を作成します。

セキュリティ管理者は、システム全体で使用するスキーマ・オブジェクトのパブリック・シノニムを作成して追加することもできます。プライベート・スキーマ・オブジェクトとパブリック・シノニムに同じ名前を使用しないことをお薦めします。

関連項目:

「シノニムの概要」

データ・ディクショナリ・キャッシュ

データ・ディクショナリの情報の多くは、**データ・ディクショナリ・キャッシュ**に含まれており、これは、ユーザーのアクセスの妥当性のチェックや、スキーマ・オブジェクトの状態検証のため、データベースで常にこれらの情報が必要とされるためです

通常、キャッシュには解析情報が含まれます。表とそれらの列について記述しているCOMMENTS列は、ディクショナリ・キャッシュには保持されませんが、**データベース・バッファ・キャッシュ**に保持されることがあります。

関連項目:

データ・ディクショナリ・キャッシュ

他のプログラムとデータ・ディクショナリ

他のOracle Database製品は、既存のビューを参照したり、独自のデータ・ディクショナリ表またはビューを追加できます。

データ・ディクショナリを参照するプログラムを記述するアプリケーション開発者は、基礎となる表ではなくパブリック・シノニムを使用することをお薦めします。これは、シノニムの方がリリース間での変更が少ないためです。

動的パフォーマンス・ビューの概要

Oracle Databaseは、操作中ずっと、カレント・データベース・アクティビティを記録する一連の仮想表を保持しています。

これらのビューは、データベースがオープンで使用中の間は継続的に更新されるため、動的です。このビューは、名前がV$で始まるため、*V$*ビューと呼ばれることもあります。

動的パフォーマンス・ビューには次のような情報が含まれています。

- システムおよびセッション・パラメータ
- メモリー使用量および割当て
- ファイルの状態(RMANバックアップ・ファイルなど)
- ジョブおよびタスクの進行状況
- SQLの実行
- 統計およびメトリック

動的パフォーマンス・ビューの主な用途は、次のとおりです。

- Oracle Enterprise Managerがデータベースに関する情報を取得するためにこのビューを使用します。
- 管理者はこのビューを使用して、パフォーマンスを監視し、デバッグを実行できます。

関連項目:

- Oracle Enterprise Manager
- 動的パフォーマンス・ビューの完全なリストは、Oracle Databaseリファレンスを参照してください。

動的パフォーマンス・ビューの内容

動的パフォーマンス・ビューは、データベース管理者が変更または削除できないため、固定ビューと呼ばれます。しかし、データベース管理者は、これらの表のビューに問合せを発行したり、ビューを作成したり、それらのビューにアクセスする権限を他のユーザーに付与できます。

動的パフォーマンス表はSYSが所有し、その名前はV_$で始まります。これらの表に対してビューが作成され、V$という接頭辞を持つパブリック・シノニムが作成されます。たとえば、V$DATAFILEビューにはデータファイルに関する情報が格納されます。V$FIXED_TABLEビューにはすべての動的パフォーマンス表およびビューに関する情報が格納されます。

ほとんどのV$ビューに対して、対応するGV$ビューが存在します。Oracle Real Application Clusters (Oracle RAC)では、GV$ビューに対して問合せを実行すると、該当するすべてのインスタンスからV$ビュー情報が取り出されます。

Database Configuration Assistant(DBCA)を使用してデータベースを作成する場合は、データ・ディクショナリがOracleによって自動的に作成されます。Oracle Databaseでは、自動的にcatalog.sqlスクリプトが実行され、このスクリプトには、その動的パフォーマンス・ビューの、ビューおよびパブリック・シノニムの定義が含まれます。これらのビューとシノニムを作成するには、catalog.sqlを実行する必要があります。

関連項目:

- データベース・サーバー・グリッド
- DBCAについて学習するには、「データベースのインストールおよび構成ツール」を参照してください
- catalog.sqlを手動で実行する方法を学習するには、Oracle Database管理者ガイドを参照してください。
- Oracle RACでパフォーマンス・ビューを使用する方法について学習するには、Oracle Real Application Clusters管理およびデプロイメント・ガイドを参照してください。

動的パフォーマンス・ビューの記憶域

動的パフォーマンス・ビューは、データベースのメモリー構造から構築された仮想表に基づいています。

これらのビューはデータベース内に格納される従来型の表ではありません。これらのビューではデータが動的に更新されるため、読取り一貫性は保証されません。

動的パフォーマンス・ビューは実際の表ではないため、データはデータベースおよびデータベース・インスタンスの状態によって異なります。たとえば、データベースが起動されており、マウントされていないとき、V$INSTANCEおよびV$BGPROCESSに対する問合せを発行できます。ただし、データベースがマウントされるまでは、V$DATAFILEに対する問合せは発行できません。

関連項目:

データの同時実行性と整合性

データベース・オブジェクト・メタデータ

DBMS_METADATAパッケージには、データベース・オブジェクトの完全な定義を抽出するためのインタフェースが用意されています。

これらの定義は、XMLまたはSQL DDLのいずれかで表現されます。Oracle Databaseでは、柔軟で洗練された、プログラム

制御用のインタフェースと、単純化された、非定型問合せ用のインタフェースの2つのスタイルのインタフェースが提供されます。

関連項目:

DBMS_METADATAの詳細は、『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』を参照してください

第II部 Oracleデータ・アクセス

Structured Query Language (SQL)は、すべてのプログラムおよびユーザーがOracleデータベースのデータへアクセスする際に使用する、高度な宣言型コンピュータ言語です。サーバー側の手続き型言語である**PL/SQL**および**Java**を使用すると、データベース自体にデータ・ロジックを格納できます。

この部は、次の章で構成されています。

- SQL
- サーバー側プログラミング: PL/SQLとJava

7 SQL

この章では、**Structured Query Language (SQL)**の概要およびOracle DatabaseによるSQL文の処理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。

- SQLの概要
- SQL文の概要
- オプティマイザの概要
- SQL処理の概要

SQLの概要

SQL(「シーキューエル」と読みます)は、すべてのプログラムおよびユーザーがOracleデータベースのデータへアクセスする際に使用する、セットベースの高度な宣言型コンピュータ言語です。

いくつかのOracleツールおよびアプリケーションでは、SQLの使用はマスクされていますが、すべてのデータベース・タスクはSQLを使用して実行されます。その他のデータ・アクセス方法を使用すると、Oracle Databaseに組み込まれているセキュリティが活用されず、データのセキュリティと整合性が損われる可能性があります。

SQLはOracle Databaseなどのリレーショナル・データベースに対するインタフェースを提供します。SQLでは、次のようなタスクを1つの一貫した言語に統合します。

- オブジェクトの作成、置換、変更および削除
- 表の行の挿入、更新および削除
- データの問合せ
- データベースおよびそのオブジェクトへのアクセスの制御
- データベースの整合性と一貫性の保証

SQLは対話的に使用できます(つまり、文を手動でプログラムに入力できます)。SQL文はCまたはJavaなどの異なる言語で記述されたプログラム内に埋め込むこともできます。

関連項目:

- 「サーバー側プログラミングの概要」および「クライアント側のデータベース・プログラミング」
- SQLの概要は、『Oracle Database SQL言語リファレンス』を参照してください

SQLデータ・アクセス

コンピュータ言語には大きく分けて2つの種類があり、1つは実行目的を表す非手続き型の**宣言型言語**、もう1つは処理の実行経緯を表すC++およびJavaなどの**手続き型言語**です。

SQLは、ユーザーが結果の導出方法ではなく目的とする結果を指定するという意味において、宣言型です。たとえば、次の文を使用すると、姓がKで始まる従業員のレコードを問い合せることができます。

データベースでは、データをナビゲートし、リクエストされた結果を取得するプロシージャを生成する作業を実行します。SQLの宣言型の性質により、論理レベルでデータを操作できます。実装の詳細は、データを操作する場合にのみ、考慮する必要があります。

```
SELECT   last_name, first_name
FROM     hr.employees
WHERE    last_name LIKE 'K%'
ORDER BY last_name, first_name;
```

データベースはWHERE条件(述語とも呼ばれる)を満たすすべての行を1つのステップで取得します。データベースは、ユーザー、別のSQL文またはアプリケーションに、これらの行をまとめて渡すことができます。アプリケーションで行を1つずつ処理する必要はなく、また、開発者が行の物理的な格納方法または取得方法を認識する必要もありません。

SQL文を実行すると、データベースのオプティマイザが働き、指定したデータに最も速くアクセスする方法が決定されます。Oracle Databaseでは、オプティマイザにジョブを適切に実行させるための技術もサポートされています。

関連項目:

SQL文およびSQLのその他の部分(演算子、関数、書式モデルなど)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規格

オラクル社は業界で承認されている標準への準拠に努め、SQLの規格協会に積極的に参加しています。

業界が認可している協会として、米国規格協会(ANSI)および国際標準化機構(ISO)があります。ANSIおよびISO/IECはいずれも、リレーショナル・データベースの標準言語としてSQLを認可しています。

SQL標準は10のパートで構成されます。1つのパート(SQL/RPR:2012)は2012年の新しいものです。5つのパートは2011年に改訂されました。他の4つのパートは2008年バージョンで、元のままです。

Oracle SQLには、ANSI/ISO標準SQL言語に対応する多くの拡張機能が組み込まれており、Oracle Databaseのツール製品とアプリケーションを使用することで、文を追加できます。SQL*Plus、SQL DeveloperおよびOracle Enterprise Managerなどのツールを使用すると、Oracle Databaseに対してANSI/ISO標準のSQL文を実行でき、さらにこれらのツール製品で使用可能な追加の文や機能を実行できます。

関連項目:

- 「データベース管理者向けのツール」および「データベース開発者向けのツール」
- Oracle SQLと標準SQLの相違点の詳細は、Oracle Database SQL言語リファレンスを参照してください
- SQL文との区別など、SQL*Plusコマンドの詳細は、『SQL*Plusユーザーズ・ガイドおよびリファレンス』を参照してください。

SQL文の概要

Oracle Database内の情報に対するすべての操作は、SQL**文**を使用して実行します。SQL文は識別子、パラメータ、変数、名前、データ型およびSQL**予約語**で構成されるコンピュータ・プログラムまたは命令です。

ノート:

SQL 予約語は、SQL で特別な意味を持ち、他の目的には使用できません。たとえば、SELECT と UPDATE は予約語のため、表名には使用できません。

SQL文は、完全なSQL文と等価であることが必要です。次に例を示します。

```
SELECT last_name, department_id FROM employees
```

Oracle Databaseでは、完全なSQL文のみが実行されます。次のような不完全文を実行しようとすると、テキストの不足を示すエラーが発生します。

```
SELECT last_name;
```

Oracle SQL文は、次のカテゴリに分類されます。

- データ定義言語(DDL)文
- データ操作言語(DML)文
- トランザクション制御文
- セッション制御文
- システム制御文
- 埋込みSQL文

データ定義言語(DDL)文

データ定義言語(**DDL**)文は、スキーマ・オブジェクトに対し、定義、構造の変更および削除を実行します。

DDLを使用すると、オブジェクトにアクセスするアプリケーションを変更することなく、オブジェクトの属性を変更できます。たとえば、アプリケーションをリライトせずに、人事管理アプリケーションからアクセスする表に列を追加できます。また、DDLを使用して、データベース・ユーザーがデータベース内で作業を実行している間に、オブジェクトの構造を変更することもできます。

具体的には、DDL文によって次のことが可能になります。

- スキーマ・オブジェクトと他のデータベース構造(データベースとデータベース・ユーザーを含む)を作成、変更および削除します。ほとんどのDDL文はCREATE、ALTERまたはDROPのキーワードで始まります。
- スキーマ・オブジェクトの構造を削除することなく、それらのオブジェクトの中のすべてのデータを削除します(TRUNCATE)。

  ノート:

  DELETE とは異なり、TRUNCATE は UNDO データを生成しないため、DELETE よりも処理が速くなります。また、TRUNCATE は削除のトリガーを起動しません。

- 権限とロールを付与および取り消します(GRANT、REVOKE)。
- 監査オプションをオンまたはオフに切り換えます(AUDIT、NOAUDIT)。

- データ・ディクショナリにコメントを追加します(COMMENT)。

例7-1 DDL文

次の例では、DDL文を使用してplants表を作成した後、DMLを使用して表に2つの行を挿入します。次に、DDLを使用して表構造を変更し、この表に対するユーザーの読取り権限を付与および取り消してから、表を削除します。

```
CREATE TABLE plants
    ( plant_id    NUMBER PRIMARY KEY,
      common_name VARCHAR2(15) );

INSERT INTO plants VALUES (1, 'African Violet'); # DML statement

INSERT INTO plants VALUES (2, 'Amaryllis'); # DML statement

ALTER TABLE plants ADD
    ( latin_name VARCHAR2(40) );

GRANT READ ON plants TO scott;

REVOKE READ ON plants FROM scott;

DROP TABLE plants;
```

データベースがDDL文を実行する直前に暗黙的なCOMMITが行われ、直後にCOMMITまたはROLLBACKが行われます。次の例では、2つのINSERT文の後にALTER TABLE文が続くため、データベースによって2つのINSERT文がコミットされます。ALTER TABLE文が成功した場合は、この文はデータベースによってコミットされ、失敗した場合は、この文はデータベースによってロールバックされます。いずれの場合も、2つのINSERT文はすでにコミットされています。

関連項目:

- 権限およびロールについて学習するには、「データベース・セキュリティの概要」を参照してください
- スキーマ・オブジェクトの作成方法を学習するには、『Oracle Database 2日で開発者ガイド』および『Oracle Database管理者ガイド』を参照してください
- ブロック化DDLと非ブロック化DDLの相違について学習するには、『Oracle Database開発ガイド』を参照してください
- DDL文のリストは、『Oracle Database SQL言語リファレンス』を参照してください

データ操作言語(DML)文

データ操作言語(**DML**)文は、既存のスキーマ・オブジェクト内のデータの問合せや操作を実行します。

DDL文ではデータベースの構造を変更するのに対して、DML文では内容の問合せまたは変更を行います。たとえば、ALTER TABLEは表の構造を変更するのに対して、INSERTは1つ以上の行を表に追加します。

DML文は、最も頻繁に使用するSQL文であり、次の操作を実行できます。

- 1つ以上の表またはビューからデータの取出しまたはフェッチを行います(SELECT)。
- 列値のリストを指定するか、既存のデータを選択および操作する副問合せを使用して、表またはビューに新しいデータ行を追加します(INSERT)。
- 表またはビューの既存の行の列値を変更します(UPDATE)。

- 行の更新または、条件付きで表またはビューに行の挿入を行います(MERGE)。
- 表またはビューから行を削除します(DELETE)。
- SQL文の実行計画を表示します(EXPLAIN PLAN)。
- 表またはビューをロックして、一時的に他のユーザーのアクセスを制限します(LOCK TABLE)。

次の例では、DMLを使用してemployees表に問い合せます。例では、DMLを使用してemployeesに行を挿入し、この行を更新した後、行を削除します。

```
SELECT * FROM employees;

INSERT INTO employees (employee_id, last_name, email, job_id, hire_date, salary)
  VALUES (1234, 'Mascis', 'JMASCIS', 'IT_PROG', '14-FEB-2008', 9000);

UPDATE employees SET salary=9100 WHERE employee_id=1234;

DELETE FROM employees WHERE employee_id=1234;
```

論理作業単位を形成するDML文のコレクションは、トランザクションと呼ばれます。たとえば、預金を振り替えるトランザクションは、普通預金口座の残高の減額、当座預金口座の残高の増額および預金口座履歴表への振替えの記録という3つの別々の操作を伴います。DDL文とは異なり、DML文は現行のトランザクションを暗黙的にコミットしません。

関連項目:

- 「DML処理とDDL処理の違い」
- 「トランザクションの概要」
- データの問合せおよび操作方法を学習するには、Oracle Database 2日で開発者ガイドを参照してください
- DML文のリストは、Oracle Database SQL言語リファレンスを参照してください

SELECT文

問合せは、表またはビューからデータを取得する操作です。

SELECTはデータの問合せに使用できる唯一のSQL文です。SELECT文の実行から取得した一連のデータは**結果セット**と呼ばれます。

次の表に、SELECT文で一般的に使用される2つの必須キーワードおよび2つのキーワードを示します。また、表では、キーワードとSELECT文の機能を関連付けています。

表7-1 SQL文のキーワード

キーワード	必須	説明	機能
SELECT	はい	結果に表示される列を指定します。投影により、表内の列のサブセットが作成されます。 式は 1 つ以上の値、演算子、および値を評価する SQL 機能の組合せです。SELECT キーワードの後かつ FROM 句の前に置かれる式のリストは SELECT 構文のリストと呼ばれます。	投影

キーワード	必須	説明	機能
FROM	はい	データの取得元の表またはビューを指定します。	結合
WHERE	いいえ	条件を指定して行をフィルタ処理し、表内の行のサブセットを作成します。条件は、1 つ以上の式および論理(ブール)演算子の組合せを指定し、TRUE、FALSE または UNKNOWN の値を戻します。	選択
ORDER BY	いいえ	行の表示順序を指定します。	

関連項目:

SELECTの構文およびセマンティクスについては、『*Oracle Database SQL言語リファレンス*』を参照してください。

結合

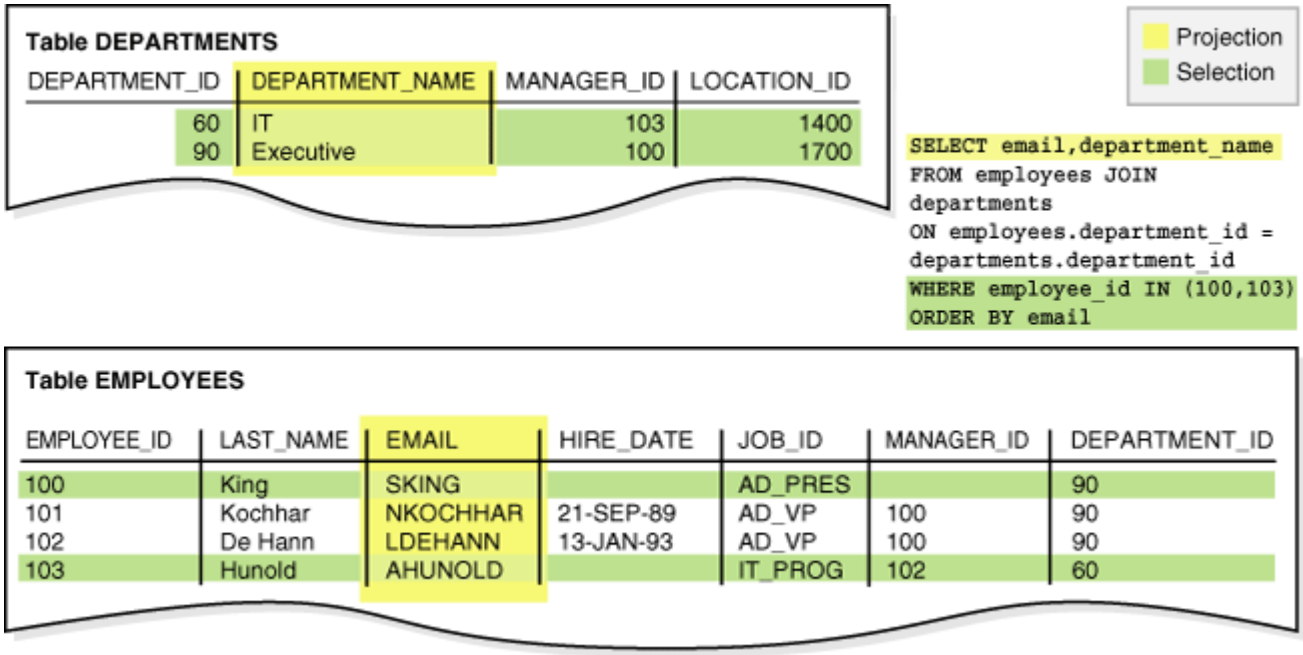
結合は2つ以上の表、ビューまたはマテリアライズド・ビューからの行を組み合せる問合せです。

次の例は、employees表およびdepartments表を結合して(FROM句)、指定した条件に一致する行のみを選択し(WHERE句)、投影を使用して2つの列からデータを取得します(SELECT)。SQL文の後にサンプルの出力を示しています。

```
SELECT email, department_name
FROM   employees
JOIN   departments
ON     employees.department_id = departments.department_id
WHERE  employee_id IN (100,103)
ORDER BY email;

EMAIL                     DEPARTMENT_NAME
------------------------- ------------------------------
AHUNOLD                   IT
SKING                     Executive
```

次の図に、前述の問合せの結合で示した投影および選択の操作を表します。

図7-1 投影と選択



ほとんどの結合には、FROM句またはWHERE句のいずれかに1つ以上の結合条件があり、それぞれ異なる表の2つの列をこの条件によって比較します。データベースでは1対の行を組み合せ、それぞれの対には、結合条件がTRUEになる各表の行を1行ずつ含みます。オプティマイザは結合条件、索引、および表に使用できる任意の統計に基づいて、データベースが表を結合する順序を決定します。

結合タイプは次のとおりです。

- 内部結合

  内部結合は、結合条件を満たす行のみを戻す2つ以上の表の結合です。たとえば、結合条件が employees.department_id=departments.department_idの場合、この条件を満たさない行は戻されません。

- 外部結合

  外部結合は結合条件を満たすすべての行を戻すとともに、一方の表からは、他方の表の条件を満たさない行に対応する行も戻します。

  結合条件が右側の表*B*のレコードに一致していない場合でも、表*A*と*B*の左側外部結合の結果には常に、左側の表*A*のすべてのレコードが含まれます。*B*に一致していない行が存在する場合、*B*の列では、*B*で一致していない行にはNULLが含まれます。たとえば、一部の従業員が部署に属していない場合、employees (左側の表)とdepartments (右側の表)の左側外部結合では、departmentsの行が結合条件を満たしていない(employees.department_idがNULLである)ときも、employeesのすべての行が取得されます。

  結合条件が左側の表*A*の行に一致していない場合でも、表*A*と*B*の右側外部結合の結果には右側の表*B*のすべてのレコードが含まれます。*A*に一致していない行が存在する場合、*A*の列では、*A*で一致していない行にはNULLが含まれます。たとえば、一部の部署に従業員がいない場合、employees (左側の表)とdepartments (右側の表)の右側外部結合では、employeesの行が結合条件を満たしていないときも、departmentsのすべての行が取得されます。

  完全外部結合は、左側外部結合と右側外部結合の組合せです。

- デカルト積

  結合問合せ内の2つの表に結合条件がない場合、データベースはデカルト結合を実行します。一方の表の各行を他方の表の各行と組み合せます。たとえば、employeesに107行ありdepartmentsには27行ある場合、デカルト積には107*27行が含まれます。デカルト積が便利な場合はほとんどありません。

関連項目:

- 結合について学習するには、Oracle Database SQLチューニング・ガイドを参照してください。
- 結合の詳細および例は、Oracle Database SQL言語リファレンスを参照してください

副問合せ

副問合せは別のSQL文内にネストされたSELECT文です。副問合せは、1つの問題を解決するために複数の問合せを実行する必要がある場合に便利です。

各文の問合せ部分は問合せブロックと呼ばれます。次の問合せでは、カッコ内の副問合せが内部問合せブロックです。

```
SELECT first_name, last_name
FROM   employees
WHERE  department_id
IN     ( SELECT department_id
         FROM departments
         WHERE location_id = 1800 );
```

内部SELECT文により、場所IDが1800の部門のIDが取得されます。これらの部門IDは、副問合せによってIDが戻された部門の従業員名を取得する外部問合せブロックに必要です。

このSQL文の構造では、内部問合せを最初に実行することをOracle Databaseに強制しているわけではありません。たとえば、Oracle Databaseにより問合せ全体がemployeesとdepartmentsの結合としてリライトされることもあり、その場合は副問合せ自体が実行されません。また、別の例として、Virtual Private Database (VPD)機能では、WHERE句を使用して従業員の問合せが絞り込まれることがあり、この場合は、Oracle Databaseでは、最初に従業員を問い合せてから部門IDが取得されます。オプティマイザにより、要求された行を取得するためのステップの最適な順序が決定されます。

関連項目:

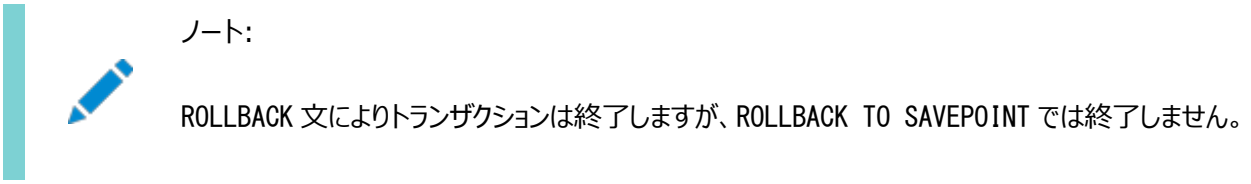
「Virtual Private Database (VPD)」

トランザクション制御文

トランザクション制御文は、DML文による変更の内容を管理し、一連のDML文をトランザクションとしてグループ化します。

これらの文によって、次のことを実行できます。

- トランザクションの変更を確定します(COMMIT)。
- トランザクションの開始以降(ROLLBACK)またはセーブポイント以降(ROLLBACK TO SAVEPOINT)に実行されたトランザクション内での変更を取り消します。セーブポイントは、トランザクションのコンテキスト内のユーザー宣言による中間マーカーです。



ノート:

ROLLBACK 文によりトランザクションは終了しますが、ROLLBACK TO SAVEPOINT では終了しません。

- ロールバックできるポイントを設定します(SAVEPOINT)。
- トランザクションのプロパティを設定します(SET TRANSACTION)。
- 後に続く各DML文の遅延可能整合性制約をチェックするかどうか、またはトランザクションをコミットするタイミングを指定します(SET CONSTRAINT)。

次の例はUpdate salariesという名前のトランザクションを開始します。例ではセーブポイントを作成し、従業員の給与を更新した後、セーブポイントまでトランザクションをロールバックします。さらに、給与を別の値に更新してコミットします。

```
SET TRANSACTION NAME 'Update salaries';

SAVEPOINT before_salary_update;

UPDATE employees SET salary=9100 WHERE employee_id=1234 # DML

ROLLBACK TO SAVEPOINT before_salary_update;

UPDATE employees SET salary=9200 WHERE employee_id=1234 # DML

COMMIT COMMENT 'Updated salaries';
```

関連項目:

- トランザクションの概要
- 「データベースが制約の妥当性をチェックするタイミング」
- トランザクション制御文について学習するには、『Oracle Database SQL言語リファレンス』を参照してください。

セッション制御文

セッション制御文は、ユーザー・**セッション**のプロパティを動的に管理します。

セッションは、データベースに対する現在のユーザー・ログインの状況を示す、データベース・インスタンス・メモリー内の論理エンティティです。セッションは、ユーザーがデータベースに認証された時点から、ユーザーが接続を切断するか、データベース・アプリケーションを終了する時点まで続きます。

セッション制御文によって、次のことを実行できます。

- デフォルトの日付形式の設定など、特化された機能を実行して現行のセッションを変更します(ALTER SESSION)。
- 現行のセッションのロール(権限のグループ)を使用可能または使用禁止にします(SET ROLE)。

次の文は、セッションのデフォルトの日付形式を動的に'YYYY MM DD-HH24:MI:SS'に変更します。

```
ALTER SESSION
   SET NLS_DATE_FORMAT = 'YYYY MM DD HH24:MI:SS';
```

セッション制御文は現行のトランザクションを暗黙的にコミットしません。

関連項目:

- 接続とセッション

- ALTER SESSIONの構文およびセマンティクスについては、Oracle Database SQL言語リファレンスを参照してください

システム制御文

システム制御文では、**データベース・インスタンス**のプロパティを変更します。

システム制御文は、ALTER SYSTEMのみです。この文は、設定値(共有サーバーの最小数など)の変更、セッションの終了およびその他のシステム・レベルでのタスクの実行に使用します。

システム制御文の例を次に示します。

```
ALTER SYSTEM SWITCH LOGFILE;

ALTER SYSTEM KILL SESSION '39, 23';
```

ALTER SYSTEM文は現行のトランザクションを暗黙的にコミットしません。

関連項目:

ALTER SYSTEMの構文およびセマンティクスは、『Oracle Database SQL言語リファレンス』を参照してください。

埋込みSQL文

埋込みSQL文は、手続き型言語プログラム内にDDL、DMLおよびトランザクション制御文を取り込みます。

埋込み文は、Oracleプリコンパイラで使用されます。埋込みSQLは、手続き型言語アプリケーションにSQLを取り込むための1つの方法です。他に、Open Database Connectivity(ODBC)またはJava Database Connectivity(JDBC)などの手続き型APIを使用する方法もあります。

埋込みSQL文によって、次のことを実行できます。

- カーソルの定義、割当ておよび解放を行います(DECLARE CURSOR、OPEN、CLOSE)。
- データベースを指定して、接続します(DECLARE DATABASE、CONNECT)。
- 変数名を割り当てます(DECLARE STATEMENT)。
- 記述子を初期化します(DESCRIBE)。
- エラー条件と警告の処理方法を指定します(WHENEVER)。
- SQL文を解析および実行します(PREPARE、EXECUTE、EXECUTE IMMEDIATE)。
- データベースからデータを取り出します(FETCH)。

関連項目:

「サーバー側プログラミングの概要」および「クライアント側のAPI」

オプティマイザの概要

Oracle DatabaseがSQL文を処理する仕組みを理解するには、データベースの**オプティマイザ**と呼ばれる部分(問合せオプティマイザまたはコストベース・オプティマイザとも呼ばれる)を理解する必要があります。すべてのSQL文でオプティマイザを使用して、指定されたデータへの最も効率的なアクセス手段を決定します。

オプティマイザの使用

オプティマイザは、使用可能な実行方法を表す実行計画を生成します。

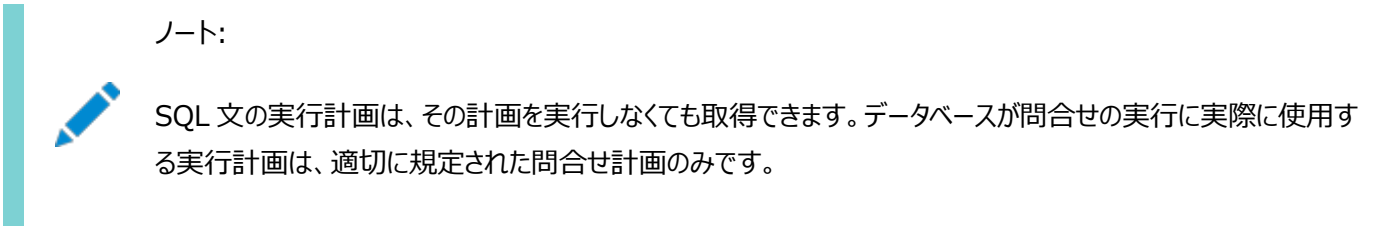
オプティマイザにより、いくつかの情報ソースが考慮されて、最も効率的な実行計画が決定されます。たとえば、オプティマイザでは、問合せ条件、使用可能なアクセス・パス、収集された対象システムの統計、ヒントなどが考慮されます。

DML文を実行するために、Oracle Databaseで多くの処理を必要とする場合があります。各処理では、データベースからデータ行を物理的に検索するか、文を発行したユーザーのためにデータ行を準備します。データベースが文を実行するときに使用するステップは、文の実行速度に大きく影響します。多くの場合、DML文の処理には多くの様々な方法を使用できます。たとえば、表または索引のアクセス順序を変更できます。

SQL文の最適な実行計画を決定する際に、オプティマイザは次の操作を実行します。

- 式および条件の評価
- データの詳細を把握し、このメタデータに基づいて最適化するための整合性制約の検査
- 文の変換
- オプティマイザの目的の選択
- アクセス・パスの選択
- 結合順序の選択

オプティマイザは、問合せの処理に使用できる方法の大部分を生成し、生成した実行計画の各ステップにコストを割り当てます。コストが最も低い計画が、実行する問合せ計画として選択されます。



ノート:

SQL 文の実行計画は、その計画を実行しなくても取得できます。データベースが問合せの実行に実際に使用する実行計画は、適切に規定された問合せ計画のみです。

オプティマイザが選択する内容は、オプティマイザの目的を設定して、オプティマイザ用の代表的な統計を収集することにより決まります。たとえば、オプティマイザの目的を次のいずれかに設定できます。

- 合計スループット

  結果の最後の行を可能なかぎり迅速にクライアント・アプリケーションに取得するよう、ALL_ROWSヒントがオプティマイザに指示します。

- 初期レスポンス時間

  最初の行を可能なかぎり迅速にクライアントに取得するよう、FIRST_ROWSヒントがオプティマイザに指示します。

典型的なエンド・ユーザーの対話型アプリケーションの場合、初期レスポンス時間による最適化が有用であるのに対して、バッチモードの非対話型アプリケーションの場合、合計スループットによる最適化が有用です。

関連項目:

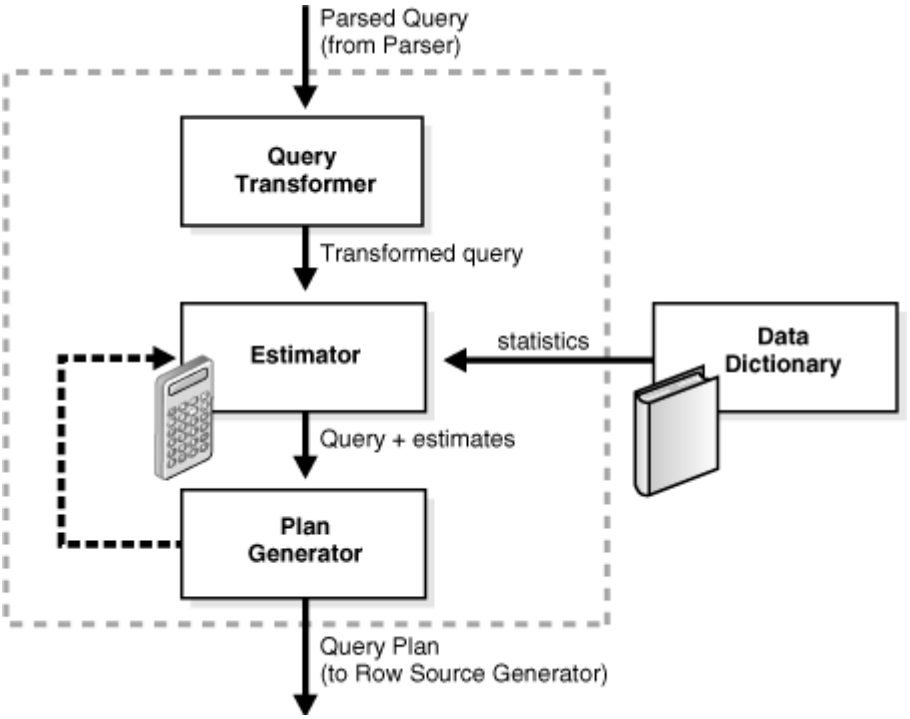
- DBMS_STATSの使用の詳細は、Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンスを参照してください
- オプティマイザの詳細およびヒントの使用方法の詳細は、Oracle Database SQLチューニング・ガイド を参照してください

オプティマイザ・コンポーネント

オプティマイザには、3つの主要コンポーネント(トランスフォーマ、エスティメータおよびプラン・ジェネレータ)が含まれています。

次の図は、そのコンポーネントを示しています。

図7-2 オプティマイザのコンポーネント



オプティマイザには、解析済の問合せが入力されます。オプティマイザは、次の操作を実行します。

1. オプティマイザは解析済の問合せを受け取り、使用可能なアクセス・パスおよびヒントに基づいてSQL文に対する一連の計画候補を生成します。
2. オプティマイザは、データ・ディクショナリ内の統計に基づき、各計画のコストを見積ります。コストとは、特定の計画を使用して文を実行するために必要な、予想されるリソース使用に比例した見積り値です。
3. オプティマイザは各計画のコストを比較し、コストが最も低い計画(問合せ計画と呼ばれる)を選択して、行ソース・ジェネレータに渡します。

関連項目:

- SQLの解析
- SQLの行ソース生成

問合せトランスフォーマ

問合せトランスフォーマは、オプティマイザがより優れた実行計画を生成できるように、問合せの形式を変更することが有用かどうかを判断します。問合せトランスフォーマへの入力は解析済の問合せで、オプティマイザでは問合せブロックのセットとして表されます。

関連項目:

「クエリー・リライト」

エスティメータ

エスティメータは指定された実行計画の全体コストを判断します。

この目的を達成するために、エスティメータは3つの異なる計測タイプを生成します。

- 選択性

  この計測タイプは、行セット内の行の一部分を表します。選択性は、問合せの述語(last_name='Smith'など)や述語の組合せに関連します。

- カーディナリティ

  この計測タイプは、行セット内の行数を表します。

- コスト

  このメジャーは、作業単位または使用されるリソースを表します。問合せオプティマイザはディスクI/O、CPU使用量、メモリー使用量を作業単位として使用します。

統計が使用できる場合、エスティメータは統計を使用して計測値を計算します。統計によって、メジャーの正確さの度合いは改良されます。

プラン・ジェネレータ

プラン・ジェネレータは、発行された問合せに対して異なる計画を試行します。オプティマイザは、コストが最も低い計画を選択します。

ネストされた副問合せおよびマージされていないビューごとに、オプティマイザはサブプランを生成します。オプティマイザは、各サブプランを個別の問合せブロックとして表します。プラン・ジェネレータは、別のアクセス・パス、結合方法および結合順序を試行することによって、問合せブロックに対する様々な計画を探索します。

適応問合せ最適化機能によって、文の実行中に収集された統計に基づいて計画が変更されます。すべての適応メカニズムにより、デフォルト・プランと異なる文の最終プランを実行できます。適応最適化では、文の実行中にサブプランの中から選択する動的プラン、または現在の実行後の実行でプランを変更する再最適化のいずれかを使用します。

関連項目:

- 「アプリケーションとSQLチューニング」
- オプティマイザのコンポーネントおよび適応最適化について学習するには、Oracle Database SQLチューニング・ガイド を参照してください

アクセス・パス

アクセス・パスは、問合せで行の取得に使用される手法です。

たとえば、索引を使用する問合せは、索引を使用しない問合せとは異なるアクセス・パスになります。通常、表内の行の小規模なサブセットを取得する文には、索引アクセス・パスが最適です。表内の大きな一部分にアクセスするには、全体スキャンがより効率的です。

データベースは、表からデータを取得するために、複数の異なるアクセス・パスを使用できます。代表的なものは次のとおりです。

- 全表スキャン

  このタイプのスキャンは、表からすべての行を読み取り、選択基準に合致しない行をフィルタ処理して除外します。データベースは、使用済領域と未使用領域を分離する最高水位標(HWM)より下のものも含め、セグメント内のすべてのデータ・ブロックを順次スキャンします(「セグメント領域と最高水位標」を参照)。

- ROWIDスキャン

  行のROWIDは、そのブロック内の行および行の位置を含むデータファイルおよびデータ・ブロックを指定します。データベースは、最初に文中のWHERE句または索引スキャンのいずれかから選択した行のROWIDを取得し、次にそのROWIDに基づいて選択した各行の位置を特定します。

- 索引スキャン

  このスキャンは、SQL文によってアクセスする索引付けされた列値に対応した索引を検索します(「索引スキャン」を参照)。文によって索引の列のみにアクセスする場合、Oracle Databaseは索引付けされた列値を索引から直接読み取ります。

- クラスタ・スキャン

  クラスタ・スキャンは、索引付きの表クラスタに格納された表からデータを取得するもので、表クラスタでは、同じクラスタ・キー値を持つすべての行が同じデータ・ブロックに格納されています(「索引付きクラスタの概要」を参照)。データベースは最初にクラスタ索引をスキャンして、選択した1つの行のROWIDを取得します。Oracle Databaseはこの ROWIDに基づいて行の位置を特定します。

- ハッシュ・スキャン

  ハッシュ・スキャンはハッシュ・クラスタ内の行の位置を特定するもので、ハッシュ・クラスタ内では、同じハッシュ値を持つすべての行が同じデータ・ブロックに格納されています(「ハッシュ・クラスタの概要」を参照)。データベースは最初に、文によって指定されたクラスタ・キー値にハッシュ関数を適用することにより、ハッシュ値を取得します。次に、Oracle Databaseはこのハッシュ値の行を含むデータ・ブロックをスキャンします。

オプティマイザは、文に対して使用可能なアクセス・パスと、各アクセス・パスまたはパスの組合せの使用時の見積りコストに基づいて、1つのアクセス・パスを選択します。

関連項目:

アクセス・パスについて学習するには、Oracle Database 2日でパフォーマンス・チューニング・ガイドおよびOracle Database SQLチューニング・ガイドを参照してください

オプティマイザ統計

オプティマイザ統計は、データベースの詳細を記述したデータとデータベース内のオブジェクトのコレクションです。統計は、アクセ

ス・パスを評価する場合にオプティマイザが使用できるデータ記憶域とデータ配分に関する、統計的に正しいピクチャを提示します。

オプティマイザ統計のタイプは次のとおりです。

- 表統計

  行数、ブロック数および行の平均長が含まれます。

- 列統計

  列内の個々の値とNULLの数およびデータ配分が含まれます。

- 索引統計

  リーフ・ブロックおよび索引レベルの数が含まれます。

- システム統計

  CPUとI/Oのパフォーマンスおよび使用率が含まれます。

Oracle Databaseは、すべてのデータベース・オブジェクトのオプティマイザ統計を自動的に収集し、これらの統計を自動メンテナンス・タスクとして保持します。また、DBMS_STATSパッケージを使用して、手動で統計を収集することもできます。このPL/SQLパッケージでは、統計を変更、表示、エクスポート、インポートおよび削除できます。

ノート:

オプティマイザ統計は問合せの最適化を目的として作成され、データ・ディクショナリに格納されます。これらの統計と動的パフォーマンス・ビューを介して表示できるパフォーマンス統計を混同しないでください。

オプティマイザ統計アドバイザは、現在の統計収集の状況、既存の統計収集ジョブの有効性、および収集された統計の品質を分析するビルトイン診断ソフトウェアです。オプティマイザ統計アドバイザは、現在の機能セットに基づくOracleベスト・プラクティスを体現する、ルールを維持します。このように、アドバイザは常に統計収集のための最新の推奨事項を提供します。

関連項目:

- 統計の収集および管理方法を学習するには、Oracle Database 2日でパフォーマンス・チューニング・ガイドおよびOracle Database SQLチューニング・ガイドを参照してください
- DBMS_STATSについて学習するには、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

オプティマイザ・ヒント

ヒントは、オプティマイザへの指示として機能するSQL文内のコメントです。

特定のアプリケーションのデータについて、オプティマイザよりもさらに詳細な知識を持つアプリケーション設計者の方が、SQL文をより効率的に実行する方法を選択できることもあります。アプリケーション設計者は、SQL文内にヒントを使用して文の実行方法を指定できます。次の例では、ヒントの使用方法を示します。

例7-2 FIRST_ROWSヒントを指定したSELECTの実行計画

対話型アプリケーションによって、50行を戻す問合せが実行されるとします。このアプリケーションでは、問合せによって戻される最

初の25行のみを初めにフェッチしてエンド・ユーザーに表示します。オプティマイザを使用して、最初の25レコードを可能なかぎり迅速に取得する計画を生成し、ユーザーが待たずに済むようにします。この指示をオプティマイザに渡すには、次の例のSELECT文およびAUTOTRACE出力に示すように、ヒントを使用します。

```
SELECT /*+ FIRST_ROWS(25) */ employee_id, department_id
FROM   hr.employees
WHERE  department_id > 50;

------------------------------------------------------------------------------
| Id | Operation                    | Name              | Rows | Bytes
------------------------------------------------------------------------------
|  0 | SELECT STATEMENT             |                   | 26   | 182
|  1 |  TABLE ACCESS BY INDEX ROWID | EMPLOYEES         | 26   | 182
|* 2 |   INDEX RANGE SCAN           | EMP_DEPARTMENT_IX |      |
------------------------------------------------------------------------------
```

この例では、実行計画は、オプティマイザがemployees.department_id列の索引を選択して部門IDが50を超えるemployeesの最初の25行を検索することを示します。オプティマイザは索引から取得したROWIDを使用してemployees表からレコードを取得し、クライアントに戻します。通常、最初のレコードの取得はほとんど即時です。

例7-3 ヒントを指定しないSELECTの実行計画

オプティマイザ・ヒントを指定しない同じ文を実行するとします。

```
SELECT employee_id, department_id
FROM   hr.employees
WHERE  department_id > 50;

------------------------------------------------------------------------------
| Id | Operation             | Name              | Rows | Bytes | Cos
------------------------------------------------------------------------------
|  0 | SELECT STATEMENT      |                   | 50   | 350   |
|* 1 |  VIEW                 | index$_join$_001  | 50   | 350   |
|* 2 |   HASH JOIN           |                   |      |       |
|* 3 |    INDEX RANGE SCAN   | EMP_DEPARTMENT_IX | 50   | 350   |
|  4 |    INDEX FAST FULL SCAN| EMP_EMP_ID_PK     | 50   | 350   |
```

この場合、実行計画では2つの索引を結合し、可能なかぎり迅速に要求されたレコードを戻します。オプティマイザは、例7-2のように索引から表へと繰り返し移動するのではなく、EMP_DEPARTMENT_IXのレンジ・スキャンを選択して、部門IDが50を超えるすべての行を検索し、ハッシュ表にこれらの行を配置します。オプティマイザは次に、EMP_EMP_ID_PK索引を読み取ることを選択します。この索引の各行について、ハッシュ表を調査して部門IDを検索します。

この場合、EMP_DEPARTMENT_IXの索引レンジ・スキャンが完了するまで、データベースはクライアントに最初の行を戻すことができません。このため、生成されたこの計画では最初のレコードを戻すために、より長い時間がかかります。索引のROWIDによって表にアクセスする例7-2の計画とは異なり、計画はマルチブロックI/Oを使用し、読取りの範囲が大きくなります。読取りによって、全結果セットの最後の行をより迅速に戻すことができます。

関連項目:

オプティマイザ・ヒントの使用方法を学習するには、Oracle Database SQLチューニング・ガイドを参照してください。

SQL処理の概要

この項では、Oracle DatabaseによるSQL文の処理の仕組みについて説明します。特に、データベースがDDL文を処理して

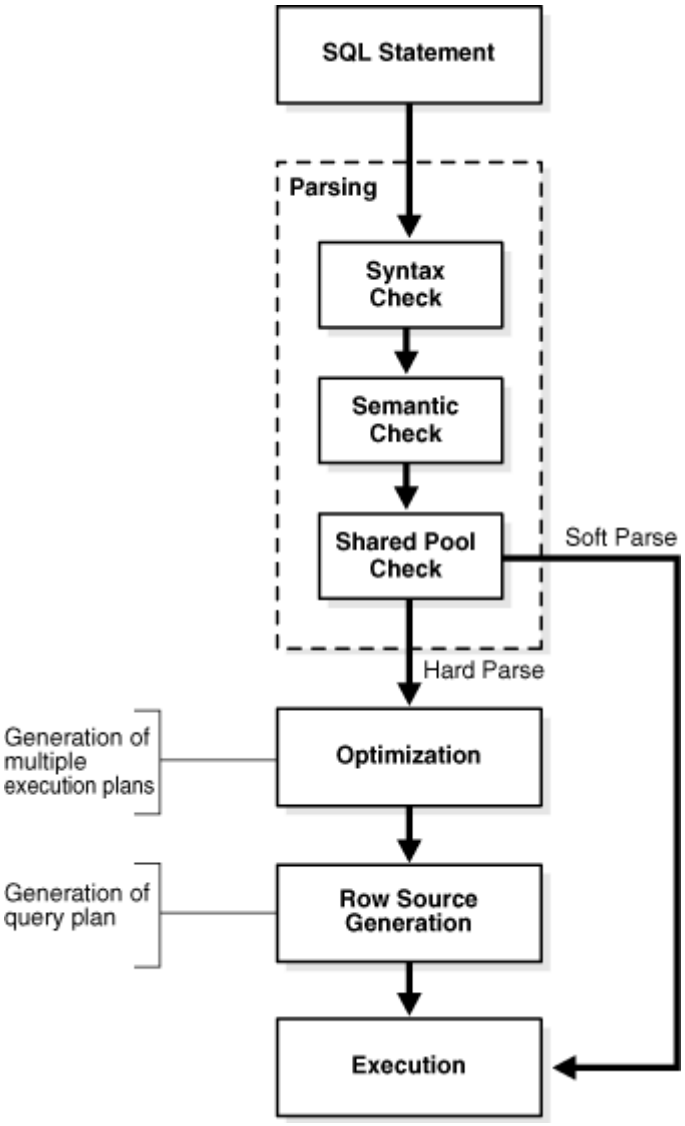
オブジェクトを作成する仕組み、DMLを使用してデータを変更する仕組み、問合せを使用してデータを取得する仕組みについて説明します。

SQL処理の段階

SQL処理の一般的な段階は、解析、最適化、行ソース生成、および実行です。文によっては、データベースがこれらのステップの一部を省略する場合があります。

次の図は、全体的な段階を示しています。

図7-3 SQL処理の段階



SQLの解析

SQL処理の最初の段階は、**SQLの解析**です。この段階では、SQL文の各部分を他のルーチンで処理できるデータ構造に分解する処理が行われます。

アプリケーションがSQL文を発行すると、アプリケーションはOracle Databaseに解析コールを出して、文の実行を準備します。解析コールによって、解析済のSQL文と処理に使用するその他の情報が保持されるセッション固有のプライベートSQL領域のハンドルであるカーソルがオープンまたは作成されます。カーソルおよびプライベートSQL領域はPGA内にあります。

解析コール時には、データベースでは次のチェックを実行します。

- 構文チェック
- セマンティクス・チェック

- 共有プール・チェック

これらのチェックによって、文の実行前に検出できるエラーを特定します。解析では捕捉できないエラーもあります。たとえば、データベースで、文の実行時にのみ、データ変換に関するデッドロックまたはエラーが発生する場合があります。

関連項目:

「ロックとデッドロック」

SQLの最適化

問合せの最適化は、SQL文を実行する最も効率的な方法を選択するプロセスです。

データベースは、アクセスする実際のデータに関して収集した統計に基づいて、問合せを最適化します。オプティマイザは行数、データ・セットのサイズおよびその他の要因を使用して、使用可能な実行計画を生成し、各計画に数値コストを割り当てます。データベースは最もコストが低い計画を使用します。

データベースは一意のDML文ごとに必ず1回以上のハード解析を実行して、この解析中に最適化を実行します。最適化を要求する副問合せなどのDMLコンポーネントを含まないかぎり、DDLは最適化されません。

関連項目:

- 「オプティマイザの概要」
- 問合せオプティマイザの詳細は、Oracle Database SQLチューニング・ガイドを参照してください。

SQLの行ソース生成

行ソース・ジェネレータは、オプティマイザから最適な実行計画を受け取り、データベースのその他の部分で使用可能な**問合せ計画**と呼ばれる反復計画を作成するソフトウェアです。

問合せ計画では、ステップの組合せ形式を使用します。各ステップでは行セットが戻されます。このセット内の行は次のステップまたは最後のステップのいずれかで使用され、SQL文を発行するアプリケーションに戻されます。

行ソースは、行を反復的に処理できる制御構造に沿って、実行計画のステップから戻された行セットです。表、ビュー、または結合処理あるいはグループ化処理の結果が行ソースになる可能性があります。

SQLの実行

実行時には、SQLエンジンが行ソース・ジェネレータによって作成されたツリー内の各行ソースを実行します。これのみがDML処理の必須ステップです。

実行中、データがメモリー内にない場合は、データベースがディスクからメモリーへとデータを読み取ります。また、データベースはデータ整合性を保証するために必要なすべてのロックおよびラッチを取り除き、SQL実行時に行われたすべての変更を記録します。SQL文の処理の最後の段階は、カーソルのクローズです。

データベースがインメモリー列ストア(IM列ストア)を使用するように構成されている場合、データベースは問合せを、可能な場合はIM列ストアに、さもなければディスクおよびデータベース・バッファ・キャッシュに透過的にルーティングします。1つの問合せでも、IM列ストア、ディスク、バッファ・キャッシュを使用できます。たとえば、ある問合せで2つの表を結合し、そのうち1つのみがIM列ストアにキャッシュされていることがあります。

関連項目:

- インメモリー領域
- 実行計画およびEXPLAIN PLAN文の詳細は、*Oracle Database SQLチューニング・ガイド*を参照してください。

DML処理とDDL処理の違い

Oracle DatabaseはDMLとは異なる方法でDDLを処理します。

たとえば、表を作成するとき、データベースはCREATE TABLE文を最適化しません。かわりに、Oracle DatabaseはDDL文を解析してコマンドを実行します。

DDLとは対照的に、ほとんどのDML文には問合せコンポーネントがあります。問合せでカーソルを実行すると、問合せで生成された行が結果セットに入力されます。

データベースでは、結果セットの行を一度に1行ずつまたはグループとしてまとめてフェッチできます。フェッチでは、データベースで行が選択され、問合せで要求された場合には行がソートされます。最後の行がフェッチされるまで、毎回のフェッチで結果の行が連続して取得されます。

関連項目:

DDLの処理、トランザクション制御およびその他のタイプの文について学習するには、『Oracle Database開発ガイド』を参照してください。

8 サーバー側プログラミング: PL/SQLとJava

「SQL」では、Structured Query Language(SQL)言語およびデータベースによるSQL文の処理方法について説明します。この章では、データベースに格納された手続き型言語/SQL(PL/SQL)またはJavaプログラムがSQLを使用できる仕組みについて説明します。

この章の内容は次のとおりです。

- サーバー側プログラミングの概要
- PL/SQLの概要
- Oracle DatabaseにおけるJavaの概要
- トリガーの概要

関連項目:

SQL言語の概要は、「SQL」を参照

サーバー側プログラミングの概要

SQLなどの非手続き型言語では、処理対象のデータ・セットは指定しますが、実行される処理またはその処理の実行方法は指定しません。

手続き型言語プログラムでは、ほとんどの文の実行が、前後の文、およびループまたは条件分岐などの制御構造に依存しますが、このような処理はSQLでは実行できません。手続き型言語と非手続き型言語の相違を例示するために、次のSQL文でemployees表に問い合せるとします。

```
SELECT employee_id, department_id, last_name, salary FROM employees;
```

この文はデータを要求しますが、データにロジックを適用しません。ただし、データ・セットの各従業員が給与および部門の業績に基づく昇給の対象者になるかどうかを判定するアプリケーションが必要であるとします。昇給の必要条件は、従業員が過去5年間に4回以上の昇給を受けていないことです。昇給する場合は、アプリケーションで給与を調整してマネージャに電子メールを送信し、昇給しない場合は、アプリケーションでレポートを更新する必要があります。

この処理を実行するために、条件付きロジックおよびプログラム・フロー制御を必要とする手続き型データベース・アプリケーションでSQLを使用できるようにします。基本的な開発アプローチは次のとおりです。

- クライアント側プログラミングを使用して、C、C++またはJavaなどの手続き型言語で記述されたアプリケーションにSQL文を埋め込みます。

  ソース・コードにSQL文を配置して、コンパイル前にそのSQL文をプリコンパイラまたはJavaトランスレータに送信できます。または、プリコンパイル・ステップを省略し、アプリケーションがデータベースと相互作用できるようにJava Database Connectivity(JDBC)またはOracle Call Interface(OCI)などのAPIを使用することもできます。

- サーバー側プログラミングを使用して、データベース内に配置するデータ・ロジックを開発します。

  アプリケーションは、PL/SQL(「ピーエルシーキューエル」と読みます)またはJavaで記述されたストアド・サブプログラム(プロシージャおよびファンクション)を明示的に起動できます。また、データベースに格納されたプログラム・ユニットの名前を指定し、特定のイベントに応答して起動するトリガーを作成することもできます。

この章では、2番目のアプローチについて説明します。サーバー側プログラミングの主な利点は、データベースに組み込まれた機能を任意の場所にデプロイできることです。指定されたオペレーティング・システム上でタスクを実行するための最善の方法を決定するのは、アプリケーションではなくデータベースです。また、サブプログラムを使用すると、サーバー上にアプリケーション処理を集中化することでスケーラビリティを向上させることができ、また、クライアントでコードを再利用することもできます。サブプログラム・コールは迅速かつ効率的であるため、1回のコールで計算集中型のストアド・サブプログラムを開始し、ネットワーク・トラフィックを軽減できます。

Oracle Databaseにデータ・ロジックを格納するには、次の言語を使用できます。

- PL/SQL

  PL/SQLは、SQLに対するOracle Databaseの手続きを拡張します。PL/SQLはデータベースに統合されており、すべてのOracle SQL文、関数およびデータ型をサポートします。データベースAPIに記述されたアプリケーションはPL/SQLストアド・サブプログラムを起動し、実行するPL/SQLコード・ブロックをデータベースに送信します。

- Java

  Oracle Databaseは、Javaアプリケーションの開発、格納およびデプロイもサポートしています。Javaストアド・サブプログラムはデータベースで実行され、中間層で実行されるプログラムからは独立しています。Javaストアド・サブプログラムは、PL/SQLに類似の実行モデルを使用してSQLとインタフェースを確立します。

関連項目:

- プリコンパイラおよびAPIでのSQLの埋込みについて学習するには、「クライアント側のデータベース・プログラミング」を参照してください
- Oracle Databaseアプリケーション開発の概要は、Oracle Database 2日で開発者ガイドを参照してください
- プログラミング環境の選択方法を学習するには、Oracle Database開発ガイドを参照してください

PL/SQLの概要

PL/SQLは、使用しやすく、SQLとシームレスで、堅牢かつ移植性を備えたセキュアなサーバー側に格納される手続き型言語を提供します。*PL/SQL*ユニットというプロシージャ型オブジェクトを使用してデータベースにアクセスし、データを処理できます。

PL/SQLユニットは、一般に、次のように分類されます。

- PL/SQLサブプログラムとは、データベースに格納され、アプリケーションから名前でコールできるPL/SQLブロックのことです。サブプログラムを作成すると、データベースではそのサブプログラムを解析し、解析後の表現をデータベースに格納します。サブプログラムは、プロシージャまたはファンクションとして宣言できます。
- PL/SQL無名ブロックとは、アプリケーション内に表示されるが名前が付いていない、またはデータベースに格納されていないPL/SQLブロックです。多くのアプリケーションでは、SQL文を記述できるところであればどこにでもPL/SQLブロックを記述できます。

PL/SQLのコンパイラとインタプリタはOracle SQL Developerに埋め込まれており、開発者にクライアントとサーバーの両方で一貫性のある高度な開発モデルを提供します。また、PL/SQLストアド・プロシージャは、Pro*C、JDBC、ODBCまたはOCIなどの複数のデータベース・クライアント、およびOracle ReportsとOracle Formsからコールできます。

関連項目:

- 「データベース開発者向けのツール」
- パッケージなどのPL/SQLの詳細は、Oracle Database PL/SQL言語リファレンスを参照してください

PL/SQLサブプログラム

PL/SQLサブプログラムは、コール元から、入力のみ、出力のみまたは入出力の値が入るパラメータを指定できる名前付きのPL/SQLブロックです。

サブプログラムは特定の問題を解決するか、関連するタスクを実行してモジュール式のメンテナンス可能なデータベース・アプリケーションのビルディング・ブロックとして機能します。サブプログラムは、PL/SQLプロシージャまたはPL/SQLファンクションのいずれかです。プロシージャとファンクションはほとんど同じものですが、プロシージャはコール元に値を戻さないのに対して、ファンクションは必ず1つの値を戻すという違いがあります。この章では、*PL/SQL*プロシージャという語でプロシージャとファンクションのいずれかを指すものとします。

関連項目:

- これらの言語のストアド・プロシージャについて学習するには、Pro*C/C++プログラマーズ・ガイドおよびPro*COBOLプログラマーズ・ガイドを参照してください
- Oracle Database PL/SQL言語リファレンス

PL/SQLサブプログラムの利点

サーバー側プログラミングにはクライアント側プログラミングよりも多くの利点があります。

次のような利点があります。

- パフォーマンスの改善
    - 情報が送信されるのは1回のみで、それ以降は使用するときに呼び出されるため、個々のSQL文を発行する場合や、PL/SQLブロック全体のテキストをOracle Databaseに送信する場合に比べて、アプリケーションがネットワークを介して送信する必要のある情報量が少なくなります。
    - データベース内でプロシージャのコンパイル済形式をそのまま使用できるため、実行時にコンパイルする必要がありません。
    - プロシージャがSGAの共有プール内にある場合、データベースはディスクからプロシージャを検索する必要がないため、ただちに実行を開始できます。
- メモリー割当て

  ストアド・プロシージャは、Oracle Databaseの共有メモリー機能を活用しているため、複数のユーザーが実行する場合も、メモリーにロードする必要があるのはプロシージャのコピー1つのみです。コードを複数のユーザーが共有すれば、アプリケーションに必要なデータベース・メモリー要件を実質的に削減できます。
- 生産性の向上

  ストアド・プロシージャにより、開発の生産性が向上します。共通のプロシージャを中心にしてアプリケーションを設計することにより、冗長なコーディングを回避できます。たとえば、プロシージャを記述して`employees`表内の行を操作できます。どのアプリケーションでも、これらのプロシージャをコールするためにSQL文を書きなおす必要はありません。データ管理の方法に変更があった場合も、修正する必要があるのはプロシージャのみで、プロシージャを使用するアプリケーションを修

正する必要はありません。

ストアド・プロシージャは、コードを再利用するための最善の方法です。任意の言語で記述されたデータベース接続を行うどのクライアント・アプリケーションもストアド・プロシージャを起動できるため、すべての環境において最大限にコードを再利用できます。

- 整合性

  ストアド・プロシージャにより、アプリケーションの整合性と一貫性が向上します。共通のプロシージャ群を中心としてアプリケーションを開発すれば、コーディング・エラーの発生回数が少なくなります。

  たとえば、サブプログラムが正確な結果を戻すかどうかをテストし、検証後は、そのサブプログラムを必要な数のアプリケーションで再利用でき、再びテストする必要はありません。プロシージャにより参照されるデータ構造が変更された場合には、プロシージャの再コンパイルのみが必要になります。プロシージャをコールするアプリケーションの修正は、必ずしも必要ありません。

- 定義者権限プロシージャでのセキュリティ

  ストアド・プロシージャを使用すると、データのセキュリティが向上します。定義者権限PL/SQLプロシージャは、現行のユーザーではなくその所有者の権限で実行されます。このため、定義者の権限で実行するプロシージャとファンクションを介してのみユーザーがデータにアクセスできるようにすると、ユーザーが実行するデータベース・タスクを制限できます。

  たとえば、表を更新するプロシージャへのアクセス権は付与しても、その表自体へのアクセス権は付与しないこともできます。ユーザーがプロシージャを呼び出すと、そのプロシージャがプロシージャの所有者の権限で実行されます。プロシージャの実行権限しかなく、表データの問合せ、更新または削除の権限を持たないユーザーは、プロシージャを呼び出すことはできても、表のデータをそれ以外の方法では操作できません。

- 実行者権限プロシージャで継承される権限およびスキーマのコンテキスト

  実行者権限PL/SQLプロシージャは、現行のユーザーの権限で現行のユーザーのスキーマ内で実行されます。言い換えると、実行者権限プロシージャは特定のユーザーまたはスキーマに結び付けられていません。アプリケーション開発者は、実行者権限プロシージャにより、基礎となるデータが複数のユーザー・スキーマに分割されていても、アプリケーション・ロジックを容易に一元化できます。

  たとえば、hr.employees表の更新プロシージャを実行するhr_managerユーザーは給与を更新できますが、同じプロシージャを実行するhr_clerkは住所データの更新のみに操作を制限されます。

関連項目:

- 「データベース・セキュリティの概要」
- PL/SQLサブプログラムの概要は、Oracle Database PL/SQL言語リファレンスを参照してください
- 定義者および実行者の権限についてさらに学習するには、Oracle Databaseセキュリティ・ガイドを参照してください

PL/SQLサブプログラムの作成

スタンドアロンのストアド・サブプログラムは、CREATE PROCEDURE文またはCREATE FUNCTION文を使用してスキーマ・レベルで作成されたサブプログラムです。パッケージに定義されているサブプログラムは**パッケージ・サブプログラム**と呼ばれ、パッケージの一部とみなされます。

データベースは、データ・ディクショナリにスキーマ・オブジェクトとしてサブプログラムを格納します。サブプログラムには、パラメータの記述などの仕様部と本体があります。

例8-1 PL/SQLプロシージャ

この例に、スタンドアロンのPL/SQLプロシージャhire_employeesを作成する文の一部を示します。プロシージャは、employees表に1つの行を挿入します。

```
CREATE PROCEDURE hire_employees
  (p_last_name VARCHAR2, p_job_id VARCHAR2, p_manager_id NUMBER,
   p_hire_date DATE, p_salary NUMBER, p_commission_pct NUMBER,
   p_department_id NUMBER)
IS
BEGIN
.
.
.
  INSERT INTO employees (employee_id, last_name, job_id, manager_id, hire_date,
    salary, commission_pct, department_id)
  VALUES (emp_sequence.NEXTVAL, p_last_name, p_job_id, p_manager_id,
          p_hire_date, p_salary, p_commission_pct, p_department_id);
.
.
.
END;
```

関連項目:

- サブプログラムの作成方法を学習するには、Oracle Database 2日で開発者ガイドを参照してください
- CREATE PROCEDURE文について学習するには、Oracle Database PL/SQL言語リファレンスを参照してください

PL/SQLサブプログラムの実行

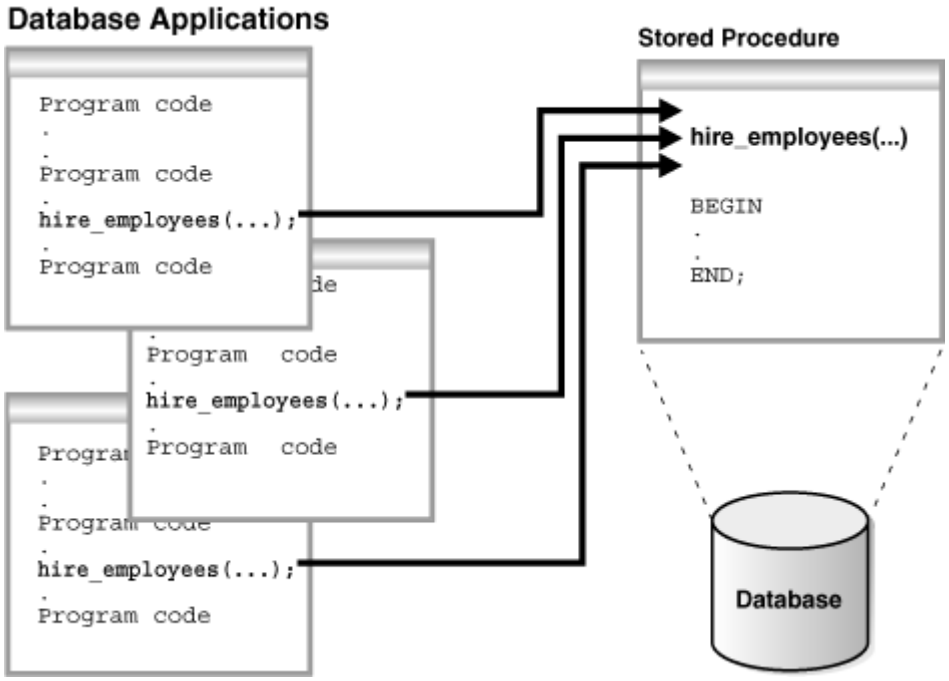
ユーザーは、複数の方法で対話的にサブプログラムを実行できます。

オプションは次のとおりです。

- SQL*PlusやSQL DeveloperなどのOracleツールを使用します
- Oracle Formsやプリコンパイラのアプリケーションなど、データベース・アプリケーションのコード内で明示的にコールします
- 別のプロシージャやトリガーのコード内で明示的にコールします。

次の図に、各種のデータベース・アプリケーションがhire_employeesをコールする状態を示します。

図8-1 ストアド・プロシージャの複数の実行



また、権限を付与されたユーザーは、Oracle Enterprise ManagerまたはSQL*Plusを使用して、次のような文でhire_employeesプロシージャを実行できます。

```
EXECUTE hire_employees ('TSMITH', 'CLERK', 1037, SYSDATE, 500, NULL, 20);
```

この文により、employees表にTSMITHのための新しいレコードが挿入されます。

ストアド・プロシージャは、プロシージャの本体で参照されるオブジェクトに依存しています。データベースは、この依存性を自動的に追跡して管理します。たとえば、プロシージャに影響を及ぼす方法でhire_employeesプロシージャが参照しているemployees表の定義を変更した場合は、プロシージャを再コンパイルして、引き続き設計どおりに機能することを確認する必要があります。通常、データベースはこのような依存性の管理を自動的に処理します。

関連項目:

- SQL*PlusおよびSQL Developerについてさらに学習するには、「データベース開発者向けのツール」を参照
- プリコンパイラについてさらに学習するには、「クライアント側のデータベース・プログラミング」を参照
- PL/SQLサブプログラムの使用方法を学習するには、Oracle Database PL/SQL言語リファレンスを参照してください
- EXECUTEコマンドについて学習するには、SQL*Plusユーザーズ・ガイドおよびリファレンスを参照してください

PL/SQLパッケージ

PL/SQLパッケージとは、関連するサブプログラムおよびこれらが使用するカーソルと変数をグループとしてまとめたもので、1単位として継続的に使用できるようにデータベースに格納されています。パッケージ化されたプログラムは、アプリケーションまたはユーザーから明示的にコールできます。

Oracle Databaseにはオラクル社が提供するパッケージが多数含まれており、これらのパッケージはデータベースの機能性を拡張してSQL機能へのPL/SQLアクセスを提供します。たとえば、UTL_HTTPパッケージを使用すると、PL/SQLおよびSQLからのHTTPコールアウトでインターネット上のデータにアクセスしたり、Oracle Web Server Cartridgesをコールできます。オラクル社が提供するパッケージは、アプリケーションの作成時や、独自のストアド・プロシージャを作成する場合に使用できます。

PL/SQLパッケージの利点

PL/SQLパッケージは、アプリケーション開発者に多くの利点を提供します。

次のような利点があります。

- カプセル化

  パッケージを使用すると、ストアド・プロシージャ、変数、データ型などに名前を付けて1つの単位としてこれらをカプセル化、つまりグループ化できます。カプセル化により開発時に優れた構成を実現でき、柔軟性も向上します。パッケージ本体を実際には作成せずに、仕様部を作成し、パブリック・プロシージャを参照するようにできます。カプセル化により、権限の管理が簡単になります。パッケージに対する権限を付与されたユーザーは、そのパッケージの構成メンバーにアクセスできるようになります。

- データ・セキュリティ

  パッケージの定義方法により、変数、カーソルおよびプロシージャをパブリックとプライベートのどちらかに指定できます。パブリックはパッケージのユーザーが直接アクセスできることを意味します。プライベートはパッケージのユーザーには非表示になっていることを意味します。

  たとえば、パッケージに10個のプロシージャが含まれているとします。この中の3つのプロシージャのみをパブリックに設定して、パッケージのユーザーが実行できるように定義できます。残りのプロシージャはプライベートなので、パッケージ内のプロシージャによってのみアクセスできます。パブリックおよびプライベートのパッケージ変数を、PUBLICへの権限付与と混同しないでください。

- 優れたパフォーマンス

  パッケージ内のプロシージャが初めてコールされた時点で、そのパッケージ全体が小さいチャンクに分割されてメモリーにロードされます。スタンドアロン・プロシージャは個別にロードする必要があるのに対し、このロードは1回の操作で完了します。関連するパッケージ・プロシージャがコールされても、メモリーにあるコンパイル済のコードを実行すると、ディスクI/Oは発生しません。

  パッケージ本体は、仕様部に影響を与えずに置換したり再コンパイルできます。その結果、パッケージ仕様部も置き換えない場合は、パッケージの構成メンバーを参照するスキーマ・オブジェクト(常に仕様部を介してアクセス)を再コンパイルする必要はありません。パッケージを使用すると、不必要な再コンパイルが最小限に抑えられるため、全体的なデータベース・パフォーマンスへの影響は少なくなります。

PL/SQLパッケージの作成

パッケージは、仕様部と本体の2つの部分に分けて作成します。パッケージの仕様部ではパッケージのすべてのパブリックな構成メンバーを宣言し、パッケージの本体ではパッケージのすべての構成メンバー(パブリックとプライベート)を定義します。

次の例は、employees_managementのパッケージ仕様を作成する文の一部で、このパッケージは、従業員データベースの管理に使用される複数のサブプログラムをカプセル化します。パッケージの各部分は、異なる文を使用して作成されます。

```
CREATE PACKAGE employees_management AS
  FUNCTION hire_employees (last_name VARCHAR2, job_id VARCHAR2, manager_id NUMBER,
    salary NUMBER, commission_pct NUMBER, department_id NUMBER) RETURN NUMBER;
  PROCEDURE fire_employees(employee_id NUMBER);
  PROCEDURE salary_raise(employee_id NUMBER, salary_incr NUMBER);
.
.
.
  no_sal EXCEPTION;
END employees_management;
```

仕様部に、ファンクションhire_employees、プロシージャfire_employeesおよびsalary_raiseと、例外であるno_salが宣言されています。パッケージへのアクセス権を持つユーザーは、これらすべてのパブリック・プログラム・オブジェクトを使用できます。

CREATE PACKAGE BODY文では、仕様部で宣言されたオブジェクトを定義します。パッケージ本体は、パッケージと同じスキーマに作成する必要があります。パッケージの作成後は、これらのうちいずれかのパブリック・プロシージャまたはファンクションをコールするか、パッケージのいずれかのパブリック例外を呼び出すアプリケーションを開発できます。

関連項目:

CREATE PACKAGE文について学習するには、Oracle Database PL/SQL言語リファレンスを参照してください

PL/SQLパッケージ・サブプログラムの実行

データベース・トリガー、ストアド・サブプログラム、3GLアプリケーション・プログラムおよびOracleツールは、パッケージの内容を参照できます。

次の図に、employees_managementパッケージ内のプロシージャおよびファンクションを呼び出すデータベース・アプリケーションを示します。

図8-2 PL/SQLパッケージ内のサブプログラムのコール

データベース・アプリケーションは、必要に応じて明示的にパッケージ・プロシージャをコールします。employees_managementパッケージに対する権限を付与されたユーザーは、そこに含まれている任意のプロシージャを明示的に実行できます。たとえば、SQL*Plusでは、次の文を発行してhire_employeesパッケージ・プロシージャを実行できます。

```
EXECUTE employees_management.hire_employees ('TSMITH', 'CLERK', 1037, SYSDATE, 500, NULL, 20);
```

関連項目:

- PL/SQLパッケージの概要は、Oracle Database PL/SQL言語リファレンスを参照してください
- PL/SQLパッケージのコーディング方法を学習するには、Oracle Database開発ガイドを参照してください

PL/SQL無名ブロック

PL/SQL無名ブロックは名前が付与されていない非永続PL/SQLユニットです。

典型的な無名ブロックの用途は次のとおりです。

- サブプログラムおよびパッケージ構成のコールの開始
- 例外処理の分離
- 他のPL/SQLブロック内でのコードのネストによる制御の管理

無名ブロックには、ストアド・サブプログラムのコードの再利用という利点はありません。表8-1に、2つのPL/SQLユニット・タイプの相違点の概要を示します。

表8-1 無名ブロックとサブプログラムの相違点

PL/SQLユニット	無名ブロック	サブプログラム
名前で指定するか	いいえ	はい
再利用ごとにコンパイルするか	いいえ	いいえ
データベースに格納されるか	いいえ	はい
他のアプリケーションから起動できるか	いいえ	はい
バインド変数値を戻すことができるか	はい	はい
関数値を戻すことができるか	いいえ	はい
パラメータを受け取ることができるか	いいえ	はい

無名ブロックは宣言部(オプション)、実行可能部分および1つ以上の例外ハンドラ(オプション)から構成されています。次のサンプルの無名ブロックでは、変数に格納する従業員の姓を選択して、その名前を出力しています。

```
DECLARE
  v_lname VARCHAR2(25);
BEGIN
  SELECT last_name
    INTO v_lname
  FROM employees
  WHERE employee_id = 101;
  DBMS_OUTPUT.PUT_LINE('Employee last name is '||v_lname);
END;
```

Oracle Databaseでは、PL/SQLブロックをコンパイルしてSGAの共有プールに配置しますが、現行インスタンス以外で再利

用するために、ソース・コードおよびコンパイル済バージョンのブロックをデータベースに格納することはありません。トリガーとは異なり、無名ブロックはメモリーにロードされるたびにコンパイルされます。共有SQLを使用すると、共有プールに入っている無名PL/SQLブロックがその共有プールからフラッシュされるまでの間は、そのブロックを再利用および共有できます。

関連項目:

無名PL/SQLブロックについてさらに学習するには、『Oracle Database開発ガイド』を参照してください

PL/SQLの言語構造

PL/SQLブロックには、様々な異なるPL/SQL言語構造を組み込むことができます。

これらの構造には、次のものが含まれます。

- 変数と定数

  プロシージャ、ファンクションまたはパッケージ内にこれらの構造を宣言できます。SQL文やPL/SQL文では、必要になった時点で値を受け渡すために変数や定数を使用できます。

- カーソル

  カーソルは、Oracle Databaseデータのレコード単位での処理を容易にするために、プロシージャ、ファンクションまたはパッケージの中で明示的に宣言できます。PL/SQLエンジンで、暗黙的にカーソルを宣言することもできます。

- 例外

  PL/SQLでは、PL/SQLコードの処理中に発生する、例外と呼ばれる内部的なエラー条件とユーザー定義のエラー条件を明示的に処理できます。

PL/SQLでは、完全なテキストが実行時まで認識されない動的SQL文を実行できます。動的SQL文は、実行時に、プログラムに入力されたりまたはプログラムにより作成される文字列に格納されます。この技法により、汎用プロシージャを作成できます。たとえば、実行時までは名前がわからない表を操作するプロシージャを作成できます。

関連項目:

- 動的SQLの詳細は、『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』を参照してください
- DBMS_SQLパッケージでの動的SQLの使用方法を学習するには、『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』を参照してください

PL/SQLのコレクションとレコード

多数のプログラミング技法は、配列、構造体、リスト、ネストした表、セットおよびツリーなどのコレクション型を使用します。

データベース・アプリケーションで収集技法をサポートするために、PL/SQLにはTABLEおよびVARRAYのデータ型が用意されています。これらの型を使用すると、連想配列、ネストした表および可変サイズ配列を宣言できます。

コレクション

PL/SQLコレクションは、すべて同じ型の要素からなる順序付けられたグループです。

各要素には、コレクション内での位置を決定する一意の添字が付いています。コレクションを作成するには、最初にコレクションの型を定義してから、その型の変数を宣言します。

コレクションの機能は、ほとんどの第3世代プログラミング言語に見られる配列と同じです。また、コレクションはパラメータとして渡すことができます。そのため、データベース表との間や、クライアント側アプリケーションとストアド・サブプログラムの間で、データの列を移動するときに使用できます。

レコード

PL/SQLレコードはC、C++またはJavaの構造型に類似した、異なる型のデータ値を格納できる複合変数です。レコードは、表内の行データまたは表内の行の特定列を保持する場合に役立ちます。

名前、給与および採用日など、従業員に関するデータがあるとします。これらの項目は、型は異なりますが論理的には関連しています。項目ごとに1フィールドを含むレコードを使用すると、データを論理単位として扱うことができます。

%ROWTYPE属性を使用すると、表の1行やカーソルからフェッチされる1行を表すレコードを宣言できます。ユーザー定義レコードの場合は、所有しているフィールドを宣言できます。

関連項目:

PL/SQLレコード方法を学習するには、『Oracle Database PL/SQL言語リファレンス』を参照してください

PL/SQLの実行方法

PL/SQLは解析済の実行およびシステム固有の実行の両方をサポートします。

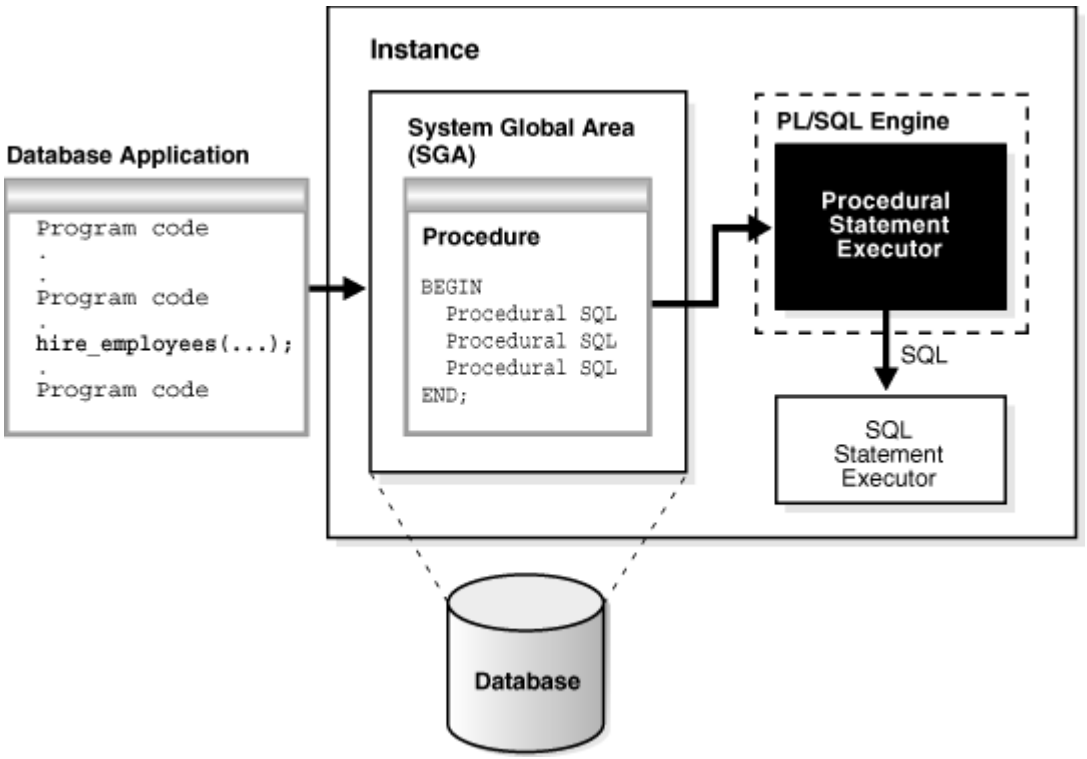
解析済の実行では、PL/SQLソース・コードがいわゆるバイトコード表現にコンパイルされます。Oracle Databaseの一部として実装されている移植可能な仮想コンピュータによって、このバイトコードが実行されます。

システム固有の実行では、計算集中型のユニットで最大限のパフォーマンスが発揮されます。この場合は、PL/SQLユニットのソース・コードが特定のプラットフォームのオブジェクト・コードに直接コンパイルされます。このオブジェクト・コードは、Oracle Databaseにリンクされます。

PL/SQLエンジンは、PL/SQLユニットを定義、コンパイルおよび実行します。このエンジンは、Oracle Databaseをはじめとする多数のOracle製品に組み込まれている特別なコンポーネントです。多くのOracle製品にPL/SQLコンポーネントが含まれていますが、このトピックでは、Oracle Databaseに格納でき、Oracle DatabaseのPL/SQLエンジンを使用して処理できるPL/SQLユニットについて説明します。各Oracleツール用のドキュメントでは、それぞれのPL/SQL機能について説明します。

次の図に、Oracle Databaseに含まれているPL/SQLエンジンを示します。

図8-3 PL/SQLエンジンとOracle Database



PL/SQLユニットは、データベースに格納されています。アプリケーションによってストアド・プロシージャがコールされると、データベースによって、コンパイル済のPL/SQLユニットがシステム・グローバル領域(SGA)内の共有プールにロードされます。PL/SQL文エグゼキュータとSQL文エグゼキュータは、連動してプロシージャ内の文を処理します。

別のPL/SQLブロック(無名ブロックまたは別のストアド・プロシージャ)からストアド・プロシージャをコールすることもできます。たとえば、Oracle Formsからストアド・プロシージャをコールできます。

Oracle Database上で実行されるPL/SQLプロシージャからは、Cプログラミング言語で作成して共有ライブラリに格納した外部プロシージャや外部ファンクションをコールできます。Cルーチンは、Oracle Databaseとは別のアドレス空間で実行されます。

関連項目:

- 共有プールの目的および内容についてさらに学習するには、「共有プール」を参照
- PL/SQLアーキテクチャについて学習するには、『Oracle Database PL/SQL言語リファレンス』を参照してください
- 外部プロシージャについてさらに学習するには、『Oracle Database開発ガイド』を参照してください

Oracle DatabaseにおけるJavaの概要

オブジェクト指向プログラミング言語の選択肢として、Javaがあります。

Javaの機能は、次のとおりです。

- Java仮想マシン(JVM)。プラットフォームの独立性を確保するための基礎を提供します。
- 自動ストレージ管理機能。ガーベージ・コレクションなどがあります。
- C言語に基づく言語構文。強い型指定が必要です。

ノート:

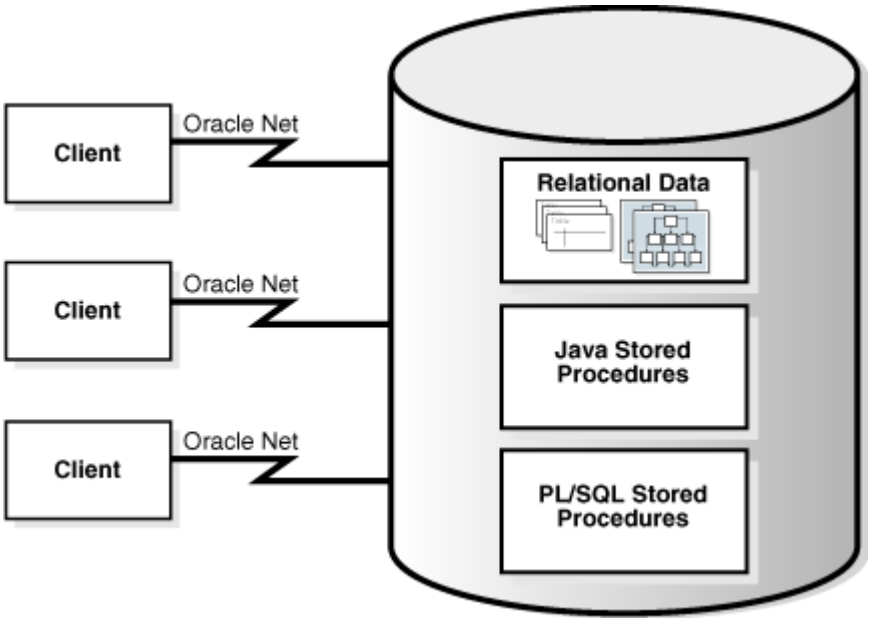
この章は、Java 言語に関するある程度の知識があることを前提としています。

データベースは、複雑な問合せおよび複数のデータ・ビューをサポートする動的データ処理エンジンを備えたJavaプログラムを提供しています。クライアント要求は、即時処理されるようにデータ問合せとして組み立てられます。問合せの結果は動的に生成されます。

JavaとOracle Databaseの組合せによって、ビジネス・ニーズの変更に応じて簡単に更新できるコンポーネントベースかつネットワーク中心のアプリケーションを作成できます。また、アプリケーションおよびデータ・ストアは、デスクトップからインテリジェント・ネットワーク上およびネットワーク中心のサーバー上に移動できます。さらに重要なのは、任意のクライアント・デバイスからこれらのアプリケーションおよびデータ・ストアにアクセスできることです。

次の図は、従来の2層のクライアント/サーバー構成を示しています。この構成では、クライアントは、PL/SQLサブプログラムをコールする際と同じ方法でJavaストアド・プロシージャをコールします。

図8-4 2層のクライアント/サーバー構成



関連項目:

Oracle DatabaseでのJavaの使用方法の概要は、『Oracle Database 2日でJava開発者ガイド』を参照してください

Java仮想マシン(JVM)の概要

JVMはコンパイル済のJavaコードを実行する仮想プロセッサです。

Javaソース・コードは、バイトコードと呼ばれる、プラットフォームに依存しない低レベルのマシン命令にコンパイルされます。Javaバイトコードは、JVMを介して、プラットフォームに依存する処理に解釈されます。

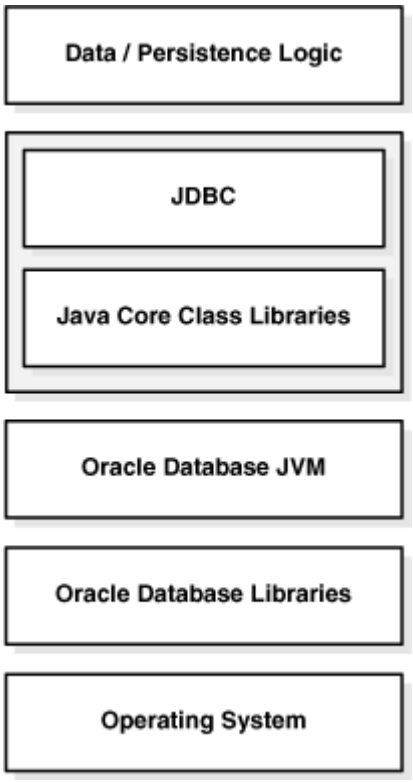
Oracle JVMの概要

Oracle JVMは、Pure Javaアプリケーションを実行する標準的なJava互換環境です。JLSおよびJVMの仕様と互換性があります。

Oracle JVMでは、標準のJavaバイナリ形式とAPIがサポートされます。また、Oracle Databaseは、実行時の動的クラス・ロードなど、標準のJava言語のセマンティクスに準拠しています。

次の図に、OracleのJavaアプリケーションがJavaコア・クラス・ライブラリの最上位に配置され、さらに、Oracle JVMの最上位に配置される様子を示します。OracleのJavaサポート・システムはデータベース内に配置されるので、JVMはオペレーティング・システムと直接対話するかわりに、データベースのライブラリと対話します。

図8-5 Javaコンポーネントの構造



他のJava環境とは異なり、Oracle JVMはOracle Database内に埋め込まれています。Oracle JVMと典型的なクライアントJVMには、いくつかの重要な違いがあります。たとえば、標準のJava環境では、コマンドラインで次のコマンドを発行し、インタプリタを介してJavaアプリケーションを実行します(*classname*は、JVMにより最初に解釈されるクラス名です)。

```
java classname
```

このコマンドによって、オペレーティング・システム上のプロセス内でアプリケーションが実行されます。ただし、コマンドライン・インタフェースを使用していない場合は、アプリケーションをデータベースにロードして、インタフェースをパブリッシュしてから、データベース・データ・ディクショナリ内のアプリケーションを実行します。

関連項目:

Oracle JVMと典型的なクライアントJVMのその他の相違の詳細は、『Oracle Database Java開発者ガイド』を参照してください

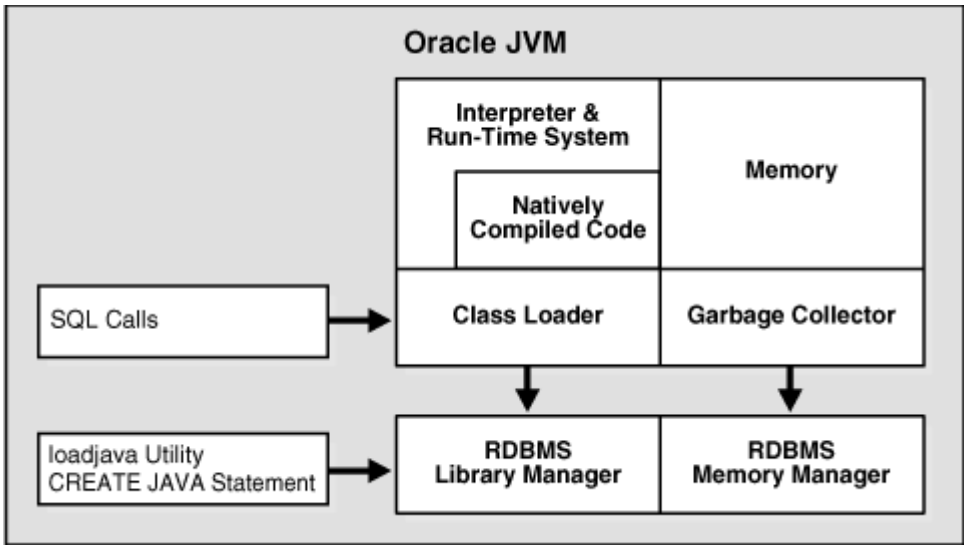
Oracle JVMの主要コンポーネント

Oracle JVMはデータベースのメモリー・ヒープを共有してリレーショナル・データに直接アクセスするため、データベース・カーネルと同じ処理領域およびアドレス領域で実行されます。この設計によりメモリーの使用を最適化して、スループットを向上させます。

Oracle JVMは、Javaオブジェクトの実行時環境を提供します。Javaのデータ構造、メソッド・ディスパッチ、例外処理および言語レベル・スレッドを完全にサポートします。また、java.lang、java.io、java.net、java.mathおよびjava.utilなど、すべてのコアJavaクラス・ライブラリもサポートしています。

次の図に、Oracle JVMの主要なコンポーネントを示します。

図8-6 Oracle JVMの主要コンポーネント



Oracle JVMは、データベース・スキーマに標準のJavaネームスペースを埋め込みます。この機能により、Javaプログラムは Oracle Databaseに格納されたJavaオブジェクトおよび企業内のアプリケーション・サーバーにアクセスできます。

また、Oracle JVMは、データベースのスケーラブルな共有メモリー・アーキテクチャと緊密に統合されています。Javaプログラムは、ユーザーの介入なしで効率的にコール、セッションおよびオブジェクトの存続期間を使用します。結果として、セッションが継続状態であっても、Oracle JVMおよび中間層のJavaビジネス・オブジェクトのスケーラビリティを維持できます。

関連項目:

Oracle JVMの主要なコンポーネントの詳細は、Oracle Database Java開発者ガイドを参照してください

Javaプログラミング環境

オラクル社は、エンタープライズ・アプリケーションの開発者に、Javaアプリケーションの作成、デプロイおよび管理のためのエンドツーエンドのJavaソリューションを提供します。

このソリューションは、クライアント側とサーバー側のプログラム・インタフェース、Java開発をサポートするためのツール、Oracle Databaseと統合されたJava仮想マシンで構成されています。これらの製品はいずれも、Java標準との互換性を持っています。

Javaプログラミング環境は、次の追加機能で構成されています。

- PL/SQLに等価および相当するJavaストアド・プロシージャ。Javaストアド・プロシージャは、PL/SQLと緊密に統合されています。Javaストアド・プロシージャはPL/SQLパッケージから、プロシージャはJavaストアド・プロシージャからコールできます。
- SQLデータにアクセスするためのJDBCおよびSQLJプログラミング・インタフェース。
- クラスの開発、ロードおよび管理を支援するツールおよびスクリプト。

Javaストアド・プロシージャ

Javaストアド・プロシージャは、SQLにパブリッシュされデータベースに格納されるJavaメソッドです。

PL/SQLサブプログラムと同様に、Javaプロシージャには、SQL*Plusなどの製品で直接起動する方法と、トリガーを使用して間接的に起動する方法があります。また、OCI、プリコンパイラまたはJDBCなど、すべてのOracle Netクライアントからアクセスできます。

Javaメソッドをパブリッシュするには、Javaメソッド名、パラメータの型およびSQLの対応部分への戻り型をマップするコール仕様を記述します。クライアント・アプリケーションからコールされると、Javaストアド・プロシージャは引数を受け取り、Javaクラスを参照してJavaの結果値を戻すことができます。

アプリケーションは、コール仕様の名前を参照することにより、Javaメソッドをコールします。実行時のシステムは、Oracleデータ・ディクショナリ内のコール仕様の定義を検索し、該当のJavaメソッドを実行します。

また、Javaを使用して、PL/SQLに依存しない強力なプログラムを開発することもできます。Oracle Databaseには、Javaプログラミング言語とJVMの完全準拠の実装が用意されています。

関連項目:

Javaでストアド・プロシージャを記述する方法、PL/SQLからアクセスする方法およびJavaからPL/SQL機能にアクセスする方法は、Oracle Database Java開発者ガイドを参照してください

JavaおよびPL/SQLの統合

既存のPL/SQLプログラムをJavaから呼び出したり、JavaプログラムをPL/SQLから起動できます。このソリューションでは、PL/SQLおよびJavaコードを保護して活用します。

Oracle Databaseは、JavaからSQLデータにアクセスするために、JDBCおよびSQLJという異なる2つのアプローチを提供しています。JDBCはクライアントおよびサーバーの両方で使用可能ですが、SQLJはクライアントのみで使用可能です。

JDBCドライバ

JDBCは、ユーザーがデータベースに接続してSQL文を実行し、データベースに問合せできるようにするデータベース・アクセス・プロトコルです。

コアJavaクラス・ライブラリには、JDBC APIのjava. sqlが1つのみ用意されています。ただし、JDBCは、ベンダーが特定のデータベースを必要に応じて特化するためのドライバを提供できるように設計されています。Oracleには、次の表に示す個々のJDBCドライバが用意されています。

表8-2 JDBCドライバ

ドライバ	説明
JDBC Thin ドライバ	JDBC Thin ドライバを使用すると、Oracle SQL のデータにアクセスする Pure Java アプリケーションおよびアプレットを作成できます。JDBC Thin ドライバは、他の Java アプレットと同様に Web ページから動的にダウンロードできるため、Web ベースのアプリケーションとアプレットに特に適しています。
JDBC OCI ドライバ	JDBC OCI ドライバは、クライアントまたは中間層にある Oracle 固有のネイティブ・コード(つまり非 Java コード)およびライブラリにアクセスし、JDBC Thin ドライバに比べてパフォーマンスが向上しますが、サイズが極端に大きくなることと、クライアント側でのインストールというコストの問題があります。
サーバー側 JDBC 内部ドライバ	Oracle Database は、Java コードがサーバー上で実行される場合に、サーバー側内部ドライバを使用します。これにより、サーバーの Oracle JVM 内で実行中の Java アプリケーションは、JDBC を使用してローカルに定義されたデータ(つまり、同じシステムの同じプロセスにある

ドライバ	説明
	データ)にアクセスできます。また、基礎となる Oracle RDBMS ライブラリを直接使用できるため、パフォーマンスが向上し、Java コードと SQL データの間にネットワーク接続が介入することによるオーバーヘッドは発生しません。Oracle Database では、サーバー上で同じ Java-SQL インタフェースがサポートされるため、デプロイ時にコードの再処理を必要としません。

関連項目:

- 「ODBCおよびJDBC」
- Oracle Database 2日でJava開発者ガイドおよびOracle Database JDBC開発者ガイド

SQLJ

SQLJは、JavaプログラムにSQL文を埋め込むためのANSI標準です。クライアント側のSQLJプログラムを使用できます。また、SQLJプログラムをJDBCと組み合せることもできます。

ノート:

Oracle Database 12c リリース 2 (12.2)以降では、データベース環境でのストアド・プロシージャ、関数およびトリガーの実行も含めて、サーバー側 SQLJ コードの実行はサポートされません。

SQLJは、Javaからデータベースにアクセスするクライアント側および中間層のアプリケーションを開発するための、単純ですが強力な方法を提供します。開発者はSQLJを使用してプログラムを記述し、その後SQLJトランスレータを使用して埋め込まれたSQLをJDBCベースのPure Javaコードに変換します。実行時に、プログラムは標準のJDBCドライバを使用して、複数のベンダーのデータベースと通信できます。

次の例は、簡単な実行可能SQLJ文を示しています。

```
String name;
#sql  { SELECT first_name INTO :name FROM employees WHERE employee_id=112 };
System.out.println("Name is " + name + ", employee number = " + employee_id);
```

関連項目:

- SQLJ
- 『Oracle Database SQLJ開発者ガイド』

トリガーの概要

データベースの**トリガー**はコンパイル済のストアド・プログラム・ユニットであり、PL/SQLまたはJavaのいずれかで記述されます。Oracle Databaseは、特定の状況で自動的にトリガーの呼出し(起動)を行います。

トリガーは、次のいずれかの操作が発生するたびに起動します。

1. 任意のユーザーによって発行された、特定の表またはビューに対するDML文

   DML文は、スキーマ・オブジェクト内のデータを変更します。たとえば、行の挿入および削除はDMLの処理です。

2. 特定のユーザーまたは任意のユーザーのいずれかが発行したDDL文

   DDL文は、スキーマ・オブジェクトを定義します。たとえば、表の作成および列の追加はDDLの処理です。

3. データベース・イベント

   ユーザーのログインまたはログオフ、エラーおよびデータベースの起動または停止は、トリガーを起動できるイベントです。

トリガーはサブプログラムに類似したスキーマ・オブジェクトですが、起動方法が異なります。サブプログラムはユーザー、アプリケーションまたはトリガーによって明示的に実行されます。トリガーは、トリガー・イベントが発生したときにデータベースによって暗黙的に起動されます。

関連項目:

- DMLおよびDDLについて学習するには、「SQL文の概要」を参照してください
- 「データベース・インスタンスの起動と停止の概要」

トリガーの利点

トリガーを正しく使用すると、ユーザーはデータベースをより効率的に使用するさらに堅牢なアプリケーションを構築してデプロイできます。

トリガーは、次のような目的で使用できます。

- 導出列の値の自動的な生成
- 不正なトランザクションの防止
- 監査およびイベント・ロギングの提供
- 表アクセスについての情報の記録

トリガーを使用して、すべてのクライアント・アプリケーションに共通の低レベルでのビジネス・ルールを強制実行できます。たとえば、複数のアプリケーションからemployees表にアクセスできます。この表でのトリガーで挿入されるデータの形式が保証される場合は、クライアントごとにこのビジネス・ロジックを再作成する必要はありません。アプリケーションはトリガーを回避できないため、トリガー内のビジネス・ロジックが自動的に使用されます。

トリガーと整合性制約を併用すると、任意の整合性規則を定義して規定できます。ただし、整合性制約を使用して定義できない複雑なビジネス・ルールを強制する場合にのみ、トリガーを使用することをお薦めします。

トリガーを使用しすぎると相互依存性が複雑になるため、大規模アプリケーションのメンテナンスが困難になります。たとえば、トリガーが起動されると、そのトリガー・アクションに含まれるSQL文によって他のトリガーが起動されてトリガーの連鎖状態になり、意図しない影響が起きる可能性があります。

関連項目:

- 「データ整合性の概要」

- 『Oracle Database 2日で開発者ガイド』
- アプリケーションのトリガーを計画する場合のガイドラインおよび制限の詳細は、Oracle Database PL/SQL言語リファレンスを参照してください

トリガーのタイプ

トリガーは、起動方法と実行するアクションのタイプに応じて分類できます。

Oracle Databaseは次のタイプのトリガーをサポートしています。

- 行トリガー

  行トリガーは、表がトリガーを実行する文の処理の影響を受けるたびに実行されます。たとえば、文が複数の行を更新する場合、UPDATEが処理する行ごとに1つずつ行トリガーが起動されます。トリガーを実行する文の影響を受ける行がなければ、行トリガーは実行されません。トリガー・アクションのコードが、トリガーを実行する文によって供給されるデータまたは影響を受ける行数に依存する場合には、行トリガーが便利です。

- 文トリガー

  文トリガーは、トリガーを実行する文の影響を受ける行数とは関係なく、トリガーを実行する文の実行ごとに1回起動されます。たとえば、文が表から100行を削除すると、文レベルのDELETEトリガーが1回のみ起動されます。トリガー・アクションのコードが、トリガーを実行する文によって供給されるデータや、影響を受ける行に依存しない場合には、文トリガーが便利です。

- INSTEAD OFトリガー

  INSTEAD OFトリガーは、トリガーを実行する文のかわりに、Oracle Databaseによって起動されます。これらのトリガーは、DML文を介して直接変更できないビューを透過的に変更する場合に便利です。

- イベント・トリガー

  トリガーを使用すると、データベース・イベントに関する情報を、サブスクライバに対して発行できます。イベント・トリガーは、次のカテゴリに分類されます。

  - システム・イベント・トリガー: データベース・インスタンスの起動および停止またはエラー・メッセージなどのイベントによって起動できます。
  - ユーザー・イベント・トリガー: ユーザーのログオンおよびログオフに関連したイベント、DDL文およびDML文によって起動されます。

関連項目:

- 『Oracle Database 2日で開発者ガイド』
- Oracle Database PL/SQL言語リファレンス

トリガーのタイミング

トリガーのタイミング(トリガー・アクションをトリガー実行文の前と後のどちらで実行するか)を定義できます。

単純なトリガーは表の単一のトリガーであり、次のいずれか1つのタイミング・ポイントに対してのみアクションを指定できます。

- 起動文の前
- 起動文によって影響を受ける各行の前
- 起動文によって影響を受ける各行の後
- 起動文の後

文および行のトリガーの場合、BEFOREトリガーによってセキュリティを高め、データベースに変更を加える前にビジネス・ルールを有効にできます。アクションのロギングには、AFTERトリガーが理想的です。

複合トリガーは、複数のタイミング・ポイントで起動できます。複合トリガーは、様々なタイミング・ポイントに対して実装するアクションで共通データを共有するためのアプローチをプログラム化する際に役立ちます。

関連項目:

複合トリガーについて学習するには、Oracle Database PL/SQL言語リファレンスを参照してください

トリガーの作成

CREATE TRIGGER文はデータベース・トリガーを作成または置き換えます。

PL/SQLトリガーの一般的な構文形式は次のとおりです。

```
CREATE TRIGGER trigger_name
  triggering_statement
  [trigger_restriction]
BEGIN
 triggered_action;
END;
```

PL/SQLトリガーの基本的なコンポーネントは次のとおりです。

- トリガー名

  名前は、同じスキーマ内の他のトリガーの中で一意であることが必要です。たとえば、part_reorder_triggerなどの名前にできます。

- トリガー・イベントまたはトリガーを実行する文

  トリガー・イベントまたはトリガーを実行する文とは、トリガーを起動させるSQL文、データベース・イベントまたはユーザー・イベントのことです。たとえば、ユーザーによる表の更新などです。

- トリガー制限

  トリガー制限にはブール式を指定し、トリガーが起動されるには、そのブール式がtrueになる必要があります。たとえば、引当可能部品の数量が現行の再発注数量より少なくなるまで、トリガーは起動されません。

- トリガー・アクション

  トリガー・アクションは、トリガーを実行する文が発行され、トリガーの制限がtrueに評価された場合に実行されるSQL文およびコードを含むプロシージャです。たとえば、ユーザーが保留中のorders表に行を挿入します。

関連項目:

- トリガーの作成方法を学習するには、Oracle Database 2日で開発者ガイドおよびOracle Database PL/SQL

言語リファレンスを参照してください

- CREATE TRIGGER文について学習するには、Oracle Database SQL言語リファレンスを参照してください

例: CREATE TRIGGER文

この例では、明細項目表でINSERT文、UPDATE文またはDELETE文が実行されると起動されるトリガーを作成します。

次の文を使用してorders表およびlineitems表を作成するとします。orders表は一意のオーダーごとに行を保持するのに対して、lineitems表は1つのオーダー内の項目ごとに行を保持します。

```
CREATE TABLE orders
( order_id NUMBER PRIMARY KEY,
  /* other attributes */
  line_items_count NUMBER DEFAULT 0 );

CREATE TABLE lineitems
( order_id REFERENCES orders,
  seq_no    NUMBER,
  /* other attributes */
  CONSTRAINT lineitems PRIMARY KEY(order_id,seq_no) );
```

次の文で、orders表内のオーダーの項目数を自動的に更新するサンプル・トリガーを作成します。

```
CREATE OR REPLACE TRIGGER lineitems_trigger
  AFTER INSERT OR UPDATE OR DELETE ON lineitems
  FOR EACH ROW
BEGIN
  IF (INSERTING OR UPDATING)
  THEN
    UPDATE orders SET line_items_count = NVL(line_items_count,0)+1
    WHERE order_id = :new.order_id;
  END IF;
  IF (DELETING OR UPDATING)
  THEN
    UPDATE orders SET line_items_count = NVL(line_items_count,0)-1
    WHERE order_id = :old.order_id;
  END IF;
END;
/
```

lineitems_triggerでトリガーを実行する文は、lineitems表に対するINSERT、UPDATEまたはDELETEです。トリガー制限はありません。変更される行ごとに、トリガーが起動されます。トリガーは、トリガーを実行する文によって影響を受ける現在の行の、新旧の列値にアクセスできます。表の変更対象の列ごとに、古い値(:old)および新しい値(:new)の2つの相関名があります。セッションで、あるオーダーについてlineitemsの行を更新または挿入すると、このアクションの後、トリガーによってこのオーダーの項目数が計算され、orders表の数値が更新されます。

例: 行レベル・トリガーの起動

このシナリオでは、顧客は2つのオーダーを開始し、オーダーの明細項目を追加および削除します。

シナリオは、「例: CREATE TRIGGER文」で作成されたトリガーに基づいています。

表8-3 行レベル・トリガーの使用例

SQL文	トリガーされるSQL文	説明

SQL文	トリガーされるSQL文	説明
SQL> INSERT INTO orders (order_id) VALUES (78); 1 row created.		顧客が ID78 のオーダーを作成します。この時点で、顧客のオーダーに項目はありません。 lineitems 表ではアクションが実行されないため、このトリガーは起動されません。
SQL> INSERT INTO orders (order_id) VALUES (92); 1 row created.		顧客が ID92 の別のオーダーを作成します。この時点で、顧客のオーダーに項目はありません。 lineitems 表ではアクションが実行されないため、このトリガーは起動されません。
SQL> INSERT INTO lineitems (order_id, seq_no) VALUES (78,1); 1 row created.	UPDATE orders SET line_items_count = NVL(NULL,0)+1 WHERE order_id = 78;	顧客がオーダー78 に項目を追加します。 INSERT によって、トリガーが起動されます。トリガーされた文が、オーダー78 の明細項目数を 0 から 1 に増やします。
SQL> INSERT INTO lineitems (order_id, seq_no) VALUES (78,2); 1 row created.	UPDATE orders SET line_items_count = NVL(1,0)+1 WHERE order_id = 78;	顧客がオーダー78 にさらに項目を追加します。 INSERT によって、トリガーが起動されます。トリガーされた文が、オーダー78 の明細項目数を 1 から 2 に増やします。
SQL> SELECT * FROM orders; ORDER_ID LINE_ITEMS_COUNT ---------- ---------- 78 2 92 0		顧客が 2 つのオーダーのステータスを問い合せます。オーダー78 には 2 つの項目があります。オーダー92 には項目がありません。
SQL> SELECT * FROM lineitems; ORDER_ID SEQ_NO ---------- ---------- 78 1 78 2		顧客は明細項目のステータスを問い合せます。各項目は注文 ID と順序番号によって一意に識別されます。
SQL> UPDATE lineitems SET order_id = 92; 2 rows updated.	UPDATE orders SET line_items_count = NVL(NULL,0)+1 WHERE order_id = 92; UPDATE orders SET line_items_count = NVL(2,0)-1	顧客がオーダー78 の明細項目をオーダー92 に移動します。 UPDATE 文は lineitems 表の 2 つの

SQL文	トリガーされるSQL文	説明
	WHERE order_id = 78; UPDATE orders SET line_items_count = NVL(1,0)+1 WHERE order_id = 92; UPDATE orders SET line_items_count = NVL(1,0)-1 WHERE order_id = 78;	行を変更します。このとき、各行に 1 回ずつトリガーが起動されます。 トリガーが起動されるたびに、トリガーの両方の IF 条件が満たされます。最初の条件はオーダー92 の数を増加させるのに対して、2 番目の条件はオーダー78 の数を減少させます。これにより、合計で 4 つの UPDATE 文が実行されます。
SQL> SELECT * FROM orders; ORDER_ID LINE_ITEMS_COUNT -------- - ---------- 78 0 92 2		顧客が 2 つのオーダーのステータスを問い合せます。実際の結果としては、オーダー92 の明細項目数が 0 から 2 に増加し、オーダー78 の数が 2 から 0 に減少しています。
SQL> SELECT * FROM lineitems; ORDER_ID SEQ_NO ---------- ------- --- 92 1 92 2		顧客は明細項目のステータスを問い合せます。各項目は注文 ID と順序番号によって一意に識別されます。
SQL> DELETE FROM lineitems; 2 rows deleted.	UPDATE orders SET line_items_count = NVL(2,0)-1 WHERE order_id = 92; UPDATE orders SET line_items_count = NVL(1,0)-1 WHERE order_id = 92;	顧客はすべてのオーダーからすべての明細項目を削除します。 DELETE 文は lineitems 表の 2 つの行を変更し、このとき、各行に 1 回ずつトリガーが起動されます。トリガーの起動ごとに、トリガーの片方の IF 条件のみが満たされます。毎回、条件によってオーダー92 の数を 1 つずつ減らします。これにより、合計で 2 つの UPDATE 文が実行されます。
SQL> SELECT * FROM orders; ORDER_ID LINE_ITEMS_COUNT -------- - ---------- 78 0 92 0 SQL> SELECT * FROM lineitems; no rows selected		顧客が 2 つのオーダーのステータスを問い合せます。どちらのオーダーにも明細項目がありません。 また、顧客は明細項目のステータスも問い合せます。項目は存在しません。

トリガーの実行

Oracle Databaseは、サブプログラムの実行と同じステップを使用してトリガーを内部的に実行します。

両者の唯一の違いは、ユーザー・アカウントがトリガー文を実行する権限を持っていれば、トリガーを起動する権限が付与されることです。この例外を除いて、データベースはストアド・サブプログラムと同じ方法でトリガーを検証して実行します。

関連項目:

トリガー実行についてさらに学習するには、Oracle Database PL/SQL言語リファレンスを参照してください

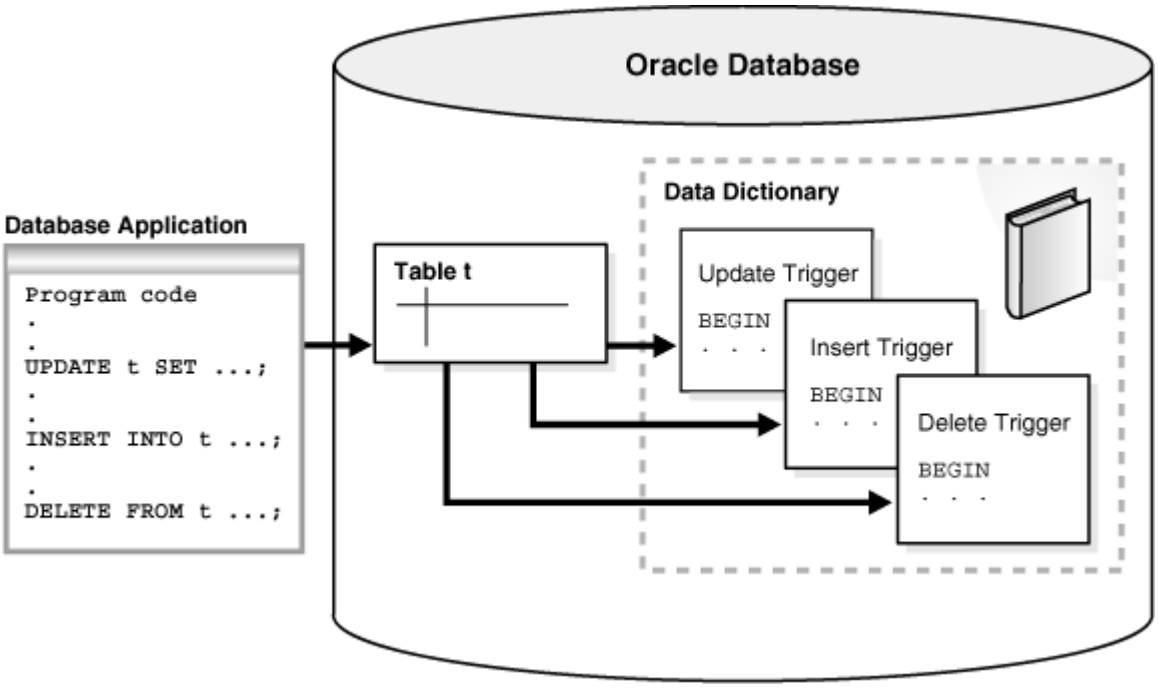
トリガーの記憶域

Oracle Databaseでは、PL/SQLストアド・プロシージャと同じように、コンパイル済の形式でデータベース・スキーマにPL/SQLトリガーを格納します。

CREATE TRIGGER文がコミットすると、コンパイルされたPL/SQLコードがデータベースに格納されます。共有プールからPL/SQLトリガーのソース・コードが削除されます。

次の図に示すデータベース・アプリケーションには、PL/SQLトリガーを暗黙のうちに起動するSQL文が含まれています。トリガーは、それに対応付けられている表とは別々に格納されています。

図8-7 トリガー



Javaトリガーは、PL/SQLトリガーと同じ方法で格納されます。ただし、Javaトリガーでは、CALL文を使用して、個別にコンパイルされたJavaコードを参照します。このため、Javaトリガーを作成する場合は、JavaコードおよびこのJavaコードを参照するトリガーを作成することが必要となります。

関連項目:

トリガーのコンパイルおよび格納について学習するには、Oracle Database PL/SQL言語リファレンスを参照してください

第III部 Oracleトランザクションの管理

トランザクションの管理とは、データの同時実行性と整合性を確保するためにトランザクションを使用することです。

この部は、次の章で構成されています。

- データの同時実行性と整合性
- トランザクション

9 データの同時実行性と整合性

この章では、マルチユーザー・データベース環境でのOracle Databaseによるデータ整合性の維持について説明します。

この章の構成は、次のとおりです。

- データの同時実行性と整合性の概要
- Oracle Databaseのトランザクション分離レベルの概要
- Oracle Databaseのロック・メカニズムの概要
- 自動ロックの概要
- 手動データ・ロックの概要
- ユーザー定義ロックの概要

データの同時実行性と整合性の概要

シングル・ユーザーのデータベースでは、他のユーザーが同時に同じデータを変更することがないため、ユーザーは何も心配せずにデータを変更できます。ただし、マルチユーザー・データベースでは、複数のトランザクション内の文によって、同じデータが同時に更新される可能性があります。同時に実行される複数のトランザクションでは、意味のある一貫した結果を出すことが必要です。

マルチユーザー・データベースには次の機能が備わっている必要があります。

- 複数のユーザーが同時にデータにアクセスできることの保証(データ同時実行性)
- ユーザー自身のトランザクションや他のユーザーのトランザクションによる参照可能な変更を含め、各ユーザーにデータの一貫したビューが表示されることの保証(データ整合性)

トランザクションが同時に実行されるときの一貫したトランザクション動作を表すために、データベース調査者は、シリアライズ可能性と呼ばれるトランザクション分離モデルを定義しています。シリアライズ可能トランザクションは、他のユーザーがデータベースのデータを変更していないように認識される環境内で動作します。

一般に、この程度までのトランザクションの分離が望ましいとされますが、シリアライズ可能モードで数多くのアプリケーションを実行すると、アプリケーションのスループットがかなり低下する可能性があります。同時に実行される各トランザクションの完全な分離とは、あるトランザクションが問合せを実行している表に対して、他のトランザクションが挿入を実行できない状態を指します。つまり、現実には、トランザクションの完全な分離とパフォーマンスとの間の妥協点を考慮する必要があります。

Oracle Databaseでは、マルチバージョン一貫性モデルや様々なタイプのロックおよびトランザクションを使用することによって、データ整合性を維持します。これにより、データベースが複数の同時ユーザーに対してデータのビューを提示する場合、それぞれ特定の時点において一貫性があるビューになります。異なるバージョンのデータ・ブロックが同時に存在する場合があるため、トランザクションでは問合せに要求された時点でコミットされているデータのバージョンが読み取られ、ある特定の時点において一貫性のある結果が戻されます。

関連項目:

「データ整合性」および「トランザクション」

マルチバージョン読取り一貫性

Oracle Databaseでは、マルチバージョン機能によって、複数のバージョンのデータが同時にマテリアライズされます。Oracle Databaseでは、マルチバージョン読取り一貫性を維持します。

Oracleデータベースの問合せには、次の特性があります。

- 読取り一貫性問合せ

  1つの問合せによって戻されるデータは、コミット済であり、ある特定の時点での一貫性を保っています。

  ノート:

  Oracle Database では、他のトランザクションにおいて未コミットのデータをトランザクションが読み取る場合に発生する、内容を保証しない読取りは許可されていません。

  内容を保証しない読取りの問題を具体的に説明するため、あるトランザクションが列値を更新し、その値がまだコミットされていない場合を考えます。2つ目のトランザクションによって、内容が保証されない更新された値(未コミットの値)が読み取られます。その後、1つ目のセッションによってトランザクションがロールバックされ、列の値が元の値に戻された場合でも、2つ目のトランザクションでは更新された値が引き続き使用されるため、データベースが破損します。内容を保証しない読取りでは、データ整合性が損われ、外部キーに違反し、一意制約が無視されます。

- 非ブロック化問合せ

  データの読取り側と書込み側は、互いにブロックすることはありません。

関連項目:

「ロック動作の概要」

文レベルの読取り一貫性

Oracle Databaseでは、**文レベルの読取り一貫性**を常に適用することにより、1つの問合せによって戻されたデータがコミット済で、ある特定の時点で一貫していることを保証します。

単一のSQL文の一貫性が保たれる基準となる時点は、トランザクション分離レベルおよび問合せの内容に応じて異なります。

- コミット読取り分離レベルでは、この時点は文が開始されたときになります。たとえば、SCN 1000でSELECT文が開始された場合、この文ではSCN 1000の時点での一貫性が保たれます。
- シリアライズ可能トランザクションまたは読取り専用トランザクションでは、この時点はトランザクションが開始されたときになります。たとえば、複数のSELECT文が実行されるトランザクションがSCN 1000で開始された場合、各文ではSCN 1000の時点での一貫性が保たれます。
- フラッシュバック問合せ操作(SELECT ... AS OF)では、SELECT文で特定の時点が明示的に指定されます。たとえば、先週の木曜日の2:00 p.m時点の表を問い合せることができます。

関連項目:

フラッシュバック問合せについて学習するには、Oracle Database開発ガイドを参照してください

トランザクション・レベルの読取り一貫性

Oracle Databaseは、トランザクションのすべての問合せに対して、**トランザクション・レベルの読取り一貫性**と呼ばれる読取り一貫性を実現することもできます。

この場合、トランザクション内の各文には同じ時点のデータが表示されます。これは、トランザクション開始時点です。

シリアライズ可能トランザクションによる問合せでは、そのトランザクション自体が変更したデータも参照できます。たとえば、employeesを更新し、employeesを問い合せたトランザクションには、その更新内容が反映されます。トランザクション・レベルの読取り一貫性によってリピータブル・リードが実現され、問合せで仮読取りが検索されなくなります。

読取り一貫性とUNDOセグメント

マルチバージョン読取り一貫性モデルを管理するには、表が同時に問合せおよび更新された場合に、読取り一貫性のある一連のデータがデータベースで作成される必要があります。

Oracle Databaseは、UNDOデータを使用して読取り一貫性を実現しています。

ユーザーがデータを変更すると、常にOracle DatabaseによってUNDOエントリが作成されて、UNDOセグメントに書き込まれます。UNDOセグメントには、未コミットのトランザクションや、最近コミットされたトランザクションによって変更されたデータの古い値が含まれています。このため、同じデータの異なる時点の複数のバージョンがデータベース内に存在することになります。データの読取り一貫性ビューを提供し、非ブロック化問合せを可能にするために、データベースでは異なる時点のデータのスナップショットを使用できます。

読取り一貫性は、単一のインスタンス内、およびOracle Real Application Clusters(Oracle RAC)環境内で保証されています。Oracle RACでは、データ・ブロックの読取り一貫性イメージをデータベース・インスタンス間で転送するために、キャッシュ・フュージョンと呼ばれるキャッシュ間ブロック転送メカニズムを使用します。

関連項目:

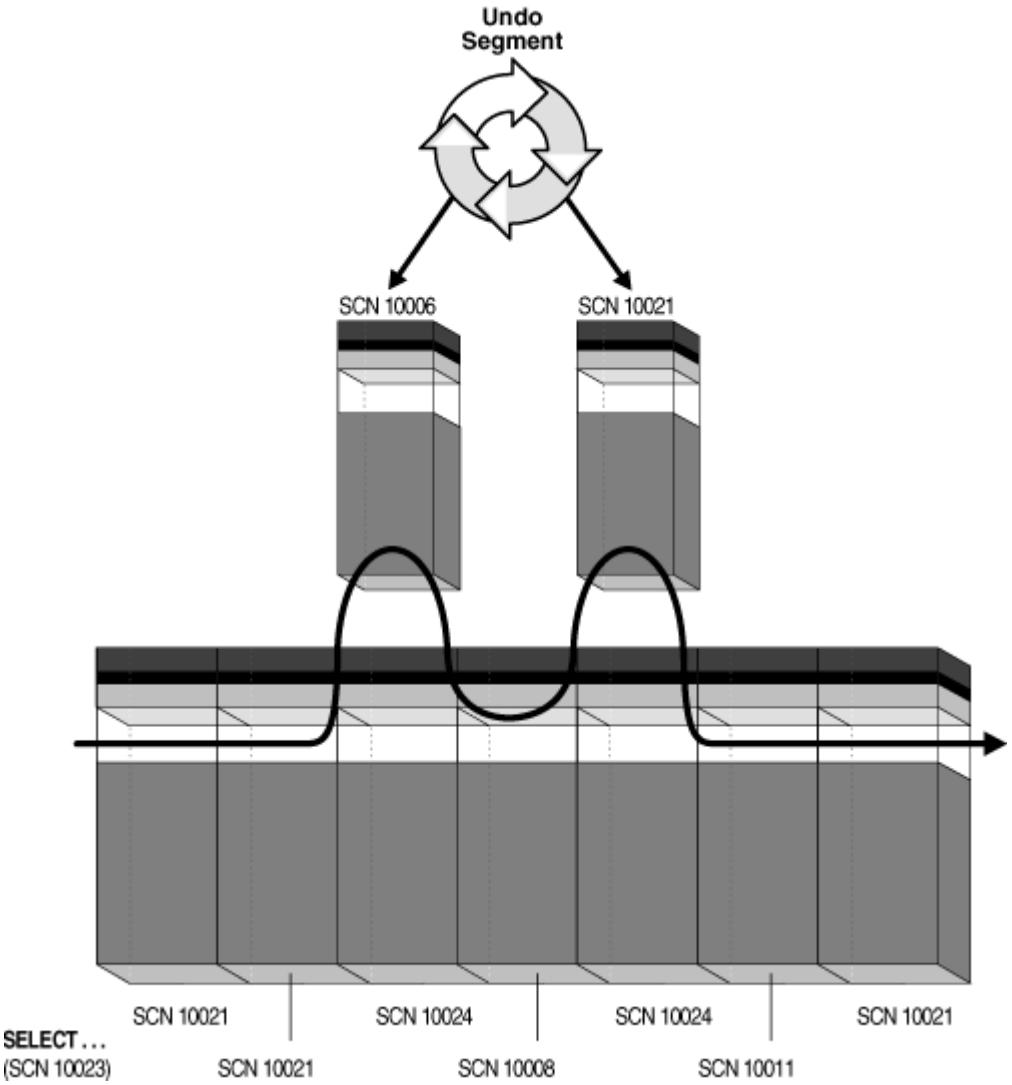
- 「UNDOセグメント」

  UNDO記憶域について学習するには
- LOBの読取り一貫性メカニズムについて学習するには、「内部LOB」を参照してください

読取り一貫性: 例

この例は、コミット読取り分離レベルで、UNDOデータを使用して文レベルの読取り一貫性を提供する問合せの例を示しています。

図9-1 コミット読取り分離レベルでの読取り一貫性



問合せによってデータ・ブロックを取得する場合、データベースによって、各ブロックのデータが問合せ開始時のブロックの内容を反映したものになることが保証されます。必要に応じて、ブロックに対する変更がロールバックされ、ブロックが問合せ処理の開始時点の内容に再構成されます。

データベースでは、SCNと呼ばれる内部順序付けメカニズムを使用して、トランザクションの順序が保証されます。SELECT文が実行フェーズに移行すると、データベースによって、問合せの実行開始時点で記録されたSCNが特定されます。図9-1では、このSCNは10023です。問合せでは、SCN 10023に対応したコミット済データが戻されます。

図9-1では、10023よりも後のSCNを持つブロック(SCN 10024の2つのブロック)は変更済データであることが示されています。この場合、SELECT文では、コミット済の変更と一貫性があるバージョンのブロックが必要となります。データベースでは、現在のデータ・ブロックが新しいバッファにコピーされ、前のバージョンのブロックを再構築するためにUNDOデータが適用されます。このように再構築されたデータ・ブロックは、読取り一貫性*(CR)*クローンと呼ばれます。

図9-1では、データベースによって2つのCRクローンが作成されており、1つのブロックではSCN 10006との一貫性が保たれ、もう1つのブロックではSCN 10021との一貫性が保たれています。問合せに対しては、この再構築されたデータが戻されます。Oracle Databaseでは、このようにして内容を保証しない読取りが回避されています。

関連項目:

「データベース・バッファ・キャッシュ」および「システム変更番号(SCN)」

読取り一貫性とInterested Transaction List

各セグメント・ブロックの**ブロック・ヘッダー**には**Interested Transaction List (ITL)**が含まれています。

データベースではITLを使用して、ブロックの変更が開始された時点でトランザクションがコミットされているかどうかが判断されます。

ITLのエントリは、ロックされた行のあるトランザクション、およびブロック内のコミット済の変更や未コミットの変更を含む行を表しています。ITLは、データベースに対する変更が行われたタイミングの情報を提供するUNDOセグメント内のトランザクション表を指しています。

ある意味で、ブロック・ヘッダーには、ブロック内の各行に影響を与えたトランザクションの最新の履歴が含まれていると考えられます。CREATE TABLE文およびALTER TABLE文のINITRANSパラメータは、保持するトランザクション履歴の量を制御します。

関連項目:

INITRANSパラメータについて学習するには、Oracle Database SQL言語リファレンスを参照してください。

読取り一貫性および遅延挿入

遅延挿入と呼ばれる特殊なタイプの挿入では、標準の読取り一貫性メカニズムは使用されません。

遅延挿入では、MEMOPTIMIZE_WRITEヒントを使用して、MEMOPTIMIZE FOR WRITEとして指定された表に挿入します。データベースではこれらの挿入を、バッファ・キャッシュではなくラージ・プールでバッファします。データベースでは、REDOおよびUNDOを使用して変更を追跡することはありません。かわりに、領域管理コーディネータ(SMCO)がバッファをディスクに書き込むと、データベースにより変更が自動的にコミットされます。この変更は、ロールバックできません。

遅延挿入は、従来の挿入と次の点で大きく異なります。

- アプリケーションではコミット済となっている、ラージ・プールに存在するデータが失われる可能性があります。たとえば、変更が保存されたことがアプリケーションにより報告された場合でも、変更がディスクに保存される前にデータベース・インスタンスが失敗する場合があります。
- データをメモリーから直接読み取ることはできません。ライターは、バックグラウンド・プロセスが変更をディスクに書き込むまで、自身の変更を読み取ることができません。読取り側は、コミットされた変更がディスクに書き込まれるまでその変更を参照できません。

データ消失を回避する必要があるクライアント・アプリケーションでは、ラージ・プールへの書込み後にデータのローカル・コピーを保持する必要があります。クライアントはDBMS_MEMOPTIMIZEパッケージを使用してメモリーへの書込みの永続性を追跡し、DBMS_MEMOPTIMIZE_ADMINパッケージを使用してデータベースを強制的にディスクに書き込むことができます。

関連項目:

- 「遅延挿入のためのラージ・プール・バッファ」
- MEMOPTIMIZE FOR WRITEヒントについてさらに学習するには、『Oracle Database SQL言語リファレンス』を参照してください

ロックのメカニズム

一般に、マルチユーザー・データベースでは、なんらかのデータ・ロックを使用してデータの同時実行性、一貫性、および整合性の

問題を解決しています。

ロックは、同じリソースにアクセスする複数のトランザクション間で、破壊的な相互作用が起きないようにするメカニズムです。

関連項目:

「Oracle Databaseのロック・メカニズムの概要」

ANSI/ISOトランザクション分離レベル

ANSIとISO/IECの両方で採用されているSQL規格では、トランザクション分離レベルが4つ定義されています。これらのレベルは、トランザクション処理のスループットに与える影響の度合いが異なります。

これらの分離レベルは、複数のトランザクションを同時に実行する場合に防ぐ必要がある現象という観点から定義されています。回避可能な現象とは、次のものです。

- 内容を保証しない読取り

  未コミットのトランザクションが書き込んだデータを、別のトランザクションが読み取る現象。

- 非リピータブル・リード(ファジー読取り)

  あるトランザクションが以前に読み取ったデータをもう一度読み取ったときに、コミットされた別のトランザクションによってそのデータが変更または削除されたことが明らかになる現象。たとえば、ユーザーがデータに変更があるかどうかを知るためにのみ、ある行を問い合せ、後で再度同じ行を問い合せるという現象などのことです。

- 仮読取り

  あるトランザクションが検索条件を満たす一連の行を戻す問合せを2回実行する間に、コミットされた別のトランザクションによってその条件を満たす新しい行が挿入されたことが明らかになる現象。

  たとえば、トランザクションから従業員数を問い合せるとします。5分後に同じ問合せを実行したところ、他のユーザーが新規従業員のレコードを挿入したため、戻される従業員数が1人増えています。問合せ基準を満たすデータは増えていますが、ファジー読取りとは異なり、すでに読み取ったデータは変更されていません。

SQL規格では、それぞれの分離レベルで実行されるトランザクションで起こり得る現象という観点で4つの分離レベルを定義しています。表9-1に、これらのレベルを示します。

表9-1 分離レベル別による回避可能な読取り現象

分離レベル	内容を保証しない読取り	非リピータブル・リード	仮読取り
非コミット読取り	可能性あり	可能性あり	可能性あり
コミット読取り	可能性なし	可能性あり	可能性あり
リピータブル・リード	可能性なし	可能性なし	可能性あり
シリアライズ可能	可能性なし	可能性なし	可能性なし

Oracle Databaseには、コミット読取り(デフォルト)およびシリアライズ可能の分離レベルが用意されています。また、読取り専

用モードも提供されています。

関連項目:

- コミット読取り、シリアライズ可能および読取り専用の各分離レベルについて学習するには、「Oracle Databaseのトランザクション分離レベルの概要」を参照してください
- Oracle DatabaseのSQL規格への準拠については、『Oracle Database SQL言語リファレンス』を参照してください

Oracle Databaseのトランザクション分離レベルの概要

トランザクション分離レベルのANSI規格は、各分離レベルで許可または防止される現象という観点から定義されています。

Oracle Databaseでは、次のトランザクション分離レベルが提供されています。

- コミット読取り分離レベル
- シリアライズ可能分離レベル
- 読取り専用分離レベル

関連項目:

- トランザクション分離レベルについてさらに学習するには、Oracle Database開発ガイドを参照してください。
- SET TRANSACTION ISOLATION LEVELについて学習するには、Oracle Database SQL言語リファレンスおよびOracle Database PL/SQL言語リファレンスを参照してください。

コミット読取り分離レベル

コミット読取り分離レベルで、トランザクションによって実行される問合せにおいて、トランザクション開始前ではなく、問合せ開始前にコミット済のデータのみが表示されます。

この分離レベルがデフォルトになります。これはトランザクションの競合がほとんど発生しないデータベース環境に適しています。

コミット読取りトランザクション内の問合せでは、問合せ処理中にコミットされるデータは読み取られません。たとえば、100万行を持つ表の問合せ中に、別のトランザクションによって行950,000への更新がコミットされたとすると、この問合せで行950,000を読み取っても、その変更は反映されていません。ただし、問合せによって読み取られたデータを他のトランザクションが変更することは可能であるため、複数の問合せ実行の間にデータが変更される可能性があります。このため、同じ問合せを2回実行するトランザクションでは、ファジー読取りと仮読取りが発生する可能性があります。

コミット読取り分離レベルでの読取り一貫性

データベースでは、ユーザーが何もしなくても、あらゆる問合せに対して一貫した結果セットが保証され、データ整合性が保証されます。

UPDATE文のWHERE句などで使用される暗黙的問合せでは、一貫した結果セットが保証されます。ただし、暗黙的問合せ内の各文では、そのDML文自体によって行われた変更は参照されず、変更前に存在したデータが参照されます。

SELECT文のリストにPL/SQLファンクションが含まれる場合、親SQLレベルではなくPL/SQLファンクション・コード内で実行される

SQL文のレベルにおいて、文レベルの読取り一貫性が適用されます。たとえば、ファンクションは、別のユーザーがデータを変更およびコミットした表にアクセスする場合があります。ファンクション内のSELECTの実行ごとに、読取り一貫性を備えた新しいスナップショットが設定されます。

関連項目:

「副問合せ」

コミット読取りトランザクション内での書込みの競合

コミット読取りトランザクションでは、未コミットの同時実行トランザクションによって更新された行の変更を試行すると、**書込みの競合**が発生します。

行の変更を防止するトランザクションは、**ブロックしているトランザクション**と呼ばれる場合があります。コミット読取りトランザクションは、ブロックしているトランザクションが終了して行ロックが解除されるまで待機します。

次の2通りの場合があります。

- ブロックしているトランザクションがロールバックされた場合、待機しているトランザクションでは、ブロックしているトランザクションが存在していなかったかのように、ロックされていた行が変更されます。
- ブロックしているトランザクションがコミットされてロックが解除された場合、待機しているトランザクションでは、新たに変更された行に対して更新が実行されます。

次の表に、トランザクション1 (シリアライズ可能トランザクションまたはコミット読取りトランザクション)とコミット読取りトランザクション2との間の相互作用を示します。これは、更新内容の消失と呼ばれる典型的な状況を示しています。トランザクション1の更新内容は、コミットされましたが表に反映されていません。このような更新内容の消失の問題をいかに回避するかは、アプリケーション開発の重要な点です。

表9-2 READ COMMITTEDトランザクションにおける書込みの競合と更新内容の消失

セッション1	セッション2	説明
SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda', 'Greene', 'Hintz'); LAST_NAME SALARY ------------- ------ Banda 6200 Greene 9500	アクションなし。	セッション 1 で、Banda、Greene および Hintz の給与を問い合せます。Hintz という名前の従業員が見つかりません。
SQL> UPDATE employees SET salary = 7000 WHERE last_name = 'Banda';	アクションなし。	セッション 1 で、Banda の給与を更新することによってトランザクションを開始します。トランザクション 1 のデフォルトの分離レベルは、READ COMMITTED です。
アクションなし。	SQL> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;	セッション 2 でトランザクション 2 を開始して、分離レベルを明示的に READ COMMITTED に設定

セッション1	セッション2	説明
		します。
アクションなし。	SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda', 'Greene', 'Hintz'); LAST_NAME SALARY ------------- ------ Banda 6200 Greene 9500	トランザクション 2 で、Banda、Greene および Hintz の給与を問い合せます。Oracle Database では読取り一貫性が使用され、トランザクション 1 によって行われた未コミットの更新前の Banda の給与を表示します。
アクションなし。	SQL> UPDATE employees SET salary = 9900 WHERE last_name='Greene';	トランザクション 1 では Banda の行のみがロックされているため、トランザクション 2 で Greene の給与が正常に更新されます。
SQL> INSERT INTO employees (employee_id, last_name, email, hire_date, job_id) VALUES (210, 'Hintz', 'JHINTZ', SYSDATE, 'SH_CLERK');	アクションなし。	トランザクション 1 で、従業員 Hintz の行を挿入しますが、コミットはしません。
アクションなし。	SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda', 'Greene', 'Hintz'); LAST_NAME SALARY ------------- ------ Banda 6200 Greene 9900	トランザクション 2 で、従業員 Banda、Greene および Hintz の給与を問い合せます。 トランザクション 2 では、このトランザクション自体で更新した Greene の給与を参照できます。トランザクション 2 では、トランザクション 1 によって行われた未コミットの Banda の給与の更新や Hintz の挿入は参照できません。
アクションなし。	SQL> UPDATE employees SET salary = 6300 WHERE last_name = 'Banda'; -- prompt does not return	トランザクション 2 で、現在トランザクション 1 によってロックされている Banda の行の更新を試行したため、書込みの競合が発生します。トランザクション 2 は、トランザクション 1 の終了

セッション1	セッション2	説明
		を待機します。
SQL> COMMIT;	アクションなし。	トランザクション 1 では作業をコミットし、トランザクションを終了します。
アクションなし。	1 row updated. SQL>	Banda の行に対するロックが解除されたため、トランザクション 2 では Banda の給与が更新されます。
アクションなし。	SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda', 'Greene', 'Hintz'); LAST_NAME SALARY ------------- ------ Banda 6300 Greene 9900 Hintz	トランザクション 2 で、従業員 Banda、Greene および Hintz の給与を問い合せます。トランザクション 1 によってコミットされた Hintz の行の挿入がトランザクション 2 でも参照可能になりました。トランザクション 2 では、このトランザクション自体で行った Banda の給与への更新が表示されます。
アクションなし。	COMMIT;	トランザクション 2 では作業をコミットし、トランザクションを終了します。
SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda', 'Greene', 'Hintz'); LAST_NAME SALARY ------------- ------ Banda 6300 Greene 9900 Hintz	アクションなし。	セッション 1 で、Banda、Greene および Hintz の行を問い合せます。Banda の給与は 6300 ですが、これはトランザクション 2 で更新された値です。トランザクション 1 で Banda の給与を 7000 に更新しましたが、この更新内容は消失しました。

関連項目:

- 更新内容の消失について学習するには、「ロックの使用」を参照
- データベースが行ロックを取得するタイミングと理由を学習するには、「行ロック(TX)」

シリアライズ可能分離レベル

シリアライズ可能分離レベルでは、問合せ開始時ではなく、トランザクション開始時にコミット済の変更およびこのトランザクション自体が行った変更のみを参照できます。

シリアライズ可能トランザクションは、他のユーザーがデータベースのデータを変更していないように認識される環境内で動作します。シリアライズ可能な分離は、次のような環境に適しています。

- 大規模データベースを使用し、短いトランザクションでごく少数の行しか更新しない環境。
- 2つの同時実行トランザクションが同一の行を更新するようなことが比較的少ない環境。
- 比較的長時間にわたって実行されるトランザクションが主に読取り専用である環境。

シリアライズ可能分離では、通常は文レベルで確保される読取り一貫性が、トランザクション全体に拡張されます。トランザクションによって読み取られるすべての行は、再度読み取っても同じであることが保証されます。トランザクション実行中は、問合せによって戻される結果が同じであることが保証されるため、トランザクションの実行時間に関係なく、他のトランザクションによる変更は問合せで参照できません。シリアライズ可能トランザクションでは、内容を保証しない読取り、ファジー読取りまたは仮読取りの現象は発生しません。

Oracle Databaseの場合、シリアライズ可能トランザクションでは、他のトランザクションで変更され、シリアライズ可能トランザクションが開始されたときにすでにコミットされている行のみ変更できます。シリアライズ可能トランザクションが、その開始後にコミットされた別のトランザクションによって変更されたデータの更新または削除を試行すると、データベースでは次のようなエラーが生成されます。

```
ORA-08177: Cannot serialize access for this transaction
```

シリアライズ可能トランザクションでORA-08177エラーが発生した場合、アプリケーションでは次のいずれかのアクションを実行できます。

- 実行した作業をそのポイントまでコミットします。
- 追加の(異なる)文を実行します(トランザクション内で以前に設定したセーブポイントまでロールバックした後など)。
- トランザクション全体をロールバックします。

次の表に、シリアライズ可能トランザクションと他のトランザクションとの間の相互作用を示します。シリアライズ可能トランザクションにおいては、シリアライズ可能トランザクションの開始後に他のトランザクションでコミットされた行の変更を試行しなければ、シリアライズ・アクセスの問題を回避できます。

表9-3 シリアライズ可能トランザクション

セッション1	セッション2	説明
SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda', 'Greene', 'Hintz'); LAST_NAME SALARY ------------- ---------- Banda 6200 Greene 9500	アクションなし。	セッション 1 で、Banda、Greene および Hintz の給与を問い合せます。Hintz という名前の従業員が見つかりません。
SQL> UPDATE employees SET salary = 7000 WHERE last_name='Banda';	アクションなし。	セッション 1 で、Banda の給与を更新することによってトランザクション 1 を開始します。デフォルトの分離レベルは READ

セッション1	セッション2	説明
		COMMITTED です。
アクションなし。	SQL> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;	セッション 2 でトランザクション 2 を開始して、SERIALIZABLE 分離レベルに設定します。
アクションなし。	SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda', 'Greene', 'Hintz'); LAST_NAME SALARY ------------- ------ ----- Banda 6200 Greene 9500	トランザクション 2 で、Banda、Greene および Hintz の給与を問い合せます。Oracle Database では読取り一貫性が使用され、トランザクション 1 によって行われた未コミットの更新前の Banda の給与を表示します。
アクションなし。	SQL> UPDATE employees SET salary = 9900 WHERE last_name = 'Greene';	Banda の行のみがロックされているため、トランザクション 2 で Greene の給与は正常に更新されます。
SQL> INSERT INTO employees (employee_id, last_name, email, hire_date, job_id) VALUES (210, 'Hintz', 'JHINTZ', SYSDATE,'SH_CLERK');	アクションなし。	トランザクション 1 で、従業員 Hintz の行を挿入します。
SQL> COMMIT;	アクションなし。	トランザクション 1 では作業をコミットし、トランザクションを終了します。
SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda', 'Greene', 'Hintz'); LAST_NAME SALARY ------------- ------ ------ Banda 7000 Greene 9500 Hintz	SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda', 'Greene', 'Hintz'); LAST_NAME SALARY ------------- ------ ----- Banda 6200 Greene 9900	セッション 1 で従業員 Banda、Greene および Hintz の給与を問い合せると、トランザクション 1 によってコミットされた変更が表示されます。セッション 1 では、トランザクション 2 での未コミットの Greene の更新は参照できません。 トランザクション 2 で、従業員 Banda、Greene および Hintz の給与を問い合せます。Oracle Database の読取り一貫性により、トランザク

セッション1	セッション2	説明
		ション 1 でコミットされた Hintz の行の挿入および Banda の更新は、トランザクション 2 では表示されません。トランザクション 2 では、このトランザクション自体で行った Greene の給与への更新が表示されます。
アクションなし。	COMMIT;	トランザクション 2 では作業をコミットし、トランザクションを終了します。
SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda', 'Greene', 'Hintz'); LAST_NAME SALARY ------------- ---------- Banda 7000 Greene 9900 Hintz	SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda', 'Greene', 'Hintz'); LAST_NAME SALARY ------------- ---------- Banda 7000 Greene 9900 Hintz	両方のセッションで、Banda、Greene および Hintz の給与を問い合せます。両方のセッションで、トランザクション 1 およびトランザクション 2 で行われたコミット済のすべての変更を参照できます。
SQL> UPDATE employees SET salary = 7100 WHERE last_name = 'Hintz';	アクションなし。	セッション 1 で、Hintz の給与を更新することによってトランザクション 3 を開始します。トランザクション 3 のデフォルトの分離レベルは、READ COMMITTED です。
アクションなし。	SQL> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;	セッション 2 でトランザクション 4 を開始して、SERIALIZABLE 分離レベルに設定します。
アクションなし。	SQL> UPDATE employees SET salary = 7200 WHERE last_name = 'Hintz'; -- prompt does not return	トランザクション 4 で、Hintz の給与の更新を試みますが、トランザクション 3 で Hintz の行がロックされているためにブロックされます。トランザクション 4 は、トランザクション 3 の後のキューに入ります。
SQL> COMMIT;	アクションなし。	トランザクション 3 で Hintz の給与の更新をコミットして、トラ

セッション1	セッション2	説明
		ンザクションを終了します。
アクションなし。	UPDATE employees SET salary = 7200 WHERE last_name = 'Hintz' * ERROR at line 1: ORA-08177: can't serialize access for this transaction	コミットしてトランザクション 3 を終了することによって、トランザクション 4 における Hintz の更新は ORA-08177 エラーが発生して失敗します。この問題は、トランザクション 4 が開始された後にトランザクション 3 で Hintz の更新をコミットしたことによって発生します。
アクションなし。	SQL> ROLLBACK;	セッション 2 で、トランザクション 4 をロールバックして終了します。
アクションなし。	SQL> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;	セッション 2 でトランザクション 5 を開始して、SERIALIZABLE 分離レベルに設定します。
アクションなし。	SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda', 'Greene', 'Hintz'); LAST_NAME SALARY ---------- ---------- Banda 7000 Greene 9900 Hintz 7100	トランザクション 5 で、Banda、Greene および Hintz の給与を問い合せます。トランザクション 3 によってコミットされた Hintz の給与の更新を参照できます。
アクションなし。	SQL> UPDATE employees SET salary = 7200 WHERE last_name='Hintz'; 1 row updated.	トランザクション 5 で、Hintz の給与を異なる値に更新します。トランザクション 5 が開始される前にトランザクション 3 での Hintz の更新をコミットしているため、シリアライズ・アクセスの問題は回避されます。 **ノート:** トランザクション 5 が開始された後に、別のトランザクションで Hintz の行を更新してコミットした場合は、再度シリアライズ・アクセスの問題が発生します。

セッション1	セッション2	説明
アクションなし。	SQL> COMMIT;	セッション 2 で、問題なく更新をコミットして、トランザクションを終了します。

関連項目:

- 行ロック(TX)
- 「トランザクション制御の概要」

読取り専用分離レベル

読取り専用分離レベルはシリアライズ可能分離レベルと類似していますが、読取り専用トランザクションでは、SYSユーザー以外はトランザクションでデータを変更できません。

読取り専用トランザクションでは、ORA-08177エラーは発生しません。読取り専用トランザクションは、トランザクション開始時点での一貫性を保つ必要があるレポートの生成に役立ちます。

Oracle Databaseでは、必要に応じてUNDOセグメントからデータを再構築することによって、読取り一貫性を実現しています。UNDOセグメントは循環方式で使用されるため、データベースではUNDOデータを上書きできます。長時間実行されるレポートでは、読取り一貫性を保つために必要なUNDOデータが他のトランザクションによって再利用され、snapshot too oldというエラーが発生することがあります。UNDO保存期間とは、古いUNDOデータを上書きしないで保存する最低期間のことで、この期間を設定することにより、この問題を適切に回避できます。

関連項目:

- UNDOセグメント
- UNDO保存期間の設定方法を学習するには、『Oracle Database管理者ガイド』を参照してください

Oracle Databaseのロック・メカニズムの概要

ロックは破壊的な相互作用が起きないようにするメカニズムです。

共有データにアクセスする複数のトランザクション間で不正確なデータの更新または基礎となるデータ構造の不正な変更を引き起こす場合に、相互作用は破壊的になります。ロックは、データベースの同時実行性および一貫性を維持するために非常に重要です。

ロック動作の概要

データベースでは、ロックを取得する操作に応じて、複数のタイプのロックが使用されます。

一般に、データベースでは、排他ロックと共有ロックの2種類のロックが使用されます。排他ロックは、1つのリソース(行や表など)に対して1つのみ取得でき、共有ロックは、1つのリソースに対して複数取得できます。

ロックは、読取り側と書込み側の相互作用に影響を与えます。読取り側とはリソースの問合せを指し、書込み側とはリソースを

変更する文を指します。次に、Oracle Databaseにおける、読取り側と書込み側に対するロック動作の概要を示します。

- 行は、書込み側によって変更される場合にのみロックされます。

  文がある行を更新する場合、そのトランザクションはその行に対してのみロックを取得します。行レベルで表データをロックすることによって、同一のデータに対する競合が最小限に抑えられます。通常(脚注1)、行ロックがブロック・レベルや表レベルに拡大されることはありません。

- 行の書込み側は、同じ行に対する同時書込みをブロックします。

  あるトランザクションで行を変更している場合は、行ロックによって、他のトランザクションからの同じ行に対する同時変更が防止されます。

- 読取り側が書込み側をブロックすることはありません。

  行の読取り側は行をロックしないため、書込み側はこの行を変更できます。唯一の例外はSELECT ... FOR UPDATE文で、この文はSELECT文の特殊なタイプであり、読取り中の行をロックします。

- 書込み側が読取り側をブロックすることはありません。

  行が書込み側によって変更されている場合、データベースでは、UNDOデータを使用して読込み側に一貫性のある行のビューが表示されます。

ノート:

保留中の分散トランザクションにおける非常に特殊な状況では、データを読み取るために、同じデータ・ブロックへの書込みの待機が必要になることもあります。

関連項目:

- SELECT ... FOR UPDATEについて学習するには、『Oracle Database SQL言語リファレンス』を参照してください
- インダウト分散トランザクションに関連する待機について学習するには、『Oracle Database管理者ガイド』を参照してください

ロックの使用

シングル・ユーザーのデータベースでは、情報を変更するのは1人のユーザーのみであるため、ロックは必要ありません。ただし、複数のユーザーがデータにアクセスして変更する場合、データベースには、同じデータに対する同時変更を防止する手段が必要です。

ロックによって、次のようなデータベースの重要な要件を満たすことが可能になります。

- 一貫性

  あるセッションが表示または変更しているデータは、そのユーザーがデータの使用を終了するまで他のセッションでは変更できません。

- 整合性

  データおよび構造には、すべての変更内容が適切な順序で反映される必要があります。

Oracle Databaseでは、ロック・メカニズムを使用して、トランザクション間のデータの同時実行性、一貫性および整合性を実現します。ロックは自動的に実行されるため、ユーザーの操作は必要ありません。

ロックの必要性については、単一の行に対する同時更新で説明できます。次の例では、単純なWebベースのアプリケーションによって、エンド・ユーザーに対して従業員の電子メールおよび電話番号が表示されます。アプリケーションでは、次のようなUPDATE文を使用してデータが変更されます。

```
UPDATE employees
SET    email = ?, phone_number = ?
WHERE  employee_id = ?
AND    email = ?
AND    phone_number = ?
```

この例のUPDATE文では、WHERE句中の電子メールと電話番号は、特定の従業員についての変更前の元の値です。この更新では、アプリケーションで変更される行は、最後に読み取られてユーザーに表示された後、変更されていないことが保証されます。このようにして、アプリケーションでは、あるユーザーによって行われた変更を他のユーザーが上書きして更新内容が消失する、更新内容の消失の問題が回避されます(更新内容の消失の例は、表9-2を参照)。

表9-4に、2つのセッションがほぼ同時にemployees表の同じ行の変更を試行した場合の一連のイベントを示します。

表9-4 行ロックの例

T	セッション1	セッション2	説明
t0	SELECT employee_id as ID, email, phone_number FROM hr.employees WHERE last_name='Himuro';ID EMAIL PHONE_NUMBER --- ------- - ---------- 118 GHIMURO 515.127.4565		セッション 1 で、hr1 ユーザーが hr.employees の Himuro のレコードを問い合せて、employee_id(118)、email(GHIMURO)および phone_number(515.127.4565)属性を表示します。
t1		SELECT employee_id as ID, email, phone_number FROM hr.employees WHERE last_name='Himuro';ID EMAIL PHONE_NUMBER --- ------- - ---------- 118 GHIMURO 515.127.4565	セッション 2 で、hr2 ユーザーが hr.employees の Himuro のレコードを問い合せて、employee_id(118)、email(GHIMURO)および phone_number(515.127.4565)属性を表示します。
t2	UPDATE hr.employees SET phone_number='515.555.1234' WHERE employee_id=118 AND email='GHIMURO' AND phone_number = '515.127.4565'; 1 row updated.		セッション 1 で、hr1 ユーザーがこの行の電話番号を 515.555.1234 に更新します。この場合、GHIMURO の行に対して

T	セッション1	セッション2	説明
			ロックを取得します。
t3		UPDATE hr.employees SET phone_number='515.555.1235' WHERE employee_id=118 AND email='GHIMURO' AND phone_number = '515.127.4565'; -- SQL*Plus does not show -- a row updated message or -- return the prompt.	セッション 2 で、hr2 ユーザーが同じ行を更新しようとしますが、現在 hr1 がその行を処理中であるためブロックされます。 hr2 による更新は、hr1 による更新とほぼ同時に試行されました。
t4	COMMIT; Commit complete.		セッション 1 で、hr1 ユーザーがトランザクションをコミットします。 このコミットによって Himuro に対する変更が確定され、待機しているセッション 2 のブロックが解除されます。
t5		0 rows updated.	セッション 2 で、hr2 ユーザーは、GHIMURO の行が変更されたため、条件に一致しなくなったことに気付きます。 条件が一致しないため、セッション 2 ではレコードが更新されません。
t6	UPDATE hr.employees SET phone_number='515.555.1235' WHERE employee_id=118 AND email='GHIMURO' AND phone_number='515.555.1234'; 1 row updated.		セッション 1 で、hr1 ユーザーは、GHIMURO の行を誤った電話番号で更新したことに気付きます。そのためユーザーは新たにトランザクションを開始して、この行の電話番号を 515.555.1235 に更新し、これにより GHIMURO の行がロックされます。

T	セッション1	セッション2	説明
t7		SELECT employee_id as ID, email, phone_number FROM hr.employees WHERE last_name='Himuro';ID EMAIL PHONE_NUMBER --- ------- ------------ 118 GHIMURO 515.555.1234	セッション 2 で、hr2 ユーザーは、hr.employees の Himuro のレコードを問い合せます。レコードには、t4 でセッション 1 によってコミットされた電話番号の更新内容が表示されます。Oracle Database の読取り一貫性によって、セッション 2 では t6 の未コミットの変更は表示されません。
t8		UPDATE hr.employees SET phone_number='515.555.1235 WHERE employee_id=118 AND email='GHIMURO' AND phone_number = '515.555.1234'; -- SQL*Plus does not show -- a row updated message or -- return the prompt.	セッション 2 で、hr2 ユーザーが同じ行を更新しようとしますが、現在 hr1 がその行を処理中であるためブロックされます。
t9	ROLLBACK; Rollback complete.		セッション 1 で、hr1 ユーザーがトランザクションをロールバックして終了します。
t10		1 row updated.	セッション 1 による更新がロールバックされたため、セッション 2 による電話番号の更新は成功します。GHIMURO の行が条件に一致するため、更新は成功します。
t11		COMMIT; Commit complete.	セッション 2 で、更新をコミットして、トランザクションを終了します。

Oracle Databaseでは、SQL文を実行するときに、必要なロックを自動的に取得します。たとえば、セッションによるデータの変更が許可される前に、最初にデータをロックする必要があります。ロックによってセッションは更新対象データに対する排他制御を取得するため、ロックが解除されるまでは他のトランザクションはロックされたデータを変更できません。

Oracle Databaseのロック・メカニズムはトランザクション制御と密接に結び付けられているため、アプリケーション設計者がトランザクションを適切に定義するのみで、ロックはOracle Databaseによって自動的に管理されます。ユーザーはリソースを明示的にロックする必要はありませんが、Oracle Databaseには、ユーザーが手動でデータをロックするための方法も用意されていま

す。

次の項では、Oracle Databaseでデータの同時実行性が実現される方法を理解するために重要な概念について説明します。

関連項目:

更新内容の消失の防止に役立つサブプログラムを含むOWA_OPT_LOCKパッケージについて学習するには、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

ロック・モード

Oracle Databaseでは、最高水準のデータ同時実行性を実現しながらフェイルセーフなデータの整合性を確保するため、可能なかぎり低い制限レベルが自動的に使用されます。

制限レベルが低いほど、他のユーザーによるデータへのアクセスがより容易になります。逆に、制限レベルが高いほど、他のトランザクションで取得できるロックのタイプがより制限されます。

Oracle Databaseでは、マルチユーザー・データベースで2つのロック・モードを使用します。

- 排他ロック・モード

  このモードは、関連リソースの共有を防止します。トランザクションでは、データを変更するときに排他ロックを取得します。リソースを排他的にロックした最初のトランザクションは、その排他ロックが解除されるまで、そのリソースを変更できる唯一のトランザクションになります。

- 共有ロック・モード

  このモードでは、操作の種類に応じて、関連するリソースの共有が可能です。データの読取りを実行する複数のユーザーはそのデータを共有でき、またそれぞれが共有ロックを保持することによって、排他ロックを必要とする書込みユーザーの同時アクセスを防ぎます。複数のトランザクションが同じリソースについて共有ロックを取得できます。

あるトランザクションで、SELECT ... FOR UPDATE文を使用して単一の表の行を選択するとします。このトランザクションでは、排他行ロックおよび行共有表ロックを取得します。行ロックでは、他のセッションは、ロックされた行以外の任意の行を変更できますが、表ロックでは、他のセッションによる表の構造の変更が禁止されます。したがって、データベースでは、任意の数の文を実行できます。

ロック変換とロックの段階的拡大

Oracle Databaseでは、必要に応じて**ロック変換**が実行されます。

ロック変換は、低い制限の表ロックを、より高い制限の表ロックに自動的に変換します。たとえば、あるトランザクションで特定の従業員に対するSELECT ... FOR UPDATEを発行し、その後ロックされた行を更新するとします。この場合、データベースでは、行共有表ロックが行排他表ロックに自動的に変換されます。トランザクションは、トランザクション内で挿入、更新または削除の対象になる行すべてに対して排他ロックを保持します。行ロックは、制限の最も高い程度で取得されるため、ロック変換はまったく必要なく、実行されません。

ロック変換は、あるレベル(行レベルなど)で多数のロックが保持されている場合に、各ロックをデータベースが上位レベルの粒度(表レベルなど)の別のロックに拡大するロックの段階的拡大とは異なります。あるセッションが表の中の多数の行をロックした場合、データベースによっては、行ロックが単一の表ロックに自動的に拡大される場合があります。ロックの数は少なくなりますが、ロックされている対象の制限は大きくなります。

*Oracle Database*では、ロックの段階的拡大が発生することはありません。ロックの段階的拡大はデッドロックの可能性を大幅

に増します。たとえば、システムがトランザクション1のためにロックの拡大を試行しますが、トランザクション2によって保持されているロックのために拡大できないとします。トランザクション2の処理を続行するために同じデータのロックの段階的拡大が必要な場合、デッドロックが発生します。

ロックの期間

Oracle Databaseでは、なんらかのイベントが発生し、トランザクションがそのリソースを必要としなくなると、ロックは自動的に解除されます。

通常、トランザクション内の文で取得したロックは、トランザクションの存続期間中保持されます。これらのロックによって、同時トランザクションによる内容を保証しない読取り、更新内容の消失、有害なDDLなどの破損を招く干渉を防止できます。

ノート:

外部キーに索引が作成されていないために子表に対して設定される表ロックは、トランザクション存続期間中ではなく、文の存続期間中のみ保持されます。また、DBMS_LOCK パッケージを使用すると、ユーザー定義のロックを自由に割当ておよび解除でき、トランザクションの境界を越えて保持することもできます。

Oracle Databaseでは、トランザクションをコミットまたはロールバックすると、そのトランザクション内の文で取得したすべてのロックが解除されます。また、Oracle Databaseでは、セーブポイントまでロールバックすると、そのセーブポイントより後で取得されたロックが解除されます。ただし、ロックを解除されて使用可能となったリソースに対してロックを取得できるのは、ロック中だったリソースを待機していなかったトランザクションのみです。ロック中のリソースを待機していたトランザクションは、元のトランザクションが完全にコミットまたはロールバックされるまで待機し続けます。

関連項目:

- トランザクションの待機動作を示す表10-3を参照
- DBMS_LOCKについてさらに学習するには、「ユーザー定義ロックの概要」を参照

ロックとデッドロック

デッドロックとは、2人以上のユーザーが、相手がロックしているデータをお互いに待機している状況を指します。デッドロックが発生すると、トランザクションの処理を継続できなくなる場合があります。

Oracle Databaseでは、デッドロックを自動的に検出し、そのデッドロックに関係する文の一方をロールバックして、競合する一連の行ロックを解放することにより、デッドロックを解決します。また、データベースでは、文レベルのロールバックが実行されたトランザクションに対して対応するメッセージが戻されます。ロールバックされる文は、デッドロックが検出されたトランザクションに属しています。通常、通知されたトランザクションは明示的にロールバックする必要がありますが、待機した後でロールバック文を再試行できます。

表9-5に、デッドロック状態の2つのトランザクションを示します。

表9-5 デッドロック・トランザクション

T	セッション1	セッション2	説明

T	セッション1	セッション2	説明
t0	SQL> UPDATE employees SET salary = salary*1.1 WHERE employee_id = 100; 1 row updated.	SQL> UPDATE employees SET salary = salary*1.1 WHERE employee_id = 200; 1 row updated.	セッション 1 でトランザクション 1 を開始して、従業員 100 の給与を更新します。セッション 2 でトランザクション 2 を開始して、従業員 200 の給与を更新します。各トランザクションでは、そのトランザクションで更新する行のみがロックされるため、問題は発生しません。
t1	SQL> UPDATE employees SET salary = salary*1.1 WHERE employee_id = 200; -- prompt does not return	SQL> UPDATE employees SET salary = salary*1.1 WHERE employee_id = 100; -- prompt does not return	トランザクション 1 で、現在トランザクション 2 でロックされている従業員 200 の行の更新を試行します。トランザクション 2 で、現在トランザクション 1 でロックされている従業員 100 の行の更新を試行します。 どちらのトランザクションでも、処理を続行または終了するために必要なリソースを取得できないため、デッドロックが発生します。各トランザクションがいくら待機しても、競合するロックは保持されたままです。
t2	UPDATE employees * ERROR at line 1: ORA-00060: deadlock detected while waiting for resource SQL>		トランザクション 1 でデッドロックが検出されて、t1 で発行された UPDATE 文がロールバックされます。ただし、t0 で行われた更新はロールバックされません。セッション 1 にプロンプトが戻されます。 **ノート:** デッドロック状態になっているセッションのうちの一方のみが実際にデッドロック・エラーを受信しますが、どちらのセッションもエラーになる可能性があります。
t3	SQL> COMMIT; Commit complete.		セッション 1 で、t0 で行った更新をコミットして、トランザクション 1 を終了します。t1 で正常に行われなかった更新はコミットされません。
t4		1 row updated. SQL>	トランザクション 1 によってブロックされ

T	セッション1	セッション2	説明
			ていた、t1 でのトランザクション 2 による更新が実行されます。プロンプトが戻されます。
t5		SQL> COMMIT; Commit complete.	セッション 2 で、t0 および t1 において行った更新をコミットして、トランザクション 2 を終了します。

デッドロックが最も頻繁に発生するのは、トランザクションがOracle Databaseのデフォルト・ロックを明示的にオーバーライドする場合です。Oracle Databaseではロックの段階的拡大は実行されず、問合せで読取りロックは使用されませんが、行レベルのロック(ページ・レベルのロックではありません)のみは使用されるため、デッドロックはほとんど起こりません。

関連項目:

- 「手動データ・ロックの概要」
- 明示的に表をロックしたときのデッドロックの処理方法を学習するには、『Oracle Database開発ガイド』を参照してください。

自動ロックの概要

Oracle Databaseのトランザクションでは、同じリソースに対する排他的アクセスを必要とする処理が他のトランザクションで行われないように、自動的にリソースがロックされます。

データベースは、リソースおよび実行される操作に応じて、様々な制限レベルの様々なタイプのロックを自動的に取得します。

ノート:

単純な読取り時に行がロックされることはありません。

Oracle Databaseのロックは、次の表に示すカテゴリに分類されます。

表9-6 ロックのカテゴリ

ロック	説明	さらに学習するには
DML ロック	データを保護します。たとえば、表ロックは表全体をロックし、行ロックは選択された行をロックします。	「DML ロック」
DDL ロック	スキーマ・オブジェクトの構造、たとえば表とビューのディレクトリ定義を保護します。	「DDL ロック」

ロック	説明	さらに学習するには
システム・ロック	データファイルなどの内部データベース構造を保護します。ラッチ、mutex および内部ロックは完全に自動的です。	「システム・ロック」

DMLロック

DMLロックは、データ・ロックとも呼ばれ、複数のユーザーが同時にアクセスするデータの整合性を保証します。

たとえば、DMLロックでは、オンライン書店で最後の1冊を2人の顧客が購入可能になることを防止できます。DMLロックは、同時に実行される矛盾する複数のDML操作またはDDL操作の破損を招く干渉を防ぎます。

DML文では、次のタイプのロックを自動的に取得します。

- 行ロック(TX)
- 表ロック(TM)

後続の項において、それぞれのタイプのロックまたはロック・モードの後のカッコ内にある頭字語は、Oracle Enterprise Manager(Enterprise Manager)のロック・モニターで使用される略称です。Enterprise Managerでは、表ロックのモード(RSやSRXなど)が示されるかわりに、すべての表ロックにTMと表示される可能性があります。

関連項目:

「Oracle Enterprise Manager」

行ロック(TX)

行ロックは、*TX*ロックとも呼ばれ、表の単一の行に対するロックです。トランザクションでは、INSERT、UPDATE、DELETE、MERGEまたはSELECT . . . FOR UPDATE文によって変更される各行で行ロックを取得します。行ロックは、トランザクションがコミットされるかロールバックされるまで保持されます。

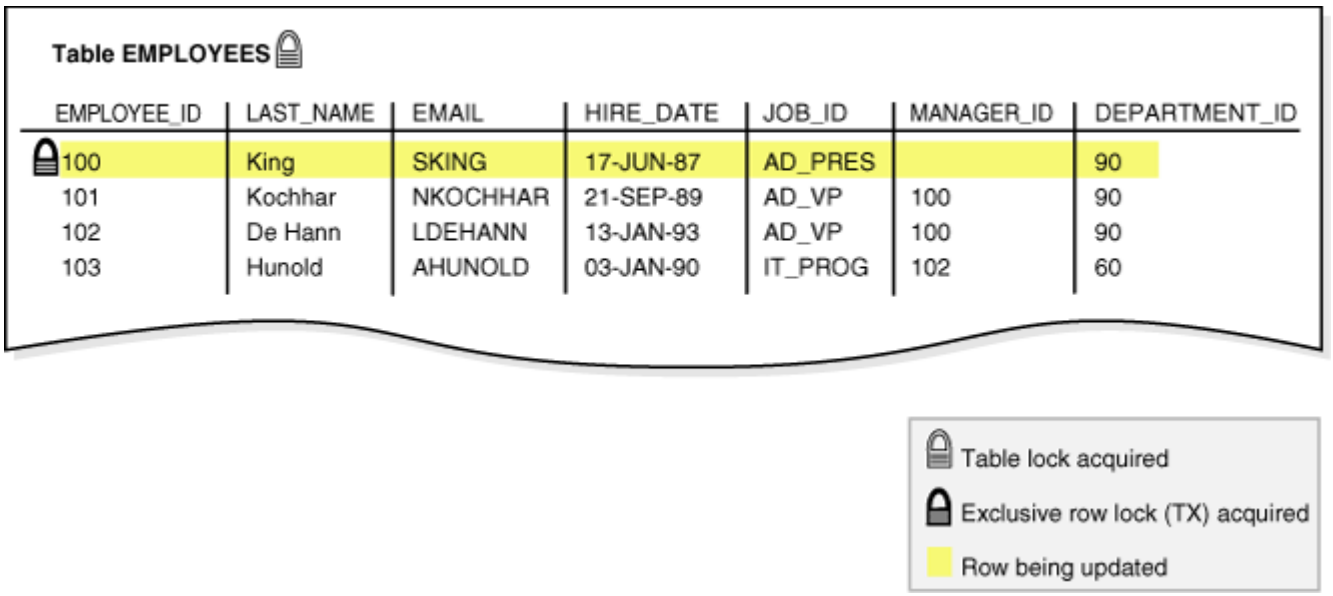
行ロックは、主に2つのトランザクションによる同じ行の変更を防止するためのキューイング・メカニズムとして機能します。変更される行は常に排他モードでロックされるため、ロックを保持しているトランザクションがコミットまたはロールバックされるまで、他のトランザクションはその行を変更できません。行ロックでは、最もきめの細かいロックが実現されるため、最高の同時実行性とスループットが得られます。

ノート:

データベースのインスタンス障害が原因でトランザクションが終了した場合は、トランザクション全体がリカバリされる前に、ブロックレベルのリカバリによって行が使用可能になります。

ある行のロックを取得したトランザクションは、その行を含む表のロックも取得します。表ロックがあると、現行のトランザクションでのデータ変更をオーバーライドするDDL操作の競合が回避されます。図9-2に、表の行3を更新する例を示します。Oracle Databaseでは、更新される行には排他ロックが、表には副排他ロックが自動的に設定されます。

図9-2 行ロックと表ロック



行ロックと同時実行性

このシナリオは、Oracle Databaseが同時実行性のために行ロックを使用する方法を示しています。

3つのセッションから同時に同じ行を問い合せます。セッション1およびセッション2では異なる行に対してコミットせずに更新しますが、セッション3では更新は行われません。各セッションでは、そのセッション自体で行った未コミットの更新を参照できますが、他のセッションによる未コミットの更新は参照できません。

表9-7 データ同時実行性の例

T	セッション1	セッション2	セッション3	説明
t0	SELECT employee_id, salary FROM employees WHERE employee_id IN (100, 101);EMPLOYEE_ID SALARY ---------- ------ 100 512 101 600	SELECT employee_id, salary FROM employees WHERE employee_id IN (100, 101);EMPLOYEE_ID SALARY ---------- ------ 100 512 101 600	SELECT employee_id, salary FROM employees WHERE employee_id IN (100, 101);EMPLOYEE_ID SALARY ---------- ------ 100 512 101 600	3 つの異なるセッションから、従業員 100 および 101 の ID と給与を同時に問い合せます。それぞれの問合せからは、同じ結果が戻されます。
t1	UPDATE hr.employees SET salary = salary+100 WHERE employee_id=100;			セッション 1 で従業員 100 の給与を更新しますが、コミットはしません。更新時には、書込み側は更新する行に対してのみ行レベル・ロックを取得して、他の書込み側からこの行を変更されないようにします。
t	SELECT employee_id, salary FROM employees	SELECT employee_id, salary FROM employees	SELECT employee_id, salary FROM employees	各セッションが、同時

T	セッション1	セッション2	セッション3	説明
2	WHERE employee_id IN (100, 101);EMPLOYEE_ID SALARY ----------- ------ 100 **612** 101 600	WHERE employee_id IN (100, 101);EMPLOYEE_ID SALARY ----------- ------ 100 512 101 600	WHERE employee_id IN (100, 101);EMPLOYEE_ID SALARY ----------- ------ 100 512 101 600	に元の問合せを発行します。セッション 1 では、t1 での更新結果が反映された給与 612 が表示されます。セッション 2 およびセッション 3 における読取りでは、セッション 1 のトランザクション終了を待機せず、即座に行が戻されます。データベースでは、セッション 1 での更新前の状態の給与を表示するためにマルチバージョン読取り一貫性が使用されます。
t 3		UPDATE hr.employee SET salary = salary+100 WHERE employee_id=101;		セッション 2 で従業員 101 の給与を更新しますが、トランザクションをコミットしません。更新時には、書込み側は更新する行に対してのみ行レベル・ロックを取得して、他のセッションからこの行が変更されないようにします。
t 4	SELECT employee_id, salary FROM employees WHERE employee_id IN (100, 101);EMPLOYEE_ID SALARY ----------- ------ 100 **612** 101 600	SELECT employee_id, salary FROM employees WHERE employee_id IN (100, 101);EMPLOYEE_ID SALARY ----------- ------ 100 512 101 **700**	SELECT employee_id, salary FROM employees WHERE employee_id IN (100, 101);EMPLOYEE_ID SALARY ----------- ------ 100 512 101 600	各セッションが、同時に元の問合せを発行します。セッション 1 では、t1 で更新された給与 612 が表示されますが、セッション 2 で更新された従業員 101 の給与は表示されません。セッション 2 の読取り側では、セッショ

T	セッション1	セッション2	セッション3	説明
				ン 2 によって更新された給与が表示されますが、セッション 1 で更新された給与は表示されません。セッション 3 での読取りでは、読取り一貫性を使用して、セッション 1 およびセッション 2 で変更される前の給与が表示されます。

関連項目:

- Oracle Database SQL言語リファレンス
- V$LOCKについて学習するには、『Oracle Databaseリファレンス』を参照してください。

行ロックの記憶域

ロック・マネージャを使用してロックのリストをメモリー上に保持する一部のデータベースとは異なり、Oracle Databaseでは、ロックされた行を含むデータ・ブロックにロック情報を格納します。

Oracle Databaseでは、キューイング・メカニズムを使用して、行ロックを取得します。トランザクションで未ロックの行に対するロックが必要になった場合は、データ・ブロックにロックが配置されます。このトランザクションで変更されるそれぞれの行は、ブロック・ヘッダーに格納されたトランザクションIDのコピーをポイントします。

トランザクションが終了しても、トランザクションIDはそのままブロック・ヘッダーに残ります。別のトランザクションで行が変更される場合は、そのトランザクションIDを使用してロックがアクティブであるかどうかが判断されます。ロックがアクティブである場合は、ロックが解除されたときにセッションに通知されるように設定されます。ロックがアクティブでない場合は、そのトランザクションがロックを取得します。

関連項目:

- データ・ブロック・ヘッダーについてさらに学習するには、「データ・ブロックの概要」を参照
- V$TRANSACTIONについて学習するには、『Oracle Databaseリファレンス』を参照してください

表ロック(TM)

表ロックは*TM*ロックとも呼ばれ、INSERT、UPDATE、DELETE、MERGE、FOR UPDATE句付きのSELECTまたはLOCK TABLE文で表を変更する場合にトランザクションに取得されます。

DML操作では、トランザクションのために表へのDMLアクセスを確保して、このトランザクションと競合する可能性のあるDDL操

作を防止するために表ロックが必要になります。

表ロックは、次のいずれかのモードで保持できます。

- 行共有(RS)

  このロックは、副共有表ロック*(SS)*とも呼ばれ、表に対するロックを保持しているトランザクションがその表の行をロック済であり、それらの行を更新する予定であることを示しています。行共有ロックは、最も制限の緩やかなモードの表ロックであり、最高水準の同時実行性を表にもたらします。

- 行排他表ロック(RX)

  このロックは、副排他表ロック*(SX)*とも呼ばれ、一般的にこのロックを保持しているトランザクションが表の行を更新したか、SELECT ... FOR UPDATEを発行したことを示しています。SXロックでは、他のトランザクションが同じ表の行に対して問合せ、挿入、更新、削除またはロックを実行できます。そのため、SXロックでは、複数のトランザクションが同じ表に対して同時にSXロックおよび副共有表ロックを取得できます。

- 共有表ロック(S)

  トランザクションで共有表ロックを取得すると、他のトランザクションは表に対するSELECT ... FOR UPDATEを使用しない問合せを実行できますが、更新は単一のトランザクションが共有表ロックを保持している場合にのみ許可されます。複数のトランザクションが共有表ロックを同時に保持することがあるため、トランザクションで表を変更できるかどうかは、このロックを保持しているのみでは保証されません。

- 共有行排他表ロック(SRX)

  このロックは、共有副排他表ロック*(SSX)*とも呼ばれ、共有表ロックよりも多くの制限を課します。特定の表に対して、同時に1つのトランザクションのみがSSXロックを取得できます。トランザクションでSSXロックを取得すると、他のトランザクションは表に対してSELECT ... FOR UPDATE以外の問合せを実行できますが、表は更新できません。

- 排他表ロック(X)

  このロックは、最も制限が強くなります。他のトランザクションは、一切のDML文を実行できなくなり、表に一切のロックをかけられなくなります。

関連項目:

- Oracle Database SQL言語リファレンス
- 表ロックについてさらに学習するには、Oracle Database開発ガイドを参照してください。

ロックと外部キー

Oracle Databaseでは、親キーとそれに依存する外部キーとの関連において、同時実行性制御が最適化されます。

ロックの動作は、外部キーの列が索引付けされているかどうかに依存します。外部キーが索引付けされていない場合、子表が頻繁にロックされ、デッドロックが発生し、同時実行性が低下する可能性があります。このため外部キーは、ほとんどの場合、索引付けが必要です。唯一の例外は、対応する一意キーまたは主キーの更新や削除が発生しないことが確実な場合です。

ロックと索引のない外部キー

子表の外部キー列に索引が存在せず、セッションによって(行の削除や主キーの属性の変更などで)親表の主キーが変更されるか、親表に行がマージされた場合、データベースは子表に対する完全表ロックを取得します。

次の両方の条件に当てはまる場合、データベースは子表に対する完全表ロックを取得します。

- 子表の外部キー列に索引が存在しない場合。
- セッションで、親表の主キーが変更されるか(行の削除や主キーの属性の変更など)、親表に行がマージされた場合。

ノート:

親表に挿入しても、子表に対する DML を阻止する、ブロックしている表ロックは取得しません。挿入の場合、データベースは子表に対するロックを取得します。これにより、構造変更は阻止されますが、既存の行または新規に追加された行の変更は阻止されません。

hr.departments表が、索引付けされていない外部キーemployees.department_idを含むhr.employeesの親である場合など。次の図に、departments表の部門60の主キー属性を変更するセッションを示します。

図9-3 外部キーの索引がない場合のロック・メカニズム

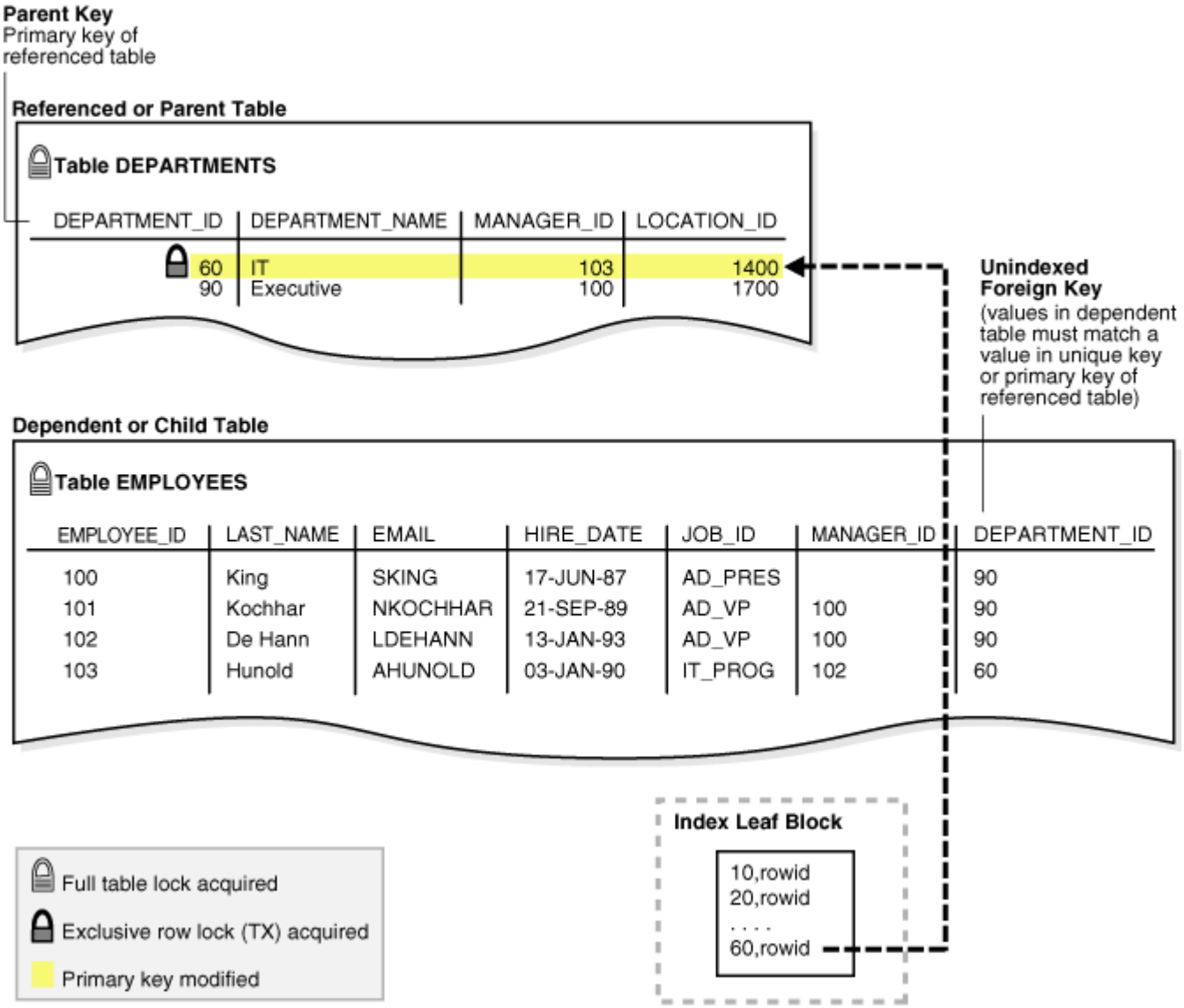


図9-3では、データベースは、部門60の主キーの変更時にemployeesに対する完全表ロックを取得しています。このロックによって、他のセッションはemployees表の問合せはできますが、更新できなくなります。たとえば、セッションでは従業員の電話番号を更新できません。employeesに対する表ロックはdepartments表への主キーの変更が完了すると、即時に解除されます。departmentsの複数の行で主キーの変更が実行される場合、employeesに対する表ロックは、departmentsで変更される行ごとに取得および解除されます。

ノート:

子表に対する DML では、親表の表ロックは取得されません。

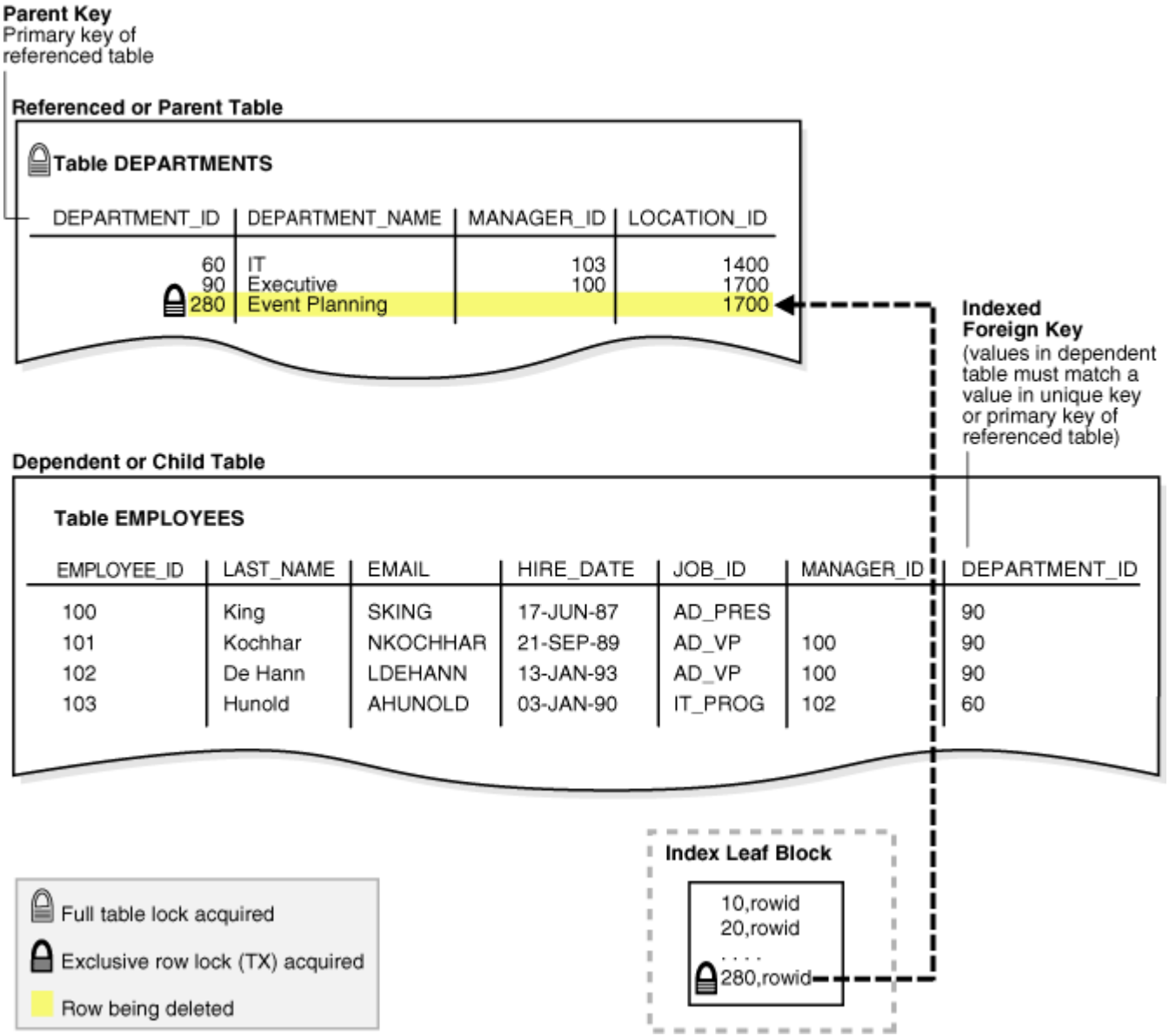
ロックと索引付き外部キー

子表の外部キー列が索引付けされていて、セッションによって(行の削除や主キーの属性の変更などで)親表の主キーが変更されるか、親表に行がマージされた場合、データベースは子表に対する完全表ロックを取得しません。

親表に対するロックにより、トランザクションは排他表ロックを取得できませんが、主キーの変更時に、親表または子表に対するDMLは阻止されません。この状況は、子表への更新が発生している間に、親表での主キーの変更が発生する場合に適しています。

図9-4に、department_id列が索引付けられた子表employeesを示します。トランザクションにより、部門280がdepartmentsから削除されます。この削除により、「ロックと索引のない外部キー」に示すようにデータベースがemployees表に対する完全表ロックを取得することはありません。

図9-4 外部キーが索引付けされている場合のロック・メカニズム



子表でON DELETE CASCADEを指定すると、親表からの削除によって、子表からも削除することができます。たとえば、部門280

を削除した場合、削除された部門の従業員に関するレコードをemployeesから削除できます。この場合、待機とロックに関するルールは、親表から行を削除した後に子表から行を削除する場合と同じです。

関連項目:

- 「外部キー制約」
- 「索引の概要」

DDLロック

データ・ディクショナリ・ロック(DDLロック)によって、DDL操作によるスキーマ・オブジェクトの処理中または参照中に、そのオブジェクトの定義が保護されます。

DDL操作中には、修正や参照の対象となる個々のスキーマ・オブジェクトのみがロックされます。データ・ディクショナリ全体がロックされることはありません。

Oracle Databaseは、DDLロックを必要とするDDLトランザクションのために、DDLロックを自動的に取得します。ユーザーはDDLロックを明示的に要求できません。たとえば、ユーザーがストアド・プロシージャを作成する場合、Oracle Databaseはプロシージャ定義内で参照されるすべてのスキーマ・オブジェクトについてのDDLロックを自動的に取得します。DDLロックにより、プロシージャのコンパイル完了前にそれらのオブジェクトの変更または削除が防止されます。

排他DDLロック

排他DDLロックは、他のセッションによるDDLロックまたはDMLロックの取得を防止します。

ほとんどのDDL操作では、同じスキーマ・オブジェクトを変更または参照する可能性のある他のDDL操作によって破壊的な干渉が起きないように、リソースの排他DDLロックが必要です。たとえば、ALTER TABLEで表に列を追加している間は、DROP TABLE操作でその表を削除できず、その逆も同様です。

排他ロックは、DDL文実行の継続中と自動コミット中ずっと存続します。排他DDLロックの取得では、別の操作によってそのスキーマ・オブジェクトに対して別のDDLロックが保持されている場合、その取得は古いDDLロックが解除されるまで待機し、その後に処理されます。

関連項目:

「共有DDLロック」では、破壊的な干渉を防止するうえで排他ロックが不要である状況について説明します

共有DDLロック

リソースに対する共有DDLロックでは、競合するDDL操作によって破壊的な干渉が防止されますが、同様のDDL操作でのデータ同時実行性は確保されます。

たとえば、CREATE PROCEDURE文の実行時には、その文を含むトランザクションは、参照されるすべての表について共有DDLロックを取得します。他のトランザクションは同じ表を参照するプロシージャを同時に作成できるため、同じ表について同時実行の共有DDLロックの取得はできますが、参照中の表について排他DDLロックを取得できるトランザクションはありません。

共有DDLロックは、DDL文実行の継続中と自動コミット中ずっと存続します。その結果、共有DDLロックを保持するトランザクションでは、参照中のスキーマ・オブジェクトの定義がそのトランザクションの継続時間にわたって変化しないことが保証されます。

ブレーク可能解析ロック

SQL文またはPL/SQLプログラム・ユニットは、参照する各スキーマ・オブジェクトについて**解析ロック**を保持します。

参照オブジェクトが変更または削除された場合に、対応する共有SQL領域を無効にできるようにするため、解析ロックが取得されます。解析ロックは、どのようなDDL操作も拒否せず、一時的に解除して競合するDDL操作を実行することも許可するために、ブレーク可能解析ロックと呼ばれます。

解析ロックは、SQL文実行の解析フェーズ中に共有プールに取得されます。解析ロックは、その文の共有SQL領域が共有プールに残っているかぎり保持されます。

関連項目:

「共有プール」

システム・ロック

Oracle Databaseでは、様々なタイプのシステム・ロックを使用して、内部データベースおよびメモリー構造を保護します。ユーザーは、システム・ロックの出現や継続期間を制御できないため、このメカニズムにアクセスできません。

ラッチ

ラッチは、共有データ構造、オブジェクトおよびファイルへのマルチユーザー・アクセスを調整する、単純な低レベルのシリアライズ化メカニズムです。

ラッチによって、複数のプロセスからアクセスされる共有メモリー・リソースが破損しないように保護されます。特に、ラッチは次の状況からデータ構造を保護します。

- 複数のセッションによる同時変更
- あるセッションで変更中の場合の他のセッションによる読取り
- アクセス中のメモリーの割当て解除(エージ・アウト)

通常、1つのラッチによってSGA内の複数のオブジェクトが保護されます。たとえば、DBWやLGWRなどのバックグラウンド・プロセスでは、共有プールのメモリーを割り当ててデータ構造を作成します。このメモリーを割り当てるために、これらのプロセスでは、アクセスをシリアライズする共有プール・ラッチを使用して、2つのプロセスが共有プールを同時に検査または変更できないようにします。メモリーが割り当てられた後、解析に必要なライブラリ・キャッシュなどの共有プール領域に他のプロセスがアクセスする必要がある場合があります。この場合、共有プール全体ではなくライブラリ・キャッシュのみがラッチされます。

行ロックなどのエンキュー・ラッチとは異なり、ラッチではセッションのキューイングは許可されません。ラッチが使用可能になると、ラッチを要求する最初のセッションがラッチへの排他的アクセスを取得します。プロセスがループ内のラッチを繰り返し要求するとラッチ・スピンが発生し、プロセスがラッチ要求を更新する前にCPUを解放するとラッチ・スリープが発生します。

通常、Oracleプロセスは、データ構造の1つを操作または参照する、非常に短い間のみラッチを取得します。たとえば、1人の従業員の給与更新の処理中に、データベースは数千のラッチを取得および解除する場合があります。ラッチの実装(特に、プロセスがラッチを待機するかどうか、およびどのくらいの時間待機するか)は、オペレーティング・システムに依存します。

ラッチの数を増やすと、同時実行可能な操作の数が減少します。たとえば、ハード解析操作を過度に行うと、ライブラリ・キャッシュ・ラッチの競合が発生します。V$LATCHビューには、各ラッチが要求された回数や待機された回数を含む、各ラッチの詳細な使用統計が含まれています。

関連項目:

- SQLの解析
- V$LATCHについて学習するには、Oracle Databaseリファレンスを参照してください
- 待機イベント統計について学習するには、Oracle Databaseパフォーマンス・チューニング・ガイドを参照してください

mutex

相互排他オブジェクト(mutex)は、メモリー内のオブジェクトが同時プロセスでアクセスされた場合にエージ・アウトまたは破損しないようにするための低レベルのメカニズムです。mutexはラッチに似ていますが、通常、ラッチではオブジェクトのグループが保護されるのに対して、mutexでは単一のオブジェクトが保護されます。

mutexには、複数の利点があります。

- mutexによって、競合の可能性が減少します。

  ラッチでは複数のオブジェクトを保護するため、複数のプロセスで保護対象のオブジェクトのいずれかへの同時アクセスが試行された場合に、ボトルネックが生じる可能性があります。mutexでは、グループではなく個別のオブジェクトへのアクセスをシリアライズするため、可用性が向上します。

- mutexでは、ラッチよりも少ないメモリーが使用されます。
- 共有モードでmutexを使用すると、複数のセッションからの同時参照が可能になります。

内部ロック

内部ロックは、ラッチやmutexよりも高レベルで複雑なメカニズムであり、様々な目的に使用されます。

データベースでは、次のタイプの内部ロックを使用します。

- ディクショナリ・キャッシュ・ロック

  これらのロックの継続時間は非常に短く、ディクショナリ・キャッシュ内のエントリが修正されたり使用されている間、そのエントリについて保持されます。これらのロックは、解析されている文が矛盾したオブジェクト定義を参照しないことを保証します。ディクショナリ・キャッシュ・ロックは、共有または排他で保持されます。共有ロックは解析が完了すると解除されますが、排他ロックはDDL操作が完了したときに解除されます。

- ファイルとログの管理ロック

  これらのロックは様々なファイルを保護します。たとえば、ある内部ロックは制御ファイルを保護し、一度に1つのプロセスのみがそれを変更できるようにします。別のロックは、オンラインREDOログ・ファイルの使用とアーカイブを調整します。データベースを複数インスタンスが共有モードでマウントしたり、1つのインスタンスが排他モードでマウントすることを保証するために、データファイルがロックされます。ファイルとログのロックはファイルの状態を示すため、必然的に長い間保持されます。

- 表領域とUNDOセグメントのロック

  これらのロックは、表領域とUNDOセグメントを保護します。たとえば、データベースにアクセスするすべてのインスタンスは、表領域のオンライン/オフラインの状態について一致している必要があります。UNDOセグメントは、1つのセグメントに1つのデータベース・インスタンスしか書き込めないようにロックされています。

関連項目:

「データ・ディクショナリ・キャッシュ」

手動データ・ロックの概要

Oracle Databaseでは、手動でデフォルトのロック・メカニズムを置き換えることもできます。

データの同時実行性、整合性および文レベルの読取り一貫性を確保するために、Oracle Databaseは自動的にロックを実行します。ただし、デフォルト・ロックの置換えは、次のような状況で有効です。

- アプリケーションで、トランザクション・レベルの読取り一貫性またはリピータブル・リードが必要な場合。

  この場合、問合せによって作成されるデータが、他のトランザクションによる変更を反映せず、トランザクションの継続時間にわたって一貫性を維持する必要があります。トランザクション・レベルの読取り一貫性は、明示的ロック、読取り専用トランザクションまたはシリアライズ可能トランザクションを使用するか、デフォルトのロックをオーバーライドすると実現できます。

- アプリケーションで、あるトランザクションが他のトランザクションの完了まで待機せずに済むように、そのトランザクションがリソースに排他的アクセスできるようにする必要がある場合。

Oracle Databaseの自動ロックは、セッション・レベルまたはトランザクション・レベルでオーバーライドできます。セッション・レベルでは、ALTER SESSION文を使用して必要なトランザクション分離レベルを設定できます。トランザクション・レベルでは、次のSQL文を含むトランザクションによってOracle Databaseのデフォルト・ロックがオーバーライドされます。

- SET TRANSACTION ISOLATION LEVEL文
- LOCK TABLE文(表をロックする文、またはビューを使用するときに実表をロックする文)
- SELECT ... FOR UPDATE文

前述の文で取得したロックは、トランザクション終了後に解除されるか、セーブポイントへのロールバックによって解除されます。

Oracle Databaseのデフォルト・ロックを任意のレベルで置き換える場合、データベース管理者やアプリケーション開発者は、ロック置換の手順が確実に正しく実行されるようにしてください。ロックの手順は、データ整合性が保証される、データ同時実行性が許容範囲内である、デッドロックは発生する可能性がないかまたは適切に処理される、などの基準を満たすものにする必要があります。

関連項目:

- LOCK TABLEおよびSELECTの詳細は、Oracle Database SQL言語リファレンスを参照してください。
- 手動で表をロックする方法を学習するには、Oracle Database開発ガイドを参照してください。

ユーザー定義ロックの概要

Oracle Databaseのロック・マネージメント・サービスを使用すると、特定のアプリケーションで独自のロックを定義できます。

たとえば、ファイルシステム上のメッセージ・ログに対するアクセスをシリアライズするためのロックを作成できます。確保したユーザー・ロックは、Oracle Databaseロックと同一とみなされるため、デッドロックの検出などのOracle Databaseロックの機能をすべて備えています。ユーザー・ロックは接頭辞ULで識別されるため、Oracle Databaseロックと矛盾することはありません。

Oracle Databaseのロック・マネージメント・サービスは、DBMS_LOCKパッケージ内のプロシージャを介して利用できます。

PL/SQLブロックには、次の処理を行う文を含めることができます。

- 特定のタイプのロックを要求します。
- そのロックに、同一のインスタンスまたは別のインスタンス内にある別のプロシージャからも識別できる、一意の名前を指定します。
- ロック・タイプを変更します。
- ロックを解除します

関連項目:

- Oracle Database Lock Managementサービスの詳細は、『Oracle Database開発ガイド』を参照
- DBMS_LOCKの詳細は、『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』を参照

脚注の凡例

脚注1:

分散トランザクションの2フェーズ・コミットを処理する場合、特殊な状況では、データベースで読取りアクセスが一時的に禁止されることがあります。具体的には、問合せが準備フェーズとコミット・フェーズの間に開始され、コミット前にデータの読取りを試みる場合、データベースでは、読取りの整合性を保証するため、行レベルからブロック・レベルにロックが拡大されます。

10 トランザクション

この章では、トランザクションを定義し、データベースでトランザクションが処理される方法について説明します。

この章の構成は、次のとおりです。

- トランザクションの概要
- トランザクション制御の概要
- トランザクション・ガードの概要
- アプリケーション・コンティニュイティの概要
- 自律型トランザクションの概要
- 分散トランザクションの概要

トランザクションの概要

トランザクションは、1つ以上のSQL文を含むアトミックな論理作業単位です。

トランザクションは、SQL文をグループ化して、すべての文がコミットされた(データベースに適用された)状態にするか、すべての文がロールバックされた(データベースから取り消された)状態にできる必要があります。Oracle Databaseでは、すべてのトランザクションにトランザクションIDと呼ばれる一意の識別子が割り当てられます。

すべてのOracleトランザクションは、ACIDプロパティと呼ばれる、データベース・トランザクションの基本プロパティに従います。ACIDは次の単語の頭字語です。

- Atomicity(原子性)

  1つのトランザクションでは、すべてのタスクが実行されるか、1つも実行されないかのいずれかです。部分的に実行されるトランザクションは存在しません。たとえば、トランザクションで100行の更新が開始され、20行が更新された後にシステムで障害が発生した場合、データベースではこの20行に行われた変更がロールバックされます。

- Consistency(一貫性)

  トランザクションにより、データベースは一貫したある状態から一貫した別の状態に移行します。たとえば、普通預金口座の借方に記入し、当座預金口座の貸方に記入する銀行のトランザクションにおいて、障害によって、データベースで一方の口座でのみ貸方に記入されたためにデータの一貫性がなくなることは許されません。

- Isolation(独立性)

  トランザクションの結果は、そのトランザクションがコミットされるまで、他のトランザクションに表示されません。たとえば、hr.employees表を更新しているユーザーには、別のユーザーがemployeesに対して同時に行った未コミットの変更は表示されません。このため、ユーザーには、トランザクションがシリアルに実行されているように見えます。

- Durability(永続性)

  コミットされたトランザクションによって行われた変更は、永続的に確定されます。トランザクションの完了後、データベースではリカバリ・メカニズムを使用して、トランザクションによる変更が失われていないことが確認されます。

トランザクションの使用は、データベース管理システムとファイルシステムの最も重要な違いの1つです。

サンプル・トランザクション: 口座の借方と貸方

トランザクションの概念を具体的に説明するため、銀行業務データベースについて考えてみます。

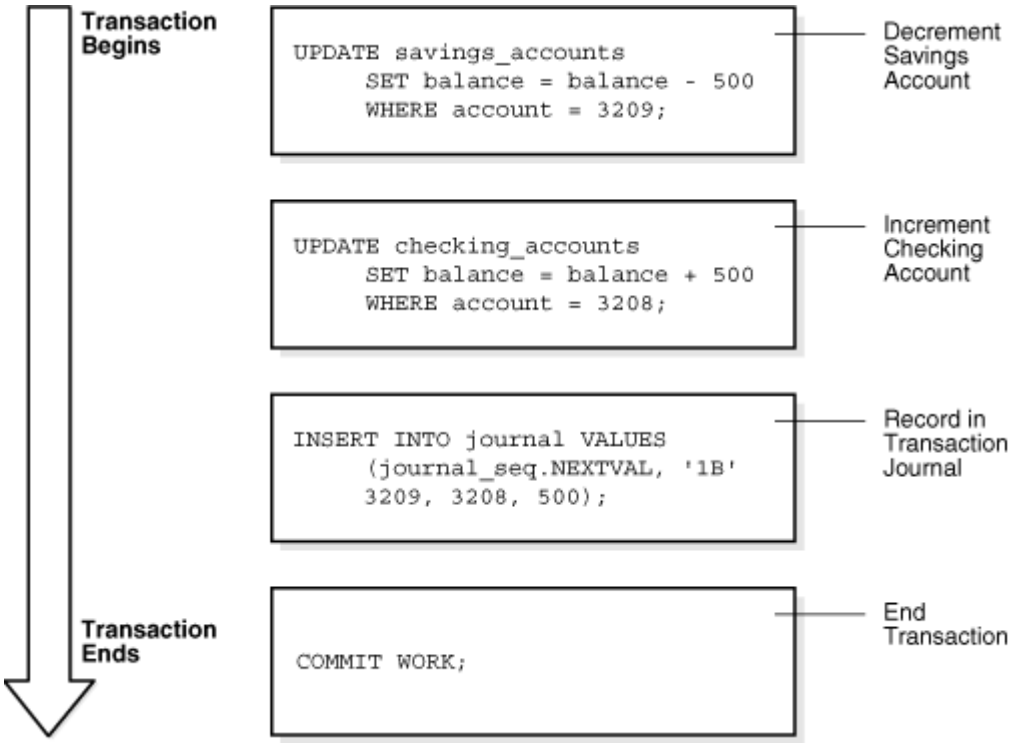
顧客が普通預金口座から当座預金口座へ預金を振り替える場合、トランザクションは次の3つの別々の操作で構成する必要があります。

- 普通預金口座の減額
- 当座預金口座の増額
- トランザクション・ジャーナルへのトランザクションの記録

Oracle Databaseでは、2つの状況が考慮されます。3つのSQL文のすべてにおいて口座の差引残高の帳尻が合えば、トランザクションの結果をデータベースに適用できます。ただし、預金額の不足、口座番号が無効、またはハードウェア障害などの問題が原因で、トランザクション内の1つまたは2つの文が完了しない場合は、データベースではすべての口座の差引残高が正しく維持されるようにトランザクション全体をロールバックする必要があります。

次の図に、銀行のトランザクションを示します。最初の文では、普通預金口座3209から$500を減額します。2番目の文では、当座預金口座3208に$500を増額します。3番目の文では、ジャーナルの表にこの振替のレコードを挿入します。最後の文で、トランザクションをコミットします。

図10-1 銀行のトランザクション



トランザクションの構造

データベース・トランザクションは、1つ以上の文で構成されています。

具体的には、トランザクションは次のいずれかで構成されています。

- 全体でデータベースに対するアトミックな変更を構成する1つ以上のデータ操作言語(DML)文
- 1つのデータ定義言語(DDL)文

トランザクションには、開始と終了があります。

関連項目:

- 「SQL文の概要」
- SQL文のタイプについては、Oracle Database SQL言語リファレンスを参照してください。

トランザクションの開始

トランザクションは、最初の実行可能SQL文が検出された時点で開始されます。

実行可能SQL文は、DML文やDDL文、およびSET TRANSACTION文など、データベース・インスタンスへのコールを生成するSQL文のことです。

トランザクションが開始されると、Oracle Databaseはそのトランザクションを使用可能なUNDOデータ・セグメントに割り当てて、新しいトランザクションのUNDOエントリを記録します。トランザクションIDは、UNDOセグメントおよびトランザクション表スロットが割り当てられるまでは割り当てられません(割当ては、最初のDML文の実行中に行われます)。トランザクションIDはトランザクションに対して一意であり、UNDOセグメント番号、スロットおよび順序番号を表します。

次の例では、UPDATE文を実行してトランザクションを開始し、このトランザクションの詳細をV$TRANSACTIONに問い合せます。

```
SQL> UPDATE hr.employees SET salary=salary;

107 rows updated.

SQL> SELECT XID AS "txn id", XIDUSN AS "undo seg", XIDSLOT AS "slot",
  2  XIDSQN AS "seq", STATUS AS "txn status"
  3  FROM V$TRANSACTION;

txn id             undo seg       slot        seq txn status
---------------- ---------- ---------- ---------- ----------------
0600060037000000          6          6         55 ACTIVE
```

関連項目:

UNDOセグメント

トランザクションの終了

トランザクションは、様々な状況で終了する可能性があります。

トランザクションは、次のいずれかのアクションが発生すると終了します。

- COMMIT文またはROLLBACK文が、SAVEPOINT句なしで発行される場合。

  コミットでは、ユーザーが明示的または暗黙的に、このトランザクションの変更内容を確定するように要求します。トランザクションによって行われた変更内容が確定され、トランザクションのコミット後にのみ他のユーザーが参照できるようになります。

- CREATE、DROP、RENAMEまたはALTERなどのDDLコマンドが実行される場合。

  データベースでは、すべてのDDL文の前後に暗黙的なCOMMIT文が発行されます。現行のトランザクションにDML文が含まれている場合、Oracle Databaseは最初にそのトランザクションをコミットし、新しい単一文トランザクションとしてそのDDL文を実行し、コミットします。

- ユーザーがほとんどのOracle Databaseユーティリティおよびツールを正常に終了する場合(このような場合は、現行のトランザクションが暗黙的にコミットされます)。ユーザーが切断したときのコミット動作は、アプリケーションに依存しており、構成可能です。

ノート:

アプリケーションでは、プログラムを終了する前に、必ず明示的にトランザクションをコミットまたは取り消す必要があります。

- クライアント・プロセスが異常終了し、トランザクション表およびUNDOセグメントに格納されていたメタデータを使用して、トランザクションが暗黙的にロールバックされる場合。

1つのトランザクションが終了すると、次の実行可能SQL文によって次のトランザクションが自動的に開始されます。次の例では、UPDATEを実行してトランザクションを開始し、ROLLBACK文でトランザクションを終了してから、UPDATEを実行して新しいトランザクションを開始します(トランザクションIDが異なることに注意してください)。

```
SQL> UPDATE hr.employees SET salary=salary;
107 rows updated.

SQL> SELECT XID, STATUS FROM V$TRANSACTION;

XID              STATUS
---------------- ----------------
0800090033000000 ACTIVE

SQL> ROLLBACK;

Rollback complete.

SQL> SELECT XID FROM V$TRANSACTION;

no rows selected

SQL> UPDATE hr.employees SET last_name=last_name;

107 rows updated.

SQL> SELECT XID, STATUS FROM V$TRANSACTION;

XID              STATUS
---------------- ----------------
0900050033000000 ACTIVE
```

関連項目:

- コミットで終了するトランザクションの例については、「サンプル・トランザクション: 口座の借方と貸方」を参照してください。
- COMMITについて学習するには、Oracle Database SQL言語リファレンスを参照してください

文レベルの原子性

Oracle Databaseでは、**文レベルの原子性**(SQL文がアトミックな作業単位であり、完全に成功するか、完全に失敗するかのいずれかであること)がサポートされます。

正常に実行される文とコミットされるトランザクションとは異なります。1つのSQL文は、アトミックな単位として解析され、エラーなしで実行された場合、たとえば複数行の更新ですべての行が変更されたときなどに、正常に実行されます。

実行中にSQL文が原因でエラーが発生すると成功しないため、この文のすべての結果がロールバックされます。この操作が文レベルのロールバックです。この操作には次のような特徴があります。

- SQL文が成功しなかった場合は、SQLでの実行予定の作業のみが影響を受けます。

  成功しなかった文が、現行のトランザクションにおけるこれまでの作業に影響を及ぼすことはありません。たとえば、「サンプル・トランザクション: 口座の借方と貸方」で、2番目のUPDATE文でエラーが発生してロールバックされた場合、最初のUPDATE文によって実行された作業はロールバックされません。最初のUPDATE文は、ユーザーが明示的にコミットまたはロールバックできます。

- ロールバックの効果は、その文がまったく実行されなかった場合と同じ状態にすることです。

  アトミックな文の副次的な効果としては、たとえば、文の実行時に起動されたトリガーが、アトミックな文の一部とみなされることがあげられます。アトミックな文の一部として生成された作業は、すべてが成功するか、1つも成功しないかのいずれかです。

文レベル・ロールバックの原因となるエラーの例として、重複する主キーの挿入を試行する場合があります。1つのデッドロック(同じデータに関する競合)に単一SQL文が複数関係している場合も、文レベル・ロールバックが発生します。ただし、SQL文の解析中に構文エラーなどのエラーが検出された場合は、まだSQL文は実行されていないため、文レベルのロールバックは発生しません。

関連項目:

- SQLの解析
- ロックとデッドロック
- トリガーの概要

システム変更番号(SCN)

システム変更番号(SCN)は、Oracle Databaseで使用される論理的な内部タイムスタンプです。

SCNを使用して、トランザクションのACIDプロパティに準拠する必要があるデータベース内で発生するイベントに順序を付けます。Oracle DatabaseではSCNを使用して、すべての変更がディスク上に存在することが認識される前のSCNをマークし、リカバリで不要なREDOが適用されないようにします。さらに、データの集合に対するREDOが存在しない時点のマークにもSCNを使用し、リカバリを停止できるようにします。

SCNは直線的に増加する順序で発生します。監視されたSCNは時間における論理的なある時点を示し、監視を繰り返すと等しい値または前よりも大きな値が戻されるため、Oracle DatabaseではSCNを実時間のように使用できます。あるイベントのSCNが別のイベントのSCNよりも小さい場合、このイベントはそのデータベースで先に発生したことになります。複数のイベントが同じSCNを共有する場合もあり、これはイベントがデータベースで同じ時間に発生したことを意味します。

すべてのトランザクションには、SCNがあります。たとえば、トランザクションで行が更新されると、データベースではこの更新が発生したときのSCNが記録されます。このトランザクションで行われたその他の変更にも同じSCNが付けられます。トランザクションがコミットされると、データベースではこのコミットのSCNが記録されます。

Oracle Databaseでは、SCNがシステム・グローバル領域(SGA)で増分されます。トランザクションでデータが変更されると、データベースではこのトランザクションに割り当てられたUNDOデータ・セグメントに新しいSCNが書き込まれます。次に、ログ・ライター・プロセスによって、このトランザクションのコミット・レコードがオンラインREDOログに即時に書き込まれます。コミット・レコードには、トランザクションの一意のSCNが付けられています。Oracle Databaseでは、SCNをインスタンス・リカバリ・メカニズムおよ

びメディア・リカバリ・メカニズムの一部としても使用します。

関連項目:

- 「インスタンス・リカバリの概要」
- バックアップおよびリカバリ

トランザクション制御の概要

トランザクション制御は、DML文による変更の内容を管理し、一覧のDML文をトランザクションとしてグループ化する処理です。

通常、アプリケーション設計者は、作業が論理単位で完了してデータの一貫性が保たれるよう、トランザクション制御を考慮します。

トランザクション制御では、「トランザクション制御文」で説明したように、次の文が使用されます。

- COMMIT文は、現行のトランザクションを終了し、トランザクションで実行されたすべての変更を確定します。さらに、COMMIT文により、トランザクション内のすべてのセーブポイントが消去され、トランザクション・ロックが解放されます。
- ROLLBACK文は、現行のトランザクションで行われた作業を元に戻し、最後のCOMMITまたはROLLBACK以降に行われたすべてのデータ変更が破棄されます。ROLLBACK TO SAVEPOINT文は、最後のセーブポイント以降の変更を取り消しますが、トランザクション全体は終了しません。
- SAVEPOINT文は、後からロールバックできるトランザクションの時点を特定します。

表10-1のセッションは、トランザクション制御の基本概念を示しています。

表10-1 トランザクション制御

T	セッション	説明
t0	COMMIT;	セッション内の既存のトランザクションを終了します。
t1	SET TRANSACTION NAME 'sal_update';	トランザクションを開始し、トランザクションに sal_update という名前を付けます。
t2	UPDATE employees SET salary = 7000 WHERE last_name = 'Banda';	Banda の給与を 7000 に更新します。
t3	SAVEPOINT after_banda_sal;	after_banda_sal という名前のセーブポイントを作成し、このトランザクションで行われた変更をこの時点までロールバックできるようにします。
t4	UPDATE employees SET salary = 12000 WHERE last_name = 'Greene';	Greene の給与を 12000 に更新します。
t5	SAVEPOINT after_greene_sal;	after_greene_sal という名前のセーブポイントを作成し、このトランザクションで行われた変更をこの時点までロールバックできるようにします。

T	セッション	説明
t6	ROLLBACK TO SAVEPOINT after_banda_sal;	トランザクションを t3 までロールバックし、t4 で行われた Greene の給与の更新を取り消します。sal_update トランザクションは終了していません。
t7	UPDATE employees SET salary = 11000 WHERE last_name = 'Greene';	トランザクション sal_update で、Greene の給与を 11000 に更新します。
t8	ROLLBACK;	トランザクション sal_update で行われたすべての変更をロールバックし、トランザクションを終了します。
t9	SET TRANSACTION NAME 'sal_update2';	セッションで新しいトランザクションを開始し、トランザクションに sal_update2 という名前を付けます。
t10	UPDATE employees SET salary = 7050 WHERE last_name = 'Banda';	Banda の給与を 7050 に更新します。
t11	UPDATE employees SET salary = 10950 WHERE last_name = 'Greene';	Greene の給与を 10950 に更新します。
t12	COMMIT;	トランザクション sal_update2 で行われたすべての変更をコミットし、トランザクションを終了します。このコミットにより、変更がオンライン REDO ログ・ファイルに保存されることが保証されます。

関連項目:

トランザクション制御文について学習するには、『Oracle Database SQL言語リファレンス』を参照してください。

トランザクション名

トランザクション名は、ユーザー指定のオプションのタグであり、トランザクションが実行する作業のリマインダとして機能します。トランザクションの命名にはSET TRANSACTION . . . NAME文を使用し、トランザクションの最初の文として使用する必要があります。

表10-1では、最初のトランザクションがsal_update、2つ目のトランザクションがsal_update2と命名されています。

トランザクション名を使用すると、次のような利点があります。

- 名前を付けることで、長時間実行中のトランザクションの監視およびインダウト分散トランザクションの解決が容易になります。
- アプリケーションのトランザクションIDとともにトランザクション名を表示できます。たとえば、システムの活動状況を監視する場合、データベース管理者はOracle Enterprise Manager(Enterprise Manager)内のトランザクション名を表示できます。
- トランザクション名はトランザクション監査REDOレコードに書き込まれるため、LogMinerを使用してREDOログ内の特定のトランザクションを検索できます。

- トランザクション名を使用して、V$TRANSACTIONなどのデータ・ディクショナリ・ビュー内の特定のトランザクションを検索できます。

関連項目:

- Oracle Enterprise Manager
- V$TRANSACTIONについて学習するには、『Oracle Databaseリファレンス』を参照してください
- SET TRANSACTIONについて学習するには、『Oracle Database SQL言語リファレンス』を参照してください

アクティブなトランザクション

アクティブなトランザクションとは、開始した後、まだコミットまたはロールバックされていないトランザクションのことです。

表10-1では、sal_updateトランザクションでデータを変更する最初の文が、Bandaの給与の更新です。この更新の実行が成功してからROLLBACK文がトランザクションを終了するまで、sal_updateトランザクションはアクティブです。

トランザクションによるデータ変更は、トランザクションがコミットまたはロールバックされるまでは一時的です。トランザクションが終了する前のデータの状態は、次の表のとおりです。

表10-2 トランザクションが終了する前のデータの状態

状態	説明	さらに学習するには
Oracle Database により、SGA に UNDO 情報が生成されています。	この UNDO データには、トランザクションの SQL 文によって変更された古いデータ値が含まれています。	「コミット読取り分離レベルでの読取り一貫性」
Oracle Database により、SGA のオンライン REDO ログ・バッファに REDO が生成されています。	REDO ログ・レコードには、データ・ブロックおよび UNDO ブロックに対する変更内容が含まれています。	「REDO ログ・バッファ」
SGA のデータベース・バッファに変更が加えられます。	コミットされたトランザクションのデータ変更は、SGA のデータベース・バッファに格納されており、データベース・ライター(DBW)によりデータファイルに必ずしもすぐに書き込まれるわけではありません。ディスクへの書込みは、コミットの前に行われることも、後に行われることもあります。	データベース・バッファ・キャッシュ
データ変更の影響を受ける行はロックされます。	他のユーザーは、影響を受ける行のデータを変更することも、未コミットの変更を表示することもできません。	「ロック動作の概要」

セーブポイント

セーブポイントは、トランザクションのコンテキスト内のユーザー宣言による中間マーカーです。

内部的には、セーブポイント・マーカーはSCNに解決されます。セーブポイントは、ロング・トランザクションをいくつかの小さい部分

に分割します。

ロング・トランザクションでセーブポイントを使用すると、そのトランザクション内の宣言されたセーブポイントからトランザクション内の現時点までの間に実行された作業を、後でロールバックできるようになります。したがって、エラーが発生してもすべての文を再発行する必要はありません。表10-1では、セーブポイントafter_banda_salを作成しているため、Greeneの給与に対する更新をこのセーブポイントまでロールバックできます。

セーブポイントへのロールバック

未コミットのトランザクションにおけるセーブポイントへのロールバックは、指定したセーブポイント後に行われたすべての変更の取消しを意味しますが、トランザクション自体のロールバックは意味しません。

表10-1でROLLBACK TO SAVEPOINT after_banda_salを実行したときのように、トランザクションがセーブポイントまでロールバックされると、次の処理が行われます。

1. Oracle Databaseにより、セーブポイントの後に実行された文のみがロールバックされます。

   表10-1では、ROLLBACK TO SAVEPOINTにより、Greeneに対するUPDATEがロールバックされますが、Bandaに対するUPDATEはロールバックされません。

2. Oracle Databaseでは、ROLLBACK TO SAVEPOINT文に指定されたセーブポイントは保存されますが、それより後のセーブポイントはすべて失われます。

   表10-1では、ROLLBACK TO SAVEPOINTによってafter_greene_salセーブポイントが失われます。

3. Oracle Databaseにより、指定したセーブポイントの後に取得された表と行のすべてのロックが解放されますが、そのセーブポイントより前に取得されたすべてのデータ・ロックは保持されます。

トランザクションはアクティブであり、継続できます。

関連項目:

- ROLLBACK文およびSAVEPOINT文について学習するには、『Oracle Database SQL言語リファレンス』を参照してください。
- トランザクション処理および制御について学習するには、『Oracle Database PL/SQL言語リファレンス』を参照してください。

トランザクションのエンキュー

状況によっては、以前にロックされたリソースを待機中のトランザクションが、セーブポイントへのロールバック後も引き続きブロックされる場合があります。

トランザクションが別のトランザクションによってブロックされた場合、このトランザクションはブロックしているトランザクション上でエンキューされるため、ブロックされたトランザクションを続行するには、ブロックしているトランザクション全体をコミットまたはロールバックする必要があります。

次の表に示すシナリオでは、セッション1でDML文が実行される前に作成されたセーブポイントに、セッション1がロールバックされます。ただし、セッション2はセッション1のトランザクションの完了を待機するため、引き続きブロックされます。

表10-3 セーブポイントへのロールバックの例

T	セッション1	セッション2	セッション3	説明

T	セッション1	セッション2	セッション3	説明
t0	UPDATE employees SET salary=7000 WHERE last_name= 'Banda';			セッション 1 でトランザクションが開始されます。このセッションでは、Banda の行(TX)に排他ロック、表に副排他表ロック(SX)が適用されます。
t1	SAVEPOINT after_banda_sal;			セッション 1 により、after_banda_sal という名前のセーブポイントが作成されます。
t2	UPDATE employees SET salary=12000 WHERE last_name= 'Greene';			セッション 1 により、Greene の行がロックされます。
t3		UPDATE employees SET salary=14000 WHERE last_name= 'Greene';		セッション 2 により、Greene の行の更新が試行されますが、セッション 1 でこの行がロックされているため、ロックを取得できません。セッション 2 ではトランザクションが開始しません。
t4	ROLLBACK TO SAVEPOINT after_banda_sal;			セッション 1 により、Greene の給与の更新がロールバックされ、Greene の行ロックが解放されます。t0 で取得された表ロックは解放されません。 この時点で、セッション 2 は引き続きセッション 1 にブロックされています。これは、まだ完了していないセッション 1 のトランザクション上でセッション 2 がエンキューされているためです。
t5			UPDATE employees SET salary=11000 WHERE last_name= 'Greene';	Greene の行は現在ロックが解放されているため、

T	セッション1	セッション2	セッション3	説明
				セッション 3 により、Greene の行を更新するためのロックが取得されます。この文により、セッション 3 のトランザクションが開始されます。
t6	COMMIT;			セッション 1 がコミットし、そのトランザクションが終了します。セッション 2 は現在、セッション 3 のトランザクションの後で Greene の行を更新するようにエンキューされています。

関連項目:

Oracle Databaseによってロックが解放される時期についてさらに学習するには、「ロックの期間」を参照してください

トランザクションのロールバック

未コミットのトランザクションをロールバックすると、そのトランザクション内のSQL文によって実行された、データに対するすべての変更が取り消されます。

トランザクションのロールバック後には、そのトランザクション内で実行された作業の結果は存在しません。セーブポイントを参照しないで、トランザクション全体をロールバックした場合、Oracle Databaseでは次の処理が実行されます。

- トランザクションのすべてのSQL文によるすべての変更が、対応するUNDOセグメントを使用して取り消されます。

  すべてのアクティブなトランザクションのトランザクション表エントリには、そのトランザクションのすべてのUNDOデータを示すポインタが(適用時とは逆の順序で)格納されています。データベースでは、UNDOセグメントからデータを読み取り、操作を逆順にして、適用されたとおりにUNDOエントリをマークします。したがって、トランザクションで行が挿入されると、ロールバックによって削除されます。トランザクションで行が更新されると、ロールバックによって更新が元に戻されます。トランザクションで行が削除されると、ロールバックによって再挿入されます。表10-1では、GreeneとBandaの給与の更新が、ROLLBACKによって元に戻されています。

- トランザクションで保持されているすべてのデータ・ロックが解放されます。

- トランザクション内のすべてのセーブポイントが消去されます。

  表10-1では、ROLLBACKによってセーブポイントafter_banda_salが削除されています。セーブポイントafter_greene_salは、ROLLBACK TO SAVEPOINT文によって削除されています。

- トランザクションを終了します

  表10-1では、ROLLBACKによって、データベースは最初のCOMMITが実行された後と同じ状態で残ります。

ロールバックの持続時間は、変更されたデータ量の関数になります。

関連項目:

- UNDOセグメント
- ROLLBACKコマンドについて学習するには、Oracle Database SQL言語リファレンスを参照してください。

トランザクションのコミット

コミットは、現行のトランザクションを終了し、トランザクションで実行されたすべての変更を確定します。

表10-1では、2つ目のトランザクションがsal_update2で開始され、明示的なCOMMIT文で終了しています。これにより、2つのUPDATE文の結果である変更が確定されます。

トランザクションをコミットすると、次のアクションが実行されます。

- データベースにより、COMMITのSCNが生成されます。

  関連するUNDO表領域の内部トランザクション表に、トランザクションのコミットが記録されます。トランザクションに対応する一意のSCNは、トランザクション表に割り当てられて記録されます。

- ログ・ライター・プロセス(LGWR)により、REDOログ・バッファ内に残されたREDOログ・エントリがオンラインREDOログに書き込まれ、さらにトランザクションのSCNがオンラインREDOログに書き込まれます。これが、トランザクションのコミットを構成するアトミック・イベントです。

- Oracle Databaseにより、行と表に対して保持されているロックが解放されます。

  未コミットのトランザクションによって保持されているロックを待機してエンキューされていたユーザーが、作業を続行できるようになります。

- Oracle Databaseにより、セーブポイントが削除されます。

  表10-1では、sal_updateトランザクション内にセーブポイントが存在しないため、消去されたセーブポイントはありません。

- Oracle Databaseにより、コミットのクリーンアウトが実行されます。

  コミットされたトランザクションのデータが含まれている変更済ブロックが引き続きSGA内に存在し、このブロックを変更する他のセッションがない場合は、データベースにより、このブロックからロックに関するトランザクション情報(ITLエントリ)が削除されます。

  COMMITでブロックを消去し、後続のSELECTによるこのタスクの実行を不要にすると理想的です。指定された行に対してITLエントリが存在しない場合はロックされません。指定された行に対してITLエントリが存在する場合はロックされている可能性があるため、セッションはこの対象トランザクションがコミットされたかどうかを判断するために、UNDOセグメント・ヘッダーをチェックする必要があります。対象のトランザクションがコミットされている場合、セッションはブロックをクリーンアウトし、これによってREDOが生成されます。ただし、COMMITが以前にITLをクリーンアウトした場合、チェックとクリーンアウトは不要です。

ノート:

ブロックのクリーンアウトでは REDO が生成されるため、問合せによって REDO が生成され、結果として

次のチェックポイント時にブロックが書き込まれる場合があります。

- Oracle Databaseにより、トランザクションに完了のマークが付けられます。

トランザクションをコミットした後、ユーザーは変更を表示できます。

通常のコミットは、トランザクションのサイズにかかわらず高速な操作です。コミットの速度は、トランザクション内で変更されるデータのサイズによって遅くなりません。コミットで最も時間がかかる部分は、LGWRによって実行される物理ディスクI/Oです。ただし、LGWRではREDOログ・バッファの内容が段階的にバックグラウンドで書き込まれるため、LGWRにかかる時間は削減されます。

デフォルトの動作では、LGWRによりREDOがオンラインREDOログに同時に書き込まれ、トランザクションはバッファ内のREDOがディスクに移動するまで待機してから、コミットをユーザーに戻します。ただし、アプリケーション開発者は、トランザクションのコミット待機時間を短縮するために、REDOが非同期に書き込まれ、REDOがディスクに移動されるまでトランザクションが待機せずに、COMMITコールからすぐに戻れるよう指定できます。

関連項目:

- 「シリアライズ可能分離レベル」
- 「ロックのメカニズム」
- LGWRの詳細は、「バックグラウンド・プロセスの概要」を参照してください
- 非同期コミットの詳細は、『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』を参照

トランザクション・ガードの概要

トランザクション・ガードは、アプリケーションが**トランザクションの冪等性**を提供するために使用できるAPIの1つであり、これにより、データベースは、トランザクションがコミットされて完了しているかどうかを示す保証付きのコミット結果を保持できます。Oracle Databaseには、JDBCシン、OCI、OCCIおよびODP.Net用のAPIが備わっています。

リカバリ可能エラーは、実行中のアプリケーション・セッション論理と関係なく、外部システムの障害が原因で発生します。リカバリ可能なエラーは、フォアグラウンド・プロセス、ネットワーク、ノード、記憶域およびデータベースの計画および計画外停止の後に発生します。停止によりクライアント・アプリケーションとデータベース間の接続が中断した場合、アプリケーションでは切断エラー・メッセージを受信します。接続切断時に実行中だったトランザクションは、処理中のトランザクションと呼ばれます。

トランザクションを再送信するか、それとも結果(コミット済または未コミット)をクライアントに戻すかどうかを決めるため、アプリケーションでは処理中のトランザクションの結果を判断する必要があります。Oracle Database 12cまでは、クライアントに戻されたコミット・メッセージは残りませんでした。トランザクションをチェックしても、チェック後にそれがコミットされないと保証するものではなく、トランザクションの重複や他の形の論理的な破損が生じる可能性があります。たとえば、ユーザーが本をオンラインで購入する際、Webブラウザをリフレッシュして、同じ本に対して2回請求される可能性があります。

関連項目:

- トランザクションの概要
- トランザクション・ガードについて学習するには、Oracle Database開発ガイドを参照してください
- トランザクション・ガードのサービスを構成する方法を学習するには、Oracle Real Application Clusters管理およ

びデプロイメント・ガイドを参照してください

トランザクション・ガードの利点

Oracle Database 12cから、トランザクション・ガードでは、リカバリ可能な停止の後に、処理中のトランザクションのステータスを判断するためのツールがアプリケーションに提供されるようになりました。

トランザクション・ガードを使用すれば、アプリケーションでは、トランザクションが1回しか実行されていないことを確認できます。たとえば、ネット書店アプリケーションで前に送信したコミットが失敗したことが確認されれば、アプリケーションは安全に再送信できます。

トランザクション・ガードでは、アプリケーションで重複送信が実行されないように、実行を1回のみにするためのツールを提供します。トランザクション・ガードでは、トランザクションごとに既知の結果を提供します。

トランザクション・ガードは、Oracle Databaseの中核機能です。アプリケーション・コンティニュイティでは、エンド・ユーザーに停止を隠すときにトランザクション・ガードを使用します。トランザクション・ガードがないと、エラー後に再試行するアプリケーションで、重複トランザクションがコミットされる可能性があります。

関連項目:

- トランザクション・ガードを使用して、開発者がアプリケーションの高可用性を実現するのを助けるアプリケーション・コンティニュイティについて学習するには、「アプリケーション・コンティニュイティの概要」を参照してください
- サポートおよび含まれているトランザクションのタイプなど、トランザクション・ガードについて学習するには、Oracle Database開発ガイドを参照してください。

トランザクション・ガードにより行われる処理

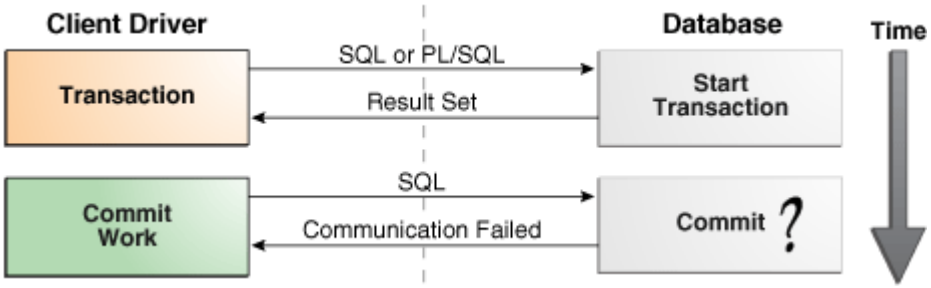
この項では、失われたコミット・メッセージの問題と、トランザクション・ガードにより、論理トランザクションIDを使用してその問題がどのように解決されるかについて説明します。

失われたコミット・メッセージ

冪等性の設計時に、開発者は、コミット文送信後の通信エラーの問題に対処する必要があります。コミット・メッセージは、データベースに残らないため、エラー後に取得できません。

次の図は、クライアント・アプリケーションとデータベース間の対話をハイレベルに表現したものです。

図10-2 失われたコミット・メッセージ



標準コミットの場合、データベースではトランザクションをコミットすると、クライアントに成功メッセージを戻します。図10-2では、クライアントがコミット文を送信し、通信が失敗したというメッセージを受信します。このタイプの失敗は、データベース・インスタンスの障害またはネットワークの停止など、いくつかの原因で発生する可能性があります。このシナリオでは、クライアントにはトランザク

ションの状態がわかりません。

通信の失敗後も、データベースではその送信を続行し、クライアントが切断されていることを認識していない場合があります。トランザクション状態をチェックしても、チェック後にアクティブ・トランザクションがコミットされないという保証にはなりません。この古い情報のために、クライアントがコミットを再送信した場合、データベースではトランザクションを繰り返し、その結果論理的な破損が生じる可能性があります。

論理トランザクションID

Oracle Databaseでは、**論理トランザクションID**というグローバルに一意の識別子を使用することで、通信障害を解決します。

このIDには、セッションの最初の接続時に割り当てられた論理セッション番号と、セッションのコミットまたはロールバックごとに更新される実行コミット番号が含まれます。[脚注1] アプリケーション側からすると、論理トランザクションIDは、失敗したセッションで送信された最後のデータベース・トランザクションを一意に識別します。

データベースでは、1つ以上のトランザクションがコミットされるクライアントからのラウンド・トリップごとに1つの論理トランザクションIDが格納されます。このIDにより、データをコミットするラウンド・トリップごとに、アプリケーションとデータベース間の対話のために、トランザクション冪等性を提供できます。

1回かぎりのプロトコルにより、データベースに次のことを要求することで、コミット結果へのアクセスが可能になります。

- 再試行のために認められた保持期間中の論理トランザクションIDの維持
- コミット時の論理トランザクションIDの持続

トランザクションの実行中、データベースとクライアントの両方で、論理トランザクションIDが保持されます。データベースでは、認証時には接続プールから、各ラウンド・トリップでは1つ以上のコミット操作を実行するクライアント・ドライバから借りて、論理トランザクションIDをクライアントに与えます。

リカバリ可能なエラーの後、最後のトランザクションの結果が確認できるように、アプリケーションでは、Java、OCI、OCCIまたはODP.Net APIを使用して、クライアントで保持されている論理トランザクションIDを取得します。次に、アプリケーションではPL/SQLプロシージャDBMS_APP_CONT.GET_LTXID_OUTCOMEを論理トランザクションIDで起動して、最後の送信結果のコミット済(trueまたはfalse)およびユーザー・コール完了(trueまたはfalse)を確認します。

トランザクション・ガードを使用すると、アプリケーションでは、エラーがリカバリ可能で、セッションでの最後のトランザクションがコミットされていない場合、トランザクションを再生できます。最後のトランザクションがコミットされていて、ユーザー・コールが完了している場合、アプリケーションは続行可能です。アプリケーションでは、クライアントが次にとるべきアクションを決定できるように、トランザクション・ガードを使用して既知の結果をクライアントに戻すことができます。

関連項目:

- 論理トランザクションIDについて学習するには、Oracle Database開発ガイドを参照してください
- DBMS_APP_CONT.GET_LTXID_OUTCOMEプロシージャについてさらに学習するには、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

トランザクション・ガード: 例

このシナリオでは、リカバリ可能エラーのためにコミット・メッセージが失われています。

トランザクション・ガードでは、論理トランザクションIDを使用して、COMMIT文の結果を保存し、トランザクションに既知の結果が

あることを保証します。

図10-3 論理トランザクションのステータスのチェック

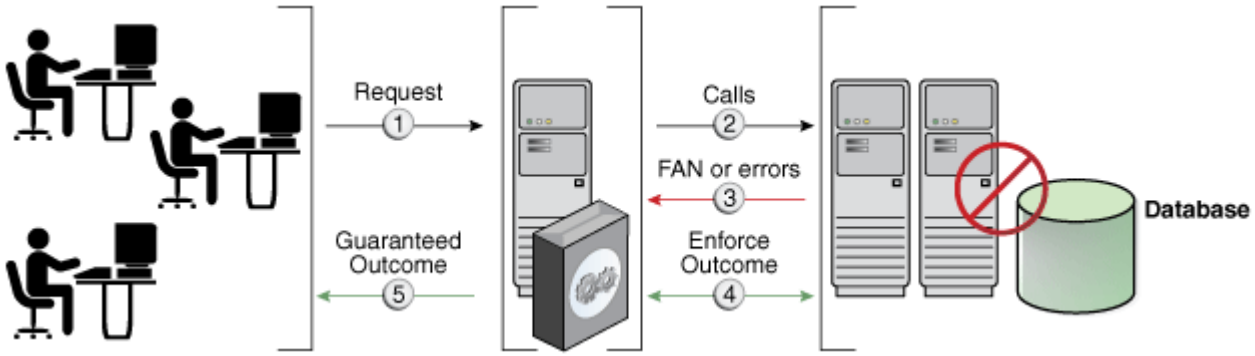


図10-3では、データベースがアプリケーションに、トランザクションがコミットされたかどうか、および最後のユーザー・コールが完了したかどうかを知らせます。その結果、アプリケーションではエンド・ユーザーに結果を戻すことができます。可能性は次のとおりです。

- トランザクションがコミットされ、ユーザー・コールが完了している場合、アプリケーションではエンド・ユーザーに結果を戻し、続行できます。
- トランザクションはコミットされているが、ユーザー・コールが完了していない場合、アプリケーションではエンド・ユーザーに警告付きで結果を戻すことができます。例としては、OUTバインドの喪失または処理済行数の喪失などがあります。一部のアプリケーションでは、その他の情報に依存しますが、その他のアプリケーションではそれはありません。
- ユーザー・コールが完了していない場合、アプリケーションではエンド・ユーザーにこの情報を戻すか、安全に再送信することができます。プロトコルは保証されています。コミット・ステータスからfalseが戻された場合、最後の送信はコミットを阻止されます。

関連項目:

トランザクション・ガードの使用方法を学習するには、Oracle Database開発ガイドを参照してください

アプリケーション・コンティニュイティの概要

アプリケーション・コンティニュイティは、計画および計画外の停止後に完了していないアプリケーション・リクエストをリプレイすることで、アプリケーションから停止を隠そうとします。このような状況では、リクエストがアプリケーションからの作業単位です。

通常、リクエストは、1つのデータベース接続におけるDML文や、1つのWebリクエストに対するその他のデータベース・コールに対応しています。一般に、リクエストは、データベース接続のチェックアウトとチェックインの間で、接続プールから行われるコールにより区別されます。

この項では、次の項目について説明します。

- アプリケーション・コンティニュイティの利点
- アプリケーション・コンティニュイティ・アーキテクチャ

アプリケーション・コンティニュイティの利点

開発者にとっての基本的な問題は、失われたデータベース・セッションをエンド・ユーザーから隠す方法です。

アプリケーション・コンティニュイティでは、コンポーネントによりデータベースとクライアント間の対話が中断した場合、データベース・セッションをリストアすることで、失われたセッションの問題を解決しようとします。リストアされたデータベース・セッションには、すべて

の状態、カーソル、変数、およびトランザクションが存在する場合は最新のトランザクションが含まれます。

アプリケーション・コンティニュイティのユースケース

典型的な例では、クライアントがデータベースにリクエストを送信すると、トランザクション状態と非トランザクション状態の両方が構築されます。

クライアントでの状態は現行のままですが、入力されたデータ、戻されたデータ、キャッシュされたデータや変数がある可能性があります。ただし、アプリケーションが動作するために必要とするデータベース・セッションの状態は失われます。

クライアント・リクエストにより1つ以上のトランザクションが開始される場合、アプリケーションでは次の可能性に直面します。

- コミットが発行されている場合、クライアントに戻されるコミット・メッセージは永続的ではありません。クライアントでは、そのリクエストがコミットされたかどうか、非トランザクション処理状態のどこに到達したのかがわかりません。
- コミットが発行されていないか、発行されたけれども実行されなかった場合、処理中のトランザクションはロールバックされ、正しい状態のセッションを使用して再送信する必要があります。

リプレイが成功した場合、計画および計画外停止のためのデータベース・ユーザー・サービスは中断されません。アプリケーションで表示され、場合によっては実行されているデータ変更をデータベースが検出した場合、リプレイは拒否されます。リプレイ開始の許容時間を超過している場合、アプリケーションで制限されたコールを使用する場合、またはアプリケーションで`disableReplay`メソッドを使用してリプレイを明示的に無効にしている場合、リプレイは試行されません。

関連項目:

データベース・セッションに対するアプリケーション・コンティニュイティの動作についてさらに学習するには、Oracle Real Application Clusters管理およびデプロイメント・ガイドを参照してください

計画的メンテナンスに対するアプリケーション・コンティニュイティ

計画済停止に対するアプリケーション・コンティニュイティにより、アプリケーションは確実に排出または移行できるデータベース・セッションの操作を続行できます。

計画的メンテナンスでは、アプリケーション作業を中断する必要はありません。アプリケーション・コンティニュイティにより、現在の位置からメンテナンスの影響を現在受けていない新しい位置まで排出するためのアクティブな作業時間が提供されます。排出間隔の終了時に、メンテナンスが計画されたデータベース・インスタンス上にセッションが残る可能性があります。アプリケーション・コンティニュイティでは、これらのセッションを強制的に切断するかわりに、正常に稼働しているサイトにこれらのセッションをフェイルオーバーし、処理中のトランザクションを再送信できます。

アプリケーション・コンティニュイティを有効にすると、データベースで次のことを実現できます。

- メンテナンス中に新規または既存のいずれの作業でもエラーを報告しない
- アクティブなデータベース・セッションを他の機能サービスにリダイレクトする
- メンテナンス中およびメンテナンス後にデータベース・セッションを必要に応じて均衡化する

SRVCTLユーティリティ、Global Data Services制御ユーティリティ(GDSCTL)およびOracle Data Guard Brokerの`drain_timeout`および`stop_option`サービス属性を使用して、計画的メンテナンス中の排出動作を制御します。`DBMS_SERVICE`パッケージは、基礎となるインフラストラクチャを提供します。

関連項目:

- アプリケーション・コンティニュイティをさらに学習するには、Oracle Real Application Clusters管理およびデプロイメント・ガイドを参照してください
- DBMS_SERVICEについてさらに学習するには、『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』を参照
- SRVCTLコマンド・リファレンスについては、『Oracle Real Application Clusters管理およびデプロイメント・ガイド』を参照
- GDSCTLコマンド・リファレンスについては、Oracle Database Global Data Services概要および管理ガイドを参照

アプリケーション・コンティニュイティ・アーキテクチャ

アプリケーション・コンティニュイティの主なコンポーネントは、実行時、再接続およびリプレイです。

フェーズは次のとおりです。

1. 通常の実行時

   このフェーズでは、アプリケーション・コンティニュイティによって次のタスクが実行されます。

   - データベース・リクエストを識別する
   - ローカル・コールとデータベース・コールがリプレイ可能かどうかを判定する
   - リプレイの有効化と(必要に応じて)キューの管理を行うためのプロキシ・オブジェクトを作成する
   - データベース・リクエストが終了するか、リプレイが無効になるまで、元のコールとこれらのコールの検証を保持する

2. 再接続

   このフェーズは、リカバリ可能なエラーによってトリガーされます。アプリケーション・コンティニュイティによって次のタスクが実行されます。

   - データベース・リクエストのリプレイが有効であることを確認する
   - タイムアウトを管理する
   - データベースへの新しい接続を取得し、それが有効なデータベース・ターゲットであることを検証する
   - トランザクション・ガードを使用して、最後のトランザクションが正常にコミットされたかどうかを判定する(コミットされたトランザクションは再送信されない)

3. リプレイ

   アプリケーション・コンティニュイティによって次のタスクが実行されます。

   - キューに保持されているコールをリプレイする
   - ユーザーが参照可能な結果の変更がリプレイ中に出現した場合は、リプレイを無効にする
   - コミットは許可しないが、(エラーが発生した)最後のリコールのコミットは許可する

リプレイに成功した後は、障害点からリクエストが続行されます。

関連項目:

- 「トランザクション・ガードの概要」
- アプリケーション・コンティニュイティについてさらに学習するには、Oracle Real Application Clusters管理およびデプロイメント・ガイドを参照してください
- JDBCおよびアプリケーション・コンティニュイティについてさらに学習するには、*Oracle Database JDBC開発者ガイド*および*Oracle Database JDBC Java APIリファレンス*を参照してください

自律型トランザクションの概要

自律型トランザクションは、メイン・トランザクションと呼ばれる他のトランザクションからコールできる独立したトランザクションです。コール側のトランザクションを一時停止して、自律型トランザクション内でSQL操作を実行し、その操作をコミットまたは取り消してから、コール側トランザクションを再開できます。

自律型トランザクションは、コール側トランザクションでコミットまたはロールバックするかどうかに関係なく、独立して実行する必要があるアクションを実装する場合に便利です。たとえば、株購入のトランザクションでは、全体的な株購入が成立するかどうかにかかわらず、顧客データをコミットする必要があります。また、トランザクション全体がロールバックされた場合でも、デバッグ表にエラー・メッセージを記録することも必要となります。

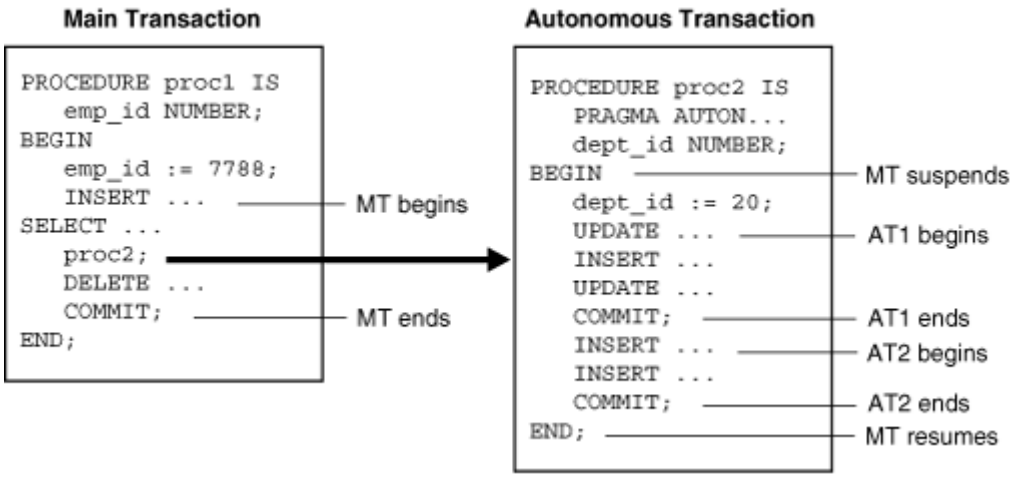
自律型トランザクションの特性は、次のとおりです。

- 自律型トランザクションでは、メイン・トランザクションによって加えられたがコミットされていない変更は取り扱わず、ロックやリソースをメイン・トランザクションと共有することもありません。
- 自律型トランザクション内の変更は、自律型トランザクションのコミット時に他のトランザクションに表示されるようになります。そのため、ユーザーはメイン・トランザクションのコミットを待機しなくても更新情報にアクセスできます。
- 自律型トランザクションで他の自律型トランザクションを開始できます。自律型トランザクションをコールできるレベル数には、リソース制限以外の制限はありません。

PL/SQLでは、自律型トランザクションは自律型スコープ内で実行されます(自律型スコープとは、プラグマAUTONOMOUS_TRANSACTIONでマークされたルーチンです)。このコンテキストでは、トップレベルの無名PL/SQLブロックおよびPL/SQLサブプログラムとPL/SQLトリガーが、ルーチンに含まれています。プラグマは、コンパイラにコンパイル・オプションの実行を指示するディレクティブです。プラグマAUTONOMOUS_TRANSACTIONによって、このプロシージャは実行時に親トランザクションから独立した新しい自律型トランザクションとして実行する必要があることが、データベースに指示されます。

次の図に、制御がメイン・ルーチン(MT)から自律型ルーチンに移り、再びメイン・ルーチンに戻ってくるまでのフローを示します。メイン・ルーチンはproc1、自律型ルーチンはproc2です。自律型ルーチンでは、制御がメイン・ルーチンに戻る前に、複数のトランザクション(AT1およびAT2)をコミットできます。

図10-4 トランザクション制御フロー



自律型ルーチンの実行可能セクションに入ると、メイン・ルーチンが一時停止します。この自律型ルーチンを終了すると、メイン・ルーチンが再開されます。

図10-4では、proc1内部のCOMMITにより、この文自体の作業のみでなく、そのセッション内で実行されたすべての保留中の作業が確定されます。ただし、proc2内のCOMMITでは、proc2トランザクション内で実行された作業のみが確定されます。したがって、トランザクションAT1およびAT2内のCOMMIT文は、MTトランザクションに影響を与えません。

関連項目:

自律型トランザクションの使用方法を学習するには、Oracle Database開発ガイドおよびOracle Database PL/SQL言語リファレンスを参照してください

分散トランザクションの概要

分散トランザクションとは、データベース・リンクと呼ばれるスキーマ・オブジェクトを使用し、分散データベースの複数の異なるノード上でデータを更新する、1つ以上の文を含むトランザクションのことです。

分散データベースは、分散システム内のデータベースの集合であり、アプリケーションには1つのデータソースとして認識されます。データベース・リンクは、1つのデータベース・インスタンスが別のデータベース・インスタンスにログインする様子を記述します。

ローカル・データベースでのトランザクションとは異なり、分散トランザクションでは、複数データベース上にあるデータが変更されます。その結果、データベースではトランザクションでの変更のコミットやロールバックをアトミックな単位として処理する必要があるため、分散トランザクション処理はより複雑になります。トランザクション全体をコミットまたはロールバックする必要があります。Oracle Databaseはネットワーク全体でトランザクション制御を調整し、ネットワーク障害やシステム障害が発生した場合にもデータ整合性が保たれるようにする必要があります。

関連項目:

分散トランザクションの管理方法を学習するには、『Oracle Database管理者ガイド』を参照してください

2フェーズ・コミット

2フェーズ・コミット・メカニズムは、分散トランザクションに関係するすべてのデータベースが、そのトランザクション内の文のすべてをコミットするか、すべてを取り消すかのどちらか一方のみになるように保証するものです。このメカニズムでは、整合性制約、リモート・プロシージャ・コールおよびトリガーによって実行される暗黙的DMLも保護されます。

複数データベース間の2フェーズ・コミットでは、1つのデータベースで分散トランザクションが調整されます。開始ノードはグローバル・コーディネータと呼ばれます。このコーディネータによって、他のデータベースでコミットの準備ができているかどうかが確認されます。いずれかのデータベースで否定のレスポンスがあった場合は、トランザクション全体がロールバックされます。すべてのデータベースから肯定のレスポンスがあった場合は、コーディネータからメッセージがブロードキャストされ、各データベース上でコミットが確定されます。

2フェーズ・コミット・メカニズムは、分散トランザクションを発行するユーザーからは意識されません。事実、ユーザーは、トランザクションが分散していることも認識する必要はありません。トランザクションの終了を示すCOMMIT文は、自動的に2フェーズ・コミット・メカニズムをトリガーします。データベース・アプリケーション本体に分散トランザクションを含めるのに、コーディングや複雑な構文の記述は必要ありません。

関連項目:

- 2フェーズ・コミットについて学習するには、Oracle Database管理者ガイドを参照してください。
- Oracle Database SQL言語リファレンス

インダウト・トランザクション

インダウト分散トランザクションは、システム障害やネットワーク障害によって2フェーズ・コミットが中断されたときに発生します。

たとえば、2つのデータベースから調整データベースにコミットの準備ができたことが報告されたものの、このメッセージの受信直後に調整データベース・インスタンスが失敗した場合などです。この場合、コミットの準備ができている2つのデータベースはハング状態となり、結果の通知を待機し続けます。

リカバラ(RECO)・バックグラウンド・プロセスでは、インダウト分散トランザクションの結果が自動的に解決されます。障害が修復され、通信が再確立された後、各ローカルのOracleデータベースのRECOプロセスが、関係するすべてのノード上でインダウト分散トランザクションを自動的にコミットするか、ロールバックします。

長時間にわたる障害が発生した場合、Oracle Databaseでは、各ローカル管理者が障害によって発生したインダウト分散トランザクションを手動でコミットまたは取り消すことができます。このオプションにより、ローカル・データベース管理者は、長時間にわたる障害によって無期限にロックされる可能性のあるリソースを解放できます。

あるデータベースを過去のある時点までリカバリする必要がある場合、データベースのリカバリ機能を使用すると、他のサイトのデータベース管理者が、自分のデータベースを過去のその同じ時点に戻すことができます。この操作により、グローバル・データベースは一貫した状態に保たれます。

関連項目:

- 「リカバラ・プロセス(RECO)」
- インダウト・トランザクションの管理方法を学習するには、『Oracle Database管理者ガイド』を参照してください

脚注の凡例

脚注1:

Oracle Real Application Clusters (Oracle RAC)の場合、論理トランザクションIDには、接頭辞としてデータベース・インスタンス番号が含まれます。

第IV部 Oracleデータベース記憶域構造

この部では、論理的および物理的な記憶域構造など、Oracleデータベースの基本的な構造上のアーキテクチャについて説明します。

この部は、次の章で構成されています。

- 物理記憶域構造
- 論理記憶域構造

11 物理記憶域構造

Oracle Databaseの物理データベース構造は、オペレーティング・システム・レベルで表示可能です。

この章の構成は、次のとおりです。

- 物理記憶域構造の概要
- データファイルの概要
- 制御ファイルの概要
- オンラインREDOログの概要

物理記憶域構造の概要

RDBMSの特徴の1つは、表、ビュー、索引などの論理データ構造が物理記憶域構造から独立していることです。

物理構造と論理構造は分離されているため、論理構造へのアクセスに影響を与えずに、データの物理記憶域を管理できます。たとえば、データベース・ファイルの名前を変更しても、ファイルに格納されている表の名前は変更されません。

Oracleデータベースは、Oracleデータを永続記憶域に格納する一連のファイルです。この項では、CREATE DATABASE文を発行したときに生成されるデータベース・ファイルについて説明します。

- データファイルと一時ファイル

  データファイルは永続ストレージ上にOracle Databaseが作成した物理ファイルであり、表や索引などのデータ構造が含まれます。一時ファイルは一時表領域に属するデータファイルです。データベースでは、他のプログラムでは読み取れないOracle独自の形式で、データをこれらのファイルに書き込みます。

- 制御ファイル

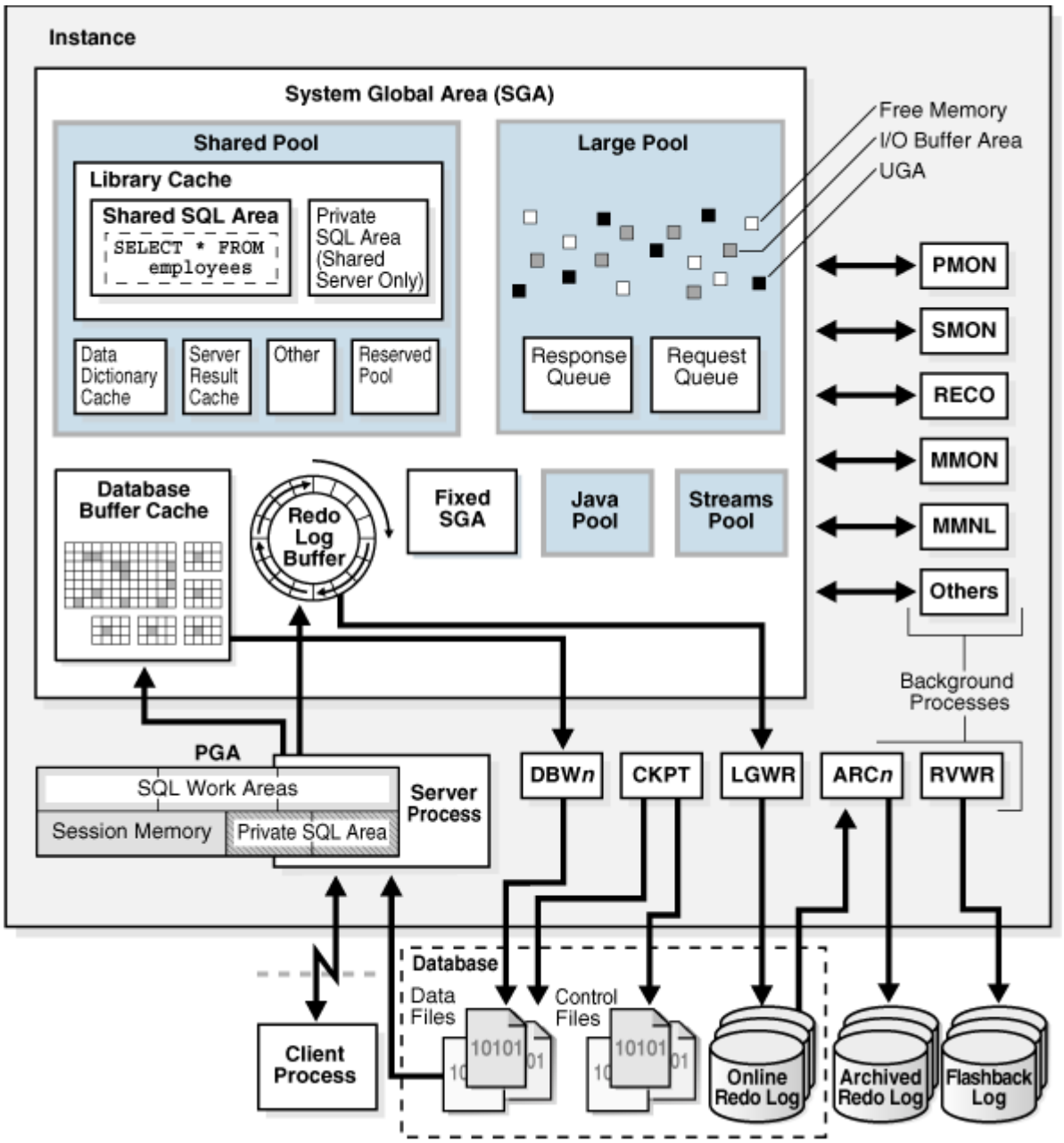
  制御ファイルは、データベースの物理コンポーネントを追跡するためのルート・ファイルです。

- オンラインREDOログ・ファイル

  オンラインREDOログは、データに対する変更が記録される一連のファイルです。

データベース・インスタンスは、データベース・ファイルを管理する一連のメモリー構造です。次の図に、インスタンスとインスタンスによって管理されるファイルの間の関係を示します。

図11-1 データベース・インスタンスとデータベース・ファイル



関連項目:

- データベースの作成方法を学習するには、『Oracle Database管理者ガイド』を参照してください
- CREATE DATABASEのセマンティクスおよび構文については、『Oracle Database SQL言語リファレンス』を参照してください

データベース・ファイル格納のメカニズム

データベース・ファイルの記憶域の割当ておよび管理には、複数のメカニズムを使用できます。

一般的なメカニズムは、次のとおりです。

- Oracle Automatic Storage Management(Oracle ASM)

  Oracle ASMには、Oracle Databaseでのみ使用できるよう設計されたファイルシステムが含まれています。

- オペレーティング・システムのファイルシステム

  ほとんどのOracleデータベースでは、ファイルをファイル・システムに格納します(ファイル・システムは、連続するディスク・ア

ドレス空間内に構築されたデータ構造です)。すべてのオペレーティング・システムには、ファイルシステム内でディスク領域のファイルへの割当ておよび割当て解除を実行するファイル・マネージャがあります。

ファイルシステムを使用すると、多数のファイルにディスク領域を割り当てることができます。各ファイルには名前があり、Oracle Databaseなどのアプリケーションに対して連続するアドレス空間として表示されます。データベースでは、ファイルの作成、読取り、書込み、サイズ変更および削除が可能です。

通常、ファイル・システムは、論理ボリューム・マネージャ(LVM)と呼ばれるソフトウェア・パッケージによって作成された論理ボリュームの最上部に構築されます。LVMを使用すると、複数の物理ディスクの断片を単一の連続するアドレス空間に結合し、ソフトウェアの上位レイヤーに対して1つのディスクとして表示できます。

- クラスタ・ファイル・システム

  クラスタ・ファイル・システムとは、連携して高パフォーマンスのサービスをクライアントに提供するサーバーのクラスタによる分散ファイル・システムです。Oracle RAC環境でクラスタ・ファイルシステムを使用すると、共有記憶域が、クラスタ化環境内の多数のコンピュータによって共有されるファイルシステムとして表示されます。クラスタ・ファイルシステムを使用すると、クラスタ内のいずれかのコンピュータで障害が発生しても、ファイルシステムが使用できなくなることはありません。ただし、オペレーティング・システムのファイルシステムで、NFSなどの方法を使用してファイルを共有するコンピュータに障害が発生した場合、ファイルシステムは使用できません。

データベースでは、これらのストレージ・メカニズムを組み合せて使用します。たとえば、データベースでは、従来のファイルシステムに制御ファイルとオンラインREDOログ・ファイルを、RAWパーティションに一部のユーザー・データファイルを、Oracle ASMに残りのデータファイルを、クラスタ・ファイルシステムにアーカイブREDOログ・ファイルを格納できます。

関連項目:

- 「Oracle Automatic Storage Management (Oracle ASM)」
- Oracle Enterprise Manager Database Express (EM Express)でデータベース記憶域の構造を表示する方法を学習するには、Oracle Database 2日でデータベース管理者を参照してください
- データベース・ビューを問い合せてデータベース記憶域構造を表示する方法の詳細は、Oracle Database管理者ガイドを参照してください

Oracle Automatic Storage Management(Oracle ASM)

Oracle ASMは、高パフォーマンスで管理しやすいOracle Databaseファイルのストレージ・ソリューションです。Oracle ASMはボリューム・マネージャであり、データベースでのみ使用できるよう設計されたファイルシステムが備えられています。

Oracle ASMには、従来のファイルシステムおよびストレージ・マネージャと比較して、次のような利点があります。

- データベースの作成およびレイアウトやディスク領域の管理など、ストレージ関連のタスクが簡略化されます。
- 複数の物理ディスクにわたってデータが配分されるため、ホットスポットがなくなり、ディスク間のパフォーマンスが均一になります。
- ストレージ構成が変わると、データのバランスが自動的に再調整されます。

Oracle ASMを使用するには、ストライプ化およびミラー化のプリファレンスを使用して、Oracle Databaseのパーティション化したディスクを割り当てます。Oracle ASMによってディスク領域が管理され、使用可能なすべてのリソースにI/O負荷が配分されるため、I/Oを手動でチューニングしなくてもパフォーマンスが最適化されます。たとえば、データベースを停止しなくても、データベースのディスクのサイズを増加し、データベースの一部を新しいデバイスに移動できます。

Oracle ASMのストレージ・コンポーネント

Oracle Databaseでは、データファイルをOracle ASMディスク・グループのOracle ASMファイルとして格納できます。ディスク・グループ内では、各データベース・ファイルに使用するファイルシステム・インタフェースがOracle ASMによって公開されます。

次の図に、Oracle ASMを使用するデータベース内のストレージ・コンポーネント間の関係を示します。このダイアグラムは、Oracle ASMファイルとデータファイルの間の関係を表していますが、Oracle ASMには他のタイプのファイルも格納できます。線の末端の分岐は、1対多の関係を表しています。

図11-2 Oracle ASMのコンポーネント

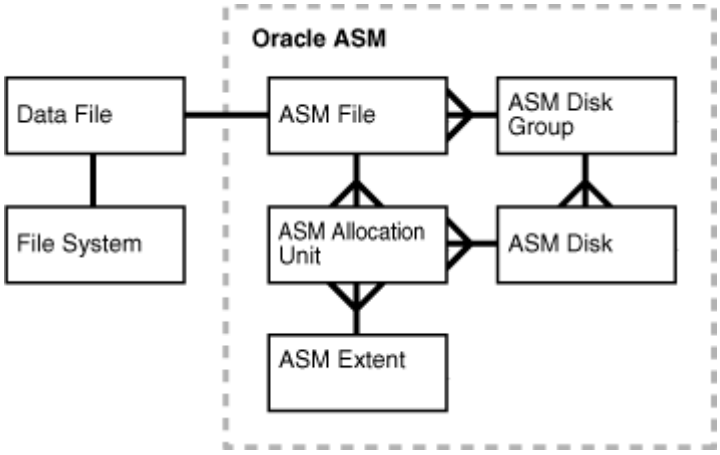


図11-2は、次のOracle ASMの概念を示しています。

- Oracle ASMディスク

  Oracle ASMディスクは、Oracle ASMディスク・グループにプロビジョニングされるストレージ・デバイスです。Oracle ASMディスクは、物理ディスクまたはパーティション、ストレージ・アレイからの論理ユニット番号(LUN)、論理ボリュームまたはネットワーク接続ファイルのいずれでもかまいません。

  Oracle ASMディスク・グループ内のOracle ASMディスクは、データベースの稼働中に追加または削除できます。ディスクをディスク・グループに追加するときは、ディスク名を割り当てます(割り当てない場合は、Oracle ASMディスク名が自動的に指定されます)。

- Oracle ASMディスク・グループ

  Oracle ASMディスク・グループは、1つの論理単位として管理されるOracle ASMディスクのコレクションです。ディスク・グループのデータ構造は自己完結型で、ディスク・グループの一部のディスク領域を使用します。

  ディスク・グループ内では、各Oracleデータベース・ファイルに使用するファイルシステム・インタフェースがOracle ASMによって公開されます。ディスク・グループに格納される各ファイルの内容は均等に配分(つまりストライプ化)されるため、ホットスポットがなくなり、ディスク間のパフォーマンスが均一化されます。

- Oracle ASMファイル

  Oracle ASMファイルは、Oracle ASMディスク・グループに格納されるファイルです。Oracle Databaseは、ファイルに関してOracle ASMと通信します。データベースには、データファイル、制御ファイル、オンラインREDOログ・ファイルなどの様々なファイルをOracle ASMファイルとして格納できます。データベースから要求されると、Oracle ASMによってOracle ASMファイルが作成され、プラス記号(+)の後にディスク・グループ名が続く名前(+DISK1など)が、このファイルに割り当てられます。

ノート:

Oracle ASM ファイルは、サード・パーティ製ファイル・システムなどの他のストレージ管理オプションと共存

させることができます。この機能により、Oracle ASM を既存の環境に統合する作業が簡素化されます。

- Oracle ASMエクステント

  Oracle ASMエクステントは、Oracle ASMファイル内のセクションです。1つのOracle ASMファイルは1つ以上のファイル・エクステントで構成されます。それぞれのOracle ASMエクステントは、特定のディスク上にある1つ以上の割当て単位で構成されます。

  ノート:

  Oracle ASM エクステントは、データをセグメントに格納するために使用されるエクステントとは異なります。

- Oracle ASM割当て単位

  Oracle ASM割当て単位は、ディスク・グループ内の割当ての基本単位です。割当て単位は、Oracle ASMによって割り当てられる最小の連続するディスク領域になります。1つ以上の割当て単位によって、1つのOracle ASMエクステントが構成されます。

関連項目:

Oracle ASMについてさらに学習するには、Oracle Automatic Storage Management管理者ガイドを参照してください

Oracle ASMインスタンス

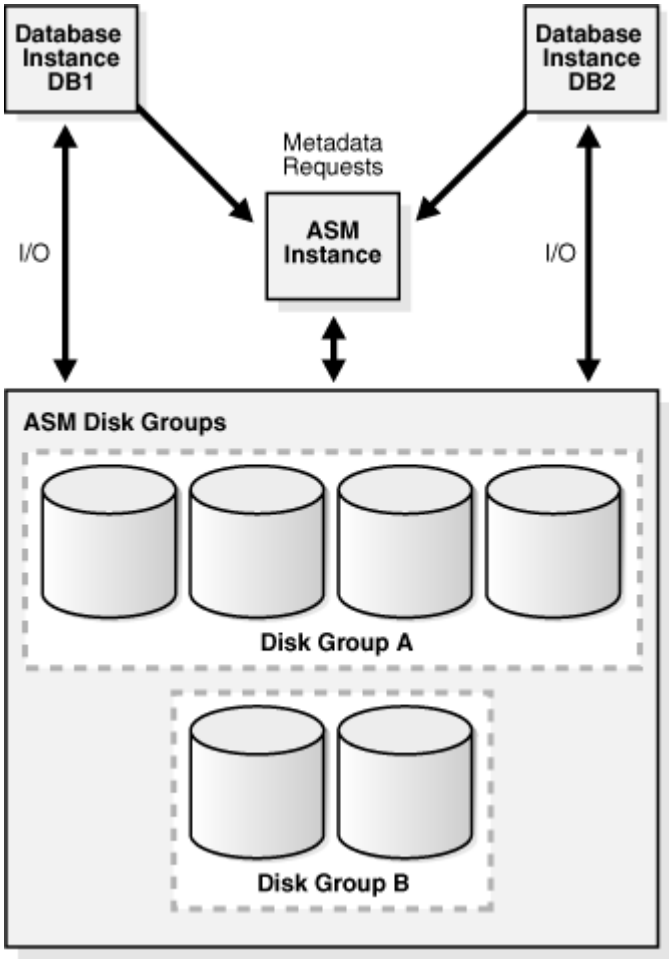
Oracle ASMインスタンスは、Oracle ASMディスクを管理する特殊なOracleインスタンスです。

Oracle ASMインスタンスとデータベース・インスタンスのどちらにも、Oracle ASMディスク・グループのディスクへの共有アクセスが必要です。Oracle ASMインスタンスはディスク・グループのメタデータを管理して、ファイルのレイアウト情報をデータベース・インスタンスに提供します。データベース・インスタンスでは、Oracle ASMインスタンスを経由せずに、I/OをOracle ASMディスクに送信します。

Oracle ASMインスタンスは、データベース・インスタンスと同じテクノロジに基づいています。たとえば、Oracle ASMインスタンスには、データベース・インスタンスと同様のシステム・グローバル領域(SGA)およびバックグラウンド・プロセスがあります。ただし、Oracle ASMインスタンスではデータベースをマウントできず、実行されるタスクもデータベース・インスタンスより少なくなります。

図11-3に、Oracle ASMインスタンスが1つとデータベース・インスタンスが2つあり、それぞれが異なる単一インスタンス・データベースに関連付けられている単一ノード構成を示します。このOracle ASMインスタンスによってメタデータが管理され、2つのデータベースのデータが格納されているOracle ASMファイルの領域が割り当てられます。一方のOracle ASMディスク・グループには4つのOracle ASMディスクがあり、もう一方には2つのディスクがあります。いずれのデータベース・インスタンスも、これらのディスク・グループにアクセスできます。

図11-3 Oracle ASMインスタンスとデータベース・インスタンス



関連項目:

Oracle ASMについてさらに学習するには、Oracle Automatic Storage Management管理者ガイドを参照してください

Oracle Managed Filesおよびユーザー管理ファイル

Oracle Managed Filesとは、ファイル名ではなく、データベース・オブジェクトの観点で操作を指定できるようにするためのファイル命名方法です。たとえば、表領域のデータファイルの名前を指定しなくても、表領域を作成できます。

Oracle Managed Filesでは、データベース内のオペレーティング・システム・ファイルを管理者が直接管理する必要がなくなります。Oracle ASMではOracle Managed Filesが必要です。

ノート:

この機能は、トレース・ファイル、監査ファイル、アラート・ログなどの管理ファイルの作成および命名には影響を与えません。

ユーザー管理ファイルを使用すると、データベース内のオペレーティング・システム・ファイルをユーザーが直接管理できます。ユーザーは、ファイルの構造および名前を決定できます。たとえば、表領域を作成する場合は、表領域データファイルの名前とパスを設定します。

初期化パラメータを使用して、特定タイプのファイルに使用するファイルシステム・ディレクトリを指定します。Oracle Managed Filesの主な役割は、データベースによって一意のファイルが作成され、そのファイルが不要になったら削除されるようにすることで

す。データベースでは、標準のファイルシステム・インタフェースを内部で使用して、データファイルと一時ファイル、制御ファイル、および高速リカバリ領域に格納されているリカバリ関連ファイルなどのファイルが作成および削除されます。

Oracle Managed Filesを使用しても、既存の機能は削除されません。以前のファイルを手動で管理しながら、新しいファイルを作成できます。このため、データベースにはOracle Managed Filesとユーザー管理ファイルが混在する場合があります。

関連項目:

- 「診断ファイルの概要」
- Oracle Managed Filesの使用方法を学習するには、『Oracle Database管理者ガイド』を参照してください。

データファイルの概要

オペレーティング・システム・レベルで、Oracle Databaseは**データファイル**と呼ばれる構造にデータベース・データを格納します。すべてのOracleデータベースには、少なくとも1つのデータファイルが必要です。

データファイルの使用

Oracle Databaseでは、表領域データがデータファイルに物理的に格納されます。

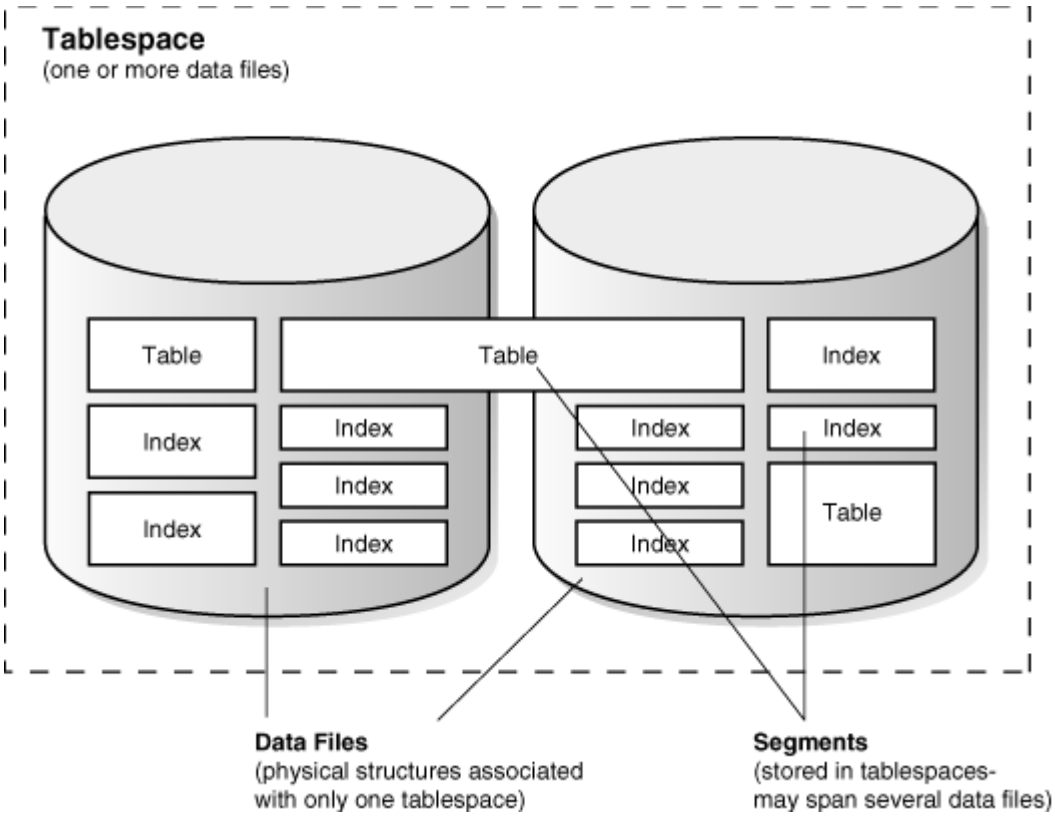
パーティション化されていない各**スキーマ・オブジェクト**とオブジェクトの各パーティションは、独自の**セグメント**(1つの表領域のみに属している)に格納されます。たとえば、パーティション化されていない表のデータは1つのセグメントに格納され、このセグメントは1つの表領域に格納されます。表領域とデータファイルは密接に関連していますが、次のような重要な違いがあります。

- 各表領域は1つ以上のデータファイルから構成され、このデータファイルはOracle Databaseを実行しているオペレーティング・システムに準拠します。
- データベースのデータは、そのデータベースの各表領域内にあるデータファイルにまとめて格納されます。
- セグメントは1つ以上のデータファイルにまたがることがありますが、複数の表領域にまたがることはありません。
- データベースには、SYSTEM表領域とSYSAUX表領域が必要です。Oracle Databaseでは、データベースの作成時に、データベースの最初のデータファイルがSYSTEM表領域に自動的に割り当てられます。

  SYSTEM表領域には、データベースのメタデータを含む一連の表であるデータ・ディクショナリが含まれています。通常、データベースにはUNDO表領域と一時表領域(通常の名前はTEMP)もあります。

次の図に、表領域、データファイルおよびセグメントの関係を示します。

図11-4 データファイルと表領域



関連項目:

- 「表領域の概要」
- データファイルの管理方法を学習するには、『Oracle Database管理者ガイド』を参照してください

永続データファイルと一時データファイル

永続表領域には、永続スキーマ・オブジェクトが格納されます。永続表領域内のオブジェクトはデータファイル内に格納されます。

一時表領域には、セッションの持続時間中のみスキーマ・オブジェクトが存在します。ローカルで管理される一時表領域には、ハッシュやソートなどの操作でデータを格納するための特殊なファイルである一時的なファイル(一時ファイル)が含まれています。一時ファイルには、メモリー内の領域が不足したときの結果セットのデータも格納されます。

一時ファイルは永続データファイルと似ていますが、次のような例外があります。

- 表などの永続データベース・オブジェクトは、一時ファイルには格納されません。
- 一時ファイルは常にNOLOGGINGモードに設定され、これは、一時ファイルにはREDOが生成されないということを意味します。メディア・リカバリは一時ファイルを認識しません。
- 一時ファイルを読取り専用にすることはできません。
- ALTER DATABASE文では一時ファイルを作成できません。
- 一時ファイルを作成またはサイズ変更する場合、指定したファイル・サイズのディスク領域が常に保証されるとはかぎりません。LinuxやUNIXなどのファイルシステムでは、一時ファイルはスパース・ファイルとして作成されます。この場合、ディスク・ブロックはファイルの作成時やサイズ変更時ではなく、最初にブロックがアクセスされるときに割り当てられます。

ノート:

スパース・ファイルを使用すると、一時ファイルの作成とサイズ変更が素早く行えますが、一時ファイルにアクセスしたときに、ディスク領域が不足する可能性があります。

- 一時ファイルの情報は、データ・ディクショナリ・ビューDBA_TEMP_FILESと動的パフォーマンス・ビューV$TEMPFILEには表示されますが、DBA_DATA_FILESビューやV$DATAFILEビューには表示されません。

関連項目:

- 「一時表領域」
- 一時ファイルの管理方法を学習するには、Oracle Database管理者ガイドを参照してください

オンライン・データファイルとオフライン・データファイル

すべてのデータファイルは、オンライン(使用可能)またはオフライン(使用不可能)のいずれかです。

個々のデータファイルまたは一時ファイルの可用性は、ファイルをオフラインにするか、オンラインにして変更できます。データベースでは、オフライン・データファイルには、それらがオンラインになるまでアクセスできません。

データファイルをオフラインにする理由は、オフライン・バックアップの実行やブロックの破損など数多くあります。データベースでは、データファイルに書き込めない場合に、ファイルが自動的にオフラインになります。

データファイルと同様に、表領域自体もオフラインまたはオンラインになります。オンラインの表領域にあるデータファイルをオフラインにしても、表領域自体はオンラインのままです。表領域自体をオフライン化することによって、表領域のすべてのデータファイルを一時的に使用禁止にできます。

Oracle Database 12c以降、ALTER DATABASE MOVE DATAFILE文を使用して、データベースがオープンした状態でファイルにアクセスしている間に、異なる物理ファイル間でオンライン・データ・ファイルを移動できます。このテクニックを使用すれば、次の目的を達成できます。

- 異なる種類のストレージ間での表領域の移動
- 頻繁にアクセスしないデータファイルを、より低コストの記憶域に移動します。
- 表領域を読取り専用にし、そのデータ・ファイルを1回のみ書込み可能なストレージ(WORM (write once read many)ドライブなど)に移動
- データベースをOracle ASMに移動

関連項目:

- 「オンライン表領域とオフライン表領域」
- データ・ファイルの可用性を変更する方法を学習するには、Oracle Database管理者ガイドを参照してください
- オンライン・データ・ファイルの移動方法を学習するには、Oracle Database管理者ガイドを参照してください
- ALTER DATABASE . . . MOVE DATAFILEについて学習するには、Oracle Database SQL言語リファレンスを参照してください

データファイルの構造

Oracle Databaseでは、表領域のデータファイルを作成する際に、指定されたディスク領域に加えて、データファイル・ヘッダーのオーバーヘッドの領域が割り当てられます。Oracle Databaseを実行するオペレーティング・システムによって、ファイルから古い情報や認可がクリアされた後、そのファイルがデータベースに割り当てられます。

データファイル・ヘッダーには、サイズやチェックポイントSCNなど、データファイルに関するメタデータが含まれています。それぞれのヘッダーには、データベース内でデータファイルを一意に識別する絶対ファイル番号と、表領域内でデータファイルを一意に識別する相対ファイル番号が含まれます。

Oracle Databaseでデータファイルが初めて作成されると、割り当てられたディスク領域がフォーマットされますが、ユーザー・データは含まれません。ただし、データベースでは、関連付けられた表領域の将来のセグメントのデータを保持する領域が確保されます。Oracle Databaseでは、表領域内のデータが増加すると、データファイル内の空き領域を使用してセグメントにエクステントを割り当てます。

次の図に、データファイル内の各種の領域を示します。エクステントは、使用中(セグメント・データが含まれている場合)または空き(再使用が可能な場合)のいずれかです。表領域内のオブジェクトの更新または削除が繰り返されると、単独では新しいデータに再使用できるほど大きくはないものの、孤立した空き領域ができる場合があります。このような空き領域は、断片化空き領域と呼ばれます。

図11-5 データファイル内の領域

関連項目:

データファイル情報の表示方法を学習するには、『Oracle Database管理者ガイド』を参照してください

制御ファイルの概要

データベースの**制御ファイル**は、1つのデータベースのみに関連付けられた小さなバイナリ・ファイルです。各データベースには一意の制御ファイルが1つあります(ただし、複数の同一コピーも使用できます)。

制御ファイルの使用

Oracle Databaseでは制御ファイルを使用して、データベース・ファイルを検索し、データベース全体の状態を管理します。

制御ファイルには、次のような情報が含まれています。

- データベース名およびデータベースの一意の識別子(DBID)
- データベースを作成したときのタイムスタンプ
- データファイル、オンラインREDOログ・ファイルおよびアーカイブREDOログ・ファイルに関する情報

- 表領域情報
- RMANバックアップ

制御ファイルは、次のような目的で使用されます。

- データベースをオープンするために必要となるデータファイル、およびオンラインREDOログ・ファイルなどに関する情報を格納します。

  制御ファイルは、データベースに対する構造上の変更を追跡します。たとえば、管理者がデータファイルやオンラインREDOログ・ファイルの追加、名前変更または削除を実行すると、データベースで制御ファイルが更新され、変更が反映されます。

- データベースがオープンしていないときにアクセスする必要があるメタデータを格納します。

  制御ファイルには、チェックポイントなどデータベースのリカバリに必要な情報などが含まれています。チェックポイントは、インスタンス・リカバリを開始する必要があるREDOストリーム内のSCNを示します。チェックポイントSCNの前にコミットされたすべての変更は、データファイルのディスクに保存されることが保証されます。チェックポイント・プロセスは、オンラインREDOログ内のチェックポイント位置に関する情報を、少なくとも3秒ごとに制御ファイルに記録します。

Oracle Databaseでは、データベースの使用中に制御ファイルの読取りと書込みが絶えず実行されるため、データベースがオープンしているときは、常に書込み可能であることが必要です。たとえば、データベースのリカバリには、データベースに含まれているすべてのデータファイルの名前を制御ファイルから読み取る必要があります。データファイルの追加など他の操作を実行すると、制御ファイルに格納された情報が更新されます。

関連項目:

- 「インスタンス・リカバリの概要」
- チェックポイント・プロセス(CKPT)
- 制御ファイルの管理方法を学習するには、Oracle Database管理者ガイドを参照してください

多重制御ファイル

Oracle Databaseでは、同一の制御ファイルを同時に複数オープンし、同じデータベースに書き込むことができます。1つの制御ファイルを複数のディスク上で多重化することにより、データベースの冗長性が実現されるため、シングル・ポイント障害を回避できます。

ノート:

多重制御ファイルのコピーをそれぞれ異なるディスクで維持することをお薦めします。

制御ファイルが使用できなくなった場合、その破損した制御ファイルにアクセスしようとするデータベース・インスタンスは失敗します。現行の制御ファイルのコピーが他にある場合は、メディア・リカバリを実行しなくても、データベースを再マウントしてオープンできます。ただし、データベースのすべての制御ファイルが失われた場合は、データベース・インスタンスが失敗し、メディア・リカバリが必要になります。現行の制御ファイルを使用できず、制御ファイルの古いバックアップを使用する場合、メディア・リカバリは容易ではありません。

関連項目:

- 多重制御ファイルを維持する方法を学習するには、Oracle Database管理者ガイドを参照してください
- 制御ファイルのバックアップおよびリストア方法を学習するには、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

制御ファイルの構造

データベースに関する情報は、制御ファイルの異なるセクションに格納されます。各セクションは、データベースの特定の側面に関するレコードの集合です。

たとえば、制御ファイルの1つのセクションではデータファイルが追跡され、レコードの集合がデータファイルごとに1つ含まれています。各セクションは、複数の論理的な制御ファイル・ブロックに格納されます。レコードは、1つのセクション内の複数のブロックにまたがる場合があります。

制御ファイルには、次のようなレコードが含まれています。

- 循環再利用レコード

  循環再利用レコードには、必要に応じて上書き可能な、重要度が低い情報が含まれます。使用可能なレコード・スロットがすべて使用されると、データベースは、制御ファイルを拡張して新規レコード用の領域を作成するか、または最も古いレコードを上書きします。たとえば、アーカイブREDOログ・ファイルやRMANバックアップのレコードなどです。

- 非循環再利用レコード

  非循環再利用レコードには、頻繁には変更されず、上書きできない重要な情報が含まれます。たとえば、表領域、データファイル、オンラインREDOログ・ファイル、REDOスレッドなどの情報です。Oracle Databaseでは、対応するオブジェクトが表領域から削除されないかぎり、これらのレコードを再使用しません。

動的パフォーマンス・ビュー(V$ビューとも呼ばれる)を問い合せて、制御ファイルに格納されている情報を表示できます。たとえば、V$DATABASEを問い合せると、データベース名およびDBIDを取得できます。ただし、制御ファイルの情報を変更できるのは、そのデータベースのみです。

制御ファイル・ブロックの読取りおよび書込みは、データ・ブロックの場合とは異なります。制御ファイルの場合、Oracle Databaseでは読取りおよび書込みがディスクからプログラム・グローバル領域(PGA)に直接行われます。各プロセスでは、そのPGAメモリーの一定量が制御ファイル・ブロックに割り当てられます。

関連項目:

- 「動的パフォーマンス・ビューの概要」
- V$CONTROLFILE_RECORD_SECTIONビューについて学習するには、Oracle Databaseリファレンスを参照してください
- CONTROL_FILE_RECORD_KEEP_TIME初期化パラメータについて学習するには、Oracle Databaseリファレンスを参照してください

オンラインREDOログの概要

リカバリにとって最も重要な構造が**オンラインREDOログ**であり、このログは、データベースの変更が発生したときにその変更が格

納される、事前に割り当てられた複数のファイルで構成されます。オンラインREDOログには、データファイルに対する変更が記録されます。

オンラインREDOログの使用

データベースでは、オンラインREDOログ・ファイルを維持してデータ損失を防ぎます。特に、インスタンス障害,の後には、オンラインREDOログ・ファイルを使用すると、Oracle Databaseではまだデータファイルに書き込まれていないコミット済のデータをリカバリできます。

複数のプロセスですべてのトランザクションを同時にREDOログ・バッファに書き込むと、LGWRにより処理され、オンラインREDOログに書き込まれます。オンラインREDOログの内容には、コミットされていないトランザクションや、スキーマおよびオブジェクトの管理文などが含まれます。

データベースでUNDOセグメントに変更を加えると、変更内容はオンラインREDOログにも書き込まれます。その結果、オンラインREDOログには常に永続オブジェクトのUNDOデータが含まれます。一時UNDOセグメントにある一時オブジェクトのすべてのUNDOデータを格納するようにデータベースを構成でき、これによって領域が節約されてパフォーマンスが向上するか、データベースで永続および一時の両方のUNDOデータを、オンラインREDOログに格納できるようになります。

Oracle Databaseでは、オンラインREDOログはリカバリにのみ使用されます。ただし、管理者はOracle LogMinerユーティリティでSQLインタフェースを使用して、オンラインREDOログ・ファイルを問い合せることができます(「Oracle LogMiner」を参照)。REDOログ・ファイルは、データベース・アクティビティに関する履歴情報の入手に役立ちます。

関連項目:

- 「インスタンス・リカバリの概要」
- 一時UNDOセグメント
- Oracleのプロセスについて学習するには、「プロセス・アーキテクチャ」を参照してください
- 一時UNDOセグメントについて学習するには、Oracle Database管理者ガイドを参照してください
- TEMP_UNDO_ENABLED初期化パラメータについて学習するには、Oracle Databaseリファレンスを参照してください

Oracle DatabaseでのオンラインREDOログの書込み方法

データベース・インスタンスのオンラインREDOログは、**REDOスレッド**と呼ばれます。

単一インスタンス構成では、1つのインスタンスしかデータベースにアクセスできないため、REDOスレッドは1つのみです。ただし、Oracle Real Application Clusters (Oracle RAC)構成では、複数のインスタンスが同時にデータベースにアクセスし、各インスタンスには独自のREDOスレッドがあります。インスタンスごとにREDOスレッドを分けると、オンラインREDOログ・ファイルの1つの集合に対する競合を回避できます。

オンラインREDOログは、複数のオンラインREDOログ・ファイルで構成されています。Oracle Databaseでは、ファイルの1つがクリア中またはアーカイブ中であっても、もう1つのファイルが常に書込み可能であることを保証するため、少なくとも2つ以上のファイルが必要です。

関連項目:

Oracle RACのオンラインREDOログ・グループについて学習するには、『Oracle Real Application Clusters管理およびデプロイメント・ガイド』を参照してください。

オンラインREDOログ・スイッチ

Oracle Databaseでは、オンラインREDOログ・ファイルを一度に1つのみ使用して、REDOログ・バッファから書き込まれるレコードを格納します。

ログ・ライター(LGWR)・プロセスによる書込みがアクティブなオンラインREDOログ・ファイルは、現行のオンラインREDOログ・ファイルと呼ばれます。

ログ・スイッチは、データベースで1つのオンラインREDOログ・ファイルへの書込みが停止し、別のファイルへの書込みが開始するときに発生します。スイッチは通常、現行のオンラインREDOログ・ファイルがいっぱいになり、さらに書込みを継続する必要があるときに発生します。ただし、ログ・スイッチは、現行のオンラインREDOログ・ファイルがいっぱいになったかどうかに関係なく、定期的にログ・スイッチが発生するように構成し、手動で強制的に実行できます。

ログ・ライターでは、オンラインREDOログ・ファイルが循環的に書き込まれます。ログ・ライター・プロセスは、使用可能な最後のオンラインREDOログ・ファイルがいっぱいになると、最初のログ・ファイルへ書き込み、循環が再び始まります。図11-6に、REDOログの循環書込みを示します。

図11-6 オンラインREDOログ・ファイルの再使用

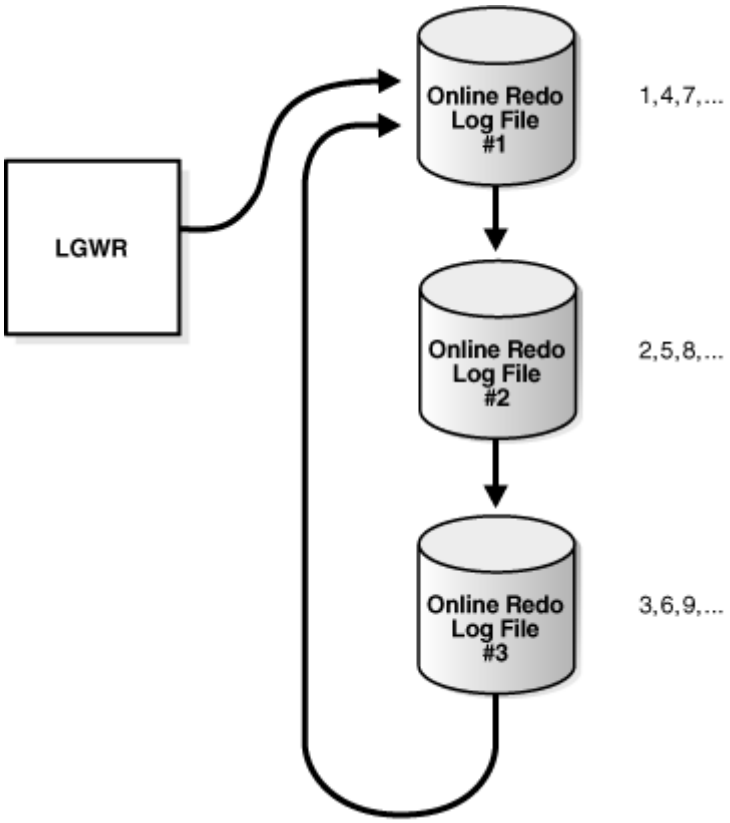


図11-6の数字は、それぞれのオンラインREDOログ・ファイルへのLGWRによる書込みの順序を示しています。データベースでは、ログ・スイッチが発生し、ログ・ライターが書込みを開始したときに、各ファイルに新しいログ順序番号を割り当てます。データベースでオンラインREDOログ・ファイルが再使用されると、このファイルには次に使用可能なログ順序番号が与えられます。

いっぱいになったオンラインREDOログ・ファイルは、アーカイブ・モードに応じて再利用できます。

- アーカイブが無効、つまりデータベースがNOARCHIVELOGモードの場合、ファイルに記録された変更のチェックポイント(書込み)処理がデータベース・ライター(DBW)によってディスクに実行されるまで、一杯になったオンラインREDOログ・ファイルを使用できません。
- アーカイブが有効、つまりデータベースがARCHIVELOGモードの場合は、変更がデータファイルに書き込まれ、そのファイルがアーカイブされた後、いっぱいになったオンラインREDOログ・ファイルをログ・ライターで使用できるようになります。

状況によっては、既存のオンラインREDOログ・ファイルをログ・ライターで再利用できない場合もあります。アクティブなオンラインREDOログ・ファイルはインスタンスのリカバリに必要で、アクティブでないオンラインREDOログ・ファイルはインスタンスのリカバリに必要ありません。また、オンラインREDOログ・ファイルがクリア・プロセスの途中である場合もあります。

関連項目:

- 「バックグラウンド・プロセスの概要」
- オンラインREDOログの管理方法を学習するには、Oracle Database管理者ガイドを参照してください

オンラインREDOログ・ファイルの複数コピー

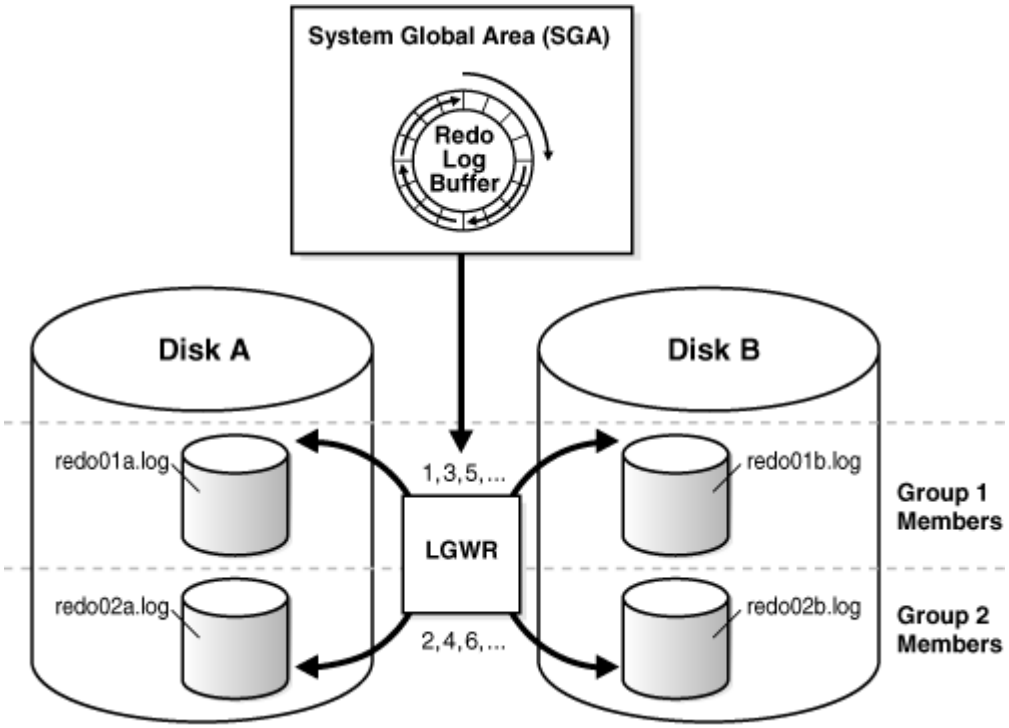
Oracle Databaseでは、オンラインREDOログの複数の同一コピーを別個の場所で自動的に維持できます。

オンラインREDOログ・グループは、1つのオンラインREDOログ・ファイルと複数の冗長コピーで構成されています。それぞれの同一コピーは、オンラインREDOログ・グループのメンバーです。各グループはグループ1、グループ2のように番号で定義されます。

オンラインREDOログ・グループの複数のメンバーを維持すると、REDOログの損失を防ぐことができます。メンバーの場所を別個のディスクに配置し、1つのディスクで障害が発生してもオンラインREDOログ全体が失われないようにすることが理想的です。

図11-7では、A_LOG1とB_LOG1はグループ1の同一のメンバーであり、A_LOG2とB_LOG2はグループ2の同一のメンバーです。同じグループの各メンバーは同じサイズである必要があります。LGWRでは、グループ1(メンバーA_LOG1とB_LOG1)への書込みが同時に実行された後、グループ2(メンバーA_LOG2とB_LOG2)への書込みが同時に実行され、次に、再びグループ1への書込みが実行されます。LGWRでは、異なるグループのメンバーに同時に書き込まれることはありません。

図11-7 オンラインREDOログ・ファイルの複数コピー



ノート:

オンライン REDO ログは、多重化することをお薦めします。ログ・ファイルの損失は、リカバリが必要な場合に壊滅的な被害を招く可能性があります。オンライン REDO ログを多重化する場合は、データベースで実行される I/O の量が増加します。システムによっては、I/O をこのように追加すると、データベース全体のパフォーマンスに影響を与える

場合があります。

関連項目:

- オンラインREDOログ・ファイルの複数コピーを維持する方法を学習するには、Oracle Database管理者ガイドを参照してください
- Data Guard構成のメンバー間のREDOデータの自動転送について学習するには、Oracle Data Guard概要および管理を参照してください

アーカイブREDOログ・ファイル

アーカイブREDOログ・ファイルとは、オンラインREDOログ・グループ内の一杯になったメンバーのコピーです。このファイルはデータベースの一部とはみなされませんが、データベースによって作成されたオンラインREDOログ・ファイルのオフライン・コピーであり、ユーザーが指定した場所に書き込まれます。

アーカイブREDOログ・ファイルは、バックアップおよびリカバリ計画にとって重要です。アーカイブREDOログ・ファイルは、次の目的に使用できます。

- データベース・バックアップのリカバリ
- スタンバイ・データベースの更新
- Oracle LogMinerユーティリティを使用したデータベースの履歴情報の取得

アーカイブREDOログ・ファイルを生成する操作をアーカイブといいます。この操作は自動または手動のいずれかで実行できます。これは、データベースがARCHIVELOGモードの場合にのみ実行できます。

アーカイブREDOログ・ファイルには、オンラインREDOログ・グループの同一メンバーのREDOエントリ、およびログ順序番号が含まれています。「オンラインREDOログ・ファイルの複数コピー」では、ファイルA_LOG1とB_LOG1がグループ1の同一メンバーです。データベースがARCHIVELOGモードで、自動アーカイブが有効になっている場合は、アーカイバ・プロセス(ARCn)によってこれらのファイルの1つがアーカイブされます。A_LOG1が破損している場合は、B_LOG1をアーカイブできます。アーカイブREDOログには、アーカイブを有効にしたときから存在したすべてのREDOログ・グループのコピーが含まれます。

関連項目:

- Oracle LogMiner
- 「データファイル・リカバリ」
- アーカイブREDOログの管理方法を学習するには、Oracle Database管理者ガイドを参照してください
- スタンバイREDOログ・アーカイバルの構成方法を学習するには、Oracle Data Guard概要および管理を参照してください

オンラインREDOログの構造

オンラインREDOログ・ファイルには、REDOレコードが含まれています。

REDOレコードは、変更ベクトルのグループで構成され、それぞれの変更ベクトルは、データ・ブロックに対する変更を表します。た

とえば、employees表の給与を更新すると、この表のデータ・セグメント・ブロック、UNDOセグメント・データ・ブロックおよびUNDOセグメントのトランザクション表に対する変更を記述するREDOレコードが生成されます。

REDOレコードには、変更に関係するすべてのメタデータが設定されます。たとえば、次のものがあります。

- 変更のSCNおよびタイムスタンプ
- 変更が生成されたトランザクションのトランザクションID
- トランザクションがコミットされたときのSCNおよびタイムスタンプ(トランザクションがコミットされた場合)
- 変更を実行した操作のタイプ
- 変更されたデータ・セグメントの名前およびタイプ

関連項目:

「データ・ブロックの概要」

12 論理記憶域構造

この章では、論理記憶域構造の性質と論理記憶域構造間の関係について説明します。これらの構造は、Oracle Databaseによって作成されて認識されるものであり、オペレーティング・システムでは認識されません。

この章の構成は、次のとおりです。

- 論理記憶域構造の概要
- データ・ブロックの概要
- エクステントの概要
- セグメントの概要
- 表領域の概要

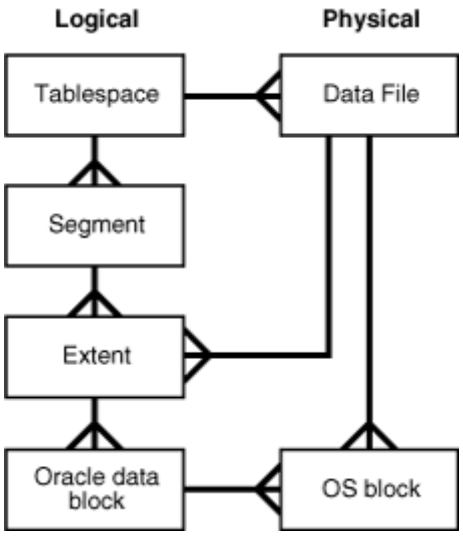
論理記憶域構造の概要

Oracle Databaseでは、データベース内のすべてのデータに対して論理領域が割り当てられます。

データベース領域の論理単位は、データ・ブロック、エクステント、セグメントおよび表領域です。物理レベルでは、データはディスク上のデータ・ファイル内に格納されます。データファイル内のデータは、オペレーティング・システムのブロックに格納されます。

次の図に、物理記憶域と論理記憶域のエンティティ関係ダイアグラムを示します。線の末端の分岐は、1対多の関係を表しています。

図12-1 論理記憶域と物理記憶域



関連項目:

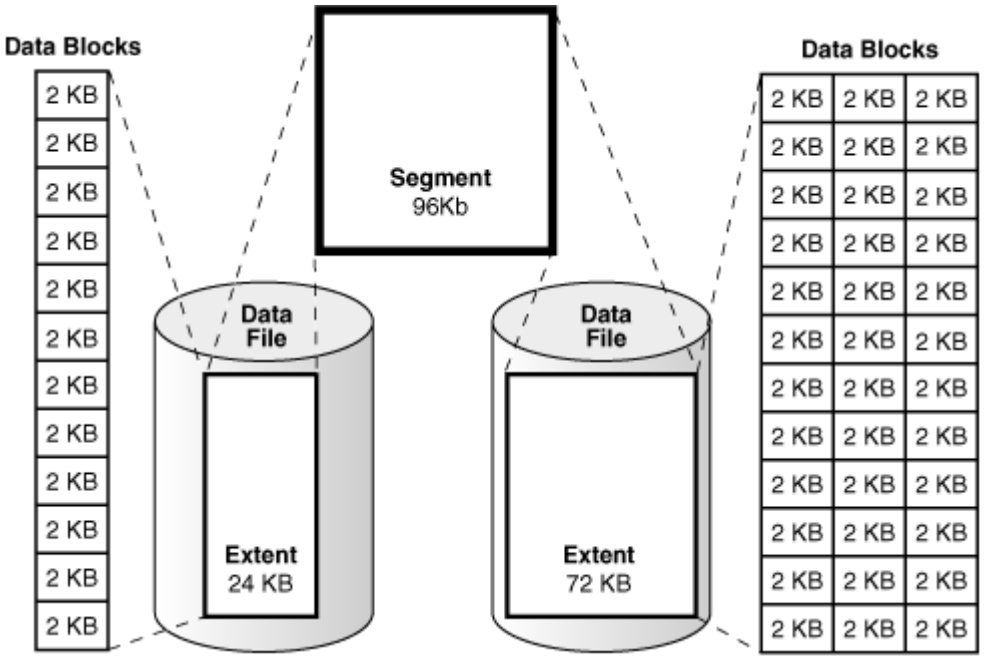
物理記憶域構造

論理記憶域階層

セグメントには、それぞれが複数のデータ・ブロックを含む1つ以上のエクステントが含まれます。

次の図に、表領域内のデータ・ブロック、エクステントおよびセグメント間の関係を示します。この例では、1つのセグメントに対して、異なるデータファイルに格納された2つのエクステントがあります。

図12-2 表領域内のセグメント、エクステントおよびデータ・ブロック



最低レベルから最高レベルの粒度まで、Oracle Databaseはデータを格納します

- データ・ブロックは、Oracle Databaseのデータ記憶域の最小論理単位です。

  1つの論理データ・ブロックは、一定のバイト数(たとえば2KB)の物理ディスク領域に対応します。データ・ブロックは、Oracle Databaseで使用または割当て可能な記憶域の最小単位です。

- エクステントとは、論理的に連続する一連のデータ・ブロックで、特定のタイプの情報を格納するために割り当てられます

  前の図では、24KBのエクステントには12のデータ・ブロックが存在し、72KBのエクステントには36のデータ・ブロックが存在しています。

- セグメントとは、表などの特定のデータベース・オブジェクトのために割り当てられるエクステントの集合です。

  たとえば、employees表のデータはそれぞれ専用のデータ・セグメントに格納されており、employeesの各索引はそれぞれ専用の索引セグメントに格納されています。記憶域を使用する各データベース・オブジェクトは、1つのセグメントを構成します。

- 表領域は、1つ以上のセグメントが含まれるデータベース記憶域単位です。

  各セグメントは、1つの表領域にのみ属しています。このため、1つのセグメントのすべてのエクステントは、同じ表領域に格納されます。前の図に示すように、表領域内のセグメントには、複数のデータファイルのエクステントが含まれている場合があります。たとえば、セグメントのあるエクステントはusers01.dbfに格納され、他のセグメントはusers02.dbfに格納されます。1つのエクステントが複数のデータファイルにまたがることはありません。

関連項目:

「データファイルの概要」

論理領域管理

Oracle Databaseでは、表領域内でエクステントの追跡および割当てを行うために論理領域管理を使用する必要があります。

データベース・オブジェクトでエクステントが必要な場合、データベースには、エクステントを検索および提供する手段が必要です。

同様に、オブジェクトでエクステントが不要になった場合、データベースに空きエクステントを使用可能にする手段が必要です。

Oracle Databaseでは、作成する表領域のタイプに基づいて、表領域内の領域が管理されます。次のいずれかのタイプの表領域を作成できます。

- ローカル管理表領域(デフォルト)

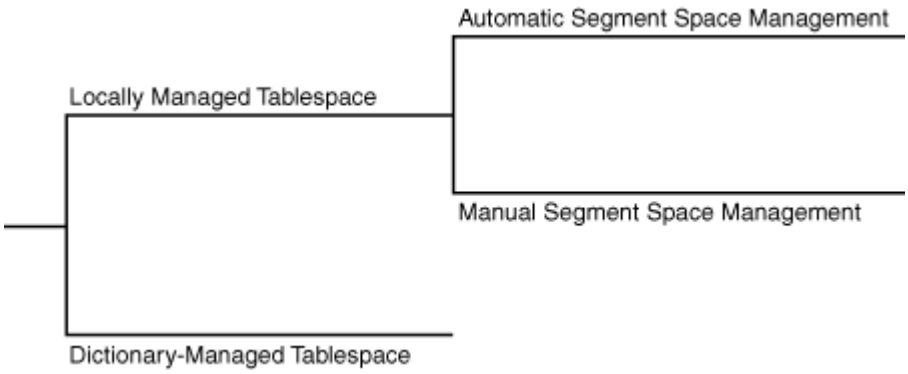
  エクステントの管理に、表領域自体にあるビットマップが使用されます。したがって、ローカル管理表領域の一部には、ビットマップ用の表領域があります。表領域内のセグメントは、自動セグメント領域管理(ASSM)または手動セグメント領域管理(MSSM)を使用して管理できます。

- ディクショナリ管理表領域

  エクステントの管理に、データ・ディクショナリが使用されます。

図12-3に、表領域の論理領域管理の方法を示します。

図12-3 論理領域管理



関連項目:

「データ・ディクショナリの概要」

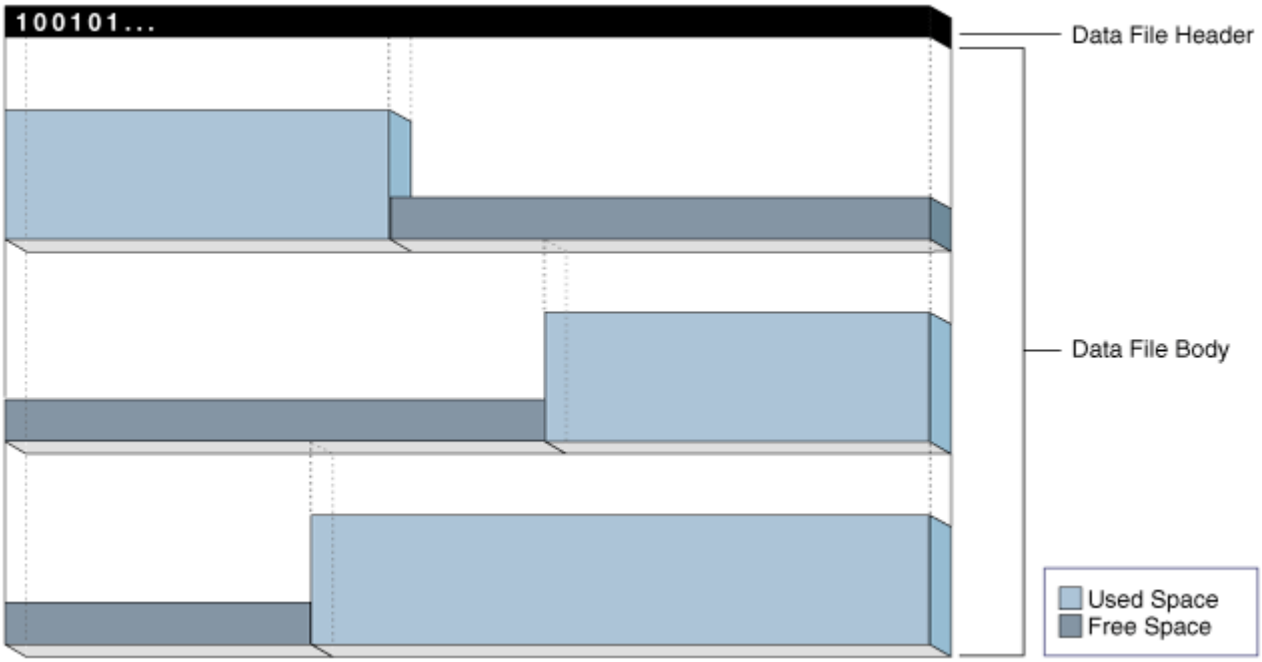
ローカル管理表領域

ローカル管理表領域では、データファイル・ヘッダー内に、データファイル本体の空き領域および使用済領域を追跡するためのビットマップが保持されています。

各ビットは、1ブロック・グループに対応します。領域が割り当てられるか、解放されると、Oracle Databaseでは、ビットマップの値がブロックの新しい状態を反映する値に変更されます。

ビットマップ管理記憶域を概念的に表現すると、次の図のようになります。ヘッダー内の1は使用済領域を表し、0は空き領域を表します。

図12-4 ビットマップ管理記憶域



ローカル管理表領域には、次のような利点があります。

- エクステントの管理にデータ・ディクショナリを使用する必要がありません。

  ディクショナリ管理表領域では、エクステント内の領域を使用または解放することにより、データ・ディクショナリ表やUNDOセグメントの領域を使用または解放する他の操作が行われた場合、再帰的な操作が発生することがあります。

- 隣接する空き領域が自動的に追跡されます。

  これにより、データベースで空きエクステントを結合する必要がなくなります。

- ローカル管理エクステントのサイズが自動的に決定されます。

  また、ローカル管理表領域内では、すべてのエクステントを同じサイズにすることもでき、さらにオブジェクト記憶域オプションがオーバーライドされます。

ノート:

ローカル管理表領域と自動セグメント領域管理を使用することをお薦めします。

セグメント領域管理は、セグメントを含む表領域から継承される属性です。ローカル管理表領域内では、セグメントは自動または手動で管理できます。たとえば、表領域users内はセグメントを自動で、表領域tools内のセグメントは手動で管理できます。

自動セグメント領域管理

自動セグメント領域管理(ASSM)方式では、ビットマップを使用して表領域の領域が管理されます。

ビットマップには、次の利点があります。

- 管理の簡素化

  ASSMでは、多数の記憶域パラメータの設定を手動で厳密に決定する必要がありません。唯一の重要なSQLパラメータPCTFREEによって、領域割当てが制御されます。このパラメータは、将来の更新用にブロック内に予約する領域の割合を指定します(「データ・ブロック内の空き領域の割合」を参照)。

- 同時実行性の向上

複数のトランザクションで別々の空きデータ・ブロックのリストを検索できるため、競合や待機時間が減少します。多数の標準的なワークロードでは、適切にチューニングされた、MSSMを使用するアプリケーションよりも、ASSMを使用するアプリケーションの方が高いパフォーマンスを示します。

- Oracle Real Application Clusters(Oracle RAC)環境における、インスタンスへの領域の動的アフィニティ

ASSMは永続的なローカル管理表領域に対するデフォルトであり、より効率的です。

ノート:

この章では、論理記憶域のすべての説明において、ASSM を使用することを前提としています。

手動セグメント領域管理

従来の**手動セグメント領域管理(MSSM)**方式では、**空きリスト**と呼ばれるリンク・リストを使用して、セグメントの空き領域が管理されます。

空き領域が存在するデータベース・オブジェクトでは、空きリストによって、使用済のセグメント領域と未使用のセグメント領域を分割するラインである最高水位標(HWM)より下のブロックが追跡されます。ブロックが使用済になると、データベースは必要に応じて、空きリストにブロックを登録したり、空きリストからブロックを削除したりします。

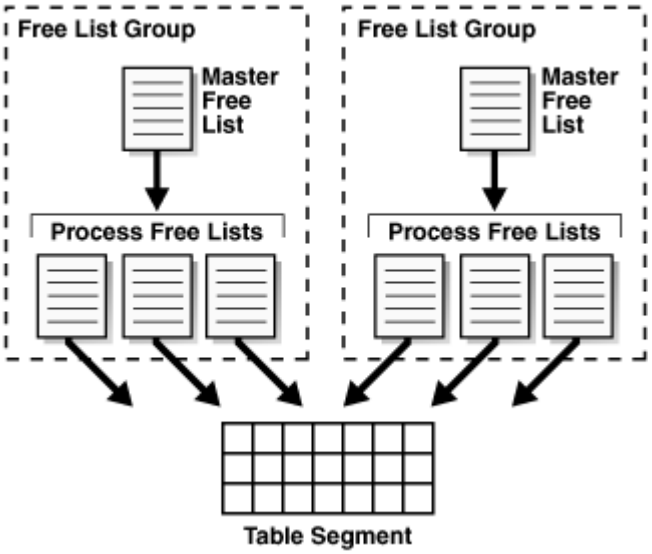
MSSMでは、PCTFREE以外にもPCTUSED、FREELISTS、FREELIST GROUPSなどのSQLパラメータを使用して、領域割当てを制御する必要があります。PCTUSEDでは、現在使用されているブロックが空きリストに登録されるためには、どの程度の空き領域が必要なのかの割合を設定します。たとえば、CREATE TABLE文でPCTUSEDを40に設定すると、ブロックの使用済領域の割合が40%未満になるまでは、セグメント内のブロックに行を挿入できません。

たとえば、表に行を挿入するとします。データベースは表の空きリストを参照し、最初の空き領域がどこなのかをチェックします。行がブロックに格納し切れず、ブロック内の使用済領域がPCTUSED以上になっている場合、そのブロックはリストから削除され、別のブロックが検索されます。ブロックから行を削除した場合は、ブロック内の使用済領域がPCTUSED未満になったかどうかがチェックされます。その場合は、そのブロックが空きリストの先頭に登録されます。

1つのオブジェクトに対して、空きリストを複数にすることもできます。こうしておくと、表に対してDMLを実行している複数のセッションが別々のリストを使用できるため、競合を回避できます。各データベース・セッションがセッション中に使用する空きリストは1つです。

図12-5に示すように、1つ以上の空きリスト・グループ(空きリストの集合)が存在するオブジェクトを1つ作成することもできます。各グループにはマスター空きリストがあり、グループ内の各プロセス空きリストを管理します。空きリストに必要な領域は、特に空きリスト・グループの場合、大きくなることがあります。

図12-5 空きリスト・グループ



セグメント領域を手動で管理する手順は複雑です。行の移行を少なくし、領域を無駄にしないようにするには、PCTFREEおよびPCTUSEDを調整する必要があります。たとえば、セグメント内のどの使用済ブロックも半分の領域が一杯になっており、PCTUSEDが40になっている場合は、これらのブロックのいずれにも挿入できません。領域割当てパラメータの詳細なチューニングは難しいため、ASSMの使用をお薦めします。ASSMでは、ブロックに新しい行を挿入可能かどうかを判断するためにPCTFREEが使用されますが、空きリストは使用されず、PCTUSEDは無視されます。

関連項目:

- 「行の連鎖と移行」
- ローカル管理表領域について学習するには、『Oracle Database管理者ガイド』を参照してください。
- 自動セグメント領域管理についてさらに学習するには、『Oracle Database管理者ガイド』を参照してください。
- PCTFREEやPCTUSEDなどの記憶域パラメータについて学習するには、『Oracle Database SQL言語リファレンス』を参照してください。

ディクショナリ管理表領域

ディクショナリ管理表領域では、データ・ディクショナリを使用してそのエクステントが管理されます。

Oracle Databaseでは、エクステントが割り当てられたり、再利用のために解放されるたびに、データ・ディクショナリ内の表が更新されます。たとえば、表でエクステントが必要な場合は、データ・ディクショナリ表が問い合せられ、空きエクステントが検索されます。領域が見つかると、あるデータ・ディクショナリ表が変更され、別のデータ・ディクショナリ表に行が挿入されます。このようにデータを変更および移動することによって、領域が管理されます。

データベース・オブジェクト用の領域を取得するため、データベースでバックグラウンドで実行されるSQLは、再帰的SQLです。データ・ディクショナリへの更新はシリアライズされる必要があるため、再帰的SQLを頻繁に使用するとパフォーマンスに悪影響を与える可能性があります。デフォルトのローカル管理表領域では、このようなパフォーマンス上の問題は回避されます。

関連項目:

ディクショナリ管理表領域からローカル管理表領域への移行方法を学習するには、『Oracle Database管理者ガイド』を参照してください

データ・ブロックの概要

Oracle Databaseでは、データベースのデータファイル内の論理記憶域は、**データ・ブロック**と呼ばれる単位で管理されます(データ・ブロックは、*Oracle*ブロックまたはページとも呼ばれます)。データ・ブロックは、データベースI/Oの最小単位です。

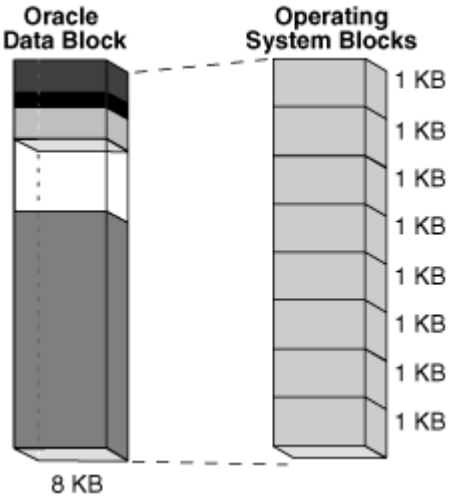
データ・ブロックとオペレーティング・システム・ブロック

物理レベルでは、データベースのデータは、オペレーティング・システム・ブロックによって構成されているディスク・ファイル内に格納されます。

オペレーティング・システム・ブロックは、オペレーティング・システムが読取りまたは書込みできるデータの最小単位です。一方、Oracleブロックは、論理記憶域構造であり、そのサイズや構造はオペレーティング・システムには認識されません。

次の図に、オペレーティング・システム・ブロックのサイズがデータ・ブロックのサイズと異なる場合があることを示します。データベースでは、オペレーティング・システム・ブロックではなく、データ・ブロックの倍数を単位としてデータを要求します。

図12-6 データ・ブロックとオペレーティング・システム・ブロック



データベースがデータ・ブロックを要求すると、オペレーティング・システムは、この操作を永続ストレージ内のデータに対する要求に変換します。データ・ブロックをオペレーティング・システム・ブロックから論理的に分離することは、次のことを意味します。

- アプリケーションでは、データのディスク上の物理アドレスを特定する必要がありません。
- データベースのデータは、複数の物理ディスクにストライプ化またはミラー化できます。

データベース・ブロック・サイズ

すべてのデータベースには、データベース・ブロック・サイズがあります。

DB_BLOCK_SIZE初期化パラメータでは、データベース作成時のデータ・ブロック・サイズが設定されます。このサイズは、SYSTEM表領域およびSYSAUX表領域に対して設定され、他のすべての表領域のデフォルトとなります。データベース・ブロック・サイズは、データベースを再作成する場合以外は変更できません。

DB_BLOCK_SIZEが設定されていない場合、デフォルトのデータ・ブロック・サイズは、オペレーティング・システム固有のサイズになります。データベースの標準的なデータ・ブロック・サイズは、4KBまたは8KBです。データ・ブロックのサイズとオペレーティング・システム・ブロックのサイズが異なる場合、データ・ブロックのサイズは、オペレーティング・システム・ブロックのサイズの倍数である必要があります。

関連項目:

- DB_BLOCK_SIZE初期化パラメータについて学習するには、『Oracle Databaseリファレンス』を参照してください

- ブロック・サイズの選択方法を学習するには、『Oracle Database管理者ガイド』および『Oracle Databaseパフォーマンス・チューニング・ガイド』を参照してください。

表領域ブロック・サイズ

DB_BLOCK_SIZEの設定とは異なるブロック・サイズの表領域を個別に作成できます。

非標準のブロック・サイズは、トランスポータブル表領域を別のプラットフォームに移動する場合に役立ちます。

関連項目:

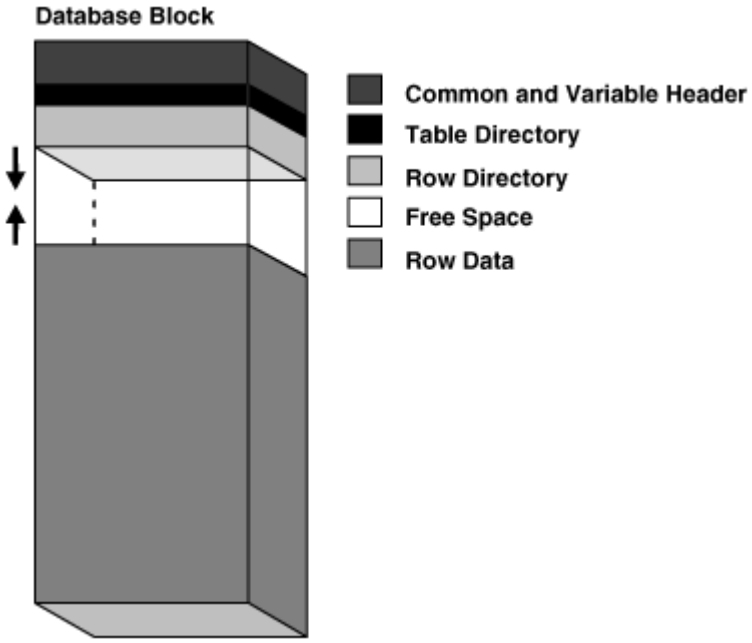
表領域に非標準のブロック・サイズを指定する方法を学習するには、『Oracle Database管理者ガイド』を参照してください

データ・ブロックの形式

すべてのデータ・ブロックには、形式、つまりデータベースでのブロック内のデータや空き領域の追跡を可能にする内部構造があります。この形式は、データ・ブロックに表、索引、表クラスタのいずれのデータが含まれている場合でも同じです。

次の図に、非圧縮データ・ブロックの形式を示します。

図12-7 データ・ブロックの形式



関連項目:

圧縮ブロックについて学習するには、「データ・ブロックの圧縮」を参照してください

データ・ブロック・オーバーヘッド

Oracle Databaseでは、**ブロック・オーバーヘッド**を使用してブロック自体を管理します。ブロック・オーバーヘッドには、ユーザー・データは格納できません。

「データ・ブロックの形式」に示すように、ブロック・オーバーヘッドには次の構成要素が含まれています。

- ブロック・ヘッダー

  ブロック・ヘッダーには、ディスク・アドレスやセグメント・タイプなどブロックについての一般的な情報が含まれています。トランザクション管理ブロックの場合、ブロック・ヘッダーにはアクティブなトランザクション情報とトランザクション履歴情報が含まれています。

  ブロックを更新する各トランザクションには、トランザクション・エントリが必要です。Oracle Databaseでは、トランザクション・エントリ用の領域が最初にブロック・ヘッダーに予約されます。トランザクションによる変更をサポートするセグメントに割り当てられたデータ・ブロックでは、ヘッダー領域の空きがなくなると、トランザクション・エントリが空き領域にも保持されます。トランザクション・エントリに必要な領域のサイズは、オペレーティング・システムによって異なります。ただし、ほとんどのオペレーティング・システムで、トランザクション・エントリのサイズは約23バイトです。

- 表ディレクトリ

  ヒープ構成表では、このディレクトリには、該当するブロックに行を格納する表についてのメタデータが含まれています。表クラスタ内では、複数の表が同じブロック内に行を格納できます。

- 行ディレクトリ

  ヒープ構成表では、このディレクトリはブロックのデータ部分内の行の位置を示しています。データベースでは、ブロックの下部の任意の場所に行を配置できます。行アドレスは、行ディレクトリ・ベクターのいずれかのスロットに記録されます。

  ROWIDは、特定のファイル、ブロックおよび行番号を指します。たとえば、ROWID AAAPecAAFAAAABSAAAでは、最後のAAAは行番号を表します。行番号は、行ディレクトリのエントリへの索引です。行ディレクトリ・エントリには、データ・ブロックの行の位置のポインタが含まれます。データベースがブロック内で移動すると、データベースは行ディレクトリ・エントリを更新してポインタを変更します。ROWIDは一定です。

  データベースが行ディレクトリに領域を割り当てると、行が削除された後もデータベースはこの領域を再利用しません。したがって、現在は空き状態でも、以前に最大50行が格納されていた場合は、行ディレクトリには100バイトが割り当てられたままになります。データベースがこの領域を再利用するのは、セッションが新しい行をブロックに挿入する場合のみです。

一部のブロック・オーバーヘッドはサイズが固定ですが、全体のサイズは可変です。ブロック・オーバーヘッド全体のサイズは、平均で84から107バイトです。

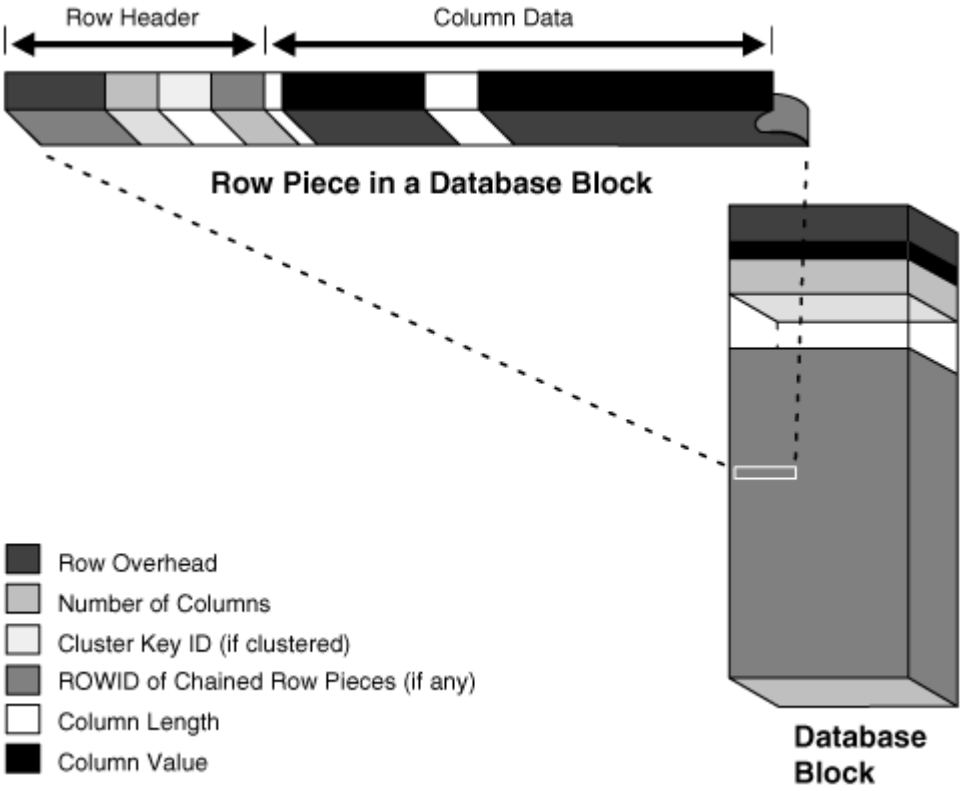
行形式

ブロックの行データ部分には、表の行や索引キー・エントリなどの実際のデータが格納されます。すべてのデータ・ブロックに内部形式があるように、すべての行に行形式があり、これにより、データベースにおける行のデータ追跡が可能になります。

Oracle Databaseでは、行は可変長レコードとして格納されます。1つの行は1つ以上のセクションに格納されます。各セクションは、行断片と呼ばれます。各行断片は、行ヘッダーおよび列データで構成されます。

次の図に、行の形式を示します。

図12-8 行断片の形式



行ヘッダー

Oracle Databaseでは、行ヘッダーを使用して、ブロック内に格納された行断片を管理します。

行ヘッダーには、次のような情報が含まれています。

- 行断片内の列
- 他のデータ・ブロック内にある行の断片

  Oracle Databaseでは、行全体を単一のデータ・ブロックに挿入できる場合、その行は1つの行断片として格納されます。ただし、行データのすべてを単一のブロックに挿入できない場合、または行の更新によって既存の行が単一のブロックに収まらなくなった場合には、その行は複数の行断片として格納されます。データ・ブロックには、通常1つの行につき1つの行断片のみが含まれています。
- 表クラスタのクラスタ・キー

1つのブロックに完全に収まる行は、最低3バイトの行ヘッダーを持っています。

関連項目:

- 「行の連鎖と移行」
- 「表クラスタの概要」

列データ

行ヘッダーの後ろにある列データ・セクションに、行の実際のデータが格納されます。行断片には通常、CREATE TABLE文で指定された順序で列が格納されますが、必ずこの順序であるとはかぎりません。たとえば、LONG型の列は最後に作成されます。

Oracle Databaseでは、「行形式」の図に示すように、行断片内の各列の長さとデータが別々に格納されます。必要な領域のサイズは、データ型によって異なります。列のデータ型が可変長である場合、値を保持するために必要な領域は、データの更新によって変動します。

各行には、データ・ブロック・ヘッダーの行ディレクトリにスロットがあります。このスロットは行の先頭を指しています。

関連項目:

「表の記憶域」および「索引記憶域」

ROWID形式

Oracle Databaseでは、**ROWID**を使用して行が一意に識別されます。内部的には、ROWIDは、データベースが行にアクセスするために必要な情報を保持する構造です。ROWIDは、物理的にはデータベースに格納されませんが、データが格納されているファイルやブロックから推論されます。

拡張ROWIDには、データ・オブジェクト番号が含まれています。このROWIDタイプは、各行の物理アドレスをBASE64エンコーディングで表現したものです。エンコーディング文字は、AからZ、aからz、0から9、+および/です。

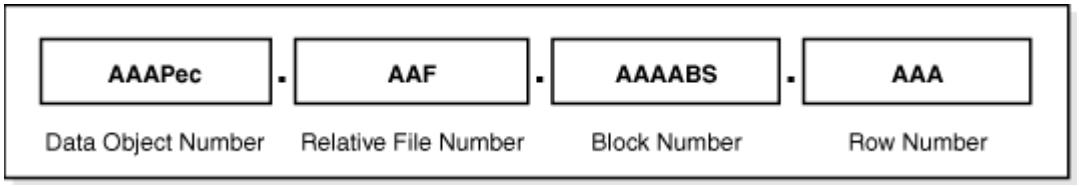
例12-1 ROWID擬似列

次の例では、ROWID擬似列を問い合せて、employees表にある従業員100の行の拡張ROWIDを表示します。

```
SQL> SELECT ROWID FROM employees WHERE employee_id = 100;

ROWID
------------------
AAAPecAAFAAAABSAAA
```

次の図に、拡張ROWIDの形式を示します。

図12-9 ROWID形式



拡張ROWIDは、000000FFFBBBBBBRRRという4つの部分から構成される形式で表示されます。各部分の内容は、次のとおりです。

- 000000

  セグメントを識別するデータ・オブジェクト番号(問合せの例では、データ・オブジェクトAAAPecです)。データ・オブジェクト番号は、各データベース・セグメントに割り当てられます。同じセグメント内のスキーマ・オブジェクト(表クラスタなど)は、同じデータ・オブジェクト番号を持っています。

- FFF

  行を含むデータファイルを識別する表領域関連のデータファイル番号(問合せの例では、ファイルAAFです)。

- BBBBBB

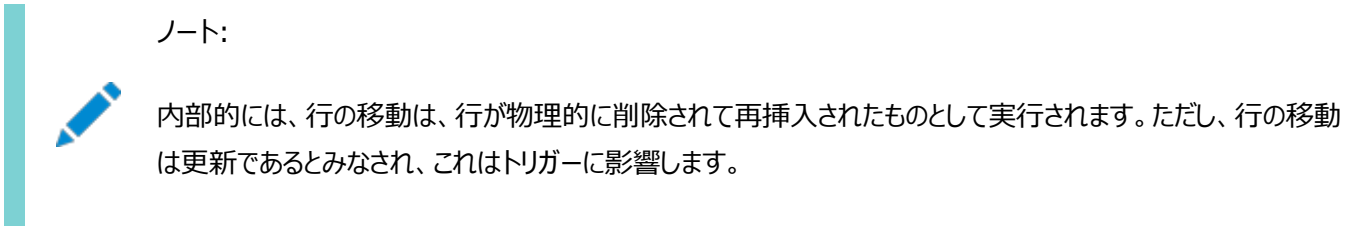
  行を含むブロックを識別するデータ・ブロック番号(問合せの例では、ブロックAAAABSです)。ブロック番号は、表領域ではなくデータファイルに対する相対値です。このため、同じブロック番号の行が、同じ表領域の異なるデータファイルに存在することもあります。

- RRR

  ブロック内の行を識別する行番号(問合せの例では、行AAAです)。

特殊な状況では、ROWIDが行断片に割り当てられた後、ROWIDが変更される場合があります。たとえば、行の移動が可能

な場合、ROWIDは、パーティション・キーの更新、フラッシュバック表の操作、表の縮小操作などのために変更される可能性があります。行の移動が無効な場合は、Oracle Databaseユーティリティを使用して行をエクスポートおよびインポートした場合にROWIDが変更される可能性があります。



ノート:

内部的には、行の移動は、行が物理的に削除されて再挿入されたものとして実行されます。ただし、行の移動は更新であるとみなされ、これはトリガーに影響します。

関連項目:

- 「ROWIDデータ型」
- ROWIDについて学習するには、Oracle Database SQL言語リファレンスを参照してください

データ・ブロックの圧縮

データベースでは、**表の圧縮**を使用して、データ・ブロック内の重複値を排除します。この項では、圧縮を使用するデータ・ブロックの形式について説明します。

基本表および高度な行圧縮を使用するデータ・ブロックの形式は、基本的には非圧縮ブロックの形式と同じです。相違点は、ブロックの先頭にあるシンボル表に、行および列の重複値が格納されることです。Oracle Databaseでは、発生した重複値はシンボル表の短い参照で置き換えられます。

例12-2 圧縮されたデータ・ブロックの形式

次の行が、7列で構成されたsales表のデータ・ブロックに格納されるとします。

```
2190,13770,25-NOV-00,S,9999,23,161
2225,15720,28-NOV-00,S,9999,25,1450
34005,120760,29-NOV-00,P,9999,44,2376
9425,4750,29-NOV-00,I,9999,11,979
1675,46750,29-NOV-00,S,9999,19,1121
```

この表に基本表または高度な行圧縮が適用されると、重複値がシンボル参照に置き換えられます。次の圧縮の概念的な表現では、29-NOV-00がシンボル*、9999が%に置き換えられています。

```
2190,13770,25-NOV-00,S,%,23,161
2225,15720,28-NOV-00,S,%,25,1450
34005,120760,*,P,%,44,2376
9425,4750,*,I,%,11,979
1675,46750,*,S,%,19,1121
```

表12-1は、シンボルを値にマップするシンボル表の概念を示します。

表12-1 シンボル表

シンボル	値	列	行
*	29-NOV-00	3	958-960

シンボル	値	列	行
%	9999	5	956-960

関連項目:

「表圧縮」

データ・ブロック内の領域管理

Oracle Databaseでは、データ・ブロックは下から順に使用されるため、行データとブロック・ヘッダー間の空き領域は減少します。

データ・ブロック内の空き領域は、行の末尾のNULL値がNULL以外の値に変更されるときなどの更新の際にも減少します。Oracle Databaseでは、パフォーマンスを最適化し、領域を無駄にしないようにするために、データ・ブロック内の空き領域が管理されます。

ノート:

この項では、自動セグメント領域管理の使用を前提としています。

データ・ブロック内の空き領域の割合

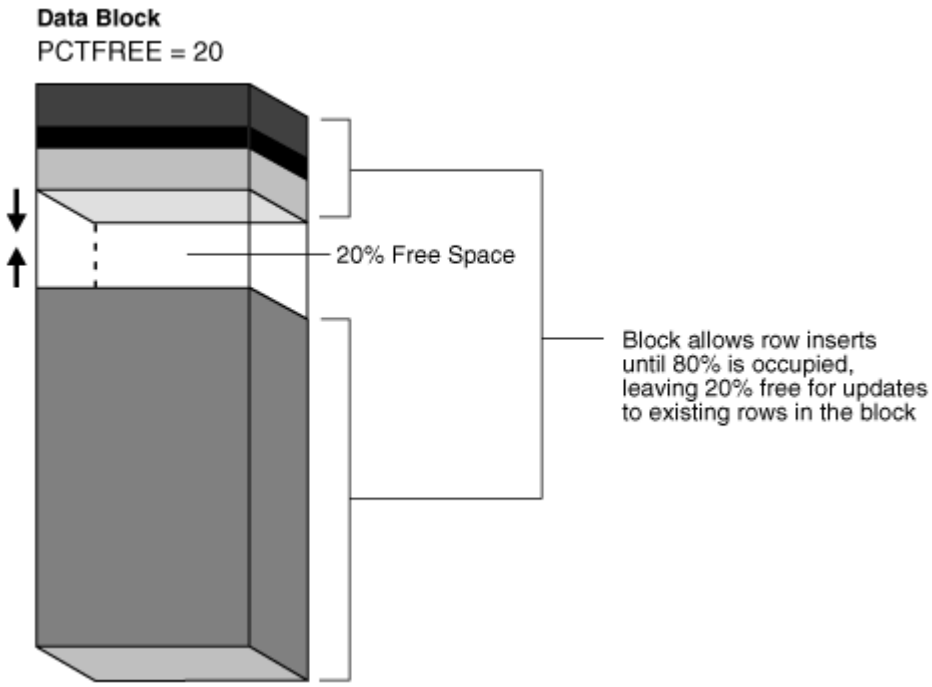
PCTFREE SQLパラメータでは、既存の行の更新用に空き領域として予約されるデータ・ブロックの最小の割合を設定します。行の移行を防止し、領域を無駄にしないようにするためにPCTFREEが必要です。

たとえば、ほとんど更新されず、更新される場合でも既存のデータのサイズがほとんど増加しない表を作成するとします。CREATE TABLE文内のPCTFREEパラメータを次のように指定します。

```
CREATE TABLE test_table (n NUMBER) PCTFREE 20;
```

図12-10に、PCTFREEを20に設定した場合の領域管理への影響について示します。時間の経過とともにブロックに行が追加され、行データがブロック・ヘッダーの方向へ上向きに増加し、ブロック・ヘッダー自体も、行データの方向へ下向きに増加します。PCTFREEの設定によって、データ・ブロックの20%以上が空き領域となることが保証されます。たとえば、行を挿入すると行データとヘッダーの合計がブロック領域全体の90%を占め、空き領域が10%になる場合には、INSERT文は実行できません。

図12-10 PCTFREE



ノート:

この説明は、PCTFREE 記憶域パラメータや空きリストを使用しない LOB データ型については該当しません。

関連項目:

- 「LOBの概要」
- PCTFREEパラメータの構文およびセマンティクスについては、『Oracle Database SQL言語リファレンス』を参照してください

データ・ブロック内の空き領域の最適化

空き領域の割合は、PCTFREEよりも小さくなることはありませんが、大きくなることがあります。たとえば、PCTFREEを20%に設定すると、空き領域の合計がブロックの5%になることはありませんが、50%になることはあります。

空き領域の拡大による最適化

一部のDML文を使用すると、データ・ブロックの空き領域のサイズが増えます。

次の文を使用すると、空き領域のサイズが増えます。

- DELETE文
- 既存の値をより小さいサイズの値に更新するか、既存の値をより大きいサイズの値に更新して行の移行を強制するUPDATE文
- 高度な行圧縮を使用する表に対するINSERT文

  INSERT文によってブロックがデータでいっぱいになると、データベースによってブロック圧縮が開始され、ブロックにより多くの空き領域が生じる可能性があります。

解放された領域は、次の条件下でINSERT文に使用できます。

- INSERT文が同じトランザクション内に存在し、領域を解放した文の後にある場合、その文で使用可能になった領域を

使用できます。

- INSERT文が、領域を解放する文とは別のトランザクションに含まれている(別々のユーザーが実行している)場合、その文では、もう一方のトランザクションがコミットされた後で、しかもその領域が必要とされる場合にかぎり、使用可能になった領域を使用できます。

関連項目:

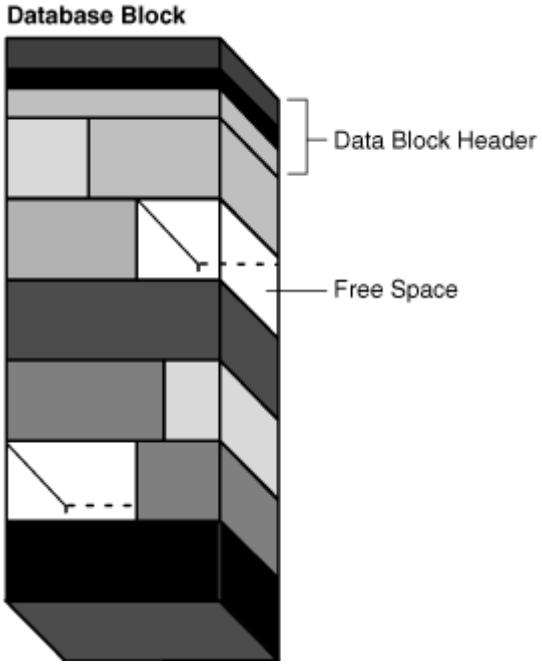
拡張行圧縮について学習するには、『Oracle Database管理者ガイド』を参照してください

断片化領域の結合による最適化

解放された領域は、データ・ブロック内の主な空き領域と連続しているとはかぎりません。連続しない空き領域は、断片化領域と呼ばれます。

次の図に、連続しない空き領域を含むデータ・ブロックを示します。

図12-11 断片化領域があるデータ・ブロック

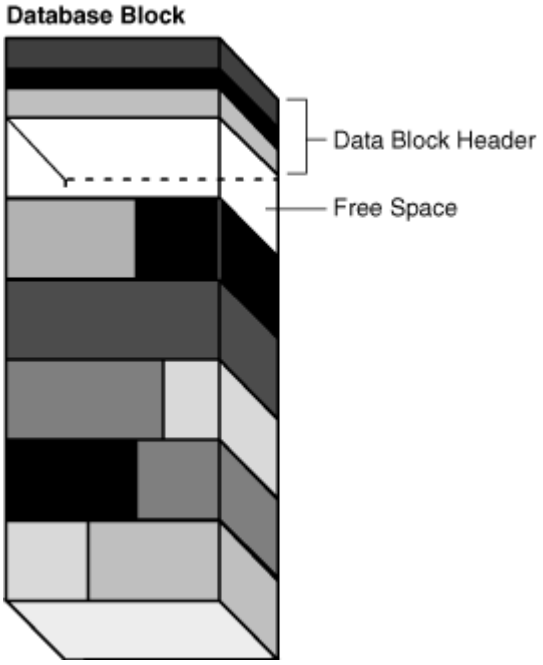


Oracle Databaseでは、次の条件に当てはまる場合にのみ、データ・ブロックの空き領域が自動的かつ透過的に結合されます。

- INSERT文またはUPDATE文において、新しい行断片を格納するために十分な空き領域のあるブロックの使用が試行された場合。
- 空き領域が断片化されており、行断片をブロック内の連続したセクションに挿入できない場合。

結合後の空き領域のサイズは結合前と同じですが、領域は連続的に存在するようになります。図12-12に、領域結合後のデータ・ブロックを示します。

図12-12 空き領域結合後のデータ・ブロック



Oracle Databaseでは、前述の状況でのみ結合が実行されます。それ以外の状況で結合を実行すると、データ・ブロック内の空き領域が連続的に結合されることによって、パフォーマンスが低下する可能性があるためです。

行の連鎖と移行

Oracle Databaseでは、大きすぎる行を1つのブロック内に収まるように管理するため、連鎖と移行が使用されます。

> ノート:
>
> 標準表の連鎖行は、ブロックチェーン表の行連鎖とは異なります。Oracle Database では、様々なテクノロジを使用してブロックチェーン表内の行を管理します。

次の状況が考えられます。

- 最初に行を挿入するとき、大きすぎて1つのデータ・ブロック内に収まらない場合。

  行連鎖では、Oracle Databaseにより、その行のデータは、セグメント用に確保された1つ以上のデータ・ブロックの連鎖に格納されます。多くの場合、行連鎖は、行が大きい場合に発生します。大きい行の例には、LONGまたはLONG RAWデータ型の列を含む行、または行に含まれる列の数が多すぎる場合があります。そのような場合の行連鎖は、避けられません。

- 1つのデータ・ブロックに収まっていた行が更新されて、行全体の長さが増加したが、更新後の行を保持する十分な空き領域がない場合。

  行の移行では、Oracle Databaseにより、行全体が新しいブロックに収まることを前提として、行全体が新しいデータ・ブロックに移動されます。移行された行の元の行断片には、行の移行先の新しいブロックへのポインタまたは「転送先アドレス」が含まれます。そのため、移動された行のROWIDは変わりません。

- 行に255を超える列が含まれる場合。

  Oracle Databaseでは、1つの行断片に格納できる列数は、255列のみです。したがって、1000列の行を表に挿入すると、データベースでは通常、複数のブロックに連鎖された4つの行断片が作成されます。

図12-13に、データ・ブロックへの大きなサイズの行の挿入を示します。この行は左側のブロックには大きすぎて格納し切れないため、Oracle Databaseでは行を連鎖し、最初の行断片を左側のブロックに配置し、2つ目の行断片を右側のブロックに配

置します。

図12-13 行の連鎖

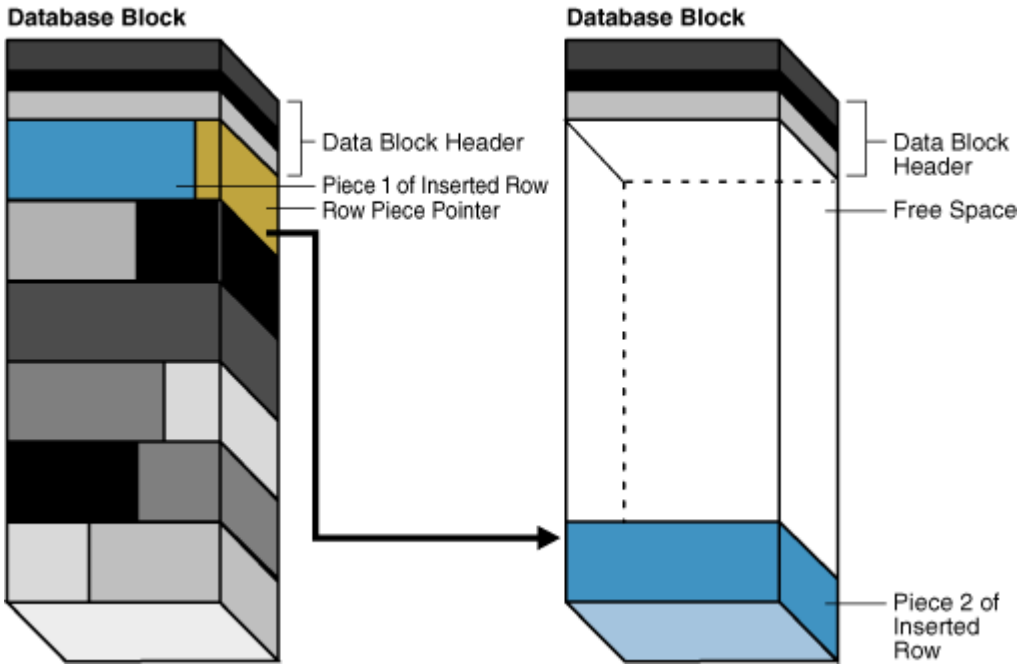
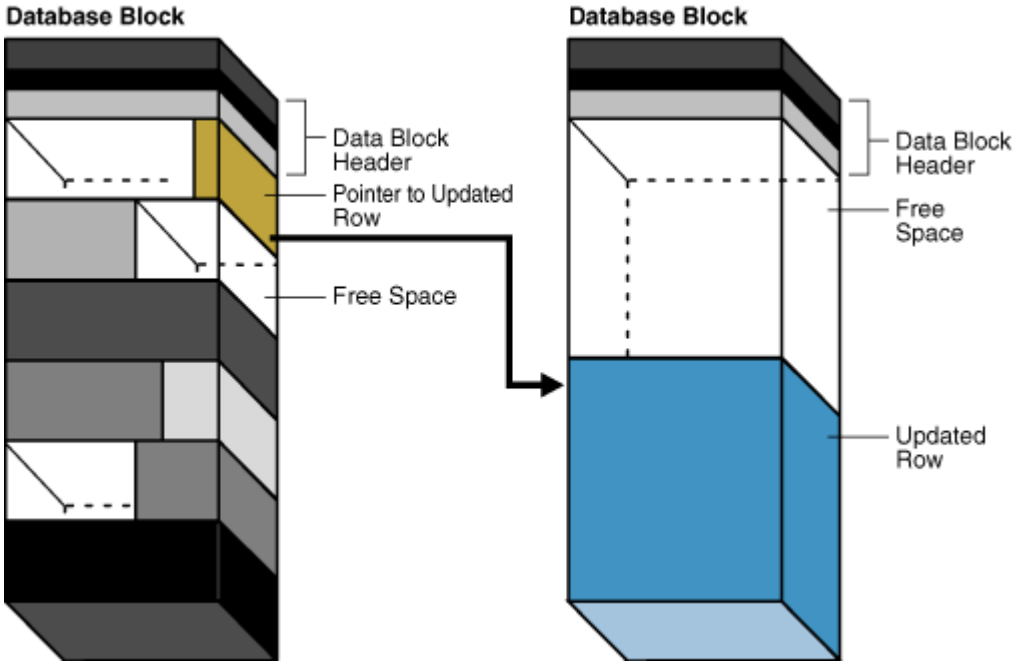


図12-14では、左側のブロックには、更新によってサイズが増大してこのブロックに収まらなくなった行が格納されています。Oracle Databaseにより、この行全体が右側のブロックに移動され、この移行後の行へのポインタが左側のブロックに残されます。

図12-14 行の移行



行が連鎖または移行されると、データの取出しに必要なI/Oが増加します。これは、その行の情報を取り出す際に、Oracle Databaseが複数のブロックをスキャンする必要があるためです。たとえば、索引の読取りに対して1回のI/Oを行い、移行されていない表の行を読み取るために1回のI/Oを行う場合、移行された行のデータを取得するために追加のI/Oが必要になります。

セグメント・アドバイザは、再利用可能な領域のあるセグメントを特定するOracle Databaseコンポーネントであり、手動または自動で実行できます。セグメント・アドバイザを使用すると、多くの空き領域のあるオブジェクト、または多くの連鎖行があるオブジェクトについてのアドバイスを入手できます。

関連項目:

- 「行の格納」および「行断片のROWID」。
- ブロックチェーン表の概要
- 未使用領域の再利用方法を学習するには、『Oracle Database管理者ガイド』を参照してください。
- 行の連鎖と移行を削減する方法について学習するには、『Oracle Databaseパフォーマンス・チューニング・ガイド』を参照してください。

索引ブロックの概要

索引ブロックは、領域を表ブロックとは異なる方法で管理する特殊なタイプのデータ・ブロックです。Oracle Databaseでは、索引ブロックを使用して、索引の論理記憶域を管理します。

索引ブロックのタイプ

索引には、ルート・ブロック、ブランチ・ブロックおよびリーフ・ブロックが含まれています。

ブロック・タイプは次のように定義されています。

- ルート・ブロック

  このブロックは、索引へのエントリ・ポイントを識別します。

- ブランチ・ブロック

  データベースでは、索引キーを検索する際、ブランチ・ブロックを検索します。

- リーフ・ブロック

  これらのブロックには、関連行を指す索引付きキー値の行IDが含まれています。リーフ・ブロックにはキー値がソート順に格納され、データベースではキー値の範囲内のすべての行を効率的に検索できます。

索引エントリの記憶域

索引エントリは、表の行がデータ・ブロックに格納されるのと同じように、索引ブロックに格納されます。ブロック部分の索引エントリはバイナリの順序ではなく、ヒープで格納されます。

データベースは、データ・ブロックのディレクトリとは異なる索引ブロックの行ディレクトリを管理します。行ディレクトリのエントリ(索引ブロックの本体のエントリではない)は、キー値で順序付けられます。たとえば、行ディレクトリでは、索引キー000000のディレクトリ・エントリは、索引キー111111のディレクトリ・エントリよりも優先されます。

行ディレクトリのエントリの順序は、索引スキャンの効率を改善します。レンジ・スキャンでは、データベースはレンジに指定されたすべての索引キーを読み取る必要があります。データベースは、ブランチ・ブロックを横断して、最初のキーを含むリーフ・ブロックを特定します。行ディレクトリのエントリがソートされるため、データベースは、バイナリ検索を使用してレンジ内の最初のキーを検索し、最後のキーが見つかるまで行ディレクトリのエントリを順に検索します。このようにして、データベースはリーフ・ブロック本体のすべてのキーを読み取らないようにできます。

関連項目:

「データ・ブロック・オーバーヘッド」

索引ブロック内のスロットの再利用

データベースでは、索引ブロック内の領域を再利用できます。

たとえば、アプリケーションはある値を列に挿入してから、その値を削除する場合があります。行に領域が必要な場合、データベースでは、削除された値によって以前専有されていた索引スロットを再利用できます。

索引ブロックには通常、ヒープ構成表ブロックより多くの行があります。単一の索引ブロックに多数の行を格納する機能により、新規データ格納のための頻繁なブロック分割が避けられることから、データベースでの索引の管理が容易になります。

索引は、それ自体は結合できませんが、REBUILDまたはCOALESCEオプションを指定したALTER INDEX文により、手動で結合できます。たとえば、列に1から500000の値を移入し、その後偶数を含む行を削除した場合、索引には250000の空のスロットが含まれます。データベースでは、空のスロットを含む索引ブロックに適合するデータを挿入できる場合のみ、スロットを再利用します。

索引ブロックの結合

索引の結合は、既存の索引データを所定の位置で圧縮し、再編成によりブロックが解放された場合、空きブロックをその索引構造に残します。したがって、結合により索引ブロックは他のユーザーには解放されず、その索引ではブロックが再割当てされます。

Oracle Databaseでは索引は自動的に圧縮されず、REBUILDまたはCOALESCEオプションを指定したALTER INDEX文を実行する必要があります。

図12-15に、索引結合前のemployees.department_id列の索引を示します。最初の3つのリーフ・ブロックは、グレーの格納ラインで示されているように、まだ満杯ではありません。

図12-15 結合前の索引

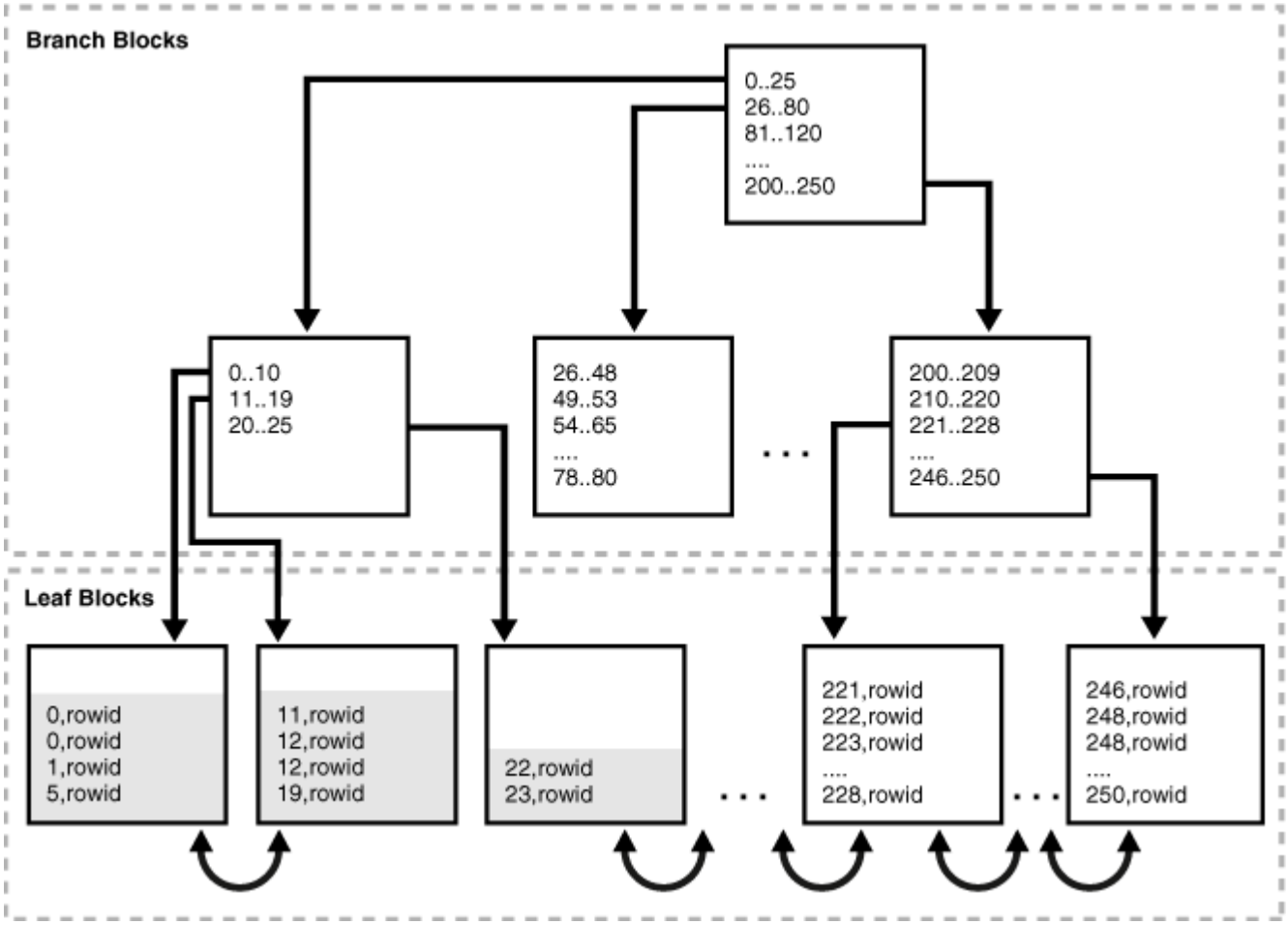
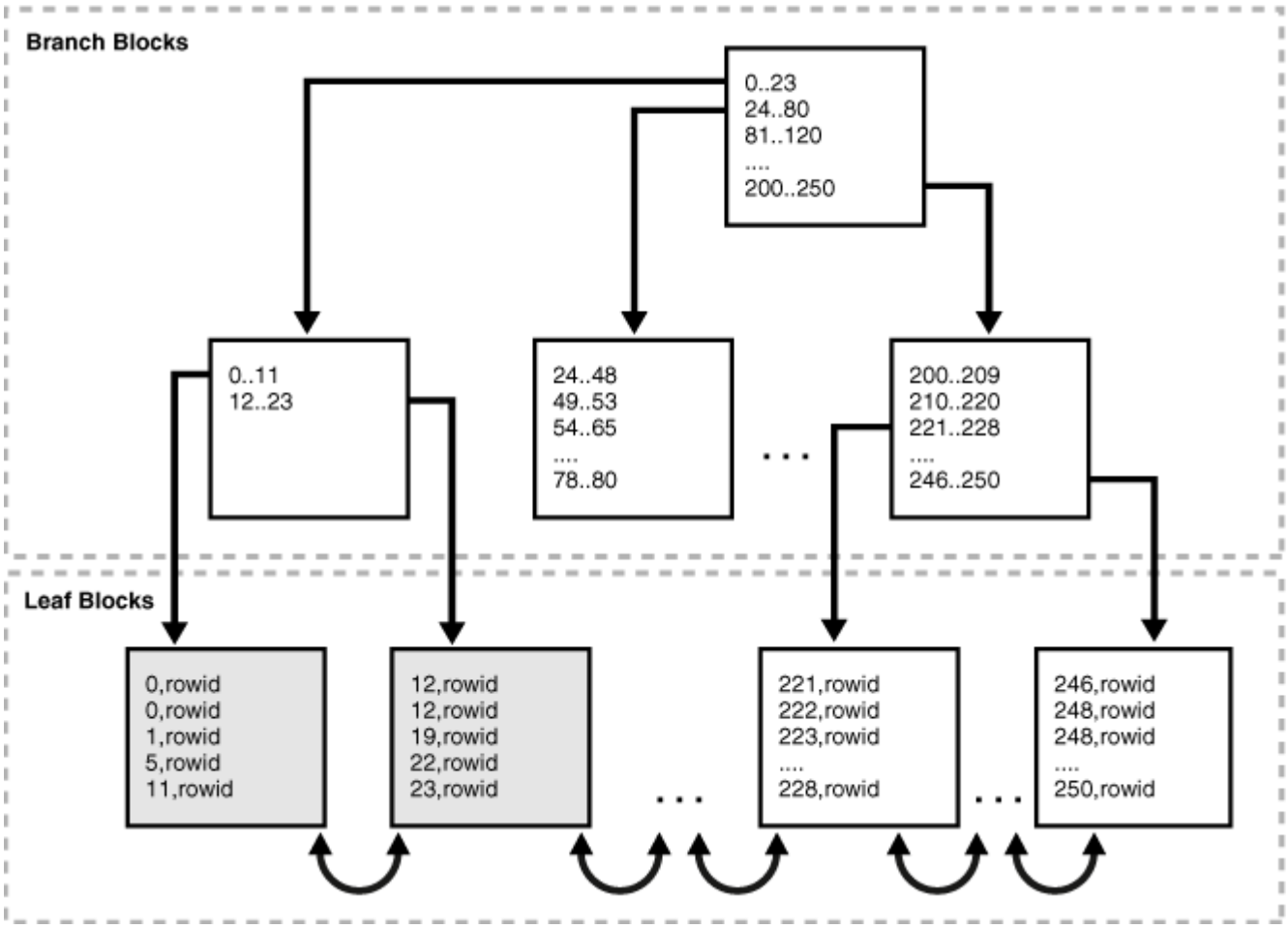


図12-16に、図12-15の索引の結合後を示します。最初の2つのリーフ・ブロックは、グレーの格納ラインで示されているように満杯になり、3つ目のリーフ・ブロックは解放されています。

図12-16 結合後の索引



関連項目:

- 索引を結合および再構築する方法を学習するには、Oracle Database管理者ガイドを参照してください
- COALESCE文について学習するには、Oracle Database SQL言語リファレンスを参照してください

エクステントの概要

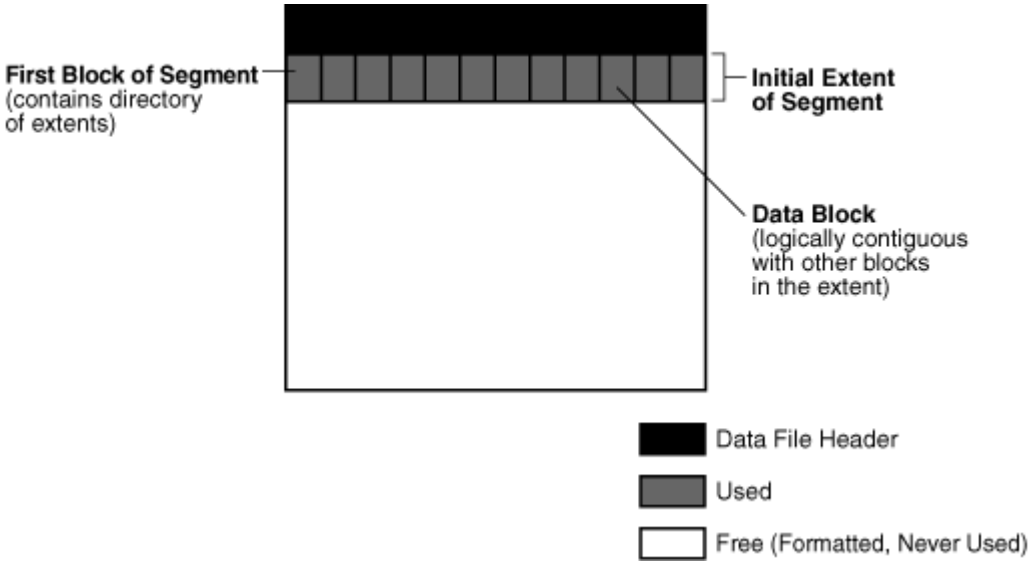
エクステントは、連続したデータ・ブロックで論理的に構成される、データベース記憶域のユニットです。データ・ブロックは、RAIDのストライプ化やファイルシステムの実装によって、物理的にはディスク上に断続的に存在する可能性があります。

エクステントの割当て

データベースでは、セグメントが作成されたときにデータ・セグメント用の初期エクステントがデフォルトで割り当てられます。エクステントは、常に1つのデータファイルに含まれています。

セグメントにデータが追加されていない場合でも、初期エクステント内にこのセグメント専用のデータ・ブロックが予約されます。各セグメントの最初のデータ・ブロックには、そのセグメント内のエクステントのディレクトリが含まれています。図12-17に、データが含まれていなかったデータファイル内の、セグメントの初期エクステントを示します。

12-17 セグメントの初期エクステント

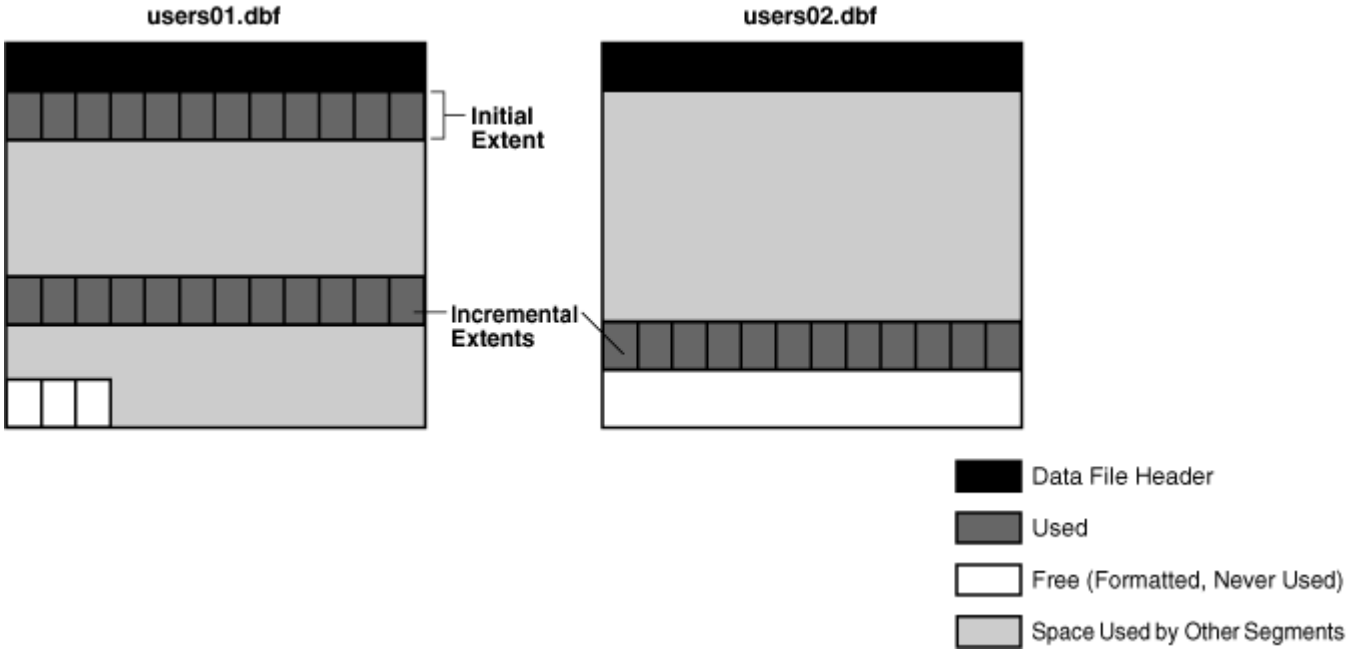


初期エクステントがいっぱいになり、さらに追加の領域が必要になると、データベースによって自動的にこのセグメントの増分エクステントが割り当てられます。増分エクステントは、セグメントに対して後から作成されるエクステントです。

割当てアルゴリズムは、ローカル管理表領域であるか、ディクショナリ管理表領域であるかに応じて異なります。ローカル管理表領域の場合は、データファイルのビットマップで、隣接する空きブロックが検索されます。データファイルに十分な領域がない場合は、他のデータファイルが検索されます。セグメントのエクステントは常に同じ表領域内に存在しますが、異なるデータファイル内に存在する場合もあります。

図12-18に、セグメントのエクステントは表領域内の任意のデータファイル内に割当て可能であることを示します。たとえば、セグメントの初期エクステントをusers01.dbfに、最初の増分エクステントをusers02.dbfに、次のエクステントをusers01.dbfに割り当てることができます。

図12-18 セグメントの増分エクステント



新しく割り当てられたエクステントのブロックは、空き領域であったブロックですが、古いデータが残っていることがあります。ASSMでは、新たに割り当てられたエクステントの使用開始時に、Oracle Databaseによってそのエクステントのブロックがフォーマットされます(ただし、必要な場合のみ)。

ノート:

この項は、1 つのサーバー・プロセスによって文を解析して実行する、シリアル操作に適用されます。複数のサー

バー・プロセスを必要とするパラレル SQL 文では、エクステントは異なる方法で割り当てられます。

関連項目:

- 「セグメント領域と最高水位標」
- 手動でのエクステントの割当て方法を学習するには、Oracle Database管理者ガイドを参照してください

エクステントの割当て解除

一般的に、ユーザー・セグメントのエクステントは、DROP文を使用してオブジェクトを削除するまでは表領域に戻されません。

たとえば、表の行をすべて削除しても、データベースによりデータ・ブロックが再利用されて表領域内の他のオブジェクトがそれらのブロックを使用できるようになることはありません。セグメントは、DBMS_SPACE_ADMINパッケージを使用して削除することもできます。

ノート:

UNDO セグメントでは、サイズに OPTIMAL が指定されている場合、またはデータベースが自動 UNDO 管理モードである場合には、Oracle Database により定期的に 1 つ以上のエクステントの割当てが解除されます。

領域の割当てを手動で解除できる場合もあります。Oracle Segment Advisorは、オブジェクト内の断片化のレベルに応じて、オブジェクトに再利用可能な領域があるかどうかを判断する際に役立ちます。次の方法でエクステントを解放できます。

- オンラインでのセグメントの縮小を使用すると、セグメント内の断片化領域を再利用します。セグメントの縮小は、オンラインのインプレース操作です。一般的に、データ圧縮によって、より効率的にキャッシュを利用できるようになり、全表スキャン時に読み取る必要があるブロック数が少なくなります。
- パーティション化されていない表や表パーティションのデータを新しいセグメントに移動し、必要であれば割当て制限を持つ別の表領域に移動します。
- 索引を再作成または結合します。
- 表または表クラスタを切り捨てて、すべての行を削除します。デフォルトで、Oracle Databaseでは、MINEXTENTS記憶域パラメータで指定されたサイズを除き、削除された行で使用していたすべての領域の割当てが解除されます。Oracle Database 11gリリース2 (11.2.0.2)以降、DROP ALL STORAGEオプションを指定してTRUNCATEを使用すれば、セグメント全体を削除することもできます。
- 未使用領域の割当てを解除すると、データベース・セグメントの最高水位標の最後で未使用領域を解放して、表領域の他のセグメントで領域を使用できるようにします。

エクステントが解放されると、Oracle Databaseでは、ローカル管理表領域のデータファイル内のビットマップを変更し、再度取得されたエクステントを使用可能領域として反映します。解放されたエクステントのブロックにあるデータにはアクセスできなくなります。

関連項目:

- 「索引ブロックの結合」

- 「UNDO表領域」
- 「セグメント領域と最高水位標」
- セグメント領域の再利用方法を学習するには、『Oracle Database管理者ガイド』を参照してください

エクステントの記憶域パラメータ

各セグメントは、エクステントについての設定である記憶域パラメータによって定義されます。記憶域パラメータにより、Oracle Databaseで特定のセグメントに空き領域がどのように割り当てられるかが制御されます。

記憶域設定は、次に示す優先順位で決定されます。つまり、リストの上位の設定は下位の設定よりも優先されます。

1. セグメントのSTORAGE句
2. 表領域のSTORAGE句
3. Oracle Databaseのデフォルト

ローカル管理表領域の場合は、均一のエクステント・サイズでも、システムによって自動的に決定される可変エクステント・サイズでもかまいません。

- 均一エクステントの場合は、エクステント・サイズを指定するか、デフォルト・サイズである1MBを使用できます。表領域内のすべてのエクステントが指定したサイズになります。ローカル管理一時表領域の場合は、均一サイズの割当てのみ使用できます。
- 自動割当てエクステントの場合は、追加エクステントの最適サイズがOracle Databaseにより自動的に決定されます。

ローカル管理表領域では、一部の記憶域パラメータは表領域レベルでは指定できません。ただし、これらのパラメータをセグメント・レベルで指定することはできます。この場合、データベースでは、すべてのパラメータをまとめて使用して、セグメントの初期サイズが計算されます。内部のアルゴリズムによって、その後に割り当てられる各エクステントのサイズが決定されます。

関連項目:

- ローカル管理表領域を作成する場合のエクステント管理についての考慮事項について学習するには、『Oracle Database管理者ガイド』を参照してください
- STORAGE句のオプションについて学習するには、『Oracle Database SQL言語リファレンス』を参照してください

セグメントの概要

セグメントは、表領域内の特定の論理記憶域構造のデータがすべて入っている、エクステントの集合です。

たとえば、Oracle Databaseでは、1つ以上のエクステントを割り当てることによって、表のデータ・セグメントを形成します。データベースでは、表の索引の索引セグメントも、1つ以上のエクステントを割り当てることによって形成します。

Oracle Databaseにより、自動または手動でセグメント領域が管理されます。この項では、ASSMを使用することを前提としています。

関連項目:

ASSMについてさらに学習するには、「論理領域管理」を参照

ユーザー・セグメント

データベース内の1つのデータ・セグメントには、1つのユーザー・オブジェクトのデータが格納されます。

セグメントには様々なタイプがあります。次に、ユーザー・セグメントの例を示します。

- 表、表パーティションまたは表クラスタ
- LOBまたはLOBパーティション
- 索引または索引パーティション

パーティション化されていない各オブジェクトおよび各オブジェクト・パーティションは、それぞれ独自のセグメントに格納されます。たとえば、索引に5つのパーティションがある場合は、5つのセグメントに索引データが格納されます。

ユーザー・セグメントの作成

デフォルトで、データベースでは遅延セグメント作成により、表、索引およびパーティションを作成する際、データベース・メタデータのみが更新されます。

ユーザーが最初の行を表またはパーティションに挿入するときに、その表またはパーティション、LOB列およびその索引のセグメントが作成されます。遅延セグメント作成が発生すると、データベース・リソースが必要以上に使用されなくなります。たとえば、アプリケーションをインストールすると、数千ものオブジェクトが作成されて、ディスク領域を大量に消費する場合があります。このようなオブジェクトの多くは、実際には使用されない可能性があります。

DBMS_SPACE_ADMINパッケージは、空のオブジェクト用のセグメントを管理します。このPL/SQLパッケージを使用すれば、次のことができます。

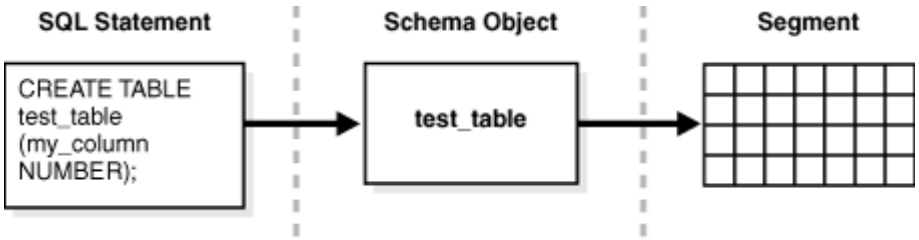
- セグメントが作成されていない空の表またはパーティション用のセグメントの手動マテリアライズ
- 空のセグメントが割り当てられている空の表またはパーティションからのセグメントの削除

オブジェクトの作成とセグメントの作成との関係がわかりやすくなるように、遅延セグメント作成は発生しないものとします。次のような表を作成するとします。

```
CREATE TABLE test_table (my_column NUMBER);
```

図12-19に示すように、表のセグメントが1つ作成されます。

図12-19 ユーザー・セグメントの作成

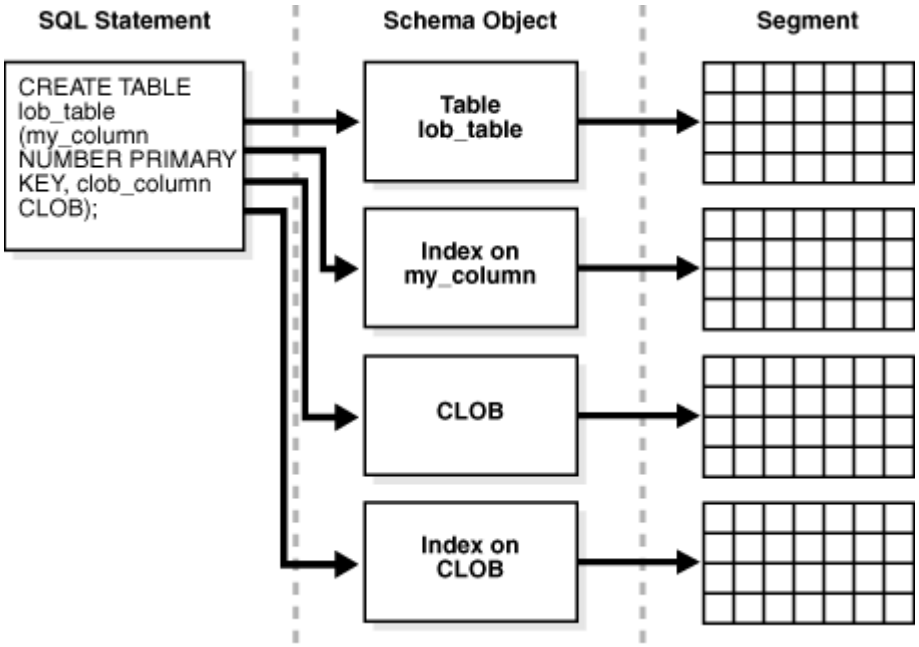


主キーまたは一意キーを持つ表を作成する場合は、Oracle Databaseによって、このキーの索引が自動的に作成されます。ここでも、遅延セグメント作成は発生しないものとします。次のような表を作成するとします。

```
CREATE TABLE lob_table (my_column NUMBER PRIMARY KEY, clob_column CLOB);
```

図12-20は、lob_tableのデータがあるセグメントに格納され、暗黙的に作成された索引が別のセグメントに格納されることを示しています。また、CLOBデータは、関連付けられたCLOB索引と同様に、専用のセグメントに格納されます。したがって、CREATE TABLE文を実行すると4つの異なるセグメントが作成されます。

図12-20 複数のセグメント



ノート:

表のセグメントとその表の索引のセグメントは、同一の表領域に存在する必要はありません。

セグメントが作成されると、1つ以上のエクステントが割り当てられます。オブジェクトの記憶域パラメータにより、各セグメントのエクステントが割り当てられる方法が決定します。これらのパラメータは、オブジェクトに関連するデータ・セグメントのデータ検索と格納の効率に影響を与えます。

関連項目:

- 「内部LOB」
- 「エクステントの記憶域パラメータ」
- 遅延セグメント作成の管理方法を学習するには、『Oracle Database管理者ガイド』を参照
- CREATE TABLEの構文については、『Oracle Database SQL言語リファレンス』を参照

一時セグメント

Oracle Databaseでは、問合せの処理中に、SQL文実行の中間段階で、一時作業領域が必要になることがあります。

一時セグメントが必要になる一般的な操作には、ソート、ハッシング、ビットマップのマージなどがあります。また、索引の作成中には索引セグメントも一時セグメントに配置され、索引の作成が完了すると、永続セグメントに変換されます。

Oracle Databaseでは、メモリー内で操作が完了する場合、一時セグメントは作成されません。ただし、メモリーを使用できない場合には、ディスク上に一時セグメントが自動的に割り当てられます。

問合せ用の一時セグメントの割当て

Oracle Databaseでは、ユーザー・セッション中に必要に応じて問合せ用の一時セグメントを割り当て、問合せが完了するとセ

グメントを削除します。一時セグメントに対する領域管理操作以外の、一時セグメントに対する変更は、オンラインREDOログに記録されません。

データベースでは、ユーザーに割り当てられた一時表領域に一時セグメントが作成されます。表領域のデフォルトの記憶域特性によって、一時セグメント内のエクステントの特性が決定されます。一時セグメントの割当てと割当て解除は頻繁に発生するため、一時セグメント専用に1つ以上の表領域を作成してください。これにより、I/Oがディスク間で分散され、SYSTEM表領域やその他の一時セグメントがある表領域の断片化が回避されます。

ノート:

SYSTEM 表領域がローカル管理の場合は、データベースの作成時にデフォルトの一時表領域を定義する必要があります。ローカル管理の SYSTEM 表領域は、デフォルトの一時記憶域として使用できません。

関連項目:

- 「オンラインREDOログの概要」
- 一時表領域の作成方法を学習するには、『*Oracle Database*管理者ガイド』を参照してください。
- CREATE TEMPORARY TABLESPACEの構文およびセマンティクスについては、『*Oracle Database SQL*言語リファレンス』を参照してください。

一時表および索引用のセグメントの割当て

Oracle Databaseは、一時表およびその索引用の一時セグメントを割り当てることができます。

一時表には、トランザクションまたはセッションの期間中にのみ存在するデータが保持されます。各セッションは、それ自体に割り当てられたエクステントにのみアクセスし、他のセッションに割り当てられたエクステントにはアクセスできません。

Oracle Databaseが一時表に対してセグメントを割り当てるのは、グローバル一時表の場合は表への最初のINSERTが発生したときに、プライベート一時表の場合は必要に応じてです。この挿入は、明示的に行われたものでも、CREATE TABLE AS SELECTによって行われたものでもかまいません。データベースは、表とその索引にセグメントを割り当て、索引のルート・ページを作成し、LOBセグメントを割り当てます。

現行ユーザーの一時表領域は、一時表のセグメントを割り当てます。たとえば、user1に割り当てられた一時表領域がtemp1であり、user2に割り当てられた一時表領域がtemp2であるとします。この場合、user1の一時データはtemp1のセグメントに格納され、user2の一時データはtemp2のセグメントに格納されます。

関連項目:

- 「一時表の概要」
- 一時表の作成方法を学習するには、『Oracle Database管理者ガイド』を参照

UNDOセグメント

Oracle Databaseには、**UNDOデータ**と総称されている、トランザクションのアクションのレコードが保持されています。

Oracle Databaseでは、次の場合にUNDOデータが使用されます。

- アクティブ・トランザクションのロールバック
- 終了したトランザクションのリカバリ
- 読取り一貫性の提供
- 一部の論理フラッシュバック操作の実行

Oracle Databaseでは、UNDOデータは、外部ログではなくデータベース内部に格納されます。UNDOデータが格納されるブロックは、データ・ブロックと同様に更新され、これらのブロックに対する変更によってREDOレコードが生成されます。このように、Oracle Databaseでは、外部ログを読み取ることなくUNDOデータに効率的にアクセスできます。

永続オブジェクトのUNDOデータは、UNDO表領域に格納されます。Oracle Databaseには、UNDO表領域内のUNDOセグメントおよび領域を管理するための自動UNDO管理モードと呼ばれる完全に自動化されたメカニズムが提供されています。

データベースでは、UNDOデータを2つのストリームに分けます。一時UNDOストリームでは、一時オブジェクトへの変更によって生成されたUNDOレコードのみをカプセル化するのに対して、永続UNDOストリームでは、永続オブジェクトのUNDOレコードのみをカプセル化します。データベースでは、一時および永続UNDOを別々に管理します。UNDOの分離により記憶域が減り、次のことを行うことでパフォーマンスが向上します。

- 永続および一時表に最適の永続およびUNDO表領域のサイズを構成できるようにします。
- オンラインREDOログに書き込まれたREDOのサイズを縮小します。
- 一時UNDOデータのバックアップの必要を回避します。

Active Data Guardインスタンスでは、グローバル一時表のDMLで、UNDOが一時UNDOセグメントに生成される必要があります。

関連項目:

- 「オンラインREDOログの使用」
- 「一時UNDOセグメント」
- 一時UNDOセグメントについて学習するには、Oracle Database管理者ガイドを参照してください
- TEMP_UNDO_ENABLED初期化パラメータについて学習するには、Oracle Databaseリファレンスを参照してください

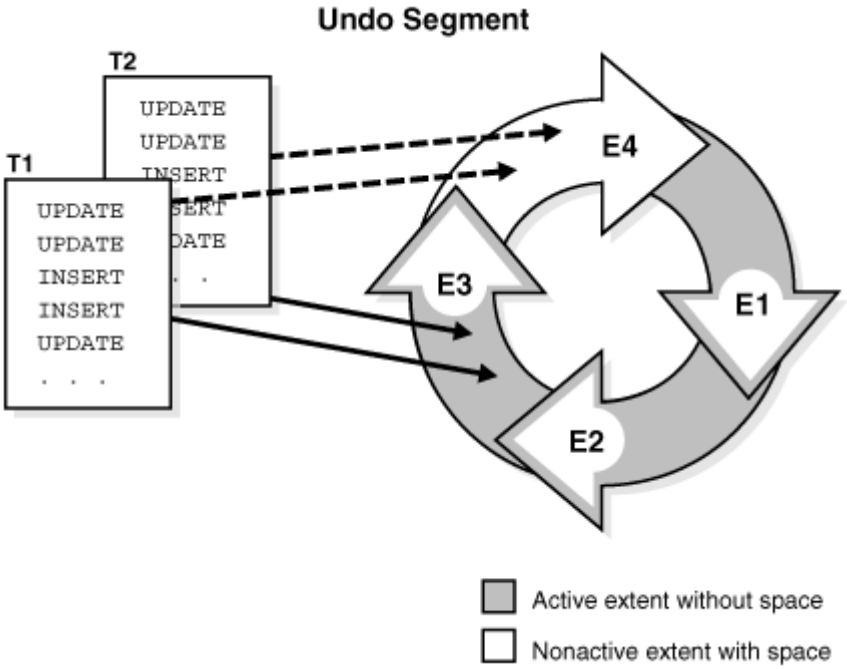
UNDOセグメントとトランザクション

トランザクションが開始するときに、そのトランザクションは、現在のUNDO表領域内のUNDOセグメント(**トランザクション表**)にバインド(割当て)されます。データベース・インスタンスに対し専用のUNDO表領域が割り当てられていない場合は、トランザクションがシステムUNDOセグメントにバインドされます。

複数のアクティブなトランザクションが、同じUNDOセグメントに同時に書き込むこともあれば、別々のセグメントに書き込むこともあります。たとえば、トランザクションT1とT2が両方ともU1というUNDOセグメントに書き込むこともあれば、T1がU1に書き込み、T2がU2というUNDOセグメントに書き込むこともあります。

概念的には、UNDOセグメント内のエクステントは輪のようなものです。トランザクションは、輪に沿って回るように、UNDOエクステントに順に書き込んでいきます。図12-21では、T1とT2という2つのトランザクションを示し、この2つのトランザクションはそれぞれ、UNDOセグメントの3番目のエクステント(E3)から書込みを開始し、4番目のエクステント(E4)まで書込みを進めます。

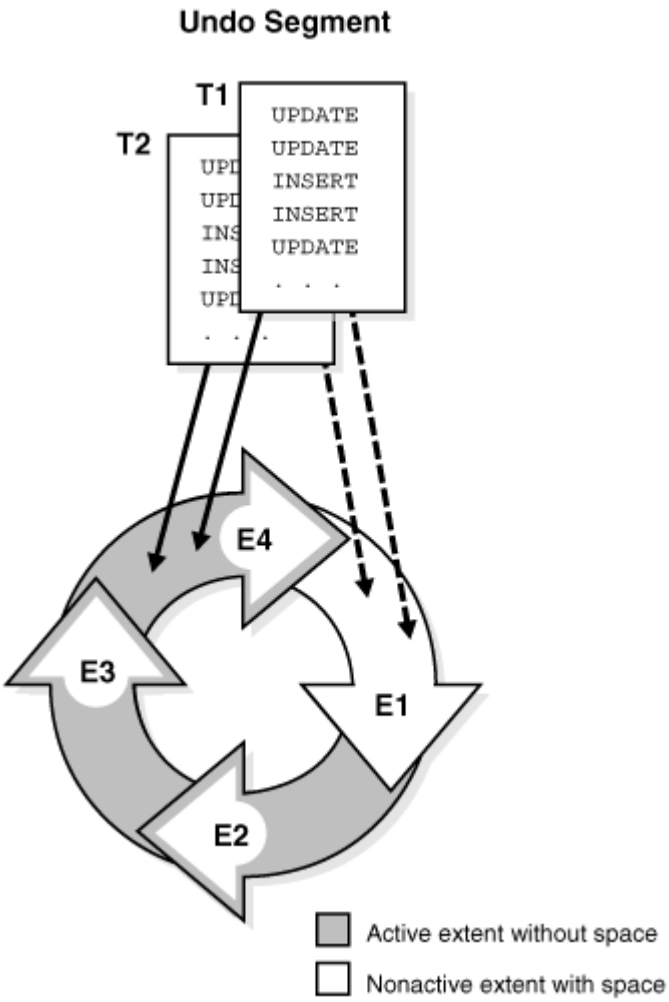
図12-21 UNDOセグメントに円環状に割り当てられたエクステント



トランザクションは必ずUNDOセグメント内のエクステントに1つずつ順番に書き込み、その時点で書込み可能なエクステントをトランザクションの現行エクステントと呼びます。複数のアクティブなトランザクションが同時に、同じ現行エクステントに書き込むことも、別々の現行エクステントに書き込むことも可能です。図12-21では、T1とT2のトランザクションがエクステントE3に同時に書き込んでいます。UNDOエクステント内では、各データ・ブロックに格納されるデータは、1つのトランザクションのデータのみです。

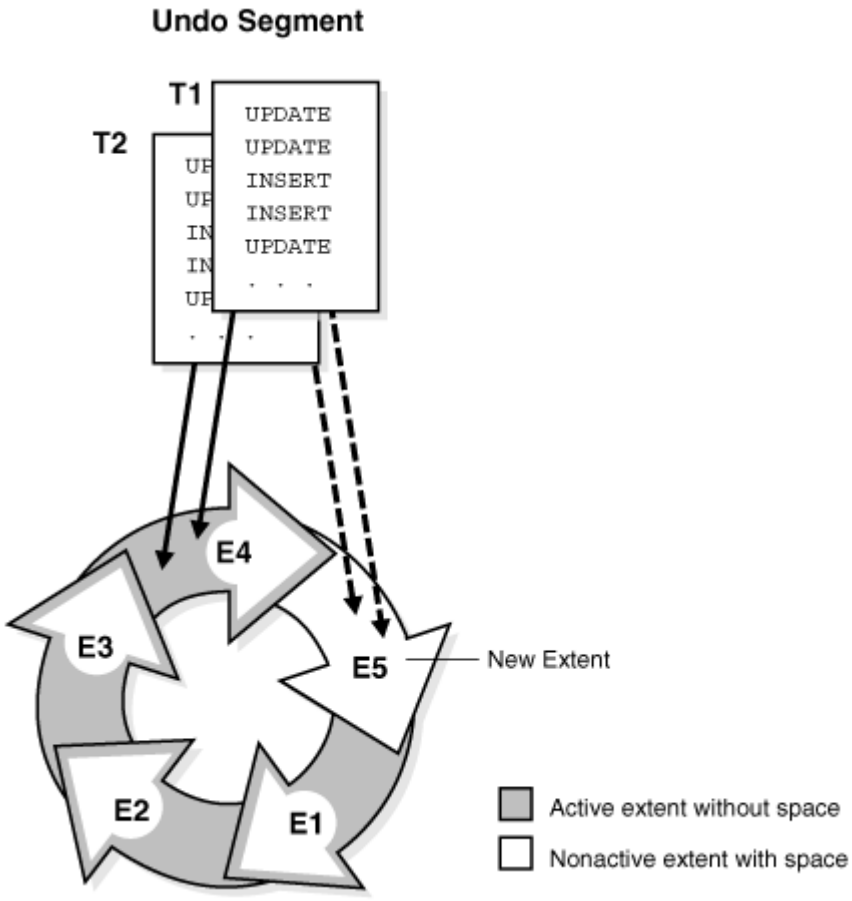
現在のUNDOエクステントがいっぱいになると、領域を必要とする最初のトランザクションによって、輪の中にある次の割当てエクステントの空き状況がチェックされます。次のエクステントにアクティブなトランザクションのデータが格納されていない場合は、このエクステントが現行エクステントになります。この時点で、領域を必要としているすべてのトランザクションが、新しい現行エクステントに書込み可能な状態になります。図12-22では、E4がいっぱいになり、T1とT2がE1に書込みを継続し、E1内のアクティブでないUNDOデータを上書きしています。

図12-22 UNDOセグメントに円環状に割り当てられたエクステントの再利用



次のエクステントにアクティブなトランザクションのデータが格納されている場合は、新しいエクステントが割り当てられます。図12-23では、T1とT2はE4に書き込んでいます。E4がいっぱいになっても、E1にはアクティブなUNDOデータが格納されているため、トランザクションはE1に書込みを継続することができません。そこで、このUNDOセグメントに新しいエクステント(E5)が割り当てられます。トランザクションはE5に書込みを継続します。

図12-23 UNDOセグメントに対する新しいエクステントの割当て



関連項目:

UNDOセグメントの管理方法を学習するには、Oracle Database管理者ガイドを参照してください

トランザクションのロールバック

ROLLBACK文が発行されると、UNDOレコードは、コミットされていないトランザクションによってデータベースに加えられた変更を元に戻すために使用されます。

リカバリ中には、オンラインREDOログからデータファイルに適用されたコミットされていないすべての変更が元に戻されます。UNDOレコードでは、あるユーザーによるデータの変更中にそのデータにアクセスしているユーザー用に、そのデータの修正前のイメージを維持することで、読取り一貫性が実現されます。

一時UNDOセグメント

一時UNDOセグメントは、一時UNDOデータ専用のオプションの領域管理コンテナです。

一時表への変更のUNDOレコードは、セッション固有であるとともに、読取りの一貫性とトランザクション・ロールバックに対してのみ役立ちます。Oracle Database 12cまでは、データベースではこれらのレコードをいつでもオンラインREDOログに格納していました。一時オブジェクトに対する変更はオンラインREDOログに記録されないため、一時オブジェクトのUNDOを一時UNDOセグメントに書き込むと、オンラインREDOログおよびアーカイブ済のREDOログ・ファイルで領域が節約されます。データベースでは、UNDOへの変更、または一時表への変更は記録されないため、パフォーマンスが向上します。

TEMP_UNDO_ENABLED初期化パラメータを設定すれば、一時表でUNDOデータを一時UNDOセグメントに格納できます。このパラメータがTRUEの場合、データベースでは一時表領域から一時UNDOセグメントを割り当てます。

関連項目:

- 一時UNDOセグメントについて学習するには、Oracle Database管理者ガイドを参照してください
- TEMP_UNDO_ENABLED初期化パラメータについて学習するには、Oracle Databaseリファレンスを参照してください

セグメント領域と最高水位標

Oracle Databaseでは、領域を管理するために、セグメント内のブロックの状態を追跡します。**最高水位標(HWM)**は、セグメント内の位置を指し示し、その位置を超えると、データ・ブロックは未フォーマットであり、使用されていません。

MSSMでは、空きリストを使用してセグメント領域が管理されます。表の作成時には、セグメント内のブロックはフォーマットされていません。セッションが最初に行を表に挿入すると、空きリストが検索され、使用可能なブロックが検出されます。使用可能なブロックが見つからなかった場合には、ブロックのグループが事前にフォーマットされ、それらが空きリストに追加され、ブロックへのデータの挿入が開始されます。MSSMでは、全表スキャンによってHWMより下のすべてのブロックが読み取られます。

ASSMでは空きリストは使用されないため、異なる方法で領域を管理する必要があります。セッションで最初にデータを表に挿入すると、MSSMの場合のようにブロックのグループが事前にフォーマットされるかわりに、1つのビットマップ・ブロックがフォーマットされます。このビットマップでは、セグメント内のブロックの状態が追跡されます(空きリストのかわりとなります)。データベースでは、ビットマップを使用して空きブロックが検索されて、データが格納される前に各ブロックがフォーマットされます。ASSMを使用すると、挿入処理が各ブロックに分散されるため、同時実行性の問題を回避できます。

ASSMセグメント内の各データ・ブロックは、次のいずれかの状態になっています。

- HWMより上

  これらのブロックは未フォーマットであり、使用されていません。

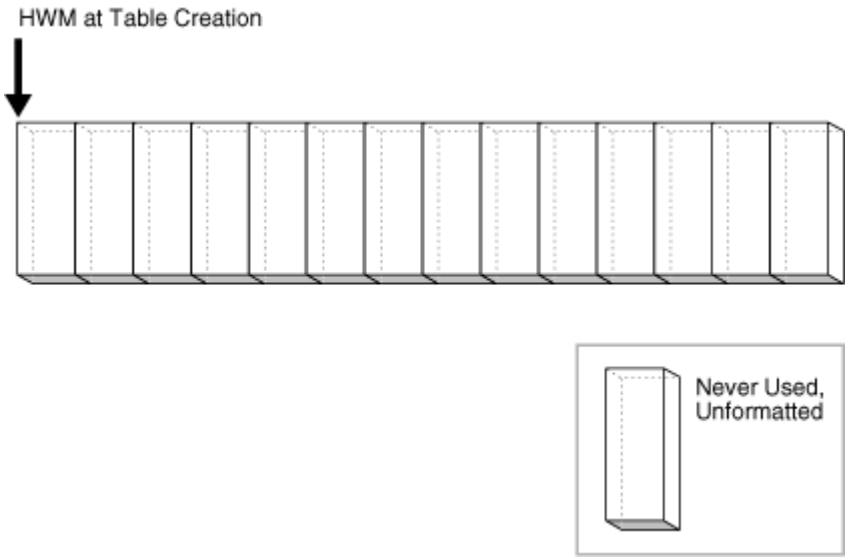
- HWMより下

  これらのデータ・ブロックは、次のいずれかの状態になっています。

  - 割当て済ですが、現時点では未フォーマットで使用されていません。
  - フォーマット済でデータが格納されています。
  - フォーマット済ですが、データが削除されたため空です。

図12-24に、ブロックを水平方向に並べて表現したASSMのセグメントを示します。表の作成時には、HWMはセグメントの先頭(左端)にあります。まだデータは挿入されていないため、セグメント内のすべてのブロックは未フォーマットで、まだ使用されたことはありません。

図12-24 表作成時のHWM



トランザクションでセグメントに行が挿入されるとします。Oracle Databaseにより、これらの行を保持するためのブロックのグループが割り当てられます。割り当てられたブロックは、HWMに達していません。データベースでは、このグループ内のビットマップ・ブロックはフォーマットされてメタデータが保持されますが、グループ内の他のブロックは事前にフォーマットされません。

図12-25では、HWMより下のブロックは割り当てられていますが、HWMより上のブロックは割当てもフォーマットも行われません。挿入が行われるときに、データベースは空き領域がある任意のブロックに書き込みます。低い最高水位標(低いHWM)は、この位置より下のすべてのブロックは、データが格納されているか、以前にデータが格納されていたため、フォーマット済であることを示しています。

図12-25 HWMと低いHWM

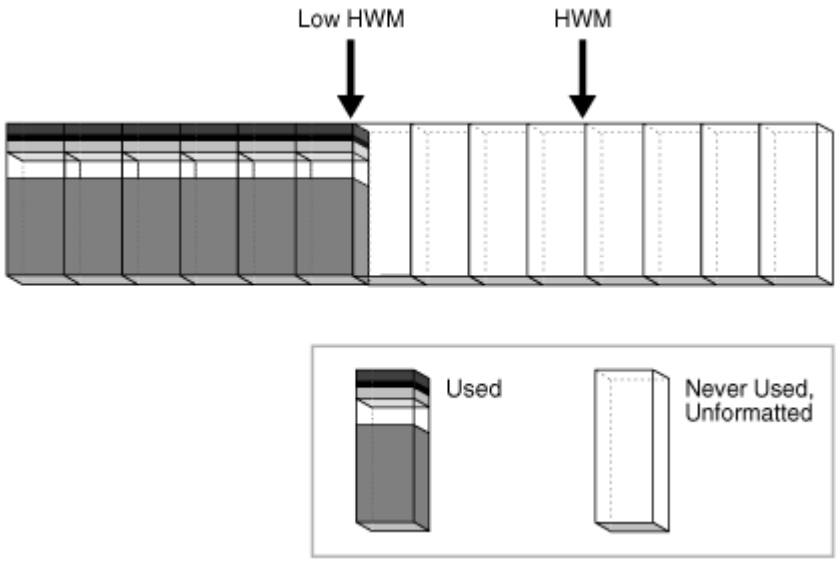
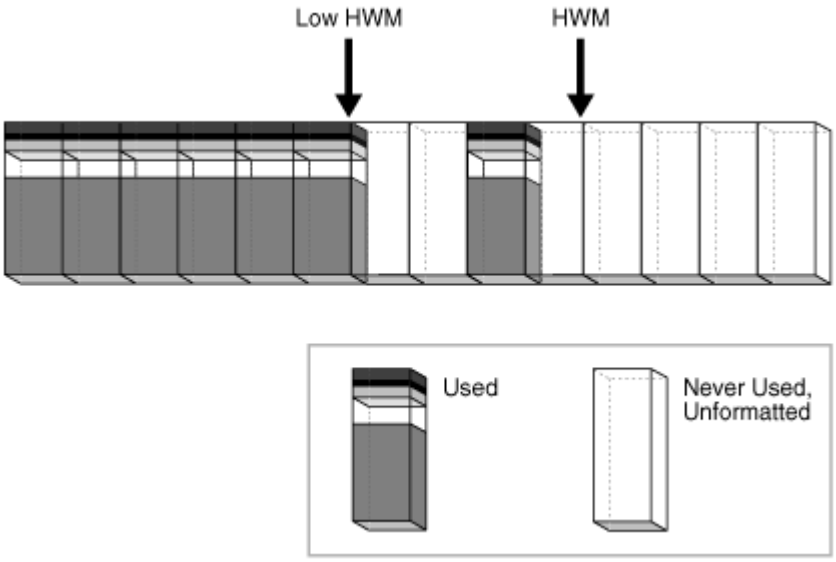


図12-26に示すように、データベースはHWMと低いHWMの間のブロックを選択し、そのブロックに書き込みます。データベースが書込み対象として選択するブロックは、HWMと低いHWMの間のブロックになることもあれば、低いHWMよりも低い、空き領域のあるブロックになることもあります。図12-26に示すように、新しくいっぱいになったブロックの両側のブロックは、未フォーマットになっています。

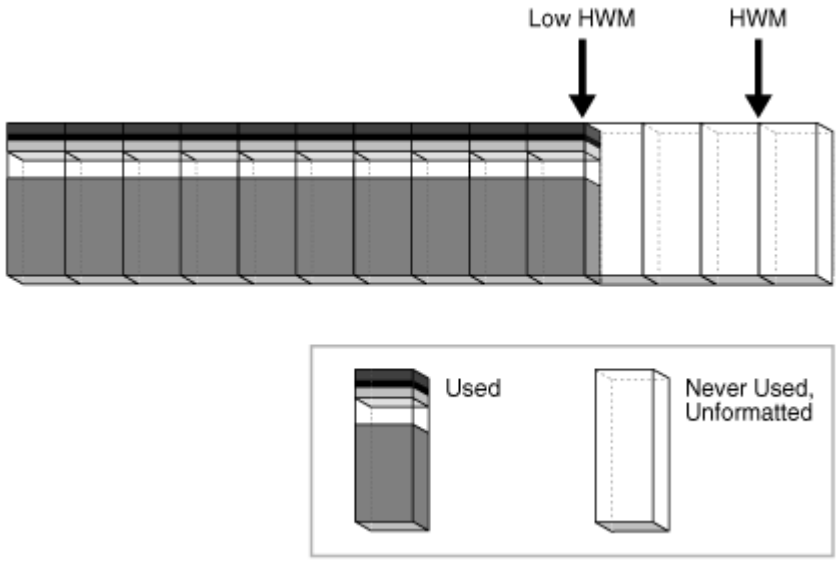
図12-26 HWMと低いHWM



全表スキャン時には、低いHWMが重要になります。HWMより下のブロックは使用時にのみフォーマットされるため、図12-26に示すように、いくつかのブロックは未フォーマットのままになる可能性があります。このため、Oracle Databaseでは、低いHWMの位置を取得するためにビットマップ・ブロックが読み取られます。低いHWMまでのブロックはフォーマット済であることが既知であるためすべて読み取られ、次に、低いHWMとHWMの間にあるフォーマット済ブロックのみが選択されて読み取られます。

新しいトランザクションによって表に行が挿入されるときに、HWMより下に十分な空き領域が存在しないことがビットマップから判明したとします。この場合は、図12-27に示すように、HWMが右側に移動されて、未フォーマット・ブロックのグループが新しく割り当てられます。

図12-27 HWMと低いHWMの拡張



HWMと低いHWMとの間のブロックに空きがなくなると、HWMが右側に拡張されて、低いHWMも以前のHWMの場所まで拡張されます。データベースにデータが徐々に挿入されるにつれて、HWMは次第に右側へと拡張され、それに伴い、低いHWMもHWMに従って拡張されます。オブジェクトを手動で再構築、切捨てまたは縮小しないかぎり、HWMが縮小することはありません。

関連項目:

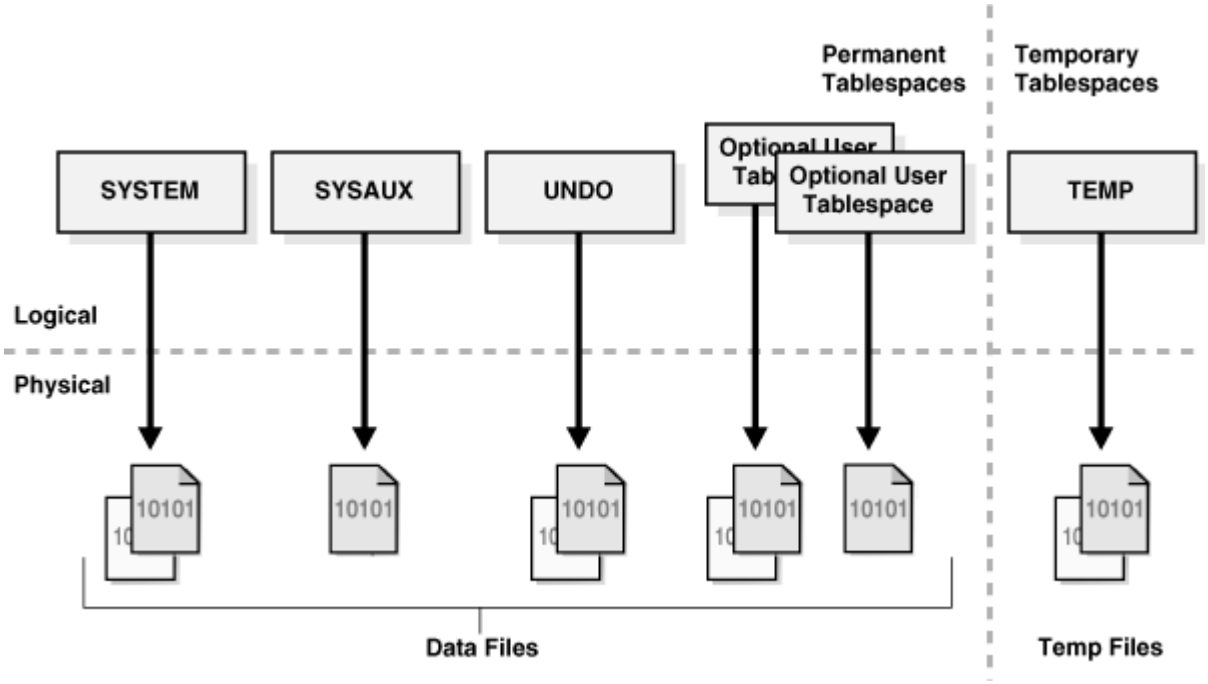
- オンラインでセグメントを縮小する方法を学習するには、Oracle Database管理者ガイドを参照してください
- TRUNCATE TABLEの構文およびセマンティクスについては、Oracle Database SQL言語リファレンスを参照してください

表領域の概要

表領域は、セグメントを格納するための論理記憶域コンテナです。セグメントは表や索引などのデータベース・オブジェクトであり、記憶域領域を使用します。物理レベルでは、表領域のデータは、1つ以上のデータファイルまたは一時ファイルに格納されます。

データベースには、SYSTEM表領域とSYSAUX表領域が必要です。次の図に、一般的なデータベースの表領域を示します。次の各項では、表領域タイプについて説明します。

図12-28 表領域



永続表領域

永続表領域には、永続スキーマ・オブジェクトがまとめられます。永続表領域内のオブジェクトのセグメントは、物理的にはデータファイルに格納されます。

各データベース・ユーザーには、デフォルトの永続表領域が割り当てられます。非常に小規模なデータベースでは、デフォルトのSYSTEM表領域およびSYSAUX表領域のみで十分な場合があります。ただし、ユーザー・データおよびアプリケーション・データを格納するための表領域を1つ以上作成することをお薦めします。表領域を使用して次の目的を達成できます。

- データベース・データに対するディスク領域の割当てを制御します。
- データベース・ユーザーに割当て制限(領域の許容量または制限)を割り当てます。
- データベース全体の可用性に影響を与えることなく、個別の表領域をオンライン化またはオフライン化します。
- 個別の表領域のバックアップとリカバリを実行します。
- Oracle Data Pumpユーティリティを使用したアプリケーション・データのインポートまたはエクスポート
- データベース間、またはプラットフォーム間でコピーし、移動できるトランスポータブル表領域を作成します。

  表領域を移動する場合は、データファイルをコピーして表領域のメタデータを統合するのみでよいため、この方法によるデータ移動は同じデータをエクスポート/インポートまたはアンロード/ロードするよりも何倍も高速です。表領域を移動する場合は、索引データも移動できます。

関連項目:

- 「Oracle Data Pumpエクスポートおよびインポート」
- 表領域の移動方法を学習するには、『Oracle Database管理者ガイド』を参照してください。
- Oracle Data Pumpについて学習するには、『Oracle Databaseユーティリティ』を参照してください。

SYSTEM表領域

SYSTEM表領域は、データベースの作成時に生成される、必須の管理表領域です。Oracle Databaseでは、データベースの管理にSYSTEMが使用されます。

SYSTEM表領域には、次の情報が含まれています。これらの情報は、すべてSYSユーザーが所有しています。

- データ・ディクショナリ
- データベースの管理情報を含む表およびビュー
- トリガー、プロシージャ、パッケージなどのコンパイル済ストアド・オブジェクト

SYSTEM表領域は、他のすべての表領域と同様に管理されますが、より高いレベルの権限が必要であり、また一部制限事項があります。たとえば、SYSTEM表領域の名前変更や削除はできません。

デフォルトで、Oracle Databaseでは、新しく作成されたすべてのユーザー表領域がローカル管理に設定されます。ローカル管理のSYSTEM表領域を持つデータベースには、ディクショナリ管理表領域を作成できません(非推奨)。ただし、手動でCREATE DATABASE文を実行してデフォルトを受け入れると、SYSTEM表領域はディクショナリ管理になります。既存のディクショナリ管理SYSTEM表領域をローカル管理形式に移行できます。

> ノート:
>
> SYSTEM 表領域を含むすべての表領域がデフォルトでローカル管理になるように、Database Configuration Assistant(DBCA)を使用して新しいデータベースを作成することをお薦めします。

関連項目:

- SYSTEM表領域の永続的なオンライン条件の詳細は、「オンライン表領域とオフライン表領域」を参照してください
- DBCAについて学習するには、「データベースのインストールおよび構成ツール」を参照してください
- ローカル管理のSYSTEM表領域を作成する方法、またはローカル管理のSYSTEM表領域に移行する方法を学習するには、『Oracle Database管理者ガイド』を参照してください
- CREATE DATABASEの構文およびセマンティクスについては、『Oracle Database SQL言語リファレンス』を参照してください

SYSAUX表領域

SYSAUX表領域は、SYSTEM表領域の補助表領域です。

SYSAUXは、以前に固有の表領域を必要としていたOracle Databaseの多くの機能と製品に対するデフォルトの表領域であるため、データベースに必要な表領域の数が削減されます。また、SYSTEM表領域の負荷も軽減されます。

データベースの作成または更新により、自動的にSYSAUX表領域が作成されます。通常のデータベース操作中には、SYSAUX表

領域の削除や名前変更は許可されません。SYSAUX表領域が使用できなくなった場合でも、中核となるデータベース機能は利用できます。SYSAUX表領域を使用するデータベース機能は、エラーが発生したり、機能が制限されることがあります。

関連項目:

SYSAUX表領域について学習するには、Oracle Database管理者ガイドを参照してください

UNDO表領域

UNDO表領域は、システムが管理しているUNDOデータ用に予約されているローカル管理表領域です。

他の永続表領域と同様、UNDO表領域にもデータファイルが含まれています。これらのファイル内のUNDOブロックは、エクステントにグループ化されています。

関連項目:

UNDOセグメント

自動UNDO管理モード

UNDO表領域を使用する場合は、データベースが**デフォルトの自動UNDOモード**にあることが必要です。

自動モードでは、手動でUNDOセグメントを管理する複雑な作業が不要です。UNDOデータを可能なかぎり最適に保存して、UNDOデータを必要とする長時間実行される問合せに対応できるように、データベース自体が自動的にチューニングされます。

Oracle Databaseを新しくインストールすると、UNDO表領域が自動的に作成されます。Oracle Databaseの以前のバージョンでは、UNDO表領域が含まれておらず、かわりに手動UNDO管理モードと呼ばれる従来のロールバック・セグメントが使用される場合があります。Oracle Database 11g以降にアップグレードすると、自動UNDO管理モードを有効にして、UNDO表領域を作成できます。Oracle Databaseには、UNDO環境についてのアドバイスを提供し、UNDO環境の自動化に役立つUNDOアドバイザが含まれています。

データベースに複数のUNDO表領域を含めることができますが、使用中となるのは常に1つのUNDO表領域のみです。インスタンスによってデータベースのオープンが試行されると、Oracle Databaseでは、使用可能な最初のUNDO表領域が自動的に選択されます。使用可能なUNDO表領域がない場合、インスタンスはUNDO表領域なしで起動され、UNDOデータはSYSTEM表領域に格納されます。ただし、SYSTEM表領域にUNDOデータを格納しないことをお薦めします。

関連項目:

- 自動UNDO管理について学習するには、『Oracle Database管理者ガイド』を参照してください
- 自動UNDO管理モードへの移行方法を学習するには、『Oracle Databaseアップグレード・ガイド』を参照してください

自動UNDO保存

UNDO保存期間とは、Oracle Databaseで古いUNDOセグメントを上書きしないで保存する最低期間のことです。

長時間実行される問合せでは、読取り一貫性を実現するためにより古いブロック・イメージが必要になる場合があるため、

UNDOデータの保存は重要です。また、一部のOracle Flashback機能は、UNDOデータの可用性に依存しています。

一般的に、古いUNDOデータは可能なかぎり長期間保存することをお薦めします。トランザクションのコミット後、ロールバックやトランザクション・リカバリの目的にはUNDOデータは不要になります。UNDO表領域に新しいトランザクション用の領域がある場合には、古いUNDOデータを保持できます。領域が少なくなると、コミットされたトランザクションのデータによる古いUNDOデータの上書きが開始されます。

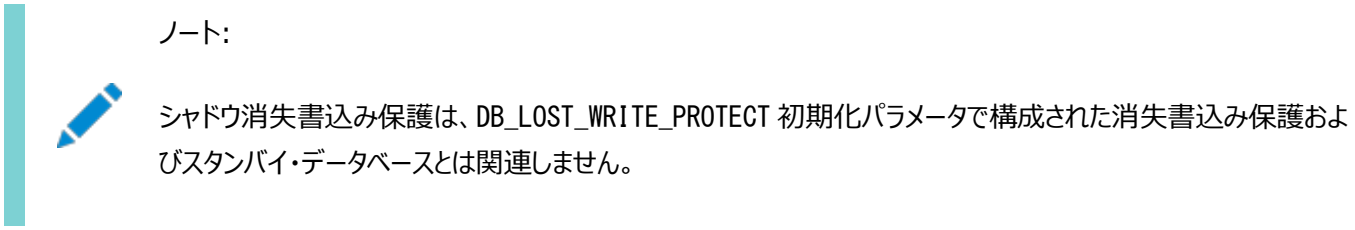
Oracle Databaseでは、現行のUNDO表領域で、可能なかぎり最適にUNDOデータが自動的に保存されます。データベースにより、使用統計が収集され、これらの統計およびUNDO表領域サイズに基づいて保存期間がチューニングされます。UNDO表領域がAUTOEXTENDオプションを使用して構成されており、最大サイズが指定されていない場合は、UNDO保存期間は異なる方法でチューニングされます。この場合には、データベースにより、実行時間が最も長い問合せより少し長くなるように、UNDO保存期間がチューニングされます(領域が許す場合)。

関連項目:

UNDO保存の自動チューニングの詳細は、『Oracle Database管理者ガイド』を参照

シャドウ表領域

シャドウ表領域は、**シャドウ書込み欠落保護**を対象とするビッグファイル表領域です。



ノート:

シャドウ消失書込み保護は、DB_LOST_WRITE_PROTECT 初期化パラメータで構成された消失書込み保護およびスタンバイ・データベースとは関連しません。

シャドウ表領域の目的

シャドウ書込み欠落保護では、**書込み欠落**の迅速な検出と即時の対応が提供されます。

データ・ブロックの書込み欠落は、書込みが実際には行われていないのにブロック書込みの完了がI/Oサブシステムで認識される場合や、ブロックの前回イメージが現在のイメージを上書きする場合に発生します。

書込み欠落を検知しないと、不適切なデータが他のDMLトランザクションに使用されるためにデータ破損が生じる可能性があります。たとえば、ある表から古い不適切なデータがトランザクションで読み取られて、このデータに基づいて他の何百という表が更新される場合があります。このようにして、データ破損がデータベース全体に広がる可能性があります。

シャドウ書込み欠落保護には次のような利点があります。

- 標準のDML、SQL*Loaderの従来型パス・ロード、ダイレクト・パス・ロードおよびRMANバックアップで使用される前に、書込み欠落が検知されます。
- Oracle Database 11gで導入された書込み欠落保護と同様に、スタンバイ・データベースが不要です。
- 特定の表領域およびデータ・ファイルについてシャドウ書込み欠落保護を有効化できます。すべてのデータを追跡する必要はありません。
- あるシャドウ表領域を別のシャドウ表領域に置き換えて、その構成や場所を変更できます。
- 表領域またはデータ・ファイルに対してシャドウ書込み欠落保護を中断および再開できます。

- ALTER DATABASE ... LOST WRITE TRACKING文単体で非CDBまたはPDBを全体的に有効または無効にできます。PROP$表は、PDBに対してトラッキングが有効かどうかを示す点に注意してください。

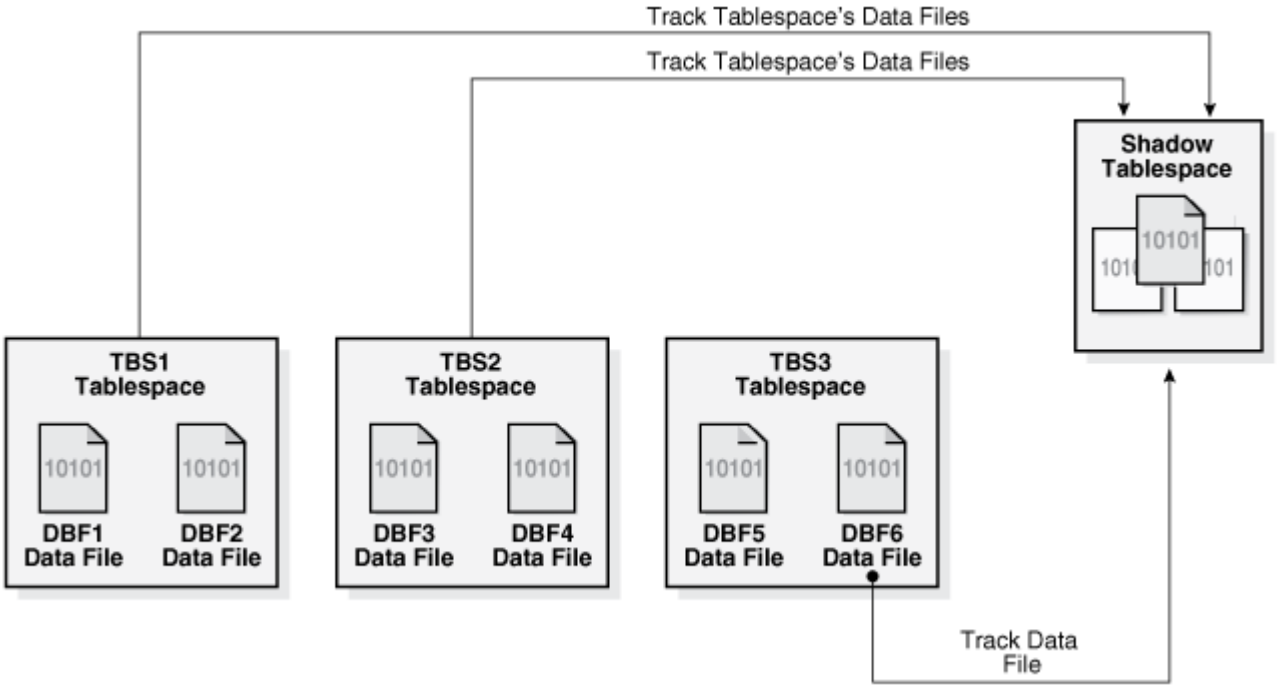
関連項目:

LOST WRITE TRACKING句についてさらに学習するには、Oracle Database SQL言語リファレンスを参照してください。

シャドウ表領域の機能

書込み欠落保護ではシャドウ表領域と、シャドウ表領域によって追跡されるブロックを持つ非シャドウ表領域の2つの表領域が必要です。

次の図は、サンプル・シナリオを示しています。表領域TBS1およびTBS2内のデータ・ファイルがシャドウ表領域によって追跡されます。表領域TBS3内のデータ・ファイルDBF6のみがシャドウ表領域によって追跡されます。



1つの追跡データ・ファイルがシャドウ表領域の1つのシャドウ・エクステントにマッピングされます。追跡データ・ファイル内のすべてのデータ・ブロックは、シャドウ・ブロック内に対応するエントリを持ちます。このエントリには、追跡データ・ブロックのSCNが含まれます。追跡データ・ブロックがディスクから読み取られる際、シャドウ書込み欠落保護ではシャドウ表領域内のブロックのSCNと、追跡データ・ブロック内の最近の書込みのSCNが比較されます。シャドウ・エントリのSCNが読取り中のデータ・ブロックを上回る場合、書込み欠落が起こってエラーが発生します。

データファイルの自動サイズ変更によってシャドウ・エクステントが大きくなりすぎるのを防ぐために、シャドウ・エクステントはかなりの追加領域を持つサイズに設定されます。追跡データ・ファイルが手動または自動でサイズ変更された場合、およびシャドウ・エクステントを大きくする必要がある場合、データベースはトラッキング・データのサイズを変更しようとします。シャドウ表領域に十分な領域が存在しない場合、データベースはアラート・ログに警告を書き込み、可能な限りの数のデータ・ブロックを追跡します。

関連項目:

シャドウ書込み欠落保護の管理方法を学習するには、Oracle Database管理者ガイドを参照してください。

シャドウ表領域のユーザー・インタフェース

シャドウ書込み欠落保護は、ALTER DATABASEコマンドを使用して有効化および無効化します。

シャドウ書込み欠落保護で特定の表領域またはデータ・ファイルを保護するためには、次の条件が満たされる必要があります。

- ALTER DATABASE ENABLE LOST WRITE PROTECTION文を使用して、非CDBまたはPDBの全体に対してシャドウ書込み欠落保護を有効化しておく必要があります。

ノート:

CDB では、シャドウ書込み欠落保護をルートで有効化した場合は、PDB はそれを継承しません。保護対象の PDB すべてでシャドウ書込み欠落保護を有効にする必要があります。

- ENABLE LOST WRITE PROTECTION句を使用して、保護対象の表領域またはデータ・ファイルに対してシャドウ書込み欠落保護を有効化しておく必要があります。

  表領域に対してシャドウ書込み欠落保護を有効化すると、表領域のすべてのデータ・ファイルが保護され、表領域に追加されたデータ・ファイルもすべて保護されます。一時表領域または別の書込み消失表領域では書込み欠落保護を有効にすることはできません。

- LOST WRITE PROTECTION句を含むCREATE BIGFILE TABLESPACE文を使用して、シャドウ表領域を1つ以上作成しておく必要があります。

Oracle Databaseでは、追跡されたデータ・ファイルが特定のシャドウ表領域に自動的に割り当てられます。特定のデータ・ファイルに使用されるシャドウ表領域を指定することはできません。

次のデータ・ディクショナリ・ビューでは、シャドウ表領域を監視します。

- DBA_TABLESPACES

  問合せによりどの表領域がシャドウ表領域かを示します。

- DBA_DATA_FILES.LOST_WRITE_PROTECT

  書込み欠落保護がデータファイルに対して有効になっているかどうかを示します

- USER_TABLESPACES.LOST_WRITE_PROTECT

  特定の表領域に対して書込み欠落保護が有効になっているかどうかを示します。DBA_DATA_FILESは、表領域で書込み欠落保護がオンになっているかどうかを示しません。かわりに、USER_TABLESPACESを参照する必要があります。

関連項目:

- データ破損
- シャドウ表領域の管理方法を学習するには、『Oracle Database管理者ガイド』を参照
- CREATE TABLESPACE文について学習するには、『Oracle Database SQL言語リファレンス』を参照
- DBA_TABLESPACESについて学習するには、『Oracle Databaseリファレンス』を参照

例: 書込み欠落保護の構成

この例では、一連の表領域に対してシャドウ書込み欠落保護を有効化する方法を示します。

この例での目的は、非CDB内のsalestbsおよびhrtbs表領域を保護することです。また、oetbs領域内のoetbs01.dbfデータファイルのみを保護します。次の操作を行います。

1. SYSTEMとしてデータベースにログインします。
2. 1つのシャドウ表領域を次のように作成します。

```
CREATE BIGFILE TABLESPACE shadow_lwp1
  DATAFILE 'shadow_lwp1_df' SIZE 10M LOST WRITE PROTECTION;
```

3. データベース全体に対する書込み欠落保護を次のように有効化します。

```
ALTER DATABASE ENABLE LOST WRITE PROTECTION;
```

4. salestbsおよびhrtbs表領域に対するシャドウ書込み欠落保護を次のように有効化します。

```
ALTER TABLESPACE salestbs ENABLE LOST WRITE PROTECTION;
ALTER TABLESPACE hrtbs ENABLE LOST WRITE PROTECTION;
```

5. 次のように、oetbs01.dbfデータファイルのシャドウ書込み欠落保護を有効にします。

```
ALTER DATABASE DATAFILE 'oetbs01.dbf' ENABLE LOST WRITE PROTECTION;
```

関連項目:

- シャドウ表領域の管理方法を学習するには、『Oracle Database管理者ガイド』を参照
- CREATE TABLESPACE文について学習するには、『Oracle Database SQL言語リファレンス』を参照

一時表領域

一時表領域には、セッションの存続期間中のみ保持される一時データが含まれています。永続スキーマ・オブジェクトを一時表領域に格納することはできません。**一時ファイル**には一時的な表領域データが格納されます。

一時表領域によって、メモリー内に収まらない複数のソート操作の同時実行性が向上します。また、一時表領域によって、ソート中の領域管理操作の効率も向上します。

共有およびローカル一時表領域

一時表領域は共有またはローカルのいずれかになります。

共有一時表領域は共有ディスク上に一時ファイルを格納するため、一時領域はすべてのデータベース・インスタンスにとってアクセス可能です。対照的に、ローカル一時表領域はデータベース・インスタンスごとに個別の非共有一時ファイルを格納します。ローカル一時表領域は、Oracle Real Application ClustersまたはOracle Flex Clustersに対して有効です。

ノート:

ローカル一時表領域は Oracle Database 12c リリース 2 (12.2)の新機能です。前のリリースでは、共有一時表領域が単に一時表領域と呼ばれていました。このリリース以降では、特に記載がないかぎり、一時表領域という用語は共有一時表領域を意味します。

読取り専用および読取り/書込みの両方のデータベース・インスタンスについて、ローカル一時表領域を作成できます。多数の

読取り専用インスタンスが1つのデータベースにアクセスする際、ソート、ハッシュ集計および結合が含まれる問合せのパフォーマンスがローカル一時表領域によって向上します。この利点は次のとおりです。

- 共有ではなくローカルのディスク記憶域を使用することによる、I/Oパフォーマンスの向上
- コストの高いインスタンス間の一時領域管理の回避
- ディスク上の領域メタデータ管理の排除による、インスタンスの起動パフォーマンスの向上

次の表では、共有およびローカル一時表領域の各特性を比較します。

表12-2 共有およびローカル一時表領域

共有一時表領域	**ローカル一時表領域**
CREATE TEMPORARY TABLESPACE 文によって作成されます。	CREATE LOCAL TEMPORARY TABLESPACE 文によって作成されます。 **ノート:** ローカル一時表領域は常に bigfile 表領域ですが、作成文に BIGFILE キーワードは必要ありません。
データベースの単一の一時表領域を作成します。	データベース・インスタンスごとに個別の一時表領域を作成します。FOR LEAF オプションは、読取り専用インスタンスのみの表領域を作成します。FOR ALL オプションは、読取り専用および読取り/書込みの両方のすべてのインスタンスの表領域を作成します。
表領域グループをサポートしています。	表領域グループをサポートしていません。
一時ファイル・メタデータを制御ファイルに格納します。	すべてのインスタンスに共通の一時ファイル・メタデータを制御ファイルに格納して、インスタンス固有のメタデータ(例: 割当て用ビットマップ、現在の一時ファイル・サイズ、ファイル・ステータス)を SGA に格納します。

関連項目:

「Oracleデータベース・インスタンスの概要」

デフォルト一時表領域

各データベース・ユーザー・アカウントには、デフォルトの共有一時表領域が割り当てられます。データベースにローカル一時表領域が含まれる場合、各ユーザー・アカウントにはデフォルトのローカル一時記憶域も割り当てられます。

CREATE USER文またはALTER USER文によってユーザー・アカウントに異なる一時表領域を指定できます。Oracle Databaseでは、システムレベルのデフォルト一時表領域を、別の一時表領域を指定していないユーザーに対して使用します。

関連項目:

CREATE USER文についてさらに学習するには、『Oracle Database SQL言語リファレンス』を参照

デフォルト一時表領域の作成

データベースを作成する際、デフォルトの一時記憶域はSYSTEM表領域がローカルに管理されるかどうかによって異なります。

次の表に、データベース作成時にデフォルトの一時表領域がOracle Databaseでどのように選択されるかを示します。

表12-3 デフォルト一時表領域の作成

SYSTEM表領域がローカルに管理されていますか。	CREATE DATABASE文がデフォルトの一時表領域を指定しますか。	データベースの次の動作
はい	はい	指定された表領域をデフォルトとして使用します。
はい	いいえ	一時表領域を作成します。
いいえ	はい	指定された表領域をデフォルトとして使用します。
いいえ	いいえ	デフォルトの一時記憶域に SYSTEM を使用します。データベースによってアラート・ログに、デフォルトの一時表領域を推奨する警告が書き込まれます。

データベースの作成後、ALTER DATABASE DEFAULT TEMPORARY TABLESPACE文によってデータベースのデフォルト一時表領域を変更できます。

ノート:

デフォルトの一時表領域を永続表領域にすることはできません。

関連項目:

- 「永続データファイルと一時データファイル」
- デフォルトの一時表領域の作成方法を学習するには、『*Oracle Database*管理者ガイド』を参照してください。
- CREATE DATABASEおよびALTER DATABASEのDEFAULT TEMPORARY TABLESPACE句の構文の詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

一時記憶域へのアクセス

ユーザーに一時表領域が割り当てられている場合、データベースは最初にそれにアクセスします。それ以外の場合、データベースはデフォルト一時表領域にアクセスします。データベースは問合せのために一時表領域にアクセスした後、別の表領域には切り替えません。

ユーザー問合せは共有またはローカルのいずれかの一時記憶域にアクセスできます。さらに、ユーザーは読取り専用インスタンスに対して1つのデフォルトのローカル一時表領域を割り当てて、読取り/書込みインスタンスに対して別のデフォルトのローカル一時表領域を割り当てることができます。

読取り/書込みインスタンスの場合、データベースは共有一時表領域の優先度を高くします。読取り専用インスタンスの場合、データベースはローカル一時表領域の優先度を高くします。データベース・インスタンスが読取り/書込みの場合、データベースは

次の順序で領域を検索します。

1. ユーザーに共有一時表領域が割り当てられていますか。
2. ユーザーにローカル一時表領域が割り当てられていますか。
3. データベースのデフォルト一時表領域に領域がありますか。

前述の質問のいずれかに対する回答が「はい」の場合、データベースは検索を停止し、指定された表領域から領域を割り当てます。それ以外の場合は、データベースのデフォルトのローカル一時表領域から領域が割り当てられます。

データベース・インスタンスが読取り専用の場合、データベースは次の順序で領域を検索します。

1. ユーザーにローカル一時表領域が割り当てられていますか。
2. 割り当てられたデータベースのデフォルトのローカル一時表領域に領域がありますか。
3. ユーザーに共有一時表領域が割り当てられていますか。

前述の質問のいずれかに対する回答が「はい」の場合、データベースは検索を停止し、指定された表領域から領域を割り当てます。それ以外の場合は、データベースのデフォルトの共有一時表領域から領域が割り当てられます。

表領域モード

表領域モードによって、表領域のアクセス可能性が決まります。

読取り/書込み表領域と読取り専用表領域

すべての表領域は、書込み可能であるかどうかを指定する書込みモードのうちいずれかのモードになっています。

次の相互に排他的なモードがあります。

- 読取り/書込みモード

  ユーザーは、表領域の読取りおよび書込みが可能です。すべての表領域は、最初は読取り/書込み用として作成されます。SYSTEM表領域とSYSAUX表領域、および一時表領域は永続的に読取り/書込みモードであるため、読取り専用にはできません。

- 読取り専用モード

  表領域内のデータファイルへの書込み操作を行うことができません。読取り専用表領域は、DVDドライブやWORMドライブなどの読取り専用メディア上にのみ存在します。

  読取り専用表領域では、データベースの静的な部分を大量にバックアップしたり、リカバリする必要がなくなります。読取り専用表領域は変更されないため、繰り返しバックアップする必要はありません。メディア障害後にデータベースをリカバリする場合、読取り専用表領域はリカバリする必要がありません。

関連項目:

- 表領域を読取り専用または読取り/書込みモードに変更する方法を学習するには、『Oracle Database管理者ガイド』を参照してください
- ALTER TABLESPACEの構文およびセマンティクスについては、『Oracle Database SQL言語リファレンス』を参照してください
- リカバリの詳細は、『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

オンライン表領域とオフライン表領域

表領域は、データベースがオープンされているときはいつでも、オンライン(アクセス可能)またはオフライン(アクセス不能)にできます。

表領域は通常はオンラインになっており、ユーザーはその表領域内のデータを使用できます。SYSTEM表領域と一時表領域は、オフラインにはできません。

表領域は、自動または手動でオフラインにできます。たとえば、メンテナンスまたはバックアップとリカバリのために表領域をオフラインにできます。データベース・ライター(DBW)プロセスによるデータファイルへの書込みが数回失敗した場合など、特定のエラーが発生した場合には、表領域は自動的にオフラインになります。オフライン表領域にある表にアクセスしようとしたユーザーは、エラーを受け取ります。

表領域がオフラインになると、データベースで次のような処理が実行されます。

- データベースでは、表領域がオフラインになると、その表領域内のオブジェクトをDML文で参照できなくなります。Oracle Database以外のユーティリティでは、オフライン表領域の読取りまたは編集はできません。
- その表領域内のデータを参照する文が完了しているアクティブなトランザクションは、トランザクション・レベルでは影響を受けません。
- これらの完了した文に対応するUNDOデータは、SYSTEM表領域内の遅延UNDOセグメントに保存されます。表領域が再びオンライン化されたとき、必要に応じてUNDOデータが表領域に適用されます。

関連項目:

- 「オンライン・データファイルとオフライン・データファイル」
- 「データベース・ライター・プロセス(DBW)」
- 表領域の可用性を変更する方法を学習するには、『Oracle Database管理者ガイド』を参照してください

表領域ファイル・サイズ

表領域には、**bigfile表領域**と**smallfile表領域**があります。これらの表領域は、データファイルまたは一時ファイルを明示的に参照しないSQL文の実行時には区別できません。

これらの表領域の違いは、次のとおりです。

- smallfile表領域には、複数のデータファイルや一時ファイルを含むことができますが、bigfile表領域ほど大きなファイルは含むことができません。これがデフォルトの表領域タイプです。
- bigfile表領域には、1つの非常に大きなデータファイルまたは一時ファイルを含むことができます。このタイプの表領域には、次のような利点があります。
    - データベースの記憶域容量の増加

      データベース内のデータファイルの最大数には制限があるため、各データファイルのサイズを増加することによって、全体的な記憶域容量を増加できます。
    - 数多くのデータファイルおよび一時ファイルを管理する負担の軽減

      bigfile表領域により、新規ファイルを追加したり複数ファイルを取り扱う必要がなくなり、Oracle Managed Filesと自動ストレージ管理(Oracle ASM)によるファイル管理が簡素化されます。

- 個々のファイルではなく表領域に対する操作の実行

  bigfile表領域では、表領域はディスク領域の管理、バックアップおよびリカバリなどの主要単位となります。

bigfile表領域がサポートされるのは、ASSMを使用するローカル管理表領域の場合のみです。ただし、ローカル管理のUNDO表領域と一時表領域は、セグメントが手動管理される場合にもbigfile表領域として使用できます。

関連項目:

- バックアップおよびリカバリ
- bigfile表領域の管理方法を学習するには、『Oracle Database管理者ガイド』を参照してください

第V部 Oracleインスタンス・アーキテクチャ

この部では、Oracleデータベース・インスタンスの基本的な構造上のアーキテクチャについて説明します。

この部は、次の章で構成されています。

- Oracleデータベース・インスタンス
- メモリー・アーキテクチャ
- プロセス・アーキテクチャ
- アプリケーションとOracle Net Servicesのアーキテクチャ

13 Oracleデータベース・インスタンス

この章では、Oracleデータベース・インスタンスの性質、インスタンスに関連付けられたパラメータ・ファイルと診断ファイル、およびインスタンスの作成時やデータベースのオープン時とクローズ時に発生する処理について説明します。

この章の構成は、次のとおりです。

- Oracleデータベース・インスタンスの概要
- データベース・インスタンスの起動と停止の概要
- チェックポイントの概要
- インスタンス・リカバリの概要
- パラメータ・ファイルの概要
- 診断ファイルの概要

Oracleデータベース・インスタンスの概要

データベース・インスタンスは、データベース・ファイルを管理する一連のメモリー構造です。

物理レベルでは、CDBはCREATE DATABASE文によって作成されたディスク上のファイルのセットです。CDBには、ユーザーが作成した1つ以上のPDBが含まれています。PDBには、CDBに属するデータ・ファイルの全体的なセット内に、独自のデータ・ファイルのセットが含まれています。データベース・インスタンスは、CDBとそのPDBに関連付けられたデータを管理し、ユーザーに提供します。

稼働中のすべてのCDBは、1つ以上のOracleデータベース・インスタンスに関連付けられます。インスタンスはメモリー内に存在し、データベース(最も狭い意味で)はディスク上のファイルのセットであるため、インスタンスがデータベースなしで存在することも、データベースがインスタンスなしで存在することも可能です。

データベース・インスタンスの構造

インスタンスが起動されると、Oracle Databaseによって**システム・グローバル領域(SGA)**と呼ばれるメモリー領域が割り当てられ、1つ以上の**バックグラウンド・プロセス**が起動されます。

SGAは、次のような様々な目的で機能します。

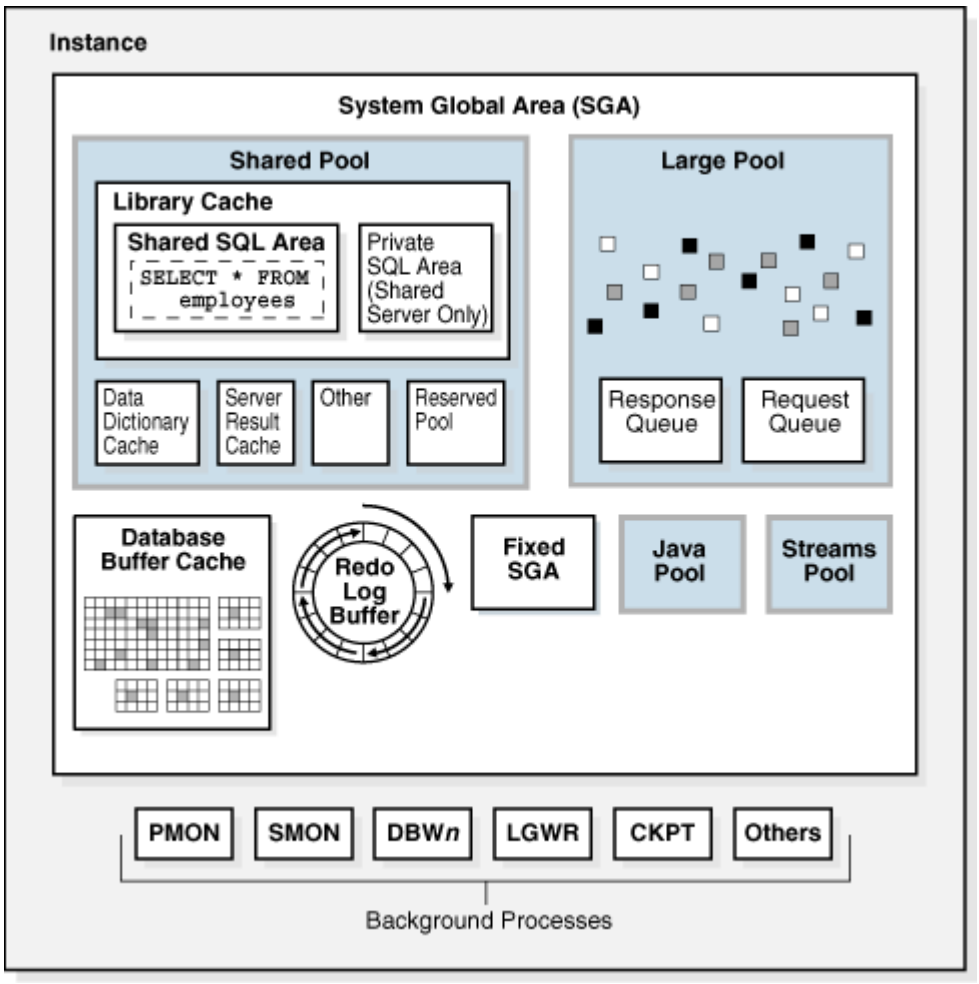
- 多くのプロセスおよびスレッドから同時にアクセスされる内部データ構造の維持
- ディスクから読み取られるデータ・ブロックのキャッシュ
- オンラインREDOログ・ファイルに書き込まれる前のREDOデータのバッファリング
- SQL実行計画の格納

1つのコンピュータで実行中の複数のOracleプロセスは、SGAを共有します。OracleプロセスがSGAと関連付けられる方法は、オペレーティング・システムによって異なります。

データベース・インスタンスにはバックグラウンド・プロセスが含まれています。サーバー・プロセスおよびサーバー・プロセスに割り当てられたプロセス・メモリーもインスタンスに存在します。サーバー・プロセスが終了しても、インスタンスは機能し続けます。

次の図に、Oracleデータベース・インスタンスの主要なコンポーネントを示します。

図13-1 データベース・インスタンス



関連項目:

- 「システム・グローバル領域(SGA)の概要」
- 「バックグラウンド・プロセスの概要」

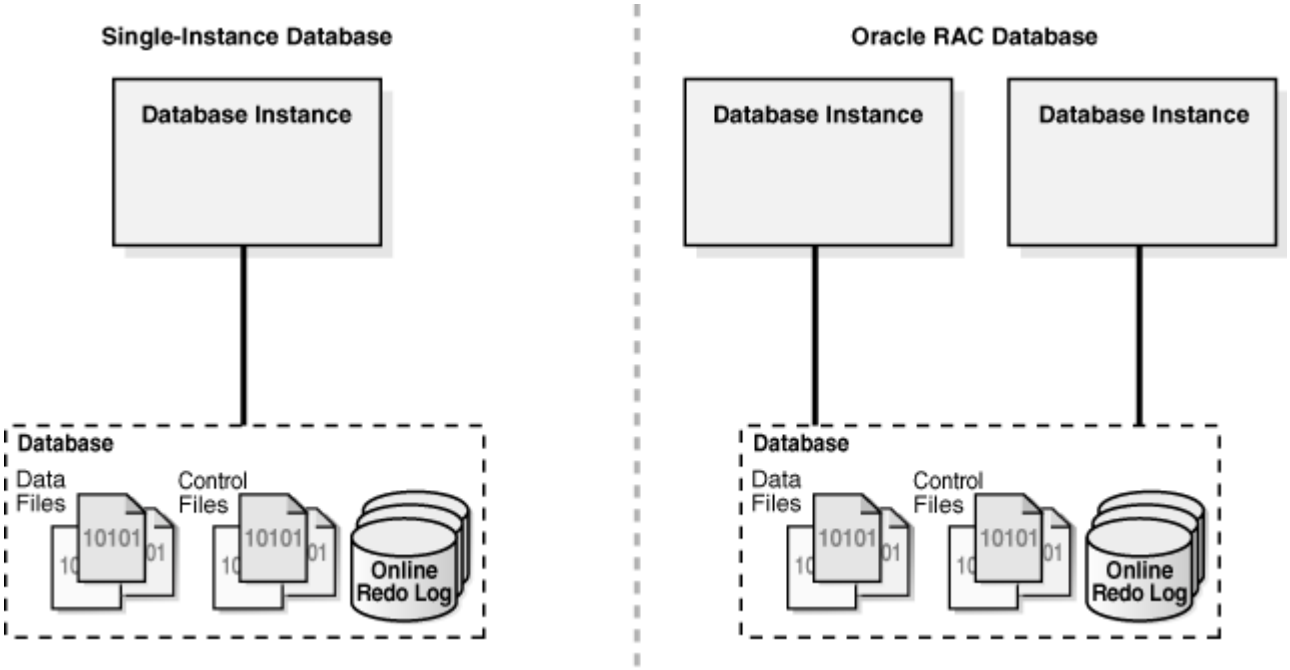
データベース・インスタンス構成

Oracle Databaseは、単一インスタンス構成またはOracle Real Application Clusters (Oracle RAC)構成のいずれかで実行されます。これらの構成は相互に排他的です。

単一インスタンス構成では、データベースとデータベース・インスタンスの間に1対1関係が存在します。Oracle RACでは、データベースとデータベース・インスタンスの間に1対多関係が存在します。

次の図に、可能なデータベース・インスタンス構成を示します。

図13-2 データベース・インスタンス構成



単一インスタンス構成とOracle RAC構成のどちらでも、1つのデータベース・インスタンスは一度に1つのデータベースにのみ関連付けられます。データベース・インスタンスを起動して1つのデータベースを**マウント**(インスタンスに関連付け)できますが、2つのデータベースを同じインスタンスに同時にマウントすることはできません。

ノート:

この章では、特に記載のないかぎり単一インスタンス・データベース構成について説明します。

複数のインスタンスを同時に同じコンピュータ上で実行し、それぞれ専用のデータベースにアクセスさせることができます。たとえば、1台のコンピュータで2つの異なるデータベースprod1とprod2をホストできます。1つのデータベース・インスタンスでprod1を管理し、別のインスタンスでprod2を管理します。

関連項目:

Oracle RACの詳細は、Oracle Real Application Clusters管理およびデプロイメント・ガイドを参照してください

読取り/書込みインスタンスと読取り専用インスタンス

データベース・インスタンスはすべて、読取り/書込みまたは読取り専用のいずれかです。

読取り/書込みデータベース・インスタンス(デフォルト)はDMLの処理が可能で、クライアント・アプリケーションからの直接接続をサポートしています。これに対して、読取り専用データベース・インスタンスは問合せを処理できますが、変更DML (UPDATE、DELETE、INSERTおよびMERGE)や直接クライアント接続をサポートしません。

ノート:

このマニュアルで特記していないかぎり、データベース・インスタンスへの参照はすべて、読取り/書込みインスタンスが対象です。

前のリリースでは、(スタンバイ・データベースへのアクセスでないかぎり)すべてのデータベース・インスタンスが読取り/書込みでした。Oracle Database 12cリリース2 (12.2)以降では、読取り専用インスタンスと読取り/書込みインスタンスが単一のデータベース内で共存できます。この構成は、データの問合せおよび変更の両方を行うパラレルSQL文に役立ちます。読取り/書込みインスタンスと読取り専用インスタンスは両方とも問合せを行う一方、読取り/書込みインスタンスは変更も行うためです。

読取り/書込みインスタンスと異なり、読取り専用インスタンスには次の特性があります。

- 読取り/書込みインスタンスによってすでに開かれているデータベースのみを開きます
- チェックポイントおよびアーカイバ・プロセスも含めて、多数の不要なバックグラウンド・プロセスを無効化します
- 無効化されたREDOスレッド、またはオンラインREDOログのないスレッドをマウントできます

インスタンスを読取り専用として指定するには、INSTANCE_MODE初期化パラメータをREAD_ONLYに設定します。パラメータのデフォルト値はREAD_WRITEです。

関連項目:

- チェックポイントおよびアーカイバ・バックグラウンド・プロセスについてさらに学習するには、「バックグラウンド・プロセスの概要」を参照
- 「オンラインREDOログの概要」
- INSTANCE_MODE初期化パラメータについてさらに学習するには、『Oracle Databaseリファレンス』を参照

データベース・インスタンスの継続時間

データベース・インスタンスは、STARTUPコマンドを使用して作成されたときに開始し、終了されたときに終了します。

この期間中、データベース・インスタンスはそのインスタンス自体を1つのデータベースにのみ関連付けることができます。さらに、そのインスタンスでは、データベースのマウント、クローズおよびオープンをそれぞれ1回のみ実行できます。データベースがクローズまたは停止された後、このデータベースをマウントしてオープンするには、別のインスタンスを起動する必要があります。

次の表に、以前にクローズされたデータベースの再オープンを試行するデータベース・インスタンスを示します。

表13-1 インスタンスの存続期間

文	説明
SQL> STARTUP ORACLE instance started. Total System Global Area 468729856 bytes Fixed Size 1333556 bytes Variable Size 440403660 bytes Database Buffers 16777216 bytes Redo Buffers 10215424 bytes Database mounted. Database opened.	STARTUP コマンドによってインスタンスが作成されます。これにより、データベースがマウントおよびオープンされます。
SQL> SELECT TO_CHAR(STARTUP_TIME,'MON-DD-RR HH24:MI:SS') AS "Inst Start Time" FROM V$INSTANCE; Inst Start Time ------------------- JUN-18-14 **13:14:48**	この問合せによって、現在のインスタンスが起動された時刻が表示されます。
SQL> SHUTDOWN IMMEDIATE	インスタンスはデータベースをクローズして停止し、そのインスタン

文	説明
	スの存続期間は終了します。
SQL> STARTUP Oracle instance started. . . .	STARTUP コマンドによって新しいインスタンスが作成され、データベースがマウントおよびオープンされます。
SQL> SELECT TO_CHAR(STARTUP_TIME, 'MON-DD-RR HH24:MI:SS') AS "Inst Start Time" FROM V$INSTANCE; Inst Start Time ------------------- JUN-18-14 13:16:40	この問合せによって、現在のインスタンスが起動された時刻が表示されます。起動時刻が異なるため、このインスタンスがデータベースを停止したインスタンスと異なることがわかります。

データベース・インスタンスの識別

複数のデータベース・インスタンスを単一のホストに配置できます。このため、アクセスするインスタンスをなんらかの方法で指定する必要があります。

Oracle Optimal Flexible Architecture (OFA)のルールは、Oracleインストールを適切に体系化するために作成された一連の構成ガイドラインです。この項の例では、OFAアーキテクチャを前提としています。

この項では、次の項目について説明します。

- Oracleベース・ディレクトリ
- Oracleホーム・ディレクトリ
- Oracleシステム識別子(SID)

関連項目:

OFAの概要については、Oracle Database インストレーション・ガイド for Linuxを参照してください

Oracleベース・ディレクトリ

Oracleベース・ディレクトリには、Oracle製品のバイナリが格納されます。

Oracleベース・ディレクトリは、Oracle Databaseインストール所有者のデータベース・ホーム・ディレクトリです。ホスト上には多数のOracle Databaseインストールおよび多数のOracle Databaseソフトウェア・インストール所有者が存在する可能性があります。

次の例では、オペレーティング・システムのユーザー・アカウントoracleのOracleベース・ディレクトリを示しています。

```
/u01/app/oracle
```

上記のパスでは、/u01/がマウント・ポイントであり、/u01/app/がアプリケーション・ソフトウェアのサブツリーです。

関連項目:

Oracleベースの概要については、Oracle Database インストレーション・ガイド for Linuxを参照してください

Oracleホーム・ディレクトリ

Oracleホームは、Oracleデータベースのソフトウェアがある場所です。

Oracle Databaseソフトウェアを新しくインストールするたびに、新しいOracleホーム・ディレクトリを指定する必要があります。デフォルトでは、Oracleホーム・ディレクトリはOracleベース(ORACLE_BASE)ディレクトリ・ツリーの子孫です。

Oracleホーム・ディレクトリが異なれば、このリリース以前のデータベース・ソフトウェアを、同じホストの単一Oracleベース内に複数回インストールできます。異なるバージョンの、異なるユーザー・アカウントが所有する複数のデータベースが同時に共存できます。

次の例は、同じOracleベース・ディレクトリ(/u01/app/oracle/)内にある、3つの異なるOracleホームのフルパス名を示しています。

```
/u01/app/oracle/product/12.1.0/dbhome_1
/u01/app/oracle/product/12.1.0/dbhome_2
/u01/app/oracle/product/18.0.0/dbhome_1
```

Oracleベース(/u01/app/oracle/)の後のパス名の一部には、製品リリース番号(たとえば、12.1.0)およびOracleホームの相対ディレクトリ(たとえば、dbhome_1)が含まれています。/u01/app/oracle/product/12.1.0/ディレクトリには2個の個別のOracleホーム、dbhome_1とdbhome_2が含まれています。

Oracle Database 18c以降では、ソフトウェア・イメージとして読取り専用のOracleホームを作成できます。読取り専用Oracleホームにはバイナリなどの静的ファイルが格納されます。ORACLE_BASE/homes/home_nameにあるOracleベース・ホーム(ORACLE_BASE_HOME)ディレクトリには、Oracleホームに固有の動的ファイルが格納されます。OracleベースのすべてのOracleホームによって共有されるOracleベース構成ディレクトリ(ORACLE_BASE_CONFIG)には、インスタンス固有の動的ファイルが格納されます。

次の例では、最初のパスは読取り専用のOracleホーム、2番目のパスはこのOracleホームのOracleベース・ホームです。

```
/u01/app/oracle/product/18.0.0/ro_dbhome_1
/u01/app/oracle/homes/ro_dbhome_1
```

関連項目:

Oracleホームの概要については、Oracle Database インストレーション・ガイド for Linuxを参照してください

Oracleシステム識別子(SID)

システム識別子(SID)とは、特定のホスト上でOracleデータベース・インスタンスを示す一意の名前です。

UNIXおよびLinuxでは、Oracle DatabaseはSIDとOracleホームの値を使用して共有メモリーに対するキーを作成します。また、Oracle Databaseはデフォルトで SIDを使用して初期化パラメータ・ファイルを検索し、これによってデータベース制御ファイルなどの関連ファイルが検索されます。

ほとんどのプラットフォームでは、ORACLE_SID環境変数によってSIDが設定され、ORACLE_HOME変数によってOracleホームが設定されます。データベース・インスタンスへの接続時に、クライアントはOracle Net接続のSIDを指定するか、ネット・サービス名を使用できます。サービス名は、Oracle DatabaseによってORACLE_HOMEとORACLE_SIDに変換されます。

従来の読取り/書込みOracleホームには、インスタンス固有のファイルが含まれています。ただし、Oracleホームが読取り専用の場合、インスタンス固有のファイルがOracleベースに別個に格納されます。どちらの場合も、名前にSIDが含まれているファイルは、Oracleベース構成ディレクトリ(ORACLE_BASE_CONFIG)のdbsサブディレクトリに格納されます。このように分離することで、ユーザーは既存の読取り専用Oracleホームのソフトウェアを使用してデータベースを作成し、新しい読取り専用のOracleホー

ムのソフトウェアを使用してそのデータベースのインスタンスを起動します。

関連項目:

- サービス名
- Oracle SIDの指定方法を学習するには、『Oracle Database管理者ガイド』を参照してください。

データベース・インスタンスの起動と停止の概要

データベース・インスタンスにより、ユーザーはデータベースにアクセスできます。インスタンスおよびデータベースを様々な状態にすることができます。

インスタンスとデータベースの起動の概要

通常、手動でインスタンスを起動してから、データベースをマウントおよびオープンし、ユーザーが使用できるようにします。SQL*Plus STARTUPコマンド、Oracle Enterprise Manager(Enterprise Manager)またはSRVCTLユーティリティを使用して、これらのステップを実行できます。

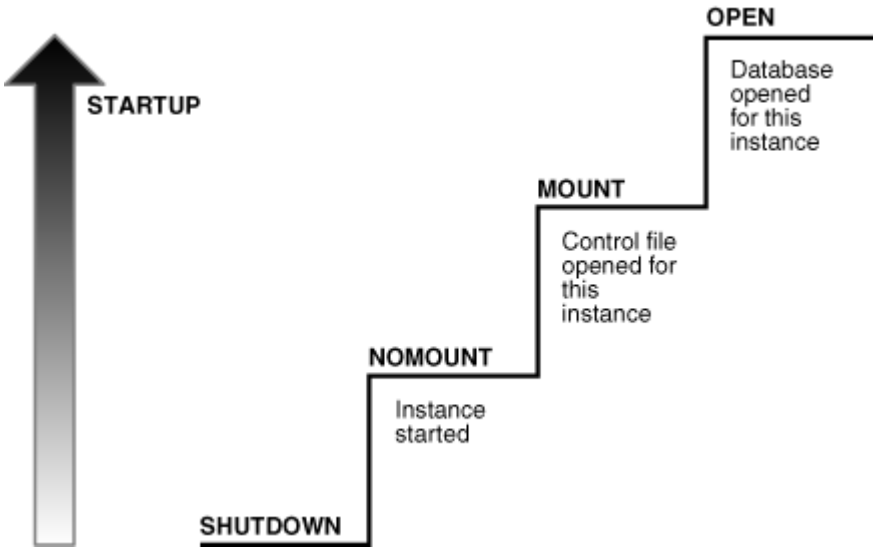
Oracle Netを使用してデータベース・インスタンスを起動するには、次の条件が満たされている必要があります。

- データベースが静的にOracle Netリスナーに登録されていること。
- 使用クライアントが、SYSDBA権限によりデータベースに接続されていること。

リスナーでは、専用サーバーが作成され、これによりデータベース・インスタンスを起動できます。

次の図に、データベースの停止状態からオープン状態までの経過を示します。

図13-3 インスタンスとデータベースの起動順序



データベースが停止状態からオープン状態になるとき、次のフェーズを経由します。

表13-2 インスタンスの起動のステップ

フェーズ	マウント状態	説明	さらに学習するには
1	データベースにマウントせずにイ	インスタンスは起動されますが、まだデータベースと関連付けられ	「インスタンスの起動方法」

フェーズ	マウント状態	説明	さらに学習するには
	ンスタンスを起動	ていません。	
2	データベースのマウント	インスタンスが起動され、データベースの制御ファイルを読み取ってデータベースと関連付けられます。データベースは、ユーザーに対してクローズされています。	「データベースのマウント方法」
3	データベースのオープン	インスタンスが起動され、オープン状態のデータベースと関連付けられます。データファイルに含まれるデータは、許可されたユーザーからアクセス可能になります。	「データベースのオープン方法」

関連項目:

- 「Oracle Net Listener」
- 「制御ファイルの概要」
- インスタンスの起動方法を学習するには、『Oracle Database管理者ガイド』を参照してください
- SRVCTLの使用方法を学習するには、『Oracle Database管理者ガイド』を参照してください

管理者権限での接続

データベースの起動と停止は、管理者権限を使用してOracle Databaseに接続するユーザーのみが実行できる強力な管理オプションです。

一般のユーザーは、Oracleデータベースの現在の状態を制御できません。ユーザーの管理者権限を確立するには、オペレーティング・システムに応じて、次のいずれかの条件が必要です。

- ユーザーのオペレーティング・システム権限により、そのユーザーが管理者権限を使用して接続できること。
- ユーザーが特別なシステム権限を付与されており、データベースがパスワード・ファイルを使用し、ネットワークを介してデータベース管理者を認証していること。

次の特別なシステム権限により、データベースがオープンされていなくても、データベースにアクセスできます。

- SYSDBA
- SYSOPER
- SYSBACKUP
- SYSDG
- SYSKM

前述の権限の制御は、データベース機能の範囲外にあります。SYSDBAシステム権限でデータベースに接続すると、そのユーザーはSYSが所有しているスキーマ内に置かれます。SYSOPERとして接続すると、そのユーザーはパブリック・スキーマ内に置かれます。

SYSOPER権限は、SYSDBA権限のサブセットです。

関連項目:

- 「SYSスキーマとSYSTEMスキーマ」
- データベース管理者を認証するときのパスワード・ファイルと認証方式について学習するには、「データベース・セキュリティの概要」を参照してください
- 管理権限の管理について学習するには、『Oracle Databaseセキュリティ・ガイド』を参照してください
- システム権限について学習するには、『Oracle Database管理者ガイド』を参照してください
- オペレーティング・システム権限グループについてさらに学習するには、『Oracle Databaseインストレーション・ガイド』を参照してください

インスタンスの起動方法

Oracle Databaseがインスタンスを起動する場合、段階的に行われます。

次のようなステージがあります。

1. プラットフォーム固有のデフォルトの格納場所でサーバー・パラメータ・ファイルを検索します。見つからなかった場合は、テキスト形式の初期化パラメータ・ファイルを検索します(STARTUPにSPFILEまたはPFILEパラメータを指定すると、デフォルトの動作が上書きされます)。
2. パラメータ・ファイルを読み取り、初期化パラメータの値を決定します。
3. 初期化パラメータの設定に基づいてSGAを割り当てます。
4. Oracleバックグラウンド・プロセスを起動します。
5. アラート・ログとトレース・ファイルをオープンし、明示的に設定されたすべてのパラメータ設定をパラメータの有効な構文でアラート・ログに書き込みます。

この時点では、このインスタンスにデータベースは関連付けられていません。NOMOUNT状態が必要とされるシナリオには、データベースの作成および特定のバックアップ操作とリカバリ操作があります。

関連項目:

サーバー・パラメータ・ファイルを使用した初期化パラメータの管理方法を学習するには、『Oracle Database管理者ガイド』を参照してください

データベースのマウント方法

インスタンスは、データベースを**マウント**して、データベースをそのインスタンスに対応付けます。

データベースをマウントすると、そのインスタンスはCONTROL_FILES初期化パラメータに指定されたデータベース制御ファイルの名前を取得し、これらのファイルをオープンします。Oracle Databaseでは、制御ファイルを読み取ることによって、データベースのオープン時にアクセスを試行するデータファイルとオンラインREDOログ・ファイルの名前を検出します。

マウントされたデータベースでは、データベースはクローズされているため、データベース管理者のみがそのデータベースにアクセスできます。管理者は、特定のメンテナンス操作を完了するまでの間、データベースをクローズしたままにしておくことができます。ただ

し、データベースに対して通常の操作を行うことはできません。

Oracle Databaseで、複数のインスタンスによる同じデータベースの同時マウントが許可されている場合、CLUSTER_DATABASE初期化パラメータの設定で、複数のインスタンスによるデータベースの使用を可能にできます。データベースの動作は、設定によって異なります。

- データベースをマウントする最初のインスタンスのCLUSTER_DATABASEパラメータがfalse(デフォルト)の場合は、最初のインスタンスのみがデータベースをマウントできます。
- このCLUSTER_DATABASEパラメータがtrueの場合は、他のインスタンスもCLUSTER_DATABASEパラメータの設定がtrueに設定されている状態でデータベースをマウントできます。データベースをマウントできるインスタンスの数は、データベースの作成時に指定された最大数によって決まります。

関連項目:

- データベースのマウント方法を学習するには、Oracle Database管理者ガイドを参照してください
- 1つのデータベースでの複数のインスタンスの使用方法の詳細は、Oracle Real Application Clusters管理およびデプロイメント・ガイドを参照してください

データベースのオープン方法

マウントされたデータベースをオープンすると、そのデータベースで通常のデータベース操作を実行できるようになります。

有効なユーザーは、オープン状態のデータベースに接続して、データベースの情報にアクセスできます。通常、データベース管理者は、データベースをオープンして一般に使用できる状態にします。

データベースをオープンするときに、Oracle Databaseでは次のアクションを自動的に実行します。

- UNDO表領域以外の表領域でオンライン・データファイルをオープンします。

  データベースを前回停止したときに表領域がオフラインであった場合は、データベースを再オープンしたとき、その表領域と対応するデータファイルはオフラインのままです。

- UNDO表領域を取得します。

  複数のUNDO表領域が存在する場合には、使用するUNDO表領域をUNDO_TABLESPACE初期化パラメータで指定できます。このパラメータが設定されていない場合、使用可能な最初のUNDO表領域が選択されます。

- オンラインREDOログ・ファイルをオープンします。

関連項目:

- 「オンライン表領域とオフライン表領域」
- 「データ修復」

読取り専用モード

デフォルトでは、データベースは**読取り/書込みモード**でオープンします。このモードでは、ユーザーがデータを変更し、オンラインREDOログにREDOを生成できます。また、**読取り専用モード**でオープンすると、ユーザー・トランザクションによってデータが変更されるのを防ぐことができます。

ノート:

デフォルトでは、フィジカル・スタンバイ・データベースは読取り専用モードでオープンします。

読取り専用モードでは、データベース・アクセスは読取り専用トランザクションに限定され、データファイルやオンラインREDOログ・ファイルに書き込むことはできません。ただし、データベースではリカバリや、REDOを生成しないでデータベースの状態を変更する操作を実行できます。たとえば、読取り専用モードでは、次の操作を実行できます。

- データファイルのオフラインとオンラインを切り替えることができます。ただし、永続表領域はオフライン化できません。
- オフラインのデータファイルと表領域をリカバリできます。
- 制御ファイルは、データベースの状態の更新に引き続き使用できます。
- CREATE TEMPORARY TABLESPACE文を使用して作成された一時表領域は、読取り/書込み可能です。
- オペレーティング・システムの監査証跡、トレース・ファイルおよびアラート・ログへの書込みは引き続き実行できます。

関連項目:

- データベースを読取り専用モードでオープンする方法を学習するには、『Oracle Database管理者ガイド』を参照してください。
- 『Oracle Data Guard概要および管理』

データベース・ファイルのチェック

インスタンスがデータベースをオープンするとき、データファイルまたはREDOログ・ファイルが存在しない場合、またはこれらのファイルは存在するが、一貫性テストに失敗した場合は、データベースからエラーが戻されます。メディア・リカバリが必要な場合があります。

関連項目:

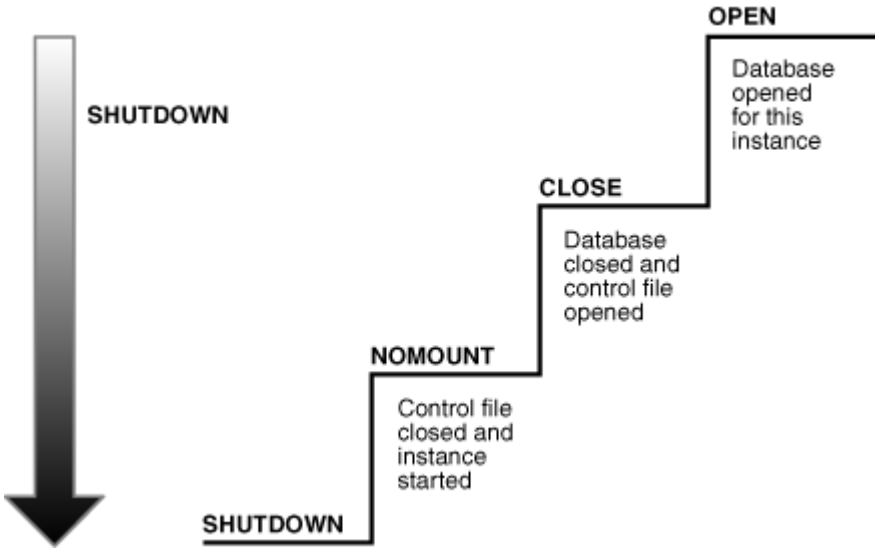
バックアップおよびリカバリ

データベースとインスタンスの停止の概要

通常のユースケースでは、手動でデータベースを停止し、メンテナンスや他の管理作業の実行中にユーザーがデータベースを使用できないようにします。SQL*Plus SHUTDOWNコマンドまたはEnterprise Managerを使用して、これらのステップを実行できます。

次の図に、オープン状態から一貫性のある停止状態までの経過を示します。

図13-4 インスタンスとデータベースの停止順序



オープン状態のデータベースが一貫性のある状態で停止されるときは、Oracle Databaseが次のステップを自動的に実行します。

表13-3 一貫性のある停止のステップ

フェーズ	マウント状態	説明	さらに学習するには
1	データベースのクローズ	データベースはマウントされていますが、オンライン・データファイルとREDO ログ・ファイルはクローズされます。	「データベースのクローズ方法」
2	データベースのアンマウント	インスタンスは起動状態にありますが、データベースの制御ファイルとの関連付けは解消されます。	「データベースのアンマウント方法」
3	データベース・インスタンスの停止	データベース・インスタンスは起動状態にありません。	「インスタンスの停止方法」

インスタンス障害やSHUTDOWN ABORTの場合、Oracle Databaseは前述のステップのすべてを実行せず、インスタンスを即時に終了します。

関連項目:

データベースの停止方法を学習するには、Oracle Database管理者ガイドを参照してください

停止モード

SYSDBAまたはSYSOPER権限を持つデータベース管理者は、SQL*PlusのSHUTDOWNコマンドまたはEnterprise Managerを使用してデータベースを停止できます。SHUTDOWNコマンドには、停止動作を指定するオプションがあります。

次の表に、様々な停止モードの動作の概要を示します。

表13-4 データベース停止モード

データベースの動作	ABORT	IMMEDIATE	TRANSACTIONAL	NORMAL
新しいユーザー接続を許可	いいえ	いいえ	いいえ	いいえ
現行セッションが終了するまで待機	いいえ	いいえ	いいえ	はい
現行トランザクションが終了するまで待機	いいえ	いいえ	はい	はい
チェックポイントを実行し、オープン状態のファイルをクローズ	いいえ	はい	はい	はい

使用可能なSHUTDOWN文は、次のとおりです。

- SHUTDOWN ABORT

  このモードは、他の停止方法が機能しない場合などの非常時のために用意されています。これは最速の停止モードです。ただし、停止後にこのデータベースをオープンする際に、データファイルの整合性を保つためにインスタンス・リカバリを実行する必要があるため、非常に長い時間がかかる場合があります。

  SHUTDOWN ABORTでは、オープン状態のデータファイルのチェックポイントが実行されないため、データベースを再オープンする前にインスタンス・リカバリが必要です。その他の停止モードでは、データベースを再オープンする前にインスタンス・リカバリを実行する必要はありません。

  ノート:

  CDB では、PDB に対して SHUTDOWN ABORT を発行することは、非 CDB に対して SHUTDOWN IMMEDIATE を発行することと同じです。

- SHUTDOWN IMMEDIATE

  通常は、SHUTDOWN ABORTの次に速い停止モードです。Oracle Databaseは、実行中のSQL文をすべて終了し、ユーザーとの接続を切断します。アクティブなトランザクションは終了され、コミットされていない変更はロールバックされます。

- SHUTDOWN TRANSACTIONAL

  このモードでは、ユーザーが新しいトランザクションを開始することはできませんが、すべての現行トランザクションが完了するまで待機してから停止します。現行トランザクションの性質によっては、停止までにかなりの時間がかかる場合があります。

- SHUTDOWN NORMAL

  これはデフォルトの停止モードです。データベースは、接続されたすべてのユーザーの接続が解除されるまで待機してから停止します。

関連項目:

- 様々な停止モードについて学習するには、『Oracle Database管理者ガイド』を参照してください。
- SHUTDOWNコマンドについて学習するには、『SQL*Plusユーザーズ・ガイドおよびリファレンス』を参照してください。

データベースのクローズ方法

データベースのクローズ操作は、データベース停止時に暗黙的に実行されます。この操作の性質は、データベースが正常停止されたか、異常停止されたかによって異なります。

正常停止時のデータベースのクローズ方法

ABORT以外のオプションが指定されたSHUTDOWNの一環としてデータベースがクローズされると、Oracle DatabaseはSGA内のデータをデータファイルとオンラインREDOログ・ファイルに書き込みます。

次に、データベースはオンライン・データファイルとオンラインREDOログ・ファイルをクローズします。オフライン表領域のオフライン・データファイルは、すでにクローズされています。データベースを再オープンしたとき、オフラインになっていた表領域はオフラインのままです。

この時点でデータベースはクローズされているため、通常の操作は実行できません。データベースがクローズされても、制御ファイルはオープンされたままになります。

異常停止時のデータベースのクローズ方法

SHUTDOWN ABORTまたは異常終了が発生すると、オープン状態のデータベースのインスタンスはクローズされ、データベースはただちに停止されます。

異常終了時に、Oracle DatabaseはSGAのバッファ内のデータをデータファイルとREDOログ・ファイルに書き込みません。次にデータベースを再オープンするときに必要なインスタンス・リカバリは、Oracle Databaseで自動的に実行されます。

データベースのアンマウント方法

データベースがクローズされると、Oracle Databaseはデータベースをアンマウントしてインスタンスから切り離します。

データベースをアンマウントすると、Oracle Databaseでは、データベースの制御ファイルがクローズされます。この時点ではデータベース・インスタンスはメモリーに残っています。

インスタンスの停止方法

データベース停止の最後の操作は、インスタンスの停止です。データベース・インスタンスが停止すると、SGAではメモリーの占有を中止し、バックグラウンド・プロセスが終了します。

異常な状況では、データベース・インスタンスが正常に停止しないことがあります。たとえば、メモリー構造がメモリーから削除されない場合や、バックグラウンド・プロセスの1つが終了されない場合があります。前のインスタンスの一部が残っていると、後続のインスタンスの起動が失敗します。このような状況では、残っている前のインスタンスを削除してから新しいインスタンスを起動するか、SHUTDOWN ABORT文を発行することで、新しいインスタンスの起動を強制実行できます。

場合によっては、プロセスのクリーンアップ自体でエラーが発生し、その結果としてプロセス・モニター(PMON)またはインスタンスが終了することもあります。動的な初期化パラメータINSTANCE_ABORT_DELAY_TIMEは、内部で発生したインスタンス障害を遅延させる秒数を指定します。この遅延により、対応する機会が得られます。終了の遅延が始動したとき、データベースによってアラート・ログにメッセージが書き込まれます。状況によっては、特定のデータベース・リソースを隔離できるようにすることで、インスタンスの終了を回避できます。

関連項目:

- データベースの停止の詳細は、『Oracle Database管理者ガイド』を参照
- INSTANCE_ABORT_DELAY_TIME初期化パラメータについてさらに学習するには、『Oracle Databaseリファレンス』を参照

チェックポイントの概要

一般的に、**チェックポイント**は、一貫性のあるデータベースの停止、インスタンス・リカバリおよびOracle Databaseの操作に不可欠なメカニズムです。

この用語の意味は次のとおりです。

- **チェックポイント位置**を示すデータ構造。これは、インスタンス・リカバリを開始する必要があるREDOストリーム内のSCNです。

  チェックポイント位置は、データベース・バッファ・キャッシュ内の最も古い使用済バッファによって決まります。チェックポイント位置は、REDOストリームへのポインタとして機能し、制御ファイル内と各データファイル・ヘッダー内に格納されます。
- データベース・バッファ・キャッシュ内の変更済データベース・バッファのディスクへの書込み

関連項目:

「システム変更番号(SCN)」

チェックポイントの目的

Oracle Databaseでは、複数の目的を達成するためにチェックポイントを使用します。

目的には次のものがあります。

- インスタンス障害またはメディア障害時のリカバリに要する時間を短縮します。
- データベースでは、ディスクへのバッファ・キャッシュで、使用済バッファを定期的に書き込みます。
- 一貫性のある停止が実行されるとき、すべてのコミット済データがデータベースによりディスクに書き込まれることを保証します。

Oracle Databaseでチェックポイントが開始される時期

チェックポイント・プロセス(CKPT)は、データファイル・ヘッダーと制御ファイルにチェックポイントを書き込みます。

チェックポイントは、様々な状況で発生します。たとえば、Oracle Databaseでは、次のタイプのチェックポイントを使用します。

- スレッドのチェックポイント

  データベースは、特定のターゲットより前の特定のスレッド内でREDOによって変更されたすべてのバッファをディスクに書き込みます。データベース内の全インスタンスに対するスレッドのチェックポイントのセットが、**データベースのチェックポイント**です。スレッドのチェックポイントは、次の状況で発生します。

  - 一貫性のあるデータベースの停止
  - ALTER SYSTEM CHECKPOINT文

- オンラインREDOログの切替え
- ALTER DATABASE BEGIN BACKUP文

- 表領域とデータファイルのチェックポイント

  データベースは、特定のターゲットより前のREDOによって変更されたすべてのバッファをディスクに書き込みます。表領域のチェックポイントは、表領域のデータファイルごとに1つあるデータファイルのチェックポイントのセットです。これらのチェックポイントは、表領域の読取り専用化やNORMALモードでのオフライン化、データファイルの縮小、またはALTER TABLESPACE BEGIN BACKUPの実行時など、様々な状況で発生します。

- 増分チェックポイント

  増分チェックポイントはスレッドのチェックポイントの一種であり、オンラインREDOログを切り替えるとき、大量のブロックの書込みを防止することが目的の1つとなっています。DBWは、実行する処理の有無を少なくとも3秒ごとにチェックします。DBWが使用済バッファを書き込むと、チェックポイント位置が前進し、これにより、CKPTでは、このチェックポイント位置をデータファイル・ヘッダーではなく制御ファイルに書き込みます。

その他のタイプには、インスタンスとメディア・リカバリのチェックポイント、およびスキーマ・オブジェクトが削除または切り捨てられたときのチェックポイントがあります。

関連項目:

- 「チェックポイント・プロセス(CKPT)」
- Oracle RACでのグローバル・チェックポイントの詳細は、『Oracle Real Application Clusters管理およびデプロイメント・ガイド』を参照してください

インスタンス・リカバリの概要

インスタンス・リカバリは、オンラインREDOログ内のレコードをデータファイルに適用し、最新のチェックポイントの後に行われた変更を再構築するプロセスです。

管理者が前回一貫性のない状態で停止されたデータベースの再オープンを試行すると、インスタンス・リカバリが自動的に実行されます。

インスタンス・リカバリの目的

インスタンス・リカバリは、インスタンス障害後にデータベースが一貫性のある状態になることを保証します。Oracle Databaseでのデータベースの変更管理方法により、データベースのファイルが一貫性のない状態のままになることがあります。

REDOスレッドは、インスタンスによって生成された変更のすべてのレコードです。単一インスタンス・データベースには1つのREDOスレッドがあり、Oracle RACデータベースには各データベース・インスタンスに対応する複数のREDOスレッドがあります。

トランザクションがコミットされると、ログ・ライター・プロセス(LGWR)は、メモリー内の残りのREDOエントリとトランザクションのSCNの両方をオンラインREDOログに書き込みます。ただし、データベース・ライター(DBW)プロセスでは、変更されたデータ・ブロックを最も効率的な時点でデータファイルに書き込みます。このため、コミットされた変更がまだデータファイルに存在しないとき、コミットされていない変更が一時的にデータファイルに存在することがあります。

SHUTDOWN ABORT文または異常終了のいずれかにより、オープン状態のデータベースのインスタンスに障害が発生すると、次の状態になることがあります。

- トランザクションによってコミットされたデータ・ブロックはデータファイルに書き込まれず、オンラインREDOログにのみ表示されます。これらの変更は、データ・ファイルに再適用される必要があります。
- インスタンスに障害が発生した時点でコミットされていない変更がデータファイルに含まれています。これらの変更は、トランザクションの一貫性を保証するためにロールバックする必要があります。

インスタンス・リカバリは、オンラインREDOログ・ファイルと現行のオンライン・データファイルのみを使用してデータファイルを同期し、これらが一貫性のある状態になることを保証します。

関連項目:

- 「データベース・ライター・プロセス(DBW)」および「データベース・バッファ・キャッシュ」
- 「データの同時実行性と整合性の概要」

Oracle Databaseでインスタンス・リカバリが実行される時期

インスタンス・リカバリが必要かどうかは、REDOスレッドの状態によります。

制御ファイル内では、データベース・インスタンスが読取り/書込みモードでオープンすると、REDOスレッドにオープンのマークが付けられ、インスタンスが一貫性のある状態で停止されると、クローズ済のマークが付けられます。REDOスレッドに制御ファイルでオープンのマークが付けられると、稼働中のインスタンスはこれらのスレッドに対応するスレッドのエンキューを保持しないため、データベースでインスタンス・リカバリが必要となります。

Oracle Databaseでは、次の場合にインスタンス・リカバリが自動的に実行されます。

- 単一インスタンス・データベースまたはOracle RACデータベースの全インスタンスの障害後に、最初にデータベースをオープンするとき。このタイプのインスタンス・リカバリは**クラッシュ・リカバリ**とも呼ばれます。Oracle Databaseでは、終了したインスタンスのオンラインREDOスレッドも同時にリカバリされます。
- Oracle RACデータベースに含まれる全インスタンスではなく一部のインスタンスに障害が発生したとき。インスタンス・リカバリは、構成内で正常に稼働しているインスタンスにより自動的に実行されます。

SMONバックグラウンド・プロセスではインスタンス・リカバリが実行され、オンラインREDOが自動的に適用されます。ユーザーが介入する必要はありません。

関連項目:

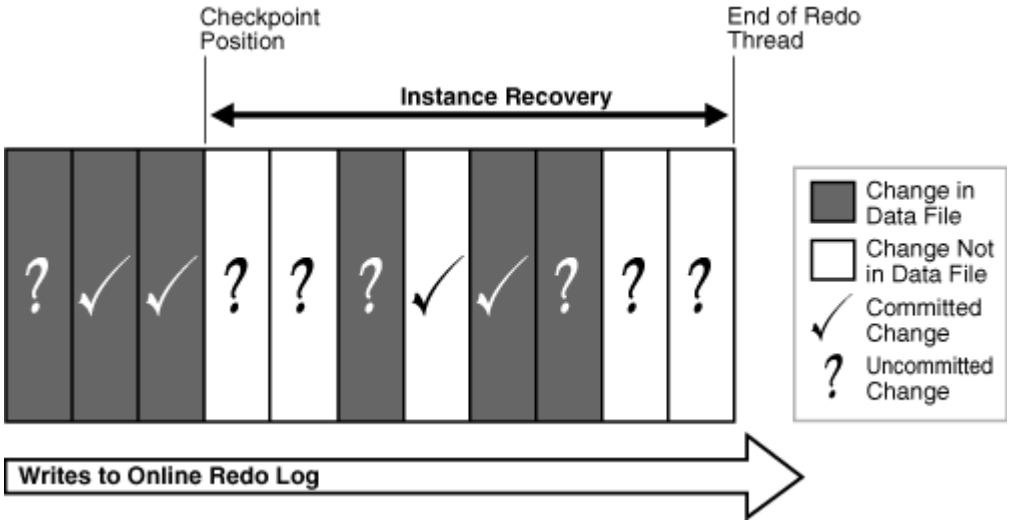
- 「システム・モニター・プロセス(SMON)」
- Oracle RACデータベースでのインスタンス・リカバリについて学習するには、『Oracle Real Application Clusters管理およびデプロイメント・ガイド』を参照してください

インスタンス・リカバリでのチェックポイントの重要性

インスタンス・リカバリでは、データファイルに適用する必要がある変更を判断するためにチェックポイントが使用されます。チェックポイント位置は、チェックポイントSCNより小さいSCNを持つすべてのコミット済の変更がデータファイルに保存されることを保証するものです。

次の図に、オンラインREDOログ内のREDOスレッドを示します。

図13-5 オンラインREDOログ内のチェックポイント位置



インスタンス・リカバリ時には、データベースによってチェックポイント位置からREDOスレッドの最後までに発生した変更が適用される必要があります。図13-5に示されているように、変更の一部はすでにデータファイルに書き込まれています。ただし、ディスクに書き込まれるのは、チェックポイント位置より小さいSCNを持つ変更のみであることが保証されます。

関連項目:

リカバリ時間の制限方法を学習するには、Oracle Databaseパフォーマンス・チューニング・ガイドを参照してください

インスタンス・リカバリのフェーズ

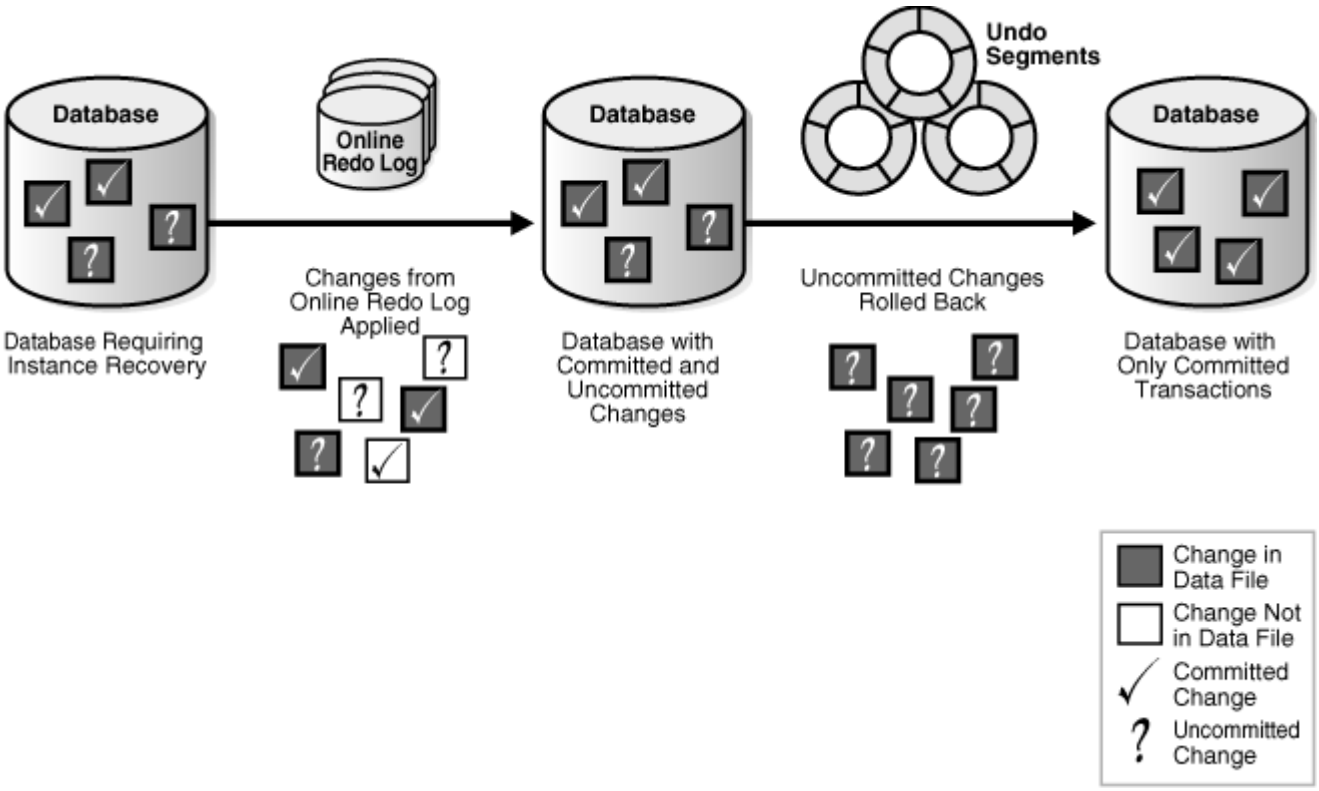
インスタンス・リカバリの最初のフェーズは、**キャッシュ・リカバリ**または**ロールフォワード**と呼ばれ、オンラインREDOログに記録されているすべての変更をデータファイルに再適用します。

オンラインREDOログにはUNDOデータが含まれているため、ロールフォワードでは対応するUNDOセグメントも再生成されます。ロールフォワードでは、必要に応じて多数のオンラインREDOログ・ファイルをたどり、データベースを後の時点まで進行させます。ロールフォワード後のデータ・ブロックには、オンラインREDOログ・ファイルに記録されたコミット済の変更がすべて含まれています。これらのファイルには、障害発生前にデータファイルに保存されていたか、オンラインREDOログに記録されてキャッシュ・リカバリ時に取り込まれた、コミットされていない変更も含まれることがあります。

ロールフォワード後は、コミットされなかった変更を取り消す必要があります。Oracle Databaseでは、チェックポイント位置が使用され、これにより、チェックポイントSCNより小さいSCNを持つすべてのコミット済の変更がディスクに保存されることが保証されます。Oracle Databaseでは、UNDOブロックの適用により、障害発生前に書き込まれたか、キャッシュ・リカバリ中に取り込まれたデータ・ブロック内の、コミットされていない変更がロールバックされます。このフェーズは、**ロールバック**またはトランザクション・リカバリと呼ばれます。

次の図に、ロールフォワードとロールバックを示します。この2つは、データベース・インスタンスの障害からのリカバリに必要なステップです。

図13-6 インスタンス・リカバリの基本ステップ: ロールフォワードとロールバック



Oracle Databaseは、必要に応じて同時に複数のトランザクションをロールバックできます。障害時点でアクティブになっていたトランザクションすべてに、終了マークが付けられます。終了したトランザクションがSMONプロセスによりロールバックされるのを待たずに、新しいトランザクションによりトランザクション自体で個別のブロックをロールバックし、必要なデータを取得できます。

関連項目:

- UNDOデータについてさらに学習するには、「UNDOセグメント」を参照してください
- インスタンス・リカバリのメカニズムとチューニングの詳細は、Oracle Databaseパフォーマンス・チューニング・ガイドを参照してください

パラメータ・ファイルの概要

データベース・インスタンスを起動するには、Oracle Databaseは**サーバー・パラメータ・ファイル**(推奨)または**テキスト形式の初期化パラメータ・ファイル**(従来型の実装)を読み取る必要があります。これらのファイルには、構成パラメータのリストが格納されます。

データベースを手動で作成するには、パラメータ・ファイルを使用してインスタンスを起動してから、CREATE DATABASE文を発行する必要があります。このため、データベース自体が存在していない場合でも、インスタンスとパラメータ・ファイルが存在することがあります。

初期化パラメータ

初期化パラメータは、インスタンスの基本操作に影響する構成パラメータです。インスタンスは、起動時にファイルから初期化パラメータを読み取ります。

Oracle Databaseには、様々な環境に応じて操作を最適化する多数の初期化パラメータが用意されています。これらのパラメータのごく一部は、通常はデフォルト値のままで十分であり、明示的な設定が必要です。

初期化パラメータの機能グループ

初期化パラメータは様々な機能グループに分類されます。

ほとんどの初期化パラメータは、次のグループのいずれかに属しています。

- ファイルやディレクトリなどのエンティティに名前を指定するパラメータ
- プロセス、データベース・リソースまたはデータベース自体の制限を設定するパラメータ
- SGAのサイズなどの容量に影響するパラメータ(**可変パラメータ**と呼ばれる)

可変パラメータを使用するとデータベースのパフォーマンスを改善できるため、これらのパラメータは、特にデータベース管理者によって頻繁に使用されます。

基本的な初期化パラメータと拡張初期化パラメータ

初期化パラメータは、基本と拡張という2つのグループに分かれています。

通常、妥当なパフォーマンスを得るには、約30種類の基本パラメータのみを設定およびチューニングする必要があります。基本パラメータでは、データベース名、制御ファイルの場所、データベース・ブロック・サイズ、UNDO表領域などの特性を設定します。

ただし最適なパフォーマンスを得るために、拡張パラメータの変更が必要になる場合があります。拡張パラメータを使用することによって、熟練したDBAは固有の要件にあわせてOracle Databaseの動作を調整できます。

Oracle Databaseは、データベース・ソフトウェアで提供される最初の初期化パラメータ・ファイル、またはDatabase Configuration Assistantにより作成される最初の初期化パラメータ・ファイルに値を提供します。構成およびデータベースのチューニング計画に応じて、これらのOracle提供の初期化パラメータを編集し、他のパラメータを追加できます。該当する初期化パラメータがパラメータ・ファイルに組み込まれていない場合には、Oracle Databaseによりデフォルトが指定されます。

関連項目:

- 「データベースのインストールおよび構成ツール」
- 初期化パラメータの指定方法を学習するには、『Oracle Database管理者ガイド』を参照してください
- 初期化パラメータのタイプの詳細は、『Oracle Databaseリファレンス』を参照してください
- V$PARAMETERの詳細は『Oracle Databaseリファレンス』を、SHOW PARAMETER構文については『SQL*Plusユーザーズ・ガイドおよびリファレンス』を参照してください

サーバー・パラメータ・ファイル

サーバー・パラメータ・ファイルは、初期化パラメータのリポジトリです。

サーバー・パラメータ・ファイルには、主に次の特徴があります。

- Oracle Databaseのみが、サーバー・パラメータ・ファイルに対する読取りおよび書き込みを行います。
- サーバー・パラメータ・ファイルは、データベースに対して1つのみ存在します。このファイルは、データベース・ホストに常駐する必要があります。
- サーバー・パラメータ・ファイルはバイナリであり、テキスト・エディタでは編集できません。
- サーバー・パラメータ・ファイルに格納されている初期化パラメータは永続的です。データベース・インスタンスの実行中に行われたパラメータの変更は、インスタンスを停止して起動しても残ります。

サーバー・パラメータ・ファイルを使用すると、クライアント・アプリケーション用に複数のテキスト形式の初期化パラメータ・ファイルを保持する必要がなくなります。最初のサーバー・パラメータ・ファイルは、CREATE SPFILE文を使用して、テキスト形式の初期化パラメータ・ファイルから作成します。また、Database Configuration Assistantにより直接作成することもできます。

関連項目:

- サーバー・パラメータ・ファイルについてさらに学習するには、Oracle Database管理者ガイドを参照してください。
- CREATE SPFILEについて学習するには、Oracle Database SQL言語リファレンスを参照してください。

テキスト形式の初期化パラメータ・ファイル

テキスト形式の初期化パラメータ・ファイルは、初期化パラメータのリストが格納されるテキスト・ファイルです。

このタイプのパラメータ・ファイルは、従来型の実装のパラメータ・ファイルであり、主に次の特徴があります。

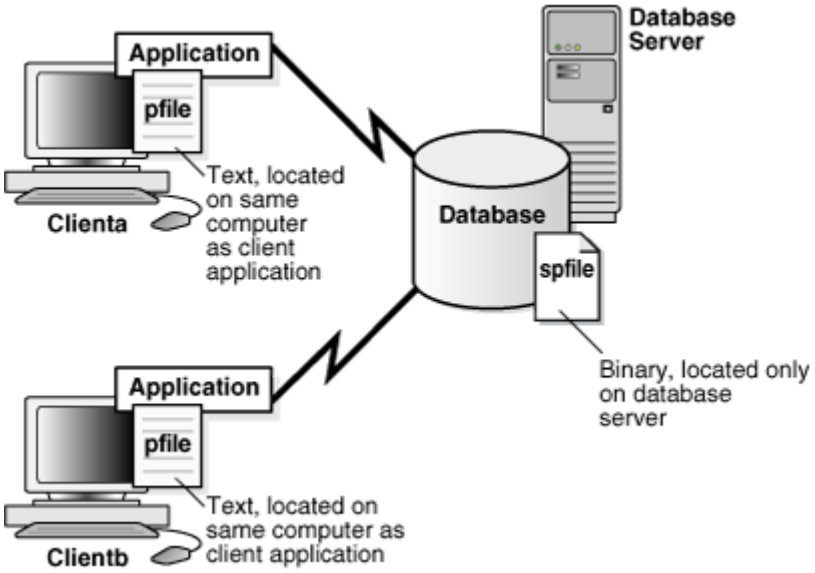
- データベースを起動または停止するときに、テキスト形式の初期化パラメータ・ファイルは、データベースに接続するクライアント・アプリケーションと同じホストに常駐する必要があります。
- テキスト形式の初期化パラメータ・ファイルはテキストベースであり、バイナリではありません。
- Oracle Databaseは、テキスト形式の初期化パラメータ・ファイルの読取りはできますが、書込みはできません。パラメータ値を変更するには、テキスト・エディタを使用してファイルを手動で編集する必要があります。
- ALTER SYSTEMを使用した初期化パラメータ値の変更は、現行インスタンスにのみ有効になります。変更内容を確認するには、テキスト形式の初期化パラメータ・ファイルを手動で更新し、インスタンスを再起動する必要があります。

テキスト形式の初期化パラメータ・ファイルには、一連のキー=値のペア(行ごとに1ペア)が格納されています。たとえば、初期化パラメータ・ファイルの一部は、次のように表示されます。

```
db_name=sample
control_files=/disk1/oradata/sample_cf.dbf
db_block_size=8192
open_cursors=52
undo_management=auto
shared_pool_size=280M
pga_aggregate_target=29M
.
.
.
```

テキスト形式のパラメータ・ファイルの管理性に関する問題を具体的に説明するため、clientaとclientbというコンピュータを使用し、いずれのコンピュータ上でもSQL*Plusでデータベースを起動できるようにする必要があるとします。この場合、図13-7に示すように、個別のテキスト形式の初期化パラメータ・ファイルが2つ(コンピュータごとに1つ)存在する必要があります。サーバー・パラメータ・ファイルは、パラメータ・ファイルが増加する問題を解決します。

図13-7 複数の初期化パラメータ・ファイル



関連項目:

- テキスト形式の初期化パラメータ・ファイルについてさらに学習するには、Oracle Database管理者ガイドを参照してください
- CREATE PFILEについて学習するには、Oracle Database SQL言語リファレンスを参照してください

初期化パラメータ値の変更

初期化パラメータを調整してデータベースの動作を変更できます。パラメータの分類が**静的**であるか、**動的**であるかによって、パラメータの変更方法が決まります。

以下の表にその違いをまとめています。

表13-5 静的な初期化パラメータと動的な初期化パラメータ

特徴	静的	動的
パラメータ・ファイル(テキストまたはサーバー)を変更する必要性	はい	いいえ
設定を有効にする前にデータベース・インスタンスを再起動する必要性	はい	いいえ
『Oracle Database リファレンス』の初期化パラメータのエントリでの変更可能という説明	いいえ	はい
データベースまたはインスタンスによってのみ変更可能かどうか	はい	いいえ

静的なパラメータには、DB_BLOCK_SIZE、DB_NAMEおよびCOMPATIBLEがあります。動的なパラメータは、現行のユーザー・セッションにのみ影響する**セッション・レベルのパラメータ**と、データベースおよびすべてのセッションに影響する**システム・レベルのパラ**

メータに分類されます。たとえば、MEMORY_TARGETはシステム・レベルのパラメータであり、NLS_DATE_FORMATはセッション・レベルのパラメータです。

パラメータ変更の**有効範囲**は、変更が有効になる時期によって異なります。インスタンスがサーバー・パラメータ・ファイルで起動されている場合は、ALTER SYSTEM SET文を使用して、次のようにシステム・レベルのパラメータの値を変更できます。

- SCOPE=MEMORY

  変更はデータベース・インスタンスにのみ適用されます。データベースが停止して再起動された場合、変更は存続しません。

- SCOPE=SPFILE

  変更はサーバー・パラメータ・ファイルに適用されますが、現行インスタンスには影響しません。このため、インスタンスが再起動されるまで変更は有効になりません。

ノート:

『Oracle Database リファレンス』で変更不可と説明されているパラメータの値を変更する場合は、SPFILE を指定する必要があります。

- SCOPE=BOTH

  Oracle Databaseでは、変更はメモリーとサーバー・パラメータ・ファイルの両方に書き込まれます。これは、データベースがサーバー・パラメータ・ファイルを使用している場合のデフォルトの有効範囲です。

データベースにより、初期化パラメータの新しい値と古い値がアラート・ログに出力されます。無効な値がサーバー・パラメータ・ファイルに書き込まれるのを防ぐため、予防策として、データベースにより基本的なパラメータの変更が検証されます。

関連項目:

- 「ロケール固有の設定」
- 初期化パラメータ設定の変更方法を学習するには、『Oracle Database管理者ガイド』を参照してください。
- すべての初期化パラメータの詳細は、『Oracle Databaseリファレンス』を参照してください。
- ALTER SYSTEMの構文およびセマンティクスは、『Oracle Database SQL言語リファレンス』を参照してください。

診断ファイルの概要

Oracle Databaseには、データベースの問題を防止、検出、診断および解決するための**障害診断性インフラストラクチャ**が含まれています。問題には、コードの不具合、メタデータの破損、顧客データの破損などの重大なエラーがあります。

この高度な障害診断性インフラストラクチャの目的は、次のとおりです。

- 問題の予防的な検出
- 問題が検出された後の損害および中断の抑制
- 問題の診断時間と解決時間の短縮
- トレース・ファイルのパーティション化を有効にすること、ユーザーがピース当たりのサイズおよび保持するピースの最大数

を定義できるようにすること、ユーザー指定のディスク領域制限に到達後にトレースを無効にすることによる管理容易性の向上

- 顧客とOracleサポートのやり取りの簡略化

マルチテナント・コンテナ・データベース(CDB)と非CDBには、アーキテクチャに違いがあります。この項では、特に他に指示がないかぎり、非CDBのアーキテクチャを前提としています。

関連項目:

CDBでの診断ファイルの管理方法を学習するには、『Oracle Database管理者ガイド』を参照してください

自動診断リポジトリ

自動診断リポジトリ(ADR)は、トレース・ファイル、アラート・ログ、DDLログ、状態モニター・レポートなどのデータベース診断データを格納するファイルベース・リポジトリです。

ADRの主な特徴は、次のとおりです。

- 統合ディレクトリ構造
- 一貫性のある診断データ形式
- 統合されたツール・セット

これらの特性により、ユーザーとOracleサポートは、複数のOracleインスタンス、コンポーネントおよび製品間で診断データを相関付けて、分析できます。

ADRはデータベースの外側に位置するため、物理データベースが使用不可の場合でも、Oracle DatabaseはADRにアクセスして管理できます。データベース・インスタンスにより、データベースの作成が完了する前にADRが作成されることがあります。

問題とインシデント

ADRは、データベース内で重大なエラーとなる**問題**を予防的に追跡します。

重大なエラーとは、ORA-600などの内部エラーや他のサーバー・エラーのことです。各問題には、問題を説明する文字列である**問題キー**があります。

1つの問題が複数回発生した場合は、発生するたびに、タイムスタンプの付いた**インシデント**がADRによって作成されます。インシデントは、数字の**インシデントID**によって一意に識別されます。インシデントが発生すると、ADRでは**インシデント・アラート**をEnterprise Managerに送信します。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。

1つの問題で短時間に多数のインシデントが生成される場合があるため、ADRでは特定のしきい値に到達した後、インシデントの生成にフラッド制御を適用します。**大量データを制御されたインシデント**では、アラート・ログ・エントリは生成されますが、インシデントのダンプは生成されません。この方法では、ADRから重大なエラーが継続中であることが通知され、診断データによるシステムのオーバーロードは回避されます。

関連項目:

障害診断性インフラストラクチャの詳細は、『Oracle Database管理者ガイド』を参照してください

ADRの構造

ADRベースは、ADRのルート・ディレクトリです。

ADRベースには、複数のADRホームを格納でき、それぞれのADRホームは、Oracle製品やコンポーネントのインスタンスごとにすべての診断データ(トレース、ダンプ、アラート・ログなど)を格納するルート・ディレクトリです。たとえば、共有記憶域とOracle ASMがあるOracle RAC環境では、各データベース・インスタンスと各Oracle ASMインスタンスにはそれぞれ独自のADRホームがあります。

図13-8に、データベース・インスタンスのADRディレクトリ階層を示します。この階層内の同じADRベースの下に、Oracle ASMまたはOracle Net Servicesなどの他のOracle製品やコンポーネントに対する別のADRホームを格納できます。

図13-8 Oracleデータベース・インスタンスのADRディレクトリ構造

次のLinuxの例に示すように、データベースを作成する前に一意のSIDとデータベース名でインスタンスを起動すると、Oracle Databaseは、ホスト・ファイルシステムのディレクトリ構造としてデフォルトでADRを作成します。SIDとデータベース名は、ADRホーム内のファイルのパス名の一部になります。

例13-1 ADRの作成

```
% setenv ORACLE_SID osi
% echo "DB_NAME=dbn" > init.ora
% sqlplus / as sysdba
.
.
.
Connected to an idle instance.

SQL> STARTUP NOMOUNT PFILE="./init.ora"
ORACLE instance started.

Total System Global Area  146472960 bytes
Fixed Size                  1317424 bytes
Variable Size              92276176 bytes
Database Buffers           50331648 bytes
Redo Buffers                2547712 bytes

SQL> COL NAME FORMAT a21
SQL> COL VALUE FORMAT a60
SQL> SELECT NAME, VALUE FROM V$DIAG_INFO;
```

```
NAME                 VALUE
-------------------- -----------------------------------------------------
Diag Enabled         TRUE
ADR Base             /d1/3910926111/oracle/log
ADR Home             /d1/3910926111/oracle/log/diag/rdbms/dbn/osi
Diag Trace           /d1/3910926111/oracle/log/diag/rdbms/dbn/osi/trace
Diag Alert           /d1/3910926111/oracle/log/diag/rdbms/dbn/osi/alert
Diag Incident        /d1/3910926111/oracle/log/diag/rdbms/dbn/osi/incident
Diag Cdump           /d1/3910926111/oracle/log/diag/rdbms/dbn/osi/cdump
Health Monitor       /d1/3910926111/oracle/log/diag/rdbms/dbn/osi/hm
Default Trace File   /d1/3910926111/oracle/log ... osi/trace/osi_ora_6825.trc
Active Problem Count 0
Active Incident Count 0
```

アラート・ログ

各データベースには、**アラート・ログ**があり、これは、データベースのメッセージとエラーの履歴ログが格納されたXMLファイルです。

アラート・ログの内容は次のとおりです。

- すべての内部エラー(ORA-600)、ブロック破損エラー(ORA-1578)およびデッドロック・エラー(ORA-60)
- 管理操作(たとえば、SQL*PlusコマンドのSTARTUP、SHUTDOWN、ARCHIVE LOG、およびRECOVERなど)
- 共有サーバーとディスパッチャ・プロセスの機能に関係するいくつかのメッセージとエラー。
- マテリアライズド・ビューの自動リフレッシュにおけるエラー。

Oracle Databaseでは、Enterprise ManagerのGUIに情報を表示するかわりに、アラート・ログを使用します。管理操作が成功すると、Oracle Databaseはタイムスタンプとともに「completed」というメッセージをアラート・ログに書き込みます。

最初にデータベース・インスタンスを起動すると、データベースがまだ作成されていない場合でも、Oracle Databaseは図13-8に示すalertサブディレクトリにアラート・ログを作成します。このファイルはXML形式です。トレース・サブディレクトリにはテキストのみのアラート・ログが含まれ、次の例にその一部を示しています。

```
Fri Nov 02 12:41:58 2014
SMP system found. enable_NUMA_support disabled (FALSE)
Starting ORACLE instance (normal)
CLI notifier numLatches:3 maxDescs:189
LICENSE_MAX_SESSION = 0
LICENSE_SESSIONS_WARNING = 0
Initial number of CPU is 2
Number of processor cores in the system is 2
Number of processor sockets in the system is 2
Shared memory segment for instance monitoring created
Picked latch-free SCN scheme 3
Using LOG_ARCHIVE_DEST_1 parameter default value as /disk1/oracle/dbs/arch
Autotune of undo retention is turned on.
IMODE=BR
ILAT =10
LICENSE_MAX_USERS = 0
SYS auditing is disabled
NOTE: remote asm mode is local (mode 0x1; from cluster type)
Starting up:
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Production
With the Partitioning, Advanced Analytics and Real Application Testing options.
.
.
.
```

```
Using parameter settings in client-side pfile
System parameters with nondefault values:
  processes                = 100
  sessions                 = 172
```

例13-1に示すように、アラート・ログを検索するにはV$DIAG_INFOを問い合せます。

DDLログ

DDLログは、形式および基本的な動作がアラート・ログと同じですが、DDL文および詳細のみが含まれる点が異なります。データベースでは、DDL情報を自身のファイルに書き込み、アラート・ログでの乱雑さを減らします。

DDLログ・レコードはDDLテキストであり、オプションで補足情報より拡張できます。DDL文ごとに1つのログ・レコードが存在します。DDLログは、ADRホームのlog/ddlサブディレクトリに格納されています。

トレース・ファイル

トレース・ファイルは、問題の調査に使用される診断データが含まれるファイルです。また、トレース・ファイルはアプリケーションまたはインスタンスのチューニングにガイダンスを提供することもできます。

関連項目:

「パフォーマンスおよびチューニング」

トレース・ファイルのタイプ

それぞれのサーバーとバックグラウンド・プロセスは、関連付けられたトレース・ファイルに定期的に書き込むことができます。トレース・ファイルには、プロセスの環境、ステータス、アクティビティおよびエラーに関する情報が含まれています。

SQLトレース機能でも、個別のSQL文に関するパフォーマンス情報を提供するトレース・ファイルが作成されます。クライアント識別子、サービス、モジュール、アクション、セッション、インスタンスまたはデータベースのトレースを様々な方法で有効化できます。たとえば、DBMS_MONITORパッケージの適切なプロシージャを実行することも、イベントを設定することもできます。

関連項目:

- 「セッション制御文」
- トレース・ファイル、ダンプおよびコア・ファイルについて学習するには、『Oracle Database管理者ガイド』を参照してください。
- アプリケーションのトレースについて学習するには、Oracle Database SQLチューニング・ガイドを参照してください。

トレース・ファイルの場所

ADRでは、traceサブディレクトリにトレース・ファイルが格納されます。トレース・ファイル名はプラットフォーム依存であり、拡張子.trcが使用されます。

通常、データベース・バックグラウンド・プロセスのトレース・ファイル名には、Oracle SID、バックグラウンド・プロセス名およびオペレーティング・システムのプロセス番号が含まれています。たとえば、RECOプロセスのトレース・ファイル名は、mytest_reco_10355.trcのようになります。

サーバー・プロセスのトレース・ファイル名には、Oracle SID、文字列oraおよびオペレーティング・システムのプロセス番号が含まれています。たとえば、サーバー・プロセスのトレース・ファイル名は、mytest_ora_10304.trcのようになります。

トレース・ファイルには、対応するトレース・メタデータのファイル(拡張子.trmで終わる)が付くことがあります。これらのファイルには**トレース・マップ**と呼ばれる構造に関する情報が含まれており、データベースでは検索やナビゲーションに使用されます。

関連項目:

- 「図13-8」
- トレース・ファイルの検索方法を学習するには、『Oracle Database管理者ガイド』を参照してください

トレース・ファイルのセグメント化

トレース・ファイルのサイズが制限されていると、データベースがトレース・ファイルを最大5つのセグメントに自動的に分割する場合があります。セグメントとは、アクティブなトレース・ファイルと同じ名前の個別のファイルですが、ora_1234_2.trcのように、セグメント番号が追加されます。

通常、各セグメントは、MAX_DUMP_FILE_SIZEによって設定される制限の20%になります。すべてのセグメントの結合サイズが制限を超えると、データベースは古いセグメントから削除し(ただし、プロセスの初期状態に関する情報を含む可能性のある最初のセグメントは削除しません)、新しい空のセグメントを作成します。

関連項目:

トレース・ファイルのサイズを制御する方法を学習するには、『Oracle Database管理者ガイド』を参照してください

診断ダンプ

診断ダンプ・ファイルは、特殊なタイプのトレース・ファイルで、状態または構造に関する特定の時点の詳細な情報が含まれています。

多くの場合、トレースでは診断データが連続して出力されます。一方、通常、ダンプではイベントに対して診断データが1回出力されます。

トレース・ダンプおよびインシデント

ほとんどのダンプは、インシデントが原因で発生します。

インシデントが発生すると、データベースは、そのインシデント用に作成されたインシデント・ディレクトリに1つ以上のダンプを書き込みます。インシデントのダンプには、ファイル名にインシデント番号も付きます。

インシデントの作成中、アプリケーションではアクションの一部として、ヒープまたはシステム状態のダンプを取得する場合があります。そのような場合、データベースでは、デフォルトのトレース・ファイル名のかわりに、ダンプ名をインシデント・ファイル名に追加します。たとえば、プロセス内の1つのインシデントのために、データベースではprod_ora_90348.trcファイルが作成されます。そのインシデントでのダンプにより、prod_ora_90348_incident_id.trcファイルが生成されます(incident_idはインシデントの数値ID)。インシデントの一部として作成されたヒープ・ダンプ・アクションにより、ヒープ・ダンプ・ファイルprod_ora_90348_incident_id_dump_id.trcが生成されます(dump_idは、トレース・ダンプの数値ID)。

14 メモリー・アーキテクチャ

この章では、データベース・インスタンスのメモリー・アーキテクチャについて説明します。

この章の構成は、次のとおりです。

- Oracle Databaseのメモリー構造の概要
- ユーザー・グローバル領域の概要
- プログラム・グローバル領域(PGA)の概要
- システム・グローバル領域(SGA)の概要
- ソフトウェア・コード領域の概要

関連項目:

メモリーの構成方法および管理方法は、Oracle Database管理者ガイドを参照してください

Oracle Databaseのメモリー構造の概要

インスタンスが開始されると、Oracle Databaseによってメモリー領域が割り当てられ、バックグラウンド・プロセスが起動されます。

メモリー領域には、次のような情報が格納されています。

- プログラム・コード
- 接続されているセッションについての情報(現在はアクティブかどうかは不問)
- プログラムの実行時に必要な情報(行をフェッチしている問合せの現在の状態など)
- プロセス間で共有および通信されるロック・データなどの情報
- データ・ブロックやREDOレコードなどの、ディスク上にも存在するキャッシュ・データ

関連項目:

プロセス・アーキテクチャ

基本的なメモリー構造

Oracle Databaseには、それぞれに複数のサブコンポーネントを含む、複数のメモリー領域があります。

Oracle Databaseに関連する基本的なメモリー構造は、次のような領域で構成されています。

- システム・グローバル領域(SGA)

  SGAは、*SGA*コンポーネントと呼ばれる共有メモリー構造のグループで、1つのOracle Databaseインスタンスに関するデータと制御情報が保存されています。すべてのサーバーとバックグラウンド・プロセスでSGAを共有します。SGAに格納されるデータには、キャッシュ・データ・ブロックや共有SQL領域などがあります。

- プログラム・グローバル領域(PGA)

PGAは、Oracleプロセス専用のデータや制御情報を含む、非共有メモリー領域です。Oracle Databaseでは、Oracleプロセスの起動時にPGAが作成されます。

PGAは、サーバー・プロセスごと、およびバックグラウンド・プロセスごとに存在します。個々のPGAの集合は、合計インスタンスPGAまたはインスタンスPGAと呼ばれています。個々のPGAではなく、データベース初期化パラメータを使用してインスタンスPGAのサイズを設定します。

- ユーザー・グローバル領域(UGA)

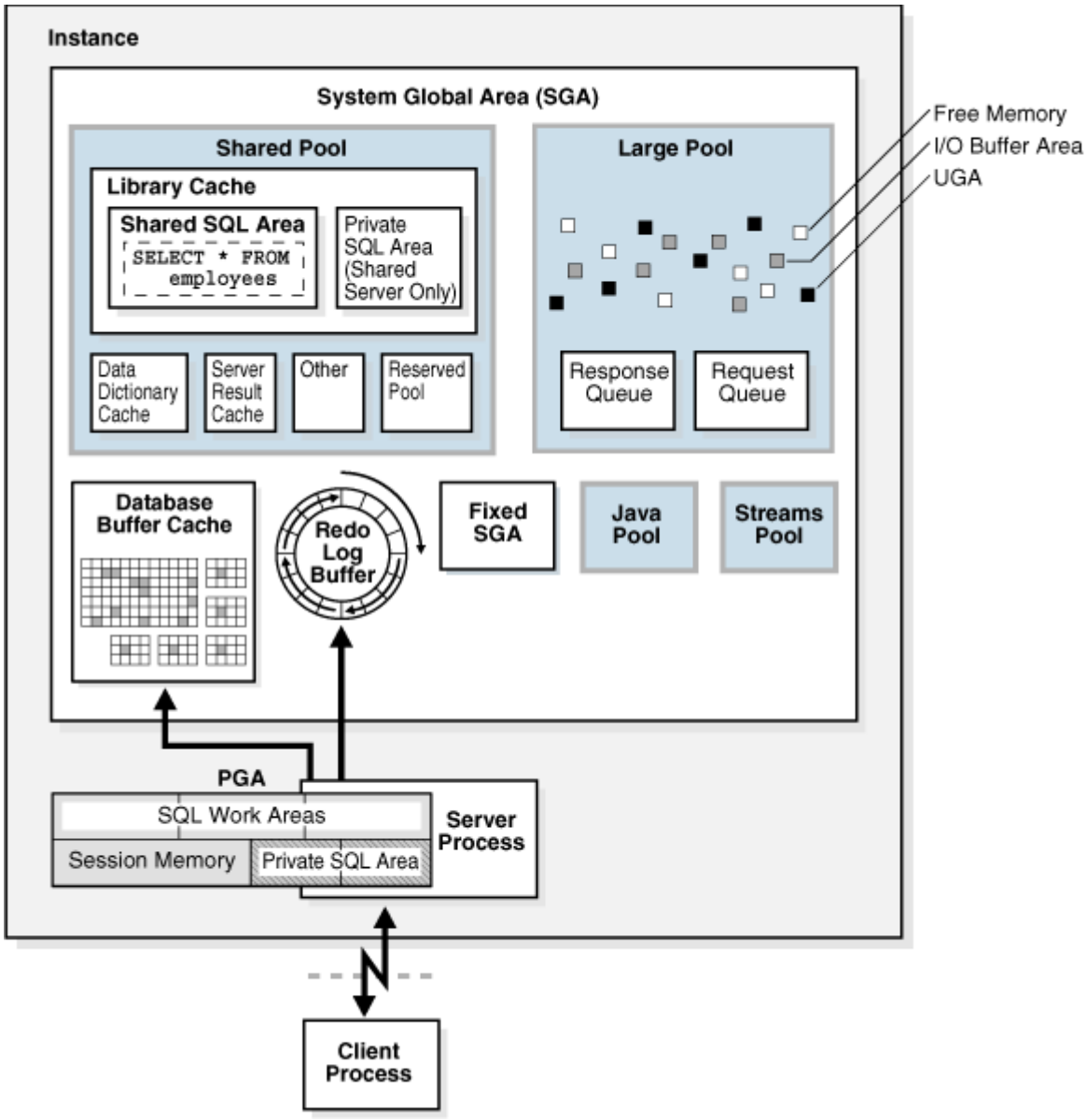
  UGAは、ユーザー・セッションに関連付けられたメモリーです。

- ソフトウェア・コード領域

  ソフトウェア・コード領域とは、実行中または実行される可能性のあるコードを格納するためのメモリー部分です。Oracle Databaseのコードはソフトウェア領域に格納され、通常、この場所はユーザー・プログラムの格納場所とは異なり、より排他的で保護されています。

次の図に、これらのメモリー構造の関係を示します。

図14-1 Oracle Databaseのメモリー構造



Oracle Databaseのメモリー管理

メモリー管理では、データベースの変化に応じて、Oracleインスタンスのメモリー構造を最適なサイズに維持します。Oracle

Databaseでは、メモリー関連の初期化パラメータの設定に基づいてメモリーが管理されます。

メモリー管理の基本的なオプションは、次のとおりです。

- 自動メモリー管理

  データベース・インスタンス・メモリーのターゲット・サイズを指定します。インスタンスは自動的にターゲット・メモリー・サイズにチューニングされ、必要に応じて、SGAとインスタンスPGAとの間でメモリーの再配分が行われます。

- 自動共有メモリー管理

  この管理モードでは、一部の処理が自動化されます。SGAは、ターゲット・サイズを設定し、PGAは、ターゲット・サイズの合計を設定するか、作業領域を個別に管理できます。

- 手動メモリー管理

  合計メモリー・サイズを設定するかわりに、SGAのコンポーネントおよびインスタンスPGAを個別に管理するための数多くの初期化パラメータを設定します。

Database Configuration Assistant(DBCA)を使用してデータベースを作成し、基本インストール・オプションを選択した場合のデフォルトは、自動メモリー管理です。

関連項目:

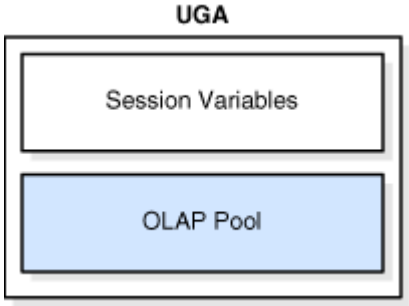
- DBAのメモリー管理オプションの詳細は、「メモリーの管理」を参照してください
- DBCAについて学習するには、「データベースのインストールおよび構成ツール」を参照してください
- メモリー管理オプションについて学習するには、『Oracle Database管理者ガイド』を参照してください

ユーザー・グローバル領域の概要

UGAはセッション・メモリーで、ログオン情報やデータベース・セッションに必要なその他の情報などのセッション変数に割り当てられるメモリーです。基本的に、UGAにはセッション状態が格納されています。

次の図に、UGAを示します。

図14-2 ユーザー・グローバル領域(UGA)



セッションによってPL/SQLパッケージがメモリーにロードされると、特定の時点においてすべてのパッケージ変数に格納されている値のセットであるパッケージ状態がUGAに格納されます。パッケージのサブプログラムによって変数が変更されると、パッケージ状態も変更されます。デフォルトで、パッケージ変数はセッションの存続期間に対して固有の変数であり、セッションの存続期間中保持されます。

UGAには、OLAPページ・プールも格納されています。このプールでは、OLAPデータ・ページが管理され、データ・ページはデータ・ブロックと同じものです。ページ・プールは、OLAPセッションの開始時に割り当てられて、セッションの終了時に解放されます。

OLAPセッションは、ユーザーがキューブなどのディメンション・オブジェクトを問い合せると、常に自動的に開かれます。

データベース・セッションの存続期間中は、セッションでUGAが使用可能である必要があります。このことは、PGAは単一のプロセス固有の領域であるため、共有サーバー接続を使用する場合にはUGAをPGAに格納できないことを意味しています。したがって、共有サーバー接続を使用する場合は、すべての共有サーバー・プロセスからUGAにアクセスできるよう、UGAはSGAに格納されます。専用サーバー接続を使用する場合は、UGAはPGAに格納されます。

関連項目:

- PL/SQLパッケージ
- 接続とセッション
- 共有サーバー接続について学習するには、Oracle Database Net Services管理者ガイドを参照してください
- Oracle OLAPの概要は、『Oracle OLAPユーザーズ・ガイド』を参照してください。

プログラム・グローバル領域(PGA)の概要

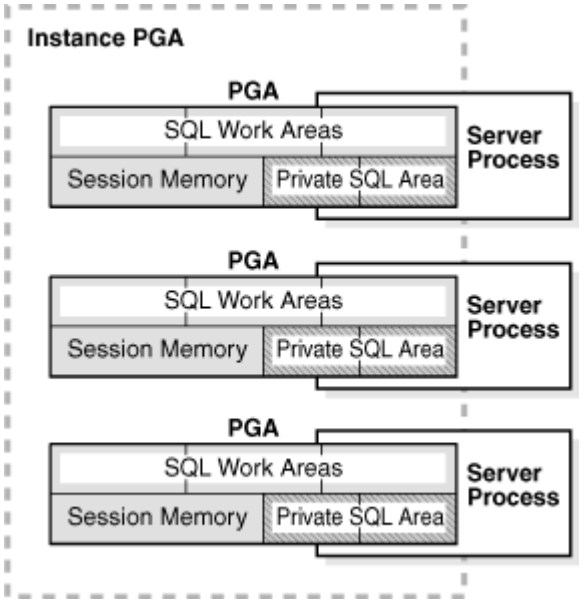
PGAは、システムの他のプロセスやスレッドによって共有されていない、動作中のプロセスやスレッドに固有のメモリーです。PGAはプロセス固有であるため、SGA内で割り当てられることはありません。

PGAは、専用サーバー・プロセスまたは共有サーバー・プロセスが必要とするセッション依存の変数を含むメモリー・ヒープです。サーバー・プロセスでは、必要なメモリー構造がPGA内に割り当てられます。

PGAは、文書係が使用するカウンタ上の一時的な作業領域に似ています。このたとえの場合、文書係はサーバー・プロセスを表しており、顧客(クライアント・プロセス)のために作業を行います。文書係は、作業のための場所をカウンタ上に準備し、その場所に顧客のリクエストの詳細をまとめ、顧客が要求するフォルダをソートし、作業が終わるとその場所を解放します。

次の図に、共有サーバー用に構成されていないインスタンスのインスタンスPGA (すべてのPGAの集合)を示します。初期化パラメータを使用して、インスタンスPGAの最大ターゲット・サイズを設定できます。個々のPGAでは、必要に応じて最大このターゲット・サイズまでサイズを増加できます。

図14-3 インスタンスPGA



ノート:

バックグラウンド・プロセスにも独自の PGA が割り当てられます。ここでは、サーバー・プロセス PGA のみを説明します。

関連項目:

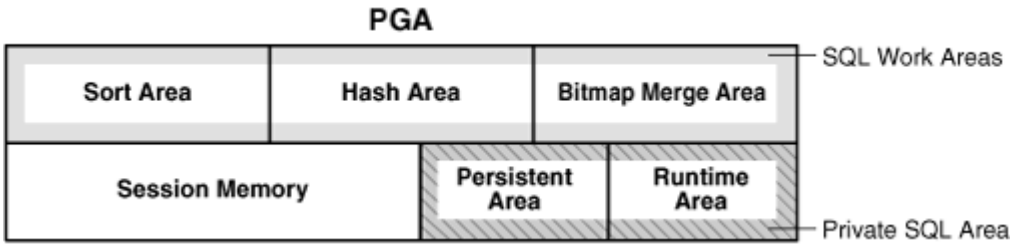
「メモリー管理方法の概要」

PGAの内容

PGAは、それぞれ異なる目的を持つ複数の領域によって構成されています。

次の図に、専用サーバー・セッションのPGAの内容を示します。どの場合にもすべてのPGA領域が揃っているとはかぎりません。

図14-4 PGAの内容



プライベートSQL領域

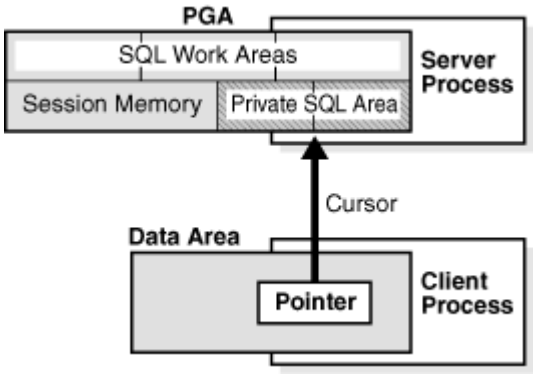
プライベートSQL領域には、解析済のSQL文に関する情報と、処理に使用するその他のセッション固有の情報が保持されます。

サーバー・プロセスでSQLまたはPL/SQLのコードが実行される場合、プロセスではプライベートSQL領域を使用してバインド変数値、問合せ実行状態情報、問合せ実行作業領域が格納されます。

PGA内のプライベートSQL領域は、SGAにおいて実行計画が格納される共有SQL領域とは異なります。同じセッション内または異なるセッション内の複数のプライベートSQL領域から、SGA内の単一の実行計画を指すことができます。たとえば、あるセッションにおけるSELECT * FROM salesの20回の実行および別のセッションにおける同じ問合せの10回の実行において、同じ計画を共有できます。各実行におけるプライベートSQL領域は共有されず、異なる値やデータが含まれている可能性があります。

カーソルは、特定のプライベートSQL領域の名前またはハンドルです。次の図に示すように、カーソルはクライアントから見るとポインタ、サーバーから見ると状態のようなものです。カーソルはプライベートSQL領域と密接に関連しているため、これらの用語は同じ意味で使用されることがあります。

図14-5 カーソル



プライベートSQL領域は、次の領域によって構成されています。

- ランタイム領域

  この領域には、問合せ実行状態情報が含まれています。たとえば、ランタイム領域では、全表スキャンで現在までに取得された行数が追跡されます。

  Oracle Databaseは、ランタイム領域を、実行要求の最初のステップとして作成されます。DML文では、SQL文が終了するとランタイム領域が解放されます。

- 持続領域

  この領域には、バインド変数値が含まれています。バインド変数値は、SQL文が実行される実行時にSQL文に指定されます。持続領域は、カーソルのクローズ時にのみ解放されます。

クライアント・プロセスは、プライベートSQL領域の管理を担当します。クライアント・プロセスが割り当てることのできるプライベートSQL領域の数は、初期化パラメータOPEN_CURSORSによって制限されますが、プライベートSQL領域の割当ておよび割当て解除は、アプリケーションに大きく依存します。

ほとんどのユーザーはデータベース・ユーティリティの自動カーソル処理を使用しますが、Oracle Databaseのプログラム・インタフェースを利用すると、開発者はカーソルをより制御できるようになります。一般的に、持続領域を解放し、アプリケーション・ユーザーが必要とするメモリーを最小限に抑えるには、オープンされているカーソルのうち再利用しないすべてのカーソルをアプリケーション側でクローズする必要があります。

関連項目:

- 「共有SQL領域」
- カーソルの使用方法を学習するには、Oracle Database開発ガイドおよびOracle Database PL/SQL言語リファレンスを参照してください

SQL作業領域

作業領域とは、メモリー集中型の操作に使用されるPGAメモリーを個別に割り当てるための領域です。

たとえば、ソート演算子では、ソート領域を使用して行のセットがソートされます。同様に、ハッシュ結合演算子ではハッシュ領域を使用して左側の入力からハッシュ表が構築され、ビットマップ・マージ演算子ではビットマップ・マージ領域を使用して複数のビットマップ索引のスキャンにより取得されたデータがマージされます。

次の例では、employeesおよびdepartmentsの結合をその問合せ計画とともに示しています。

```
SQL> SELECT *
  2  FROM   employees e JOIN departments d
  3  ON     e.department_id=d.department_id
  4  ORDER BY last_name;
.
.
.
------------------------------------------------------------------------------
| Id| Operation           | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
| 0 | SELECT STATEMENT    |             |   106 |  9328 |    7  (29)| 00:00:01 |
| 1 |  SORT ORDER BY      |             |   106 |  9328 |    7  (29)| 00:00:01 |
|*2 |   HASH JOIN         |             |   106 |  9328 |    6  (17)| 00:00:01 |
| 3 |    TABLE ACCESS FULL| DEPARTMENTS |    27 |   540 |    2   (0)| 00:00:01 |
```

```
| 4 |    TABLE ACCESS FULL| EMPLOYEES    |   107 |  7276 |    3   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

前述の例では、ランタイム領域で全表スキャンの進行状況が追跡されます。セッションでは、ハッシュ領域を使用してハッシュ結合が実行されて、2つの表の行が照合されます。ORDER BYによるソートは、ソート領域で行われます。

演算子で処理するデータが作業領域に収まらない場合は、Oracle Databaseによって入力データがより小さく分割されます。データベースでは、このようにして一部のデータがメモリー内で処理され、残りのデータは後で処理するために一時ディスク記憶域に格納されます。

自動PGAメモリー管理が有効になっている場合、作業領域サイズはデータベースにより自動的にチューニングされます。作業領域のサイズを手動で制御してチューニングすることもできます。詳細は、「メモリーの管理」を参照してください。

通常、作業領域を増やすとメモリー使用率は増えますが、演算子のパフォーマンスが大幅に改善されます。作業領域は、入力データや関連するSQL演算子によって割り当てられる補助メモリー構造を格納するために十分なサイズにすることをお薦めします。十分なサイズが確保されていない場合は、一部の入力データがディスクにキャッシュされるため、レスポンス時間が長くなります。作業領域のサイズが入力データ・サイズに比べて非常に小さい場合は、データを分割して何度も渡すことが必要となるため、レスポンス時間が大幅に長くなります。

関連項目:

- 自動PGA管理の使用方法を学習するには、Oracle Database管理者ガイドを参照してください
- PGAメモリーのチューニング方法を学習するには、Oracle Databaseパフォーマンス・チューニング・ガイドを参照してください

専用サーバー・モードおよび共有サーバー・モードにおけるPGAの使用

PGAにメモリーが割り当てられるかどうかは、データベースで専用サーバー接続が使用されるか、共有サーバー接続が使用されるかに応じて異なります。

次の表にこの相違を示します。

表14-1 専用サーバーと共有サーバーのメモリー割当て方法の相違点

メモリー領域	専用サーバー	共有サーバー
セッション・メモリーの性質	プライベート	共有
持続領域の位置	PGA	SGA
DML および DDL 文のランタイム領域の位置	PGA	PGA

関連項目:

共有サーバーを使用するためのデータベースの構成方法を学習するには、『Oracle Database管理者ガイド』を参照してください。

システム・グローバル領域(SGA)の概要

SGAは、Oracleバックグラウンド・プロセスとともにデータベース・インスタンスを構成する、読取り/書込み可能なメモリー領域です。

ノート:

サーバー・プロセスおよびバックグラウンド・プロセスは、SGA 内ではなく、別のメモリー領域内に存在しています。

ユーザー用に実行されるすべてのサーバー・プロセスで、インスタンスSGA内の情報の読取りが可能です。データベース操作中にSGAに書き込むプロセスもあります。

各データベース・インスタンスには、独自のSGAがあります。Oracle Databaseでは、インスタンスの開始時にSGAのメモリーが自動的に割り当てられて、インスタンスが停止するとメモリーが再利用されます。SQL*PlusまたはOracle Enterprise Managerを使用してインスタンスを開始した場合、SGAは次の例に示すサイズになります。

```
SQL> STARTUP
ORACLE instance started.

Total System Global Area  368283648 bytes
Fixed Size                  1300440 bytes
Variable Size             343935016 bytes
Database Buffers           16777216 bytes
Redo Buffers                6270976 bytes
Database mounted.
Database opened.
```

図14-1に示すように、SGAはいくつかのメモリー・コンポーネントによって構成され、これらのコンポーネントは、特定の種類のメモリー割当て要求を満たすために使用されるメモリーのプールです。REDOログ・バッファを除くすべてのSGAコンポーネントは、グラニュルと呼ばれる連続したメモリーの単位で割当ておよび割当て解除されます。グラニュルのサイズはプラットフォーム固有であり、SGAサイズの合計から決定されます。

SGAコンポーネントの情報は、V$SGASTATビューに問い合せることができます。

次に、最も重要なSGAコンポーネントを示します。

- データベース・バッファ・キャッシュ
- インメモリー領域
- REDOログ・バッファ
- 共有プール
- ラージ・プール
- Javaプール
- 固定SGA
- オプションのパフォーマンス関連SGAサブ領域

関連項目:

- 「Oracleデータベース・インスタンスの概要」
- グラニュルのサイズ設定についてさらに学習するには、『Oracle Databaseパフォーマンス・チューニング・ガイド』を参照してください

データベース・バッファ・キャッシュ

データベース・バッファ・キャッシュとも呼ばれる**データベース・バッファ・キャッシュ**とは、データファイルから読み取られたデータ・ブロックのコピーを格納するメモリー領域のことです。

バッファとは、現在使用されているデータ・ブロックまたは最近使用されたデータ・ブロックがバッファ・マネージャによって一時的にキャッシュされるメイン・メモリーのアドレスのことです。データベース・インスタンスに同時接続されたユーザーはすべて、バッファ・キャッシュへのアクセスを共有します。

データベース・バッファ・キャッシュの目的

Oracle Databaseでは、複数の目的でバッファ・キャッシュが使用されます。

目的には次のものがあります。

- 物理I/Oを最適化します。

  データベースでは、キャッシュ内のデータ・ブロックが更新されて、REDOログ・バッファに変更についてのメタデータが格納されます。COMMITの後、データベースによってREDOバッファがオンラインREDOログに書き込まれますが、データ・ブロックが即座にデータ・ファイルに書き込まれるわけではありません。かわりに、データベース・ライター(DBW)によって、バックグラウンドで遅延書込みが実行されます。

- 頻繁にアクセスされるブロックをバッファ・キャッシュに保持して、頻繁にアクセスされないブロックはディスクに書き込みます。

  データベース・スマート・フラッシュ・キャッシュ(フラッシュ・キャッシュ)を有効にすると、バッファ・キャッシュの一部をフラッシュ・キャッシュ内に保持できます。このバッファ・キャッシュ拡張は、フラッシュ・メモリーを使用するソリッド状態の記憶装置である1つ以上のフラッシュ・ディスク装置に格納されます。フラッシュ・メモリー内にバッファがキャッシュされ、磁気ディスクからの読取りが必要なくなるため、データベースのパフォーマンスが向上します。

  DB_FLASH_CACHE_FILEおよびDB_FLASH_CACHE_SIZE初期化パラメータを使用して、複数のフラッシュ・デバイスを構成します。バッファ・キャッシュでは、各デバイスを追跡し、デバイスに均等にバッファを分配します。

ノート:

データベース・スマート・フラッシュ・キャッシュは、Solaris および Oracle Linux 環境でのみ使用できます。

関連項目:

DB_FLASH_CACHE_FILE初期化パラメータについて学習するには、『Oracle Databaseリファレンス』を参照してください

バッファ状態

データベースでは、内部アルゴリズムを使用してキャッシュ内のバッファが管理されます。

バッファには、次のような相互に排他的な状態があります。

- 未使用

  このバッファはまだ使用されていないか、現在使用されていないため、使用可能です。このタイプのバッファは、最も使用しやすいバッファです。

- クリーン

  このバッファは過去に使用され、現在はある時点において読取り一貫性があるバージョンのブロックが含まれています。このブロックにはデータが含まれていますが、クリーンであるため、チェックポイントを作成する必要はありません。データベースでは、このブロックを確保して再利用できます。

- 使用済

  このバッファには、まだディスクに書き込まれていない変更されたデータが含まれています。データベースでこのバッファを再利用する前に、このブロックのチェックポイントを作成する必要があります。

すべてのバッファには、使用中または使用可能(確保解除)というアクセス・モードがあります。バッファは、ユーザー・セッションによってアクセスされている間にメモリーからエージ・アウトされないように、キャッシュ内で確保されます。確保されたバッファを複数のセッションから同時に変更することはできません。

バッファ・モード

クライアントがデータを要求すると、Oracle Databaseでは、現在のモードまたは一貫したモードいずれかでデータベース・バッファ・キャッシュからバッファが取得されます。

モードの違いは次のとおりです。

- 現在のモード

  現在のモード取得は、*DB*ブロック取得とも呼ばれ、バッファ・キャッシュ内の現行のブロックをそのまま取得します。たとえば、コミットされていないトランザクションによってブロック内の2つの行が更新された場合、現行モード取得により、これらのコミットされていない行を含むブロックが取得されます。Oracle Databaseでは、ブロックの現行バージョンのみを更新する必要がある変更文を実行する際に、最も頻繁にDBブロック取得が使用されます。

- 一貫したモード

  読取り一貫性取得は、ブロックの読取り一貫性バージョンを取得します。この取得では、UNDOデータが使用される場合があります。たとえば、トランザクションによってブロック内の2つの行が更新されたがまだコミットされていない場合に、別のセッションの問合せがそのブロックを要求すると、Oracle DatabaseではUNDOデータを使用して、コミットされていない更新を含まない、このブロックの読取り一貫性バージョン(読取り一貫性クローンと呼ばれます)が作成されます。一般に、問合せは一貫性モードでブロックを取得します。

関連項目:

- 「読取り一貫性とUNDOセグメント」
- DBブロック取得や読取り一貫性取得などのデータベース統計の詳細は、Oracle Databaseリファレンスを参照してください

バッファI/O

論理I/Oは、バッファ*I/O*とも呼ばれ、バッファ・キャッシュ内でのバッファの読取りおよび書込みを表しています。

要求されたバッファがメモリーにない場合、Oracle Databaseは物理I/Oを実行してフラッシュ・キャッシュまたはディスクからメモリーにバッファをコピーします。その後、論理I/Oを実行して、キャッシュされたバッファを読み取ります。

バッファ置換アルゴリズム

効率的にバッファにアクセスできるようにするため、データベースはメモリーにキャッシュするバッファと、ディスクからアクセスするバッファを決定する必要があります。

データベースでは次のアルゴリズムを使用します。

- LRUベースでブロック・レベルの置換アルゴリズム

  この高度なアルゴリズムはデフォルトのアルゴリズムで、使用済バッファと使用済でないバッファへのポインタを含む、最近の使用頻度が最も低い(LRU)リストを使用します。LRUリストにはホット・エンドとコールド・エンドがあります。コールド・バッファは、最近使用されていないバッファです。ホット・バッファは、頻繁にアクセスされ、最近使用されたバッファです。概念的にはLRUは1つのみですが、データベースでは実際にはデータ同時実行性のために複数のLRUが使用されています。

- 温度ベースでオブジェクト・レベルの置換アルゴリズム

  Oracle Database 12cリリース1 (12.1.0.2)以降、自動ビッグ・テーブル・キャッシング機能により、表スキャンで次のシナリオに異なるアルゴリズムを使用できるようになりました。

  - パラレル問合せ

    単一インスタンスおよびOracle Real Applications Cluster (Oracle RAC)データベースでは、DB_BIG_TABLE_CACHE_PERCENT_TARGET初期化パラメータがゼロ以外の値に設定され、PARALLEL_DEGREE_POLICYがautoまたはadaptiveに設定されている場合、パラレル問合せでビッグ・テーブル・キャッシュを使用できます。

  - シリアル問合せ

    単一インスタンス構成の場合のみ、DB_BIG_TABLE_CACHE_PERCENT_TARGET初期化パラメータがゼロ以外の値に設定されている場合、シリアル問合せでビッグ・テーブル・キャッシュを使用できます。

  表がメモリー内に収まらない場合、データベースはアクセス・パターンに基づいてキャッシュするバッファを決定します。ポピュラーな表の95%しかメモリー内に収まらない場合、データベースは、周期的にブロックをメモリーに読み込んでブロックをディスクに書き込むかわりに(スラッシングと呼ばれる現象)、ブロックの5%をディスクに残す選択を行うことができます。複数のラージ・オブジェクトをキャッシュする際、データベースはよりポピュラーな表は人気がありポピュラーでない表は人気がないと見なし、これはキャッシュするブロックに影響を与えます。DB_BIG_TABLE_CACHE_PERCENT_TARGET初期化パラメータは、このアルゴリズムを使用するバッファ・キャッシュの比率を設定します。

ノート:

このドキュメントでは、LRU ベースでブロック・レベルの置換アルゴリズムについて説明します。

関連項目:

温度ベースのアルゴリズムについてさらに学習するには、『Oracle Database VLDBおよびパーティショニング・ガイド』を参照

バッファ書込み

データベース・ライター(DBW)プロセスでは、コールドかつ使用済のバッファが定期的にディスクに書き込まれます。

DBWでは、次の状況でバッファが書き込まれます。

- サーバー・プロセスで、新しいブロックをデータベース・バッファ・キャッシュに読み取るときにクリーンなバッファが見つからなかった場合。

  バッファが使用されるにつれて、使用可能バッファの数が減少します。使用可能バッファの数が内部のしきい値未満になり、クリーンなバッファが必要な場合には、サーバー・プロセスからDBWに対して、書込みを行うように通知されます。

  Oracle Databaseでは、LRUを使用して、どの使用済バッファに書き込むかが決定されます。使用済バッファがLRUのコールド・エンドに達すると、そのバッファはLRUから書込みキューに移動されます。DBWによって、可能な場合にはマルチブロック書込みを使用して、キュー内のバッファがディスクに書き込まれます。このメカニズムによって、LRUの終端に使用済バッファが滞留しなくなり、クリーンなバッファを検出して再利用できるようになります。

- データベースで、REDOスレッド内のインスタンス・リカバリを開始する位置であるチェックポイントを進める必要がある場合。

- 表領域が読取り専用ステータスに変更されたか、オフラインにされた場合。

関連項目:

- 「データベース・ライター・プロセス(DBW)」

- バッファ書込みの問題を診断およびチューニングする方法を学習するには、Oracle Databaseパフォーマンス・チューニング・ガイドを参照してください

バッファ読取り

未使用バッファの数が減少したら、データベースにより、バッファ・キャッシュからバッファが削除されます。

アルゴリズムはフラッシュ・キャッシュが有効かどうかに応じて変わります。

- フラッシュ・キャッシュが無効な場合

  各クリーン・バッファは、必要に応じて上書きされ、再使用されます。上書きされたバッファが後から必要になった場合は、そのバッファが磁気ディスクから読み取られます。

- フラッシュ・キャッシュが有効な場合

  DBWがフラッシュ・キャッシュにクリーン・バッファ本体を書き込めるため、インメモリーのバッファの再利用が可能になります。データベースは、フラッシュ・キャッシュ内のバッファ本体の状態と場所を追跡できるように、メイン・メモリー内のLRUリストにバッファのヘッダーを保持します。バッファが後で必要になったときに、データベースは磁気ディスクからではなく、フラッシュ・キャッシュからバッファを読み取ります。

クライアント・プロセスからバッファが要求されると、サーバー・プロセスは最初にバッファ・キャッシュ内でそのバッファを検索します。データベースによるメモリー内のバッファの検索時には、キャッシュ・ヒットの処理が実行されます。検索順序は、次のとおりです。

1. サーバー・プロセスによってバッファ・キャッシュ内にバッファ全体があるかどうかが検索されます。

バッファ全体が見つかった場合は、データベースによってそのバッファの論理読取りが実行されます。

2. サーバー・プロセスによってフラッシュ・キャッシュのLRUリスト内にバッファのヘッダーがあるかどうかが検索されます。

   バッファのヘッダーが見つかった場合、データベースによってフラッシュ・キャッシュからインメモリーのキャッシュにバッファ本体の最適化物理読取りが実行されます。

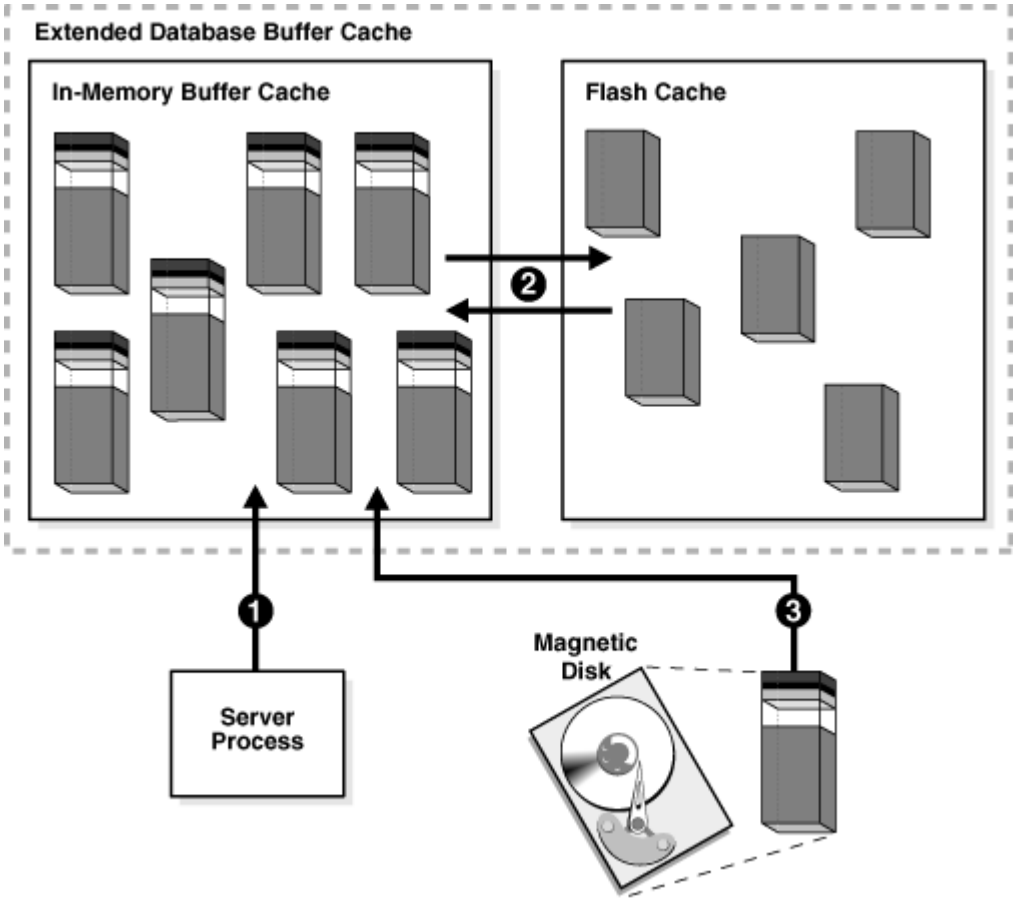
3. キャッシュ・メモリー内にバッファが見つからない場合は(キャッシュ・ミス)、サーバー・プロセスによって次のステップが実行されます。

   a. データファイルからメモリーへのブロックのコピー(物理読取り)

   b. メモリーに読み取られたバッファの論理読取りの実行

図14-6に、バッファの検索順序を示します。拡張バッファ・キャッシュは、バッファ全体を格納するインメモリーのバッファ・キャッシュと、バッファ本体を格納するフラッシュ・キャッシュから構成されます。図では、データベースによってバッファ・キャッシュ内にバッファがあるかどうか検索され、バッファが見つからなければ、磁気ディスクからメモリーにバッファが読み込まれます。

図14-6 バッファの検索



通常、キャッシュにヒットした場合のデータへのアクセスは、キャッシュ・ミスの場合よりも高速です。バッファ・キャッシュ・ヒット率は、要求されたブロックがバッファ・キャッシュ内に見つかり、ディスクからの読取りが不要である確率を示しています。

データファイルまたは一時ファイルから物理読取りが実行されます。データファイルからの読取りの後には、論理I/Oが実行されます。一時ファイルからの読取りは、メモリー不足によってデータベースがデータを一時表に書き込み、後からそのデータを読み取る必要がある場合に行われます。このような物理読取りでは、バッファ・キャッシュにはアクセスされないため、論理I/Oは発生しません。

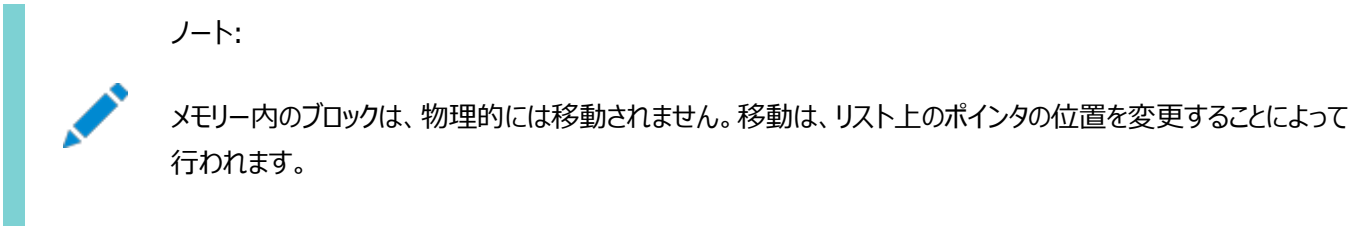
関連項目:

バッファ・キャッシュ・ヒット率の計算方法を学習するには、Oracle Databaseパフォーマンス・チューニング・ガイドを参照してくだ

さい

バッファのタッチ・カウント

データベースでは、タッチ・カウントを使用して、LRUリスト上のバッファへのアクセス頻度が測定されます。このメカニズムによって、バッファが確保されたときに常にLRUリスト上でバッファをシャッフルするかわりに、カウンタを増加できます。



ノート:

メモリー内のブロックは、物理的には移動されません。移動は、リスト上のポインタの位置を変更することによって行われます。

バッファが確保されると、最後にタッチ・カウントが増加された時間が特定されます。カウントが3秒前より以前に増加されている場合は、カウントが増加され、そうでない場合は、カウントはそのままです。この3秒ルールによって、バッファが非常に多くの回数確保されたときに、同じ数だけタッチ・カウントが増加されることを防止できます。たとえば、セッションでデータ・ブロックに複数の行を挿入する場合に、データベースではこれらの挿入が1回のタッチであるとみなされます。

バッファがLRUのコールド・エンドにあり、タッチ・カウントが大きい場合、バッファはホット・エンドに移動されます。タッチ・カウントが小さい場合、バッファはキャッシュからエージ・アウトされます。

バッファ・プール

バッファ・プールとは、バッファの集合のことです。

データベース・バッファ・キャッシュは、1つ以上のバッファ・プールによって構成されていて、これらのバッファ・プールはほとんど同じ方法でブロックを管理します。プールのアルゴリズムは、経過期間ブロックとキャッシュ・ブロックで根本的には異なりません。

バッファ・キャッシュ内のデータを保持するバッファ・プール、または新しいデータがデータ・ブロックを使用した直後に使用可能なバッファを提供するバッファ・プールを手動で個別に構成できます。その後、適切なバッファ・プールに特定のスキーマ・オブジェクトを割り当て、ブロックがキャッシュからエージ・アウトされる方法を制御できます。たとえば、セグメントをホット、ウォームおよびコールド・バッファ・プールに分割することができます。

次のバッファ・プールがあります。

- DEFAULTプール

  このプールは、ブロックが通常キャッシュされる場所です。手動で個別にプールを構成しない場合は、DEFAULTプールが唯一のバッファ・プールとなります。他のプールのオプション構成によって、デフォルト・プールは影響を受けません。

  Oracle Database 12cリリース1 (12.1.0.2)以降、ビッグ・テーブル・キャッシュは温度ベースでオブジェクト・レベルの置換アルゴリズムを使用するDEFAULTプールのオプションのセクションとなっています。単一インスタンスおよびOracle RACデータベースでは、DB_BIG_TABLE_CACHE_PERCENT_TARGET初期化パラメータがゼロ以外の値に設定され、PARALLEL_DEGREE_POLICYがautoまたはadaptiveに設定されている場合、パラレル問合せでビッグ・テーブル・キャッシュを使用できます。単一インスタンス構成の場合のみ、DB_BIG_TABLE_CACHE_PERCENT_TARGETが設定されている場合、シリアル問合せでビッグ・テーブル・キャッシュを使用できます。

- KEEPプール

  このプールは、頻繁にアクセスされたが、領域が不足したためにDEFAULTプールからエージ・アウトされたブロック用のプールです。KEEPバッファ・プールの目的は、メモリー内にオブジェクトを保存して、I/O操作を避けることにあります。

ノート:

KEEP プールは他のプールと同じようにバッファを管理し、バッファをピン留めするために特別なアルゴリズムを使用しません。KEEP という単語は命名規則です。保存する表はより大規模な保存プールに配置し、保存しない表はより小規模なリサイクル・プールに配置できます。

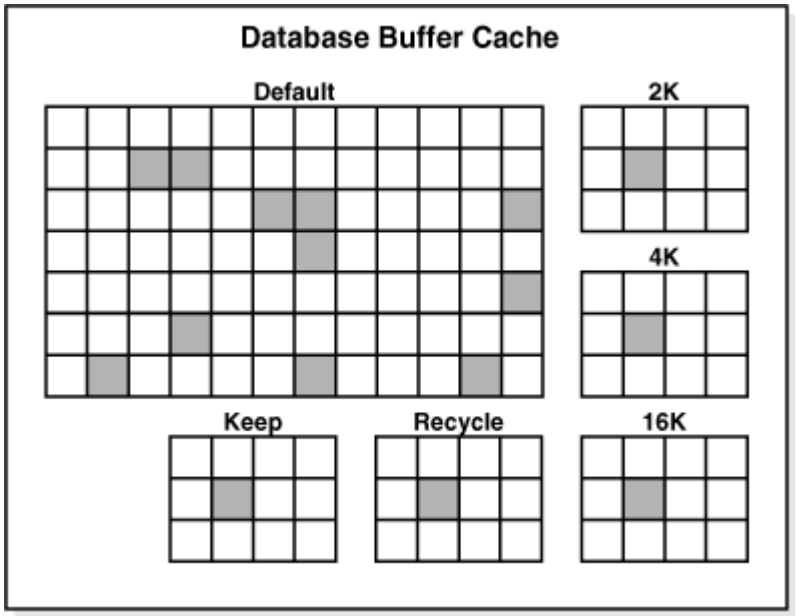
- RECYCLEプール

  このプールは、頻繁に使用されないブロック用のプールです。RECYCLEプールによって、キャッシュ内の領域が不要なオブジェクトによって消費されることを防止できます。

データベースには、標準ブロック・サイズがあります。標準のサイズとは異なるブロック・サイズを持つ表領域を作成できます。非デフォルトのブロック・サイズには、それぞれ独自のプールがあります。Oracle Databaseでは、これらのプールのブロックもDEFAULTプールと同様に管理されます。

次の図に、複数のプールが使用された場合のバッファ・キャッシュの構造を示します。このキャッシュには、DEFAULTプール、KEEPプールおよびRECYCLEプールが含まれています。デフォルトのブロック・サイズは8KBです。このキャッシュには、2KB、4KBおよび16KBという非標準のブロック・サイズを使用する表領域用の個別のプールが含まれています。

図14-7 データベース・バッファ・キャッシュ



関連項目:

- 「データベース・ブロック・サイズ」
- バッファ・プールについてさらに学習するには、Oracle Database管理者ガイドを参照してください
- 複数のバッファ・プールを使用する方法を学習するには、Oracle Databaseパフォーマンス・チューニング・ガイドを参照してください
- DB_BIG_TABLE_CACHE_PERCENT_TARGET初期化パラメータについて学習するには、Oracle Databaseリファレンスを参照してください

バッファと全表スキャン

データベースは複雑なアルゴリズムを使用して表スキャンを管理します。デフォルトでは、バッファをディスクから読み取る必要がある場合、そのバッファはLRUリストの中央に挿入されます。これにより、ホット・ブロックがキャッシュ内に維持されて、再度ディスクから読み取る必要がなくなります。

表の最高水位標(HWM)より下のすべての行を順次読み取る全表スキャンでは、問題が発生します。表セグメント内のブロックの合計サイズが、バッファ・キャッシュのサイズよりも大きいとします。この表を全体スキャンすると、バッファ・キャッシュが一掃されて、頻繁にアクセスされるブロックのキャッシュが維持されなくなる可能性があります。

関連項目:

「セグメント領域と最高水位標」

完全表スキャンのデフォルト・モード

デフォルトでは、データベースは全表スキャンに対して厳密的なアプローチを取り、テーブル・サイズの割合がバッファ・キャッシュに対して小さい場合のみ、小規模な表をメモリーにロードします。

データベースは、中規模の表をキャッシュするかどうか決定するため、最後の表スキャンの間隔、バッファ・キャッシュの経過期間タイムスタンプおよびバッファ・キャッシュに残っている容量を考慮するアルゴリズムを使用します。

通常データベースは、非常に大規模な表に対しては、ダイレクト・パス読取りを使用します。ダイレクト・パス読取りでは、バッファ・キャッシュの移入を避けるため、ブロックが直接PGAにロードされ、SGA全体が省略されます。データベースは、中規模の表に対しては、ダイレクト読取りまたはキャッシュ読取りを使用する場合があります。データベースは、キャッシュ読取りを使用することを決定した場合、スキャンによって実質的にバッファ・キャッシュが一掃されることを防止するために、これらのブロックをLRUの最後に配置します。

Oracle Database 12cリリース1 (12.1.0.2)以降、データベース・インスタンスのバッファ・キャッシュでは自動的に内部計算を実行して、データベースをインスタンスSGAに完全にキャッシュするのに十分なメモリーがあるかどうか、アクセス時に表をキャッシュすることがパフォーマンス上有益かどうかを判断します。データベース全体をメモリー内に完全に収めることができ、他の様々な内部条件が満たされる場合、Oracle Databaseはデータベース内のすべての表を小規模な表として処理し、キャッシュの対象であると見なします。ただし、データベースではNOCACHE属性が付けられたLOBはキャッシュしません。

パラレル問合せの実行

全表スキャンの実行時、複数のパラレル実行サーバーを使用すると、レスポンス時間が改善されることがあります。

一部の例では、データベースに大量のメモリーがある場合、データベースでは、プログラム・グローバル領域(PGA)へのダイレクト・パス読取りを使用するかわりに、システム・グローバル領域(SGA)にパラレル問合せのデータをキャッシュできます。パラレル問合せは、リソースを使用する可能性があるため、通常では同時実行性の低いデータ・ウェアハウスで実行されます。

関連項目:

- データ・ウェアハウスの概要は、Oracle Databaseデータ・ウェアハウス・ガイドを参照してください
- パラレル実行についてさらに学習するには、Oracle Database VLDBおよびパーティショニング・ガイドを参照してください

CACHE属性

まれに、デフォルトのキャッシュ動作が望ましくないことがありますが、その場合は、ALTER TABLE ... CACHEを使用して、大規模な表からデータベース・バッファ・キャッシュにブロックが読み取られる方法を変更できます。

CACHE属性が設定された表では、データベースはバッファ・キャッシュでブロックを強制またはピン留めしません。かわりに、他の表ブロックと同じようにブロックをキャッシュからエージ・アウトします。このオプションを使用すると、大規模な表の全体スキャンを行ったときに、キャッシュ内の他のほとんどのブロックが一掃される可能性があるため、注意が必要です。

ノート:

ALTER TABLE ... CACHE を実行しても、表はキャッシュされません。

KEEP属性

大規模な表に対しては、ALTER TABLE ... STORAGE BUFFER_POOL KEEPを使用して、これらの表のブロックをKEEPプールにロードできます。

表をKEEPプールに配置すると、ブロックが保管されているバッファ・キャッシュの部分が変更されます。ブロックは、デフォルト・バッファ・プールにキャッシュされるのではなく、KEEPバッファ・プールにキャッシュされます。KEEPプールのキャッシュを制御する別個のアルゴリズムはありません。

関連項目:

- CACHE句およびKEEP属性の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。
- バッファ・キャッシュ・アドバイザ統計の解析方法を学習するには、『*Oracle Databaseパフォーマンス・チューニング・ガイド*』を参照してください。

強制全データベース・キャッシュ・モード

状況によっては、パフォーマンスを向上するために、ALTER DATABASE ... FORCE FULL DATABASE CACHING文を明示的に実行して**全データベース・キャッシュ強制モード**を有効にできます。

自動的なデフォルト・モードとは逆に、強制完全データベース・キャッシュ・モードは、データベース・バッファ・キャッシュに適しているNOCACHE LOBを含むデータベース全体を考慮します。このモードは、Oracle Database 12cリリース1 (12.1.0.2)以降で使用可能です。

ノート:

全データベース・キャッシュ強制モードを有効にしても、データベースはメモリーに強制的に入れられません。正確に言えば、データベース全体がバッファ・キャッシュにキャッシュされる対象となります。Oracle Database は、アクセスされたときにのみ表をキャッシュします。

個々のインスタンスのバッファ・キャッシュ・サイズがデータベース・サイズよりも大きい場合にのみ、完全データベース・キャッシュ強制モードを有効化することをお薦めします。このガイドラインは、単一インスタンスとOracle RACデータベースの両方に適用され

ます。ただし、Oracle RACアプリケーションが適切にパーティション化されている場合、すべてのインスタンスのバッファ・キャッシュの合計にインスタンス間の重複キャッシュ・ブロックを処理する追加領域を加えたものがデータベース・サイズよりも大きい場合は、完全データベース・キャッシュ強制モードを有効化できます。

関連項目:

- 完全データベース・キャッシュ強制モードを有効にする方法を学習するには、Oracle Database管理者ガイドを参照してください
- ALTER DATABASE ... FORCE FULL DATABASE CACHING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

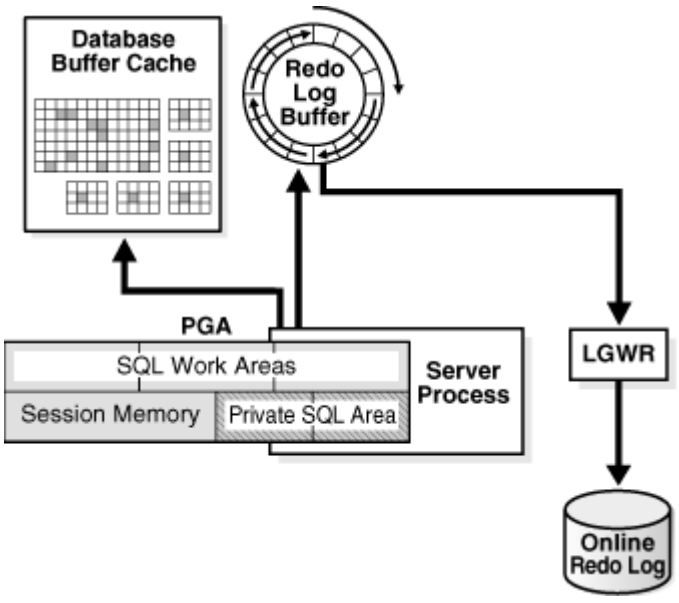
REDOログ・バッファ

REDOログ・バッファとは、SGA内の循環バッファであり、データベースに加えられた変更についてのREDOエントリが格納されます。

REDOレコードは、DML操作やDDL操作によって行われた変更を再構築または再実行するために必要な情報が含まれているデータ構造です。データベースのリカバリでは、データファイルにREDOエントリを適用して、失われた変更が再構築されます。

REDOエントリは、データベース・プロセスによってユーザーのメモリー領域からSGA内のREDOログ・バッファにコピーされます。REDOエントリは、バッファ内で連続した領域を占めます。REDOログ・バッファは、バックグラウンド・プロセスのログ・ライター・プロセス(LGWR)によって、ディスク上のアクティブなオンラインREDOログ・グループに書き込まれます。図14-8に、このREDOバッファのアクティビティを示します。

図14-8 REDOログ・バッファ



DBWではデータ・ブロックは分散してディスクに書き込まれますが、LGWRではREDOログ・バッファはディスクに対して順次書き込まれます。通常、分散書込みの速度は順次書込みよりも大幅に遅くなります。LGWRでは、ユーザーはDBWによる遅い書込みが完了するまで待機する必要がないため、パフォーマンスが向上します。

LOG_BUFFER初期化パラメータでは、Oracle DatabaseでREDOエントリをバッファリングする場合に使用するメモリー量を指定します。他のSGAコンポーネントとは異なり、REDOログ・バッファおよび固定SGAバッファのメモリーは、グラニュルに分割されません。

関連項目:

- 「ログ・ライター・プロセス(LGWR)」および「インスタンス・リカバリでのチェックポイントの重要性」
- オンラインREDOログの詳細は、Oracle Database管理者ガイドを参照してください

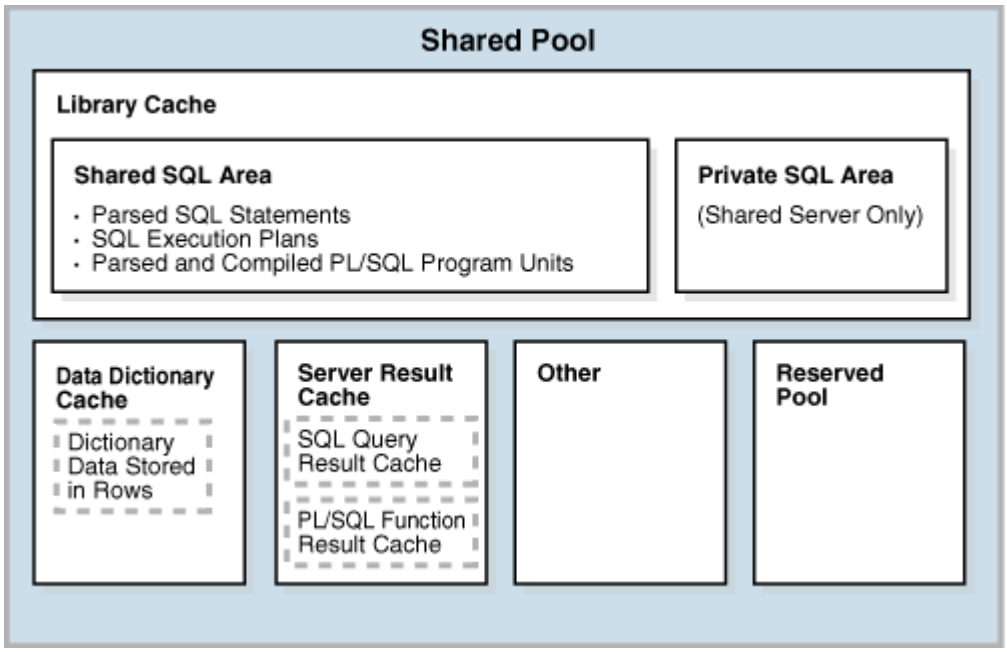
共有プール

共有プールには、様々なタイプのプログラム・データがキャッシュされます。

たとえば、共有プールには、解析済のSQLやPL/SQLのコード、システム・パラメータ、データ・ディクショナリ情報などが格納されます。共有プールは、データベースで行われるほぼすべての操作に関連します。たとえば、ユーザーがSQL文を実行する場合にも、Oracle Databaseは共有プールにアクセスします。

共有プールは、複数のサブコンポーネントから構成されています。図14-9に、最も重要なサブコンポーネントを示します。

図14-9 共有プール



この項には次のトピックが含まれます:

- ライブラリ・キャッシュ
- データ・ディクショナリ・キャッシュ
- サーバー結果キャッシュ
- 予約プール

ライブラリ・キャッシュ

ライブラリ・キャッシュとは、実行可能なSQLコードやPL/SQLコードが格納されている共有プール・メモリー構造です。

このキャッシュには、SQLとPL/SQLの共有領域や、ロックやライブラリ・キャッシュ・ハンドルなどの制御構造が含まれています。共有サーバー・アーキテクチャでは、ライブラリ・キャッシュにはプライベートSQL領域も含まれています。

SQL文が実行されると、データベースでは、以前に実行されたコードの再利用が試みられます。ライブラリ・キャッシュ内にSQL文の解析済表現が存在し、共有可能である場合には、そのコードが再利用され、これはソフト解析またはライブラリ・キャッシュ・ヒットと呼ばれます。それ以外の場合には、データベースでアプリケーション・コードの実行可能バージョンを新しく構築する必要があり、これはハード解析またはライブラリ・キャッシュ・ミスと呼ばれます。

共有SQL領域

データベースでは、実行される各SQL文の表現が共有SQL領域とプライベートSQL領域に保持されます。

SQL文が最初に実行されたときは、共有SQL領域を使用してSQL文が処理されます。この領域にはすべてのユーザーがアクセス可能であり、文の解析ツリーと実行計画が含まれています。一意の文に対して、共有SQL領域は1つのみ存在します。SQL文を発行するセッションは、それぞれのPGAにプライベートSQL領域を持っています。同じ文を発行する各ユーザーのプライベートSQL領域は、同じ共有SQL領域を指しています。これにより、異なるPGA内の数多くのプライベートSQL領域を同一の共有SQL領域に関連付けることができます。

データベースでは、アプリケーションによって同じSQL文が発行されたことが自動的に判断されます。ユーザーやアプリケーションが直接発行したSQL文も、他の文によって内部的に発行された再帰的SQL文も対象となります。

データベースでは、次のステップが実行されます。

1. 構文およびセマンティクスが同一の文について共有SQL領域が存在しているかどうかを確認するため、共有プールがチェックされます。

    - 同一の文が存在する場合、これ以降この文の新たなインスタンスが実行されるときには共有SQL領域が使用されるため、メモリー消費が削減されます。
    - 同一の文が存在しない場合は、共有プールに新しい共有SQL領域が割り当てられます。構文が同じであるもののセマンティクスが異なる文の場合は、子カーソルが使用されます。

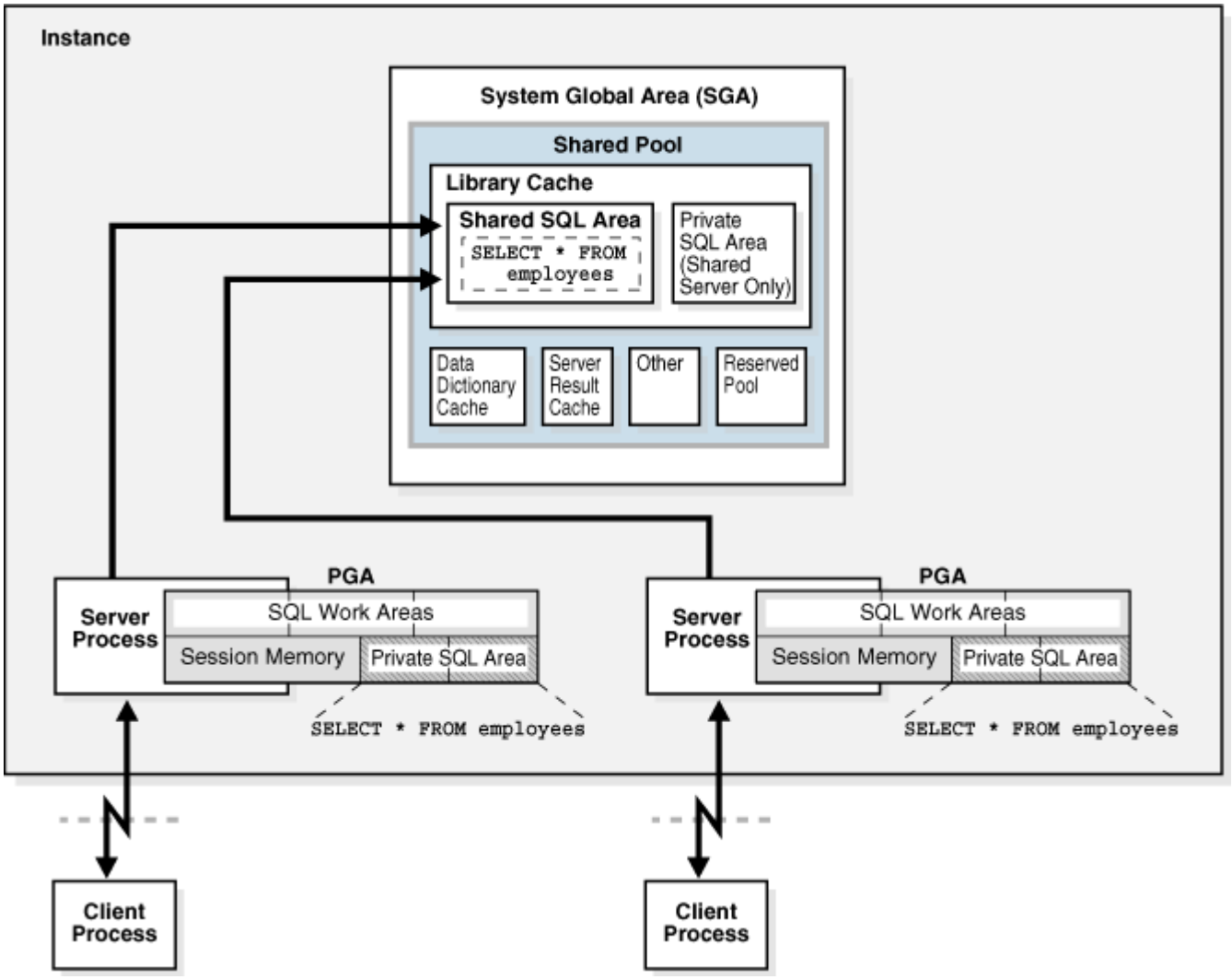
    どちらの場合も、ユーザーのプライベートSQL領域が、その文と実行計画を含む共有SQL領域に対応付けられます。

2. セッションのためにプライベートSQL領域が割り当てられます。

    プライベートSQL領域の位置は、セッションのために確立される接続に応じて異なります。セッションが共有サーバーを介して接続されている場合、プライベートSQL領域の一部はSGA内に保持されます。

図14-10に、専用サーバー・アーキテクチャにおいて、2つのセッションが同じSQL文のコピーをそれぞれのPGA内に保持する様子を示します。共有サーバーでは、このコピーはラージ・プール内のUGAにあり、ラージ・プールが存在しない場合は、共有プール内のUGAにあります。

図14-10 プライベートSQL領域と共有SQL領域



関連項目:

- 「プライベートSQL領域」
- ライブラリ・キャッシュの管理についてさらに学習するには、Oracle Databaseパフォーマンス・チューニング・ガイドを参照してください
- 共有SQLの詳細は、Oracle Database開発ガイドを参照してください

プログラム・ユニットとライブラリ・キャッシュ

ライブラリ・キャッシュには、PL/SQLプログラムおよびJavaクラスの実行可能形式が保持されます。これらは、プログラム・ユニットと総称されます。

データベースでは、プログラム・ユニットはSQL文と同様に処理されます。たとえば、データベースでは、割り当てられた共有領域にPL/SQLプログラムの解析済かつコンパイル済の形式が保持されます。データベースでは、ローカル変数、グローバル変数、パッケージ変数、SQL実行用バッファなど、プログラムを実行するセッションに固有の値を保持するためのプライベート領域が割り当てられます。複数のユーザーが同じプログラムを実行する場合は、各ユーザーが独自のプライベートSQL領域を持ち、セッション固有の値はそこに保持され、単一の共有SQL領域にアクセスします。

データベースでは、PL/SQLプログラム・ユニット内の個別のSQL文はすでに説明したように処理されます。PL/SQLプログラム・ユニット内の起点には関係なく、これらのSQL文は解析された表現を保持するために共有領域を使用し、その文を実行するセッションごとにプライベートSQL領域を使用します。

共有プール内のメモリーの割当てと再利用

データベースでは、新しいSQL文が解析されたときに共有プール・メモリーが割り当てられますが、共有可能と想定されないDDLの文である場合は例外です。割り当てられるメモリーのサイズは、文の複雑度に応じて決められます。

一般的に共有プール内の項目は、最低使用頻度(LRU)アルゴリズムに従ってデータベースが削除するまで残されたままです。多数のセッションが使用している共有プール項目は、その項目を作成したデータベース・プロセスが終了しても、その項目が使用されているかぎりメモリーに残ります。このメカニズムによって、SQL文のオーバーヘッドと処理を最小限に抑えることができます。新しい項目の領域が必要になった場合は、頻繁に使用されていない項目で使用されるメモリーが解放されます。

ALTER SYSTEM FLUSH SHARED_POOL文では共有プール内のすべての情報が削除され、同様にグローバル・データベース名も変更されます。

関連項目:

- 共有SQL領域のライフサイクルの概要は、『Oracle Database SQLチューニング・ガイド』を参照
- ALTER SYSTEM FLUSH SHARED_POOLの使用方法の詳細は、『Oracle Database SQL言語リファレンス』を参照
- V$SQLおよびV$SQLAREA動的ビューの詳細は、『Oracle Databaseリファレンス』を参照

データ・ディクショナリ・キャッシュ

データ・ディクショナリとは、データベース、データベースの構造およびそのユーザーについての参照情報を含むデータベース表およびビューの集合です。

Oracle Databaseは、SQL文の解析時にデータ・ディクショナリに頻繁にアクセスします。データ・ディクショナリはOracle Databaseによって頻繁にアクセスされるので、ディクショナリ・データを保持するために、次のような特別な場所がメモリー内に指定されています。

- データ・ディクショナリ・キャッシュ

  このキャッシュには、データベース・オブジェクトについての情報が保持されています。データがバッファではなく行として保持されるため、このキャッシュは行キャッシュとも呼ばれます。

- ライブラリ・キャッシュ

これらのキャッシュは、データ・ディクショナリ情報にアクセスするすべてのサーバー・プロセスによって共有されます。

関連項目:

- 「データ・ディクショナリと動的パフォーマンス・ビュー」
- データ・ディクショナリ・キャッシュに追加のメモリーを割り当てる方法を学習するには、Oracle Databaseパフォーマンス・チューニング・ガイドを参照してください

サーバー結果キャッシュ

サーバーの結果キャッシュは共有プール内のメモリー・プールです。バッファ・プールとは異なり、サーバー結果キャッシュにはデータ・ブロックではなく結果セットが保持されています。

サーバー結果キャッシュには、同一のインフラストラクチャを共有するSQL問合せ結果キャッシュとPL/SQLファンクション結果

キャッシュが含まれています。

ノート:

クライアント結果キャッシュは、サーバー結果キャッシュとは異なります。クライアント・キャッシュはアプリケーション・レベルで構成されており、データベース・メモリーではなくクライアント・メモリー内に配置されます。

関連項目:

- 結果キャッシュのサイズ設定の詳細は、『Oracle Database管理者ガイド』を参照してください
- DBMS_RESULT_CACHE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
- クライアント結果キャッシュの詳細は、『Oracle Databaseパフォーマンス・チューニング・ガイド』を参照してください

SQL問合せ結果キャッシュ

SQL問合せ結果キャッシュは問合せおよび問合せの断片の結果を格納するサーバー結果キャッシュのサブセットです。

ほとんどのアプリケーションにとって、このキャッシュによるパフォーマンス向上は有益です。繰り返し同じSELECT文が実行されるアプリケーションについて考えてみます。結果がキャッシュされている場合には、データベースから即座に結果が戻されます。これにより、データベースでは、ブロックの再読取りおよび結果の再計算という多くのリソースを消費する操作を行う必要がなくなります。

問合せを実行すると、データベースはメモリーを検索してその結果が結果キャッシュに存在するかどうかを判断します。結果が存在する場合、問合せを実行せずにメモリーから結果を取得します。結果がキャッシュされていない場合、データベースは問合せを実行して結果を出力として戻し、その結果を結果キャッシュに格納します。データまたはキャッシュされた結果の構成に使用されているデータベース・オブジェクトのメタデータがトランザクションで変更されると、キャッシュされた結果はデータベースにより自動的に無効化されます。

ユーザーは、RESULT_CACHEヒントを使用して問合せまたは問合せ断片に注釈を付け、結果をSQL問合せ結果キャッシュに保存することを指定できます。RESULT_CACHE_MODE初期化パラメータによって、SQL問合せ結果キャッシュが可能なかぎりすべての問合せで使用されるか、注釈が付けられた問合せにのみ使用されるかが決定されます。

関連項目:

- RESULT_CACHE_MODE初期化パラメータについてさらに学習するには、『Oracle Databaseリファレンス』を参照してください
- RESULT_CACHEヒントについて学習するには、『Oracle Database SQL言語リファレンス』を参照してください

PL/SQLファンクション結果キャッシュ

PL/SQLファンクション結果キャッシュはファンクションの結果セットを格納するサーバー結果キャッシュのサブセットです。

キャッシュがない場合には、1コール当たり1秒かかるファンクションを1000回コールすると、1000秒かかります。キャッシュがある場合は、このファンクションを同じ入力を使用して1000回コールしても全体で1秒しかかかりません。結果キャッシュは、比較的静的なデータに依存し、頻繁に起動されるファンクションに適しています。

PL/SQLファンクション・コードには、その結果をキャッシュするように要求するコードを含めることができます。このようなファンクションが起動されると、システムによってキャッシュがチェックされます。同じパラメータ値が指定された以前のファンクション・コールの結果がキャッシュに含まれている場合は、実行者にはキャッシュされた結果が戻され、ファンクション本体は再実行されません。キャッシュに結果が含まれていない場合は、ファンクション本体が実行され、実行元に制御が戻される前に、指定されたパラメータ値での結果がキャッシュに追加されます。

ノート:

キャッシュされた結果を計算するために Oracle Database で使用されるデータベース・オブジェクトを指定して、これらのオブジェクトのいずれかが更新された場合にキャッシュされた結果が無効になり、再計算されるように設定できます。

キャッシュには数多くの結果を含むことができ、ファンクションの起動に使用されたパラメータの一意の組合せごとに1つの結果がキャッシュされます。データベースでさらにメモリーが必要になった場合は、1つ以上のキャッシュ済の結果がエージ・アウトされます。

関連項目:

- PL/SQLファンクション結果キャッシュについてさらに学習するには、『Oracle Database開発ガイド』を参照してください
- PL/SQLファンクション結果キャッシュについてさらに学習するには、『Oracle Database PL/SQL言語リファレンス』を参照してください

予約プール

予約プールは共有プール内のメモリー領域であり、Oracle Databaseで大量の連続したメモリーのチャンクを割り当てる場合に使用できます。

データベースによって、共有プールからメモリーがチャンクで割り当てられます。チャンクを使用することによって、単一の連続した領域がない場合でもキャッシュに大きなオブジェクト(5KBよりも大きいオブジェクト)をロードできます。これにより、データベースでは、断片化によって連続したメモリーが使用できなくなる可能性が低減されます。

まれに、Java、PL/SQLまたはSQLカーソルによって、5KBを超える領域が共有プールから割り当てられることがあります。このような割当てが最も効率的に行われるように、データベースでは共有プールの一部分が予約プールとして分離されています。

関連項目:

予約プールの構成方法を学習するには、『Oracle Databaseパフォーマンス・チューニング・ガイド』を参照してください

ラージ・プール

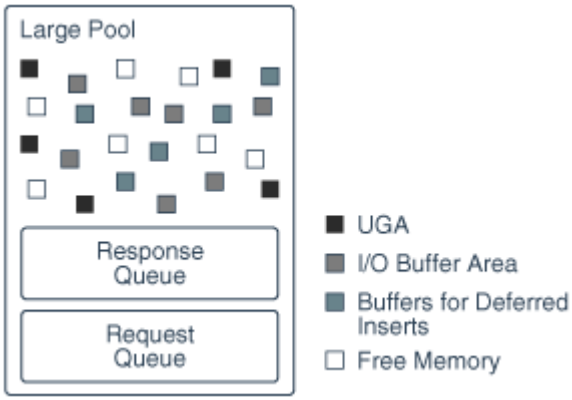
ラージ・プールは、共有プールでの割当てには適さない大量のメモリー割当て用のオプションのメモリー領域です。

ラージ・プールでは、次のような領域に対して大量のメモリー割当てを行うことができます。

- 共有サーバーおよびOracle XAインタフェース用のUGA(トランザクションが複数のデータベースと対話する環境で使用)
- パラレル実行で使用されるメッセージ・バッファ
- Recovery Manager(RMAN)I/Oスレーブ用のバッファ
- 遅延挿入のバッファ(MEMOPTIMIZE_WRITEヒントによる挿入)

次の図は、ラージ・プールを示しています。

図14-11 ラージ・プール



関連項目:

- パラレル実行用にメモリーを割り当てる方法は、「問合せコーディネータ」を参照してください
- 共有サーバーのセッション・メモリーの割当てについて学習するには、「ディスパッチャのリクエスト・キューとレスポンス・キュー」を参照してください
- Oracle XAについて学習するには、『Oracle Database開発ガイド』を参照してください。
- ラージ・プールの詳細は、Oracle Databaseパフォーマンス・チューニング・ガイドを参照してください

ラージ・プール・メモリー管理

ラージ・プールのメモリーの管理方法は共有プールとは異なります。共有プールの場合は、メモリーの一部をエージ・アウトできるようにLRUリストが使用されます。

ラージ・プールには、LRUリストはありません。データベース・セッションにラージ・プール・メモリーが割り当てられると、このメモリーは、セッションで解放されるまで、解放できません。メモリーの一部が解放されると、その領域は他のプロセスがすぐに使用できます。セッション・メモリーをラージ・プールから割り当てることにより、共有プールで発生する可能性のある断片化を防ぎます。

遅延挿入のためのラージ・プール・バッファ

遅延挿入と呼ばれる特殊なタイプの挿入の場合、データベースはバッファをラージ・プールから割り当てます。

Internet of Things (IoT)アプリケーションからのファイア・アンド・フォーゲット方式の高速な挿入においては、データベース・インフラストラクチャにおいてパフォーマンス・オーバーヘッドが発生します。たとえば、コード・パスには、バッファ・キャッシュ・ナビゲーション、バッファ・ピンおよび同時実行性保護が含まれます。配列の挿入によってこれらのコストは最小限に抑えられるものの、配列はクライアント側で構築される必要があり、これは、Internet of Thingsアプリケーションにおいては一般的ではありません。この問題に対処するために、Oracleアプリケーションではヒントを使用して、MEMOPTIMIZE FOR WRITEとして指定された表に行を挿入できます。

この挿入はラージ・プールにバッファされ、バックグラウンド・プロセスによって後から非同期にディスクに書き込まれるため、遅延されます。データベースでは遅延挿入は次のように処理されます。

1. アプリケーションはMEMOPTIMIZE_WRITE挿入を、データを集計できる中間層に送ります。ほとんどの場合、IoTアプリケーションは挿入を中間層に送信しますが、データベースに挿入を直接送信することもできます。たとえば、SQL*Plusを使用すると挿入がデータベースに直接送信されます。
2. 中間層は、挿入の集計をデータベース・サーバーに書き込みます。
3. オプションで、中間層クライアントは、前のステップで書き込んだデータのローカル・コピーを保持します。
4. サーバー・プロセスは、ラージ・プールのバッファにデータを書き込みます。

   競合を防ぐために、各バッファには独自の内部ロック・メカニズムがあります。このロック・メカニズムは、データベース・バッファ・キャッシュがバッファに対して使用するロック・メカニズムとは異なります。基本的な書込みプロセスは次のとおりです。

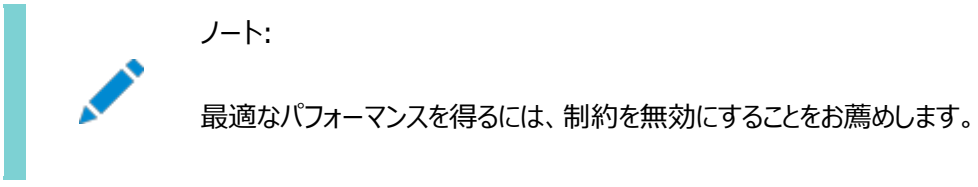
   a. インスタンスの起動後、最初のMEMOPTIMIZE_WRITE挿入によって、ラージ・プールからバッファが割り当てられます。

   b. ライターは、使用可能なバッファのリストからバッファを選択します。

   c. 選択したバッファがロックされておらず、かつそのバッファに空き領域がある場合、クライアントはそのバッファへの書込みを行い、セッション固有の順序番号が各バッファ書込みにスタンプされます。そうでない場合、ライターは前のステップに戻り、バッファが見つかるか、ラージ・プールで十分な領域が解放されるまでこの方法を続行します。
5. データベースで、バッファされたデータからサーバー側配列が作成されます。
6. Space Management Coordinator (SMCO)とそのヘルパー・プロセス(W*nnn*)は、標準のデータ・ブロックの形式を使用して配列をディスクに非同期に書き込みます。

   標準挿入とは異なり、遅延挿入は自動的にコミットされ、ロールバックできません。データベースは、指定されたオブジェクトへの挿入をセッションに出現する順序でコミットします。オブジェクトまたはセッションどうしの順序付けは保証されません。

   データベースでは、通常の挿入と同様に制約および索引のメンテナンスがサポートされます。ただし、データベースは、ラージ・プールへの書込み時ではなくディスクへの書込み時に評価を実行します。

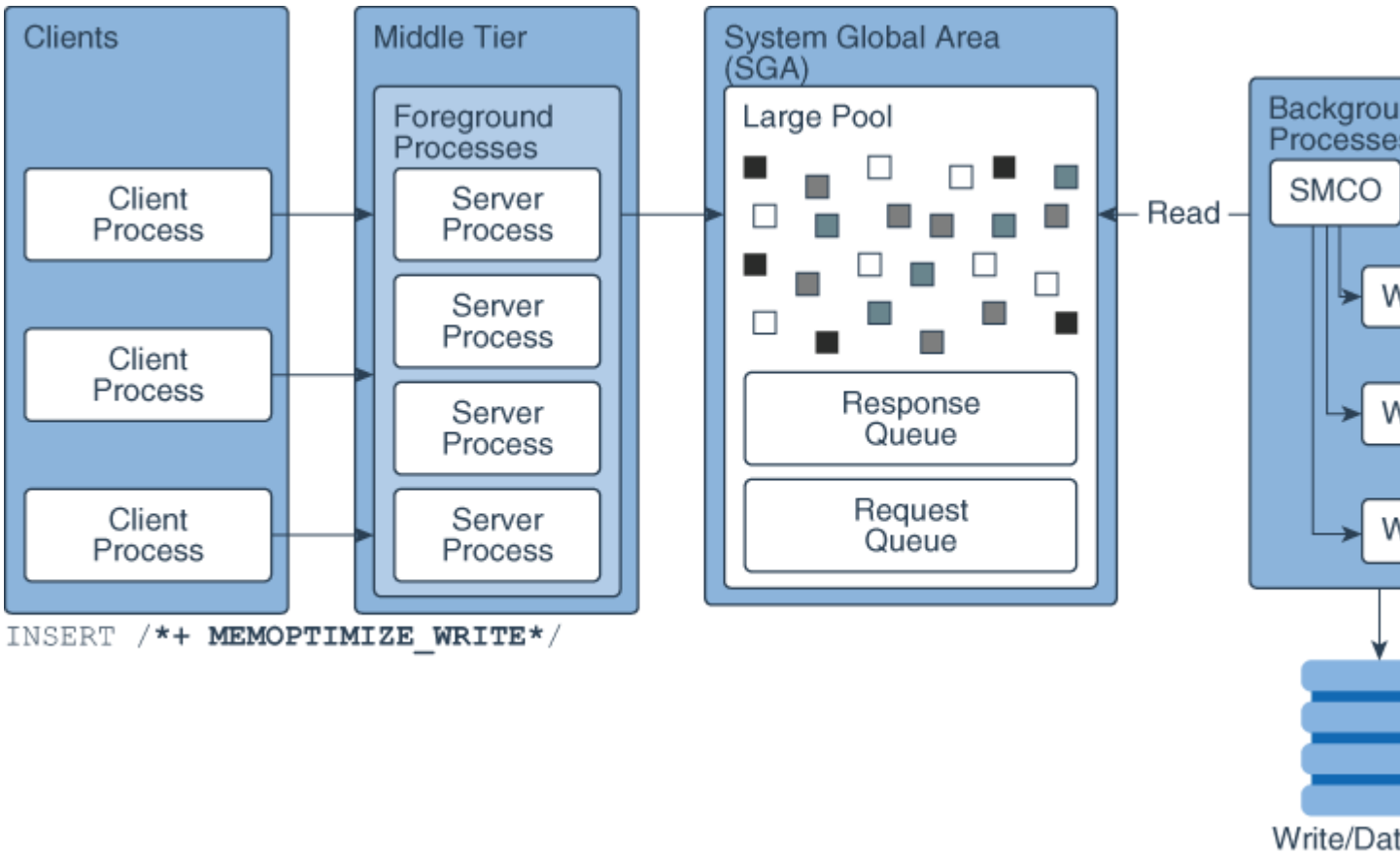


ノート:

最適なパフォーマンスを得るには、制約を無効にすることをお薦めします。

次の図は、遅延挿入のワークフローを示しています。

図14-12 遅延挿入のメカニズム



関連項目:

- 「読取り一貫性および遅延挿入」
- 表をMEMOPTIMIZE FOR WRITEとして指定する方法を学習するには、『Oracle Databaseパフォーマンス・チューニング・ガイド』を参照してください
- INSERTの構文およびセマンティクスについては、Oracle Database SQL言語リファレンスを参照してください
- MEMOPTIMIZE_POOL_SIZEについてさらに学習するには、Oracle Databaseリファレンスを参照してください。

Javaプール

Javaプールは、Java仮想マシン(JVM)内のセッション固有のすべてのJavaコードとデータが格納されるメモリー領域です。このメモリーには、コール終了時にJavaセッション領域に移行されるJavaオブジェクトが含まれています。

専用サーバー接続では、Javaプールには各Javaクラスの共有部分(メソッドおよびコード配列などの読取り専用メモリー)が含まれていますが、各セッションのセッションごとのJava状態は含まれていません。共有サーバーでは、プールには、各クラスの共有部分および各セッションの状態に使用される一部のUGAが含まれています。各UGAは必要に応じて拡大および縮小されますが、UGAの合計サイズはJavaプール領域に収まる必要があります。

Java Pool Advisor統計は、Javaに使用されるライブラリ・キャッシュ・メモリーに関する情報を提供し、Javaプールのサイズの変化が解析率に及ぼす影響を予測します。Java Pool Advisorは、statistics_levelをTYPICAL以上に設定すると内部的にオンに設定されます。これらの統計は、アドバイザをオフにするとリセットされます。

関連項目:

- Oracle Database Java開発者ガイド
- Javaプール・アドバイザ統計を含むビューについて学習するには、『Oracle Databaseパフォーマンス・チューニング・ガイド』を参照してください

固定SGA

固定SGAは、内部のハウスキーピング用領域です。

固定SGAには、たとえば次のような情報が含まれています。

- バックグラウンド・プロセスがアクセスする必要のある、データベースとインスタンスの状態に関する一般的な情報
- ロックに関する情報など、プロセス間で通信される情報

固定SGAのサイズはOracle Databaseによって設定され、手動では変更できません。固定SGAのサイズは、リリースによって異なります。

関連項目:

「自動ロックの概要」

オプションのパフォーマンス関連SGAサブ領域

一部のSGA領域は、特定のパフォーマンス機能のみが使用できます。

この項では、次の項目について説明します。

- インメモリー領域
- memoptimizeプール

インメモリー領域

インメモリー領域は、**インメモリー列ストア**(IM列ストア)が含まれるオプションのSGAコンポーネントです。

IM列ストアには、迅速にスキャンできるように最適化された列形式で表のコピー、パーティションおよびマテリアライズド・ビューが格納されています。IM列ストアは、従来の行形式でデータを格納するデータベースバッファ・キャッシュを補完するものです。

ノート:

IM 列ストアを有効にするには、Oracle Database インメモリー・オプションが必要です。

関連項目:

インメモリー領域およびIM列ストアについてさらに学習するには、Oracle Database In-Memoryガイドを参照してください

Memoptimizeプール

memoptimizeプールには、バッファおよびMEMOPTIMIZE FOR READと指定されたヒープ構成表の関連構造が格納されます。

この構造は、SELECT * FROM cust WHERE cid = 10などのキー・ベースの問合せに対して優れたパフォーマンスとスケーラビリティを提供します。エンドツーエンドのレスポンス時間を短縮するために、クライアントはリクエストしたバッファをネットワーク経由でSGAから直接プルし、CPUとオペレーティング・システムのオーバーヘッドを回避します。アプリケーションは、コードを変更することなくmemoptimizeプールを利用できます。

memoptimizeプールは次の2つのパートで構成されています。

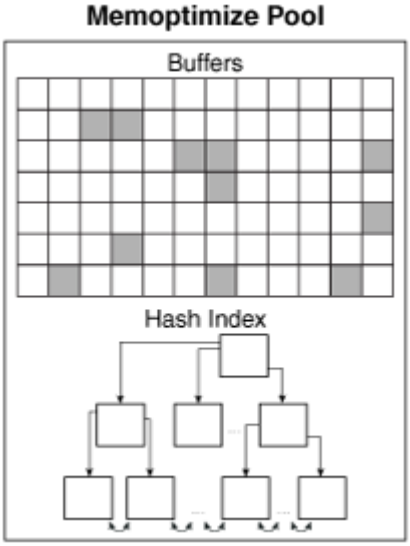
- Memoptimizeバッファ領域

  ディスクI/Oを回避するために、データベースはmemoptimizeプールでMEMOPTIMIZE FOR READ表のバッファを、表にNO MEMOPTIMIZE FOR READのマークが付くまで永続的にロックします。memoptimizeバッファには、データベース・バッファ・キャッシュ内のバッファと同じ構造が使用されます。ただし、memoptimizeプールのバッファは、データベース・バッファ・キャッシュから完全に分離され、そのサイズにカウントされません。memoptimizeバッファ領域はmemoptimizeプールの75%を占めます。

- ハッシュ索引

  ハッシュ索引は、非永続なセグメント・データ構造です。データベースは、複数の非連続メモリー単位としてハッシュ索引を割り当てます。各単位には、複数のハッシュ・バケットが含まれます。個別のマップ構造により、メモリー単位と主キーが関連付けられます。ハッシュ索引はmemoptimizeプールの25%を占めます。

図14-13 memoptimizeプール



memoptimizeプールを有効にするには、MEMOPTIMIZE_POOL_SIZE初期化パラメータを整数値に設定します(プールはデフォルトで無効になっています)。この値は、プールに割り当てるSGAの量を指定します。MEMOPTIMIZE_POOL_SIZE値はSGA_TARGETの対象ですが、データベースはmemoptimizeプールを自動では拡大および縮小しません。たとえば、SGA_TARGETが10 GBで、 MEMOPTIMIZE_POOL_SIZEが1 GBの場合、使用可能なSGAメモリーはmemoptimizeプール以外の合計9 GBです。

memoptimizeプールのサイズを変更するには、MEMOPTIMIZE_POOL_SIZEを手動で設定し、データベース・インスタンスを再起動する必要があります。ALTER SYSTEMを使用して動的にプール・サイズを変更することはできません。

DBMS_MEMOPTIMIZEパッケージを使用すると、memoptimizeプールに表を明示的に追加できます。

関連項目:

- 自動メモリー管理
- memoptimizeプールを有効にして問合せのパフォーマンスを向上させる方法を学習するには、Oracle Database パフォーマンス・チューニング・ガイドを参照してください
- DBMS_MEMOPTIMIZEパッケージについてさらに学習するには、Oracle Database PL/SQLパッケージおよびタイプ・リファレンスを参照してください
- CREATE TABLE ... MEMOPTIMIZE FOR READについてさらに学習するには、Oracle Database SQL言語リファレンスを参照してください。
- MEMOPTIMIZE_POOL_SIZEについてさらに学習するには、Oracle Databaseリファレンスを参照してください。

ソフトウェア・コード領域の概要

ソフトウェア・コード領域は、実行中または実行できるコードを保管するメモリーの部分です。Oracle Databaseのコードはソフトウェア領域に格納され、通常、この場所はユーザー・プログラムの格納場所よりも排他的であり、保護されています。

ソフトウェア領域のサイズは通常固定されており、ソフトウェアの更新時か再インストール時にかぎり変化します。これらの領域に必要なサイズは、オペレーティング・システムによって異なります。

ソフトウェア領域は読取り専用で、共有または非共有でインストールされます。Oracle FormsやSQL*Plusなど一部のデータベース・ツールおよびユーティリティは共有でインストールできますが、共有でインストールできないものもあります。データベースのコードは、メモリー内に複数のコピーを格納することなくすべてのユーザーがアクセスできるよう、可能なかぎり共有され、これにより、メイン・メモリーの使用量が削減されて、全体的にパフォーマンスが向上します。データベースの複数のインスタンスが同じコンピュータ上で実行されている場合、それらのインスタンスは異なるデータベースにおいても同じデータベース・コード領域を使用できます。

> ノート:
>
> ソフトウェアを共有でインストールするオプションは、すべてのオペレーティング・システムで使用できるわけではありません(たとえば、Microsoft Windows が稼働している PC では使用できません)。詳細は、オペレーティング・システム別のドキュメントを参照してください。

15 プロセス・アーキテクチャ

この章では、Oracle Databaseのプロセスについて説明します。

この章の構成は、次のとおりです。

- プロセスの概要
- クライアント・プロセスの概要
- サーバー・プロセスの概要
- バックグラウンド・プロセスの概要

プロセスの概要

プロセスは一連のステップを実行できるオペレーティング・システムのメカニズムです。

プロセス実行のアーキテクチャは、オペレーティング・システムによって決まります。たとえば、Windowsでは、Oracleバックグラウンド・プロセスはプロセス内の実行スレッドです。LinuxおよびUNIXにおけるOracleプロセスは、オペレーティング・システム・プロセスか、またはオペレーティング・システム・プロセス内部のスレッドです。

プロセスは、コード・モジュールで実行されます。Oracle Databaseに接続されたすべてのユーザーは、データベース・インスタンスにアクセスするために、次のモジュールを実行する必要があります。

- アプリケーションまたはOracle Databaseユーティリティ

  データベース・ユーザーは、プリコンパイラ・プログラムなどのデータベース・アプリケーションや、データベースに対してSQL文を発行するSQL*Plusなどのデータベース・ツールを実行します。

- Oracleデータベース・コード

  各ユーザーには、各ユーザーのために実行されるOracleデータベース・コードがあり、これによってアプリケーションのSQL文を解釈および処理します。

プロセスは通常、そのプロセスのプライベート・メモリー領域で実行されます。ほとんどのプロセスは、関連付けられたトレース・ファイルに定期的に書き込むことができます。

関連項目:

- 「トレース・ファイル」
- 「データベース管理者向けのツール」
- 「データベース開発者向けのツール」

プロセスのタイプ

複数のプロセスが、データベース・インスタンスに含まれているか、データベース・インスタンスと対話します。

プロセスは、次のタイプに分類されます。

- クライアント・プロセスがアプリケーションまたはOracleツール・コードを実行します。
- Oracleプロセスは、Oracleデータベース・コードを実行する実行単位です。マルチスレッド・アーキテクチャでは、

Oracleプロセスは、オペレーティング・システム・プロセスにもオペレーティング・システム・プロセス内のスレッドにもなります。Oracleプロセスには、次のサブタイプがあります。

- バックグラウンド・プロセスは、データベース・インスタンスとともに起動し、インスタンス・リカバリの実行、プロセスのクリーン・アップ、ディスクへのREDOバッファの書込みなどのメンテナンス・タスクを実行します。
- サーバー・プロセスは、クライアントの要求に基づいて処理を実行します。

  たとえば、これらのプロセスではSQL問合せを解析して共有プールに入れ、問合せごとに問合せ計画を作成して実行し、データベース・バッファ・キャッシュまたはディスクからバッファを読み取ります。

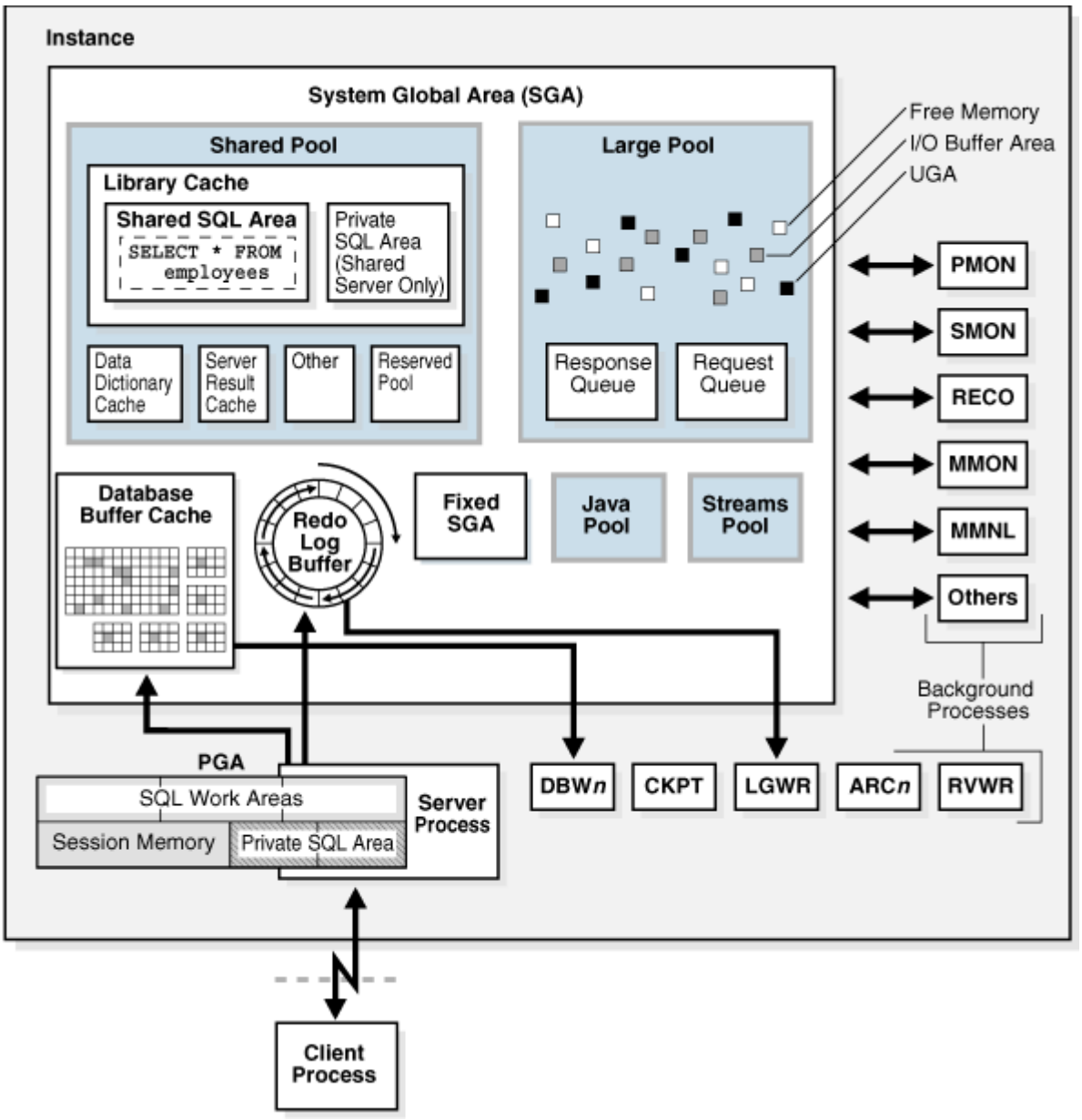
  > ノート:
  >
  > サーバー・プロセスおよびサーバー・プロセス内で割り当てられたプロセス・メモリーは、データベース・インスタンス内で実行されます。サーバー・プロセスが終了しても、インスタンスは機能し続けます。

- スレーブ・プロセスは、バックグラウンドまたはサーバー・プロセスに対して追加タスクを実行します。

プロセスの構造は、オペレーティング・システムおよびOracle Databaseオプションの選択に応じて異なります。たとえば、接続されたユーザーのコードを、専用サーバー接続または共有サーバー接続に対して構成できます。共有サーバー・アーキテクチャでは、データベース・コードを実行する各サーバー・プロセスにより複数のクライアント・プロセスに対応できます。

図15-1に、専用のサーバー接続を使用する、システム・グローバル領域(SGA)およびバックグラウンド・プロセスを示します。ユーザー接続ごとに、クライアント・プロセスによってアプリケーションが実行されます。このクライアント・プロセスは、データベース・コードを実行する専用サーバー・プロセスとは異なります。各クライアント・プロセスは、それ自体のサーバー・プロセスに関連付けられ、そのサーバー・プロセスには、それ自体のプログラム・グローバル領域(PGA)があります。

図15-1 OracleプロセスおよびSGA



関連項目:

- 「専用サーバー・アーキテクチャ」および「共有サーバー・アーキテクチャ」
- 構成の選択の詳細は、オペレーティング・システム固有のOracle Databaseマニュアルを参照してください。
- V$PROCESSビューについて学習するには、『Oracle Databaseリファレンス』を参照してください。

マルチプロセスおよびマルチスレッド化されたOracle Databaseシステム

マルチ・プロセスOracle Database(マルチユーザーOracle Databaseとも呼ばれる)は、Oracle Databaseコードの各部分の実行およびユーザー用に複数のプロセスを使用します(ユーザー用のプロセスは、接続するユーザーごとに1つのプロセスの場合や、複数のユーザーが共有する1つ以上のプロセスの場合があります)。

データベースの主な利点が、複数のユーザーが同時に必要とするデータを管理できることにあるため、多くのデータベースはマルチユーザー対応です。データベース・インスタンスの各プロセスは、特定のジョブを実行します。データベースとアプリケーションの処理を複数のプロセスに分割することにより、複数のユーザーおよびアプリケーションが1つのインスタンスに同時に接続でき、優れたパフォーマンスを確保できます。

Oracle Database 12c以前のリリースでは、OracleプロセスはUNIXおよびLinuxシステムでスレッドとして実行されませんで

した。Oracle Database 12cからは、マルチスレッドOracle Databaseモデルにより、Oracleプロセスが、別のアドレス領域でオペレーティング・システム・スレッドとして実行可能になりました。Oracle Database 12cがインストールされると、データベースはプロセス・モードで実行されます。データベースをスレッド・モードで実行するには、THREADED_EXECUTION初期化パラメータをTRUEに設定する必要があります。スレッド・モードでは、UNIXおよびLinux上の一部のバックグラウンド・プロセスは、プロセス(各プロセスが1つのスレッドを含む)として実行されるのに対し、残りのOracleプロセスは、プロセス内のスレッドとして実行されます。

スレッド・モードで実行されているデータベースでは、PMONおよびDBWがオペレーティング・システム・プロセスとして実行される可能性があるのに対して、LGWRおよびCMONは単一プロセス内のスレッドとして実行される可能性があります。2つのフォアグラウンド・プロセスおよびパラレル実行(PX)サーバー・プロセスは、2つ目のオペレーティング・システム・プロセス内のスレッドとして実行される可能性があります。3番目のオペレーティング・システム・プロセスには、複数のフォアグラウンド・スレッドが含まれている可能性があります。したがって、Oracleプロセスは、必ずしもオペレーティング・システム・プロセスとはかぎりません。

ノート:

THREADED_EXECUTION 初期化パラメータが TRUE に設定された場合、オペレーティング・システム認証はサポートされません。

例15-1 Oracleプロセス・メタデータの表示

V$PROCESSビューには、データベース・インスタンスに接続されているOracleプロセスごとに1行が含まれます。たとえば、次のSQL*Plusでの問合せを実行して、オペレーティング・システム・プロセスIDおよびオペレーティング・システム・スレッドIDをプロセスごとに取得できます。

```
COL SPID FORMAT a8
COL STID FORMAT a8
SELECT SPID, STID, PROGRAM FROM V$PROCESS ORDER BY SPID;
```

問合せにより、次の部分的サンプル出力が得られます。

```
SPID   STID   PROGRAM
-----  -----  ----------------------------
7190   7190   oracle@samplehost (PMON)
7192   7192   oracle@samplehost (PSP0)
7194   7194   oracle@samplehost (VKTM)
7198   7198   oracle@samplehost (SCMN)
7198   7200   oracle@samplehost (GEN0)
7202   7202   oracle@samplehost (SCMN)
7202   7204   oracle@samplehost (DIAG)
7198   7205   oracle@samplehost (DBRM)
7202   7206   oracle@samplehost (DIA0)
.
.
.
```

関連項目:

- V$PROCESSビューの使用方法を学習するには、『Oracle Databaseパフォーマンス・チューニング・ガイド』を参照してください
- THREADED_EXECUTION初期化パラメータについて学習するには、『Oracle Databaseリファレンス』を参照してください

クライアント・プロセスの概要

ユーザーがPro*CプログラムやSQL*Plusなどのアプリケーションを実行する場合、オペレーティング・システムでは、そのユーザー・アプリケーションを実行するために、クライアント・プロセス(ユーザー・プロセスとも呼ばれる)を作成します。クライアント・アプリケーションにはOracle Databaseライブラリがリンクされており、このライブラリによってデータベースとの通信に必要なAPIが提供されます。

クライアントおよびサーバー・プロセス

クライアント・プロセスは、インスタンスと直接対話するOracleプロセスとは大きく異なります。

クライアント・プロセスにサービスを提供するOracleプロセスでは、SGAからの読取りおよびSGAへの書込みが可能ですが、クライアント・プロセスではできません。クライアント・プロセスは、データベース・ホストとは別のホストで実行できますが、Oracleプロセスは実行できません。

たとえば、クライアント・ホストのユーザーがSQL*Plusを起動し、データベース・インスタンスが開始されていないときに、ネットワークを介して別のホストのデータベース、sampleに接続するとします。

```
SQL> CONNECT SYS@inst1 AS SYSDBA
Enter password: *********
Connected to an idle instance.
```

クライアント・ホスト上でsqlplusまたはsampleのプロセスを検索すると、sqlplusクライアント・プロセスのみが表示されます。

```
% ps -ef | grep -e sample -e sqlplus | grep -v grep
clientuser 29437 29436  0 15:40 pts/1    00:00:00 sqlplus           as sysdba
```

データベース・ホスト上でsqlplusまたはsampleのプロセスを検索すると、ローカル以外の接続を伴うサーバー・プロセスが表示されますが、クライアント・プロセスは表示されません。

```
% ps -ef | grep -e sample -e sqlplus | grep -v grep
serveruser 29441     1  0 15:40 ?        00:00:00 oraclesample (LOCAL=NO)
```

関連項目:

インスタンスが開始されていないときにクライアントがデータベースに接続する方法を学習するには、「インスタンスの開始方法」を参照してください

接続とセッション

データベース**接続**とは、クライアント・プロセスとデータベース・インスタンスとの間の物理的な通信経路のことです。

接続中、通信経路は、使用可能なプロセス間通信メカニズムまたはネットワーク・ソフトウェアを使用して確立されます。一般的に、接続はクライアント・プロセスとサーバー・プロセスまたはディスパッチャとの間で確立されますが、クライアント・プロセスとOracle Connection Manager(CMAN)との間でも確立されます。

データベース・セッションは、データベースに対する現在のユーザー・ログインの状況を示す、データベース・インスタンス・メモリー内の論理エンティティです。たとえば、ユーザーがパスワードによってデータベースから認証されると、このユーザーに対してセッションが確立されます。セッションは、ユーザーがデータベースから認証された時点から、ユーザーが接続を切断するか、データベース・アプリケーションを終了する時点まで続きます。

1つの接続では、セッションが確立されないか、1つ以上のセッションが確立されます。セッションはそれぞれ独立していて、あるセッションでのコミットは、他のセッションのトランザクションには影響しません。

ノート:

Oracle Net 接続プーリングが構成されると、接続が中断されることがありますが、セッションはそのまま残されます。

単独のデータベース・ユーザーに対して、複数のセッションが同時に存在できます。次の図に示すように、ユーザーhrが1つのデータベースに複数の接続を確立できます。専用サーバー接続の場合、それぞれの接続のかわりに1つのサーバー・プロセスがデータベースによって作成されます。これは、専用サーバーを作成するクライアント・プロセスのみが使用します。共有サーバー接続では、多数のクライアント・プロセスが1つの共有サーバー・プロセスにアクセスします。

図15-2 各接続に1セッション

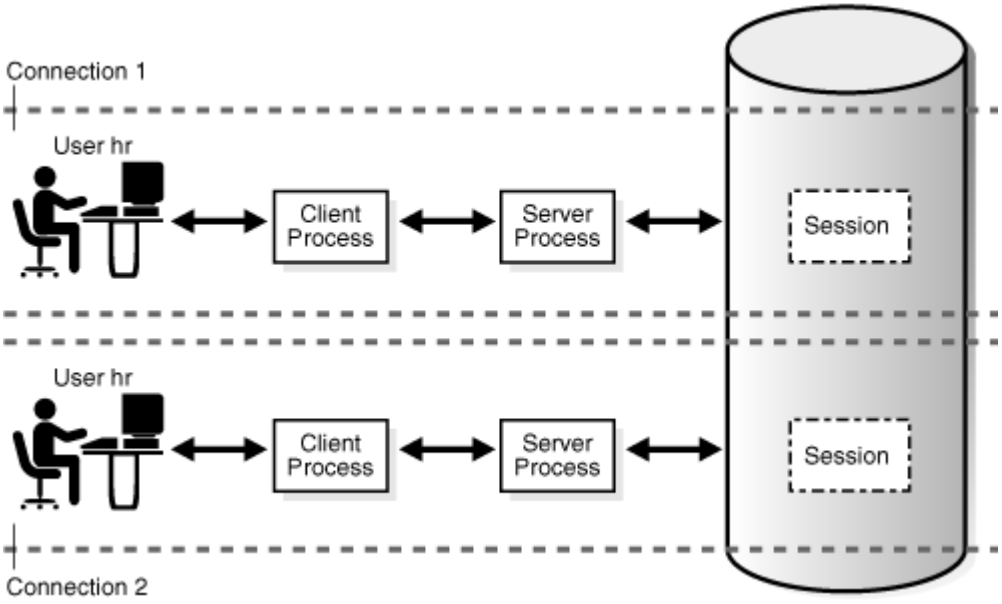
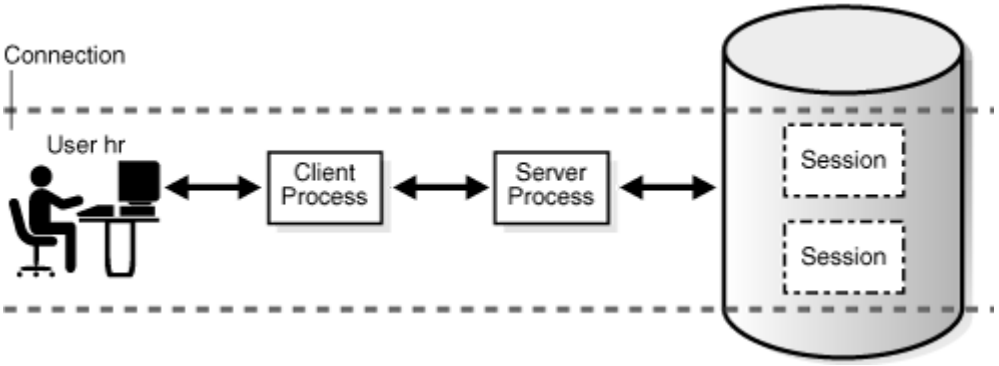


図15-3は、ユーザーhrが1つのデータベースに1つの接続を確立していることを示していますが、この接続には2つのセッションがあります。

図15-3 1つの接続に存在する2つのセッション



SQL文実行統計の自動トレース・レポートを作成すると、図15-3のシナリオが再作成されます。

例15-2のDISCONNECTコマンドは、実際には、接続ではなくセッションを終了しています。

例15-2 接続とセッション

次の例では、ユーザーSYSTEMとしてデータベースにSQL*Plusを接続してトレースを実行可能にし、新規セッションを作成してい

ます(サンプル出力を含みます)。

```
SQL> SELECT SID, SERIAL#, PADDR FROM V$SESSION WHERE USERNAME = USER;

SID SERIAL# PADDR
--- ------- --------
 90      91 3BE2E41C

SQL> SET AUTOTRACE ON STATISTICS;
SQL> SELECT SID, SERIAL#, PADDR FROM V$SESSION WHERE USERNAME = USER;

SID SERIAL# PADDR
--- ------- --------
 88      93 3BE2E41C
 90      91 3BE2E41C
...
SQL> DISCONNECT
```

DISCONNECTコマンドは、実際には、接続ではなくセッションを終了しています。新規端末を開いて別のユーザーとしてインスタンスに接続し、次の問合せを実行すると、アドレス3BE2E41Cの接続がアクティブなままであることが示されます。

```
SQL> CONNECT dba1@inst1
Password: ********
Connected.
SQL> SELECT PROGRAM FROM V$PROCESS WHERE ADDR = HEXTORAW('3BE2E41C');

PROGRAM
------------------------------------------------
oracle@stbcs09-1 (TNS V1-V3)
```

関連項目:

「共有サーバー・アーキテクチャ」

データベース操作

データベース監視のコンテキストにおける**データベースの操作**は、エンド・ユーザーまたはアプリケーション・コードによって定義されたとおりの2つの時点間のセッション・アクティビティです。

単一データベースの操作は、1つのSQL文または1つのPL/SQLプロシージャかファンクションのいずれかです。**コンポジット・データベースの操作**は、一連の単一またはコンポジット操作です。

タスクを監視、比較およびチューニングするため、大きなタスクのセットは複数のデータベース操作に分割し、複数の操作は複数のフェーズにさらに分割できます。ユースケースとしては、実行速度が通常より遅いPL/SQLバッチ・ジョブがあります。ジョブをデータベース操作として構成することで、ジョブ内のコストのかかるステップを特定してチューニングできます。

データベース操作の各実行は、操作名と実行IDという属性のペアにより一意に特定されます。あるセッションから、セッションIDとシリアル番号を指定することで、別のセッションのデータベース操作を開始または終了できます。

同じデータベース操作が2つ出現した場合、同じ名前で異なる実行IDを使用すれば、同時に実行できます。同じ名前のデータベース操作の各実行には、異なる文を含めることができます。

データベース操作は、DBMS_SQL_MONITOR PL/SQLパッケージにより作成し管理します。V$SQL_MONITOR、V$SQL_PLAN_MONITORおよびV$SQL_MONITOR_SESSTATを使用して操作を監視できます。

関連項目:

- データベース操作の監視方法を学習するには、Oracle Database SQLチューニング・ガイドを参照してください。
- 長時間のロードの監視方法を学習するには、『Oracle Databaseデータ・ウェアハウス・ガイド』を参照
- DBMS_SQL_MONITORについてさらに学習するには、Oracle Database PL/SQLパッケージおよびタイプ・リファレンスを参照してください
- V$SQL_MONITORについてさらに学習するには、Oracle Databaseリファレンスを参照してください

サーバー・プロセスの概要

Oracle Databaseでは、インスタンスに接続されたクライアント・プロセスの要求を処理するために、サーバー・プロセスが作成されます。クライアント・プロセスは常に別のサーバー・プロセスを介してデータベースと通信します。

データベース・アプリケーション用に作成されるサーバー・プロセスは、次のタスクを1つ以上実行できます。

- アプリケーションを介して発行されたSQL文を解析し、実行します。これには問合せ計画の作成および実行も含まれます。
- PL/SQLコードを実行します。
- データファイルからデータベース・バッファ・キャッシュにデータ・ブロックを読み取ります(DBWバックグラウンド・プロセスには、修正されたブロックをディスクに戻して書き込むというタスクがあります)。
- アプリケーションが情報を処理できるような方法で結果を戻します。

関連項目:

SQL処理の段階

専用サーバー・プロセス

専用サーバー接続の場合、クライアント接続は、1つのサーバー・プロセスにのみ関連付けられています。

Linuxでは、1つのデータベース・インスタンスに接続された20のクライアント・プロセスに、20のサーバー・プロセスが対応します。各クライアント・プロセスは、そのサーバー・プロセスと直接通信します。セッションの継続中、このサーバー・プロセスはそのクライアント・プロセス専用になります。サーバー・プロセスでは、プロセス固有の情報およびUGAをPGAに格納します。

関連項目:

- 「専用サーバー・アーキテクチャ」
- 「専用サーバー・モードおよび共有サーバー・モードにおけるPGAの使用」

共有サーバー・プロセス

共有サーバー接続の場合、クライアント・アプリケーションはネットワーク上でサーバー・プロセスではなく、**ディスパッチャ・プロセス**に接続します。たとえば、20のクライアント・プロセスが1つのディスパッチャ・プロセスに接続できます。

ディスパッチャ・プロセスでは、接続されたクライアントからのリクエストを受け取り、ラージ・プールのリクエスト・キューに入れます。最初に使用可能な共有サーバー・プロセスが、キューからリクエストを受け取って処理します。その後、共有サーバーは、その結果をディスパッチャ・レスポンス・キューに入れます。ディスパッチャ・プロセスはこのキューを監視し、結果をクライアントに送信します。

専用サーバー・プロセスと同様、共用サーバー・プロセスにはそれ自体のPGAがあります。ただし、共有サーバーがセッション・データにアクセスできるように、セッションのUGAはSGAにあります。

関連項目:

- 共有サーバー・アーキテクチャ
- ラージ・プール

Oracle Databaseによるサーバー・プロセスの作成方法

データベースでは接続メソッドに応じて、様々な方法でサーバー・プロセスが作成されます。

接続方法は次のとおりです。

- Bequeath

  SQL*Plus、OCIクライアントまたは別のクライアント・アプリケーションがサーバー・プロセスを直接生成します。

- Oracle Net Listener

  クライアント・アプリケーションがリスナー経由でデータベースに接続します。

- 専用ブローカ

  これは、フォアグラウンド・プロセスを作成するデータベース・プロセスです。リスナーと異なり、ブローカはデータベース・インスタンスの内部に常駐します。専用ブローカを使用するときは、クライアントがリスナーに接続した後、リスナーがその接続を専用ブローカに渡します。

接続にBequeathが使用されないときは、データベースによって次のようにサーバー・プロセスが作成されます。

1. クライアント・アプリケーションがリスナーまたはブローカから新規の接続をリクエストします。
2. リスナーまたはブローカが新規のプロセスまたはスレッドの作成を開始します。
3. オペレーティング・システムが新規のプロセスまたはスレッドを作成します。
4. Oracle Databaseが各種のコンポーネントおよび通知を初期化します。
5. データベースが接続および接続固有コードを引き渡します。

オプションで、専用ブローカ接続メソッドを使用する場合は、DBMS_PROCESSパッケージを使用してサーバー・プロセスのプールを事前に作成できます。この場合、Process Manager (PMAN)バックグラウンド・プロセスは、クライアント・リクエストとの関連付けを待機する、事前作成されたプロセスのプールを監視します。接続でサーバー・プロセスが求められる場合、データベースではプロセス作成のステップ2-4がスキップされ、ステップ5のみが実行されます。この最適化により、パフォーマンスが向上します。

ノート:

- DBMS_PROCESS PL/SQL パッケージについて学習するには、『Oracle Database PL/SQL パッケージおよびタイプ・リファレンス』を参照
- PMAN バックグラウンド・プロセスについて学習するには、『Oracle Database リファレンス』を参照

バックグラウンド・プロセスの概要

バックグラウンド・プロセスはマルチプロセスのOracleデータベースによって使用される追加のプロセスです。バックグラウンド・プロセスでは、データベースの操作、および複数ユーザーのパフォーマンスの最大化に必要なメンテナンス・タスクを実行します。

各バックグラウンド・プロセスには個別のタスクがありますが、他のプロセスと連携して動作します。たとえば、LGWRプロセスでは、REDOログ・バッファのデータをオンラインREDOログに書き込みます。いっぱいになったREDOログ・ファイルをアーカイブする準備ができたら、LGWRから別のプロセスにそのREDOログ・ファイルをアーカイブするように信号が送られます。

Oracle Databaseは、データベース・インスタンスが起動すると、自動的にバックグラウンド・プロセスを作成します。インスタンスは多数のバックグラウンド・プロセスを持つことができますが、そのすべてが常にそれぞれのデータベース構成に存在するとはかぎりません。次の問合せを実行すると、データベースで実行中のバックグラウンド・プロセスが表示されます。

```
SELECT PNAME
FROM   V$PROCESS
WHERE  PNAME IS NOT NULL
ORDER BY PNAME;
```

この項には次のトピックが含まれます:

- 必須のバックグラウンド・プロセス
- オプションのバックグラウンド・プロセス
- スレーブ・プロセス

関連項目:

すべてのバックグラウンド・プロセスの詳細は、『Oracle Databaseリファレンス』を参照

必須のバックグラウンド・プロセス

必須のバックグラウンド・プロセスは、すべての一般的なデータベース構成に存在します。

これらのプロセスは、デフォルトでは、最小限に構成された初期化パラメータ・ファイルを使用して起動される読取り/書込みデータベース・インスタンス内で実行されます。読取り専用データベース・インスタンスはこれらの一部のプロセスを無効化します。

この項では、次の必須のバックグラウンド・プロセスについて説明します。

- プロセス・モニター・プロセス(PMON)グループ
- プロセス・マネージャ(PMAN)
- リスナー登録プロセス(LREG)

- システム・モニター・プロセス(SMON)
- データベース・ライター・プロセス(DBW)
- ログ・ライター・プロセス(LGWR)
- チェックポイント・プロセス(CKPT)
- 管理性モニター・プロセス(MMONおよびMMNL)
- リカバラ・プロセス(RECO)

関連項目:

- 「読取り/書込みインスタンスと読取り専用インスタンス」
- MMAN、DIAG、VKTM、DBRM、PSP0などの他の必須プロセスの詳細は、『Oracle Databaseリファレンス』を参照してください
- Oracle RACおよびOracle Clusterwareに固有のバックグラウンド・プロセスの詳細は、『Oracle Real Application Clusters管理およびデプロイメント・ガイド』および『Oracle Clusterware管理およびデプロイメント・ガイド』を参照してください

プロセス・モニター・プロセス(PMON)グループ

PMONグループには、PMON、クリーンアップ・メイン・プロセス(CLMN)およびクリーンアップ・ヘルパー・プロセス(CL*nn*)が含まれています。これらのプロセスは、他のプロセスの監視とクリーンアップを担当します。

PMONグループは、バッファ・キャッシュのクリーンアップとクライアント・プロセスが使用していたリソースの解放を監視します。たとえば、PMONグループはアクティブ・トランザクション表の状態をリセットし、不要になったロックを解除し、アクティブ・プロセスのリストから終了したプロセスのプロセスIDを削除します。

データベースでは、終了したプロセスによって保持されていたリソースが解放され、他のプロセスで使用できるようになっていることを確認する必要があります。そうしないと、競合のためにプロセスがブロックされたり、動作不能になる可能性があります。

プロセス・モニター・プロセス(PMON)

プロセス・モニター(PMON)は、他のバックグラウンド・プロセスの終了を検出します。サーバー・プロセスまたはディスパッチャ・プロセスが異常終了すると、PMONグループがプロセスのリカバリを実行します。プロセス終了には、オペレーティング・システムのkillコマンドまたはALTER SYSTEM KILL SESSION文などの複数の原因が考えられます。

クリーンアップ・メイン・プロセス(CLMN)

PMONは、クリーンアップ作業をクリーンアップ・メイン・プロセス(CLMN)に委譲します。異常終了を検知するタスクはPMONに残されます。

CLMNでは終了したプロセス、終了したセッション、トランザクション、ネットワーク接続、アイドル・セッション、デタッチされたトランザクション、およびアイドル・タイムアウトを超過したデタッチされたネットワーク接続のクリーンアップが、定期的に実行されます。

クリーンアップ・ヘルパー・プロセス(CLnn)

CLMNは、クリーンアップ作業をCL*nn*ヘルパー・プロセスに委譲します。

CL*nn*プロセスは終了したプロセスおよびセッションのクリーンアップを補助します。ヘルパー・プロセスの数は、実行されるクリーンアップ作業の量と、現在のクリーンアップ効率に比例します。

クリーンアップ・プロセスはブロックされる場合があり、これによって他のプロセスのクリーンアップを続行できなくなります。また、複数のプロセスでクリーンアップが要求されると、クリーンアップ時間が膨大になる可能性があります。これらの理由から、Oracle Databaseでは複数のヘルパー・プロセスをパラレルに使用してクリーンアップを実行できるため、パフォーマンスの低下が緩和されます。

V$CLEANUP_PROCESSビューおよびV$DEAD_CLEANUPビューにCLMNクリーンアップに関するメタデータが含まれます。V$CLEANUP_PROCESSビューには、クリーンアップ・プロセスごとに1行が含まれます。たとえば、V$CLEANUP_PROCESS.STATEがBUSYの場合は、このプロセスが現在クリーンアップを行っています。

関連項目:

V$CLEANUP_PROCESSについてさらに学習するには、『Oracle Databaseリファレンス』を参照

データベース・リソースの隔離

プロセスまたはセッションが終了した場合、PMONグループは保持リソースをデータベースに解放します。場合によっては、破損してリカバリ不可能なリソースがPMONグループによって自動的に隔離されるため、データベース・インスタンスがただちに強制終了されることはありません。

PMONグループは、隔離されたリソースを保持していたプロセスまたはセッションに対して、できるかぎり多くのクリーンアップを実行し続けます。V$QUARANTINEビューには、リソースのタイプ、消費されたメモリー量、隔離の原因であるOracleエラーなどのメタデータが含まれます。

関連項目:

V$QUARANTINEについてさらに学習するには、『Oracle Databaseリファレンス』を参照

プロセス・マネージャ(PMAN)

プロセス・マネージャ(PMAN)は、共有サーバー、プーリングされたサーバーおよびジョブ・キュー・プロセスを含む、いくつかのバックグラウンド・プロセスを監視します。

PMANは、次のタイプのプロセスを監視、起動および停止します。

- ディスパッチャ・プロセスおよび共有サーバー・プロセス
- 接続ブローカおよびデータベース常駐接続プールのプール・サーバー・プロセス
- ジョブ・キュー・プロセス
- 再起動可能なバックグラウンド・プロセス

リスナー登録プロセス(LREG)

リスナー登録プロセス(LREG)では、データベース・インスタンスおよびディスパッチャ・プロセスに関する情報をOracle Netリスナーに登録します。

インスタンスの起動時に、LREGはリスナーに対してポーリングを実行し、リスナーが実行されているかどうかを確認します。リスナーが実行されている場合、LREGはリスナーに該当パラメータを渡します。実行されていない場合には、LREGは定期的にリスナーとの接続を試行します。

ノート:

Oracle Database 12c までのリリースでは、PMON がリスナー登録を実行していました。

関連項目:

「Oracle Net Listener」

システム・モニター・プロセス(SMON)

システム・モニター・プロセス(SMON)は、システム・レベルで様々なクリーン・アップ作業を実行します。

SMONには次のような役割があります。

- 必要に応じて、インスタンスの起動時に、インスタンス・リカバリを実行します。Oracle RACデータベースでは、障害の発生したインスタンスのインスタンス・リカバリを、データベース・インスタンスのSMONプロセスが実行できます。
- ファイル読取りまたは表領域オフライン・エラーが原因で、インスタンス・リカバリ時にスキップされた終了済トランザクションがリカバリされます。表領域またはファイルがオンラインに戻ると、SMONによってトランザクションがリカバリされます。
- 未使用の一時セグメントをクリーン・アップします。たとえば、Oracle Databaseでは、索引を作成するときにエクステントを割り当てます。この操作が失敗すると、SMONによって一時領域がクリーン・アップされます。
- ディクショナリ管理表領域内の、連続した空きエクステントを結合します。

SMONは、処理が必要かどうかを定期的にチェックします。他のプロセスは、必要性が検出された場合にSMONをコールできます。

データベース・ライター・プロセス(DBW)

データベース・ライター・プロセス(DBW)は、データベース・バッファの内容をデータファイルに書き込みます。DBWプロセスは、データベース・バッファ・キャッシュ内の変更されたバッファをディスクに書き込みます。

ほとんどのシステムでは1つのデータベース・ライター・プロセス(DBW0)があれば十分ですが、追加プロセス(DBW1からDBW9およびDBWaからDBWj)を構成して、システムでデータを頻繁に変更する場合の書込みのパフォーマンスを改善できます。これらの追加DBWプロセスは、ユニプロセッサ・システムでは効果がありません。

DBWプロセスは、次のような場合に使用済バッファをディスクに書き込みます。

- サーバー・プロセスがしきい値の数までバッファをスキャンしても、クリーンな再利用可能バッファが見つからない場合は、DBWに書込み信号が送られます。DBWは、可能であれば、他の処理中に使用済バッファをディスクに非同期に書き込みます。
- DBWは、定期的にバッファを書き込んで、REDOスレッド内のインスタンス・リカバリを開始する位置(チェックポイント)を進めます。チェックポイントのログ位置は、バッファ・キャッシュ内の最も古い使用済バッファ・キャッシュによって決まります。

多くの場合、DBWによって書き込まれるブロックは、ディスク内に分散されます。このため、この書込みは、LGWRが実行する順次書込みよりも遅くなる傾向があります。効率を向上させるために、可能であればDBWは、マルチブロック書込みを実行します。マルチブロック書込みで書き込まれるブロックの数は、オペレーティング・システムに応じて異なる。

関連項目:

- 「データベース・バッファ・キャッシュ」
- 「チェックポイントの概要」
- DBWの構成、監視およびチューニングの詳細は、Oracle Databaseパフォーマンス・チューニング・ガイドを参照してください

ログ・ライター・プロセス(LGWR)

ログ・ライター・プロセス(LGWR)では、オンラインREDOログ・バッファを管理します。

LGWRは、バッファの一部分をオンラインREDOログに書き込みます。データベース・バッファを変更するタスクを切り離すことによって、分散した使用済バッファのディスクへの書込み、およびREDOのディスクへの高速順次書込みの処理で、データベースのパフォーマンスが向上します。

次の状況では、LGWRは、最後の書込み以降にバッファにコピーされたREDOエントリすべてを、REDOログ・ファイルに書き込みます。

- ユーザーがトランザクションをコミットしたとき。
- オンラインREDOログ・スイッチが発生したとき
- LGWRによる最後の書込みから3秒経過したとき
- REDOログ・バッファが3分の1になったとき、またはバッファ・データが1MBになったとき
- DBWが修正済のバッファをディスクに書き込む必要があるとき

  DBWが使用済バッファを書き込むには、バッファの変更に関連するすべてのREDOレコードがディスクに書き込まれている必要があります(事前書込みプロトコル)。DBWは、書き込まれていないREDOレコードがあることを検出すると、そのレコードをディスクに書き込むようにLGWRに信号を送り、LGWRの処理が完了するまで待機して、データ・バッファをディスクに書き込みます。

関連項目:

「トランザクションのコミット」

LGWRおよびコミット

Oracle Databaseは、コミットされたトランザクションのパフォーマンスを向上させるために、高速コミット・メカニズムを使用します。

ユーザーがCOMMIT文を発行すると、トランザクションに**システム変更番号(SCN)**が割り当てられます。LGWRはREDOログ・バッファ内にコミット・レコードを入れ、コミットSCNおよびトランザクションのREDOエントリとともにそれを即時にディスクに書き込みます。

REDOログ・バッファは循環バッファです。LGWRがREDOログ・バッファからオンラインREDOログ・ファイルにREDOエントリを書き込んだ後、サーバー・プロセスはディスクに書き込まれたREDOログ・バッファのエントリ上に新しいエントリをコピーできます。オンラインREDOログへのアクセスが頻繁なときにも、新しいエントリを書き込めるよう常にバッファ内の領域を空けておくため、LGWRが行う書込みは通常は非常に高速になります。

トランザクションのコミット・レコードを含むREDOエントリのアトミックな書込みは、トランザクションがコミットされたかどうか確認する

1つのイベントです。Oracle Databaseでは、データ・バッファがまだディスクに書き込まれていない場合でも、コミットしたトランザクションに対する成功コードが戻されます。対応するデータ・ブロックの変更は、DBWがそれらの変更を効率的にデータファイルに書き込めるようになるまで延期されます。

ノート:

LGWR では、トランザクションがコミットする前に、REDO ログ・エントリをディスクに書き込むことができます。後でトランザクションがコミットされた場合にのみ、REDO エントリにより保護された変更が確定します。

アクティビティが高いとき、LGWRはグループ・コミットを使用できます。たとえば、ユーザーがコミットし、これによって、LGWRがトランザクションのREDOエントリをディスクに書き込むとします。この書込み中、他のユーザーがコミットします。LGWRでは、前の書込みが完了するまで、これらのトランザクションをディスクに書き込んでコミットすることはできません。完了すると、LGWRは、(まだコミットされていない)待機しているトランザクションに関するREDOエントリのリストを1回の操作で書き込むことができます。これにより、データベースでは、ディスクI/Oが最小化され、パフォーマンスが最大化されます。コミットの要求が高い頻度で続くと、LGWRによる毎回の書込みに複数のコミット・レコードが含まれることがあります。

LGWRおよびアクセス不能ファイル

LGWRは、ミラー化されたアクティブなオンラインREDOログ・ファイルのグループにも同時に書き込みます。

ログ・ファイルにアクセスできない場合、LGWRはグループ内の他のファイルに書込みを続け、LGWRトレース・ファイルおよびアラート・ログにエラーを書き込みます。グループ内のすべてのファイルが破損している場合や、アーカイブされていないためにグループ全体が使用できない場合、LGWRは機能を続行できません。

関連項目:

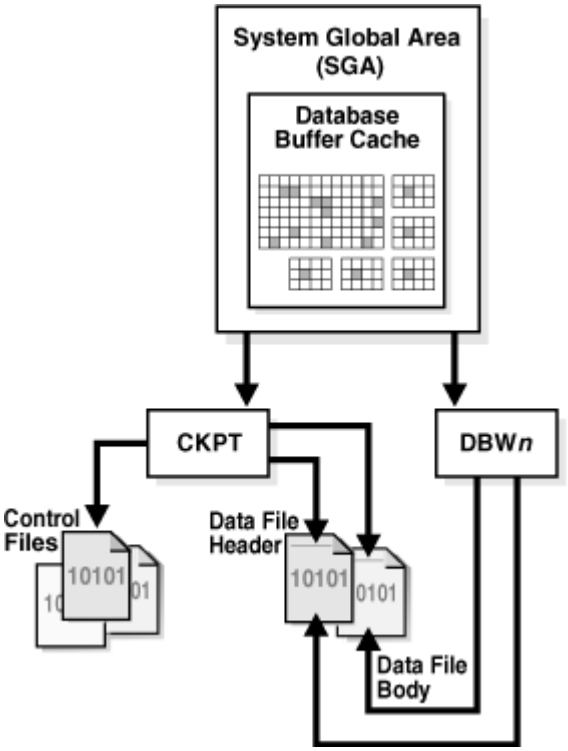
- 「Oracle DatabaseでのオンラインREDOログの書込み方法」および「REDOログ・バッファ」
- LGWRのパフォーマンスの監視およびチューニング方法の詳細は、『Oracle Databaseパフォーマンス・チューニング・ガイド』を参照してください

チェックポイント・プロセス(CKPT)

チェックポイント・プロセス(CKPT)は、チェックポイント情報により制御ファイルおよびデータファイル・ヘッダーを更新し、DBWに対してブロックをディスクに書き込むように信号を送ります。チェックポイント情報には、チェックポイント位置、SCNおよびオンラインREDOログの中でリカバリを開始する位置が含まれます。

図15-4に示すように、CKPTではデータファイルへのデータ・ブロックの書込み、およびオンラインREDOログ・ファイルへのREDOブロックの書込みは行いません。

図15-4 チェックポイント・プロセス



関連項目:

チェックポイントの概要

管理性モニター・プロセス(MMONおよびMMNL)

管理性モニター・プロセス(MMON)は、**自動ワークロード・リポジトリ(AWR)**に関連する多数のタスクを実行します。

たとえば、MMONはスナップショットを取得し、前回変更されたSQLオブジェクトの統計値を取得して、**メトリック**がそのしきい値を超えたときに書込みを行います。

管理性モニター・ライト・プロセス(MMNL)は、SGAのアクティブ・セッション履歴(ASH)バッファ内の統計をディスクに書き込みます。MMNLは、ASHバッファがいっぱいになるとディスクに書き込みます。

関連項目:

「自動ワークロード・リポジトリ(AWR)」および「アクティブ・セッション履歴(ASH)」

リカバラ・プロセス(RECO)

分散データベースでは、**リカバラ・プロセス(RECO)**によって、分散トランザクションに関連する障害が自動的に解決されます。

ノードのRECOプロセスは、インダウト分散トランザクションにかかわる他のデータベースに自動的に接続されます。RECOでは、データベース間の接続を再確立するとき、すべてのインダウト・トランザクションを自動的に解決し、解決されるトランザクションに対応する行を各データベースの保留中のトランザクション表からすべて削除します。

関連項目:

分散トランザクション・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

オプションのバックグラウンド・プロセス

オプションのバックグラウンド・プロセスは、必須として定義されていないバックグラウンド・プロセスです。

ほとんどのオプションのバックグラウンド・プロセスは、タスクまたは機能に固有です。たとえば、Oracle ASMをサポートするバックグラウンド・プロセスは、この機能が有効になっている場合のみ使用できます。

この項では、いくつかの一般的なオプションのプロセスについて説明します。

- アーカイバ・プロセス(ARCn)
- ジョブ・キュー・プロセス(CJQ0およびJnnn)
- フラッシュバック・データ・アーカイブ・プロセス(FBDA)
- 領域管理コーディネータ・プロセス(SMCO)

関連項目:

- 「Oracle Database Advanced Queuing (AQ)」
- AQおよびOracle ASMに固有のバックグラウンド・プロセスの詳細は、『Oracle Databaseリファレンス』を参照してください。

アーカイバ・プロセス(ARC*n*)

アーカイバ・プロセス(ARCn)は、REDOログ・スイッチが行われると、オンラインREDOログ・ファイルをオフライン・ストレージにコピーします。

また、トランザクションのREDOデータを収集し、そのデータをスタンバイ・データベースの宛先に転送できます。ARC*n*プロセスは、データベースがARCHIVELOGモードに設定され、自動アーカイブが使用可能になっている場合にのみ存在します。

関連項目:

- 「アーカイブREDOログ・ファイル」
- アーカイバ・プロセス数の調整方法を学習するには、『Oracle Database管理者ガイド』を参照してください
- アーカイバ・パフォーマンスのチューニング方法を学習するには、『Oracle Databaseパフォーマンス・チューニング・ガイド』を参照してください

ジョブ・キュー・プロセス(CJQ0およびJnnn)

キュー・プロセスでは、多くの場合バッチ・モードでユーザー・ジョブを実行します。ジョブとは、1回以上実行するようにスケジューリングされたユーザー定義のタスクです。

たとえば、ジョブ・キューを使用して、バックグラウンドで長時間実行される更新をスケジュールできます。開始日と時間間隔を指定すると、ジョブ・キュー・プロセスは、指定された間隔でジョブの実行を試行します。

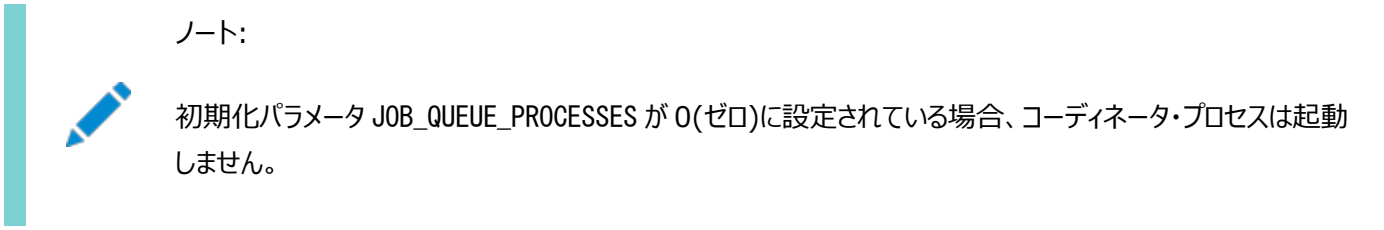
Oracle Databaseではジョブ・キュー・プロセスが動的に管理されるため、ジョブ・キュー・クライアントは、必要に応じてより多くの

ジョブ・キュー・プロセスを使用できます。新規プロセスが使用するリソースは、そのプロセスがアイドル状態になると解放されます。

動的ジョブ・キュー・プロセスは、指定された間隔で多数のジョブを同時に実行できます。各イベントの順序は次のとおりです。

1. ジョブ・コーディネータ・プロセス(CJQ0)は、Oracle Schedulerの必要に応じて、自動的に起動し、停止します。コーディネータ・プロセスは、システムのJOB$表から、実行する必要があるジョブを定期的に選択します。選択された新規ジョブは、時間順に並べ替えられます。
2. コーディネータ・プロセスは、ジョブを実行するためにジョブ・キュー・スレーブ・プロセス(Jnnn)を動的に生成します。
3. ジョブ・キュー・プロセスは、実行のためにCJQ0プロセスによって選択されたジョブの1つを実行します。ジョブ・キュー・プロセスは、一度に1つずつジョブを実行して完了します。
4. ジョブ・キュー・プロセスは、1つのジョブの実行が終了すると、次のジョブの問合せを行います。実行予定のジョブが存在しない場合、ジョブ・キュー・プロセスはスリープ状態に入り、定期的な周期で起動すると次のジョブの問合せを行います。新しいジョブがない場合、事前に設定した周期が経過した後に終了します。

初期化パラメータJOB_QUEUE_PROCESSESは、1つのインスタンスで同時に実行できるジョブ・キュー・プロセスの最大数を示します。ただし、クライアントは、すべてのジョブ・キュー・プロセスがジョブの実行に使用可能であると想定することはできません。



ノート:

初期化パラメータ JOB_QUEUE_PROCESSES が 0(ゼロ)に設定されている場合、コーディネータ・プロセスは起動しません。

関連項目:

- 「Oracle Scheduler」
- Oracle Schedulerジョブについて学習するには、『Oracle Database管理者ガイド』を参照してください。
- AQバックグラウンド・プロセスについて学習するには、『Oracle Databaseアドバンスト・キューイング・ユーザーズ・ガイド』を参照してください

フラッシュバック・データ・アーカイブ・プロセス(FBDA)

フラッシュバック・データ・アーカイブ・プロセス(FBDA)は、追跡した表の履歴行をフラッシュバック・データ・アーカイブにアーカイブします。

追跡した表に対するDMLを含むトランザクションがコミットされると、このプロセスでは、変更された行の事前イメージがフラッシュバック・データ・アーカイブに格納されます。また、現在の行にメタデータを保存します。

FBDAは、フラッシュバック・データ・アーカイブの領域、編成および保存に関する管理を自動的に行います。また、追跡対象トランザクションのアーカイブが行われた期間が記録されます。

領域管理コーディネータ・プロセス(SMCO)

SMCOプロセスでは、様々な領域管理関連タスクの実行を調整します。

通常のタスクには、事前の領域割当ておよび領域の再利用が含まれます。SMCOにより、自動的にスレーブ・プロセス(W*nnn*)が起動され、タスクが実行されます。

関連項目:

フラッシュバック・データ・アーカイブおよび一時履歴機能について学習するには、『Oracle Database開発ガイド』を参照してください

スレーブ・プロセス

スレーブ・プロセスは、他のプロセスにかわって作業を実行するバックグラウンド・プロセスです。

この項では、Oracle Databaseが使用するスレーブ・プロセスの一部について説明します。

関連項目:

Oracle Databaseのスレーブ・プロセスの詳細は、『Oracle Databaseリファレンス』を参照してください。

I/Oスレーブ・プロセス

I/Oスレーブ・プロセス(Innn)は、非同期I/Oがサポートされていないシステムおよびデバイスにかわって、非同期I/Oをシミュレートします。

非同期I/Oの場合、伝送のためのタイミング要件がないため、伝送が終了する前に他のプロセスを起動できます。

たとえば、非同期I/Oがサポートされていないオペレーティング・システムのディスクに、アプリケーションが1000ブロックを書き込むとします。それぞれの書込みは順次実行され、書込みの完了確認まで待機します。非同期ディスクの場合、アプリケーションは一括でブロックを書き込み、すべてのブロックが書き込まれたというオペレーティング・システムからのレスポンスを待機している間、他の作業を実行できます。

非同期I/Oをシミュレートするには、1プロセスがいくつかのスレーブ・プロセスを監視します。実行者プロセスでは、それぞれの書込みの完了報告を待機し、完了時にそれを実行者に転送する各スレーブ・プロセスに作業を割り当てます。真の非同期I/Oでは、オペレーティング・システムがI/Oの完了と、その報告のプロセスへの転送を待機しますが、シミュレートされた非同期I/Oの場合、スレーブが完了報告を待機し、実行者へそれを転送します。

Oracle Databaseでは、次のような様々なタイプのI/Oスレーブがサポートされています。

- Recovery Manager(RMAN)のI/Oスレーブ

  データのバックアップまたはリストアにRMANを使用するとき、ディスクおよびテープ・デバイスにI/Oスレーブを使用できます。

- データベース・ライター・スレーブ

  複数のデータベース・ライター・プロセスを使用することが現実的でない場合、たとえば、コンピュータのCPUが1つのみである場合、Oracle Databaseは、複数のスレーブ・プロセスにI/Oを分散できます。DBWは、ディスクへのブロックの書込みを可能にするため、バッファ・キャッシュLRUリストをスキャンする唯一のプロセスです。ただし、I/Oスレーブは、これらのブロックに対するI/Oを実行します。

関連項目:

- I/Oスレーブのバックアップおよびリストア操作についてさらに学習するには、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

- データベース・ライター・スレーブについてさらに学習するには、『Oracle Databaseパフォーマンス・チューニング・ガイド』を参照してください

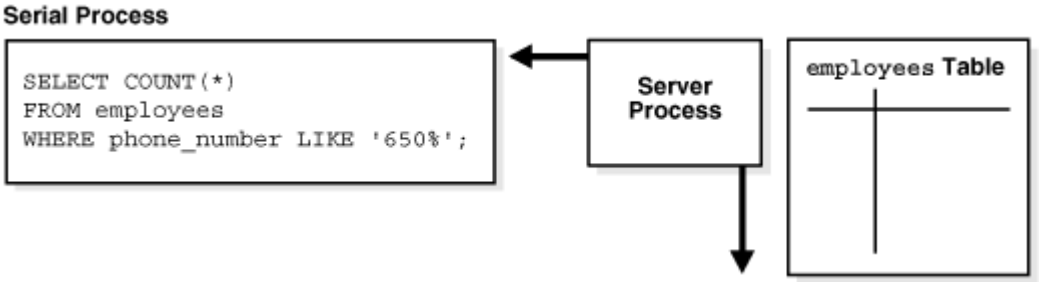
パラレル実行(PX)サーバー・プロセス

パラレル実行では、複数のプロセスが同時に機能して、単一のSQL文を実行します。

複数プロセス内で作業を分割すると、Oracle Databaseでは、より迅速に文を実行できます。たとえば、1つのプロセスで4つの四半期分をすべて処理するのではなく、4つのプロセスが1年のそれぞれの四半期を分担して処理するなどがこの例です。

パラレル実行に対して、シリアル実行の場合、SQL文の順次実行に必要な処理のすべてを1つのサーバー・プロセスが実行します。たとえば、SELECT * FROM employeesなどの全表スキャンを実行するには、図15-5に示すように、1つのサーバー・プロセスですべての処理が実行されます。

図15-5 シリアルの全表スキャン



パラレル実行では、データ・ウェアハウスなどの大規模データベースでの、データ処理集中型の操作におけるレスポンス時間が短縮されます。対称型マルチプロセッサ(SMP)およびクラスタ化されたシステムでは、文の処理を多数のCPUに分割できるため、パラレル実行によりパフォーマンスが大幅に改善されます。パラレル実行は、特定のタイプのOLTPおよびハイブリッド・システムにとっても利点があります。

Oracle RACシステムでは、特定のサービスのサービス配置によってパラレル実行が制御されます。つまり、パラレル・プロセスはサービスが構成されているノードで実行されます。デフォルトでは、Oracle Databaseは、データベースへの接続に使用されるサービスを提供するインスタンスでのみパラレル・プロセスを実行します。これは、パラレル・リカバリやGV$問合せの処理などの他のパラレル操作には影響しません。

関連項目:

- パラレル実行についてさらに学習するには、Oracle Database VLDBおよびパーティショニング・ガイドを参照してください
- Oracle RAC環境でのパラレル実行に関する考慮事項については、Oracle Real Application Clusters管理およびデプロイメント・ガイドを参照してください

問合せコーディネータ

パラレル実行において、サーバー・プロセスは**問合せコーディネータ**として動作します(パラレル実行コーディネータとも呼ばれます)。

問合せコーディネータの役割は次のとおりです。

1. 問合せの解析
2. パラレル実行サーバー・プロセスの割当てと制御

3. ユーザーへの出力の送信

問合せの問合せ計画では、まずコーディネータがSQL問合せ内の各演算子をパラレル・ピースに分割し、それらを問合せで指定した順序に従って実行してから、演算子を実行するパラレル実行サーバーによって生成された部分的な結果を統合します。

1つの操作に割り当てられたパラレル実行サーバーの数が、その操作の並列度となります。同じSQL文中の複数の操作の並列度は、すべて同じです。

プロデューサとコンシューマ

パラレル実行サーバーは、プロデューサとコンシューマに分かれます。プロデューサはデータを処理し、それを必要とするコンシューマに配布します。

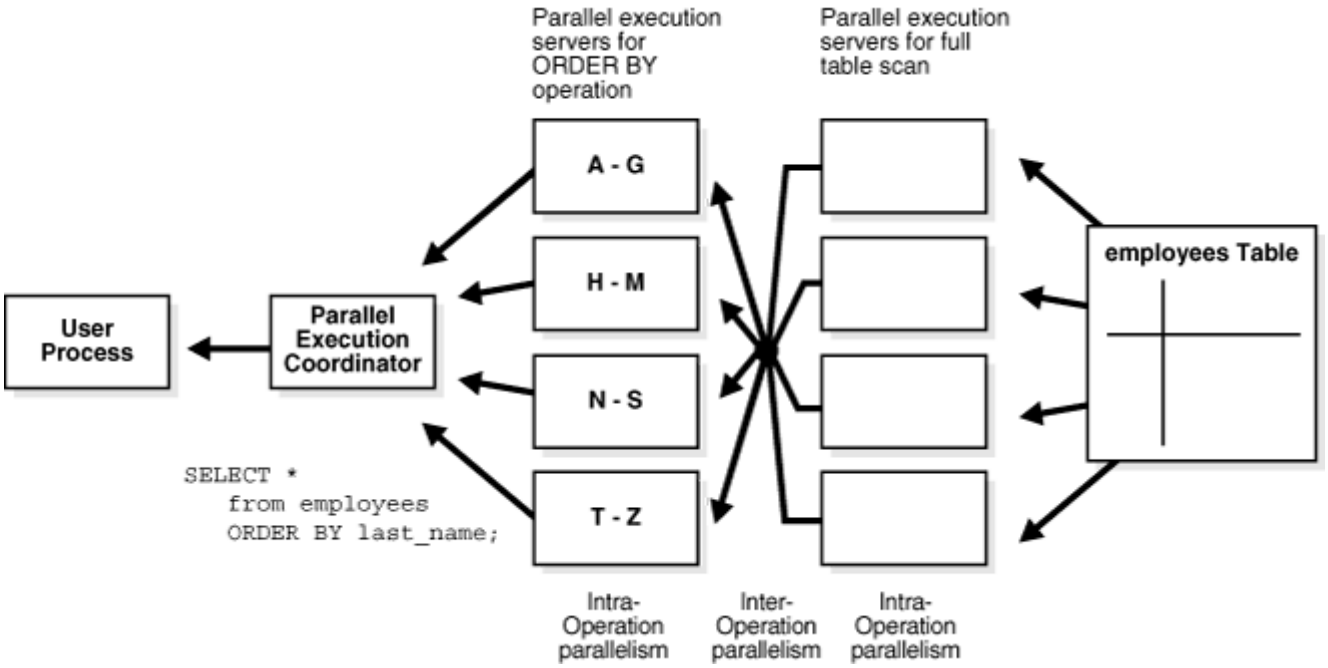
データベースは、様々なテクニックを使用して配布を実行できます。一般的な方法として、ブロードキャストとハッシュの2つがあります。ブロードキャストでは、各プロデューサがすべてのコンシューマに行を送信します。ハッシュでは、データベースが一連のキーに対するハッシュ関数を計算し、各コンシューマにハッシュ値のサブセットを担当させます。

図15-6は、次の文のパラレル実行でのプロデューサとコンシューマの相互作用を表しています。

```
SELECT * FROM employees ORDER BY last_name;
```

実行計画により、employees表の全体スキャンが実装されます。スキャンの後に、取得された行のソートが続きます。また、スキャン操作に関係するすべてのプロデューサ・プロセスが、ソートを実行する適切なコンシューマ・プロセスに行を送信します。

図15-6 プロデューサとコンシューマ



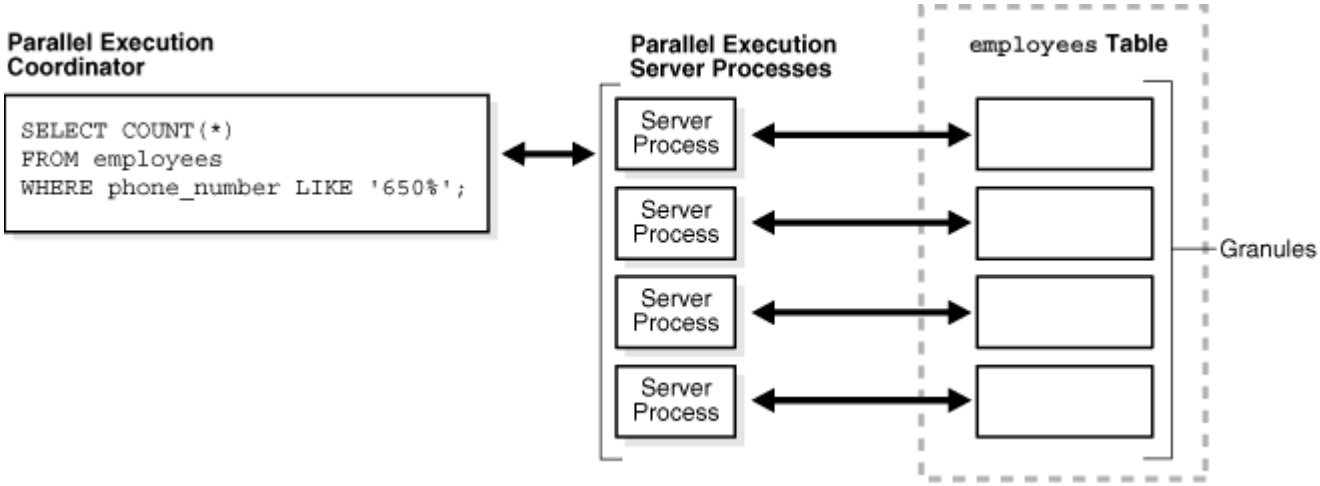
グラニュル

パラレル実行では、表は動的にロード単位に分割されます。各単位は**グラニュル**と呼ばれ、データにアクセスする際の最小の作業単位です。

ブロックに基づくグラニュルは、1つのパラレル実行サーバー(*PX*サーバー)によって読み取られる表のデータ・ブロックの範囲です(このサーバーは名前の形式としてPnnnを使用します)。パラレル・サーバー・プロセス間で作業を均等に分散するには、グラニュル数が常にリクエストDOPよりもかなり多くなります。

図15-7に、employees表のパラレル・スキャンを示します。

図15-7 パラレルの全表スキャン



データベースでは、実行時にグラニュルをパラレル実行サーバーにマップします。パラレル実行サーバーがグラニュルに対応する行の読取りを終了したとき、グラニュルがまだ残っている場合、問合せコーディネータから他のグラニュルを取得します。この操作は、表全体が読み取られるまで続行されます。実行サーバーは結果をコーディネータに送り返し、それぞれの結果が組み立てられて、最終的な全表スキャンになります。

関連項目:

- パラレル実行の使用方法を学習するには、Oracle Database VLDBおよびパーティショニング・ガイドを参照してください
- 推奨の並列性初期化パラメータについて学習するには、Oracle Databaseデータ・ウェアハウス・ガイドを参照してください

16 アプリケーションとOracle Net Servicesのアーキテクチャ

この章では、アプリケーション・アーキテクチャについて定義し、分散処理環境でOracleデータベースとデータベースのアプリケーションがどのように機能するかを説明します。このマニュアルの内容は、ほとんどすべてのタイプのOracle Database環境に適用されます。

この章の構成は、次のとおりです。

- Oracleアプリケーション・アーキテクチャの概要
- Oracle Net Servicesのアーキテクチャの概要
- プログラム・インタフェースの概要

Oracleアプリケーション・アーキテクチャの概要

この章では、**アプリケーション・アーキテクチャ**は、データベース・アプリケーションがOracleデータベースに接続するコンピューティング環境を示します。

この項では、次の項目について説明します。

- クライアント/サーバー・アーキテクチャの概要
- 複数層アーキテクチャの概要
- グリッド・アーキテクチャの概要

クライアント/サーバー・アーキテクチャの概要

Oracle Database環境では、データベース・アプリケーションとデータベースは**クライアント/サーバー・アーキテクチャ**に分割されます。

次のコンポーネントがあります。

- クライアントでは、データベース情報にアクセスし、ユーザーと対話するデータベース・アプリケーション(SQL*Plus、Visual Basicデータ入力プログラムなど)が実行されます。
- サーバーでは、Oracle Databaseソフトウェアが実行され、Oracleデータベースへの同時実行の共有データ・アクセスに必要な機能が処理されます。

クライアント・アプリケーションとデータベースを同じコンピュータで実行することもできますが、クライアント部分とサーバー部分を別々のコンピュータで実行し、それらのコンピュータをネットワークで接続する方がさらに効率的です。次の各項では、Oracle Databaseクライアント/サーバー・アーキテクチャの様々な形態について説明します。

分散処理

複数ホストを使用した個別タスクの処理は、**分散処理**と呼ばれます。

フロントエンド処理とバックエンド処理は異なるコンピュータ上で実行されます。図16-1では、クライアントとサーバーが異なるホストに配置され、Oracle Net Servicesを介して接続されています。

図16-1 クライアント/サーバー・アーキテクチャと分散処理

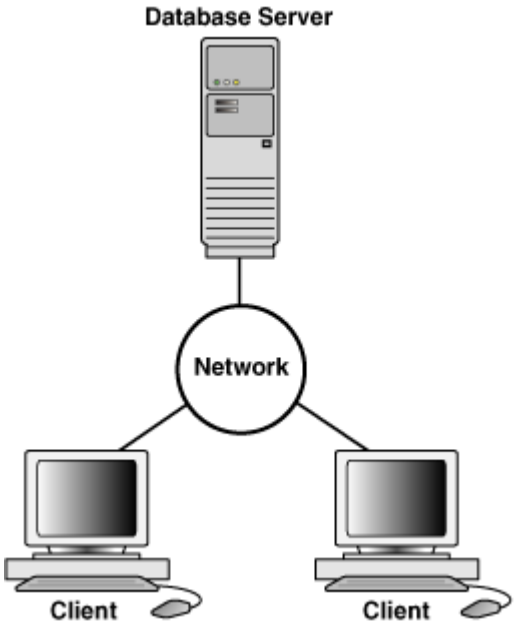
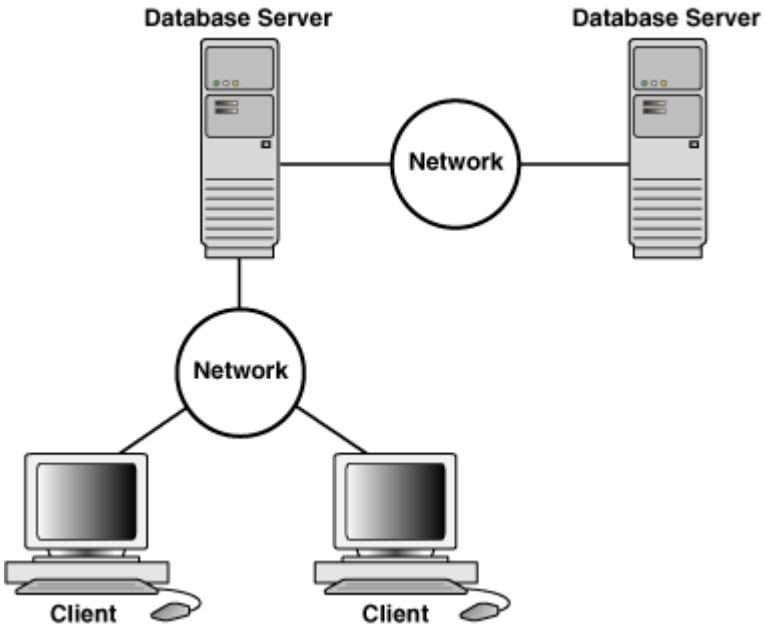


図16-2に、様々な分散データベースを示します。この例では、あるホスト上のデータベースが、異なるホスト上の別のデータベースにアクセスしています。

図16-2 クライアント/サーバー・アーキテクチャと分散データベース



ノート:

この章の以降の内容は、1 つのサーバー上に 1 つのデータベースがある環境に適用されます。

クライアント/サーバー・アーキテクチャの利点

分散処理環境でのOracle Databaseクライアント/サーバー・アーキテクチャには、多くの利点があります。

次のような利点があります。

- クライアント・アプリケーションではデータ処理は実行されません。そのかわり、クライアント・アプリケーションは、ユーザーに入力をリクエストし、必要なデータをサーバーにリクエストし、そのデータを分析してクライアント・ワークステーションまたは端末の表示機能(グラフィックスやスプレッドシート)を使用して表示します。

- クライアント・アプリケーションはデータの物理的な位置に依存しません。データを他のデータベース・サーバーに移動したり分散しても、アプリケーションはほとんど、またはまったく変更することなく、機能を続行できます。
- Oracle Databaseでは、基礎となるオペレーティング・システムのマルチタスキング機能と共有メモリー機能を活用できます。その結果、クライアント・アプリケーションに最高度の同時実行性、データ整合性およびパフォーマンスが提供されます。
- クライアント・ワークステーションや端末はデータの表示用に最適化でき(グラフィックスやマウスのサポート提供など)、サーバーはデータの処理と格納用に最適化できます(大容量のメモリーとディスク領域の搭載など)。
- ネットワーク化された環境では、安価なクライアント・ワークステーションを使用して、サーバーのリモート・データに効率的にアクセスできます。
- データベースは、システムの拡張とともにスケーリングできます。複数のサーバーを追加して、データベース処理の負荷をネットワーク全体に分散することや(水平スケール)、データベースをミニコンピュータやメインフレームなどに移動して、より大規模なシステムのパフォーマンス上の利点を活用(垂直スケール)することができます。どちらの場合も、Oracle Databaseにはシステム間での移植性があるため、データとアプリケーションをほとんど、またはまったく変更することなく維持できます。
- ネットワーク化された環境の場合、共有データは、すべてのコンピュータに格納されるのではなく、サーバーに格納されます。これにより、同時アクセスの管理が簡略化され、効率的になります。
- ネットワーク化された環境では、クライアント・アプリケーションからサーバーにデータベース・リクエストを送るときにSQL文を使用できます。サーバーが受け取ったSQL文はそのサーバーで処理され、その結果がクライアントに戻されます。ネットワークを介してやりとりされるのはリクエストと結果のみであるため、ネットワーク通信量は最小限ですみます。

関連項目:

分散データベースについてさらに学習するには、Oracle Database管理者ガイドを参照してください

複数層アーキテクチャの概要

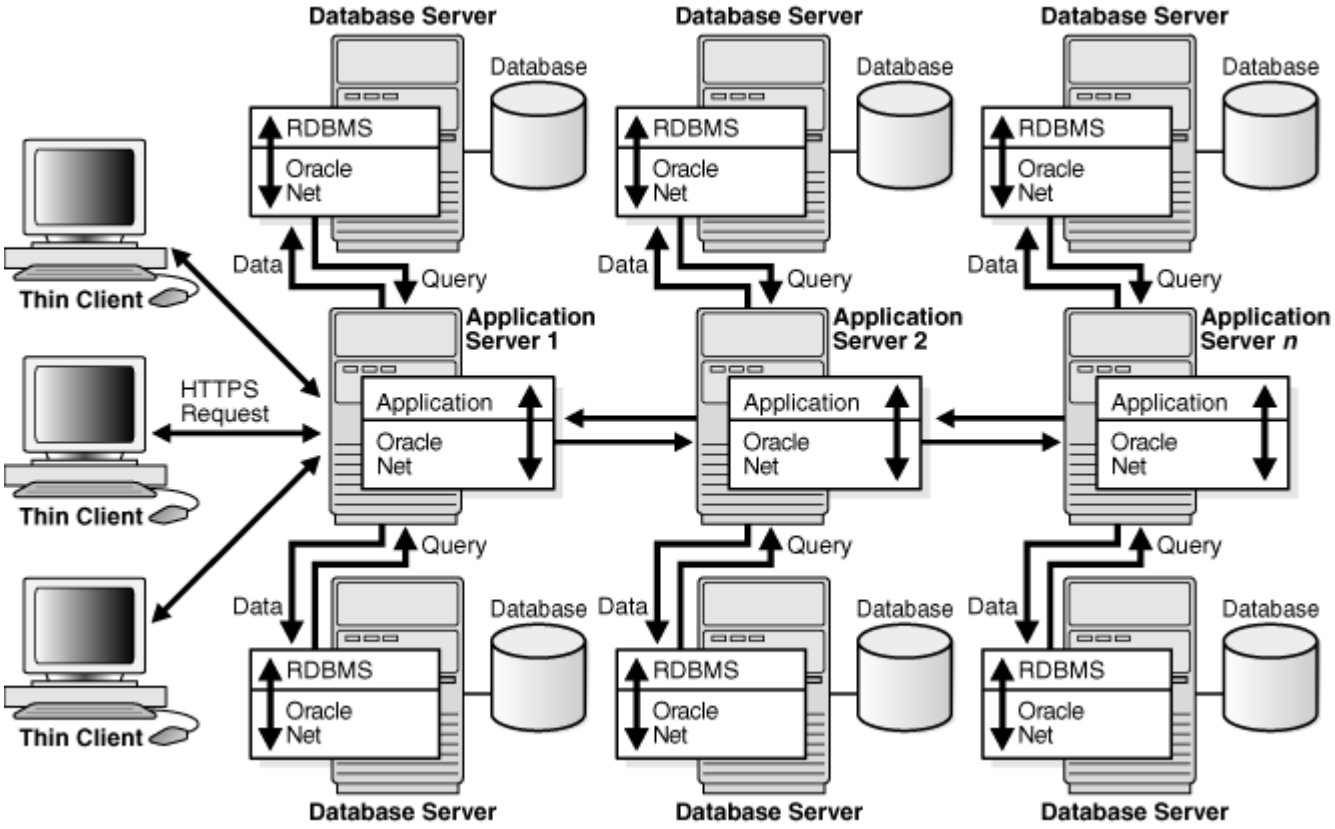
従来の複数層アーキテクチャにおいては、アプリケーション・サーバーはクライアントにデータを提供し、クライアントとデータベース・サーバー間のインタフェースとして機能します。

このアーキテクチャでは、アプリケーション・サーバーを使用して次のことを実行できます。

- Webブラウザなど、クライアントの資格証明の検証
- データベース・サーバーへの接続
- リクエストされた操作の実行

図16-3は、複数層アーキテクチャの例を示しています。

図16-3 複数層アーキテクチャ環境



クライアント

クライアントは、データベース・サーバー上で実行される操作についてリクエストを出します。

このクライアントは、Webブラウザでも他のエンド・ユーザー・プログラムでもかまいません。複数層アーキテクチャでは、クライアントは1つ以上のアプリケーション・サーバーを介してデータベース・サーバーに接続します。

アプリケーション・サーバー

アプリケーション・サーバーは、クライアントにデータ・アクセスを提供します。また、クライアントと、1つ以上のデータベース・サーバーとの間のインタフェースとして機能し、アプリケーションをホスティングします。

アプリケーション・サーバーでは、最小限のソフトウェア構成を備えたクライアントである、Thinクライアントから、クライアントの継続的なメンテナンスを必要とせずにアプリケーションにアクセスできます。また、アプリケーション・サーバーでは、クライアント用にデータを再フォーマットし、クライアント・ワークステーションの負荷を軽減できます。

アプリケーション・サーバーは、クライアントのためにデータベース・サーバー上で操作を実行するとき、そのクライアントの識別を想定します。クライアント操作中に不要な操作や望ましくない操作を実行できないように、アプリケーション・サーバーの権限を制限することが、最良の方法です。

データベース・サーバー

データベース・サーバーは、クライアントにかわってアプリケーション・サーバーからリクエストされたデータを提供します。問合せ処理は、データベースによって実行されます。

データベース・サーバーは、クライアントおよび自身のためにアプリケーション・サーバーで実行された操作を監査できます。たとえば、クライアント操作はクライアントに表示する情報をリクエストでき、アプリケーション・サーバー操作はデータベース・サーバーに接続をリクエストできます。

統合監査で、データベースでは、アプリケーション・コンテキスト(アプリケーション固有の名前/値のペア)を統合監査証跡のレコードに追加できます。データベースによってデータベース監査レコードに書き込まれるアプリケーション・コンテキストを構成できます。

関連項目:

- 「データ・アクセスの監視」
- 「データベース監査」

サービス指向アーキテクチャ(SOA)

データベースは、従来の複数層または**サービス指向アーキテクチャ(SOA)**環境でWebサービス・プロバイダとして機能できるようになりました。

SOAはネットワーク上にあるコンピュータ間での相互作用をサポートするサービスに依存する、複数層アーキテクチャです。SOAとの関連で、サービスは自給自足の機能的エンドポイントで、機能とサービス・レベルの明確な合意があり、監視および管理が可能で、ポリシー・コンプライアンスの実施に役立ちます。

通常、SOAサービスはHTTPプロトコルを介してアクセスできるWebサービスとして実装されます。これらのサービスは、WSDLやSOAPなどのXML標準に基づいています。

Oracle XML DBの一部として実装されるOracle DatabaseのWebサービス機能は、DBAが明示的に有効化する必要があります。アプリケーションは、データベースWebサービスを介して次のことを実行できます。

- SQLまたはXQuery問合せの発行およびXMLでの結果の受信
- スタンドアロンPL/SQLファンクションの起動および結果の受信
- PL/SQLパッケージ・ファンクションの起動および結果の受信

データベースWebサービスを使用すると、アプリケーション・サーバーがなくても、アプリケーション環境に簡単にWebサービスを追加できます。ただし、Oracle Fusion Middlewareなどのアプリケーション・サーバーを介してWebサービスを起動すると、SOA環境で、セキュリティ、スケーラビリティ、UDDI登録および信頼できるメッセージ機能の面でさらに利点があります。ただし、データベースWebサービスはOracle Fusion Middlewareと簡単に統合できるため、SOAソリューションの最適化に適しています。

関連項目:

- 「PL/SQLサブプログラム」
- データベースWebサービスを有効化する方法および使用方法の詳細は、『Oracle XML Developer's Kitプログラマーズ・ガイド』を参照してください
- SOAおよびWebサービスの詳細は、Oracle Fusion Middlewareのマニュアルを参照してください。

グリッド・アーキテクチャの概要

Oracle Database環境では、**グリッド・コンピューティング**は、柔軟なオンデマンド・コンピューティング・リソースに多くのサーバーおよび記憶域を効率的にプールするコンピューティング・アーキテクチャです。

ビジネスのニーズの変化に対応して、必要であればモジュール化したハードウェアおよびソフトウェア・コンポーネントのグループを接続および再結合できます。

関連項目:

サーバーおよび記憶域グリッドの詳細は、「グリッド・コンピューティングの概要」を参照してください

Oracle Net Servicesのアーキテクチャの概要

Oracle Net Servicesは分散した、異機種間コンピューティング環境において、企業全体の接続性を確保する一連のネットワーク・コンポーネントです。

Oracle Net Servicesを使用すると、アプリケーションからデータベース・インスタンスへ、およびデータベース・インスタンスから別のデータベース・インスタンスへのネットワーク・セッションを確立できます。

Oracle Net Servicesにより、位置の透過性、構成と管理の一元化および迅速なインストールと構成が実現されます。また、システム・リソースの最大化とパフォーマンスの改善が可能です。共有サーバー・アーキテクチャでは、アプリケーションのスケーラビリティが向上し、データベースへのクライアントの同時接続数が増加します。Virtual Interface (VI)プロトコルは、メッセージ機能の負荷のほとんどを高速ネットワーク・ハードウェアに負わせることによって、CPUを解放します。

Oracle Net Servicesは、広範囲のネットワークでサポートされている通信プロトコルやApplication Program Interface(API)を使用して、分散データベースと分散処理の機能を提供します。ネットワーク・セッションの確立後、Oracle Net Servicesは、接続を確立してメッセージを交換することによって、クライアント・アプリケーションとデータベース・サーバーのためのデータ伝達手段として機能します。Oracle Net Servicesはネットワーク内の各コンピュータに配置されているため、これらのタスクを実行できます。

この項では、次の項目について説明します。

- Oracle Net Servicesの動作
- Oracle Net Listener
- 専用サーバー・アーキテクチャ
- 共有サーバー・アーキテクチャ
- データベース常駐接続プーリング

関連項目:

Oracle Netアーキテクチャの概要は、『Oracle Database Net Services管理者ガイド』を参照してください

Oracle Net Servicesの動作

Oracle Databaseプロトコルでは、Oracleアプリケーションのインタフェースから SQL文を受け取り、それらをパッケージ化してからOracle Databaseに送信します。

送信は、サポートされている業界標準の高レベルのプロトコルまたはAPIを介して行われます。Oracle Databaseからの応答は、同じ高レベルの通信メカニズムを介してパッケージ化されます。これらの処理はネットワーク・オペレーティング・システムからは独立して実行されます。

Oracle Databaseを実行するオペレーティング・システムによっては、データベース・サーバーのOracle Net Servicesソフトウェアにドライバ・ソフトウェアが含まれており、ドライバ・ソフトウェアによって追加のバックグラウンド・プロセスが起動される場合があります。

関連項目:

Oracle Net Servicesの動作の詳細は、『Oracle Database Net Services管理者ガイド』を参照してください

Oracle Net Listener

Oracle Net Listener(リスナー)は、受信するクライアント接続リクエストをリスニングし、データベースへのトラフィックを管理する、サーバー側のプロセスです。データベース・インスタンスの起動時、および起動中の様々なタイミングで、インスタンスはリスナーに接続してそのインスタンスへの接続経路を確立します。

サービス登録により、リスナーは、データベース・サービスおよびそのサービス・ハンドラが使用可能かどうかを判断できます。サービス・ハンドラは、データベースへの接続ポイントとなる、ディスパッチャまたは専用サーバー・プロセスです。登録時に、LREGプロセスは、インスタンス名、データベース・サービス名、およびサービス・ハンドラのタイプとアドレスをリスナーに提供します。リスナーは、この情報を使用することにより、クライアント・リクエストの受信時にサービス・ハンドラを起動できます。

次の図に、それぞれ個別のホスト上に存在する2つのデータベースを示します。このデータ環境は、異なるホスト上にある2つのリスナーが対応しています。各データベース・インスタンスで実行されているLREGプロセスは、両方のリスナーと通信してデータベースを登録します。

図16-4 2つのリスナー

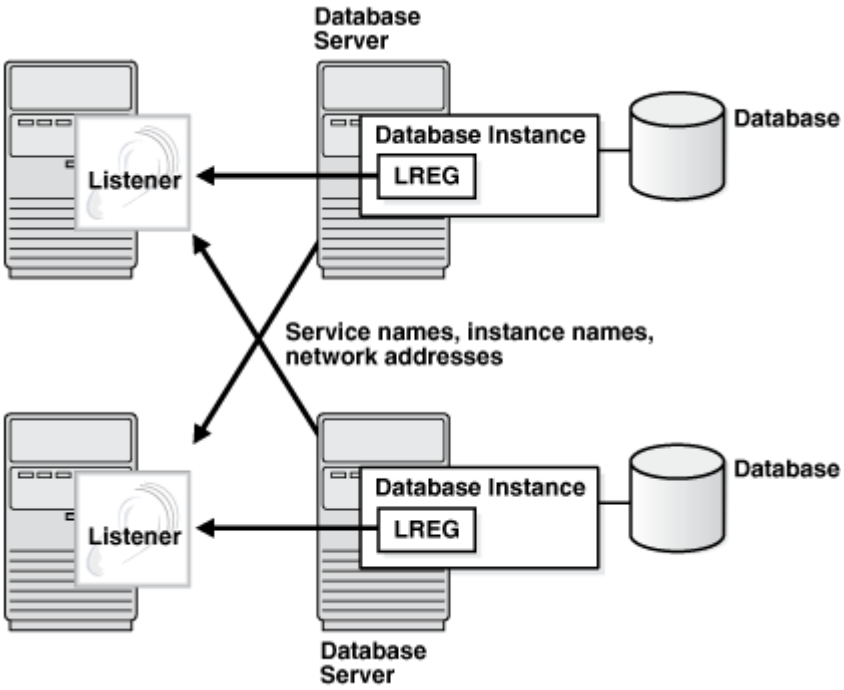
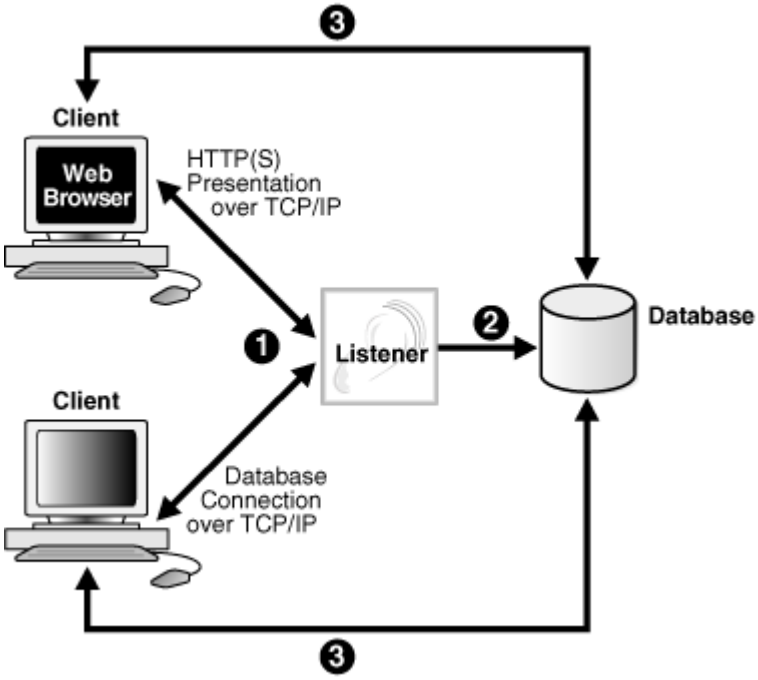


図16-5は、ブラウザによるHTTP接続の実行と、クライアントによるリスナーを介したデータベース接続の実行を示します。このリスナーは、データベース・ホストに常駐する必要はありません。

図16-5 リスナー・アーキテクチャ



クライアントがリスナーを経由して接続を確立する基本ステップを次に示します。

1. クライアント・プロセスまたは別のデータベースが接続をリクエストします。
2. リスナーは、クライアント・リクエストの処理に適したサービス・ハンドラを選択し、そのリクエストをハンドラに送ります。
3. クライアント・プロセスは、サービス・ハンドラに直接接続します。リスナーは通信には関わりません。

関連項目:

「クライアント・プロセスの概要」および「サーバー・プロセスの概要」

サービス名

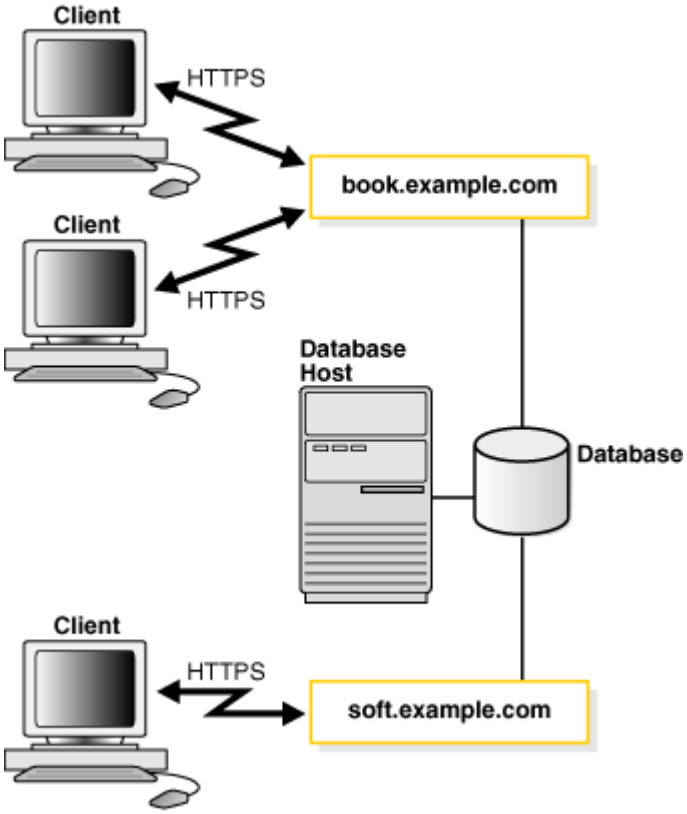
サービス名とは、クライアント接続に使用するサービスの論理表現です。

クライアントは、リスナーに接続するときにサービスへの接続をリクエストします。データベース・インスタンスは、起動時にインスタンス自体をリスナーに登録し、1つ以上のサービスとして名前で指定できるようになります。このようにして、リスナーはクライアントとインスタンスとの間の仲介役を果たし、接続リクエストを適切な場所に渡します。

各サービスは、リスナーを介して、1つ以上のデータベース・インスタンスを識別できます。また、各データベース・インスタンスは、1つ以上のサービスをリスナーに登録できます。サービスに接続するクライアントは、必要なインスタンスを指定する必要がありません。

図16-6は、book.example.comとsoft.example.comの2つのサービスに関連付けられた1つの単一インスタンス・データベースを示します。これらのサービスにより、同じデータベースを異なるクライアントが個別に識別できるようになります。データベース管理者はシステム・リソースを制限または確保できるため、これらのサービスの1つをリクエストするクライアントに対してリソースをより効率的に割り当てることができます。

図16-6 1つのデータベースに関連付けられた複数のサービス



関連項目:

ネーミング・メソッドについてさらに学習するには、Oracle Database Net Services管理者ガイドを参照してください

サービス登録

Oracle Netでの**サービス登録**は、LREGプロセスがリスナーに動的にインスタンス情報を登録する機能です。

この情報により、リスナーはクライアント接続リクエストを適切なサービス・ハンドラに転送することができます。LREGによって、次の情報がリスナーに提供されます。

- データベースが提供するデータベース・サービスの名前
- サービスと関連付けられたデータベース・インスタンスの名前、およびそのインスタンスの現在の負荷と最大負荷
- インスタンスから使用可能なサービス・ハンドラ(ディスパッチャおよび専用サーバー)とそのタイプ、プロトコル・アドレスおよび現在の負荷と最大負荷

サービス登録は動的であり、`listener.ora`ファイルでの構成は必要ありません。動的登録により、複数のデータベースやインスタンスの管理に伴うオーバーヘッドが低減します。

初期化パラメータSERVICE_NAMESでは、インスタンスが属するサービスを示します。起動時に、各インスタンスは、同じサービスに属している他のインスタンスのリスナーに登録します。データベース操作中に、各サービスのインスタンスはCPUの使用と現行の接続カウントに関する情報を、同じサービスのすべてのリスナーに渡します。この通信により、動的なロード・バランシング機能と接続フェイルオーバー機能が使用可能になります。

関連項目:

- 「リスナー登録プロセス(LREG)」
- サービス登録についてさらに学習するには、『Oracle Database Net Services管理者ガイド』を参照してください

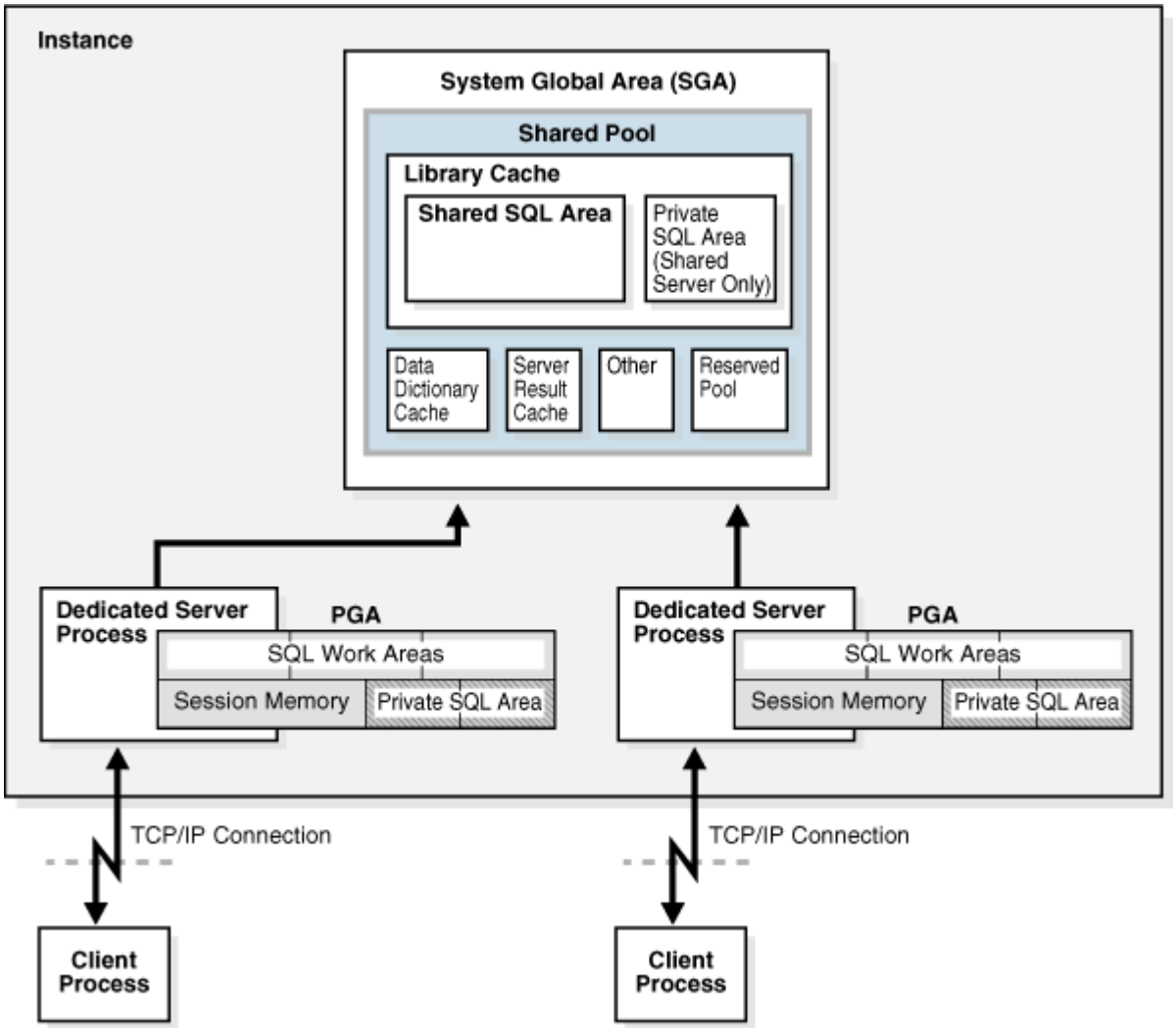
- Oracle RACでのインスタンス登録およびクライアント/サービス接続について学習するには、『Oracle Real Application Clusters管理およびデプロイメント・ガイド』を参照してください

専用サーバー・アーキテクチャ

専用サーバー・アーキテクチャでは、各クライアント・プロセスのかわりとして作成されたサーバー・プロセスは専用サーバー・プロセス(またはシャドウ・プロセス)と呼ばれます。

図16-7に示すように、専用サーバー・プロセスはクライアント・プロセスとは別のものであり、クライアント・プロセスのかわりとしてのみ動作します。

図16-7 Oracle Databaseでの専用サーバー・プロセスの使用



クライアント・プロセスとサーバー・プロセスの比率は1対1です。ユーザーが活発にデータベース・リクエストをしていない場合でも、専用サーバー・プロセスはそのまま残ります(ただし、アクティブでない状態の場合、オペレーティング・システムによっては、ページアウトされることもあります)。

図16-7には、ユーザー・プロセスとサーバー・プロセスが、ネットワーク接続された複数のコンピュータで実行されている状態を示しています。専用サーバー・アーキテクチャは、同じコンピュータ上でクライアント・アプリケーションとデータベース・コードの両方が実行される場合にも使用されますが、この2つのプログラムが1つのプロセスで実行されている場合は、ホスト・オペレーティング・システムはその2つの分離を維持できません。Linuxはそのようなオペレーティング・システムの一例です。

専用サーバー・アーキテクチャでは、ユーザー・プロセスとサーバー・プロセスは異なるメカニズムを使用して通信します。

- クライアント・プロセスと専用サーバー・プロセスが同じコンピュータで実行される場合、プログラム・インタフェースは、ホス

ト・オペレーティング・システムのプロセス間通信メカニズムを使用してジョブを実行します。

- クライアント・プロセスと専用サーバー・プロセスを別々のコンピュータで実行する場合は、プログラム・インタフェースがプログラム間の通信メカニズム(ネットワーク・ソフトウェアやOracle Net Servicesなど)を提供します。

専用サーバーを十分に活用しない場合は、オペレーティング・システムを効率的に使用できなくなることがあります。専用サーバー・プロセスを使用した注文入力システムを例にとって考えてみます。顧客から注文が入り、事務担当がデータベースにその注文を入力します。トランザクションの大部分では、事務担当が顧客と話し合っており、事務担当のクライアント・プロセス専用のサーバー・プロセスはアイドル状態になっています。サーバー・プロセスは、トランザクションの大部分で不要であり、管理されているプロセスが多数ある場合、他の事務担当が注文を入力するとき、システムの処理速度は遅くなることがあります。このようなタイプのアプリケーションでは、共有サーバー・アーキテクチャを選択してください。

関連項目:

専用サーバー・プロセスについてさらに学習するには、Oracle Database Net Services管理者ガイドを参照してください

共有サーバー・アーキテクチャ

共有サーバー・アーキテクチャでは、ディスパッチャにより、複数の受信されたネットワーク・セッション・リクエストが共有サーバー・プロセスのプールに入れられます

共有プールを使用すると、各接続の専用サーバー・プロセスの必要性が排除されます。プールにあるアイドル状態の共有サーバー・プロセスは、共通キューからリクエストをピックアップします。

共有サーバーには、次のような潜在的な利点があります。

- オペレーティング・システム上のプロセス数の削減

  少数の共有サーバーで、多数の専用サーバーと同じ量の処理を実行できます。

- インスタンスPGAメモリーの削減

  すべての専用サーバーおよび共有サーバーにはPGAがあります。サーバー・プロセスの数が減少すると、PGAおよびプロセス管理も減少します。

- アプリケーションのスケーラビリティ向上、およびデータベースに同時に接続できるクライアントの数の増加
- クライアントの接続と切断の比率が高いときに専用サーバーより高速である可能性の高さ

共有サーバーには、特定のケースにおけるレスポンス時間の長さ、機能の不完全なサポート、および設定やチューニングの複雑化などのデメリットがあります。一般的な目安としては、データベースへの同時接続数がオペレーティング・システムで処理可能な数を超える場合にのみ、共有サーバーを使用してください。

共有サーバー・アーキテクチャでは、次のプロセスが必要となります。

- クライアント・プロセスをディスパッチャまたは専用サーバーに接続するネットワーク・リスナー(リスナー・プロセスは、Oracle DatabaseではなくOracle Net Servicesの一部)

ノート:

共有サーバーを使用する際は、クライアント・プロセスは、Oracle データベース・インスタンスと同一のコン

ピュータで実行されている場合でも、Oracle Net Services を介して接続する必要があります。

- 1つ以上のディスパッチャ・プロセス(Dnnn)
- 1つ以上の共有サーバー・プロセス

データベースは、共有サーバー接続と専用サーバー接続の両方を同時にサポートできます。たとえば、1つのクライアントが専用サーバーを使用してデータベースに接続すると同時に、別のクライアントが共有サーバーを使用して同じデータベースに接続できます。

関連項目:

- 共有サーバー・アーキテクチャの詳細は、『Oracle Database Net Services管理者ガイド』を参照してください
- 共有サーバーを使用するためのデータベースの構成方法を学習するには、『Oracle Database管理者ガイド』を参照してください。

ディスパッチャのリクエスト・キューとレスポンス・キュー

ユーザーからのリクエストは、ユーザーのSQL文の一部である単一のAPIコールです。

ユーザーがコールを実行すると、次のアクションが発生します。

1. ディスパッチャがリクエストをリクエスト・キューに入れ、次に使用可能な共有サーバー・プロセスがそこからリクエストを取り出します。

   リクエスト・キューはSGA内に存在し、インスタンスのディスパッチャ・プロセスすべてに共通です。

2. 共有サーバー・プロセスは、共通のリクエスト・キューをチェックして新しいリクエストがないかどうかを調べ、先入れ先出し方式に基づいて新しいリクエストをピックアップします。

3. 1つの共有サーバー・プロセスがキュー内のリクエストを1つピックアップし、そのリクエストを完了するのに必要なデータベースに対するコールをすべて出します。

   各データベース・コールは異なるサーバー・プロセスが処理できます。このため、問合せの解析、最初の行のフェッチ、次の行のフェッチ、および結果セットのクローズそれぞれのリクエストは、別の共有サーバーが処理できます。

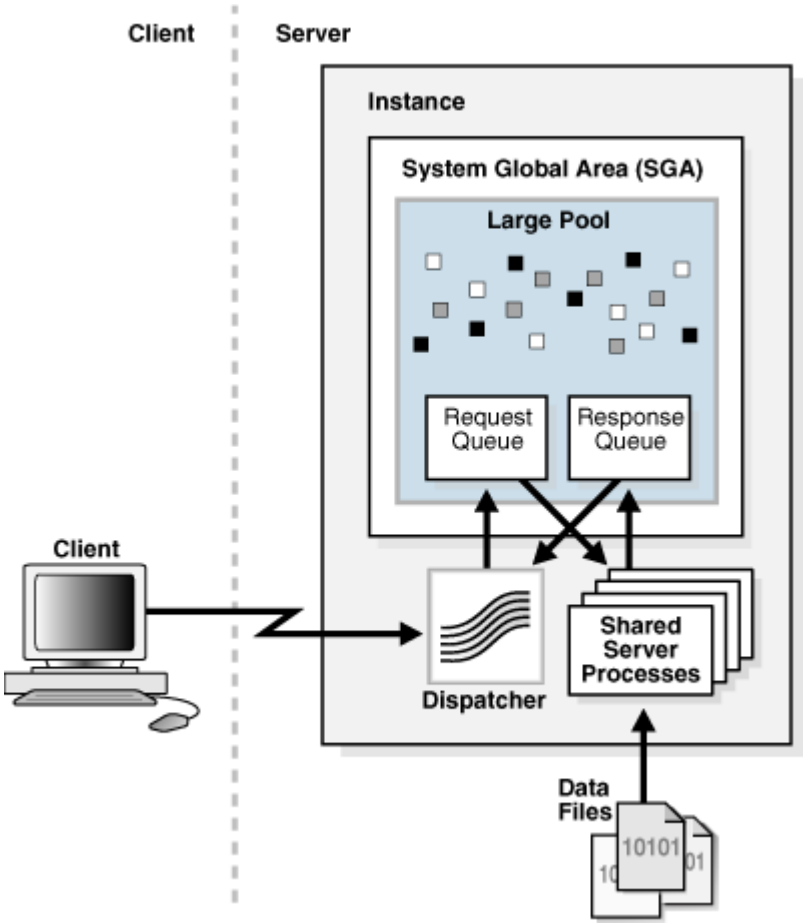
4. サーバー・プロセスはリクエストを完了すると、コール・ディスパッチャのレスポンス・キューに応答を入れます。各ディスパッチャには、固有のレスポンス・キューがあります。

5. ディスパッチャは、完了したリクエストを適切なクライアント・プロセスに戻します。

たとえば、注文入力システムで、各事務担当のクライアント・プロセスがディスパッチャに接続します。事務担当によって作成されたそれぞれのリクエストはこのディスパッチャに送られ、ディスパッチャがリクエストをキューに入れます。次に使用可能な共有サーバーは、リクエストをピックアップして処理し、レスポンス・キューにレスポンスを入れます。リクエストが完了しても、事務担当はディスパッチャに接続されたままですが、そのリクエストを処理した共有サーバーは解放されるため、別のリクエストで使用できます。1人の事務担当が顧客と話し合っている間に、別の事務担当は同じ共有サーバー・プロセスを使用できます。

図16-8に、クライアント・プロセスがAPIを介してディスパッチャと通信する方法と、ディスパッチャがユーザー・リクエストを共有サーバー・プロセスに伝達する方法を示します。

図16-8 共有サーバー構成とプロセス



関連項目:

「ラージ・プール」

ディスパッチャ・プロセス(Dnnn)

ディスパッチャ・プロセスは、クライアント・プロセスが限定された数のサーバー・プロセスを共有できるようにします。

1つのデータベース・インスタンスに対し、複数のディスパッチャ・プロセスを作成できます。ディスパッチャの最適な数は、オペレーティング・システムの制限事項と各プロセスの接続数に応じて異なります。

ノート:

ディスパッチャに接続する各クライアント・プロセスは、両方のプロセスが同じホスト上で実行される場合でも、Oracle Net Services を使用する必要があります。

ディスパッチャ・プロセスは次のように通信を確立します。

1. インスタンスが起動されると、ネットワーク・リスナー・プロセスは、ユーザーがOracle Databaseへの接続に使用する通信経路をオープンして確立します。
2. 各ディスパッチャ・プロセスは、接続リクエストをリスニングするアドレスをリスナー・プロセスに割り当てます。

   データベース・クライアントで使用するネットワーク・プロトコルごとに、最低1つ以上のディスパッチャ・プロセスを構成し起動する必要があります。

3. クライアント・プロセスが接続をリクエストすると、リスナーはそのクライアント・プロセスが共有サーバー・プロセスを使用する必要があるかどうかを判断します。
    - 共有サーバーが必要であるとリスナーが判断した場合、リスナーは負荷が最も軽いディスパッチャ・プロセスのアドレスを戻し、クライアント・プロセスはディスパッチャに直接接続します。
    - プロセスがディスパッチャと通信できない場合、またはクライアント・プロセスが専用サーバーをリクエストする場合には、リスナーは専用サーバー・プロセスを作成して適切な接続を確立します。

関連項目:

ディスパッチャの構成方法を学習するには、『Oracle Database Net Services管理者ガイド』を参照してください

共有サーバー・プロセス(Snnn)

共有サーバー構成では、各共有サーバー・プロセスは複数のクライアント・リクエストを処理します。

共有サーバー・プロセスには専用サーバー・プロセスと同じ機能がありますが、特定のクライアント・プロセスと関連付けられていません。共有サーバー・プロセスは、共有サーバー構成におけるクライアント・リクエストに対してサービスを提供します。

共有サーバー・プロセスのPGAには、すべての共有サーバー・プロセスがアクセスできる必要のあるUGAデータが含まれていません。共有サーバーのPGAには、プロセス固有のデータのみが含まれています。

セッション関連の情報はすべてSGA内に入れられます。各共有サーバー・プロセスは、サーバーがどのセッションからのリクエストでも処理できるように、すべてのセッションのデータ領域にアクセスできる必要があります。セッションのデータ領域ごとに、SGA内に領域が割り当てられます。

関連項目:

「プログラム・グローバル領域(PGA)の概要」

共有サーバーの限定的運用

インスタンスの停止、インスタンスの起動、およびメディア・リカバリを含む、特定の管理アクティビティは、ディスパッチャ・プロセスに接続されている間は実行できません。

これらのアクティビティは、一般的には管理者権限を使用して接続しているときに実行されます。共有サーバーにより構成されたシステムで管理者権限を使用して接続する場合は、専用サーバー・プロセスを使用することを指定する必要があります。

関連項目:

正しい接続文字の構文は、『Oracle Database Net Services管理者ガイド』を参照してください

データベース常駐接続プーリング

データベース常駐接続プーリング(DRCP)は、一般的なWebアプリケーション使用のシナリオにおいて、専用サーバーの接続プールを提供します。

通常、Webアプリケーションは、データベースへの接続を取得し、その接続を短時間使用した後、接続を解放します。DRCPを使用すると、データベースは同時接続数を数万にまで増やすことができます。

DRCPには、次の利点があります。

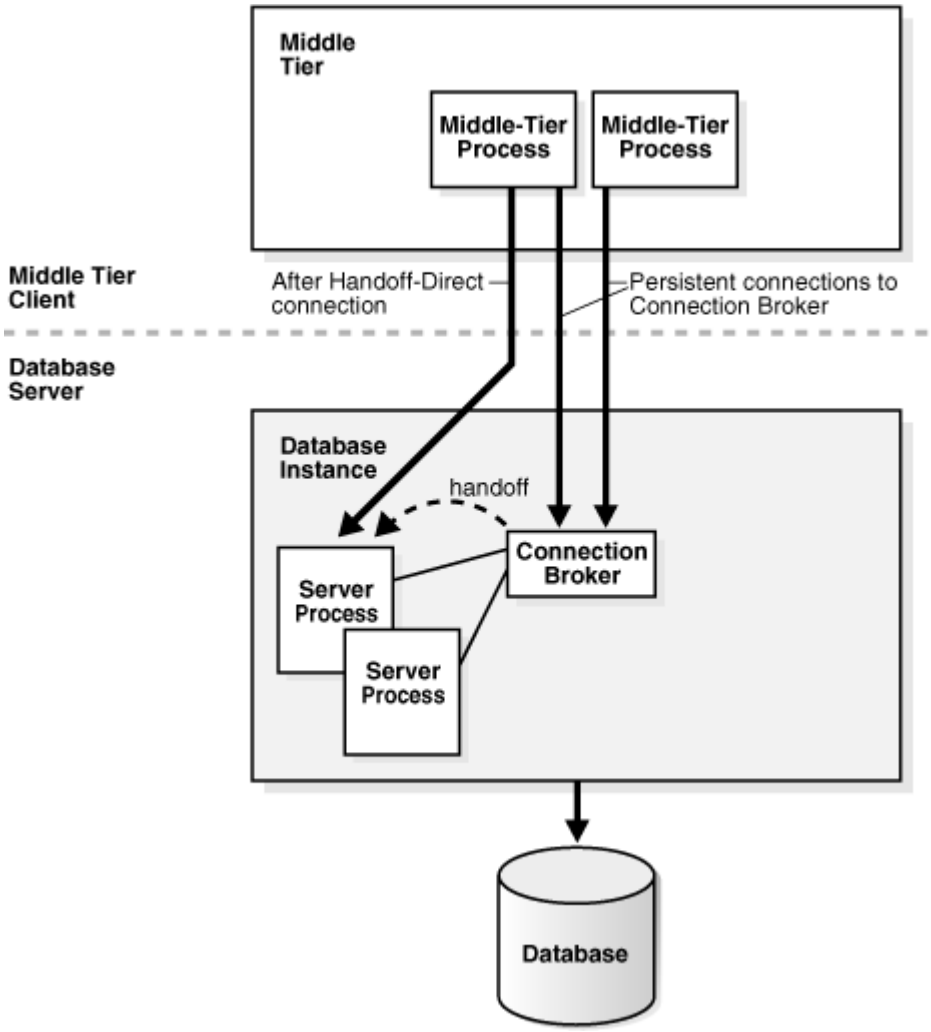
- 中間層プロセス内のスレッド間で接続を共有する、中間層接続プールを補完します。
- 複数の中間層プロセスにわたってデータベース接続を共有できるようにします。これらの中間層プロセスは、中間層ホストが同一の場合と異なる場合があります。
- 多数のクライアント接続のサポートに必要な、主要データベース・リソースを大幅に削減できます。たとえば、DRCPは、データベースに必要なメモリーの量を削減し、データベースと中間層の両方のスケーラビリティを向上させます。利用可能なサーバーのプールにより、クライアント接続を再作成するコストも削減されます。
- 中間層接続プーリングを実行できないPHPやApacheサーバーなど、マルチ・プロセスのシングルスレッド・アプリケーション・サーバーを含むアーキテクチャでプーリングが実行できるようになります。

DRCPでは、**プールされたサーバー**を使用し、専用サーバー・プロセス(共有サーバー・プロセスではない)とデータベース・セッションを組み合せたものと同じ機能を実現します。プールされたサーバー・モデルは、短時間だけサーバーを必要とする接続のすべてに専用サーバーを割り当てることによるオーバーヘッドを排除します。

データベース常駐接続プールから接続を取得したクライアントは、接続ブローカと呼ばれるOracleバックグラウンド・プロセスに接続します。接続ブローカはプール機能を実装しており、プールされたサーバーをクライアント・プロセスからのインバウンド接続間で多重化します。

図16-9に示すように、クライアントがデータベース・アクセスをリクエストすると、接続ブローカがプールから、サーバー・プロセスを選択してクライアントに渡します。クライアントは、リクエストが処理されるまで、サーバー・プロセスに直接接続します。サーバーでの処理が終了すると、サーバー・プロセスはプールに戻されます。クライアントからの接続はブローカにリストアされます。

図16-9 DRCP



DRCPでは、リソースを解放しても、セッションはそのまま残されますが、接続(サーバー・プロセス)との関連付けは失われます。共有サーバーとは異なり、このセッションではそのUGAをSGAではなくPGAに格納します。クライアントは、検出アクティビティを実行するとき、接続を透過的に再確立できます。

関連項目:

- 「接続とセッション」
- DRCPについてさらに学習するには、Oracle Database管理者ガイドおよびOracle Call Interfaceプログラマーズ・ガイドを参照してください

プログラム・インタフェースの概要

プログラム・インタフェースとは、データベース・アプリケーションとOracle Databaseの間のソフトウェア・レイヤーです。

プログラム・インタフェースでは、次の機能を実行します。

- クライアント・プロセスによるSGAへの破壊的アクセスを防ぐ、セキュリティのための障壁になります。
- 通信メカニズムとして機能し、情報リクエストの書式設定、データの受渡し、エラーのトラップと通知を行います。
- 特に異機種コンピュータ間、または外部ユーザー・プログラム・データ型に対してデータを変換し解釈します。

Oracleコードはサーバーとして働き、アプリケーション(クライアント)のために、データ・ブロックからの行のフェッチなどデータベース・タスクを実行します。プログラム・インタフェースは、Oracle Databaseソフトウェアとオペレーティング・システム固有のソフトウェアの両方によって提供される、複数の部分から構成されています。

プログラム・インタフェースの構造

プログラム・インタフェースは、様々なコンポーネントから構成されます。

これらのコンポーネントは、次のとおりです。

- Oracleコール・インタフェース(OCI)またはOracleランタイム・ライブラリ(SQLLIB)
- クライアント側またはユーザー側のプログラム・インタフェース
- 様々なOracle Net Servicesドライバ(プロトコル固有の通信ソフトウェア)
- オペレーティング・システムの通信ソフトウェア
- サーバーまたはOracle Database側のプログラム・インタフェース(OPIとも呼ばれます)

ユーザー側とOracle Database側のプログラム・インタフェースは、どちらもドライバのような仕組みでOracleソフトウェアを実行します。

プログラム・インタフェース・ドライバ

ドライバとは、通常はネットワークを介してデータを転送するソフトウェアの一部です。

ドライバは、接続、切断、エラーの通知、エラーのテストなどの操作を実行します。ドライバは通信プロトコルに固有のものです。

デフォルトのドライバが常に存在します。複数のドライバ(非同期またはDECnetドライバなど)をインストールし、その1つをデフォルト・ドライバとして選択しますが、各ユーザーは、接続の時点で必要なドライバを指定することで、他のドライバを使用できます。

プロセスが異なる場合は、異なるドライバを使用できます。1つのプロセスは、異なるOracle Net Servicesドライバを使用して、単一または複数のデータベースに同時接続できます。

関連項目:

- ドライバの選択、インストールおよび追加方法の詳細は、システムのインストレーションおよび構成ガイドを参照してください。
- JDBCドライバについて学習するには、『Oracle Database Net Services管理者ガイド』を参照してください

オペレーティング・システムの通信ソフトウェア

ユーザー側をOracle Database側のプログラム・インタフェースに接続する最下位層のソフトウェアは、ホスト・オペレーティング・システムによって提供される通信ソフトウェアです。

たとえば、DECnet、TCP/IP、LU6.2、ASYNCなどです。通信ソフトウェアはOracleから提供されることもありますが、通常、ハードウェア・ベンダーまたはサード・パーティのソフトウェア・サプライヤから別途、購入します。

第VI部 Oracle Databaseの管理およびアプリケーション開発

この部では、データベースの管理者およびアプリケーション開発者にとって重要なトピックの概要について説明します。

この部は、次の章で構成されています。

- データベース管理者および開発者向けのトピック
- データベース管理者の概念
- データベース開発者の概念

17 データベース管理者および開発者向けのトピック

この章では、データベース管理者と開発者にとって重要で一般的なデータベース・トピックを要約し、ここで網羅的にデータベースの機能を説明するかわりに、参照可能な他のマニュアルを紹介します。

この章の構成は、次のとおりです。

- データベース・セキュリティの概要
- 高可用性の概要
- グリッド・コンピューティングの概要
- データ・ウェアハウスとビジネス・インテリジェンスの概要
- Oracleにおける情報統合の概要

関連項目:

DBAに固有のトピックについては「データベース管理者の概念」を、データベース開発者に固有のトピックについては「データベース開発者の概念」を参照してください

データベース・セキュリティの概要

通常、**データベース・セキュリティ**には、ユーザー認証、暗号化、アクセス制御および監視が含まれます。

この項では、次の項目について説明します。

- ユーザー・アカウント
- データベース認証
- 暗号化
- Oracle Data Redaction
- 方向
- データ・アクセスの監視

ユーザー・アカウント

各Oracleデータベースには、有効なデータベース・ユーザーのリストがあります。

データベースには、デフォルトの管理アカウントSYSTEMを含む、複数のデフォルトのアカウントが含まれます。ユーザー・アカウントは必要に応じて作成できます。アプリケーション・ユーザーをOracleデータベースにアクセスするように構成することもできます。

データベースにアクセスするには、有効なユーザー名と認証用の資格証明を提供する必要があります。資格証明には、パスワード、Kerberosチケットまたは公開キー・インフラストラクチャ(PKI)の証明書などがあります。データベース・セキュリティでは、ログオン試行の失敗に基づいてアカウントをロックするよう構成できます。

通常、データベース・アクセス制御には、データ・アクセスおよびデータベース・アクティビティの制限が含まれます。たとえば、指定された表への問合せや指定されたデータベース文の実行を制限できます。

関連項目:

- 「SYSスキーマとSYSTEMスキーマ」
- 管理ユーザー・アカウントについて学習するには、『Oracle Database管理者ガイド』を参照してください
- アプリケーション・ユーザーの構成方法を学習するには、『Oracle Database Real Application Security管理者および開発者ガイド』を参照してください

権限

ユーザー権限とは、特定のSQL文を実行できる権利のことです。

権限は、次のカテゴリに分類されます。

- システム権限

  これは、データベース内で特定のアクションを実行する権限、または特定のタイプのオブジェクトに対してアクションを実行する権限のことです。システム権限の例として、CREATE USER、CREATE SESSIONがあります。

- オブジェクト権限

  これは、employees表への問合せなど、指定されたアクションをオブジェクトに対して実行できる権限のことです。権限のタイプは、データベースによって定義されます。

権限は、他のユーザーの裁量で任意に付与されます。管理者は、ユーザーが業務に必要な作業を実行できるよう、必要な権限をユーザーに付与する必要があります。セキュリティの観点から、権限は必要な作業を行うためにその権限を必要とするユーザーにのみ付与することをお薦めします。

関連項目:

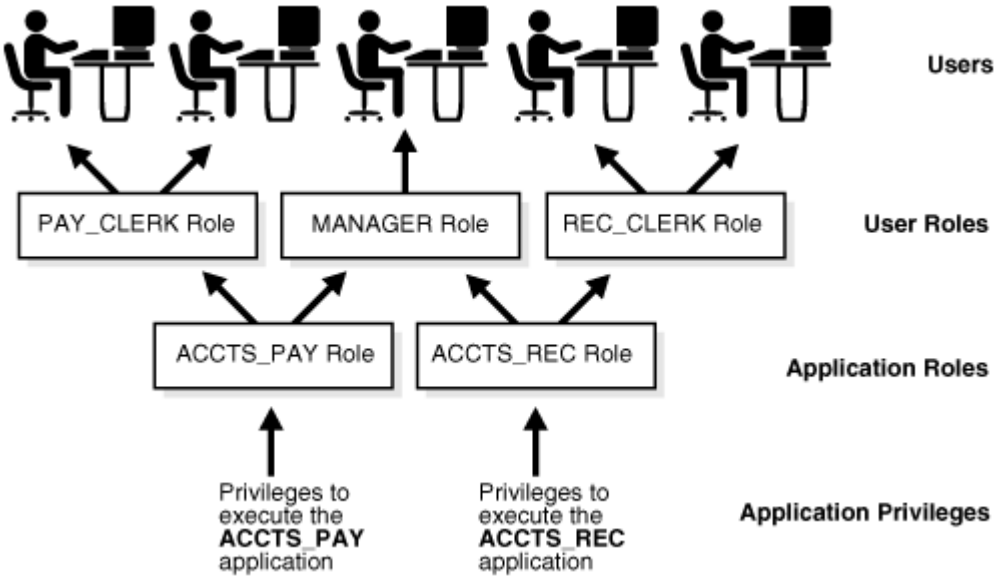
- SESSION_PRIVSビューについて学習するには、Oracle Databaseリファレンスを参照してください。

ロール

ロールは、関連する権限のグループに名前を付けたもので、ユーザーが他のユーザーまたは他のロールに付与できます。ロールを使用すると、データベース・アプリケーションまたはユーザー・グループの権限の管理が容易になります。

図17-1は、ロールの一般的な使用方法を示しています。PAY_CLERK、MANAGER、およびREC_CLERKの各ロールが、異なるユーザーに割り当てられます。ACCTS_PAYアプリケーションを実行する権限を含むACCTS_PAYアプリケーション・ロールが、PAY_CLERKおよびMANAGERロールを持つユーザーに割り当てられます。ACCTS_RECアプリケーションを実行する権限を含むACCTS_RECアプリケーション・ロールが、REC_CLERKおよびMANAGERロールを持つユーザーに割り当てられます。

図17-1 ロールの一般的な使用方法



関連項目:

- セキュリティのためのロールの使用方法について学習するには、Oracle Databaseセキュリティ・ガイドを参照してください
- ロールの管理方法を学習するには、Oracle Database管理者ガイドを参照してください

権限分析

権限分析メカニズムによって、指定した条件に従ってデータベースの権限使用状況を取得できます。

こうして、アプリケーション・モジュールの実行または特定のSQL文の実行に必要な権限を取得できます。たとえば、ユーザーが特定のデータベース・セッション中に行使した権限を見つけることができます。

本番データベースでは、権限とロール、ロールとロール、ロールとユーザーの関係は、複雑になる可能性があります。権限分析により、複雑なシステム内で不必要に付与されている権限を特定できます。取得した結果の分析に基づき、不必要な付与を削除することや、権限付与を再構成してデータベースをよりセキュアにすることができます。

関連項目:

権限分析について学習するには、『Oracle Databaseセキュリティ・ガイド』を参照してください。

ユーザー・プロファイル

システム・リソースのコンテキストにおける**ユーザー・プロファイル**とは、リソース制限とパスワード・パラメータをセットにして名前を付けたもので、ユーザーに対してデータベースの使用およびデータベース・インスタンス・リソースを制限します。

プロファイルでは、ユーザーが同時に確立できるセッションの数、各セッションに使用可能なCPU処理時間、および使用可能な論理I/O量を制限できます。たとえば、clerkプロファイルでは、ユーザーのシステム・リソースが事務作業に必要な程度に制限されます。

ノート:

リソースの制限にはデータベース・リソース・マネージャを使用し、パスワードの管理にはプロファイルを使用することをお薦めします。

プロファイルを使用すると、一連の属性を共有する複数のユーザーに対して1つの参照先を指定できます。一連のユーザーにプロファイルを割り当て、それ以外のすべてのユーザーにデフォルト・プロファイルを割り当てることができます。各ユーザーには、任意の時点で最大1つのプロファイルを割り当てることができます。

関連項目:

- 「バッファI/O」
- プロファイルを使用したリソースの管理方法を学習するには、『Oracle Databaseセキュリティ・ガイド』を参照してください
- CREATE PROFILEの構文およびセマンティクスについては、『Oracle Database SQL言語リファレンス』を参照してください

データベース認証

Oracle Databaseでは、**データベース認証**は、ユーザーがデータベースに資格証明を提示し、この資格証明をデータベースが検証してデータベースへのアクセスを許可するプロセスです。

アイデンティティを検証することで、その後の対話に関する信頼関係が確立されます。また、認証により、アクセスとアクションを特定のアイデンティティにリンクでき、アカウンタビリティが有効化されます。

Oracle Databaseでは、次のものを含む様々な認証方式が提供されています。

- データベースによる認証

  Oracleデータベースは、パスワード、KerberosチケットまたはPKIの証明書を使用してユーザーを認証できます。Oracleでは、生体認証を含む、他の認証方式用のRADIUS準拠のデバイスもサポートしています。認証のタイプは、Oracleデータベースでユーザーを作成する際に指定する必要があります。

- オペレーティング・システムによる認証

  一部のオペレーティング・システムでは、オペレーティング・システムがユーザー認証のために管理している情報を、Oracle Databaseで使用できます。オペレーティング・システムから認証を受けたユーザーは、ユーザー名やパスワードを指定することなくデータベースに接続できます。

非管理のデータベース・ユーザー・アカウントは、データベースの停止や起動などのデータベース操作を実行することはできません。これらの操作には、SYSDBA、SYSOPER、SYSBACKUPまたはSYSDG権限が必要です。

関連項目:

- 「管理者権限での接続」
- 認証方式について学習するには、『Oracle Databaseセキュリティ・ガイド』を参照してください
- 管理認証について学習するには、『Oracle Database管理者ガイド』を参照してください

暗号化

Oracle Databaseでは、**暗号化**は、秘密キーと暗号化アルゴリズムを使用して、データを読取り不可能な形式に変換するプロセスです。

多くの場合、暗号化は、Payment Card Industry Data Security Standard(PCI-DSS)や侵害通知法に関連する要件などの法令遵守要件を満たすために使用されます。たとえば、クレジット・カード番号、社会保障番号、患者の病状に関わる情報などを暗号化する必要があります。

ネットワーク暗号化

クライアントとサーバー間のネットワーク上を移動するデータを暗号化することを、**ネットワーク暗号化**といいます。

侵入者は、ネットワーク・パケット・スニファを使用することにより、ネットワーク上を移動する情報を取得してファイルにスプールし、悪用できます。このような行為は、ネットワーク上のデータを暗号化することによって防ぐことができます。

透過的データ暗号化

Oracle Advanced Securityの**透過的データ暗号化**を使用すると、表の個々の列または表領域を暗号化できます。

ユーザーが暗号化された列にデータを挿入すると、データベースによって自動的にデータが暗号化されます。ユーザーが列を選択すると、データは復号化されます。このような形式の暗号化は透過的であり、パフォーマンスが高く、実装が容易です。

透過的データ暗号化には、Advanced Encryption Standard(AES)や組込みキー管理などの複数の業界標準の暗号化アルゴリズムが含まれます。

関連項目:

Oracle Database Advanced Securityガイド

Oracle Data Redaction

Oracle Data Redactionは、Oracle Advanced Securityの一部で、これを使用すると、権限の少ないユーザーまたはアプリケーションによって問い合されるデータを隠す(リダクションする)ことができます。このリダクションは、ユーザーがデータを問い合せるときに、リアルタイムで発生します。

データ・リダクションでは次のリダクション機能タイプがサポートされています。

- 完全データ・リダクション

  この場合、表またはビューの指定した列の内容全体がリダクションされます。たとえば、姓のVARCHAR2列には、1つの空白が表示されます。

- 部分データ・リダクション

  この場合、表示された出力の一部がリダクションされます。たとえば、アプリケーションでは、1234で終わるクレジット・カード番号をxxxx-xxxx-xxxx-1234と表示できます。完全と部分のどちらのリダクションにも、正規表現を使用できます。正規表現は、検索パターンに基づいて、データをリダクションできます。たとえば、正規表現を使用して、特定の電話番号または電子メール・アドレスをリダクションできます。

- ランダム・データ・リダクション

  この場合、列のデータ型に応じて、データがランダムに生成された値として表示されます。たとえば、数値1234567は、83933895として表示される可能性があります。

データ・リダクションは、包括的なセキュリティの解決策ではありません。たとえば、直接接続し権限を付与されたユーザーが、リダクションされたデータに対して推論攻撃を実行することを阻止できません。このような攻撃では、リダクションされた列を識別し、削除の処理で、格納された値を推測するSQL問合せを繰り返すことにより、実際のデータに戻そうとします。権限を持つユーザーからの推論およびその他の攻撃を検出し、阻止するには、Oracle Data Redactionと関連のデータベース・セキュリティ製品(Oracle Audit Vault and Database Firewall、Oracle Database Vaultなど)と組み合せることをお薦めします。

データ・リダクションは、次のように機能します。

- DBMS_REDACTパッケージを使用して、指定した表のリダクション・ポリシーを作成します。
- ポリシーで、事前定義のリダクション機能を指定します。
- データベースで表示される列の値が実際の値かそれともリダクションされた値かは、ポリシーによって決まります。データがリダクションされると、そのリダクションがただちに最上位の選択リストに反映されから、ユーザーに表示されます。

次の例では、hr.employees表の従業員ID(employee_id)列をリダクションする完全データ・リダクション・ポリシーが追加されています。

```
BEGIN
 DBMS_REDACT.ADD_POLICY(
   object_schema    => 'hr'
,  object_name      => 'employees'
,  column_name      => 'employee_id'
,  policy_name      => 'mask_emp_ids'
,  function_type    => DBMS_REDACT.FULL
,  expression       => '1=1'
);
END;
/
```

この例では、表現の設定(trueと評価)により、EXEMPT REDACTION POLICY権限を付与されていないユーザーに対して、リダクションが適用されます。

関連項目:

- データ・リダクションについて学習するには、『Oracle Database Advanced Securityガイド』を参照してください。
- DBMS_REDACTについて学習するには、『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

オリエンテーション

Oracle Databaseでは、データへのアクセスを制御するための技術が数多く提供されています。この項では、これらの技術の概要を示します。

Oracle Database Vault

Oracle Database Vaultは、権限を持つユーザーによるアクセスをアプリケーション・データに制限します。

Oracle Database 12cからは、Oracle Database Vaultにより、標準データベース監査のデータ構造が拡張されます。さらに、統合監査に移行すると、データベースでは監査レコードを、Oracle Databaseの監査レコードを一元化するOracle Secure Files内の統合監査証跡に書き込みます。

Oracle Database Vaultを使用すると、データベース、データおよびアプリケーションに対するアクセスのタイミング、場所および

方法を制御できます。このため、法令遵守および責務の分離を実現しながら、関係者による脅威からの保護など、一般的なセキュリティ上の問題にも対応できます。

Oracle Database Vault管理者が責任を果たせるように、データベースではOracle Database Vaultメタデータに行われた構成の変更が強制的に監査されます。これらの変更には、Oracle Database Vault関連の実行の作成、変更および削除、保護されたロールの付与および取消し、Oracle Data PumpやJob Schedulerなどのコンポーネントの認可などが含まれます。

関連項目:

- Oracle Database Vaultの管理者ガイド
- Oracle Database Vault AuditとOracle Database Native Auditの統合について学習するには、『Oracle Databaseセキュリティ・ガイド』を参照してください。

Virtual Private Database(VPD)

Oracle Virtual Private Database (VPD)を使用すると、行および列レベルでセキュリティを確保できます。

セキュリティ・ポリシーによって、予期しないまたは悪意のあるデータの破壊やデータベース・インフラストラクチャの破損からデータベースを保護する方法が確立されます。

VPDでは、権限やロールなどのセキュリティ保護では設定できない詳細な設定が可能です。たとえば、すべてのユーザーに対してemployees表へのアクセスを許可する場合は、セキュリティ・ポリシーを作成して、ユーザーが同じ部門の従業員にアクセスすることを禁止できます。

基本的にデータベースは、Oracle VPDセキュリティ・ポリシーが適用された表、ビューまたはシノニムに対して発行されるSQL文に、動的なWHERE句を追加します。このWHERE句によって、保護データへのアクセスが、セキュリティ・ポリシーに準拠する資格証明を持つユーザーにのみ許可されます。

関連項目:

『Oracle Databaseセキュリティ・ガイド』

Oracle Label Security(OLS)

Oracle Label Security(OLS)では、**セキュリティ・ラベル**を使用してデータ分類の割当てとアクセスの制御を行うことができます。ラベルは、データまたはユーザーのいずれかに割当て可能です。

データに割り当てた場合、ラベルは表に非表示の列として添付でき、SQLに透過性を提供します。たとえば、機密性が高いデータを含む行にはHIGHLY SENSITIVEというラベルを、それほど機密性の高くない行にはSENSITIVEというラベルを付けることができます。ユーザーがデータへのアクセスを試みると、OLSによって、ユーザー・ラベルとデータ・ラベルが比較され、アクセスを許可するかどうかが決定されます。VPDとは異なり、OLSでは、即時利用可能なセキュリティ・ポリシー、およびラベルを定義し格納するためのメタデータ・リポジトリが提供されています。

統合監査が有効である場合、データベースでは、監査オプションの構成および管理のためのポリシーベースのフレームワークが提供されます。OLS操作など、様々なタイプの操作の監査オプションをグループ化し、それらを監査ポリシーとして保存できます。次に、そのポリシーを有効または無効にして、基礎となる監査オプションを実行できます。

OLSポリシーを作成すると必ず、データベースではそのポリシーのラベル列がデータベース監査証跡表に追加されます。OLS監査により、OLS管理者操作のレコードを含む監査レコードを、統合監査証跡に書き込むことができます。

関連項目:

- 「統合監査証跡」
- 『*Oracle Label Security*管理者ガイド』

データ・アクセスの監視

Oracle Databaseには、ユーザー・アクティビティを監視するために複数のツールおよび技術が採用されています。監査は、データ・アクセスの監視のために最も重要なメカニズムです。

データベース監査

データベース監査とは、選択したユーザー・データベース・アクションを監視して記録する処理のことです。

統合監査ポリシーを構成して、次のものを監査できます。

- SQL文、システム権限、スキーマ・オブジェクトおよびロール(直接付与されたシステム権限のグループとして)
- 管理および非管理ユーザー
- アプリケーション・コンテキスト値

  アプリケーション・コンテキストとは、指定されたネームスペースでの属性の名前/値ペアです。アプリケーションでは、データベースでアクションを実行する前に、様々なコンテキストを設定します。たとえば、アプリケーションでは、アプリケーション・イベントのステータスを示すモジュール名とクライアントIDなどの情報が格納されます。アプリケーションでコンテキストを構成できるため、それらに関する情報は監査レコードに追加されます。
- Real Application Security、Oracle Database Vault、Oracle Label Security、Oracle Data PumpおよびOracle SQL*Loaderのダイレクト・パス・イベントのポリシー作成

統合監査証跡では、UNIFIED_AUDIT_TRAILデータ・ディクショナリ・ビューで問い合せることができるRecovery Managerイベントを取得できます。Recovery Managerイベントの統合監査ポリシーは作成しません。

ファイングレイン監査を使用して、特定の表の列を監査し、ポリシー作成時にイベント・ハンドラを関連付けることもできます。統合およびファイングレイン監査の場合、表で特定のデータベース・アクション、またはアクティビティが発生する時期を取得する条件の有無を調べるポリシーを作成できます。たとえば、9:00 p.m.以降にアクセスされた表を監査できます。

監査を行う理由は次のとおりです。

- 現在のアクションのアカウンタビリティの有効化
- アカウンタビリティに基づいた、ユーザー(または侵入者などの非ユーザー)の不適切なアクションの阻止
- 不審なアクティビティの調査、監視および記録
- 監査要件の準拠に対応するため

Oracle Database 12cから、統合監査を使用すると、データベース監査はデフォルトで有効になります。データベース監査は、監査ポリシーを有効にすることにより制御します。ただし、統合監査を使用できるようにするには、自身のデータベースを統合監査に移行する必要があります。

関連項目:

- 統合監査の詳細は、Oracle Databaseセキュリティ・ガイドを参照してください。
- 統合監査への移行方法を学習するには、Oracle Databaseアップグレード・ガイドを参照してください。

監査ポリシー

1つのSQL文を使用して、監査オプションのセットを指定する名前付きの統合監査ポリシーを作成できます。これらのオプションにより、データベース内で監査の対象となるシステム権限、アクションまたはロールを指定できます。

監査ポリシーでは、文ごとに評価できる条件を、1セッションに1回、またはそのデータベース・インスタンスに対して1回、オプションで設定できます。イベントの監査は、適用可能な監査ポリシーの条件の評価結果によって決まります。条件がtrueと評価されれば、データベースでは監査レコードが生成されます。

次の例では、信頼できる端末term1およびterm2からユーザーがログインしなければ、hr.employees表でのアクティビティを監査するポリシーを作成します。

```
CREATE AUDIT POLICY EmployeesTableAudit
  ACTIONS update ON hr.employees, delete ON hr.employees
  WHEN SYS_CONTEXT ("userenv", "hostname") NOT IN
                   ("term1","term2") EVALUATE PER SESSION;
```

次の文により、ユーザーhrおよびhrvpのポリシーが有効になります。

```
AUDIT POLICY EmployeesTableAudit BY hr, hrvp;
```

統合監査ポリシーは、SYSDBA、SYSOPERなどの管理ユーザーを含む、すべてのデータベース・ユーザーに対して適用できます。ただし、監査ポリシーは、データベースがALTER DATABASE OPEN文によってオープンされて初めて読み取られます。したがって、管理ユーザーからの最上位のアクションは常に、データベースがオープンするまでに監査されます。データベースがオープンされた後に、監査ポリシー構成が有効になります。

統合監査が有効な場合、データベースでは自動的に監査設定に対する変更を監査します。また、データベース・インスタンスの起動および停止も監査されます。

関連項目:

監査ポリシーの管理方法を学習するには、『*Oracle Database*セキュリティ・ガイド』を参照してください。

監査管理者ロール

監査を行うには、適切なシステム権限が付与されている必要があります。

Oracle Databaseには、次のシステム提供監査管理者ロールが用意されています。

- AUDIT_ADMIN

  AUDIT_ADMINロールは、データベースの監査設定を管理します。このロールを持つユーザーには、次のことを行う権限があります。

  - 監査ポリシー(ファイングレイン監査ポリシーを含む)の作成、変更および削除
  - ビジネス要件ごとの監査ポリシーの有効化または無効化

- 監査レコードの表示
- 監査証跡の管理およびクリーンアップ

- AUDIT_VIEWER

  AUDIT_VIEWERロールは、データの表示および分析のみを必要とするユーザーのためのものです。このロールを持つユーザーは、監査証跡の内容を表示する権限のみを付与されています。

関連項目:

監査についてさらに学習するには、『Oracle Databaseセキュリティ・ガイド』を参照してください

統合監査証跡

監査レコードは、認可されていないデータ・アクセスの検出および特定に不可欠です。

Oracle Databaseでは、指定したイベントの監査を構成できます。ユーザー・セッション中にイベントが発生した場合、監査レコードが生成されます。

監査証跡は、監査レコードが格納される場所です。統合監査証跡は、Oracle Database 12cで導入され、すべてのタイプの監査から得られた監査レコード用の統合記憶域が用意されています。以前のリリースの従来の監査証跡は、手動で統合監査に移行する必要があります。

監査には、標準およびファイングレイン監査があり、次のイベント(管理ユーザーからのこれらイベントの実行を含む)の監査も含まれます。

- Oracle Data Pump
- SQL*Loaderでのダイレクト・パス・ロード
- Oracle Database Vault
- Oracle Label Security
- Recovery Manager
- Real Application Security

統合監査証跡は、読取り専用で、AUDSYSスキーマに格納されています。デフォルトでは、SYSAUX表領域に、すべてのソースからの監査レコードが格納されます。DBMS_AUDIT_MGMTパッケージを使用して、新規表領域を提供できます。

UNIFIED_AUDIT_TRAILビューでは、監査証跡から監査レコードを取得し、表形式で表示されます。APPLICATION_CONTEXTS列には、構成済のアプリケーション・コンテキスト属性の値が格納されます。AUDIT文を使用して、監査レコード内のコンテキスト属性の値を含めることができます。たとえば、次の文では、userenvネームスペースからMODULEおよびCLIENT_INFO属性を取得します。

```
AUDIT CONTEXT NAMESPACE userenv ATTRIBUTES MODULE, CLIENT_INFO BY hr;
```

監査対象のコンポーネント(Oracle Database Vaultなど)に応じて、追加の統合監査証跡関連のビューを使用できます。

関連項目:

- 統合監査証跡について学習するには、Oracle Databaseセキュリティ・ガイドを参照してください。

- 統合監査を使用するためのデータベースの移行方法を学習するには、Oracle Databaseアップグレード・ガイドを参照してください。
- UNIFIED_AUDIT_TRAILビューについて学習するには、Oracle Databaseリファレンスを参照してください。

Enterprise Managerの監査サポート

Oracle Enterprise Manager (Enterprise Manager)では、ほとんどの監査関連タスクを実行できます。

タスクは次のとおりです。

- 監査の有効化および無効化
- 文およびスキーマ・オブジェクト監査時のオブジェクトの管理

  たとえば、Enterprise Managerを使用して、現行の監査対象の文、権限およびオブジェクトのプロパティを表示および検索できます。
- 監査関連の初期化パラメータの表示および構成
- 監査レポートの表示

関連項目:

Enterprise Managerオンライン・ヘルプ

Oracle Audit Vault and Database Firewall。

Oracle Audit Vault and Database Firewall (Oracle AVDF)は、データベースおよびデータベースの統合監査データ、オペレーティング・システム、ディレクトリに対して第一線の防御を提供します。

SQL文法ベースのエンジンで不正なSQLトラフィックを監視し、データベースに到達する前にブロックします。コンプライアンス・レポートとアラートのために、Oracle AVDFはネットワークからのデータベース・アクティビティ・データを詳細な監査データと統合します。企業のセキュリティ要件を満たすように、監査および監視の制御をカスタマイズできます。

関連項目:

Oracle Audit Vault and Database Firewallなどの追加のセキュリティ・リソースについて学習するには、『Oracle Databaseセキュリティ・ガイド』を参照

高可用性の概要

可用性とは、アプリケーション、サービスおよび機能が必要なときにどの程度使用可能であるかということです。

たとえば、オンライン書店で使用されるOLTPデータベースには、本を購入する顧客がアクセスできる程度の可用性があります。信頼性、リカバリ可能性、タイムリなエラー検出および継続的稼働が高可用性の主な特徴です。

データベース環境において高可用性が重要である理由は、停止時間のコストに関連するためです(停止時間とは、リソースを使用できない時間のことです)。停止時間は、計画停止時間と計画外停止時間のいずれかに分類されます。高可用性環境を設計する場合の主要な課題は、可能性のある停止時間の原因をすべて調査し、対策を講じることです。

高可用性と計画外停止時間

Oracle Databaseでは、あらゆるタイプの予期しない障害により発生する停止時間を防止、許容および短縮する高可用性ソリューションが提供されています。

計画外停止時間は、その原因によって次のように分類されます。

- サイトの障害
- コンピュータ障害
- ストレージ障害
- データ破損
- 人的エラー

サイト障害

サイト障害は、なんらかの原因でアプリケーションのすべてまたは大部分の処理が停止したか、サービスが利用できないほど速度が低下した場合に発生します。

サイト障害は、データ・センターのすべての処理、またはデータ・センターによってサポートされているアプリケーション・サブセットに影響を及ぼす場合があります。サイト障害の例には、サイト全体にわたる広範囲な停電またはネットワーク障害、データ・センターが運用できなくなるほどの自然災害、運営あるいはサイト自体に対する悪意のある攻撃などがあります。

サイト障害に対する最も簡単な保護方法は、RMANを使用してデータベース・バックアップを作成し、オフサイトに保存することです。データベースを別のホストにリストアできます。ただし、この方法は時間がかかる場合があり、バックアップが最新でない可能性もあります。1つ以上のスタンバイ・データベースを持つData Guard環境では、本番サイトに障害が発生した場合でもデータベース・サービスの継続性を維持できます。

関連項目:

- RMANとバックアップおよびリカバリ・ソリューションの詳細は、『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
- スタンバイ・データベースの概要は、Oracle Data Guard概要および管理を参照してください。

コンピュータ障害

コンピュータ障害による停止は、データベースを実行しているシステムが停止またはアクセス不能になることによって、システムが使用不可能になった場合に発生します。

コンピュータ障害の例には、ハードウェア障害やオペレーティング・システム障害などがあります。次の表のOracle機能は、コンピュータ障害を防止、またはコンピュータ障害への対応をサポートします。

表17-1 コンピュータ障害に対する保護

機能	説明	さらに学習するには
エンタープライズ・グリッド	Oracle Real Applications Cluster(Oracle RAC)環境では、Oracle Database は単一の共有データベースに同時にアクセスしながら、クラスタ内の 2 つ以上のシステム上で稼働します。複数のハード	「グリッド・コンピューティングの概要」

機能	説明	さらに学習するには
	ウェア・システムにまたがる単一のデータベース・システムが、アプリケーションでは単一のデータベースとして認識されます。	
Oracle Data Guard	Data Guard を使用すると、スタンバイ・データベースと呼ばれる本番データベースの 1 つ以上のコピーを保持でき、スタンバイ・データベースは、地理的に離れた場所に設置することも、同じデータ・センターに設置することも可能です。停止によりプライマリ・データベースが使用できなくなった場合、Data Guard により任意のスタンバイ・データベースがプライマリ用に切り替えられるため、停止時間が最小限に抑えられます。	『Oracle Data Guard 概要および管理』
Global Data Services	Global Data Services フレームワークにより、データベース・クラウドの構成、メンテナンスおよび監視を自動化し、一元化します。Global Data Services を使用することで、クラウドで提供されるサービスのロード・バランシングおよびフェイルオーバーが可能になります。基本的に、Global Data Services は、Oracle Real Application Clusters (Oracle RAC)が 1 つのデータベースに提供する利点と同種の利点を一連のデータベースに提供します。	Oracle Database Global Data Services 概要および管理ガイド
ファスト・スタート・リカバリ	計画外停止時間の一般的な原因は、システムの故障または障害です。Oracle Database の**ファスト・スタート・リカバリ・テクノロジ**は、データベース・インスタンス・リカバリ時間を自動的にバインドします。	Oracle Database パフォーマンス・チューニング・ガイド

ストレージ障害

ストレージ障害による停止は、一部またはすべてのデータベース・コンテンツを保持しているストレージが停止またはアクセス不能になり、使用できなくなった場合に発生します。ストレージ障害の例には、ディスク・ドライブ障害やストレージ・アレイ障害などがあります。

次の表に、ストレージ障害とOracle Data Guardを示します。

表17-2 ストレージ障害に対するソリューション

ソリューション	説明	さらに学習するには
Oracle Automatic Storage Management(Oracle ASM)	Oracle ASM は、単一インスタンスの Oracle Database および Oracle RAC 構成をサポートする、Oracle データベース・ファイルのボリューム・マネージャおよびファイルシステムです。Oracle ASM はオラクル社が推奨するストレージ管理ソリューションであり、従来のボリューム・マネージャおよびファイル・システムの代替手段となります。	「Oracle Automatic Storage Management (Oracle ASM)」
バックアップおよびリカバリ	Recovery Manager (RMAN)ユーティリティでは、データのバックアップ、以前のバックアップからのリストア、障害発生時までのデータ変更のリカバリなどを実行できます。	バックアップおよびリカバリ

関連項目:

Oracle ASMについてさらに学習するには、Oracle Automatic Storage Management管理者ガイドを参照してください

データ破損

データ破損は、ハードウェア、ソフトウェアまたはネットワーク・コンポーネントによって、破損したデータが読み取られた場合、または書き込まれた場合に発生します。

データ破損の例として、不正なディスク読取りまたは書込みの原因となるボリューム・マネージャのエラーがあります。データ破損はほとんどありませんが、発生した場合にはデータベース、つまりビジネスに壊滅的な影響があります。

データ破損については、Data GuardおよびRecovery Manager以外にも、次の保護方法がOracle Databaseでサポートされています。

- 書込み欠落の問題の解決策

  書込み欠落は、I/Oサブシステムでデータ・ブロックの書込みの完了が認識されたが、実際には書込みが行われていなかった場合に発生します。この後のブロック読取りでは、失効したデータ・ブロックがI/Oサブシステムによって戻されます。このデータ・ブロックを使用してデータベースの他のブロックを更新すると、ブロックが破損する場合があります。Oracle Databaseでの解決策は次のとおりです。

  - スタンバイ・データベースを使用した書込み欠落保護

    Oracle Database 11gで導入された標準の消失書込み保護では、プライマリ・データベースとスタンバイ・データベースの両方でDB_LOST_WRITE_PROTECT初期化パラメータを有効にできます。各データベースは、バッファ・キャッシュ・ブロックの読取りをオンラインREDOログに記録します。スタンバイ・データベースは、管理リカバリ中にREDOを適用する際、対応するブロックを読み取ってSCNをREDOログ内のSCNと比較するため、差異が検出されます。

  - シャドウ表領域を使用した書込み欠落保護

    シャドウ書込み欠落保護について、シャドウ表領域を作成します。シャドウ表領域には、追跡データ・ファイルのデータ・ブロックごとに、SCNなどの短い説明レコードが含まれます。

    シャドウ書込み欠落保護が有効(ALTER DATABASE ENABLE LOST WRITE TRACKING)で、追跡されたデータ・ブロックがデータベースで更新される場合は、データベースによって対応するシャドウ表領域にSCNが書き込まれます。追跡されたデータ・ブロックを読み取る際、データベースではシャドウ・エントリが確認され、それが追跡されたブロックと比較されます。シャドウ・エントリのSCNが追跡されたブロックを上回る場合は、書込み欠落が発生しています。

    シャドウ書込み欠落保護には、スタンバイ・データベースを使用せずに、書込み欠落を検出するという利点があります。また、スタンバイ書込み欠落保護に固有の遅延のため、書込み欠落が検出された際には、そのブロックがすでにデータベースの他の部分を破損させている可能性があります。データの破損を防ぐために、シャドウ書込み欠落保護は書込み欠落が実際に使用される前にその書込み欠落を検出します。

- データ・ブロック破損の検出

  ブロック破損とは、認識されているOracle形式ではないデータ・ブロック、または内容の一貫性が内部で保たれていないデータ・ブロックです。RMANなどのいくつかのデータベース・コンポーネントおよびユーティリティでは、破損ブロックを検出し、V$DATABASE_BLOCK_CORRUPTIONに記録できます。Active Data Guardスタンバイ・データベースを使用する環境であれば、破損は自動的に修復されます。

- データ・リカバリ・アドバイザ

データ・リカバリ・アドバイザは、データ障害を自動的に診断し、適当な修復オプションを提示し、ユーザーのリクエストに基づいて修復を実行するOracleツールです。

- トランザクション・ガードおよびアプリケーション・コンティニュイティ

  データベース・セッションの停止が発生すると、それが計画済か計画外かに関係なく、エンド・ユーザーが作業のステータスを把握できないままになる可能性があります。場合によっては、ユーザーがコミット済トランザクションを再送信し、それが論理的なデータ破損につながる可能性があります。トランザクション・ガードは、トランザクションがコミットされ、完了したどうかを示す保証済のコミット結果をデータベースで保存できるようにするトランザクション冪等性を提供します。アプリケーション・コンティニュイティ(トランザクション・ガードを含む)により、アプリケーションは、リカバリ可能なエラーの後に、データベースに対してトランザクションをリプレイし、トランザクションの中断箇所から続行できるようになります。

関連項目:

- 「トランザクション・ガードの概要」
- RMANとバックアップおよびリカバリ・ソリューションの詳細は、『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

人為的エラー

人為的エラーによる停止は、過失または悪意による操作によって、データベース内のデータが論理的に破損するか、使用不可能になった場合に発生します。人為的エラーによる停止のサービス・レベルへの影響は、影響を受けたデータの量および重要度によって大きく異なる場合があります。

多くの研究で、停止時間の最大の原因は人為的エラーだと指摘されています。Oracle Databaseでは、人為的なエラーを迅速に診断してリカバリするために役立つ、管理者用の強力なツールが提供されています。また、エンド・ユーザーが、管理者の介入なしで問題からのリカバリを実行できる機能も備えられています。

Oracle Databaseを使用する場合、次の方法により人為的エラーから保護することをお薦めします。

- ユーザー・アクセスの制限

  エラーを回避する最善の方法は、データおよびサービスへのユーザー・アクセスを制限することです。Oracle Databaseでは、ユーザー認証によりアプリケーション・データへのアクセスを制御する様々なセキュリティ・ツールが提供されており、管理者は、各ユーザーに対し、それぞれの職務の実行に必要な権限のみを付与できます。

- Oracle Flashbackテクノロジ

  Oracle Flashbackテクノロジは、人為的エラーを修復する一連のOracle Database機能です。Oracle Flashbackでは、人為的エラーを迅速に分析して修復するSQLインタフェースを提供しています。たとえば、次の操作を実行できます。

  - 部分的な損害を対象としたファイングレイン分析と修復
  - 広範囲の損害の迅速な修正
  - 行、トランザクション、表、表領域、およびデータベース・レベルでのリカバリ

- Oracle LogMiner

  Oracle LogMinerは、SQLを使用してオンライン・ファイルの読取り、分析および解釈を行うことができるリレーショナル・ツールです。

関連項目:

- Oracle LogMiner
- 「データベース・セキュリティの概要」
- Oracle Flashback機能についてさらに学習するには、『Oracle Databaseバックアップおよびリカバリ・ユーザーズ・ガイド』および『Oracle Database開発ガイド』を参照してください。
- Oracle LogMinerについてさらに学習するには、『Oracle Databaseユーティリティ』を参照してください。

高可用性と計画停止時間

特に複数のタイムゾーンのユーザーをサポートしているグローバル・エンタープライズでは、計画停止時間は運用に大きな損害を与える可能性があります。このような場合、ルーチン操作、定期的なメンテナンスおよび新規デプロイメントなどの計画的な中断を最小化するようシステムを設計することが重要です。

計画停止時間は、その原因によって次のように分類されます。

- システムおよびデータベースの変更
- データ変更
- アプリケーション変更

システムおよびデータベースの変更

計画的なシステム変更は、日常的および定期的なメンテナンス操作や新規デプロイメントを実行する場合に発生し、データベースの組織データ構造の外部で行われる運用環境の計画的変更もその1つです。

その例としては、CPUやクラスタ・ノード(ノードは、データベース・インスタンスのあるコンピュータ)の追加または削除、システム・ハードウェアまたはソフトウェアのアップグレード、およびシステム・プラットフォームの移行などがあります。

Oracle Databaseでは、計画的なシステムおよびデータベース変更に対するソリューションとして、**リソースの動的プロビジョニング**を提供しています。

- データベースの動的再構成

  Oracle Databaseでは、ハードウェアおよびデータベース構成の動的な変更がサポートされており、SMPサーバーのプロセッサの追加および削除、Oracle ASMを使用したストレージ・アレイの追加および削除などを動的に実行できます。たとえば、Oracle Databaseはオペレーティング・システムを監視して、CPU数の変更を検出します。CPU_COUNT初期化パラメータがデフォルトに設定されている場合、データベースのワークロードは、新規に追加されたプロセッサでも負担されます。

- 自動調整によるメモリー管理

  Oracle Databaseでは、非一元化ポリシーを使用して、SGAおよびPGAの各サブコンポーネントのメモリーを解放および取得します。Oracle Databaseは、オペレーティング・システムにプロンプトを発行し、メモリーを必要とするコンポーネントにメモリーのグラニュルを転送することによってメモリーを自動調整します。

- データファイル、制御ファイル、およびオンラインREDOログ・ファイルの自動配布

  Oracle ASMでは、すべての使用可能なディスクにデータファイル、制御ファイル、およびログ・ファイルが自動配布されるため、それらのファイルのレイアウトが自動化され、簡素化されます。

関連項目:

- 「メモリー管理」
- Oracle ASMについてさらに学習するには、『Oracle Automatic Storage Management管理者ガイド』を参照

データ変更

計画的なデータ変更は、Oracle Databaseオブジェクトの論理構造または物理的な編成が変更された場合に発生します。これらの変更は、主にパフォーマンスまたは管理性を向上させる目的で行われます。例として、表の再定義、表パーティションの追加、および索引の作成または再作成などがあります。

Oracle Databaseでは、オンラインで再編成および再定義することによって、データ変更に伴う停止時間を最小化します。このアーキテクチャを使用すると、データベースのオープン中に、次のタスクを実行できます。

- 表の可用性に大きな影響を与えることなく表構造を変更できる、オンライン表再定義
- 索引の作成、分析および再編成
- 表パーティションの移動

関連項目:

- 「索引と索引構成表」
- パーティションの概要
- オンラインでデータ構造を変更する方法を学習するには、『Oracle Database管理者ガイド』を参照してください

アプリケーション変更

計画的なアプリケーション変更には、データ、スキーマおよびプログラムの変更が含まれる場合があります。これらの変更は、主にパフォーマンス、管理性、および機能性を向上させる目的で行われます。例として、アプリケーションのアップグレードがあります。

Oracle Databaseでは、アプリケーション・データベース・オブジェクトの変更に必要なアプリケーションの停止時間を最小限に抑えるために、次のソリューションがサポートされています。

表17-3 停止時間の最小化のためのソリューション

ソリューション	説明	さらに学習するには
ローリング・データベース・パッチ更新	Oracle Database は、ローリング方式による Oracle RAC システムのノードへのパッチの適用をサポートします。	
ローリング・データベース・リリース・アップグレード	Oracle Database では、ローリング方式によるデータベース・ソフトウェア・アップグレードのインストールおよびパッチ・セットの適用がサポートされており、この方式では、Data Guard SQL Apply および ロ	『*Oracle Database アップグレード・ガイド*』

ソリューション	説明	さらに学習するには
	ジカル・スタンバイ・データベースが使用され、データベースの停止時間はほとんどありません。	
エディションベースの再定義	エディションベースの再定義を使用すると、アプリケーションの使用中にアプリケーションのデータベース・オブジェクトをアップグレードできるため、停止時間を最小限に抑えることや解消することが可能です。Oracle Database では、エディションと呼ばれるプライベート環境でデータベース・オブジェクトを変更(再定義)することにより、このタスクを実行します。	『*Oracle Database 開発ガイド*』
デフォルトが WAIT オプションの DDL	DDL 文には、内部構造の排他ロックが必要です(「DDL ロック」を参照)。以前のリリースでは、ロックを取得できない場合、DDL 文が失敗しました。この問題は、DDL に WAIT オプションを指定することで解決できます。	
無効な状態のトリガーの作成	無効な状態のトリガーを作成でき、コードが正常にコンパイルされることを確認してから、トリガーを有効にできます。	『*Oracle Database PL/SQL 言語リファレンス*』

グリッド・コンピューティングの概要

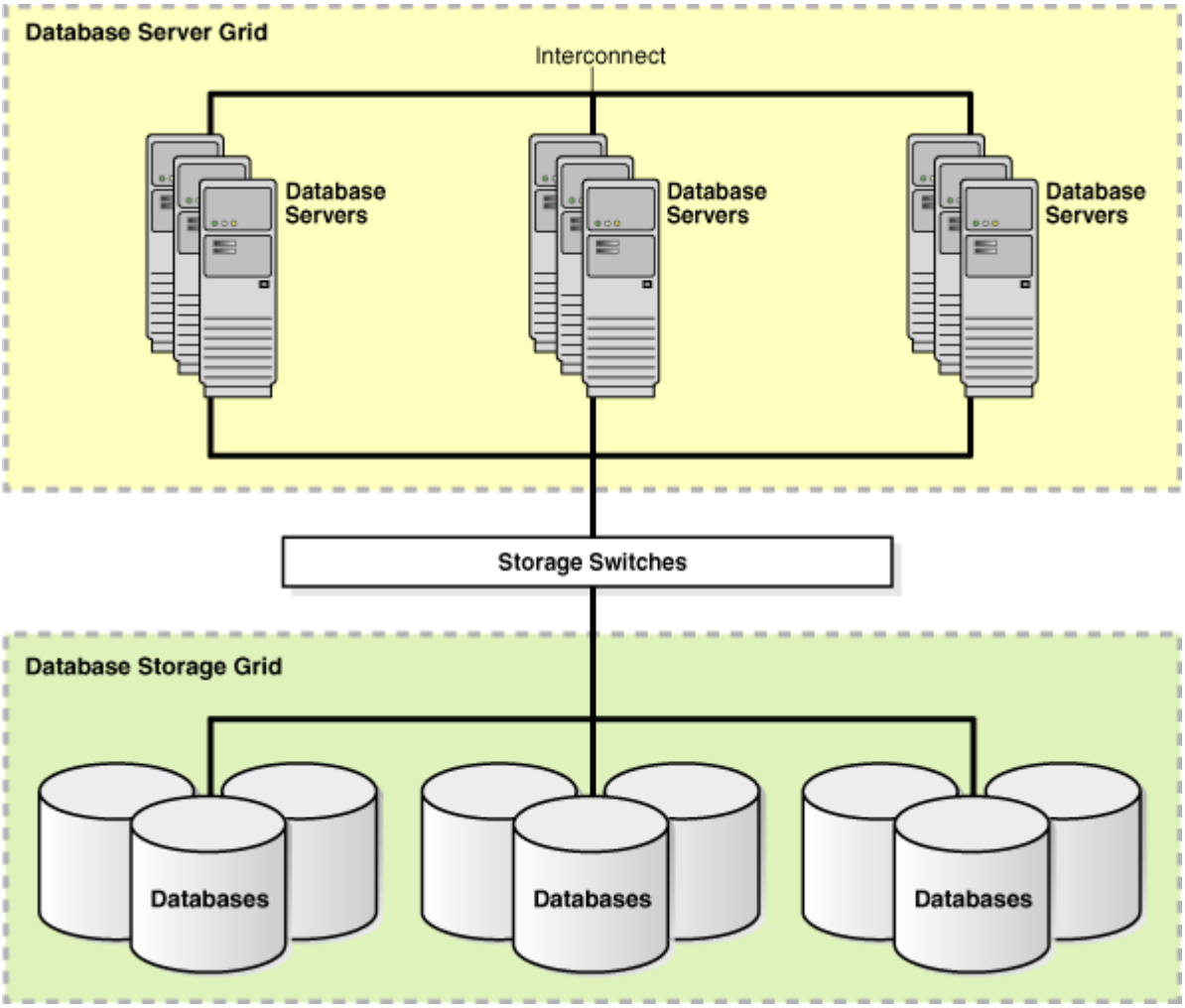
グリッド・コンピューティングとして知られるコンピューティング・アーキテクチャは、すべての企業のコンピューティング・ニーズに応じて、柔軟なオンデマンド・リソースに多くのサーバーおよび記憶域を効果的にプールします。

データベース・サーバー・グリッドは、互いに接続されたコモディティ・サーバーのコレクションで、1つ以上のデータベースを実行します。データベース・ストレージ・グリッドは、相互に組み合され、データベース・サーバー・グリッド内のコンピュータからアクセスされる低コストのモジュラ・ストレージ・アレイの集合です。

データベース・サーバー・グリッドとデータベース記憶域グリッドにより、システム・リソースのプールを構築できます。これらのシステム・リソースは、業務の優先順位に従って動的に割当ておよび割当解除できます。

図17-2に、エンタープライズ・グリッド・コンピューティング環境でのデータベース・サーバー・グリッドおよびデータベース記憶域グリッドを示します。

図17-2 グリッド・コンピューティング環境



関連項目:

標準化機構のグローバル・グリッド・フォーラム(GGF)について学習するには、http://www.gridforum.org/を参照してください。

データベース・サーバー・グリッド

Oracle Real Application Clusters (Oracle RAC)では、複数のインスタンス間でOracle Databaseへのアクセスを共有できます。インスタンスは、インターコネクトを介して接続されます。

Oracle RAC環境では、Oracle Databaseは単一の共有データベースに同時にアクセスしながら、クラスタ内の2つ以上のシステム上で稼働します。Oracle RACでは、複数の低コストのサーバーにまたがる単一のデータベース(アプリケーションでは単一の統合されたデータベース・システムとして認識される)を提供することによって、データベース・サーバー・グリッドを有効にします。

Oracle Clusterwareは、複数のサーバーを1つのサーバーのように操作できるソフトウェアです。各サーバーは、普通のスタンドアロン・サーバーのように見えます。ただし、各サーバーには相互に通信を行うための追加的なプロセスが組み込まれているため、別々のサーバーが1つのサーバーであるかのように連携して動作します。Oracle Clusterwareには、ノード・メンバーシップおよびメッセージ・サービスなど、クラスタの実行に必要なすべての機能が用意されています。

関連項目:

- Oracle RACデータベースの管理方法を学習するには、Oracle Real Application Clusters管理およびデプロイメント・ガイドを参照してください

- Oracle Clusterwareのデプロイ方法および管理方法を学習するには、『Oracle Clusterware管理およびデプロイメント・ガイド』を参照してください。

スケーラビリティ

データベース・サーバー・グリッドでOracle RACを使用すると、容量を増加するためにクラスタにノードを追加できます。

Oracle RACに実装されているキャッシュ・フュージョン・テクノロジにより、アプリケーションを変更しなくても容量を拡張できます。これにより、システムを段階的に拡張してコストを節約でき、小規模な単一ノード・システムを大規模なシステムに置き換える必要がなくなります。

既存のシステムを大規模なノードに置き換えるかわりに、ノードを段階的にクラスタに追加できます。グリッド・プラグ・アンド・プレイを使用すると、クラスタのノードの追加と削除が簡素化されるため、動的にプロビジョニングされた環境でクラスタをデプロイしやすくなります。また、グリッド・プラグ・アンド・プレイでは、データベースとサービスを位置に依存しない方法で管理することもできます。SCANを使用すると、クライアントはグリッド内の位置とは関係なくデータベース・サービスに接続できるようになります。

関連項目:

- キャッシュ・フュージョンについてさらに学習するには、Oracle Real Application Clusters管理およびデプロイメント・ガイドを参照してください。
- グリッド・ネーミング・サービスを使用してグリッド・プラグ・アンド・プレイを有効にする方法を学習するには、ご使用のプラットフォームのOracle Databaseインストレーション・ガイドを参照してください。
- グリッド・プラグ・アンド・プレイを有効にするためのグリッド・ネーミング・サービスの構成の詳細は、Oracle Clusterware管理およびデプロイメント・ガイドを参照してください。

フォルト・トレランス

高可用性アーキテクチャにおける**フォルト・トレランス**は、アーキテクチャ内のコンポーネント障害に対する保護機能です。

Oracle RACアーキテクチャの主な利点は、複数のノードによって実現される本質的なフォルト・トレランスです。物理ノードは独立して動作しているため、1つ以上のノードに障害が発生してもクラスタ内の他のノードには影響しません。

フェイルオーバーはグリッド上の任意のノードで発生します。極端な場合、1つのノードを除くすべてのノードが停止しても、Oracle RACシステムはデータベース・アクセスを提供し続けます。このアーキテクチャでは、メンテナンスのためにノードのグループを透過的にオンラインまたはオフラインにすると同時に、クラスタの残りの部分によってデータベース・アクセスを提供し続けることができます。

Oracle RACには、Oracleクライアントおよび接続プールとの統合が組み込まれています。この機能により、接続を終了する障害がプールを介してアプリケーションに即座に通知されます。アプリケーションは、TCPタイムアウトを待機することなく、適切なリカバリ・アクションを即座に実行できます。Oracle RACでは、リスナーとOracleクライアントおよび接続プールを統合することによって、最適なアプリケーション・スループットを実現します。Oracle RACは、トランザクション時の負荷に基づいて、クラスタ・ワークロードのバランスを調整します。

関連項目:

- 「データベース常駐接続プーリング」

- 自動ワークロード管理についてさらに学習するには、Oracle Real Application Clusters管理およびデプロイメント・ガイドを参照してください。

サービス

Oracle RACはサービスをサポートしており、これによりデータベースのワークロードをグループ化し、サービスを提供するために割り当てられている最適なインスタンスに処理をルーティングできます。

サービスとは、共通の属性、パフォーマンスしきい値、および優先順位を持つアプリケーションのワークロードです。ビジネス・ポリシーを定義してこれらのサービスに適用し、ピーク処理時用のノードの割当てや、サービス障害の自動処理などのタスクを実行します。サービスを使用してシステム・リソースを必要な場所と時間に応じて適用することにより、ビジネス上の目標を達成します。

サービスは、データベース・リソース・マネージャに統合されており、データベース・リソース・マネージャでは、インスタンス内のサービスで使用可能なリソースを制限できます。また、特定のインスタンスを使用するかわりに、サービスを使用してOracle Schedulerジョブを実行することもできます。

関連項目:

- データベース・リソース・マネージャ
- データベース・リソース・マネージャおよびOracle Schedulerについて学習するには、『Oracle Database管理者ガイド』を参照

Oracle Flex Clusters

Oracle Database 12cから、Oracle ClusterwareおよびOracle Real Application Clustersを大規模クラスタで構成できるようになりました。

これらの大規模クラスタは、Oracle Flex Clustersと呼ばれ、ハブ・アンドスポーク・アーキテクチャに配置された2種類のノード(ハブ・ノードおよびリーフ・ノード)を含んでいます。ハブ・ノードは、しっかり接続されており、共有記憶域に直接アクセスでき、1つ以上のリーフ・ノードのアンカーの役割を果します。リーフ・ノードは、ハブ・ノードとゆるやかに接続されていて、共有記憶域には直接アクセスできない可能性があります。

関連項目:

- Oracle Flex Clustersについてさらに学習するには、Oracle Clusterware管理およびデプロイメント・ガイドを参照してください。
- Oracle Flex Clusterの配置についてさらに学習するには、Oracle Grid Infrastructureインストレーション・ガイドを参照してください。

データベース記憶域グリッド

DBAまたはストレージ管理者は、Oracle ASMインタフェースを使用してデータベース記憶域グリッド内のディスクを指定でき、指定されたディスクは、すべてのサーバーおよび記憶域プラットフォームでOracle ASMにより管理されます。Oracle ASMは、ディスク領域をパーティションに区切り、データをOracle ASMに提供されたディスク全体に均等に配分します。また、Oracle

ASMは、データベース記憶域グリッドのストレージ・アレイのディスクが追加または削除されると、データを自動的に再配分します。

関連項目:

- 「Oracle Automatic Storage Management (Oracle ASM)」
- クラスタOracle ASMの詳細は、『*Oracle Automatic Storage Management*管理者ガイド』を参照してください。

データ・ウェアハウスとビジネス・インテリジェンスの概要

データ・ウェアハウスは、トランザクション処理ではなく問合せおよび分析用に設計されたリレーショナル・データベースです。

たとえば、データ・ウェアハウスでは、株価の履歴や所得税の記録を追跡できます。通常、ウェアハウスには履歴トランザクション・データから導出されたデータが格納されますが、他のソースからのデータも格納できます。

データ・ウェアハウス環境には、リレーショナル・データベースの他に、様々なツールがあります。典型的な環境には、ETLソリューション、OLAPエンジン、クライアント分析ツール、データを収集してユーザーに配信するその他のアプリケーションがあります。

データ・ウェアハウスとOLTP

データ・ウェアハウスを導入する際は、William Inmon氏が提唱するデータ・ウェアハウスの次の特性を十分に理解する必要があります。

特性は次のとおりです。脚注1

- サブジェクト指向

  データ・ウェアハウスでは、売上などの主題別にデータベースを定義できます。

- 統合

  データ・ウェアハウスには、様々なソースからのデータを一貫性のある形式で格納する必要があります。また、名前の競合や単位間の不一致などの問題を解決する必要があります。この目標が達成されて初めて、データ・ウェアハウスは統合されたことになります。

- 恒常的

  ウェアハウスの目的は、発生した事柄を分析できるようにすることです。このため、ウェアハウスに格納されたデータは変更されません。

- 時系列

  データ・ウェアハウスは時間経過による変化に焦点を当てています。

データ・ウェアハウスとOLTPデータベースでは、要件が異なります。たとえば、データ・ウェアハウスでは、ビジネスの傾向を把握するために、大量のデータを保持する必要があります。それとは対照的に、OLTPシステムでは、パフォーマンスを維持するために、履歴データを定期的にアーカイブに移動する必要があります。表17-4に、データ・ウェアハウスとOLTPの相違点を示します。

表17-4 データ・ウェアハウスとOLTPシステム

特徴	データ・ウェアハウス	OLTP
ワークロード	非定型問合せに適応するように設計されています。データ・ウェアハウスのワークロードは事前には	事前定義済の操作のみをサポートします。これらの操作のみをサポートするように、アプリケーション

特徴	データ・ウェアハウス	OLTP
	わからない場合があり、データ・ウェアハウスは様々な問合せを適切に実行できるように最適化する必要があります。	の特別なチューニングや設計が必要になる場合があります。
データ修正	バルク・データ修正テクニックを使用して ETL プロセスにより定期的に更新されます。データ・ウェアハウスのエンド・ユーザーがデータベースを直接更新することはありません。	エンド・ユーザーにより定期的に発行される個々の DML 文によって行われます。OLTP データベースは常に最新であり、各ビジネス・トランザクションの現在の状態が反映されます。
スキーマ設計	全体または一部が非正規化されたスキーマ(スター・スキーマなど)を使用して、問合せパフォーマンスが最適化されます。	完全に正規化されたスキーマを使用して、DML パフォーマンスが最適化され、データの整合性が保証されます。
典型的な操作	典型的な問合せでは、数千または数百万の行がスキャンされます。たとえば、すべての顧客についての先月の総売上を要求できます。	典型的な操作は、少数のレコードにのみアクセスします。たとえば、1 人の顧客の現行の注文を取得できます。
履歴データ	履歴分析をサポートするために、数か月または数年分のデータが格納されます。	数週間または数か月分のデータしか格納されません。現行のトランザクションの要件を適切に満たすために必要な履歴データが保持されます。

関連項目:

- データベース・ウェアハウスの詳細は、Oracle Databaseデータ・ウェアハウス・ガイドを参照してください。
- OLTPシステムの詳細は、Oracle Database VLDBおよびパーティショニング・ガイドを参照してください。

データ・ウェアハウスのアーキテクチャ

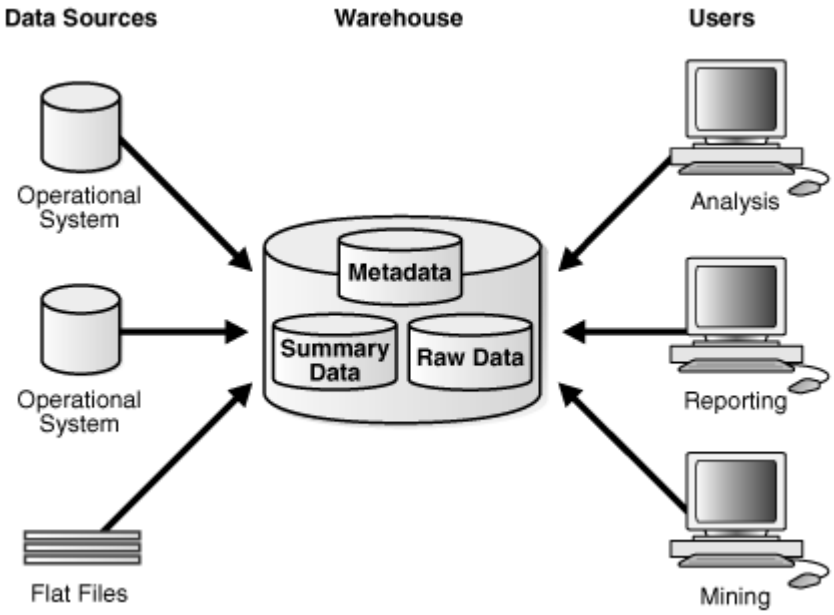
データ・ウェアハウスとそのアーキテクチャは、ビジネス要件によって異なります。

データ・ウェアハウスのアーキテクチャ(基本)

単純なデータ・ウェアハウスのアーキテクチャでは、エンド・ユーザーは、複数のソース・システムからデータ・ウェアハウスに転送されたデータに直接アクセスします。

次の図に、アーキテクチャのサンプルを示します。

図17-3 データ・ウェアハウスのアーキテクチャ



前の図には、従来型OLTPシステムのメタデータと生データの他に、サマリー・データが示されています。サマリーは集計ビューで、高コストの結合および集計操作を事前に計算して結果を表に格納することで、問合せのパフォーマンスが向上します。たとえば、サマリー表に地域別、製品別の売上合計を含めることができます。サマリーは、**マテリアライズド・ビュー**とも呼ばれます。

関連項目:

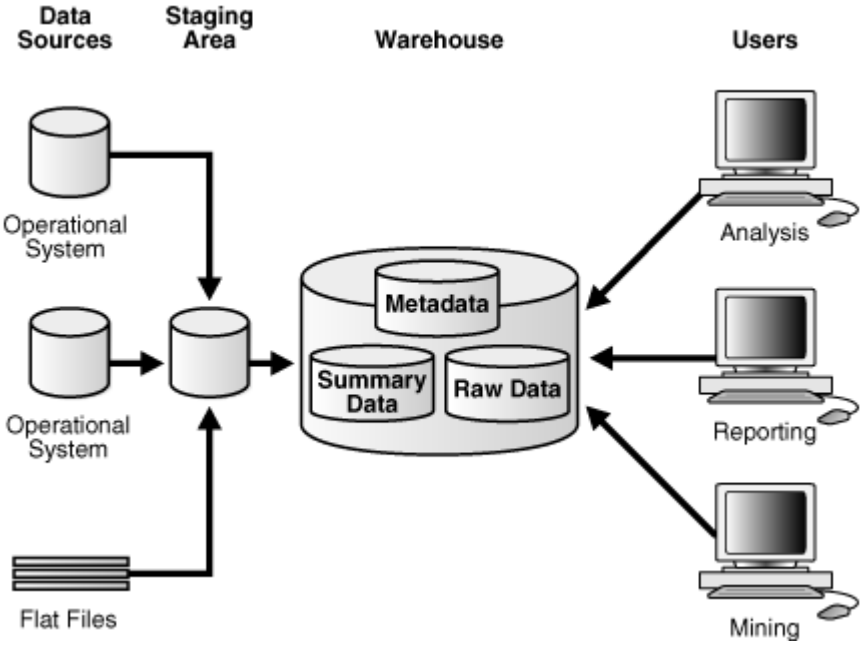
基本マテリアライズド・ビューについて学習するには、『Oracle Databaseデータ・ウェアハウス・ガイド』を参照してください

データ・ウェアハウスのアーキテクチャ(ステージング領域あり)

一部のデータ・ウェアハウスでは、ウェアハウスに入れる前にデータを事前に処理する場所である**ステージング領域**が使用されます。ステージング領域により、サマリーの作成とウェアハウス管理のタスクが簡素化されます。

次の図に、ステージング領域を示します。

図17-4 データ・ウェアハウスのアーキテクチャ(ステージング領域あり)



関連項目:

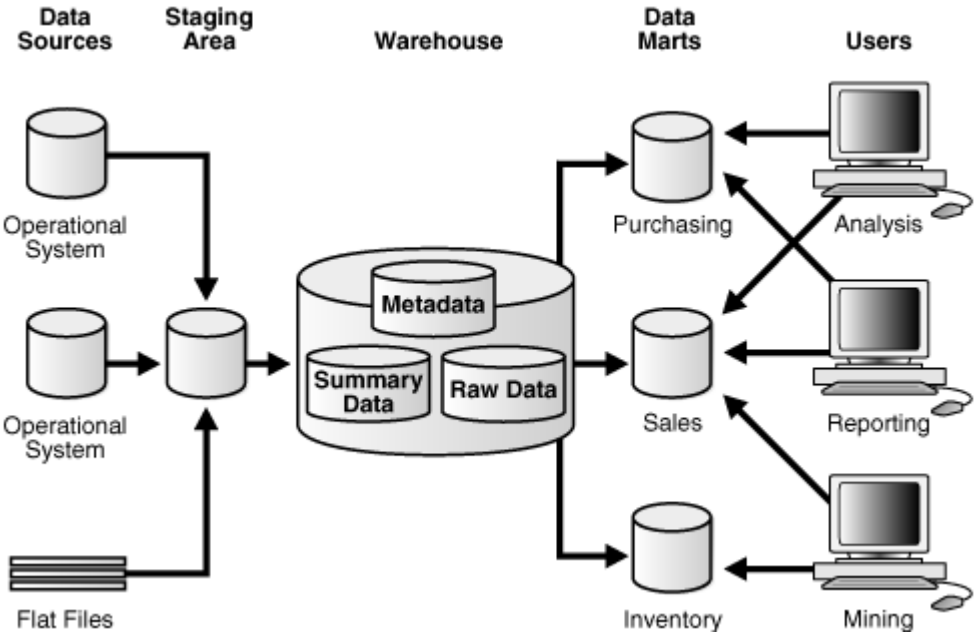
異なる転送メカニズムについて学習するには、Oracle Databaseデータ・ウェアハウス・ガイドを参照してください

データ・ウェアハウスのアーキテクチャ(ステージング領域およびデータ・マートあり)

ウェアハウス・アーキテクチャを組織内の様々なグループ向けにカスタマイズできます。そのためには、ウェアハウス内のデータを、特定の業務またはプロジェクト用に設計された独立したデータベースであるデータ・マートに転送します。通常、データ・マートには多くのサマリー表が含まれています。

図17-5では、購買、売上および在庫情報が別々のデータ・マートに分離されています。財務アナリストはデータ・マートに対して購買と売上の履歴情報を問い合せることができます。

図17-5 データ・ウェアハウスのアーキテクチャ(ステージング領域およびデータ・マートあり)



関連項目:

変換メカニズムについて学習するには、Oracle Databaseデータ・ウェアハウス・ガイドを参照してください

抽出、変換、ロード(ETL)の概要

ソース・システムからデータを抽出してウェアハウスに取り込む処理は、一般にETLと呼ばれ、抽出、変換、ロードを表します。ETLは、詳細に定義された3つのステップではなく、広範囲なプロセスを指します。

典型的なシナリオでは、1つ以上の業務系システムからデータを抽出し、処理用にターゲット・システムまたは中間システムに物理的に転送します。転送方法によっては、このプロセスでなんらかの変換処理が行われる場合もあります。たとえば、ゲートウェイを介してリモート・ターゲットに直接アクセスするSQL文では、SELECT文の一部として2つの列を連結できます。

Oracle Database自体は、ETLツールではありません。ただし、Oracle Databaseでは、ETLツールやカスタマイズされたETLソリューションで使用できる豊富な機能が提供されています。Oracle Databaseが提供するETL機能は、次のとおりです。

- トランスポータブル表領域

  異なるコンピュータ・アーキテクチャやオペレーティング・システム間で表領域を転送することもできます。トランスポータブル表領域は、2つのOracleデータベース間で大量のデータを移動する場合に最も高速な方法です。

- テーブル・ファンクション

テーブル・ファンクションは、行の集合(ネストした表またはVARRAY)を戻すユーザー定義のPL/SQLファンクションです。テーブル・ファンクションは、出力として行セットを生成でき、行セットを入力として受け入れることができます。テーブル・ファンクションでは、PL/SQL、CまたはJavaで実装されている変換のパイプライン実行とパラレル実行がサポートされており、中間のステージング表を使用する必要はありません。

- 外部表

  外部表により、最初に外部データをデータベースにロードしなくても、パラレルで直接結合できます。このため、外部表を使用すると、ロード・フェーズと変換フェーズをパイプライン処理できます。

- 表の圧縮

  ディスク使用量とメモリー使用量を減少させるために、表とパーティション表を圧縮形式で格納できます。通常は、表の圧縮を使用すると、読取り専用操作のパフォーマンスが向上し、問合せ実行も速くなります。

関連項目:

- 表の圧縮
- 「外部表の概要」
- ETLの概要は、『*Oracle Database*データ・ウェアハウス・ガイド』を参照してください。
- 『*Oracle Database*管理者ガイド』

ビジネス・インテリジェンス

ビジネス・インテリジェンスは、ビジネスにおける意志決定を支援するために組織内の情報を分析することです。

分析アプリケーションおよびビジネス・インテリジェンスは、階層のドリルアップやドリルダウン、および集計値の比較によって占められています。Oracle Databaseでは、このような操作をサポートするテクノロジを提供しています。

分析SQL

Oracle Databaseには、分析操作を実行するための多数のSQL操作が導入されています。たとえば、ランキング、移動平均、累計、対比レポート、期間対比などです。

Oracle Databaseでは、次の形式による分析SQLがサポートされています。

表17-5 分析SQL

分析SQLのタイプ	説明	さらに学習するには
集計用 SQL	COUNT などの集計ファンクションは、行のグループに基づいて、結果行を 1 つ返します。集計は、データ・ウェアハウスの基盤です。ウェアハウスでの集計効率を改善するために、データベースには問合せとレポートを容易かつ高速にする GROUP BY 句の拡張機能が用意されています。	集計について学習するには、『Oracle Database データ・ウェアハウス・ガイド』を参照

分析SQLのタイプ	説明	さらに学習するには
分析用 SQL	MAX などの分析ファンクションは行のグループ(ウィンドウと呼ばれる)を集計し、結果セットとして複数の行を返します。Oracle は、分析 SQL 関数ファミリを使用した拡張 SQL 分析処理機能を備えています。たとえば、これらの分析関数を使用すると、ランキングやパーセンタイル、変動ウィンドウを計算できます。	分析用およびレポート用 SQL について学習するには、『Oracle Database データ・ウェアハウス・ガイド』を参照
モデル化用 SQL	MODEL 句を使用すると、問合せ結果から多次元配列を作成し、この配列にルールを適用して新規の値を計算できます。たとえば、売上ビューのデータを国別にパーティション化し、複数のルールによって定義されたモデル計算を国別に実行できます。ある製品の 2006 年と 2007 年の売上の合計として 2008 年の売上を計算するルールを定義できます。	SQL のモデル化について学習するには、『Oracle Database データ・ウェアハウス・ガイド』を参照

関連項目:

SQL関数について学習するには、『Oracle Database SQL言語リファレンス』を参照してください。

分析ビュー

分析ビューは、データ・セットの内容を拡張し、ビジネス・インテリジェンス・アプリケーションの開発を容易にします。

分析ビューには、次の特性があります。

- データは、階層およびディメンションの概念を使用して編成されます。
- 分析ビューには、結合、集計および計測の計算ルールが埋め込まれています。
- SQL DDLを使用すると、データベース内の既存の表、ビューおよびその他のオブジェクトの上に階層化できます。
- 簡単なSQLを使用して問い合せることができます。

関連項目:

分析ビューの概要は、Oracle Databaseデータ・ウェアハウス・ガイドを参照してください。

OLAP

Oracleオンライン分析処理(**OLAP**)では、複数のディメンション間でのデータ分析において、システム固有の多次元記憶域を提供し、レスポンス時間の短縮を実現します。OLAPを使用すると、アナリストは複雑な反復問合せの結果を対話形式のセッションで迅速に取得できます。

Oracle OLAPには、主に次の特性があります。

- Oracle OLAPはデータベースに統合されているため、標準的なSQLの管理、問合せおよびレポート・ツールを使用できます。
- OLAPエンジンは、Oracle Databaseのカーネル内で動作します。
- ディメンション・オブジェクトは、システム固有の多次元形式でOracle Databaseに格納されます。
- キューブおよび他のディメンション・オブジェクトは、Oracleデータ・ディクショナリ内に存在する最上位のデータ・オブジェクトです。
- データ・セキュリティは、Oracle Databaseユーザーおよびロールの権限の付与と取消しにより、標準的な方法で管理されます。

Oracle OLAPの特長はシンプルさで、1つのデータベース、標準的な管理とセキュリティ、および標準的なインタフェースと開発ツールで構成されています。

関連項目:

- 「OLAP」
- ディメンションの概要
- Oracle OLAPの概要は、『*Oracle OLAPユーザーズ・ガイド*』を参照してください。

Oracle Advanced Analytics

Oracle Advanced Analyticsオプションは、Oracle Databaseを大きなデータ分析用の包括的な拡張分析プラットフォームに拡張します。

Oracle Advanced Analyticsでは、予測分析、データ・マイング、テキスト・マイニング、統計分析、拡張数値計算、およびデータベース内の対話型グラフィックを提供します。Oracle Advanced Analyticsには、次のコンポーネントがあります。

- Oracle Data Mining
- Oracle R Enterprise

Oracle Data Mining

ビジネス・インテリジェンスにおける**データ・マイニング**とは、高度な数学的アルゴリズムを使用してデータをセグメント化し、イベントが将来発生する確率を計算することです。

通常、データ・マイニングは、コール・センター、ATM、E-Businessリレーショナル管理(ERM)およびビジネス・プランニングなどで使用されます。Oracle Data Minerを使用すると、データ分析者は、データを迅速に分析し、最良の顧客にターゲットを絞り、不正行為と戦い、ビジネスの競争力向上に役立てられる重要な相互関係やパターンを見つけることができます。

Oracle Data Miningには、高パフォーマンスのインデータベースのモデル作成およびモデル配置用のネイティブSQLファンクションとして実行される、データ・マイニング・アルゴリズムが備わっています。Oracle Data Miningでは、表、ビュー、スター・スキーマ、トランザクション・データおよび非構造化データのマイニングが可能です。

Oracle Data Miningでは、PL/SQL APIおよびモデル・スコアリング用のSQLファンクションがサポートされます。これにより、Oracle Databaseはアプリケーション開発者に、データ・マイニングをデータベース・アプリケーションとシームレスに統合するためのインフラストラクチャを提供します。

SQL Developer の拡張機能であるOracle Data Minerには、Oracle Data Mining用のGUIが用意されています。

関連項目:

Oracle Data Mining概要

Oracle R Enterprise

Rは、統計計算およびグラフィック用のオープンソース言語および環境です。Oracle R Enterpriseにより、Rで企業および大量のデータに対処できるようになります。

大量のデータを必要とする問題用に設計されたOracle R Enterpriseは、RをOracle Databaseと統合します。Oracle Databaseに格納されたデータでの統計およびグラフィック分析に、Rのコマンドおよびスクリプトを実行できます。また、データベースの並列性とスケーラビリティを利用してデータ分析を自動化するRスクリプトの開発、改良およびデプロイができます。データ分析者は、Rパッケージを実行し、分析アプリケーション用のRスクリプトを1回の動作で開発することができ、SQLを学習する必要がありません。

関連項目:

『Oracle R Enterpriseユーザーズ・ガイド』

Oracleにおける情報統合の概要

組織が成長するにつれて、複数のデータベースおよびアプリケーション間で情報を共有できることに対する重要性が高まってきます。

情報を共有するには、次の基本的な方法があります。

- 連結

  情報を1つのデータベースに連結でき、これによって、それ以上の統合は必要なくなります。Oracle RAC、グリッド・コンピューティング、マルチテナント・アーキテクチャおよびOracle VPDにより、情報を1つのデータベースに連結できます。

- 統合

  情報は分散した状態にしておき、この情報を統合するツールを提供できます。これによって、情報が1つの仮想データベース内にあるかのように見せることができます。

- 共有

  情報を共有することで、複数のデータ・ストアおよびアプリケーションで情報を保持できます。

この項では、主に情報を統合および共有するためのOracleソリューションについて説明します。

統合されたアクセス

統合されたアクセスの基本となるのは**分散環境**で、これは、相互にシームレスに通信する様々なシステムで構成されるネットワークです。

分散環境では、各システムをノードと呼びます。ユーザーが直接接続するシステムを**ローカル・システム**と呼びます。このユーザーがアクセスする他のシステムを**リモート・システム**と呼びます。

分散環境では、アプリケーションからローカル・システムやリモート・システムのデータにアクセスして相互に交換できます。すべてのデータに同時にアクセスして変更できます。

分散SQL

分散SQLを使用すると、複数のデータベース間に分散されたデータに同時にアクセスして更新できます。Oracle分散データベース・システムはユーザーに対して透過的であるため、単一のOracle Databaseであるかのように見せることができます。

分散SQLには、分散問合せおよび分散トランザクションが含まれます。Oracleの分散データベース・アーキテクチャによって、問合せおよびトランザクションが透過的になります。たとえば、標準DML文の動作は、非分散データベース環境の場合と同じです。また、アプリケーションでは標準SQL文COMMIT、SAVEPOINTおよびROLLBACKを使用してトランザクションを制御します。

関連項目:

- 「分散トランザクションの概要」
- 分散トランザクションの管理方法を学習するには、『Oracle Database管理者ガイド』を参照してください

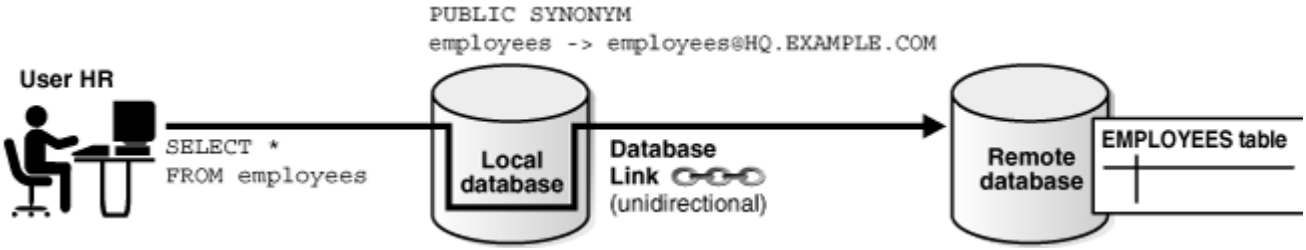
データベース・リンク

データベース・リンクは、2つの物理データベースを結ぶ接続で、これにより、クライアントは1つの論理データベースとして物理データベースにアクセスできます。

Oracle Databaseは、データベース・リンクを使用して、あるデータベース上のユーザーがリモート・データベース内のオブジェクトにアクセスできるようにします。ローカル・ユーザーは、リモート・データベースのユーザーでなくても、リモート・データベースへのリンクにアクセスできます。

図17-6の例では、ユーザーhrが、hq.example.comというグローバル名を持つリモート・データベース上にあるemployees表にアクセスしています。employeesシノニムにより、リモート・スキーマ・オブジェクトのアイデンティティおよび位置は意識されなくなります。

図17-6 データベース・リンク



関連項目:

データベース・リンクについて学習するには、Oracle Database管理者ガイドを参照してください

情報の共有

どのような統合であっても、その中心は社内のアプリケーション間でデータを共有することです。

Oracle GoldenGate

Oracle GoldenGateは、非同期のログベースのリアルタイム・データ・レプリケーション製品です。

Oracle GoldenGateは、異機種間データベース、ハードウェア、オペレーティング・システム環境全体で、大量のトランザクション・データを、影響を最小限に抑えてリアル・タイムで移動します。この製品は、次の理由でリアルタイムの情報アクセスおよび可用性を最適化します。

- Oracle Databaseおよび非Oracleデータベースの異機種混合を含むレプリケーションをサポートします。
- 重要なシステムの継続的な可用性を維持し、その結果、計画メンテナンス中の停止時間を最小限にします。
- 企業内でリアルタイムのデータ統合を可能にします。
- シャード表でシャード間の双方向レプリケーションを自動的に構成します。

典型的な環境として、取得、ポンプおよび配信プロセスがあります。各プロセスは、Oracleデータベースおよび非Oracleデータベースの両方を含む、ほとんどの一般的なオペレーティング・システムおよびデータベースで実行できます。データの一部またはすべてをレプリケートできます。これらのプロセス内のデータは、異機種環境だけでなく、異なるデータベース・スキーマでも操作できます。

Oracle GoldenGateでは、マルチマスター・レプリケーション、ハブ・アンド・スポーク方式のデプロイメント、データ統合およびデータ変換をサポートします。したがって、Oracle GoldenGateにより、クリティカルなシステムの24時間365日の稼働が可能になり、関連データを企業内に分散させて意思決定を最適化できます。

関連項目:

- Oracle Shardingの使用方法を学習するには、Oracle Shardingの使用を参照してください
- http://www.oracle.com/technetwork/middleware/goldengate/documentation/index.html

Oracle Database Advanced Queuing(AQ)

アドバンスト・キューイング(AQ)は、堅牢で豊富な機能を持つメッセージ・キューイング・システムであり、Oracle Databaseに統合されています。

組織内の異なるシステム間で通信を行う必要がある場合は、メッセージング環境によって、これらのシステム間で重要な情報を転送する、標準的で信頼性の高い方法が提供されます。

ユースケースの例として、本社のOracleデータベースに注文を入力するとします。注文が入力されると、AQにより、注文IDと注文日がウェアハウスのデータベースに送信されます。これらのメッセージにより、ウェアハウスの従業員は注文があったことを知り、商品を梱包および出荷できます。

メッセージのキューイングとデキューイング

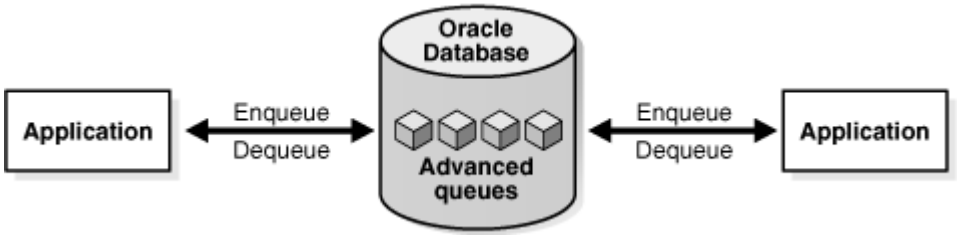
アドバンスト・キューイングでは、ユーザー・メッセージがキューと呼ばれる抽象的な記憶単位に保存されます。

エンキューは、プロデューサがメッセージをキューに入れるプロセスです。デキューは、コンシューマがキューからメッセージを取得するプロセスです。

明示的デキューがサポートされるため、開発者はXStreamおよびOracle GoldenGateを使用してメッセージを確実に交換できます。また、Oracle GoldenGateの変更取得および伝播機能を活用することで、変更をアプリケーションに通知できます。

図17-7では、サンプル・アプリケーションがアドバンスト・キューイングを介して、明示的にメッセージをエンキューおよびデキューしており、これにより、異なるメッセージ・システムを使用しているパートナとも情報を共有できます。エンキューした後、パートナのアプリケーションにデキューされる前に、メッセージを変換および伝播することもできます。

図17-7 Oracle Message Queuing



Oracle Database Advanced Queuingの機能

Oracle Database Advanced Queuing (AQ)では、メッセージ・キューイング・システムの標準的な機能がすべてサポートされています。

次の機能があります。

- 非同期によるアプリケーションの統合

  Oracle Database AQでは、複数の方法でメッセージをエンキューできます。メッセージは、取得プロセスまたは同期取得が暗黙的に取得するか、アプリケーションやユーザーが明示的に取得できます。

- 拡張可能な統合アーキテクチャ

  多くのアプリケーションは、Oracle Databaseをハブとする分散ハブ・アンド・スポーク・モデルを使用して統合されます。Oracleデータベース上の分散アプリケーションは、同じハブのキューと通信します。複数のアプリケーションが同じキューを共有できるので、サポートするアプリケーション数を増やすためにキューを追加する必要がありません。

- 異機種間アプリケーションの統合

  Oracle Database AQは、アプリケーションにOracleタイプ・システムの全機能を提供します。これには、スカラー・データ型、継承を伴うOracle Databaseオブジェクト型、XMLデータ用の追加演算子を持つXMLType、およびANYDATAのサポートが含まれます。

- 従来型アプリケーションの統合

  Oracle Messaging Gatewayは、Oracle DatabaseアプリケーションをWebSphere MQやTibcoなどの他のメッセージ・キューイング・システムと統合します。

- 標準ベースAPIのサポート

  Oracle Database AQは、業界標準のAPIであるSQL、JMSおよびSOAPをサポートしています。SQLを使用して行われた変更は、メッセージとして自動的に取得されます。

関連項目:

Oracle Databaseアドバンスト・キューイング・ユーザーズ・ガイド

脚注の凡例

脚注1:

『*Building the Data Warehouse*』(1996年John Wiley and Sons社)。

18 データベース管理者の概念

この部では、データベース管理者の業務、ツールおよび不可欠な知識について説明します。

この章の構成は、次のとおりです。

- データベース管理者の業務
- データベース管理者向けのツール
- データベース管理者のためのトピック

データベース管理者の業務

データベース管理者(DBA)の主要な役割は、企業データをユーザーに対して利用可能にすることです。

DBAは、アプリケーションがデータベースを効率よく使用できるよう、開発者と緊密に連携する必要があり、また、物理リソースを適切に保ち、効率よく使用できるよう、システム管理者と緊密に連携する必要があります。

Oracle DBAは、Oracle Databaseアーキテクチャおよびデータベースの機能を理解している必要があります。DBAは、次のタスクを実行します。

- Oracle Databaseソフトウェアのインストール、アップグレードおよびパッチ適用
- 要件の確認、論理設計(概念モデル)の作成および物理データベース設計を含む、データベースの設計
- Oracle Databaseの作成
- バックアップやリカバリ計画の開発およびテスト、Oracle Databaseの定期バックアップ、障害発生時のリカバリ
- クライアントがデータベースに接続するためのネットワーク環境の構成
- データベースの起動および停止
- データベースの記憶域の管理
- ユーザーおよびセキュリティの管理
- 表、索引、ビューなどのデータベース・オブジェクトの管理
- データベースのパフォーマンスの監視およびチューニング
- 重大なデータベース・エラーが発生した場合の調査、データの収集、Oracleサポート・サービスへの報告
- 新しいデータベース機能の評価およびテスト

ユーザーのタイプ、ロール、および責任は、データベース環境によって決まります。小規模なデータベースでは、DBAが1人の場合があります。一方、非常に大規模なデータベースでは、セキュリティ管理者、バックアップ・オペレータ、アプリケーション管理者など、複数のスペシャリストがDBAの仕事を分担する場合があります。

関連項目:

- DBAタスクの概要は、『Oracle Database 2日でデータベース管理者』を参照
- DBAの概念とタスクの詳細は、『Oracle Database管理者ガイド』を参照

データベース管理者向けのツール

Oracleでは、データベースを管理するためのいくつかのツールを提供しています。

この項では、一般的に使用される次のツールについて説明します。

- Oracle Enterprise Manager
- SQL*Plus
- データベースのインストールおよび構成ツール
- Oracle Netの構成および管理ツール
- データの移動および分析ツール

Oracle Enterprise Manager

Oracle Enterprise Manager (Enterprise Manager)は、Webベースのシステム管理ツールで、Oracleデータベース、Exadataデータベース・マシン、Fusion Middleware、Oracleアプリケーション、サーバー、記憶域、非Oracleハードウェアおよびソフトウェアを管理します。

Oracle Enterprise Manager Cloud Control。

Oracle Enterprise Manager Cloud Control (Cloud Control)は、Webベースのインタフェースで、これにより管理者は、Oracleテクノロジ・スタックおよび非Oracleコンポーネント全体を完全に監視できます。

高速アプリケーション通知(FAN)のコンポーネントが使用できなくなったり、パフォーマンスの問題が発生したりする場合があります。この場合、Cloud Controlに自動生成されたアラートが表示されるため、管理者が適切なリカバリ・アクションを実行できます。

Cloud Controlのコンポーネントには、次のものがあります。

- Oracle管理サービス(OMS)

  OMS機能は、Cloud Controlのインタフェースを提供するJ2EEアプリケーションのセットで、すべてのOracle管理エージェントと連係して監視情報を処理し、Enterprise Managerリポジトリを永続的なデータ・ストアとして使用します。

- Oracle管理エージェント

  これらのエージェントは、各監視対象ホストにデプロイされているプロセスで、ホスト上のすべてのターゲットを監視し、この情報をOMSに送信し、ホストおよびターゲットのメンテナンスを行います。

- Oracle管理リポジトリ

  リポジトリは、Oracleデータベース内のスキーマで、Cloud Controlによって管理される管理者、ターゲットおよびアプリケーションに関するあらゆる情報が格納されています。

関連項目:

- Cloud Controlを使用したデータベースの管理方法を学習するには、『Oracle Database管理者ガイド』を参照
- Cloud Controlのオンライン・ヘルプを参照してください。

Oracle Enterprise Manager Database Express 12c

Oracle Enterprise Manager Database Express (EM Express)は、Oracleデータベース内に組み込まれたWeb管理製品です。特別なインストールまたは管理は必要ありません。

EM Expressには、Cloud Controlの主要なパフォーマンス管理および基本管理のページが含まれます。これらのページには、次のものがあります。

- データベースのホームページ
- リアルタイムSQL監視
- ASH分析

これらの機能には、オンラインではEM Expressコンソールを介してアクセスでき、オフラインではActive Reportsテクノロジを介してアクセスできます。構造上の観点から、EM Expressは中間層またはミドルウェア・コンポーネントを持たないため、データベース・サーバーのオーバーヘッドはほとんどありません。

EM Expressを使用すれば、ユーザー・セキュリティの管理、データベース・メモリーおよび記憶域の管理などの管理タスクを実行できます。また、データベースのパフォーマンスおよびステータス情報を表示できます。

関連項目:

EM Expressを使用したデータベースの管理方法を学習するには、『Oracle Database 2日でパフォーマンス・チューニング・ガイド』を参照

SQL*Plus

SQL*Plusは、すべてのOracle Databaseのインストール時に含まれる、対話型のバッチ問合せツールです。

このツールには、データベースへの接続時にクライアントとして機能する、コマンドライン・ユーザー・インタフェースがあります。SQL*Plusは独自のコマンドと環境を持っています。ここでは、SQL、PL/SQL、SQL*Plus、オペレーティング・システム・コマンドを入力および実行し、次のタスクを実行できます。

- 問合せ結果の書式設定、計算の実行、保存および印刷
- 表定義およびオブジェクト定義の検証
- バッチ・スクリプトの開発および実行
- データベースの管理

SQL*Plusを使用して、対話的なレポート生成、バッチ処理としてのレポート生成、およびテキスト・ファイル、スクリーンまたはインターネットでの閲覧用のHTMLファイルへの結果の出力が可能です。HTML出力機能を使用するとレポートを動的に生成できます。

関連項目:

SQL*Plusについてさらに学習するには、『SQL*Plusユーザーズ・ガイドおよびリファレンス』を参照

データベースのインストールおよび構成ツール

Oracleでは、Oracle Databaseソフトウェアのインストール・タスクおよび構成タスクを簡略化するためのいくつかのツールを提供しています。

次の表に、サポートされているツールを示します。

表18-1 データベースのインストールおよび構成ツール

ツール	説明	さらに学習するには
Oracle Universal Installer(OUI)	OUI は、Oracle Database ソフトウェアを表示、インストールおよびアンインストールするための GUI ユーティリティです。オンライン・ヘルプがあり、ガイドに従ってインストールできます。	インストールをガイドするオンライン・ヘルプ
Database Upgrade Assistant(DBUA)	DBUA を使用すると、対話形式のガイドに従ってデータベースをアップグレードし、新規リリースに応じてデータベースを構成できます。DBUA では、通常は手動で実行されるタスクをすべて実行することによって、アップグレードを自動化します。DBUA では、表領域やオンライン REDO ログなどの構成オプションに対して推奨設定を作成します。	DBUA を使用したデータベースのアップグレード方法を学習するには、『Oracle Database 2日でデータベース管理者』を参照
Database Configuration Assistant(DBCA)	DBCA には、データベースを作成し、構成するためのグラフィカル・インタフェースおよびガイド付きワークフローが用意されています。このツールを使用すると、Oracle 提供のテンプレートからデータベースを作成したり、独自のデータベースやテンプレートを作成できます。	DBCA を使用したデータベースの作成方法を学習するには、『Oracle Database 管理者ガイド』を参照

Oracle Netの構成および管理ツール

Oracle Net Servicesにより、分散、異機種間コンピューティング環境において、企業全体の接続性が確保されます。Oracle NetはOracle Net Servicesのコンポーネントであり、クライアント・アプリケーションからデータベースへのネットワーク・セッションも確立できます。

次のツールを使用して、Oracle Net Servicesを構成および管理できます。

- Oracle Net Manager

  このツールを使用すると、ローカル・クライアントまたはサーバー・ホスト上のOracleホームにOracle Net Servicesを構成できます。Oracle Net Managerを使用して、ネーミング、ネーミング・メソッド、プロファイルおよびリスナーを設定できます。Oracle Net Managerは、Oracle Enterprise Manager Consoleを使用して起動するか、独立したアプリケーションとして起動できます。

- Oracle Netコンフィギュレーション・アシスタント

  このツールは、ソフトウェアのインストール時に自動的に実行されます。アシスタントでは、リスナー名、プロトコル・アドレス、ネーミング・メソッド、tnsnames.oraファイル内のネット・サービス名、ディレクトリ・サーバーの使用などの基本ネットワーク・コンポーネントをインストール時に構成できます。

- リスナー制御ユーティリティ

  リスナー制御ユーティリティを使用すると、クライアント接続を受信するリスナーを構成できます。ユーティリティには、Enterprise Managerから、またはスタンドアロンのコマンドライン・アプリケーションとしてアクセスできます。

- Oracle Connection Manager制御ユーティリティ

  このコマンドライン・ユーティリティを使用すると、Oracle Connection Manager (ルーターの一種で、クライアント接続リクエストは、このルーターを経由して次のホップに送信されたり、データベースに直接送信されます)を管理できます。ユーティリティ・コマンドを使用して、基本的な管理機能を1つ以上のOracle Connection Managerに対して実行できます。さらに、パラメータの設定を表示および変更できます。

関連項目:

- 「Oracle Net Listener」
- 「Oracle Net Servicesのアーキテクチャの概要」
- Oracle Net Servicesのツールについてさらに学習するには、『Oracle Database Net Services管理者ガイド』を参照してください。

データの移動および分析ツール

Oracle Databaseでは、データの移動および分析を支援するいくつかのユーティリティを提供しています。

たとえば、データベース・ユーティリティを使用して、次のことを実行できます。

表18-2 データの移動および分析のタスク

タスク	さらに学習するには
オペレーティング・システム・ファイルから Oracle Database の表にデータをロードする	「SQL*Loader」
あるデータベースから別のデータベースにデータおよびメタデータを移動する	「Oracle Data Pump エクスポートおよびインポート」
SQL インタフェースを介して REDO ログ・ファイルを問い合せる	「Oracle LogMiner」
Oracle Database のデータを管理する	「ADR コマンド・インタプリタ(ADRCI)」

その他のタスクには、DBVERIFYユーティリティを使用したオフライン・データベースまたはデータファイルに対する物理データ構造の整合性チェックの実行や、DBNEWIDユーティリティを使用した運用データベースのデータベース識別子(DBID)またはデータベース名の変更などがあります。

ノート:

関連項目:

- バックアップおよびリカバリに関連するツールについて学習するには、「バックアップおよびリカバリ」を参照
- DBVERIFYおよびDBNEWIDについて学習するには、『Oracle Databaseユーティリティ』を参照

SQL*Loader

SQL*Loaderは、データ・ファイルと呼ばれる外部ファイル(内部的なデータベースのデータ・ファイルと混同しないでください)からデータベース表にデータをロードします。強力なデータ解析エンジンによって、あらゆるデータ形式のデータファイルに対応できます。

SQL*Loaderを使用して、次のことが可能です。

- 複数のデータファイルから複数の表へのデータのロード

  ロードするデータは、SQL*Loaderデータファイルに格納します。SQL*Loader制御ファイルはテキスト・ファイルであり、このファイルには、SQL*Loaderがデータの格納場所、解析方法、解釈方法、挿入場所などを決定するときに使用するDDL命令が含まれています。

  ノート:

  SQL*Loader データファイルおよび制御ファイルは、Oracle Database データファイルおよび制御ファイルとは関連付けられていません。

- ロード操作の様々な制御

  たとえば、データの選択的なロード、データの文字セットの指定、SQL関数によるデータの操作、指定の列への一意のシーケンシャル・キー値の生成が可能です。また、複雑なエラー・レポートも生成できます。

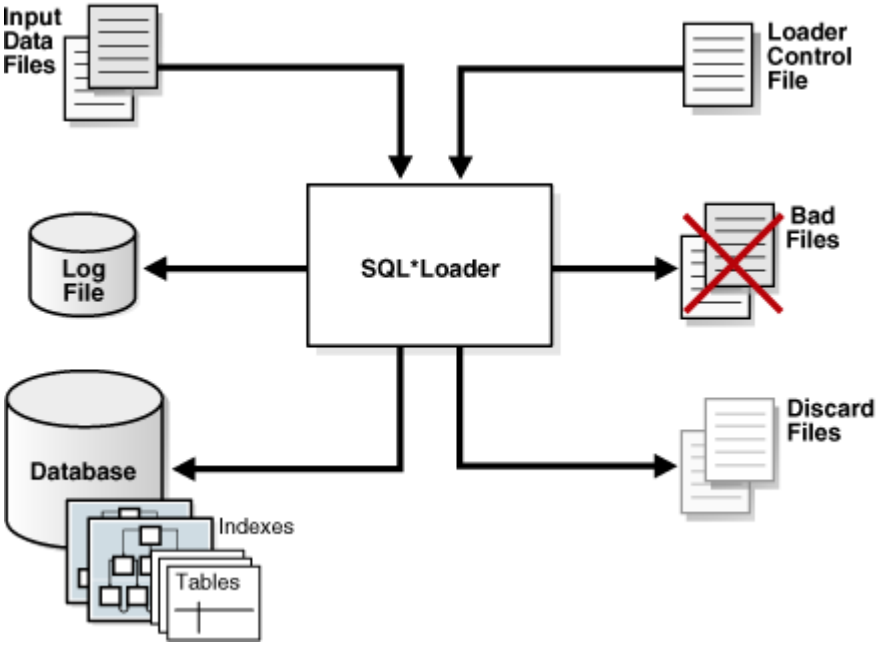
- 従来型パス、ダイレクト・パスまたは外部表ロードの使用

  従来型パスによるロードでは、表にデータを移入するときにSQL INSERT文を実行します。一方、ダイレクト・パスのINSERTでは、データ・ブロックをフォーマットし、それをデータ・ファイルに直接書き込むことによって、データベースのオーバーヘッドを大幅に軽減します。直接書き込む場合は、最高水位標(HWM)よりも高いブロックで処理され、データベース・バッファ・キャッシュを経由せずに、ディスクに直接書き込まれます。直接読み取る場合も、バッファ・キャッシュを経由せずに、ディスクからPGAに直接読み取ります。

  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。

典型的なSQL*Loaderセッションは、SQL*Loader制御ファイルと1つ以上のデータファイルを入力として使用します。出力は、Oracle Database、ログ・ファイル、不良ファイル、および存在する場合は廃棄ファイルです。次の図に、典型的なSQL*Loaderセッションのフローを示します。

図18-1 SQL*Loaderセッション



SQL*Loaderエクスプレス・モードを使用することもでき、次の例のように、SQL*Loaderコマンドでtableパラメータを指定するとアクティブになります。

```
% sqlldr hr table=employees
```

制御ファイルが使用できないため、SQL*Loaderが使用しやすくなります。制御ファイルを解析するかわりに、SQL*Loaderでは表列定義を使用して、入力データ型を確認します。SQL*Loaderは、文字セット、フィールド・デリミタ、そしてデータ・ファイル、ログ・ファイルおよび不良ファイルの名前など、複数のデフォルト推測を作成します。デフォルト値の多くは、コマンドライン・パラメータでオーバーライドできます。

関連項目:

- 「外部表の概要」
- 「文字セット」
- SQL*Loaderについて学習するには、Oracle Databaseユーティリティを参照してください

Oracle Data Pumpエクスポートおよびインポート

Oracle Data Pumpにより、データベース間でデータとメタデータを高速で移動できます。

このテクノロジは、次のOracle Databaseデータ移動ユーティリティのベースとなります。

- データ・ポンプ・エクスポート(エクスポート)

  エクスポートは、データとメタデータをダンプ・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・ファイルにアンロードする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および制御情報を含む1つ以上のバイナリ・ファイルから構成されています。

- データ・ポンプ・インポート(インポート)

  インポートは、エクスポートされたダンプ・ファイル・セットをデータベースにロードするユーティリティです。また、インポートを使用すると、ソース・データベースから中間ファイルなしで宛先データベースに直接ロードすることができ、エクスポート操作とインポート操作を同時に実行することで全体の所要時間が短縮されます。

Oracle Data Pumpは、次のような部分から構成されています。

- expdpおよびimpdpコマンドライン・クライアント

  これらのクライアントは、DBMS_DATAPUMPパッケージに対してコールし、Oracle Data Pump操作を実行します。

- DBMS_DATAPUMP PL/SQLパッケージ(データ・ポンプAPIとも呼ばれる)

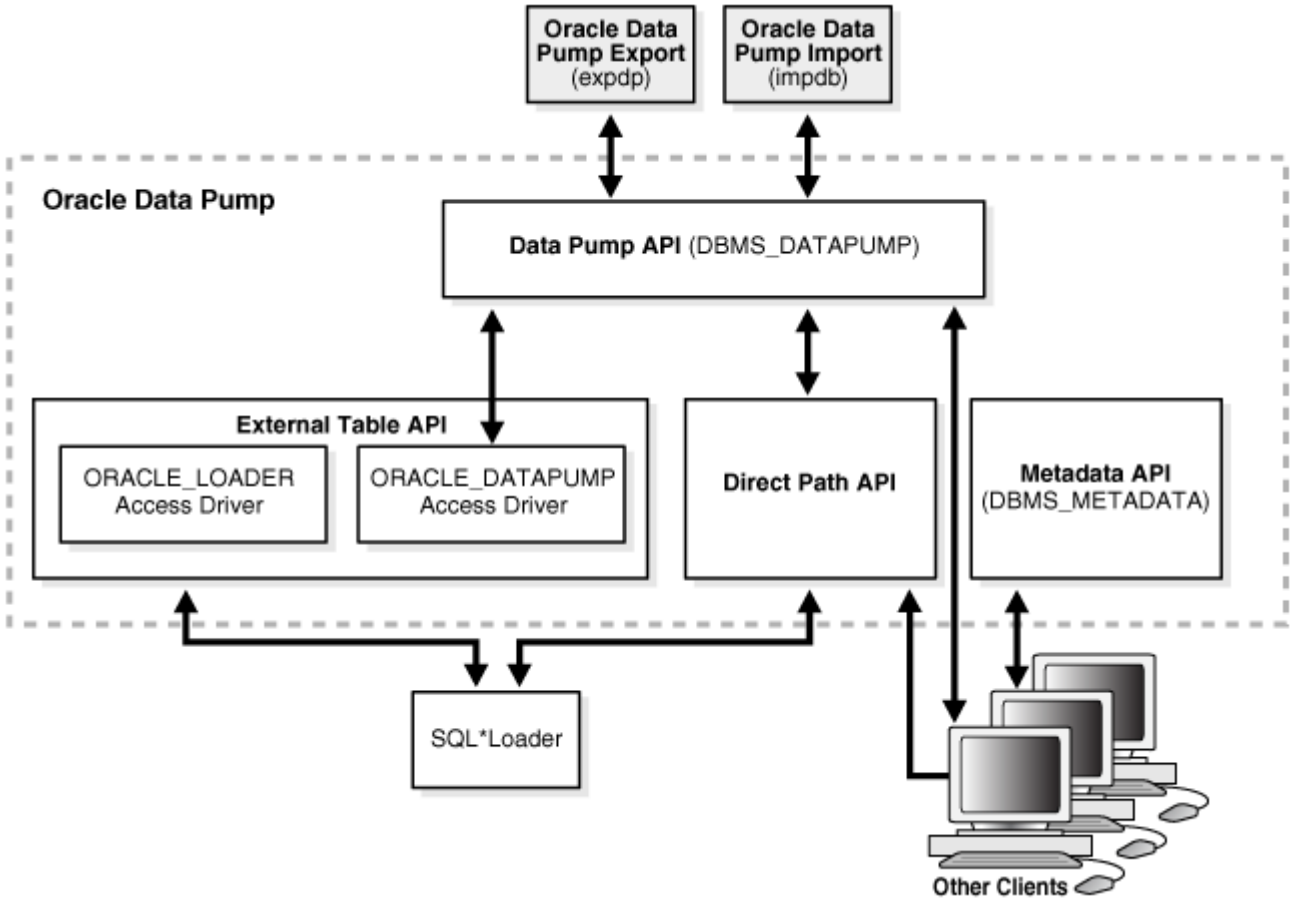
  このAPIによって、高速でインポートおよびエクスポートされます。

- DBMS_METADATA PL/SQLパッケージ(**メタデータAPI**とも呼ばれる)

  メタデータのすべてのロードおよびアンロード・プロセスでこのAPIが使用され、オブジェクト定義がXML形式で格納されます。

次の図に、Oracle Data PumpをSQL*Loaderおよび外部表と統合する方法を示します。図に示すように、SQL*Loaderは外部表APIおよびData Pump APIに統合され、データを外部表にロードします。Oracle Enterprise Manager Cloud Control (Cloud Control)やトランスポータブル表領域などのクライアントでは、Oracle Data Pumpインフラストラクチャを使用できます。

図18-2 Oracle Data Pumpアーキテクチャ



関連項目:

- 「外部表の概要」
- 「PL/SQLパッケージ」
- Oracle Data Pumpの概要は、『Oracle Databaseユーティリティ』を参照してください。
- DBMS_DATAPUMPおよびDBMS_METADATA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

Oracle LogMiner

Oracle LogMinerを使用すると、SQLインタフェースを介してREDOログ・ファイルを問い合せることができます。

REDOログ・ファイルに含まれるデータの使用方法を次に示します。

- 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な破損の開始時点を特定します。
- ユーザー・エラーを検出します。
- トランザクション・レベルでファイングレイン・リカバリを実行するために必要な操作を判断します。
- 傾向分析を使用して、更新および挿入頻度の最も高い表を判断できます。
- REDOログ・ファイルに対するLogMinerの包括的なリレーショナル・インタフェースを使用して、システムの動作を分析し、データベースの使用状況を監査します。

LogMinerには、コマンドライン・インタフェースまたはEnterprise Managerの一部であるOracle LogMiner Viewer GUIを介してアクセスできます。

関連項目:

LogMinerについてさらに学習するには、『Oracle Databaseユーティリティ』を参照

ADRコマンド・インタプリタ(ADRCI)

ADRCIは、問題の調査、ヘルス・チェック・レポートの表示、最初の障害データのパッケージ化およびOracleサポートへのアップロードに使用するコマンドライン・ユーティリティです。

また、このユーティリティでは、自動診断リポジトリ(ADR)内のトレース・ファイル名やアラート・ログを表示できます。ADRCIには、対話形式またはスクリプトで使用できる豊富なコマンド・セットが用意されています。

関連項目:

- 「自動診断リポジトリ」
- ADRおよびADRCIの詳細は、『Oracle Databaseユーティリティ』および『Oracle Database管理者ガイド』を参照してください。

データベース管理者のためのトピック

この項では、他の箇所で説明されていないDBAに不可欠なトピックについて説明します。

この項では、次の項目について説明します。

- バックアップおよびリカバリ
- メモリー管理
- リソース管理およびタスクのスケジューリング
- パフォーマンスおよびチューニング

バックアップおよびリカバリ

バックアップおよびリカバリとは、メディア障害またはユーザー・エラーによるデータ損失からのデータベース保護に関連する、一連の概念、プロシージャ、計画です。

バックアップとは、データのコピーです。バックアップには、データファイル、サーバー・パラメータ・ファイル、制御ファイルなど、データベースの重要な部分を含めることができます。バックアップおよびリカバリが必要になる一例として、ディスク・ドライブの障害によりデータファイルが失われた場合などがあります。失われたファイルのバックアップが存在する場合は、ファイルをリストアおよびリカバリできます。データの損失が発生する前の状態へのリストアにかかわる操作は、メディア・リカバリと呼ばれます。

この項では、次の項目について説明します。

- バックアップおよびリカバリ方法
- Recovery Managerのアーキテクチャ
- データベースのバックアップ
- データ修復
- Zero Data Loss Recovery Appliance。

関連項目:

- バックアップおよびリカバリの概念およびタスクの詳細は、『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
- Microsoft Windowsでデータベースをバックアップおよびリカバリするためのボリューム・シャドウ・コピー・サービス(VSS)アプリケーションの使用方法を学習するには、『*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。

バックアップおよびリカバリ方法

Recovery ManagerまたはユーザーによるManual管理方法を使用して、Oracle Databaseをバックアップ、リストアおよびリカバリできます。

2つのアプローチの主な違いは次のとおりです。

- Recovery Manager (RMAN)

  RMANは、Oracle Databaseと統合し、バックアップおよびリカバリ・アクティビティを実行するためのOracle Databaseユーティリティで、たとえば、履歴バックアップ・メタデータのリポジトリをバックアップ対象のすべてのデータベースの制御ファイルに保持できます。RMANでは、リカバリ・カタログと呼ばれる集中管理されるバックアップ・リポジトリを別のデータベースに保持することもできます。RMANはOracle Databaseの機能であるため、別途インストールする必要はありません。

  RMANは、Oracle Secure Backupと統合されており、Oracle Secure Backupを使用すると、ファイルシステム・データおよびOracle Databaseファイルを保護する、信頼性の高いテープ・バックアップの集中管理が可能となります。Oracle Secure Backup SBTインタフェースでは、RMANを使用したデータベース・ファイルのテープおよびAmazon S3などのインターネット・ベースのWebサービスへのバックアップ、およびこれらの媒体からのデータベース・ファイルのリストアが可能になります。Oracle Secure Backupでは、SANおよびSCSI環境のほとんどすべてのテープ・ドライブおよびテープ・ライブラリがサポートされています。

- ユーザーによる管理方法

  RMANを使用するかわりに、バックアップおよびファイルのリストアの場合はLinuxのddコマンド、メディア・リカバリの場合はSQL*PlusのRECOVERコマンドなど、オペレーティング・システムのコマンドを使用できます。Oracleでは、ユーザー管理のバックアップおよびリカバリが全面的にサポートされていますが、RMANの方がOracle Databaseと統合されており、管理が簡素化されているため、RMANを使用することをお薦めします。

関連項目:

- バックアップおよびリカバリ・ソリューションの概要は、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
- Oracle Secure Backupの概要は、『Oracle Secure Backup管理者ガイド』を参照してください。

Recovery Managerのアーキテクチャ

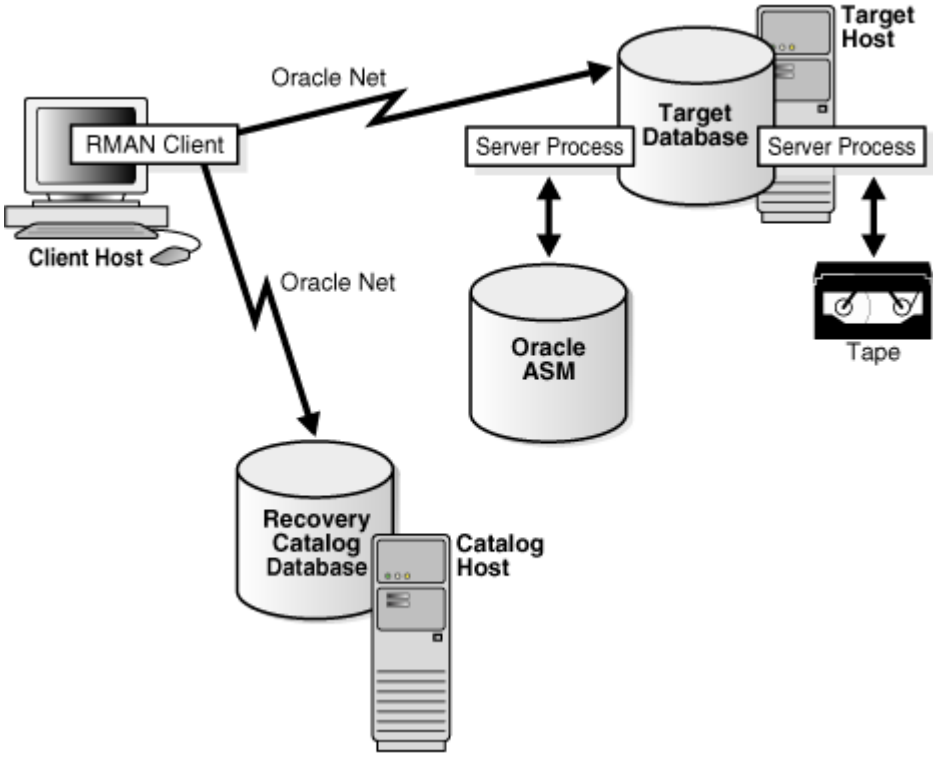
RMANおよびOracle Secure Backupには、コマンドラインおよびEnterprise Manager Cloud Control (Cloud Control)の両方からアクセスできます。

Cloud Controlには、グラフィカルなフロント・エンドと、RMAN用のスケジューリング機能が用意されています。ジョブ・パラメータを入力したら、ジョブ・スケジュールを指定します。Cloud ControlによってRMANが指定した時刻に指定した間隔で繰り返し実行され、バックアップおよびリカバリの操作が実行されます。Cloud Controlは、一連のウィザードによってRMANにアクセスできます。これらのウィザードは、データベース、使用可能なバックアップおよびデータ・リカバリの目的の分析に基づいて、様々なリカバリ手順を提供します。

Cloud Controlを使用することによって、この章で説明する簡単なリストアおよびリカバリの例を実行できます。また、Point-in-Timeリカバリやオラクル・フラッシュバック操作などのより高度なリストアおよびリカバリ方法を使用して、メディア障害およびユーザー・エラーを効率的に修復することもできます。Cloud Controlは、RMANコマンドライン・クライアントよりも簡単に使用できます。

次の図に、RMANアーキテクチャのサンプルを示します。Cloud ControlからアクセスできるRMANクライアントは、ターゲット・データベースのサーバー・セッションを使用して、データをディスクまたはテープにバックアップします。RMANは、外部にあるリカバリ・カタログをバックアップ・メタデータで更新できます。

図18-3 RMANアーキテクチャ



どちらのバックアップおよびリカバリ方法を使用する場合でも、高速リカバリ領域を構成することをお薦めします。このデータベースによって管理されるディレクトリ、ファイルシステムまたはOracle ASMディスク・グループは、アクティブな制御ファイル、オンラインおよびアーカイブREDOログ・ファイル、バックアップなどのバックアップ・ファイルおよびリカバリ・ファイルを一元化します。Oracle Databaseのリカバリ・コンポーネントは、高速リカバリ領域と対話し、データベースのリカバリ可能性を保証します。

関連項目:

- Recovery Managerによるバックアップおよびリカバリの実行方法を学習するには、『Oracle Database 2日でデータベース管理者』を参照してください
- 高速リカバリ領域の設定および管理方法は、『Oracle Database管理者ガイド』を参照してください

データベースのバックアップ

データベース・バックアップは、物理または論理のいずれかです。

物理バックアップはバックアップおよびリカバリ方針の中心概念であり、物理データベース・ファイルのコピーです。物理バックアップを作成するには、RMANユーティリティを使用する方法と、オペレーティング・システムのユーティリティを使用する方法があります。

これに対して、論理バックアップには、表、ストアド・プロシージャおよびその他の論理データが含まれます。Oracle Databaseのデータ・ポンプ・エクスポートなどのユーティリティを使用し、論理データを抽出してバイナリ・ファイルに格納できます。論理バックアップは、物理バックアップを補足できます。

物理バックアップは粒度が粗く、トランスポータビリティが制限されていますが、非常に高速です。論理バックアップは粒度が細かく、トランスポータビリティは完全ですが、物理バックアップより低速です。

関連項目:

物理バックアップおよび論理バックアップについて学習するには、『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

データベース全体のバックアップと部分バックアップ

データベース全体のバックアップとは、データベース内の各データファイルと制御ファイルのバックアップのことです。これは最も一般的なタイプのバックアップです。

部分データベース・バックアップには、個々の表領域またはデータファイルであるデータベースのサブセットが含まれます。表領域バックアップとは、1つまたは複数の表領域内のすべてのデータファイルのバックアップのことです。一貫性の有無にかかわらず、表領域バックアップが有効なのは、データベースがARCHIVELOGモードで稼働している場合のみで、これは、リストア後の表領域とデータベースの他の領域との一貫性を保つにはREDOが必要となるためです。

一貫性バックアップと非一貫性バックアップ

データベース全体のバックアップは、一貫性バックアップか非一貫性バックアップのいずれかです。

一貫性バックアップでは、すべての読取り/書込みデータファイルおよび制御ファイルのチェックポイント SCNが同じであり、このSCNまでのすべての変更が確実にこれらのファイルに格納されます。このタイプのバックアップでは、リストア後にリカバリする必要はありません。

データベースの一貫性バックアップは、一貫性が保たれた状態で停止した後にのみ実行でき、NOARCHIVELOGモードで稼働しているデータベースで有効な唯一のバックアップ・オプションです。他のバックアップ・オプションでは、一貫性のためメディア・リカバリを必要としますが、これはアーカイブREDOログ・ファイルを適用しなければ不可能です。

ノート:

REDO を適用せずにデータベース全体の一貫性バックアップをリストアすると、バックアップ後に実行されたトランザクションはすべて失われます。

非一貫性バックアップでは、読込み/書込みデータファイルおよび制御ファイルのチェックポイントSCNが同じであるという保証はなく、変更が失われる場合があります。バックアップを実行している間もデータファイルが変更されることがあるため、すべてのオンライン・バックアップは必然的に非一貫性です。

非一貫性バックアップでは、データベースを完全に保護するバックアップを作成するためにデータベースを停止する必要がないため、優れた可用性が実現されます。ARCHIVELOGモードでデータベースを実行する場合、アーカイブREDOログおよびデータファイルをバックアップする場合は、非一貫性バックアップを堅実なバックアップおよびリカバリ計画の基礎とできます。

関連項目:

- 「停止モード」
- 非一貫性バックアップについてさらに学習するには、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

バックアップ・セットおよびイメージ・コピー

RMANのBACKUPコマンドでは、イメージ・コピーまたはバックアップ・セットが生成されます。

バックアップ・タイプは次のように異なります。

- イメージ・コピー

  イメージ・コピーとは、データファイル、制御ファイルまたはアーカイブREDOログ・ファイルをビット単位でディスク上に複製したものです。オペレーティング・システムのユーティリティまたはRMANを使用して物理ファイルのイメージ・コピーを作成し、いずれかのツールを使用してファイルをリストアできます。

  ノート:

  オペレーティング・システムによるコピーとは異なり、RMAN ではファイル内のブロックが検証され、イメージ・コピーが RMAN リポジトリに記録されます。

- バックアップ・セット

  RMANは、バックアップ・セットと呼ばれる独自の形式のバックアップを作成することもできます。バックアップ・セットには、1つ以上のデータファイル、アーカイブREDOログ・ファイル、制御ファイル、または1つのサーバー・パラメータ・ファイルからのデータが格納されます。バックアップ・セットの最小単位は、バックアップ・ピースと呼ばれるバイナリ・ファイルです。バックアップ・セットは、RMANがテープ・ドライブなどのシーケンシャル・デバイスへバックアップを書き込むための唯一の方法です。

  バックアップ・セットを使用すると、テープ・デバイスが連続して動作できるようになります。たとえば、RMANでは、低速、中速および高速の各ディスクのブロックを1つのバックアップ・セットに結合することにより、テープ・デバイス内の入力ブロックを一定に保つことができます。ディスクのイメージ・コピーは、段階的な更新や所定の位置でのリカバリが可能であるため、役に立ちます。

関連項目:

バックアップ・セットおよびイメージ・コピーについてさらに学習するには、『*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。

データ修復

データベースの通常の機能が停止するような問題、およびI/O処理に影響を及ぼす問題は数多くありますが、一般的に、DBAが介入してデータを修復する必要があるのは一部の問題のみです。

データの修復は、通常、次の場合に必要です。

- メディア障害

  メディア障害は、データベース外部の問題により、データベースでファイルの読取りまたは書込みができない場合に発生します。一般的なメディア障害には、ディスクのヘッド・クラッシュなどの物理的な障害、およびデータベース・ファイルの上書き、削除または破損が含まれます。メディア障害は、ユーザーまたはアプリケーションのエラーより頻度は低いですが、バックアップおよびリカバリ計画を堅実にするには、これに備えておく必要があります。

- ユーザー・エラー

  不適切な更新、表の内容の削除またはデータベース・オブジェクトの削除など、ユーザーまたはアプリケーションによって、データベースに不要な変更が加えられる場合があります。適切なバックアップおよびリカバリ計画では、データベースの可用性に対する影響を最小化し、DBAの労力を最低限に抑えて、必要な状態にデータベースを戻すことができます。

一般的に、これらの問題の解決方法は複数あります。この項では、これらの解決方法について説明します。

関連項目:

- 「人為的エラー」
- データ修復の概念の詳細は、『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

Oracle Flashbackテクノロジ

Oracle Databaseでは、**Oracle Flashbackテクノロジ**と呼ばれる機能グループが提供されており、バックアップをリストアせずに、過去のデータの状態を表示したり、特定の時間に応じてデータを前後に巻き戻すことができます。

データベースの変更内容にもよりますが、フラッシュバック機能では、メディア・リカバリよりも迅速に、可用性に及ぼす影響も少なく、不要な変更を元に戻すことができます。バックアップおよびリカバリに最適なフラッシュバック機能には、次のものがあります。

- フラッシュバック・データベース

  非CDBでは、Oracle Databaseを以前の時点まで巻き戻し、論理データの破損やユーザー・エラーによる問題を解決できます。フラッシュバック・データベースは、Data Guard、データ・リカバリ・アドバイザの補完、およびクローン・データベースの同期化にも使用できます。フラッシュバック・データベースではファイル上でメディア・リカバリはリストアまたは実行されないため、これを使用してディスク・クラッシュなどのメディア障害を修正することはできません。

- フラッシュバックPDB

  マルチテナント・コンテナ・データベース(CDB)で、他のPDBに影響を及ぼさずにプラガブル・データベース(PDB)を巻き戻しできます。PDBリストア・ポイントを作成して、他のPDBに影響を及ぼさずにこのリストア・ポイントまでPDBを巻き戻すこともできます。

- フラッシュバック表

  1つのSQL文で、指定した時点まで表を巻き戻しできます。表データとともに、関連する索引、トリガーおよび制約をリストアでき、データベースはオンライン状態で、指定された表に対する変更のみが取り消されます。フラッシュバック表は、不良ディスクやデータ・セグメントと索引の非一貫性など、物理的な破損には対応していません。

- フラッシュバック・ドロップ

  DROP TABLE操作の効果を無効にできます。フラッシュバック・ドロップは、ポイント・イン・タイム・リカバリなどのリカバリ・メカニズムより大幅に高速であり、最近のトランザクションの損失や停止時間も発生しません。

関連項目:

- フラッシュバック機能についてさらに学習するには、『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。
- FLASHBACK DATABASE文について学習するには、Oracle Database SQL言語リファレンスおよび*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・リカバリ・アドバイザ

データ・リカバリ・アドバイザを使用して、データ障害の診断、ご使用の環境に最善の修復オプションの表示および修理の実行と検証を行います。

データ・リカバリ・アドバイザ・ツールは、データ障害を自動的に診断して、適切な修復オプションを提示し、ユーザーのリクエストに基づいて修復を実行します。データ・リカバリ・アドバイザは、自動データ修復用の集中管理ツールを提供することによって、Oracle Databaseの管理性および信頼性を向上し、リカバリ時間を短縮します。

データベースには、チェックを実行するための状態モニター・フレームワークがあります。チェッカは、データベースまたはそのコンポーネントの状態を評価するために状態モニターに登録されている操作またはプロシージャです。状態評価はデータ整合性チェックとも呼ばれ、対処的または予防的に起動できます。

障害とは、データ整合性チェックにより検出される永続的なデータ破損のことです。通常、障害は事後対応的に検出されます。破損したデータに対するデータベース操作でエラーが発生すると、データ整合性チェックが自動的に開始され、データベースでエラーに関連する障害が調査されます。障害が診断された場合、その障害はデータベースにより自動リポジトリ(ADR)に記録されます。

障害がデータベースにより検出され、ADRに保存されると、データ・リカバリ・アドバイザは最善の修復オプションおよびデータベースへの影響を自動的に判断します。通常、データ・リカバリ・アドバイザは、それぞれの障害または障害グループに対して手動および自動の両方の修復オプションを生成します。

データ・リカバリ・アドバイザは、自動修復オプションを提示する前に、修復を完了するために必要なメディア・コンポーネントが使用可能かどうか、およびそのオプションが特定の環境に適しているかどうかを検証します。自動修復を選択すると、Oracle Databaseによって自動的に修復されます。データ・リカバリ・アドバイザは、修復の成功を確認し、該当する障害をクローズします。

関連項目:

データ・リカバリ・アドバイザの使用方法を学習するには、『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

ブロック・メディア・リカバリ

ブロック・メディア・リカバリは、データファイルをオンラインの状態に保ったまま、破損データ・ブロックをリストアしてリカバリするテクニックです。

ブロック破損とは、認識されているOracle形式ではないデータ・ブロック、または内容の一貫性が内部で保たれていないデータ・ブロックです。ブロックの破損を検出したときに、破損が少数のブロックのみに限定されている場合は、データファイル・リカバリよりもブロック・リカバリが適しています。

関連項目:

- 「データ破損」
- ブロック・メディア・リカバリの実行方法を学習するには、『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

データファイル・リカバリ

データファイル・リカバリは、消失または破損した現行のデータファイルまたは制御ファイルを修復します。また、表領域がOFFLINE NORMALオプションを設定せずにオフラインになったときに消失した変更をリカバリします。

データファイルまたは制御ファイルのバックアップをリストアする場合、またはデータファイルがOFFLINE NORMALオプションを設定せず

にオフラインになった場合は、メディア・リカバリが必要です。オンライン・データファイルのメディア・リカバリが必要な場合は、データベースをオープンできず、また、メディア・リカバリが完了するまでは、メディア・リカバリを必要とするデータファイルはオンライン化できません。

データファイルまたは制御ファイルの物理バックアップのリストアとは、それを再構築してOracle Databaseで使用可能にすることです。バックアップのリカバリとは、アーカイブREDOログ・ファイルを適用して、消失した変更を再構築することです。RMANでは、前回のバックアップ後に変更されたデータ・ブロックのみを含む増分バックアップを使用して、データファイルをリカバリすることもできます。

変更がオンライン・ファイルに自動的に適用されるインスタンス・リカバリとは異なり、メディア・リカバリでは、ユーザーによる起動が必要であり、リストアされたバックアップにアーカイブREDOログ・ファイルが適用されます。データファイルのメディア・リカバリは、オフライン・データファイルか、データベース・インスタンスによりオープンされていないデータベースのデータファイルにのみ機能します。

データファイル・メディア・リカバリは、すべての変更を適用するかどうかで異なります。

- 完全リカバリ

  完全リカバリでは、アーカイブREDOログとオンラインREDOログに含まれるREDO変更すべてがバックアップに適用されます。通常、完全メディア・リカバリは、メディア障害によりデータファイルまたは制御ファイルが破損した後に実行します。完全リカバリは、データベース、表領域またはデータファイルに対して実行できます。

- 不完全リカバリ

  不完全リカバリはデータベース・ポイント・イン・タイム・リカバリとも呼ばれ、データベースの以前のバージョンを生成します。この場合、リストアされたバックアップ以後に生成されたREDOすべてを適用するわけではありません。通常、データベースのポイント・イン・タイム・リカバリは、Oracle Flashback Databaseを使用できないときにユーザー・エラーを取り消す場合に実行します。

  不完全リカバリを実行するには、リカバリが必要な時点より前に作成されたバックアップからデータファイルをすべてリストアし、リカバリの完了時に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。ログをリセットすると、新しくログ順序1からログ順序番号のストリームが開始します。

  ノート:

  現行のデータファイルが利用可能な場合は、DBPITR のかわりにフラッシュバック・データベースを使用できます。

  表領域のポイント・イン・タイム・リカバリ(TSPITR)機能を使用すると、1つ以上の表領域をデータベースの残りの部分より古い時点までリカバリできます。

関連項目:

- 「インスタンス・リカバリの概要」
- メディア・リカバリの概念の詳細は、『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

Zero Data Loss Recovery Appliance。

クラウド規模のZero Data Loss Recovery Applianceは、**リカバリ・アプライアンス**として一般的に知られる、企業全体のすべてのOracleデータベースのデータ損失とバックアップのオーバーヘッドを大幅に削減するためのエンジニアド・システムです。

リカバリ・アプライアンスはRMANと統合されており、クラウド規模で耐障害性の高いハードウェアと記憶域を使用して、多数のデータベースに対する集中管理されたバックアップとリカバリの計画をデプロイします。リカバリ・アプライアンスでは、継続してバックアップのリカバリ可能性が検証されます。

リカバリ・アプライアンスの利点

集中管理されたリカバリ・アプライアンスは大部分のデータベース・バックアップとリストア処理を実行し、記憶域使用率、パフォーマンス、およびバックアップの管理性の効率を高めます。

主な利点は次のとおりです。

- データ損失の解消

  リカバリ・アプライアンスでは、次のようなテクノロジを使用して、データ・センターのほとんどのデータベースで発生するデータ損失の可能性を解消します。

  - リアルタイムREDOトランスポートと呼ばれる、保護対象データベースのSGAからリカバリ・アプライアンスへのREDO変更の継続的な転送
  - リモート・リカバリ・アプライアンスへのレプリケーション
  - 集中管理されたリカバリ・アプライアンスによる自動テープ・バックアップ
  - エンドツーエンドのデータベース・ブロック検証

- 最小バックアップのオーバーヘッド

  データベース・サーバーのバックアップ・オーバーヘッドは、リカバリ・アプライアンスへオフロードすることによって最小化し、リカバリ・アプライアンスでは統一されたディスク・プールで複数のデータベースのバックアップを管理します。RMANの永久的増分バックアップ計画には、初期レベル0のリカバリ・アプライアンスへのバックアップ作成が含まれており、その後の増分バックアップはすべてレベル1です。リカバリ・アプライアンスでは、レベル0とレベル1のバックアップを組み合せて仮想完全バックアップを作成します。リカバリ・アプライアンスではブロック・レベルで継続的にバックアップを圧縮、重複解除、および検証します。

- エンドツーエンド・データ保護の可視性の向上

  Cloud Controlでは、RMANバックアップが開始されてからディスクまたはテープに格納されるまで、またはダウンストリーム・リカバリ・アプライアンスへレプリケートされるまでの、リカバリ・アプライアンスが管理するバックアップ・ライフサイクルに対して完全なエンドツーエンドのビューが提供されます。Enterprise Manager for Zero Data Loss Recovery Applianceプラグイン(リカバリ・アプライアンス・プラグイン)をインストールすると、監視と管理が可能になります。

- クラウド規模の保護

  リカバリ・アプライアンスがサポートするデータベースは、数十から数百、あるいは数千にのぼります。このアーキテクチャの主なコンポーネントは次のとおりです。

  - リカバリ・アプライアンスでは、保護ポリシーによって管理を簡素化し、このポリシーに関連する各データベースに対して強制されるリカバリ・ウィンドウ目標を定義します。保護ポリシーでは、データベースを共通する特性を持つ階層のグループとしてまとめると、管理性が向上します。
  - リカバリ・アプライアンスでは保護ポリシーにより、保護対象の各データベースのリカバリ・ウィンドウ目標に応じたバックアップ記憶域領域を管理します。このデータベース指向の詳細な記憶域領域の管理方法により、サー

ド・パーティ・アプライアンスと同様に、記憶域ボリューム・レベルで領域を管理する必要がなくなります。

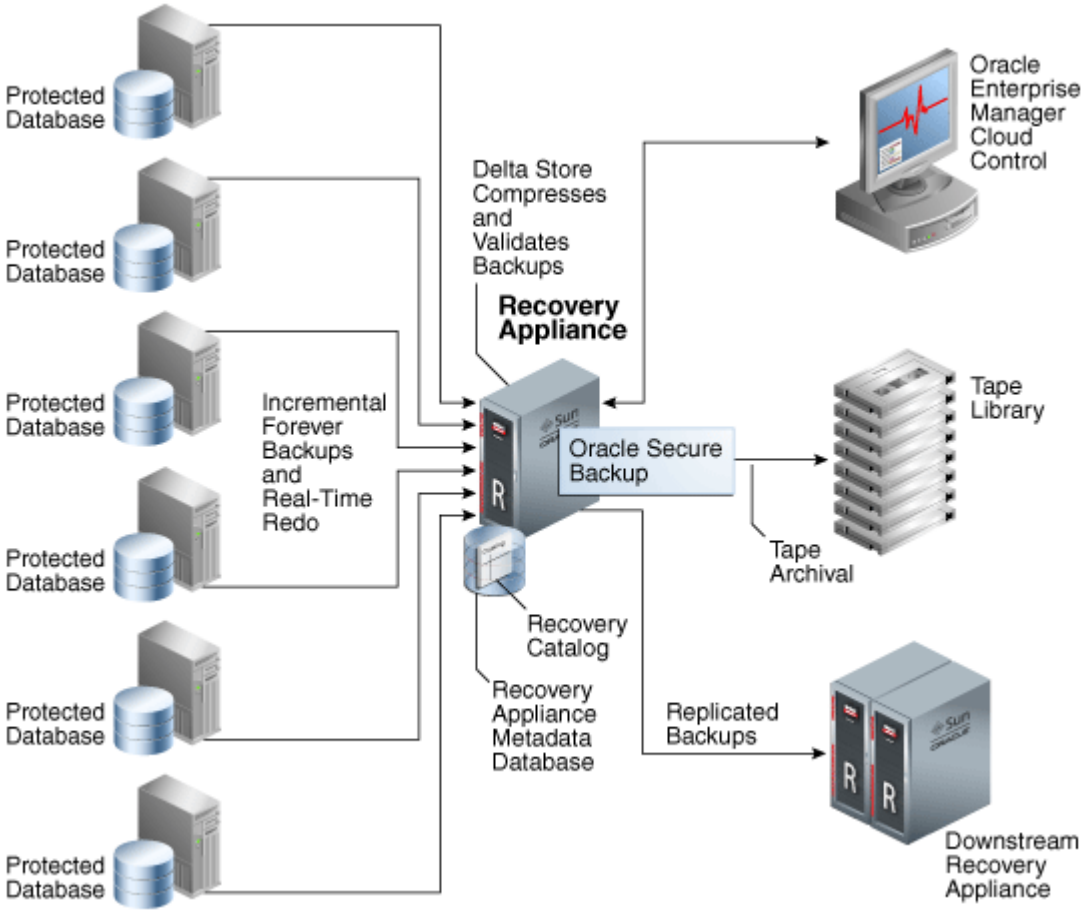
- リカバリ・アプライアンスは、記憶域リソースを単純なモジュラ形式で加算し、バックアップ・トラフィック、記憶域使用量、およびデータベース数の増加に対応するように調整できます。

リカバリ・アプライアンス環境

保護対象データベースは、データをリカバリ・アプライアンスにバックアップするクライアント・データベースです。

次の図に、6つの保護対象データベースと2つのリカバリ・アプライアンスを含むサンプルの環境を示します。それぞれのデータベースは、バックアップにZero Data Loss Recovery Applianceバックアップ・モジュール(リカバリ・アプライアンス・バックアップ・モジュール)を使用します。このモジュールはRMANがネットワーク経由でバックアップをリカバリ・アプライアンスに転送する、Oracle提供のSBTライブラリです。

図18-4 リカバリ・アプライアンス環境



それぞれのリカバリ・アプライアンスにあり、リカバリ・カタログに保存されているメタデータを管理するリカバリ・アプライアンス・メタデータ・データベース。バックアップをリカバリ・アプライアンスに送信する保護対象データベースはすべて、このリカバリ・カタログを使用する必要があります。バックアップは、Oracle ASMディスク・グループ・セットであるリカバリ・アプライアンスの記憶域の場所にあります。

ノート:

データベースでは、バックアップ・リポジトリとしてではなく、リカバリ・カタログとしてリカバリ・アプライアンスが使用されることがあります。

管理者はCloud Controlを使用して環境を管理および監視します。Cloud Controlでは、各データベースのバックアップ・ライ

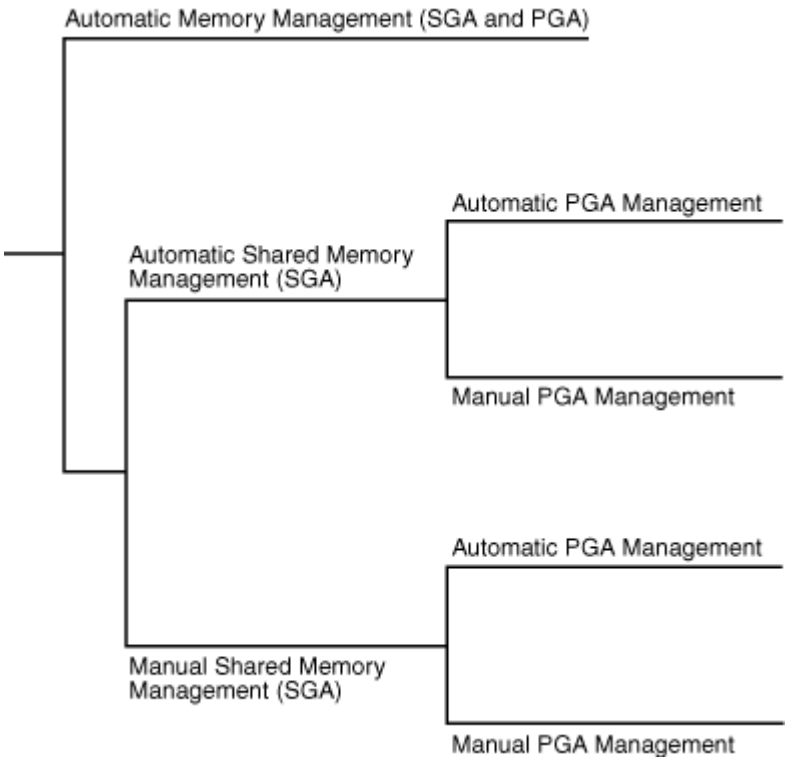
フサイクル全体、およびバックアップがディスク、テープ、または別のリカバリ・アプライアンスのどこにあるかが「1枚ガラス」のビューで提供されます。

メモリー管理

メモリー管理では、データベースの変化に応じて、Oracleインスタンスのメモリー構造を最適なサイズに維持します。初期化パラメータ設定により、SGAおよびインスタンスPGAメモリーの管理方法が決まります。

図18-5に、メモリー管理オプションのディシジョン・ツリーを示します。次の各項では、各オプションの詳細を説明します。

図18-5 メモリー管理方法



関連項目:

SGAおよびPGAについてさらに学習するには、「メモリー・アーキテクチャ」を参照してください

自動メモリー管理

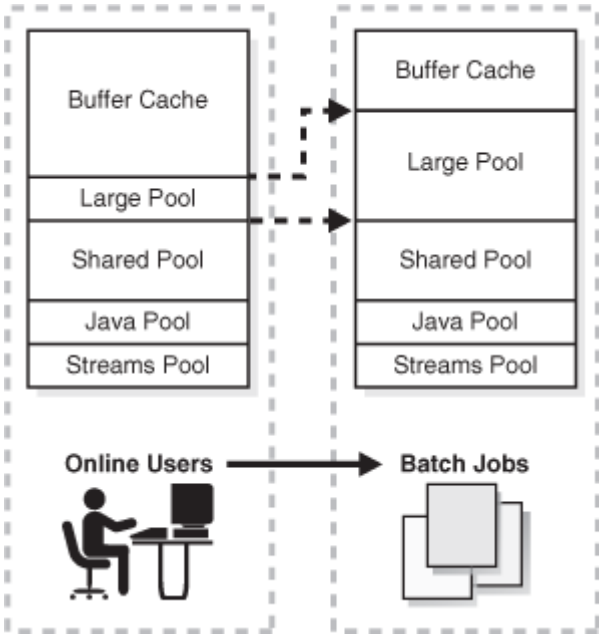
自動メモリー管理では、Oracle DatabaseによりSGAおよびインスタンスPGAメモリーが完全に自動管理されます。最もシンプルなこの方法を使用することをお薦めします。

ユーザー指定の制御は、ターゲット・メモリー・サイズ初期化パラメータ(MEMORY_TARGET)と、オプションの最大メモリー・サイズ初期化パラメータ(MEMORY_MAX_TARGET)のみです。Oracle Databaseは、ターゲット・メモリー・サイズに自動的にチューニングし、必要に応じて、SGAとインスタンスPGA間でメモリーを再配布します。

SGAには、インメモリー列ストア(IM列ストア)またはmemoptimizeプールを含めることができます(いずれもオプションです)。使用するメモリー管理方法にかかわらず、IM列ストアのサイズ設定にはINMEMORY_SIZE初期化パラメータを使用し、memoptimizeプールのサイズ設定にはMEMOPTIMIZE_POOL_SIZE初期化パラメータを使用します。IM列ストアのサイズとmemoptimizeプールはメモリー・ターゲットに含まれますが、自動サイズ変更アルゴリズムによる管理対象ではありません。たとえば、MEMORY_TARGETを5GBに設定し、INMEMORY_SIZEを1GBに設定した場合、メモリー・ターゲット全体は5GBで(6GBではない)INMEMORY_SIZEは常に1GBです。

次の図に、オンライン・ユーザーにより発行されたジョブを必要に応じて処理したり、バッチ・ジョブを処理するデータベースを示します。自動メモリー管理を使用すると、データベースは実行するジョブのタイプに応じて、自動的にラージ・プールとデータベース・バッファ・キャッシュのサイズを調整します。以下の図にはIM列ストアおよびメモリー最適化領域は示されていない点に注意してください。

図18-6 自動メモリー管理



DBCAを使用してデータベースを作成し、基本インストール・オプションを選択した場合、Oracle Databaseではデフォルトで自動メモリー管理が有効化されます。

関連項目:

- 「インメモリー領域」
- memoptimizeプール
- 自動メモリー管理について学習するには、『Oracle Database管理者ガイド』を参照してください。

SGAの共有メモリー管理

自動共有メモリー管理を使用してSGAサイズを設定するか、手動でSGAコンポーネントをチューニングすることで、SGAのサイズを手動で制御できます。

自動メモリー管理が有効でない場合、システムはSGAに共有メモリー管理を使用します。共有メモリー管理では、次のいずれかのモードを選択できます。

- 自動共有メモリー管理

  このモードは、自動メモリー管理が無効になっている場合のデフォルトで、SGAのサイズをより直接的に制御できます。データベースは、SGAの合計サイズをターゲット・サイズにチューニングし、動的にSGAコンポーネントのサイズをチューニングします。サーバー・パラメータ・ファイルを使用している場合、Oracle Databaseでは、自動チューニングされたコンポーネントのサイズがデータベース・インスタンス停止後も保持されます。

- 手動共有メモリー管理

  このモードでは、個々のSGAコンポーネントのサイズをいくつか設定し、処理ごとに個々のSGAコンポーネントを手動でチューニングします。個々のSGAコンポーネントのサイズを完全に制御できます。自動メモリー管理および自動共有メモ

リー管理が両方とも無効になっている場合は、このモードがデフォルトになります。

ノート:

自動メモリー管理が無効になっている場合、データベースはユーザー・ワークロードに応じて共有プールとバッファ・キャッシュの相対サイズを自動的に調整することがあります。

関連項目:

- 共有メモリー管理について学習するには、『Oracle Database管理者ガイド』を参照
- 手動モード中の自動サイズ変更についてさらに学習するには、My Oracle Supportノート1269139.1を参照してください。

  https://support.oracle.com/CSP/main/article?cmd=show&type=NOT&id=1269139.1

インスタンスPGAのメモリー管理

自動メモリー管理が有効でない場合は、Oracle Databaseは自動または手動のPGAメモリー管理を使用します。

PGAメモリーの管理には、次のモードを指定できます。

- 自動PGAメモリー管理

  自動メモリー管理(MEMORY_TARGET)が無効で、PGA_AGGREGATE_TARGETが0(ゼロ)以外の値に設定されている場合、データベースは自動PGAメモリー管理を使用します。このモードでは、PGA_AGGREGATE_TARGETによってインスタンスPGAの柔軟なターゲット・サイズが指定されます。ターゲットが柔軟であるのは、PGAではなく一時領域を使用するように選択できる特定タイプのメモリー割当てにのみ適用されるためです。データベースは、このターゲットに対してインスタンスPGAのサイズをチューニングし、個々のPGAのサイズを動的にチューニングします。ターゲット・サイズを明示的に設定しない場合は、適切なデフォルト値がデータベースによって自動的に構成されます。

  PGA_AGGREGATE_LIMIT初期化パラメータは、PGAメモリーに対してインスタンス全体の厳しい制限を動的に設定します。パラメータは変化するメモリーの状態に対応するため、明示的にパラメータ値を設定する必要はありません。デフォルトでは、PGA_AGGREGATE_LIMITは次の値より大きい値に設定されます。

  - 2GB
  - PGA_AGGREGATE_TARGET初期化パラメータ設定の200%
  - (PROCESSES初期化パラメータ設定の値) * 3MB

  バックグラウンド・プロセスでは、PGAサイズとPGA_AGGREGATE_LIMITで設定された制限とを定期的に比較します。制限に達した場合または制限を超過した場合、このプロセスでは、最もチューニング不可なPGAメモリーを使用してセッションのコールを終了します。これらのセッションによっても十分なメモリーが解放されない場合、それらも終了されます。

- 手動PGAメモリー管理

  自動メモリー管理が無効で、PGA_AGGREGATE_TARGETが0(ゼロ)に設定されている場合、手動PGA管理がデフォルトになります。Oracle Databaseの以前のリリースでは、DBAが作業領域の最大サイズをSQL演算子(ソートやハッシュ結合など)のタイプごとに手動で指定する必要がありました。しかし、ワークロードは常に変化するため、この技法は

非常に困難であることがわかりました。Oracle Databaseでは手動によるPGAメモリー管理もサポートされていますが、自動メモリー管理を使用することをお薦めします。

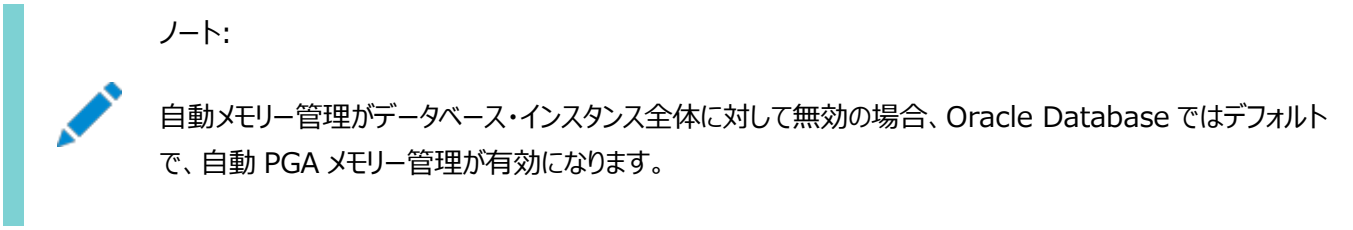
関連項目:

PGAメモリー管理について学習するには、『*Oracle Databaseパフォーマンス・チューニング・ガイド*』を参照してください。

メモリー管理方法の概要

メモリーの管理は、自動または手動のいずれかです。

自動メモリー管理を有効化しない場合は、SGAおよびPGAのそれぞれに対して個別にメモリー管理方法を構成する必要があります。



ノート:

自動メモリー管理がデータベース・インスタンス全体に対して無効の場合、Oracle Database ではデフォルトで、自動 PGA メモリー管理が有効になります。

次の表には、INMEMORY_SIZE初期化パラメータとMEMOPTIMIZE_POOL_SIZE初期化パラメータが含まれています。どちらも、オプションのメモリー領域を制御します。

図18-3 メモリー管理方法

インスタンス	SGA	PGA	説明	初期化パラメータ
自動	N/A	N/A	1 つのインスタンス・ターゲット・サイズに基づいてインスタンスのサイズがチューニングされます。	設定項目: ● データベース・インスタンスの合計メモリー・ターゲット・サイズ(MEMORY_TARGET) ● (オプション)データベース・インスタンスの最大メモリー・サイズ(MEMORY_MAX_TARGET) ● SGA 内の IM 列ストアのオプションのサイズ (INMEMORY_SIZE)
N/A	自動	自動	SGA は SGA ターゲットに基づいて自動的にチューニングされます。 PGA は PGA ターゲットに基づいて自動的にチューニングされます。	設定項目: ● SGA のターゲット・サイズ(SGA_TARGET) ● (オプション)SGA の最大サイズ (SGA_MAX_SIZE) ● SGA 内の IM 列ストアのオプションのサイズ

インスタンス	SGA	PGA	説明	初期化パラメータ
				(INMEMORY_SIZE) ● オプションで SGA の memoptimize プールのサイズ(MEMOPTIMIZE_POOL_SIZE) ● PGA 集計ターゲット・サイズ (PGA_AGGREGATE_TARGET)脚注 1 データベースによって PGA_AGGREGATE_LIMIT 初期化パラメータが自動的に構成されます。このパラメータは手動で設定できます。
N/A	自動	手動	SGA は SGA ターゲットに基づいて自動的にチューニングされます。 PGA は各タイプの SQL 演算子の作業領域の最大サイズを指定し、手動で制御します。	設定項目: ● SGA のターゲット・サイズ(SGA_TARGET) ● (オプション)SGA の最大サイズ (SGA_MAX_SIZE) ● SGA 内の IM 列ストアのオプションのサイズ (INMEMORY_SIZE) ● オプションで SGA の memoptimize プールのサイズ(MEMOPTIMIZE_POOL_SIZE) ● SORT_AREA_SIZE、HASH_AREA_SIZE、BITMAP_MERGE_AREA_SIZE などの PGA 作業領域パラメータ
N/A	手動	自動	SGA は個々のコンポーネント・サイズを設定し、手動で制御します。 PGA は PGA ターゲットに基づいて自動的にチューニングされます。	設定項目: ● 共有プール・サイズ(SHARED_POOL_SIZE) ● バッファ・キャッシュ・サイズ (DB_CACHE_SIZE) ● ラージ・プール・サイズ(LARGE_POOL_SIZE) ● Java プール・サイズ(JAVA_POOL_SIZE) ● SGA 内の IM 列ストアのオプションのサイズ (INMEMORY_SIZE) ● オプションで SGA の memoptimize プー

インスタンス	SGA	PGA	説明	初期化パラメータ
				ルのサイズ(MEMOPTIMIZE_POOL_SIZE) ● PGA 集計ターゲット・サイズ (PGA_AGGREGATE_TARGET)脚注 2 データベースによって PGA_AGGREGATE_LIMIT 初期化パラメータが自動的に構成されます。このパラメータは手動で設定できます。
N/A	手動	手動	SGA コンポーネントサイズを手動で構成する必要があります。 PGA は各タイプの SQL 演算子の作業領域の最大サイズを指定し、手動で制御します。	SGA コンポーネントサイズを手動で構成する必要があります。設定項目: ● 共有プール・サイズ(SHARED_POOL_SIZE) ● バッファ・キャッシュ・サイズ (DB_CACHE_SIZE) ● ラージ・プール・サイズ(LARGE_POOL_SIZE) ● Java プール・サイズ(JAVA_POOL_SIZE) ● SGA 内の IM 列ストアのオプションのサイズ (INMEMORY_SIZE) ● オプションで SGA の memoptimize プールのサイズ(MEMOPTIMIZE_POOL_SIZE) ● SORT_AREA_SIZE、HASH_AREA_SIZE、BITMAP_MERGE_AREA_SIZE などの PGA 作業領域パラメータ

脚注1

データベースによってPGA_AGGREGATE_LIMIT初期化パラメータが自動的に構成されます。このパラメータを手動で設定するように選択することもできます。

脚注2

データベースによってPGA_AGGREGATE_LIMIT初期化パラメータが自動的に構成されます。このパラメータを手動で設定するように選択することもできます。

関連項目:

自動メモリー管理が使用できないプラットフォームもあります。『Oracle Database管理者ガイド』を参照してください。

リソース管理およびタスクのスケジューリング

Oracle Databaseには、ユーザーへの影響を軽減するためにリソース管理およびタスクのスケジューリングに役立つツールが用意されています。

多数のユーザーがアクティブになるデータベースでは、リソース管理がデータベース管理の重要な部分になります。過度にリソースを消費するセッションのため、他のセッションの処理が停止することがあります。このため、最適なタイミングでタスクを実行するようタスクをスケジューリングする方法が問題となります。Oracle Databaseでは、これらの問題を解決するためのツールを提供しています。

データベース・リソース・マネージャ

Oracle Database Resource Manager(リソース・マネージャ)は、ユーザー・アカウント、アプリケーション、およびサービスに割り当てるデータベース・リソースを細かく制御します。リソース・マネージャは、主にゲートキーパーとして機能し、他のジョブを高速で実行できるように一部のジョブを低速で実行します。

DBMS_RESOURCE_MANAGER PL/SQLパッケージを使用することで、オペレーティング・システムでは十分に管理できない次の多くのリソース割当ての問題が解決されます。

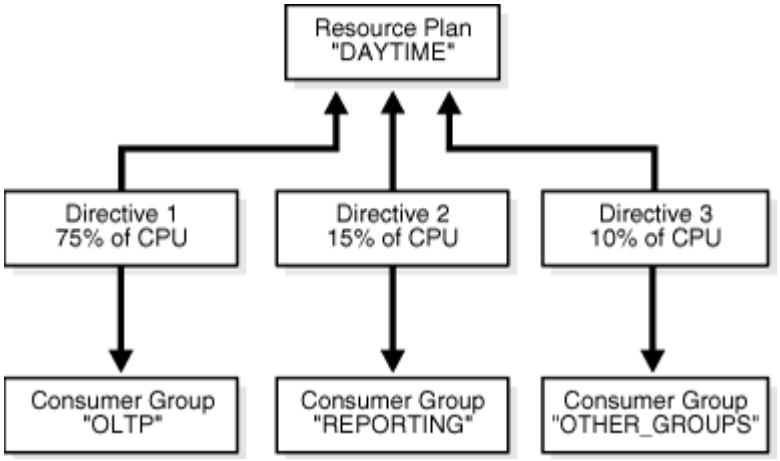
- 過度のオーバーヘッド
- 非効率的なスケジューリング
- 不適当なリソースの割当て
- データベース固有のリソース管理ができない状態

リソース・マネージャは、データベースでのハードウェア・リソース割当ておよびデータベース内の処理の優先順位付けの操作性を向上することによって、前述の問題に対応します。セッションをセッション属性に基づいてグループに分類し、これらのグループにリソースを割り当てることによって、ハードウェアの利用率を最適化できます。リソース・マネージャを使用して、PDBメモリー消費の制限を設定できます。

リソースは、データベース管理者が指定するリソース・プランに従ってユーザーに割り当てられます。リソース・プランは、リソース要件に従ってグループ化されたユーザー・セッションである、リソース・コンシューマ・グループ間でのリソースの分配方法を指定します。リソース・プラン・ディレクティブは、リソース・コンシューマ・グループにプランを関連付け、グループへのリソースの割当て方法を指定します。

図18-7に、昼間にOLTPアプリケーションとレポート・アプリケーションを同時に実行する組織で使用する簡単なリソース・プランを示します。現在、DAYTIMEのプランがアクティブになっており、3つのリソース・コンシューマ・グループ間でCPUリソースを割り当てます。具体的には、CPUタイムの75%がOLTPに、15%がREPORTSに、残りの10%がOTHER_GROUPSに割り当てられています。

図18-7 簡単なリソース・プラン



リソース・マネージャで解決できる問題の1つは、SQLモニターを使用して監視できるランナウェイ問合せです。リソース・マネージャを使用すると、ランナウェイSQL文を特定して対応するためのしきい値を指定できます。たとえば、CPU時間、経過時間、物理I/Oまたは論理I/Oのしきい値と、コンシューマ・グループの各前面プロセスによるPGA使用量を指定できます。リソース・マネージャは対応として、優先度の低いコンシューマ・グループへの切替え、SQL文の終了、またはしきい値違反のログ記録を行うことができます。

関連項目:

- リソース・マネージャの使用方法を学習するには、『*Oracle Database*管理者ガイド』を参照してください
- DBMS_RESOURCE_MANAGER PL/SQLパッケージの使用方法を学習するには、『*Oracle Database PL/SQL*パッケージおよびタイプ・リファレンス』を参照してください

Oracle Scheduler

Oracle Scheduler(スケジューラ)を使用すると、データベース管理者とアプリケーション開発者はデータベース環境で各種タスクが発生する時期と場所を制御できます。

スケジューラには、企業向けの複雑なスケジューリング機能が用意されており、次の操作を実行できます。

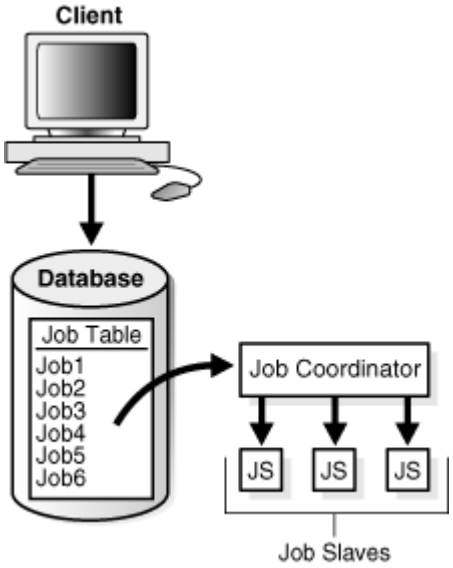
- 時間またはイベントに基づくジョブ実行スケジュール
- ビジネス要件をモデル化する方法によるジョブ・プロセスのスケジュール
- ジョブの管理と監視
- クラスタ化された環境におけるジョブの実行と管理

プログラム・オブジェクト(プログラム)には、引数のデフォルト値を含む、スケジューラで実行されるコマンドに関するメタデータが含まれます。スケジュール・オブジェクト(スケジュール)には、実行日時および反復パターンに関する情報が含まれます。ジョブ・オブジェクト(ジョブ)は、プログラムをスケジュールに対応付けます。実行するジョブおよびその時間を定義するには、プログラム、スケジュール、およびジョブ間の関係を割り当てます。

スケジューラは、DBMS_SCHEDULER PL/SQLパッケージ内の一連のファンクションおよびプロシージャとして実装されています。このパッケージまたはEnterprise Managerを使用して、スケジューラ・オブジェクトを作成および操作します。スケジューラ・オブジェクトは標準データベース・オブジェクトであるため、システム権限およびオブジェクト権限により、これらのオブジェクトへのアクセスを制御できます。

図18-8に、スケジューラの基本アーキテクチャを示します。ジョブ表はすべてのジョブのコンテナであり、データベースごとにあります。ジョブ・コーディネータ・バックグラウンド・プロセスは、必要に応じて自動的に起動および停止します。ジョブ・スレーブは、ジョブの実行時間になるとジョブ・コーディネータによって起動されます。スレーブはジョブ表からメタデータを収集し、ジョブを実行します。

図18-8 スケジューラのコンポーネント



関連項目:

- 「ジョブ・キュー・プロセス(CJQ0およびJnnn)」
- スケジューラについて学習するには、Oracle Database管理者ガイドを参照してください

パフォーマンスとチューニング

DBAは、対象とするOracle Databaseのパフォーマンスを適切に保つ責任を担っています。一般的に、パフォーマンス上の問題は、許容されないレスポンス時間(特定のワークロードを完了するためにかかる時間)またはスループット(特定の時間内で完了できる作業量)によって発生します。

一般的な問題には、次のものがあります。

- CPUのボトルネック
- メモリー構造のサイズ不足
- I/O容量の問題
- 効率の悪いまたは負荷が大きいSQL文
- SQL文のチューニング後の予期しないパフォーマンスの低下
- 同時実行性および競合に関連する問題
- データベース構成に関連する問題

チューニングの一般的な目標は通常、レスポンス時間の短縮、スループットの向上、またはその両方です。具体的に測定可能な目標の例としては、指定のSELECT文のレスポンス時間を5秒以下に短縮するなどがあります。この目標を達成できるかどうかは、DBAが制御できる要素とできない要素によって変わります。一般的に、チューニングとは、データベース・リソースを可能なかぎり効率的に使用することによって、具体的で測定可能、および達成可能なチューニング目標を達成するための作業のことです。

Oracleパフォーマンス・メソッドは、データベース内のボトルネックの特定と解消、および効率的なSQL文の開発に基づいています。Oracleパフォーマンス・メソッドには、次のタスクが含まれます。

- チューニング前の準備
- 定期的なデータベースの予防的チューニング

- ユーザーからのパフォーマンス上の問題報告に対応するデータベースのチューニング
- 負荷の高いSQL文の識別、チューニング、および最適化

この項では、アドバイザの使用など、Oracle Databaseパフォーマンス・チューニングの基本的な事柄について説明します。Oracle Databaseアドバイザは、領域、パフォーマンス、UNDO管理など、幅広い分野を網羅し、データベース管理に関係した主要な問題の対処方法について具体的なアドバイスを提供します。

関連項目:

Oracleパフォーマンス・メソッドの実施方法を学習するには、Oracle Database 2日でパフォーマンス・チューニング・ガイドおよびOracle Database SQLチューニング・ガイドを参照してください

データベースの自己監視

データベースは通常の処理を実行しながら、問題の発生時にすぐに問題を検知できるように自己を監視しています。Oracle Databaseでは、サーバー生成アラートを送信して、ユーザーに緊急の問題を通知できます。

アラートは問題が発生した場合、または毎秒の物理読取りやSQLレスポンス時間などのメトリックの予想値とデータが一致しない場合に自動的に生成されます。メトリックは、累積統計の変化率です。サーバー生成アラートは、ユーザー指定のしきい値に基づいて、またはイベントの発生に従って生成されるように設定できます。

サーバー生成アラートは問題を特定するのみでなく、報告した問題の解決方法も推奨します。たとえば、高速リカバリ領域の不足を通知するアラートでは、不要になった古いバックアップを削除するか、追加のディスク領域を追加するように推奨されます。

関連項目:

Oracle Database管理者ガイド

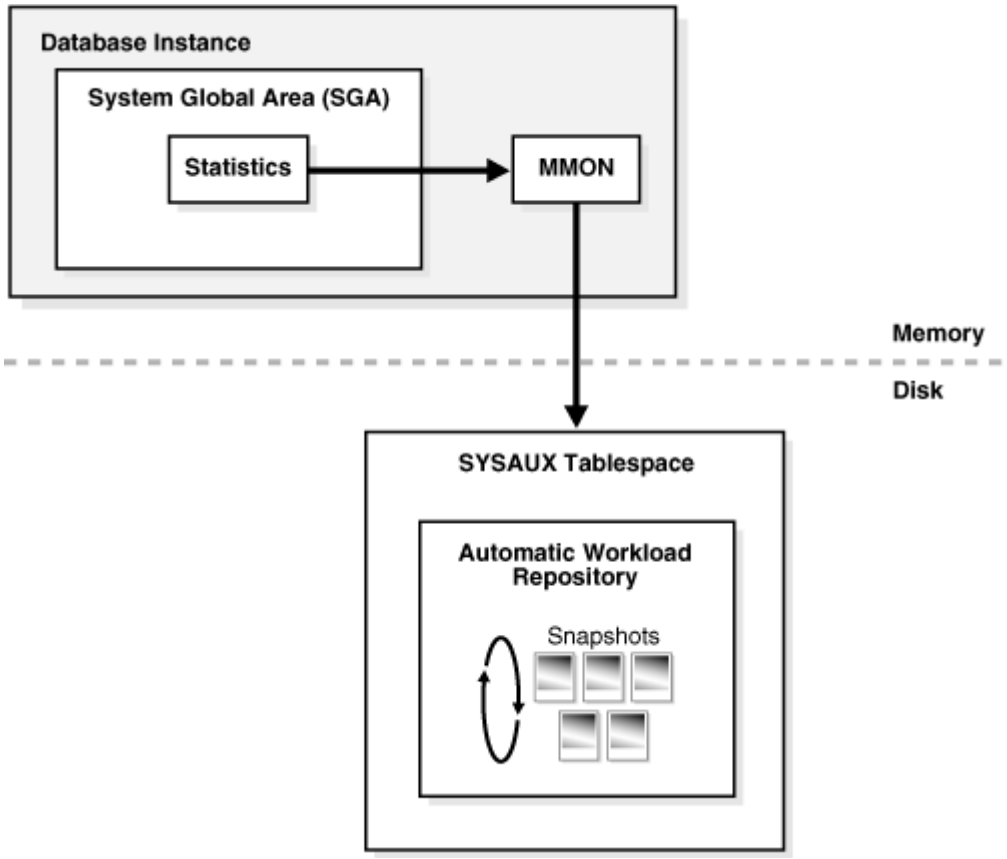
自動ワークロード・リポジトリ(AWR)

自動ワークロード・リポジトリ(AWR)は、システム、セッション、個々のSQL文、セグメント、およびサービスの累積統計などの履歴パフォーマンス・データを格納するリポジトリです。

AWR統計は、パフォーマンスのチューニングなどの基礎となります。AWRは問題の検出とチューニングに利用されるデータベース統計を自動的に集計し、データベースの自己管理において重要な役割を果します。

次の図に示すように、データベースは最新のAWR統計をSGAに格納します。デフォルトでは、MMONプロセスは、1時間ごとに統計を収集し、AWRスナップショットを作成します。AWRスナップショットは、ある時点で取得されたパフォーマンス統計のセットです。データベースは、スナップショットをSYSAUX表領域に書き込みます。AWRは、スナップショット領域を管理し、構成可能なスナップショット保存ポリシーに従って古いスナップショットをパージします。

図18-9 自動ワークロード・リポジトリ(AWR)



AWRベースラインは統計率の集合であり、通常、ピーク負荷時にシステムが良好に動作していたときに取得したものが使用されます。一対または一定の範囲のAWRスナップショットをベースラインとして指定できます。AWRレポートを使用して、パフォーマンスが低下したときに取得した統計とベースラインを比較することによって問題を診断できます。

AutoTaskと呼ばれる自動メンテナンス・インフラストラクチャは、Oracle DatabaseがどのようにAWRを使用して自己管理を行っているかを示します。AWRデータを分析することによって、AutoTaskはメンテナンス・タスクの必要性を判断し、必要なメンテナンス・タスクがOracle Schedulerメンテナンス・ウィンドウで実行されるようにスケジュールします。メンテナンス・タスクの例として、オプティマイザの統計の集計や自動セグメント・アドバイザの実行があります。

関連項目:

- 「管理性モニター・プロセス(MMONおよびMMNL)」
- 「SYSAUX表領域」
- AWRについて学習するには、『Oracle Databaseパフォーマンス・チューニング・ガイド』を参照してください
- 自動メンテナンス・タスクの管理方法を学習するには、『Oracle Database管理者ガイド』を参照してください

自動データベース・モニター(ADDM)

AWR (自動ワークロード・リポジトリ)で取得された統計を使用して、ADDMは、データベース・パフォーマンスを自動的および予防的に診断し、識別された問題の解決方法を決定します。ADDMは手動でも実行できます。

Automatic Database Diagnostic Monitor (ADDM)はOracle Databaseに組み込まれているセルフアドバイザです。

ADDMでは、コンポーネント間の共通判断尺度として時間を使用し、システムのパフォーマンスに対して総合的にアプローチします。ADDMは、最も時間のかかっているOracle Databaseの領域を特定します。たとえば、データベースが使用可能なデータベース・バッファの待機に長時間費やしていると想定します。ADDMは、単に症状を示すのではなく、問題を掘り下げて根本原因を特定し、その問題がOracle Database全体に及ぼす影響を報告します。プロセス中のオーバーヘッドは、最小限に抑え

られています。

ほとんどの場合、ADDMは解決策を推奨し、パフォーマンス向上の予想を数量化します。たとえば、ADDMはハードウェア、データベース構成、データベース・スキーマまたはアプリケーションを変更するよう推奨します。ADDMは、推奨を行う場合、時間的な利点を報告します。時間を尺度として使用することによって、問題と推奨の利点を比較検討できます。

潜在的なパフォーマンスの問題報告に加え、ADDMは、問題のないデータベース領域を示します。I/Oやメモリーなど、データベース・パフォーマンスに大きな影響を与えないサブコンポーネントは、初期段階で分類ツリーから除かれます。ADDMではこれらのサブコンポーネントがリストされるため、管理者はそれらの領域での処理による成果が少ないことを速やかに理解できます。

関連項目:

Oracle Database 2日でパフォーマンス・チューニング・ガイドおよびOracle Databaseパフォーマンス・チューニング・ガイド

アクティブ・セッション履歴(ASH)

アクティブ・セッション履歴(ASH)は、アクティブなデータベース・セッションを毎秒サンプリングし、データをメモリーおよび永続ストレージに書き込みます。ASHはデータベース自己管理フレームワークの重要な部分であり、パフォーマンス上の問題を診断する場合に役立ちます。

インスタンス・レベルで統計を集計する自動ワークロード・リポジトリ(AWR)とは異なり、ASHはセッション・レベルで統計を集計します。アクティブ・セッションは、CPUを使用しており、アイドル待機クラスのイベントを待機していないセッションです。

Enterprise ManagerまたはSQLスクリプトを使用してASHレポートを生成し、指定した期間に集計されたセッション統計を取得できます。ASHレポートは、次の目的に使用できます。

- 自動データベース診断モニター(ADDM)によって識別されない短期間のパフォーマンス上の問題の分析
- 時間、セッション、モジュール、処理、SQL IDなどの各種ディメンションまたはその組合せで範囲を絞った、または目標に応じたパフォーマンスの分析

たとえば、10:00 p.m.から10:02 p.m.の間にデータベースの速度が低下したという報告をあるユーザーから受け取ったとします。ただし、10:00 p.m.から11:00 p.m.までのAWRスナップショット間隔では、2分間のパフォーマンス低下は短すぎて、ADDMの結果には表示されません。ASHレポートは、このような一過性の問題の原因を特定する場合に役立ちます。

関連項目:

Oracle Database 2日でパフォーマンス・チューニング・ガイドおよびOracle Databaseパフォーマンス・チューニング・ガイド

アプリケーションとSQLチューニング

Oracle Databaseは、SQLのチューニング処理を完全に自動化します。

ADDMは、システム・リソースを大量に使用し、それが原因でパフォーマンス上の問題を引き起こすSQL文を識別します。さらに、CPUと共有メモリーの使用量の観点で上位を占めるSQL文は、AWRに自動的に格納されます。高い負荷のSQL文の識別は、自動的に実施されるため、人的な介入は不要です。

EXPLAIN PLAN文

EXPLAIN PLAN文などのツールを使用すると、オプティマイザが選択した実行計画を表示できます。

EXPLAIN PLANは、指定したSQLの問合せが現行のセッション内で実行された場合の、その問合せに対する問合せ計画を示します。その他のツールは、Oracle Enterprise ManagerとSQL*Plus AUTOTRACEコマンドです。

関連項目:

EXPLAIN PLANについて学習するには、『Oracle Database SQL言語リファレンス』を参照

オプティマイザ統計アドバイザ

オプティマイザ統計アドバイザは、現在の統計収集の状況、既存の統計収集ジョブの有効性、および収集された統計の品質を分析する診断ソフトウェアです。オプティマイザ統計アドバイザは、自動データベース診断モニター(ADDM)、SQLパフォーマンス・アナライザおよびその他のアドバイザと同じアドバイザ・フレームワークを使用します。

オプティマイザ統計アドバイザは、ベスト・プラクティスに依存する従来のアプローチに次の利点を提供します。

- わかりやすいレポートを提供します
- アプリケーション・コードに求められる変更を行わずに必要な修正を実装するスクリプトを提供します
- AUTO_STATS_ADVISOR_TASKという事前定義タスクをメンテナンス・ウィンドウで1日に1回実行します
- 手動によるタスクの作成および実行、データ・ディクショナリ・ビューでの結果および推奨事項の保存、タスク用のレポートの生成、および必要時の修正の実装を可能にするDBMS_STATSパッケージのAPIを提供します
- Oracle Enterprise Manager (Enterprise Manager)などの既存ツールと統合します

オプティマイザ統計アドバイザは、アドバイザがそのチェックを実行する際に使用するOracle提供の標準であるルールを維持します。ルールは現在の機能セットに基づくOracleベスト・プラクティスを体現します。ベスト・プラクティスがリリースのたびに変更される場合は、オプティマイザ統計アドバイザのルールも変更されます。このようにアドバイザは常に最新の推奨事項を提供します。

オプティマイザ統計アドバイザは統計収集プロセスを検査してから、ルール違反の結果のレポートを生成します。アドバイザが推奨を行い、その推奨に基づいてアクションを提案する場合、ユーザーはそのアクションを自動的に実装するか、編集可能かつ実行可能なPL/SQLスクリプトを生成することができます。

アドバイザ・タスクはメンテナンス・ウィンドウで自動的に実行されますが、オンデマンドでそれを実行することもできます。

関連項目:

- オプティマイザ統計アドバイザについてさらに学習するには、『Oracle Database SQLチューニング・ガイド』を参照
- DBMS_STATSパッケージについて学習するには、『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』を参照してください

SQLチューニング・アドバイザ

自動SQLチューニングのインタフェースはSQLチューニング・アドバイザで、これはシステム・メンテナンス期間中にメンテナンス・タスクとして自動的に実行されます。

それぞれの自動実行中、アドバイザはデータベース内の負荷の大きいSQL問合せを選択し、これらの問合せに対するチューニングの推奨設定を生成します。

SQLチューニング・アドバイザの推奨設定は、次のカテゴリに分類されます。

- 統計分析
- SQLプロファイリング
- アクセス・パス分析
- SQL構造分析

SQLプロファイルには、SQL文に固有の追加の統計が含まれており、これによりオプティマイザは改良した実行計画を作成できます。基本的には、SQLプロファイルが問合せを分析する方法です。アクセス・パス分析とSQL構造分析は、開発中のアプリケーションや社内の本番アプリケーションをチューニングする場合に役立ちます。

SQLチューニング・アドバイザの主要な利点は、外部ツールではなく、オプティマイザからソリューションが提供されることです。このため、チューニングは実行計画とSQLパフォーマンスを担当するデータベース・コンポーネントによって実行されます。チューニング処理では、SQL文の過去の実行統計が考慮され、その文に対するオプティマイザ設定がカスタマイズされます。

関連項目:

- 「オプティマイザの概要」
- 『Oracle Database 2日でパフォーマンス・チューニング・ガイド』および『Oracle Database SQLチューニング・ガイド』

SQLアクセス・アドバイザ

SQLアクセス・アドバイザを使用すると、SQL問合せ分析およびスキーマ・オブジェクトの最適化や問合せのチューニングの方法の決定に役立ちます。

SQLアクセス・アドバイザは、データ・アクセス・パスを最適化する方法についてのアドバイスを提供します。具体的には、パーティション、マテリアライズド・ビュー、索引、マテリアライズド・ビュー・ログを使用してデータベース・パフォーマンスを向上させる方法が推奨されます。

パーティションや索引などのスキーマ・オブジェクトは、複雑でデータ処理集中型の問合せを最適化する場合に必要不可欠です。ただし、これらのオブジェクトの作成と管理には、多大な時間と領域が必要になることがあります。SQLアクセス・アドバイザでは、指定されたワークロードに対応するデータ構造を推奨し、この推奨をパフォーマンス目標の達成に役立てることができます。

SQLアクセス・アドバイザは、ウィザードを使用してEnterprise Managerから実行するか、DBMS_ADVISORパッケージを使用して実行できます。DBMS_ADVISORは、任意のPL/SQLプログラムからコール可能な分析およびアドバイザ用のファンクションおよびプロシージャの集合です。

関連項目:

『Oracle Database 2日でパフォーマンス・チューニング・ガイド』および『Oracle Database SQLチューニング・ガイド』

SQL計画管理

SQL計画管理を使用してテスト済計画と検証済計画のみを実行することで、SQL実行計画を管理します。

SQL計画管理は、データベースで確認済の計画のみが使用されるように、オプティマイザで実行計画を自動的に管理できる予防的なメカニズムです。このメカニズムにより、承認された1つ以上の繰返し可能なSQL文の計画セットである、SQL計画ベースラインを構築できます。ベースラインの効果は、オプティマイザによる選択がベースライン内の確認済計画に限定されることです。

Oracle Database 12cから、データベースでは、適応SPMを使用できるようになりました。SPM展開アドバイザは、スケジュールされたメンテナンス・ウィンドウで毎日実行され、すべての承認されていない計画をランク付けし、ウィンドウの期間内にできるだけ多くの計画のテスト実行を行います。SPM展開アドバイザは、SQL計画ベースラインで最もコストの低い承認済計画を選択し、承認されていない各計画と比較します。承認されていない計画が既存の承認済の計画よりもパフォーマンスが高い場合、アドバイザが自動的にその計画を承認します。そうでない場合、アドバイザでは、その計画を未承認のままにして、最後の確認日を更新します。

関連項目:

- 「データベース管理者向けのツール」
- SQL計画管理について学習するには、『Oracle Database SQLチューニング・ガイド』を参照してください

19 データベース開発者の概念

Oracle Database開発者は、データベース・アプリケーションを作成およびメンテナンスします。この項では、データベース開発者の作業内容と使用可能な開発ツールの概要について説明します。

この項では、次の項目について説明します。

- データベース開発者の業務
- データベース開発者向けのツール
- データベース開発者のためのトピック

データベース開発者の業務

Oracle開発者は、Oracleテクノロジ・スタックを使用するアプリケーションのデータベース・コンポーネントの作成およびメンテナンスを担当します。

Oracle開発者は、新しいアプリケーションを開発するか、既存アプリケーションをOracle Database環境で動作するように変換します。このため、開発者はデータベース管理者と緊密に連携し、知識や情報を共有する必要があります。

Oracleデータベース開発者は、次のタスクを担当します。

- アプリケーションに必要なデータ・モデルの実装
- スキーマ・オブジェクトの作成
- データ整合性のルールの実装
- 新しい開発プロジェクトのプログラミング環境の選択
- サーバー側のPL/SQLまたはJavaサブプログラム、およびSQL文を使用するクライアント側の手続き型コードの記述
- 選択した開発ツールを使用したアプリケーション・インタフェースの作成
- グローバル・アプリケーション開発用のグローバリゼーション・サポート環境の構築
- 本番環境での開発、テスト、教育およびデプロイメント用の異なるデータベースへのアプリケーションのインスタンス化

関連項目:

- Oracle Database開発の概要およびGUIベースのチュートリアルは、『Oracle Database 2日で開発者ガイド』を参照
- データベース設計、開発者向けSQL、開発者向けPL/SQLなどのトピックの詳細は、『Oracle Database開発ガイド』を参照

データベース開発者向けのツール

Oracleでは、データベース・アプリケーションを開発するためのいくつかのツールを提供しています。この項では、一般的に使用されるいくつかの開発ツールについて説明します。

SQL Developer

SQL Developerは、データベース開発者がSQL*Plusを使用して基本的なタスクを編集および開発するのに便利な方法です。

SQL Developerは、Javaで記述された、SQL*Plusのグラフィカル版ツールで、SQLおよびPL/SQLの開発を支援します。標準的なデータベース認証を使用してOracleデータベース・スキーマに接続できます。SQL Developerでは、次のことを実行できます。

- スキーマ・オブジェクトの参照、作成、編集、削除
- SQL文の実行
- PL/SQLプログラム・ユニットの編集およびデバッグ
- データの操作およびエクスポート
- レポートの作成および表示

SQL Developerは、デフォルトのOracle Databaseインストールに含まれており、無償でダウンロードできます。

関連項目:

SQL Developerの使用方法を学習するには、『Oracle Database 2日で開発者ガイド』および『Oracle SQL Developerユーザーズ・ガイド』を参照

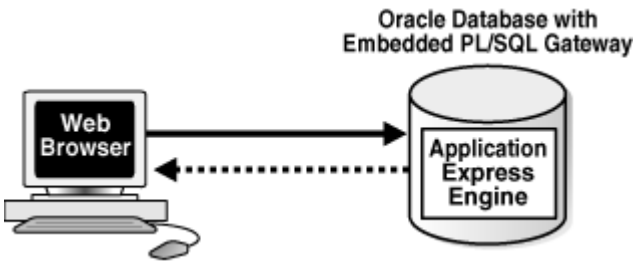
Oracle Application Express

Oracle Application Expressは、Oracle Database向けのWebアプリケーション開発ツールです。Oracle Application Expressのユーザー・インタフェースのテーマ、ナビゲーション・コントロール、フォーム・ハンドラ、柔軟なレポートなどの組込み機能を使用することで、効率よくアプリケーションを開発できます。

Oracle Application Expressはデータベースとともにインストールし、表形式のデータおよびPL/SQLコードで構成されています。アプリケーションを実行すると、Oracle Application Express PL/SQLコールに変換されたURLリクエストがブラウザから送信されます。データベースでPL/SQLが処理された後、その結果がHTMLとしてブラウザに送り返されます。このサイクルをページのリクエストまたは送信ごとに繰り返します。

Oracle Application Expressは、埋込み型PL/SQLゲートウェイと統合できます。このゲートウェイは、データベース内のOracle XML DB HTTPサーバーで実行され、動的アプリケーションの作成に必要なインフラストラクチャを提供します。図19-1に示すように、埋込み型PL/SQLゲートウェイは、中間層をなくすことで、シンプルなアプリケーション・アーキテクチャを実現します。

図19-1 埋込み型PL/SQLゲートウェイを使用したApplication Express



Oracle JDeveloper

Oracle JDeveloperは、最新の業界標準Java、XML、Webサービス、およびSQLを使用するサービス指向アプリケーションを構築するための統合開発環境(IDE)を提供します。

Oracle JDeveloperは、アプリケーションをモデル化、コーディング、デバッグ、テスト、プロファイリング、チューニングおよびデプロイするための機能を統合することによって、ソフトウェア開発全般を支援します。

Oracle JDeveloperでは、各種アプリケーション開発ツールのウィンドウを使用します。たとえば、Javaアプリケーションを作成する場合、Javaビジュアル・エディタやコンポーネント・パレットなどのツールを使用できます。これらのツールに加えて、Oracle JDeveloperでは、プロジェクト・コンテンツを整理および表示するための様々なナビゲーション機能を提供しています。

関連項目:

- JDeveloperの使用方法を学習するには、『Oracle Database 2日でJava開発者ガイド』を参照
- JDeveloperはURL http://www.oracle.com/technetwork/developer-tools/jdev/downloads/からダウンロードできます。

Oracle Developer Tools for Visual Studio

Oracle Developer Tools for Visual Studio .NETは、Visual Studio .NET環境に統合された一連のアプリケーション・ツールです。

Oracle Developer Tools for Visual Studio .NETのツールを使用すると、ユーザーはGUIを介してOracle機能にアクセスして、様々なアプリケーション開発タスクを実行でき、生産性および操作性が向上します。

Oracle Developer Toolsでは、Visual Basic、C#、およびその他の.NET言語を使用して.NETストアド・プロシージャをプログラミングおよび実装できます。これらのプロシージャは、.NET言語で記述され、SQLまたはPL/SQL文を含みます。

データベース開発者のためのトピック

この項では、他の箇所で説明されていないデータベース開発者に不可欠なトピックについて説明します。

この項では、次の項目について説明します。

- アプリケーション設計およびチューニングの原則
- クライアント側のデータベース・プログラミング
- グローバリゼーション・サポート
- 非構造化データ

アプリケーション設計およびチューニングの原則

Oracle開発者は、セキュリティおよびパフォーマンス目標を達成できるよう、データベース・アプリケーションを設計、作成、およびチューニングする必要があります。

アプリケーション設計およびチューニングに関する次の原則が、ガイドラインとして役立ちます。

- Oracle Databaseの機能の学習

  開発者としてOracle Databaseを使用するアプリケーションの開発にかかる時間を最小限に抑えるには、データベー

スのアーキテクチャおよび機能を利用する必要があります。たとえば、Oracle Databaseのデータ並行性制御およびマルチバージョン読取りの一貫性について理解していないと、アプリケーションでデータの整合性を損なったり、アプリケーションの実行が遅くなったり、スケーラビリティが低下するおそれがあります。トランザクション・ガードおよびアプリケーション・コンティニュイティの機能を理解すると、不要な例外処理コードを記述しなくて済みます。

- バインド変数の使用(使用しない妥当な理由がある場合を除く)

  問合せにバインド変数を使用すると、データベースは問合せを一度コンパイルした後で、問合せ計画を共有プールに格納します。同じ文を再度実行する場合、データベースはソフト解析を実行し、計画を再利用できます。これに対し、ハード解析は、時間がかかる上、多くのリソースを消費します。バインド変数を使用してソフト解析を可能にすることで、効率は大幅に向上し、また、データベースはそのように設計されています。

- クライアントではなく、データベース・サーバーでの整合性制約の実装

  主キーと外部キーを使用すると、複数のアプリケーションでデータを再利用できます。クライアント側で規則をコーディングすると、データベースに対する実行中、他のクライアントはこれらの規則にアクセスできません。

- 代表的なデータおよびセッション・アクティビティが利用可能なテスト環境の構築

  稼働中の本番環境をシミュレートするテスト環境の構築には、複数の利点があります。たとえば、アプリケーションのベンチマークを実施でき、拡張性およびパフォーマンスが十分かどうかを確認できます。また、テスト環境を使用して、データベースの変更によるパフォーマンスへの影響を測定したり、アップグレードおよびパッチが正常に動作するかどうかを確認できます。

- 高いパフォーマンス目標に沿ったデータ・モデルの設計

  通常、汎用データ・モデルを使用すると、パフォーマンスは低下します。データ・モデルは、最も一般的な問合せに最大限効率よく応答できるように綿密に設計します。たとえば、データ・モデルでは、最高のパフォーマンスを示す索引のタイプを使用する必要があります。論理および物理構造の変更は困難または不可能な場合があるため、デプロイ後のチューニングはお薦めしません。

- 明確なパフォーマンス目標の定義およびメトリックの履歴レコードの保管

  開発での重要な要素として、アプリケーションのパフォーマンスおよび拡張性の正確な判断があります。たとえば、予想されるユーザー負荷、1秒当たりのトランザクション、許容レスポンス時間などのメトリックを使用します。このため、パフォーマンス・メトリックの履歴レコードを保管しておくことをお薦めします。この方法で、事前および事後にパフォーマンスを監視できます。

- アプリケーション・コードのインスツルメント処理

  開発の際、アプリケーションにデバッグ・コードを追加することをお薦めします。トレース・ファイルが生成可能である場合、パフォーマンス上の問題をデバッグおよび診断するときに便利です。

関連項目:

- 「SQLの解析」
- 「データの同時実行性と整合性の概要」
- 「整合性制約の利点」
- データベース・アプリケーションの設計時の考慮事項は、『Oracle Database 2日で開発者ガイド』を参照してください
- パフォーマンス用アプリケーションの設計方法を学習するには、Oracle Database SQLチューニング・ガイドを参照してください。

クライアント側のデータベース・プログラミング

プリコンパイラまたはJavaトランスレータを使用してソース・コードにSQL文を配置したり、APIを使用してアプリケーションとデータベース間の対話を実現できます。

手続き型データベース・アプリケーションでSQLを使用するには、2つの基本テクニックがあり、1つはPL/SQLおよびJavaを使用したサーバー側のプログラミングで、もう1つはJava Database Connectivity (JDBC)やOracle Call Interface (OCI)などプリコンパイラおよびAPIを使用したクライアント側のプログラミングです。

関連項目:

プログラミング環境の選択方法を学習するには、Oracle Database開発ガイドを参照してください

確認するには、「サーバー側プログラミング: PL/SQLとJava」を参照してください

埋込みSQL

以前は、クライアント/サーバー・プログラムとデータベースの対話には埋込みSQLが使用されていました。

Oracleプリコンパイラ

一般的に、クライアント/サーバー・プログラムは、Oracle**プリコンパイラ**を使用して記述されます(プリコンパイラは、高度なプログラムにSQL文を埋め込むためのプログラミング・ツールです)。

たとえば、Oracle Pro*C/C++プリコンパイラを使用すると、CまたはC++のソース・ファイルにSQL文を埋め込むことができます。Oracleプリコンパイラは、COBOLおよびFORTRANにも使用できます。

プリコンパイラには、次のような複数の利点があります。

- 同等のOCIアプリケーションに比べて一般的にコードの記述量が少ないことによる生産性の向上
- 高度にカスタマイズされたアプリケーションを作成可能
- リソース使用、SQL文実行および各種のランタイム・インジケータの緊密な監視
- プリコンパイラにより各埋込みSQL文がOracle Databaseランタイム・ライブラリのコールへと自動変換されることによる時間の節約
- Object Type Translatorによる、Pro*C/C++アプリケーションで使用可能なCデータ型へのOracle Databaseのオブジェクト型およびコレクションのマップ
- オブジェクト型とコレクションのコンパイル時におけるタイプ・チェック機能、データベース型からCデータ型への自動型変換

SQL文を含むクライアント・アプリケーションがホスト・プログラムになります。このプログラムは、ホスト言語で記述されます。ホスト・プログラムでは、完全なSQL文と完全なC文を混合して使用でき、Cの変数または構造をSQL文に使用することもできます。SQL文を埋め込む場合は、文をEXEC SQLキーワードで開始して、セミコロンで終了する必要があります。EXEC SQL文は、Pro*C/C++によりランタイム・ライブラリSQLLIBへのコールに変換されます。

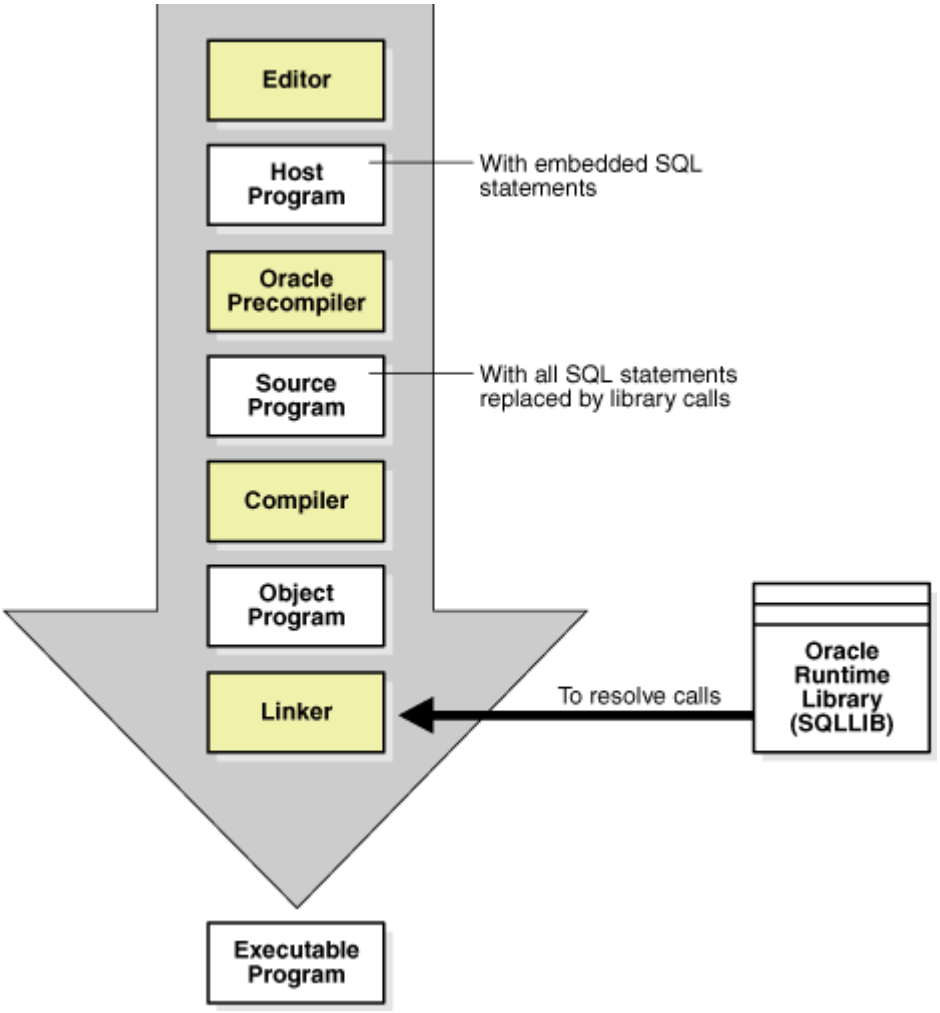
多くの埋込みSQL文は、新しい句が追加される点、またはプログラム変数が使用される点のみが同等の対話型SQLと異なります。次の例では、対話型と埋込み型のROLLBACK文を比較しています。

```
ROLLBACK;           -- interactive
EXEC SQL ROLLBACK;  -- embedded
```

これらの文は同じ働きをしますが、前者は対話型SQL環境(SQL Developerなど)で、また後者はPro*C/C++プログラムで使用します。

このプリコンパイラは入力としてホスト・プログラムを受け入れ、埋込みSQL文を標準のデータベース・ランタイム・ライブラリ・コールに変換し、通常の方法でコンパイル、リンクおよび実行できるソース・プログラムを生成します。図19-2に、プリコンパイラを使用したプログラム開発の典型的なステップを示します。

図19-2 プリコンパイラを使用したプログラム開発



関連項目:

Pro*C/C++プリコンパイラの詳細は、『Pro*C/C++プログラマーズ・ガイド』を参照してください。

SQLJ

SQLJは、SQL文をJavaソース・コードに埋め込むためのANSI SQL-1999規格です。SQLJは、クライアント側でJavaからSQLへのデータ・アクセスに使用されるJava Database Connectivity(JDBC)APIにかわるシンプルなAPIを提供します。

SQLJインタフェースは、Java版のPro*インタフェースです。SQL文をJavaソース・コードに挿入します。その後、Javaソース・ファイルをSQLJトランスレータに送信すると、SQLJトランスレータによって埋込みSQLが純粋なJDBCベースのJavaコードに変換されます。

関連項目:

「SQLJ」

クライアント側のAPI

現在は、多くの開発者がAPIを使用してデータベース・アプリケーションにSQLを埋め込みます。

たとえば、Oracle Databaseと通信するプログラムの有効化によく使用されるOpen Database Connectivity(ODBC)およびJDBCという2つのAPIがあります。その他、クライアント側のプログラミングによく使用されるOracle Call Interface(OCI)およびOracle C++ Call Interface(OCCI)という2つのAPIもあります。

OCIおよびOCCI

プリコンパイラのかわりとして、OracleではOCI APIおよびOCCI APIを提供しています。

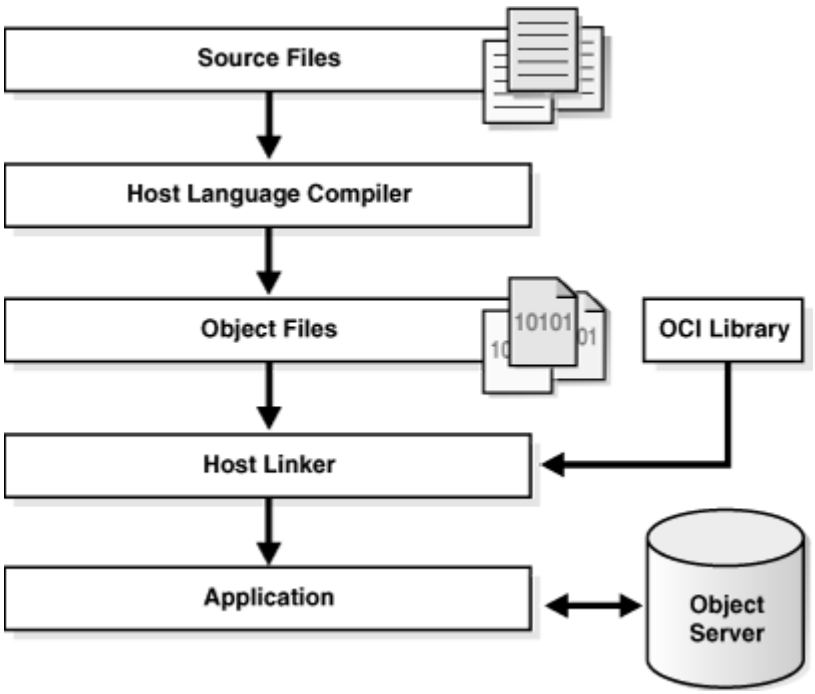
OCIでは、Cなどのホスト・プログラミング言語を使用して、データベース内のデータおよびスキーマを操作できます。OCCIは、オブジェクト指向のインタフェースで、C++を使用する場合に適しています。どちらのAPIを使用する場合も、開発者はシステム固有のサブプログラムを起動して、Oracle Databaseにアクセスし、SQLの実行を制御できます。

場合によっては、レベルの高いインタフェースよりも、OCIの方が高いパフォーマンスを示すことや、機能が豊富なことがあります。OCIおよびOCCIでは、次のような多くの機能を提供しています。

- Oracle Database全体で使用可能なすべてのSQL DDL、DML、問合せおよびトランザクション制御機能のサポート
- ディスク領域が問題になる場合のアプリケーションをデプロイする方法であるインスタント・クライアント
- Cアプリケーションからのスレッド管理、接続プール、グローバリゼーション・ファンクションおよびデータのダイレクト・パス・ロード

OCIおよびOCCIには、動的ランタイム・ライブラリ(OCILIB)という形式で、標準データベース・アクセスおよび検索関数のライブラリが提供されています。このライブラリは、実行時にアプリケーション内でリンクを設定できます。このように、非データベース・アプリケーションと同様の方法でOCIまたはOCCIプログラムをコンパイルおよびリンクすると、別の処理やプリコンパイル作業が不要になります。図19-3に、開発プロセスを示します。

図19-3 OCIまたはOCCIを使用した開発プロセス



関連項目:

- Oracle Call Interfaceプログラマーズ・ガイド
- Oracle C++ Call Interfaceプログラマーズ・ガイド

ODBCおよびJDBC

ODBCは、アプリケーションをデータベースに接続し、SQL文を準備して実行するための標準APIです。

ODBCは、プログラミング言語、データベース、およびオペレーティング・システムに依存しません。ODBCの目的は、任意のアプリケーションが任意のデータベースのデータにアクセスできるようにすることです。

データベース・ドライバは、アプリケーションとデータベースの間にあるソフトウェアです。このドライバは、アプリケーションによるAPIコールをデータベースが処理可能なコマンドへと変換します。ODBCドライバを使用することで、スプレッドシートに格納されているデータなど、あらゆるデータ・ソースにアプリケーションからアクセスできます。ODBCドライバは、ODBC標準とデータベースの間のマッピングをすべて実行します。

Oracleが提供するOracle ODBCドライバを使用すると、ODBCに準拠したアプリケーションはOracle Databaseにアクセスできます。たとえば、Visual Basicで記述されたアプリケーションでは、ODBCを使用してOracle Database表の問合せや更新を実行できます。

JDBCは、JavaアプリケーションがOracle Databaseと対話するための下位レベルのJavaインタフェースです。ODBCと同様、JDBCもベンダーに依存しないAPIです。Sun社により定義されるJDBC標準は、java.sqlインタフェースを介して実装されます。

JDBC標準では、個々のプロバイダが独自のJDBCドライバを使用して標準を実装および拡張できます。Oracleでは、クライアント側のプログラミング用に次のJDBCドライバを提供しています。

- JDBC Thinドライバ

  このPure Javaドライバは、Oracleクライアントがインストールされていないクライアント側にインストールします。これはプラットフォームに依存せず、アプレットおよびアプリケーションの両方で使用できます。

- JDBC OCIドライバ

  このドライバは、Oracleクライアントがインストールされているクライアント側にインストールします。これはアプリケーションでのみ使用できます。JDBC OCIドライバは、CおよびJavaの両方で記述されており、JDBCコールをOCIコールに変換します。

次のスニペットは、JDBC OCIドライバを使用してStatementオブジェクトを作成し、dual表に問合せを実行するJavaプログラムのスニペットです。

```
// Create a statement
Statement stmt = conn.createStatement();

// Query dual table
ResultSet rset = stmt.executeQuery("SELECT 'Hello World' FROM DUAL");
```

関連項目:

JDBCについてさらに学習するには、Oracle Database開発者ガイドおよびOracle Database 2日でJava開発者ガイドを参照してください

グローバリゼーション・サポート

Oracle Databaseグローバリゼーション・サポートを使用すると、ネイティブ言語でデータを格納、処理および取得できます。

グローバリゼーション・サポートを使用すると、世界中から同時にアクセスして実行できる多言語アプリケーションやソフトウェアの開発が可能になります。

グローバル・データベース・アプリケーションを記述する開発者は、次のことを実行する必要があります。

- Oracle Databaseグローバリゼーション・サポートのアーキテクチャ(各文字セット、地域、言語、言語ソート定義の特性など)の理解
- 中間層プログラミング環境のグローバリゼーション機能(データベースのロケール・モデルとの対話および同期方法など)の理解
- 異なるオペレーティング・システム、文字セット、およびロケール要件で実行されている複数のクライアントを同時にサポートするコードの設計と記述

たとえば、アプリケーションはネイティブ・ユーザーが使用する言語およびロケールのプリファレンスに基づいてユーザー・インタフェースの内容をレンダリングしたり、データを処理する必要がある場合があります。たとえば、アプリケーションは、ユーザーに設定の変更を求めることなく、マルチバイトの漢字データを処理し、該当する地域の形式でメッセージや日付を表示して、7ビットASCIIデータを処理する必要があります。

関連項目:

グローバリゼーションの詳細は、『Oracle Databaseグローバリゼーション・サポート・ガイド』を参照

グローバリゼーション・サポート環境

グローバリゼーション・サポート環境には、クライアント・アプリケーションとデータベースが含まれます。言語に依存する処理は、クライアントとサーバー上でパラメータおよび環境変数を設定することによって制御します(クライアントとサーバーは別々の場所に存在する場合もあります)。

ノート:

以前のリリースでは、Oracle ではグローバリゼーション・サポート機能は各国語サポート(NLS)機能と呼ばれていました。実際、NLS はグローバリゼーション・サポートの一種で、NLS を使用して、各国語を選択し、特定の文字セットでデータを格納できます。

Oracle Databaseでは、次の機能に対するグローバリゼーション・サポートを提供しています。

- ネイティブ言語および地域
- 日付、時間、数字、および通貨のローカル形式
- 暦法(グレゴリオ暦、和暦、元号暦、タイ仏教暦など)
- Unicodeなどの複数の文字セット
- キャラクタ・セマンティクス

文字セット

文字セットはグローバリゼーション・サポートの主要なコンポーネントであり、コンピュータ画面に文字を表示するときに使用されるコード体系です。

アプリケーション開発においては、次の違いが重要です。

- データベース文字セットによって、データベースで表現される言語が決定します。この文字セットは、データベースの作成時に指定します。

  ノート:

  データベースを作成した後で文字セットを変更すると、一般的に、時間およびリソースの面で大きなコストがかかります。この操作では、データベース全体のエクスポートおよび再インポートによる、すべての文字データの変換が必要になる場合があります。

- クライアント文字セットは、クライアント・アプリケーションによって入力および表示されるデータに使用される文字セットです。クライアントとデータベースの文字セットは異なる場合もあります。

文字グループ(アルファベット文字、表意文字、記号、句読点および制御文字など)は、文字セットとしてエンコードできます。エンコードされた文字セットでは、セット内の各文字について、コード・ポイントまたはエンコード値と呼ばれる一意の数値コードが割り当てられます。グローバル環境では、異なる文字セット間でデータの変換が必要になる可能性があるため、このコード・ポイントが重要になります。

コンピュータ業界では多くのエンコードされた文字セットが使用されています。これらのセットでは、使用可能な文字数、使用可能な文字および各文字に割り当てられているコード・ポイントなどが異なります。Oracle Databaseでは、各国の規格、国際規格およびベンダー固有のエンコードされた文字セット規格のほとんどすべてをサポートしています。

Oracle Databaseでは、次のクラスのエンコードされた文字セットをサポートしています。

- シングルバイト文字セット

  各文字が占めるバイト数は、1バイトです。7ビット文字セットの例には、US7ASCIIなどがあります。8ビット文字セットの例には、WE8DECなどがあります。

- マルチバイト文字セット

  1文字が2バイト以上を占めます。一般的に、マルチバイト文字セットは、アジア言語に使用されます。

- Unicode

  ユニバーサル・エンコードされた文字セットでは、1つの文字セットで任意の言語の情報を格納できます。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。

関連項目:

文字セットの移行について学習するには、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。

ロケール固有の設定

ロケールとは、システムまたはプログラムが実行されている、言語的および文化的な環境を表します。NLSパラメータは、クライアントとデータベースの両方におけるロケール固有の動作を指定します。

データベース・セッションでは、クライアントにかわって文を実行するとき、NLSの設定が使用されます。たとえば、データベースでは、クライアントの地域の書式で3桁区切りを使用します。一般的に、クライアント・ホストのNLS_LANG環境変数によって、サーバー・セッションとクライアント・アプリケーションの両方のロケールが指定されます。このプロセスは、次のとおりです。

1. クライアント・アプリケーションは起動すると、環境設定に従ってクライアントのNLS環境を初期化します。

   Oracle Developerアプリケーションの表示形式の設定など、ローカルで実行されるすべてのNLS操作にこれらの設定が使用されます。

2. クライアントは、データベースに接続すると、NLS_LANGで定義された情報をデータベースに伝達します。

3. データベース・セッションは、クライアントから伝達された設定に基づいてNLS環境を初期化します。

   クライアントにより設定が指定されていなかった場合、セッションは初期化パラメータ・ファイルの設定を使用します。データベースが初期化パラメータの設定を使用するのは、クライアントがNLS設定をまったく指定しなかった場合のみです。クライアントが一部のNLS設定を指定した場合、残りのNLS設定はデフォルトの設定になります。

クライアント・アプリケーションにかわって開始された各セッションは、他のセッションと同じまたは異なるロケールで実行されます。たとえば、一方のセッションではドイツ語ロケールが使用され、別のセッションではフランス語ロケールが使用される場合があります。また、各セッションで同じまたは異なる言語要件が指定されます。

次の表に、異なるNLS_LANG設定を使用する2つのクライアントを示します。ユーザーは、各ホスト上でSQL*Plusを起動して同じデータベースにhrとしてログオンし、同時に同じ問合せを実行します。各セッションの結果は、浮動小数点数に関するロケール固有のNLS設定のために異なります。

表19-1 ロケール固有のNLS設定

t	クライアント・ホスト1	クライアント・ホスト2
t0	$ NLS_LANG=American_America.US7 ASCII $ export NLS_LANG	$ NLS_LANG=German_Germany.US7ASCII $ export NLS_LANG
t1	$ sqlplus /nolog SQL> CONNECT hr@proddb Enter password: ******* SQL> SELECT 999/10 FROM DUAL; 999/10 ---------- 99.9	$ sqlplus /nolog SQL> CONNECT hr@proddb Enter password: ******* SQL> SELECT 999/10 FROM DUAL; 999/10 ---------- 99,9

関連項目:

NLS設定について学習するには、『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

Oracle Globalization Development Kit

Oracle Globalization Development Kit (GDK)には、包括的なプログラミングAPIが含まれています。

APIはJavaとPL/SQLのどちらでも使用でき、サンプル・コード、グローバル・アプリケーションの作成中に発生する設計、開発およびデプロイメント上の様々な問題に対処するドキュメントが含まれています。GDKは、開発プロセスを簡略化し、グローバル環境サポートに使用されるインターネット・アプリケーションの開発コストを削減します。

GDKは主に、GDK for JavaとGDK for PL/SQLの2つからなります。GDK for Javaは、Javaアプリケーションに対してグローバリゼーション・サポートを提供します。GDK for PL/SQLは、PL/SQLプログラミング環境に対してグローバリゼーション・サポートを提供します。これらの2つの要素で提供される機能は同じではありません。

関連項目:

Oracle Databaseグローバリゼーション・サポート・ガイド

非構造化データ

非構造化データは、小さな論理構造に細分化されていないデータです。

従来のリレーショナル・モデルは、単純な表に格納できる単純な構造化データを処理します。Oracle Databaseでは、標準構成要素に分解できない非構造化データもサポートしています。非構造化データには、テキスト、グラフィック・イメージ、ビデオ・クリップ、サウンド・ウェーブなどが含まれます。

Oracle Databaseには、非構造化コンテンツを処理するためのデータ型が組み込まれています。これらのデータ型は、データベースではシステム固有の型として表示され、SQLを使用して問い合せることができます。

Oracle DatabaseにおけるXMLの概要

Oracle XML DBは、高パフォーマンスのXML操作の格納と取得に関連する一連のOracle Databaseテクノロジです。Oracle XML DBは、SQLデータ・モデルとXMLデータ・モデルの相互運用が可能な方法でサポートするシステム固有のXMLを提供します。

Oracle XML DBは、アプリケーションで処理されるデータの一部または全部がXMLで表現されているJavaまたはPL/SQLアプリケーションに適しています。たとえば、多数のXML文書を収集、生成、検証および検索する必要があるアプリケーションなどに適しています。

Oracle XML DBでは、次のような多くの機能を提供しています。

- XML文書をデータベース内で表すことができ、SQLによるアクセスを可能にするシステム固有のXMLTypeデータ型
- XMLスキーマ、XPath、XQuery、XSLT、DOMなどのXML標準のサポート
- 高度に構造化されたデータから完全な非構造化データまで、すべての形式のXMLデータをサポートするXMLIndex

次の例では、XMLType型のorders表を作成します。

```
CREATE TABLE orders OF XMLType;
CREATE DIRECTORY xmldir AS path_to_folder_containing_XML_file;
INSERT INTO orders
  VALUES (XMLType(BFILENAME('XMLDIR','purOrder.xml'),NLS_CHARSET_ID('AL32UTF8')));
```

また、前述の例では、ホスト・コンピュータ上の物理ディレクトリに対するデータベース内の論理名となるSQLディレクトリ・オブジェクトも作成します。このディレクトリにXMLファイルが格納されます。次に、purOrder.xmlファイルからorders表にXMLコンテンツを挿入します。

Oracle XML Developer's Kit (XDK)には、XML文書(ファイルシステム上にあるかデータベース内にあるかは不問)の読取り、操作、変換および表示のための基本ビルディング・ブロックが用意されています。APIおよびツールは、Java、CおよびC++で使用できます。本番のOracle XDKには、商用再配布ライセンスが付属します。

例19-1 XMLType

```
CREATE TABLE orders OF XMLType;
CREATE DIRECTORY xmldir AS path_to_folder_containing_XML_file;
INSERT INTO orders
  VALUES (XMLType(BFILENAME('XMLDIR','purOrder.xml'),NLS_CHARSET_ID('AL32UTF8')));
```

関連項目:

- 『Oracle XML DB開発者ガイド』
- Oracle XML Developer's Kitプログラマーズ・ガイド

Oracle DatabaseにおけるJSONの概要

Oracle Databaseは、問合せや索引付けなど、JavaScript Object Notation (JSON)データに対するネイティブ・サポートを提供します。

この項では、次の項目について説明します。

- JSONとは
- JSONおよびXML
- JSONに対するネイティブ・データベース・サポート

JSONとは

JavaScript Object Notation (JSON)は、オブジェクト、配列およびスカラー・データを表すことができる、言語に依存しない、テキストベースのデータ形式です。JSONデータは様々なプログラミング言語で解析および生成できます。

一般に、JSONはデータ交換言語として機能します。多くの場合、構造化されたデータをシリアライズし、通常はサーバーとWebアプリケーション間でネットワークを介して交換するために使用されます。JSONは、HTML組込みJavaScriptで記述されたアプリケーションを実行するWebブラウザで主要なデータ形式です。

JavaScriptオブジェクトは、プロパティ名と関連付けられたJSON値の0個以上のペアの連想配列です。JSONオブジェクトは、JavaScriptオブジェクト・リテラルで、カンマで区切られた名前と値のペア、コロンで区切られた各ペアの名前と値が含まれ、中カッコで囲まれたプロパティ・リストとして記述されます。オブジェクト・プロパティはキーと呼ばれる場合があります。オブジェクト・プロパティの名前と値のペアは、オブジェクト・メンバーと呼ばれる場合があります。

例19-2 JSONオブジェクト

この例は、発注書を表すJSONオブジェクトを示しており、最上位のプロパティ名はPONumber、Reference、Requestor、User、Costcenter、ShippingInstruction、Special Instructions、AllowPartialShipmentおよびLineItemsです。

```
{ "PONumber"             : 1600,
  "Reference"            : "ABULL-20140421",
  "Requestor"            : "Alexis Bull",
  "User"                 : "ABULL",
  "CostCenter"           : "A50",
  "ShippingInstructions" : { "name"    : "Alexis Bull",
                             "Address": { "street"  : "200 Sporting Green",
                                          "city"    : "South San Francisco",
                                          "state"   : "CA",
                                          "zipCode" : 99236,
                                          "country" : "United States of America" },
                             "Phone" : [ { "type" : "Office", "number" : "909-555-7307" },
                                         { "type" : "Mobile", "number" : "415-555-1234" } ] },
  "Special Instructions" : null,
  "AllowPartialShipment" : false,
  "LineItems"            : [ { "ItemNumber" : 1,
                               "Part"       : { "Description" : "One Magic Christmas",
                                                "UnitPrice"   : 19.95,
                                                "UPCCode"     : 13131092899 },
                               "Quantity"   : 9.0 },
                             { "ItemNumber" : 2,
```

```
                        "Part"       : { "Description" : "Lethal Weapon",
                                         "UnitPrice"   : 19.95,
                                         "UPCCode"     : 85391628927 },
                        "Quantity"   : 5.0 } ] }
```

前述の例では、ほとんどのプロパティに文字列値があります。PONumber、zipCode、ItemNumberおよびQuantityには数値があります。Shipping InstructionsおよびAddressには値としてオブジェクトがあります。LineItemsには値として配列があります。

ノート:

JSON のより包括的な概要については、『Oracle XML DB 開発者ガイド』を参照してください

JSONおよびXML

一般に、JSONとXMLは両方ともデータ交換言語として使用されます。リレーショナル・データとは異なり、JSONデータとXMLデータは両方とも、データを定義するスキーマなしで、データベース内で格納、索引付けおよび問合せを行うことができます。

通常、JSONデータの方がXMLデータより、定義および機能が単純であるため、生成、解析および処理が簡単です。ユーザーにとっても、理解および使用することが簡単です。次の表では、JSONとXMLの相違点についてさらに説明します。

表19-2 JSONとXMLの相違点

機能	JSON	XML
単純な構造化データで役立つ	はい	はい。ただし、半構造化されたデータと複雑な構造化データもサポートします
複合コンテンツで役立つ	いいえ	はい
属性、ネームスペース、継承および置換が不足している	はい	いいえ
順序付けに重点を置く	いいえ	はい
主にデータではなくドキュメントを目的とする	いいえ	はい
日付データ型を含む	いいえ	はい

ノート:

XML および JSON の比較の詳細は、Oracle Database JSON 開発者ガイドを参照

JSONに対するネイティブ・データベース・サポート

JSONは、リレーショナル・データベース機能が不足している非SQLデータベースにも広範囲に格納されます。一方、Oracle Databaseは、トランザクション、索引付け、宣言的問合せおよびビューなどの機能を使用して、JSONをネイティブにサポートしています。

データベースに格納されたJSONデータには、他のデータベース・データへのアクセスと同じ方法(OCI、.NETおよびJDBCの使用を含む)でアクセスできます。SQLデータ型XMLTypeを使用して格納されるXMLデータとは異なり、JSONデータは、 VARCHAR2、BLOBまたはCLOBを使用して格納されます。Oracle SQLを使用して、JSONデータで次のような操作を実行できます。

- 非JSONリレーショナル・データとのJSONデータの結合
- SQL関数json_objectおよびjson_arrayを使用したリレーショナル・データからのJSONドキュメントの生成。
- SQL関数json_tableを使用したJSONデータのリレーショナル・フォーマットへの投影
- 列にJSONデータのみを格納するための、is_jsonによるチェック制約の作成。データベースは、このチェック制約を使用して、列がJSON固有の操作(簡略構文など)のためのJSONであることを確認します。
- PL/SQLオブジェクトとしてのJSONドキュメントの操作。
- Oracle JSONパス式を引数として受け入れ、ターゲットのJSONデータと照合するための、SQL関数json_queryおよびjson_valueの使用
- JSONデータの索引付け。
- 外部表に格納されたJSONデータの問合せ
- Oracle GoldenGateを使用した、JSONデータが含まれる列を持つ表のレプリケート。

テキストのJSONデータでは常にUnicode文字セット(UTF8またはUTF16)が使用されます。Oracle Databaseでは、JSONデータの解析および問合せ時にUTF8を内部で使用します。このような処理に入力されるデータまたはこのような処理から出力されるデータをUTF8とは異なる文字セットにする必要がある場合は、適切な文字セット変換が自動的に実行されます。

例19-3 JSON列を持つ表の作成、ロードおよび問合せ

この例では、JSON列po_documentを持つ表j_purchaseorderを作成し、単純なJSONデータを挿入します。一部のデータが省略されています(...)。

```
CREATE TABLE j_purchaseorder
   (id          RAW (16) NOT NULL,
    date_loaded TIMESTAMP WITH TIME ZONE,
    po_document CLOB
      CONSTRAINT ensure_json CHECK (po_document IS JSON));

INSERT INTO j_purchaseorder
  VALUES (SYS_GUID(),
          SYSTIMESTAMP,
          '{"PONumber"             : 1600,
            "Reference"            : "ABULL-20140421",
            "Requestor"            : "Alexis Bull",
            "User"                 : "ABULL",
            "CostCenter"           : "A50",
            "ShippingInstructions" : {...}
            "Special Instructions" : null,
            "AllowPartialShipment" : true,
            "LineItems"            : [...]}');
```

次の問合せでは、JSON列po_documentのオブジェクトに対するPONumberを抽出します。

```
SELECT po.po_document.PONumber FROM j_purchaseorder po;
```

ノート:

Oracle Database での JSON サポートのより包括的な概要については、*Oracle Database JSON 開発者ガイド*を参照

LOBの概要

ラージ・オブジェクト(LOB)データ型を使用すると、非構造化データの大きなブロックを、バイナリ形式または文字形式で格納し、操作できます。

ラージ・オブジェクト(LOB)データ型では、データに対して効率的、ランダム、かつピース単位のアクセスができます。

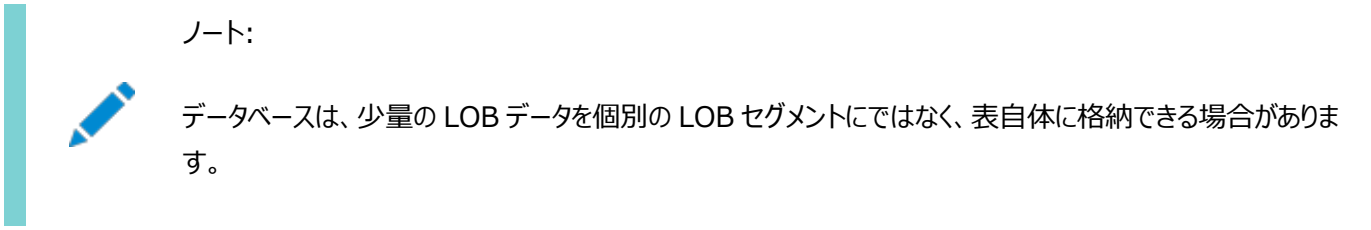
内部LOB

内部LOBは、データを外部ファイルではなく、データベース自体に格納します。

内部LOBは、次のとおりです。

- テキスト・ファイルまたはXMLファイルなどの大量のテキストをデータベース文字セットで格納するCLOB(文字LOB)
- Unicodeデータを格納するNCLOB(各国語文字セットLOB)
- 大量のバイナリ情報をビット・ストリームとして格納し、文字セットの変換を行わないBLOB(バイナリLOB)

データベースは、他のデータ型とは異なる方法でLOBを格納します。LOB列を暗黙的に作成すると、LOBセグメントおよびLOB索引が作成されます。LOBセグメントとLOB索引は常に一緒に格納され、これらを含む表領域は、表を含む表領域と異なる場合があります。



ノート:

データベースは、少量の LOB データを個別の LOB セグメントにではなく、表自体に格納できる場合があります。

LOBセグメントは、データをチャンクと呼ばれる単位に分割して格納します。チャンクは、論理的に連続するデータ・ブロックのセットであり、LOBに割り当てられる最小単位のデータです。表の行には、LOB索引を指す*LOB*ロケータと呼ばれるポインタが格納されます。表に問合せが実行されると、データベースはLOB索引を使用して、LOBチャンクの場所をすばやく特定します。

データベースは、他のデータと異なる方法でLOBセグメントの読取り一貫性を管理します。UNDOデータを使用して変更を記録するかわりに、データベースはビフォア・イメージをセグメント自体に格納します。トランザクションによってLOBに変更が加えられると、データベースは新しいチャンクを割り当て、古いデータを元の場所に残します。トランザクションがロールバックされると、データベースも古いチャンクを指す索引まで変更をロールバックします。

関連項目:

- 「ユーザー・セグメントの作成」
- 「読取り一貫性とUNDOセグメント」

外部LOB

BFILE (バイナリ・ファイル・ラージ・オブジェクトまたはLOB)は、外部ラージ・オブジェクトです。

BFILEは、データベースがオペレーティング・システム内のファイルを指定するポインタを格納するため、外部LOBです。外部データは、読取り専用です。

BFILEは、ディレクトリ・オブジェクトを使用してデータを検索します。使用領域量は、ディレクトリ・オブジェクト名の長さ、および

ファイル名の長さによって決まります。

バイナリ・ファイルはデータベース外のファイルのため、BFILEでは内部LOBと同じ読取り一貫性メカニズムを使用できません。ファイルのデータに変更が加えられると、再度同じバイナリ・ファイルから読み込んだ場合に結果が異なることがあります。

SecureFiles

SecureFiles LOB記憶域は、2つの記憶域タイプの1つです。もう1つのタイプはBasicFiles LOB記憶域です。

SECUREFILE LOBパラメータでは、圧縮と複製解除(拡張圧縮オプションの一部)、および暗号化(拡張セキュリティ・オプションの一部)を含む特定の拡張機能を使用できます。Oracle Database 12c以降では、SecureFilesはLOBのデフォルトの記憶域メカニズムです。

関連項目:

- 「Oracleデータ型」
- LOBデータ型についてさらに学習するには、『Oracle Database SecureFilesおよびラージ・オブジェクト開発者ガイド』を参照

Oracle Textの概要

Oracle Text (Text)は、Oracle Databaseに統合されている全テキスト検索テクノロジです。Oracle Textは、ファイル・システム、データベースまたはWeb上に格納されているすべてのドキュメントまたはテキスト形式のコンテンツに索引を付けます。これらのドキュメントは、テキスト形式のコンテンツ、メタデータまたは属性に基づいて検索できます。

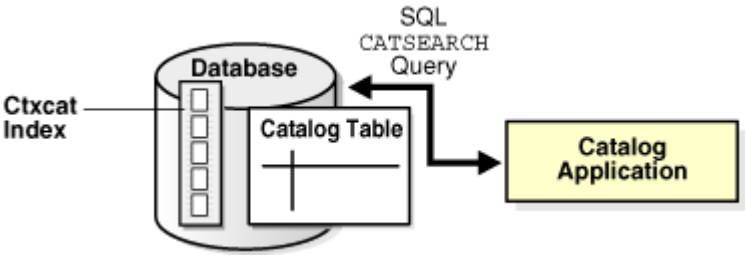
Oracle Textには、次の利点があります。

- Oracle Textを使用すると、テキスト検索と通常のデータベース検索を1つのSQL文にまとめることができます。Oracle Textの索引はデータベースにあり、Oracle Textの問合せはOracle Databaseプロセス内で実行されます。オプティマイザは、どのような問合せについても最善の実行計画を選択でき、テキストと構造化された基準を使用する非定型問合せに最高のパフォーマンスを提供します。
- Oracle Textは、XMLデータにも使用できます。特に、XMLIndexとOracle Textの索引機能を組み合せることで、XML索引と全テキスト索引の両方の利点を活用できます。
- Oracle TextのSQL APIにより、単純で直感的な方法でOracle Text索引を作成、メンテナンスしてテキスト検索を実行できます。

ユースケースとして、電子機器を販売するオークション・サイトのカタログ索引を作成する必要があるとします。新しい在庫が毎日追加されます。品物の説明、入札日、価格をともに格納する必要があります。アプリケーションは、複合問合せに対して適切な時間で応答する必要があります。最初に、catalog表を作成し、データを入力します。次に、Oracle Textを使用してCTXCAT索引を作成し、この索引は、SELECT ... WHERE CATSEARCH文のCATSEARCH演算子を使用した問合せが可能です。

図19-4に、catalog表とそのCTXCAT索引、およびCATSEARCH演算子を使用して索引を問い合せているカタログ・アプリケーションとの関係を示します。

図19-4 カタログ問合せアプリケーション



関連項目:

- 『Oracle Textアプリケーション開発者ガイド』および『Oracle Textリファレンス』
- XMLデータを対象とした全文検索の実行方法を学習するには、Oracle XML DB開発者ガイドを参照してください

Oracle Spatial and Graphの概要

Oracle Spatial and Graph (Spatial and Graph)には、空間データおよび空間分析と、物理グラフ・アプリケーション、論理グラフ・アプリケーション、ネットワーク・グラフ・アプリケーション、ソーシャル・グラフ・アプリケーションおよびセマンティック・グラフ・アプリケーション用の高度な機能が含まれています。

空間フィーチャは、Oracle Databaseへの空間フィーチャの集合の格納、検索、更新および問合せを容易にするスキーマおよびファンクションを提供します。空間データの一例は、ロード・マップです。空間データは、オブジェクトの地球上の位置(緯度や経度など)をマップ上に示します。地図のレンダリングでは、この空間データにより、オブジェクトの位置を2次元の紙面に投影します。地理情報システム(GIS)では、このような地表面と相関する空間データを格納、取出しおよびレンダリングができます。Oracleデータベースにこの空間データが格納されており、Spatial and Graphを使用すれば、このデータの操作および取得とこのデータの他のデータへの関連付けができます。

関連項目:

Oracle Spatial and Graph開発者ガイド

用語集

アクセス・ドライバ

外部表インフラストラクチャで、データベースに対する外部データを解析するAPI。アクセス・ドライバはデータベースの内部で動作し、データベースはアクセス・ドライバを使用して外部表のデータを読み取ります。

アクセス・パス

データベースからデータを取り出す方法。たとえば、索引を利用した問合せと全表スキャンを利用した問合せでは、使用されるアクセス・パスが異なります。

ACIDプロパティ

すべてのOracle Databaseトランザクションが従う必要がある、データベース・トランザクションの基本プロパティ。ACIDは、原子性(Atomicity)、一貫性(Consistency)、独立性(Isolation)および永続性(Durability)を略した頭字語です。

アクティブなオンラインREDOログ・ファイル

データベース・インスタンスのリカバリに必要なデータが含まれる場合があるオンラインREDOログ・ファイル。

アクティブ・セッション

CPUを使用していて、アイドル待機クラスのイベントを待機していないデータベース・セッション。

アクティブ・セッション履歴(ASH)

アクティブなデータベース・セッションを毎秒サンプリングし、データをメモリーおよび永続ストレージに書き込むデータベース自己管理フレームワークの一部。

アクティブなトランザクション

開始した後、まだコミットまたはロールバックされていないトランザクション。

適応問合せ最適化

適応オプティマイザが、実行計画に対する実行時の調整を行い、より優れたオプティマイザ統計を導くための追加情報を検出できるようにするための一連の機能。適応最適化は、最適な計画を生成するために既存の統計が十分でない場合に役に立ちます。

ADDM

自動データベース診断モニター。データベースがそれ自体のパフォーマンスを診断し、識別された問題の解決方法を決定できるようにするOracle Databaseインフラストラクチャ。

ADR

自動診断リポジトリ。ネットワークのトレースやロギングに関する情報などの情報を管理するための、ファイルベースの階層データストア。

ADRベース

ADRルート・ディレクトリ。ADRベースには、複数のADRホームを格納でき、それぞれのADRホームは、Oracle製品やコンポーネントのインスタンスごとにすべての診断データ(トレース、ダンプ、アラート・ログなど)を格納するルート・ディレクトリです。

ADRホーム

オラクル社の製品またはコンポーネント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。たとえば、共有記憶域とOracle ASMがあるOracle RAC環境では、各データベース・インスタンスと各Oracle ASMインスタンスにはそれぞれ独自のADRホームがあります。

拡張索引圧縮

ヒープ構成表でサポートされる一意索引および非一意索引の接頭辞圧縮の拡張機能。すべてのブロックに固定重複キー排除を使用する接頭辞圧縮とは異なり、拡張圧縮ではブロックごとに適応重複キー排除を使用します。

高度な行圧縮

任意のSQL操作によって操作されたデータを圧縮する、OLTPアプリケーション向けの表圧縮のタイプ。

基本表圧縮も参照してください。

集計関数

COUNTなど、行のグループに対して操作を行い、結果として単一の行を返す関数。

アラート・ログ

データベースのメッセージとエラーの履歴ログが記録されたファイル。アラート・ログはADRに格納されます。

分析関数

行のグループに対して操作を行い、結果として複数行を返す関数。

分析問合せ

ビジネス上の質問に答える仮定(what if)の問合せ通常、分析問合せには結合と集計が含まれ、比較的少量の出力を作成するために非常に大量の入力データをスキャンする必要があります。

分析ビュー

ファクト・データの集計、計算および結合をカプセル化するビューのタイプ。分析ビューは、ディメンション・モデルを使用してデータを編成します。これらを使用すると、集計および計算をデータ・セットに簡単に追加でき、比較的単純なSQLで問い合せることができるデータをビューに表示できます。

アンチ結合

述語の右側に対応する行を持たない述語の左側の行を戻す結合。

アプリケーション

アプリケーション・ルート内で、アプリケーションは共通ユーザーによって作成されるデータおよびメタデータの名前付けおよびバージョニングされたセットです。アプリケーションには、アプリケーション共通ユーザー、アプリケーション共通オブジェクトまたはそれら複数や組合せが含まれる場合があります。

アプリケーション・アーキテクチャ

データベース・アプリケーションがOracleデータベースに接続されているコンピューティング環境。データベースの最も一般的なアーキテクチャは、クライアント/サーバーと複数層の2種類です。

アプリケーション・コンテナ

アプリケーション・ルートに接続されたアプリケーションPDBの名付けられたセット。アプリケーション・コンテナにはアプリケーション・シードが含まれる場合があります。

アプリケーション・コンテキスト

指定したネームプレース内の属性の名前/値ペア。アプリケーションでは、データベースでアクションを実行する前に、様々なコンテキストを設定します。

アプリケーション・コンティニュイティ

データベース・セッションを使用不可にするリカバリ可能なエラーの後に、データベースに対するリクエストを中断せず、迅速な方法でリプレイできるようにする機能。

アプリケーション・ドメイン索引

アプリケーション固有のカスタマイズされた索引。

アプリケーションPDB

アプリケーション・コンテナに接続されたPDB。

アプリケーション・ルート

アプリケーション・コンテナ内のルート・コンテナ。各アプリケーション・コンテナにはアプリケーション・ルートが1つずつあります。アプリケーション・ルートは、共通オブジェクトが含まれる可能性があるためCDBルートと一部の特性を共有し、CREATE PLUGGABLE DATABASE文で作成されるためPDBと一部の特性を共有します。

アプリケーション・サーバー

クライアントと1つ以上のデータベース・サーバーの間のインタフェースを提供し、アプリケーションをホストするソフトウェア。

アーカイブ圧縮

COLUMN STORE COMPRESS FOR ARCHIVEで指定されるハイブリッド列圧縮。このタイプを使用すると、COLUMN STORE COMPRESS FOR QUERYより高い圧縮率が得られます。長期間格納するデータを圧縮する場合に有効です。

アーカイブREDOログ・ファイル

Oracle Databaseによってアーカイブされたオンライン REDOログのメンバー。メディア・リカバリ時にアーカイブREDOログ・ファイルをデータベース・バックアップに適用できます。

ARCHIVELOGモード

オンライン REDOログをアーカイブできるデータベースのモード。

アーカイバ・プロセス(ARCn)

オンラインREDOログ・ファイルをアーカイブするバックグラウンド・プロセス。

アーカイブ

アーカイブREDOログ・ファイルを生成する処理。

昇順索引

データが昇順に格納される索引。デフォルトでは、文字データは値の各バイトに含まれるバイナリ値に従って順序付けられ、数値データは最小の数から最大の数へ、日付は古い値から新しい値への順になります。

属性クラスタ表

ディスク上の近接するデータを、ユーザーが指定したクラスタリング・ディレクティブに基づいて格納するヒープ構成表です。

監査証跡

監査レコードを保管する場所。

自動データベース診断モニター(ADDM)

「ADDM」を参照してください。

自動診断リポジトリ(ADR)

「ADR」を参照してください。

自動メモリー管理

Oracle DatabaseがSGAメモリーとインスタンスPGAメモリーを完全に自動的に管理するモード。

自動セグメント領域管理(ASSM)

空リストのかわりにビットマップを使用してセグメント領域を管理する記憶領域管理方法。

自動UNDO管理モード

専用UNDO表領域のUNDO領域が自動的に管理されるデータベースのモード。

「手動UNDO管理モード」を参照してください。

自動ワークロード・リポジトリ(AWR)

「AWR」を参照してください。

自律型トランザクション

メイン・トランザクションと呼ばれる他のトランザクションからコールできる独立したトランザクション。

AWR

自動ワークロード・リポジトリ(AWR)。すべてのOracleデータベースに組み込まれているリポジトリ。Oracle Databaseでは、定期的に、稼働統計とワークロード情報のスナップショットが作成され、作成されたスナップショットがAWRに格納されます。

AWRベースライン

通常、ピーク負荷時にシステムが良好に動作していたときに取得した統計率の集合。

AWRスナップショット

ある時点でAWRに取得されたパフォーマンス統計のセット。

Bツリー索引

ツリーを上下反転させたような形状の索引。Bツリー索引には、検索用のブランチ・ブロックと、値を格納するリーフ・ブロックの2種類のブロックがあります。リーフ・ブロックには、すべての索引付きデータ値と、実際の行を検索するための対応するROWIDが含まれています。Bはbalanced(バランスが取れた)の略で、これは、すべてのリーフ・ブロックが自動的に同じ深さになるためです。

バックグラウンド・プロセス

クライアント・プロセスごとに実行される複数のOracleプログラムによって処理される関数を統合するプロセス。バックグラウンド・プロセスは、I/Oを非同期的に実行し、他のOracleプロセスを監視します。

「データベース・インスタンス」、「Oracleプロセス」も参照してください。

バックアップ

データのコピー。バックアップには、データファイル、サーバー・パラメータ・ファイル、制御ファイルなど、データベースの重要な部分を含めることができます。

バックアップ・ピース

バックアップ・セットの最小単位。

バックアップ・セット

1つ以上のデータファイル、アーカイブREDOログ・ファイル、制御ファイル、または1つのサーバー・パラメータ・ファイルからのデータが格納される独自のRMANバックアップ形式。

基本表圧縮

バルク・ロード操作向けの表圧縮のタイプ。基本表圧縮を実行するには、ダイレクト・パスのINSERT操作、ALTER TABLE . . . MOVE操作または表のオンライン再定義を使用する必要があります。

大規模表キャッシュ

従来のLRUベースでブロック・レベルの置換アルゴリズムではなく温度ベースでオブジェクト・レベルの置換アルゴリズムを使用する、データベース・バッファ・キャッシュの統合された部分(オプション)です。

bigfile表領域

1つの非常に大きなデータファイルまたは一時ファイルを含む表領域。

バインド変数

SQL文の中のプレースホルダで、SQL文を正常に実行するために、有効な値または値のアドレスと置換される必要があります。バインド変数を使用すると、実行時に入力データまたはパラメータを受け取るSQL文を作成できます。次の例は、v_empidをバインド変数として使用した問合せです。

```
SELECT * FROM employees WHERE employee_id = :v_empid;
```

ビットマップ索引

各索引キーのビットマップがROWIDのリストのかわりにデータベースに格納されるデータベース索引。

ビットマップ結合索引

2つ以上の表を結合するためのビットマップ索引。

ビットマップ・マージ

ビットマップ索引スキャンで取得したビットマップをマージする操作。たとえば、gender列とDOB列にビットマップ索引が設定されていて、問合せの述語がWHERE gender='F' AND DOB > 1966である場合、データベースでビットマップ・マージを使用できます。

ブロック破損

認識されているOracle形式ではないか、または内容に内部的な一貫性がないデータ・ブロック。

ブロック・ヘッダー

データ・ブロックの一部分で、ブロックのタイプやブロックのアドレスの情報が含まれます。また、トランザクション情報が含まれている場合もあります。

ブロック・オーバーヘッド

データ・ブロック内の領域で、ブロックの管理に必要なメタデータが格納されています。オーバーヘッドには、ブロック・ヘッダー、表ディレクトリおよび行ディレクトリが含まれています。

ブランチ・ブロック

Bツリー索引で、データベースが検索に使用するブロック。リーフ・ブロックが索引エントリを格納します。Bツリー索引の上位ブランチ・ブロックには、下位レベルの索引ブロックを指す索引データが含まれます。

バッファ

データベース・バッファ・キャッシュ内のメイン・メモリー・アドレス。バッファには、ディスクから読み取られた最新のブロックおよび最近使用されたブロックがキャッシュされます。データベースで新しいブロックが必要になったときは、古いデータ・ブロックを新しいデータ・ブロックと置換できます。

バッファ・キャッシュ・ヒット率

要求されたブロックがバッファ・キャッシュ内に見つかり、ディスクからの読取りが不要である確率を示している単位。

バッファ・ヘッダー

バッファに関するメタデータを格納するメモリー構造。

バッファ・プール

SGAでのバッファの集まり。

ビジネス・インテリジェンス

ビジネスにおける意志決定を支援するために組織内の情報を分析すること。

バイト・セマンティクス

文字列を一連のバイトとして取り扱うこと。文字列へのオフセットと文字列長はバイトで表されます。

キャッシュ・リカバリ

インスタンス・リカバリの自動フェーズ。Oracle Databaseにより、コミット済かどうかを問わず、オンラインREDOログ内のすべての変更が、対象となるデータ・ブロックに適用されます。

カーディナリティ

表の行数に対する個別値の数の比率。100万行を持つ表内で個別値が2つしかない列の場合、カーディナリティは低くなります。

デカルト結合

1つ以上の表が文中に他の表への結合条件を持たない結合。オプティマイザは、データ・ソースにあるすべての行を、2つのセットのデカルト積を作成しながら、別のデータ・ソースにあるすべての行と結合します。

CDB

最低1つのPDBを含むOracle Databaseインストール。PDBは、Oracle Netクライアントに従来のOracleデータベースとして表示されます。すべてのOracleデータベースは、CDBか非CDBのいずれかです。

CDB管理者

CDBを管理するデータベース管理者。PDB管理者は、CDB内の個々のPDBを管理します。

CDBフリート

1つの論理CDBとして管理できる様々なCDBの集合。

CDBリストア・ポイント

CDBで、ルートへの接続時、およびFOR PLUGGABLE DATABASE句が指定されていないときに作成されるリストア・ポイント。PDBリストア・ポイントと異なり、CDBリストア・ポイントはすべてのPDBによって使用可能です。

CDBルート

マルチテナント・コンテナ・データベース(CDB)において、すべてのPDBが属するスキーマ、スキーマ・オブジェクトおよび非スキーマ・オブジェクトの集合。各CDBには、PDBの管理に必要なシステム・メタデータが格納されるルート・コンテナが1つずつあります。すべてのPDBはCDBルートに属します。

文字エンコーディング

指定されたレパートリの各文字とコード単位を組み合せて、データ記憶域を可能にするコード。

キャラクタ・セマンティクス

文字列を一連の文字として取り扱うこと。文字列へのオフセットと文字列長は文字(文字コード)数で表されます。

文字セット

コンピュータ画面に文字を表示するために使用するコード体系。

チェック制約

指定した条件がすべての行で真または不明であることを必要とする、列または列のセットに対する制約。

チェックポイント

1. チェックポイント位置を表すデータ構造で、チェックポイント位置は、インスタンス・リカバリの開始位置を示すREDOスレッド内のSCNです。チェックポイントは、制御ファイルと各データファイルのヘッダーに記録される重要なリカバリ要素です。

2. データベース・バッファ・キャッシュ内のディスクに書き込まれていないデータ・ブロックをディスクに書き込むこと。データベース・ライター(DBW)・プロセスは、ブロックをディスクに書き込み、バッファ・キャッシュとデータファイルを同期します。

チェックポイント・プロセス(CKPT)

チェックポイント情報により制御ファイルおよびデータファイル・ヘッダーを更新し、DBWに対してブロックをディスクに書き込むように信号を送るバックグラウンド・プロセス。

子カーソル

親カーソルにテキストが格納されている文の計画、コンパイル環境およびその他の情報が含まれるカーソル。親カーソルの番号は0、最初の子の番号は1、のようになります。子カーソルは親カーソルとまったく同じSQLテキストを参照しますが、異なります。たとえば、テキストSELECT * FROM mytableを含む2つの文は、別のスキーマのmytableという名前の表を参照するときに、別のカーソルを使用します。

循環再利用レコード

必要に応じて上書き可能な、重要度が低い情報が含まれる制御ファイル・レコードの一種。使用可能なレコード・スロットがすべて使用されると、データベースは、制御ファイルを拡張して新規レコード用の領域を作成するか、または最も古いレコードを上書きします。

クライアント

クライアント/サーバー・アーキテクチャにおいて、ユーザーと対話するフロントエンドのデータベース・アプリケーション。クライアント部分ではデータをアクセスしません。

クライアント文字セット

クライアント・アプリケーションで入力または表示されるデータの文字セット。クライアントとデータベースの文字セットは異なる場合もあります。

クライアント・プロセス

アプリケーションまたはOracleツールのコードを実行するプロセス。ユーザーがSQL*Plusなどのクライアント・アプリケーションを実行すると、オペレーティング・システムではクライアント・プロセスを作成し、アプリケーションを実行します。

関連項目:「Oracleプロセス」

クライアント/サーバー・アーキテクチャ

2つのCPU間で処理を分割するソフトウェア・アーキテクチャ。1つのCPUは、トランザクションでクライアントとして機能し、サービスを要求して受け取ります。もう1つはトランザクションでサービスを提供するサーバーとして機能します。

クラスタ・ファイル・システム

一貫性と高パフォーマンスをクライアントに提供するために連携するサーバーのクラスタによる分散ファイル・システム。

クラスタ索引

クラスタ・キーのBツリー索引。

クラスタ・キー

表クラスタで、クラスタ化表で共有される1つ以上の列。たとえば、employeesおよびdepartments表はdepartment_id列を共有しています。表クラスタの作成時と表クラスタに追加されるすべての表の作成時にクラスタ・キーを指定します。

コールド・バッファ

データベース・バッファ・キャッシュで、最近使用されていないバッファ。

列

表内の垂直方向の領域で、データ・ドメインを表します。表の定義には、表名と列の集合が含まれています。各列には名前とデータ型があります。

列形式

インメモリー列ストアにあるオブジェクト用の列ベースの形式。列形式は、データベースでオブジェクトをデータベース・バッファ・キャッシュとデータ・ファイルに格納するために使用する行形式と対比されます。

コミット

データベース・トランザクションを終了し、トランザクションで実行されたすべての変更を永続的にするアクション。

コミット・クリーンアウト

コミット後に、ロックに関するトランザクション情報(ITLエントリ)がブロックから自動的に削除されること。コミットされたトランザクションのデータが含まれている変更済ブロックが引き続きSGA内に存在し、これを変更する他のセッションがない場合のみ、データベースによってこのITLエントリが削除されます。

共通ロール

マルチテナント・コンテナ・データベース(CDB)のすべてのコンテナに存在するロール。

完全リフレッシュ

マテリアライズド・ビューを定義する問合せの実行。マテリアライズド・ビューで事前作成表を参照するか、ユーザーが表をBUILD DEFERREDとして定義しないかぎり、完全リフレッシュは、マテリアライズド・ビューを最初に作成したときに実行されます。

複合データベース操作

単一のデータベース・セッションでの2つの時点間のアクティビティ。

コンポジット索引

表の複数の列の索引。

コンポジット・パーティション化

1つのデータ配分方法で表をパーティション化し、別のデータ配分方法を使用して、各パーティションをさらに小さなサブパーティションに分割するパーティション化計画。

コンポジット一意キー

一意キー制約のある2つ以上の列のセット。

複合トリガー

複数のタイミング・ポイントで起動できるトリガー。たとえば、トリガー実行文を実行する前と後の両方に、複合トリガーを起動する場合があります。

圧縮ユニット

ハイブリッド列圧縮で、行の集合を格納する論理的な構造。表にデータをロードする場合、データベースでは、行のグループは列形式で、それぞれの列の値としてまとめて圧縮されて格納されます。データベースでは、一連の行の列データが圧縮された後、そのデータが圧縮ユニットに収められます。

連結索引

「コンポジット索引」を参照してください。

条件

SQL文内の1つ以上の式および論理演算子で、TRUE、FALSEまたはUNKNOWNの値を戻します。たとえば、1=1という条件は、常にTRUEとなります。

書込みの競合

コミット読取りトランザクションにおいて、コミットされていない同時トランザクションによって更新された行を変更しようとしたときに発生する状況。

接続

クライアント・プロセスとOracleデータベース・インスタンスとの間の通信経路。

「セッション」も参照してください。

接続プーリング

共有サーバーとの制限された数のプロトコル接続を介してセッションの数を最大化する、リソース使用率とスケーラビリティのための機能。

一貫性バックアップ

メディア・リカバリを実行せずにRESETLOGSオプションを指定してオープンできるデータベース全体のバックアップ。データベース全体の一貫性バックアップでは、その性質から、REDOが一貫して適用されていることを必要としません。

関連項目:「非一貫性バックアップ」

読取り一貫性取得

特定のSCNと一貫性のある、データベース・バッファ・キャッシュ内のブロックのバージョン取得(読取り一貫性の一部)。データベースで問合せに対応するためにブロックが必要とされる場合、そしてデータベース・バッファ・キャッシュ内のブロックが正しいSCNと一貫性がない場合は、UNDOデータからの正しいバージョンのブロックの取得がデータベースで試行されます。

コンテナ

マルチテナント・コンテナ・データベース(CDB)で、ルートまたはPDBのいずれか。

コンテキスト

アプリケーションによって定義された、アプリケーションを検証および保護する一連の属性。CREATE CONTEXTというSQL文により、コンテキストの名前空間が作成されます。

制御ファイル

データベースの物理構造を記録し、REDOログ・ファイルの名前と位置、データベース作成のタイムスタンプ、現行のログ順序番号、チェックポイント情報などを含むバイナリ・ファイル。

キューブ

同一のディメンションおよびその他の共通する特性を持つ、メジャーの編成。キューブのエッジにはディメンション・メンバーが含まれ、キューブの本体にはデータ値が含まれています。

現在のモード取得

現在バッファ・キャッシュ内に存在するとおりのデータ・ブロックのバージョン取得(読取り一貫性は使用しません)。現在のモードで存在するブロックのバージョンはどの時点でも1つだけです。

現行のオンラインREDOログ・ファイル

ログ・ライター(LGWR)・プロセスがアクティブに書込みを行っているオンラインREDOログ・ファイル。

カーソル

PGAのプライベートSQL領域のハンドルまたは名前。カーソルはプライベートSQL領域と密接に関連しているため、これらの用語

は同じ意味で使用されることがあります。

データ・ブロック

Oracle Database内にあるデータ記憶域の最小論理単位。データ・ブロックは、Oracleブロックまたはページとも呼ばれます。1つのデータ・ブロックは、ディスク上の特定のバイト数の物理領域に対応します。

関連項目:「エクステント」、「セグメント」

データ同時実行性

多数のユーザーによる同一のデータへの同時アクセス。マルチユーザー・データベース管理システムには、データが正しく更新または変更され、データ整合性が維持されるように、適切な同時実行性制御が必要です。

関連項目:「データ整合性」

データ整合性

マルチユーザー・データベースの各ユーザーごとのデータの一貫したビュー。

「データ同時実行性」も参照してください。

データ破損

ハードウェア、ソフトウェアまたはネットワーク・コンポーネントによって、破損したデータが読み取られた場合、または書き込まれた場合に発生するエラー。

データ・ディクショナリ

データベース、データベースの構造およびそのユーザーについての参照情報を含むデータベース表およびビューの読取り専用の集合。

データ・ディクショナリ・キャッシュ

共有プール内でデータ・ディクショナリ情報を保持するメモリー領域。バッファはデータ・ブロック全体を保持しますが、データ・ディクショナリ・キャッシュはデータをバッファではなく行として保持するため、行キャッシュとも呼ばれます。

データ・ディクショナリ・ロック(DDLロック)

DDL操作によるスキーマ・オブジェクトの処理中または参照中に、そのオブジェクトの定義を保護するロック。Oracle Databaseは、DDLロックを必要とするDDLトランザクションのために、DDLロックを自動的に取得します。ユーザーはDDLロックを明示的に要求できません。

データ・ディクショナリ・ビュー

データ・ディクショナリの表または他のビューの事前定義済ビュー。データ・ディクショナリ・ビューは、接頭辞DBA_、ALL_またはUSER_で始まります。

データ・ファイル

Oracle Databaseによってディスク上に作成される物理ファイルであり、データベースのデータが含まれます。データファイルは、オペレーティング・システムのファイルシステムまたはOracle ASMディスク・グループのいずれにも配置できます。

データ整合性

受入れ可能なデータに関する規格を指定するビジネス・ルール。ビジネス・ルールは整合性制約およびトリガーを使用してデータベースに適用され、無効なデータ入力を防止します。

データ・リンク

PDBにおいて、ルート内のデータ(メタデータではない)を指し示す内部メカニズム。たとえば、AWRデータはルートに常駐します。各PDBがオブジェクト・リンクを使用してルート内のAWRデータを指し示すことにより、DBA_HIST_ACTIVE_SESS_HISTORYやDBA_HIST_BASELINEなどのビューが個別のコンテナそれぞれでアクセス可能になります。

データベースの統合

1つ以上の非CDBからマルチテナント・コンテナ・データベース(CDB)にデータを移動する一般プロセス。

データ・マイニング

大規模なデータ・ストアでパターンおよび傾向を検索する、単純な分析よりも優れた自動検索。

データ・リカバリ・アドバイザ

永続的なデータ障害を自動的に診断してユーザーに修復オプションを提示し、ユーザーのリクエストで修復を実行するOracle Databaseインフラストラクチャ。

データ・セグメント

クラスタ化されていない表、表パーティションまたは表クラスタのデータを含むセグメント。

「エクステント」も参照してください。

データ型

SQLにおいて、列の値または定数に関連付けられた、固定された一連のプロパティ。例として、VARCHAR2やNUMBERがあります。Oracle Databaseでは、様々なデータ型の値をそれぞれ異なる方法で処理します。

データ・ウェアハウス

OLTP用ではなく問合せおよび分析用に設計されたリレーショナル・データベース

データベース・アクセス制御

データ・アクセスおよびデータベース・アクティビティの制限。たとえば、指定された表への問合せや指定されたデータベース文の実行を制限します。

データベース・アプリケーション

データベースと対話してデータのアクセスと操作を行うソフトウェア・プログラム。

データベース認証

ユーザーがデータベースに資格証明を提示し、この資格証明をデータベースが検証してデータベースへのアクセスを許可するプロセス。

データベース・ブロック・サイズ

作成時に設定されたデータベースのデータ・ブロック・サイズ。このサイズは、SYSTEM表領域およびSYSAUX表領域に対して設定され、他のすべての表領域のデフォルトとなります。データベース・ブロック・サイズは、データベースを再作成する場合以外は変更できません。

データベース・バッファ・キャッシュ

システム・グローバル領域(SGA)でデータ・ブロックのコピーを保持している部分。データベース・インスタンスに同時接続されたクライアント・プロセスはすべて、バッファ・キャッシュへのアクセスを共有します。

データベース文字セット

データベースで表現される言語を決定するキャラクタ・エンコーディング・スキーム。

データベースの統合

1つ以上の非CDBからマルチテナント・コンテナ・データベース(CDB)にデータを移動する一般プロセス。

データベース・ドライバ

アプリケーションとOracleデータベースの間にあるソフトウェア。このドライバは、アプリケーションによるAPIコールをデータベースが処理可能なコマンドへと変換します。ODBCドライバを使用することで、スプレッドシートに格納されているデータなど、あらゆるデータ・ソースにアプリケーションからアクセスできます。ODBCドライバは、ODBC標準とデータベースの間のマッピングをすべて実行します。

データベース・インスタンス

システム・グローバル領域(SGA)とバックグラウンド・プロセスの組合せ。インスタンスは1つのデータベースのみに関連付けられます。データベース・インスタンスはすべて、読取り/書込みデータベース・インスタンスまたは読取り専用データベース・インスタンスのいずれかです。Oracle Real Application Clusters構成では、複数のインスタンスが1つのデータベースにアクセスします。

データベース・リンク

ユーザーが別のデータベースのオブジェクトにアクセスできるようにする、1つのデータベース内のスキーマ・オブジェクト。

データベース管理システム(DBMS)

データの格納、編成および取得を制御するソフトウェア。

データベース・オブジェクト

SQLで操作できるデータベース内のオブジェクト。表や索引などのスキーマ・オブジェクトは、スキーマに常駐します。ディレクトリやロールなどの非スキーマ・オブジェクトは、スキーマに常駐しません。

データベース操作

データベース監視のコンテキストで、SQL文、PL/SQLブロック、またはその両方を含む論理エンティティ。

データベースのPoint-in-Timeリカバリ

データベースの以前のバージョンを生成するメディア・リカバリのタイプ。この場合、リストアされたバックアップ以後に生成されたREDOすべてを適用するわけではありません。

データベース・セキュリティ

ユーザー認証、暗号化、アクセス制御および監視が含まれるデータベース管理機能。

データベース・サーバー

マルチユーザー環境において、複数のユーザーが同時に同じデータにアクセスできるように大量のデータを確実に管理するサーバー。権限のないアクセスに対して保護機能を備えながら、障害のリカバリについても効率のよい解決方法を提供します。

データベース・サーバー・グリッド

1つ以上のデータベースで実行するために接続された汎用サーバーの集合。

データベース・サービス

1つ以上のデータベース・インスタンスの名前付きの表現。Oracle Databaseのサービス名は、通常、そのグローバル・データベース名です。クライアントは、サービス名を使用して、1つ以上のデータベース・インスタンスに接続します。

データベース記憶域グリッド

相互に組み合され、データベース・サーバー・グリッド内のコンピュータからアクセスされる低コストのモジュラ・ストレージ・アレイの集合。

データベース・ユーザー

Oracleデータベースへのログインに使用できるアカウント。

データベース・ライター(DBW)

データベース・バッファ・キャッシュ内のバッファ・データをデータファイルに書き込むバックグラウンド・プロセス。

DDL

データ定義言語。データ構造を定義または変更するCREATE TABLEやALTER INDEXなどの文があります。

デッドロック

2人以上のユーザーが互いに他方のユーザーによってロックされたデータを待機している状況。このようなデッドロックは、Oracle Databaseではまれです。

宣言型言語

方法ではなく、何をするかについて記述する非手続き型の言語。宣言型言語の例として、SQLやPrologがあります。SQLは、ユーザーが結果の導出方法ではなく目的とする結果を指定するという意味において、宣言型です。

専用サーバー

サーバー・プロセスが単一のクライアント・プロセスのリクエストを処理するデータベース構成。

「共有サーバー」も参照してください。

遅延可能制約

SET CONSTRAINT文でCOMMIT文が発行されるまで制約チェックの遅延を許可する制約。遅延可能制約を使用すると、制約に違反する可能性がある変更を行う際に制約を一時的に無効にできます。

遅延挿入

MEMOPTIMIZE FOR WRITEとして指定された表へのMEMOPTIMIZE_WRITEヒントによる挿入。遅延挿入はロールバックできず、データベース・バッファ・キャッシュの一貫性メカニズムは使用されません。遅延挿入はバックグラウンド・プロセスによりデータ・ファイルに非同期に書き込まれます。

定義者権限PL/SQLプロシージャ

現行ユーザーではなく、所有者の権限で実行されるプロシージャ。

並列度

単一の処理に関連付けられているパラレル実行サーバーの数。パラレル実行は複数のCPUを効率よく使用するためのものです。Oracle Databaseのパラレル実行フレームワークでは、特定の並列度をユーザーが明示的に選択することも、Oracle Databaseによって自動的に並列度を制御することもできます。

依存オブジェクト

スキーマ・オブジェクト依存性で、定義が別のオブジェクトを参照するオブジェクト。たとえば、オブジェクトAの定義がオブジェクトBを参照している場合、AはBの依存オブジェクトになります。

降順索引

指定された列(1つまたは複数)でデータが降順に格納される索引。

ディメンション

ユーザーのビジネス上の疑問に答えられるようにデータを分類する構造。よく使用されるディメンションとしてcustomers、

products、timeがあります。

ディメンション表

スター・スキーマまたはスノーフレーク・スキーマにおいて、ディメンションの全部または一部の値が格納されるリレーショナル表。通常、ディメンション表にはディメンションのキー、レベルおよび属性の列が含まれています。

ダイレクト・パスのINSERT

データベースにより、データベース・バッファ・キャッシュを迂回して、データがデータファイルに直接書き込まれるINSERT。データベースは、挿入データを表内の既存データに追加します。

ダイレクト・パス読取り

PGAへのシングルまたはマルチブロックでの読込み(SGAを回避)。

ディレクトリ・オブジェクト

サーバーのファイルシステム上の、外部バイナリ・ファイルLOB(BFILE)および外部表データが保存されているディレクトリの別名を指定するデータベース・オブジェクト。ディレクトリ・オブジェクトはすべて1つの名前空間に作成され、個別のスキーマによって所有されません。

内容を保証しない読取り

トランザクションが別のトランザクションによって書き込まれた未コミット・データを読み取るときに発生する状況。Oracle Databaseでは、内容を保証しない読取りは許可されません。

ディスパッチャ

「ディスパッチャ・プロセス(Dnnn)」を参照してください。

ディスパッチャ・プロセス(D*nnn*)

オプションのバックグラウンド・プロセスで、共有サーバー構成を使用している場合のみ存在します。各ディスパッチャ・プロセスは、接続されたクライアント・プロセスからのリクエストを使用可能な共有サーバー・プロセスにルーティングし、応答を戻します。

分散データベース

分散システムにおいて、アプリケーションで1つのデータ・ソースとして処理できるデータベースの集合。

分散環境

相互にシームレスに通信する様々なシステムのネットワーク。

分散処理

アプリケーションが自身のタスクをネットワーク内の異なるコンピュータ間に分散するときに発生する操作。

分散トランザクション

分散データベースのノード上のデータを個別に、またはグループとして更新する文を含むトランザクション。Oracle Databaseでは、分散トランザクションのデータ整合性が2フェーズ・コミット・メカニズムを使用して保証されます。

DML

データ操作言語。SELECT、INSERT、UPDATE、DELETEなどの文があります。

DMLロック

同時に実行される矛盾するDML操作またはDDL操作による破壊的な干渉を防ぐロック。DML文では、行ロックと表ロックが自動的に取得されます。

動的パフォーマンス・ビュー

データベースがオープンで使用中の間は継続的に更新される特殊なビュー。動的パフォーマンス・ビューは*V$*ビューと呼ばれることもあります。

動的SQL

完全なテキストが実行時まで認識されないSQL。動的SQL文は、実行時に、プログラムに入力されたりまたはプログラムにより作成される文字列に格納されます。

エディション

データベース・オブジェクトの再定義が可能なプライベートな環境。エディションベースの再定義を使用すると、アプリケーションの使用中にアプリケーションのデータベース・オブジェクトをアップグレードできるため、停止時間を最小限に抑えることや解消することが可能です。

暗号化

秘密キーと暗号化アルゴリズムを使用して、データを読取り不可能な形式に変換するプロセス。

等価結合

等価演算子が含まれている結合条件を使用した結合。

ETL

抽出、変換、ロード(ETL)。ソース・システムからデータを抽出してデータ・ウェアハウスに取り込む処理。

排他ロック

関連リソースの共有を防止するロック。リソースの排他ロックを取得した最初のトランザクションは、そのロックが解除されるまで、そのリソースを変更できる唯一のトランザクションになります。

実行SQL文

DML文やDDL文、SET TRANSACTION文など、データベース・インスタンスへのコールを生成するSQL文。

実行計画

SQL文を実行するためにデータベースによって使用されるステップの組合せ。各処理では、データベースからデータ行を物理的に検索するか、文を発行したユーザーのためにデータ行を準備します。ヒントを使用すると実行計画を上書きできます。

式

1つ以上の値、演算子および値を評価するSQL機能の組合せ。たとえば、2*2という式の結果は4になります。一般に、式ではその構成要素のデータ型が想定されます。

エクステント

特定の型の情報を格納するために割り当てられた複数の連続データ・ブロック。セグメントは1つ以上のエクステントで構成されます。

関連項目:「データ・ブロック」

外部表

メタデータがデータベースに格納されている一方、データがデータベース外部のファイルに格納されている読取り専用の表。データベースは外部表を記述したメタデータを使用して、外部表をリレーショナル表とみなし、外部表のデータを公開します。

抽出、変換、ロード(ETL)

「ETL」を参照してください。

ファクト

売上やコスト・データなど、ビジネス・メジャーを表すデータ。

ファクト表

データ・ウェアハウスのスター・スキーマにおいて、ファクト・データを含む表。ファクト表には一般にファクトを含む列と、ディメンション表の外部キーである列の2種類の列があります。

高速全索引スキャン

データベースがマルチブロック読取りを使用して索引のすべてのブロックを読み取り、分岐ブロックを破棄し、特定の順序によらず索引ブロックを返す全索引スキャン。

高速リカバリ領域

制御ファイル、オンラインREDOログのコピー、アーカイブREDOログ・ファイル、フラッシュバック・ログ、RMANバックアップなどのリカバリ関連ファイルが格納されているオプションのディスク位置。

フォルト・トレランス

高可用性アーキテクチャによって提供される、アーキテクチャ内のコンポーネント障害に対する保護機能。

フィールド

表の行または列の交差部分。

ファイル・システム

連続するディスク・アドレス空間内に構築されるデータ構造。

ファイングレイン監査

特定の表の列を監査し、ポリシー作成時にイベント・ハンドラを関連付けることができるデータベース監査のタイプ。

固定SGA

データベースとインスタンスの状態に関する全般情報、プロセス間で通信される情報など、様々な情報を含む内部ハウスキーピング領域。

フラッシュバック・データ・アーカイブ・プロセス(FBDA)

追跡した表の履歴行をフラッシュバック・データ・アーカイブにアーカイブするバックグラウンド・プロセス。追跡した表に対するDMLを含むトランザクションがコミットされると、このプロセスでは、変更された行の事前イメージがフラッシュバック・データ・アーカイブに格納されます。また、現在の行にメタデータを保存します。

全データベース・キャッシュ強制モード

ALTER DATABASE ... FORCE FULL DATABASE CACHING文を実行して手動で有効化するキャッシュ・モード。デフォルトのキャッシュ・モードとは異なり、Oracle Databaseではデータベース全体、NOCACHE属性で指定されたLOBがキャッシュされます。

外部キー

1つの列または列の集合の値が、それぞれ関連する表の一意キーまたは主キーの値と一致する必要がある整合性制約。外部キーに対する整合性制約で定義されるアクションは、参照先のデータが変更された場合のデータベースの動作を決定します。

外部キー制約

Oracle Databaseで、1つ以上の共通の列が含まれる2つの表に関連を規定する制約。この制約では、制約が定義される列の各値に対して、他の表と列を指定するもう一方の列の値が一致する必要があります。たとえば、参照整合性規則により、従業員が既存の部門にのみ従事できることを規定できます。

書式モデル

日時の書式を文字列で記述する文字リテラル。

空きリスト

手動セグメント領域管理(MSSM)でセグメント内の空き領域を管理するための、空きリストと呼ばれるリンクされたリスト。空き領域が存在するデータベース・オブジェクトでは、空きリストによって、使用済のセグメント領域と未使用のセグメント領域を分割するラインである最高水位標より下のブロックが追跡されます。ブロックが使用済になると、データベースは必要に応じて、空きリストにブロックを登録したり、空きリストからブロックを削除したりします。

全索引スキャン

データベースがルート・ブロックと左側の分岐ブロックのみを読み取って最初のリーフ・ブロックを検索し、単一のブロックI/Oを使用して、索引ソート順序でリーフ・ブロックを読み取る索引スキャン。

完全外部結合

内部結合の結果と、左側外部結合および右側外部結合の結果を返す2つの表の結合。

全表スキャン

データベースによって表のすべての行が順に読み取られ、選択基準を満たさない行が除外される表データのスキャン。データベースでは、最高水位標(HWM)より下のすべてのフォーマット済データ・ブロックがスキャンされます。

関数

PL/SQLプロシージャに類似した、常に単一の値を戻すスキーマ・オブジェクト。

関数ベース索引

1つ以上の列を含む関数や式の値を計算し、索引に格納する索引。

GDS

「グローバル・データ・サービス(GDS)」を参照してください。

GDSカタログ

Oracleデータベース内に存在する、GDS構成に関連付けられたメタデータ・リポジトリ。各クラウドのカタログは、1つのみです。

GDS構成

高パフォーマンス、可用性、およびリソースの最適使用率を確保しながら、GDSフレームワークによって1つ以上のグローバル・サービスを提供する単一仮想サーバーに統合された一連のデータベース。

「グローバル・サービス」も参照してください。

GDSプール

一意のグローバル・サービス・セットを提供し同じ管理ドメインに属するGDS構成内の一連のデータベース。

GDSリージョン

地理的あるいは他の方法で互いに関連付けられたデータベース・クライアントおよびサーバーが含まれる、GDS構成内の論理境界。

グローバル・データ・サービス(GDS)

レプリケートされたデータベース用の自動化されたワークロード管理ソリューションです。データベース・サービスとは、1つ以上のデータベース・インスタンスが名前付きで表現されたものです。GDSでは、レプリケートされた一連のデータベース間でOracle Databaseサービス・モデルを実装します。

グローバル・データベース名

データベース名(DB_NAME)とネットワーク・ドメイン(DB_DOMAIN)の組合せで、たとえばorcl.example.comのようになります。グローバル・データベース・ドメインはネットワーク内で一意です。

グローバル・パーティション索引

索引付き表で使用されるパーティション化スキームとは関係なくパーティション化されるBツリー索引。単一の索引パーティションがいずれかまたはすべての表パーティションを指すことができます。

グローバル・サービス

データ・レプリケーションにより同期化された複数のデータベースにより提供されるデータベース・サービス。

グローバル・サービス・マネージャ

Global Data Servicesフレームワーク内の中央管理ツール。GDS構成の各GDSリージョンには、少なくとも1つのグローバル・サービス・マネージャが存在する必要があります。

グローバル一時表

特定の期間、セッション固有の中間データを格納する特殊な一時表。

グラニュル

パラレルな作業の基本単位。Oracle Databaseによって、パラレル化対象の操作(表のスキャン、表の更新、索引の作成など)がグラニュル単位に分割されます。パラレル実行プロセスは操作を1回に1グラニュルずつ実行します。

グリッド・コンピューティング

多くのサーバーと記憶域を調整して単一の大容量コンピュータとして動作させるコンピューティング・アーキテクチャ。

グリッド・インフラストラクチャ

エンタープライズ・グリッド・アーキテクチャ用のインフラストラクチャを提供するソフトウェア。クラスタの場合、このソフトウェアにはOracle ClusterwareとOracle ASMが含まれます。スタンドアロン・サーバーの場合、このソフトウェアにはOracle ASMが含まれます。Oracle Databaseでは、これらの製品が**グリッド・ホーム**と呼ばれる1つのソフトウェア・インストールにまとめられてい

ます。

ハード解析

新しい実行可能なバージョンのアプリケーション・コードを構築するためにデータベースによって実行されるステップ。実行を依頼された文の解析済の表現が共有プール内に存在しない場合、データベースはソフト解析ではなくハード解析を行います。

ハッシュ・クラスタ

索引キーがハッシュ関数に置き換わることを除いて、索引付きクラスタと似ている表クラスタの一種。別個のクラスタ索引は存在しません。ハッシュ・クラスタでは、データが索引になります。

ハッシュ衝突

複数の入力値を同じ出力値にハッシュすること。

ハッシュ関数

任意の長さの入力値に対して作用し、固定長のハッシュ値を戻す関数。

ハッシュ索引

主キーによってメモリー領域に索引を付ける、memoptimizeプールの内部構造。

ハッシュ結合

データベースで、2つの表またはデータ・ソースの小さい方を使用して、メモリー内にハッシュ表を作成する結合。大きい方の表をスキャンしてハッシュ表を調べ、小さい方の表の対応する行のアドレスを見つけます。

ハッシュ・キー値

ハッシュ・クラスタで、クラスタ・キー列に挿入される実際の値または推定値。たとえば、クラスタ・キーがdepartment_idの場合は、ハッシュ・キー値が10、20、30のような値になります。

ハッシュ・パーティション化

ユーザーが指定するパーティション化キーに適用されるハッシング・アルゴリズムに基づいて行をパーティションにマッピングするパーティション化計画。行のマップ先は、データベースが行に適用する内部ハッシュ関数によって決定されます。ハッシュ・アルゴリズムは、各パーティションにほぼ同じ行数が含まれるように、行を複数のデバイスに均等に分散させるように設計されています。

ハッシュ表

ハッシュ結合で結合キーを行と関連付けるインメモリー・データ構造。たとえば、employees表とdepartments表の結合では、部門IDが結合キーになる場合があります。ハッシュ関数は結合キーを使用してハッシュ値を生成します。このハッシュ値は、ハッシュ表である配列内の索引です。

ハッシュ値

ハッシュ・クラスタで、バケットを識別する一意の数値ID。Oracle Databaseでは、無数にあるハッシュ・キー値を入力として受

け付け、限定された数のバケットに選別するハッシュ関数が使用されます。各ハッシュ値は、ハッシュ・キー値(部門10、20、30など)に対応する行の格納ブロックのデータベース・ブロック・アドレスにマップされます。

ハッシング

入力値の無限集合をハッシュ値と呼ばれる出力値の有限集合に対応付ける数学的技法。ハッシングはハッシュ表内のデータの迅速な検索に役立ちます。

ヒープ構成表

データ行が特定の順序によらずディスク上に格納される表。デフォルトでは、CREATE TABLEでヒープ構成表が作成されます。

階層データベース

データがツリー構造で編成されたデータベース。ファイルシステムの構造と同様に、各親レコードが1つ以上の子レコードを持ちます。

最高水位標(HWM)

セグメント内の使用済領域と未使用領域の境界。

ヒント

SQL文の中のコメントを介してオプティマイザに渡される命令。オプティマイザはヒントを使用して文の実行計画を選択します。

ホット・バッファ

データベース・バッファ・キャッシュで、頻繁にアクセスされ、最近使用されたバッファ。

人為的なエラーによる停止

データベースのデータを論理的に破損させるか、使用不可能な状態にする意図的ではないアクションまたは悪意のあるアクションが実行されたときに発生する停止状態。

ハイブリッド列圧縮

行の手法と列の手法を組み合せてデータ・ブロック内のデータを圧縮する方法。ハイブリッド列圧縮の行の集合を格納するために、圧縮ユニットと呼ばれる論理的な構造が使用されます。

ハイブリッド・パーティション表

一部のパーティションがデータ・ファイル・セグメントに格納され、一部が外部データ・ソースに格納される表。

イメージ・コピー

データファイル、制御ファイルまたはアーカイブREDOログ・ファイルをビット単位でディスク上に複製したもの。オペレーティング・システムのユーティリティまたはRMANを使用して物理ファイルのイメージ・コピーを作成し、いずれかのツールを使用してファイルをリストアできます。

暗黙的問合せ

副問合せなしでデータを取得するDML文の構成要素。明示的にSELECT文が含まれていないUPDATE、DELETEまたはMERGE文では、変更対象の行が暗黙的問合せを使用して取得されます。

IM列ストア

迅速スキャン用に最適化された表とパーティションのコピーを、列形式で格納するオプションのSGA領域。

インメモリー列ストア

IM列ストアを参照してください。

アクティブでないオンラインREDOログ・ファイル

インスタンスのリカバリに必要ないオンラインREDOログ・ファイル。

非一貫性バックアップ

バックアップの一部のファイルに、チェックポイントより後に行われた変更が含まれているバックアップ。一貫性バックアップとは異なり、非一貫性バックアップではメディア・リカバリに一貫性を持たせる必要があります。

増分リフレッシュ

マテリアライズド・ビューで既存データの変更のみを処理するリフレッシュ。この方法により、完全リフレッシュは不要になります。

永久的増分バックアップ計画

初期レベル0のバックアップが、レベル1で発生する後続の増分バックアップとともにリカバリ・アプライアンスに作成される計画。リカバリ・アプライアンスでは、初期レベル0と後続レベル1のバックアップを組み合せ、仮想完全バックアップが作成されます。

索引

クラスタ化されていない表、表パーティションまたは表クラスタに関連付けられた、オプションのスキーマ・オブジェクト。索引によってデータ・アクセスが高速化される場合があります。

索引ブロック

領域を表ブロックとは異なる方法で管理する特殊なタイプのデータ・ブロック。

索引クラスタ

索引を使用してデータを検索する表クラスタ。クラスタ索引とは、クラスタ・キーのBツリー索引のことです。

索引クラスタ化係数

姓などの索引付きの値を対象に、行の順序性を測定した指標。行の記憶域の中でこの値に関する順序性がよいほど、クラスタ化係数は小さくなります。

索引レンジ・スキャン

次の特性を持つ索引の順序付きスキャン。

- 索引の1つ以上の先頭列を条件に指定します。条件は、1つ以上の式および論理(ブール)演算子の組合せを指定し、TRUE、FALSEまたはUNKNOWNの値を戻します。
- 0個、1個、または複数の値を索引キーにできます。

索引スキャン

文で指定された索引付きの列の値を使用して索引を検索することで、行を取得すること。

索引セグメント

パーティション化されていない索引または索引パーティションのデータを格納するセグメント。

索引スキップ・スキャン

コンポジット索引の論理副索引を使用する索引スキャン。個別の索引を検索する場合と同様に、データベースでは単一の索引をスキップします。

索引の一意スキャン

索引キーに関連付けられたROWIDが0個または1個である必要のある索引スキャン。データベースでは、述語が等価演算子を使用して一意索引のキーのすべての列を参照する場合に、一意スキャンが実行されます。

索引構成表

記憶域組織がプライマリBツリー索引の改良型となる表。ヒープ構成表とは異なり、データは主キーの順に格納されます。

インダウト分散トランザクション

システム障害やネットワーク障害によって2フェーズ・コミットが中断されたときに発生する分散トランザクション。

処理中のトランザクション

故障によりクライアント・アプリケーションとデータベース間の接続が切れたときに実行中のトランザクション。

情報システム

情報を格納および処理するための形式が定められたシステム。

初期化パラメータ

データベース・インスタンスの操作に影響するDB_NAMEやSGA_TARGETなどの構成パラメータ。初期化パラメータ設定はテキストベースの初期化パラメータ・ファイルまたはバイナリ・サーバー・パラメータ・ファイルに格納されます。

初期化パラメータ・ファイル

データベース・インスタンスの初期化パラメータ設定が含まれているテキスト・ファイル。

内部結合

結合条件を満たす行のみを戻す2つ以上の表の結合。

インスタンス障害

ハードウェア障害、Oracle内部エラーまたはSHUTDOWN ABORT文によるデータベース・インスタンスの終了。

インスタンスPGA

データベース・インスタンスの個別のPGAの集まり。

インスタンス・リカバリ

障害後のデータベース・インスタンス再起動時の、コミットされていないデータ・ブロックへのREDOログ・レコードの自動的な適用。

INSTEAD OFトリガー

トリガー文を実行するかわりにOracle Databaseによって起動されるトリガー。これらのトリガーは、DML文を介して直接変更できないビューを透過的に変更する場合に便利です。

整合性

「データ整合性」を参照してください。

整合性制約

列に関する規則を定義するための宣言型の方法。整合性制約はビジネス・ルールを適用し、無効な情報が表に入力されるのを防止します。

Interested Transaction List (ITL)

ブロックの変更が開始された時点でトランザクションがコミットされているかどうかを指定する、ブロック・ヘッダー内の情報。ITLのエントリは、ロックされた行のあるトランザクション、およびブロック内のコミット済の変更や未コミットの変更を含む行を表しています。

時間隔パーティション

指定した範囲または時間間隔のパーティションを作成するようにデータベースに指示する、レンジ・パーティション化の拡張。表に挿入されたデータが既存のすべてのレンジ・パーティションを超えると、データベースによってパーティションが自動的に作成されます。

非参照用索引

DML操作によって管理されるが、デフォルトではオプティマイザにより使用されない索引。索引を非表示にするのは、使用禁止または削除の代替手段です。

実行者権限PL/SQLプロシージャ

現行のユーザーの権限で現行のユーザーのスキーマ内で実行されるプロシージャ。

Javaプール

Java仮想マシン(JVM)内のセッション固有のすべてのJavaコードとデータが格納されるメモリー領域。

Java Publisher (JPublisher)

SQLオブジェクトやPL/SQLパッケージなどのデータベース・エンティティをJavaクライアント・プログラムで表すためにJavaクラスを生成するユーティリティ。

Javaストアド・プロシージャ

SQLにパブリッシュされデータベースに格納されるJavaメソッド。

JavaScriptオブジェクト

プロパティ名と関連付けられたJavaScript Object Notation (JSON)値の0個以上のペアの連想配列。

JavaScript Object Notation(JSON)

オブジェクト、配列およびスカラー・データを表すことができる、言語に依存しない、テキストベースのデータ形式。

ジョブ・キュー・プロセス

多くの場合、バッチ・モードでユーザー・ジョブを実行するオプションのバックグラウンド・プロセス。ジョブとは、1回以上実行するようにスケジューリングされたユーザー定義のタスクです。

結合

FROM句に指定された複数の表からデータを取得する文。結合のタイプには、内部結合、外部結合およびデカルト結合があります。

結合属性クラスタリング

属性クラスタ表における、結合された列に基づくクラスタリング。

結合条件

別々の表にある2つの列を結合時に比較する条件。データベースでは1対の行を組み合せ、それぞれの対には、結合条件がTRUEになる各表の行を1行ずつ含みます。

結合ビュー

定義でFROM句に複数の表またはビューが含まれるビュー。

JSONオブジェクト

中カッコで囲まれたプロパティ・リストとして記述されたJavaScriptオブジェクト・リテラル。

「JavaScript Object Notation (JSON)」も参照してください。

JVM

コンパイル済のJavaコードを実行する仮想プロセッサ。

キー

特定タイプの整合性制約の定義に含まれる列または列の集合。

キー圧縮

「接頭辞圧縮」を参照してください。

キー圧縮

接頭辞圧縮の代替名。

キー値

キー内の個別の値。

キー保存表

結合問合せで、問合せの出力にそれぞれの行が最大1回しか出現しない表。

ラージ・オブジェクト(LOB)

「LOB」を参照してください。

ラージ・プール

バックアップおよびリストア操作、I/Oサーバー・プロセスおよび共有サーバーとOracle XAのセッション・メモリー用に、大量のメモリー割当てを提供するSGA内のオプション領域。

ラッチ

SGA内の共有データ構造を同時アクセスから保護するために使用される、低レベルのシリアライズ制御メカニズム。

ラッチ・スリープ

プロセスがラッチ要求を更新する前にCPUを解放すると発生する現象。

ラッチ・スピン

プロセスがループ内のラッチを繰り返し要求すると発生する現象。

リーフ・ブロック

Bツリー索引で、索引エントリを格納する低レベルのブロック。Bツリー索引の上位ブランチ・ブロックには、下位レベルの索引ブロックを指す索引データが含まれます。

左側外部結合

結合条件が右側の表*B*のレコードに一致していない場合でも、表*A*と*B*の左側外部結合の結果には左側の表*A*のすべてのレコードが含まれます。たとえば、employees (左)からdepartments (右)への左側外部結合を実行した場合、一部の従業員が部署にいなくても、問合せではdepartmentsの一致なしでemployeesの行が返されます。

ライブラリ・キャッシュ

共有プール内のメモリー領域。このキャッシュには、共有SQL領域、プライベートSQL領域(共有サーバー構成の場合)、PL/SQLプロシージャおよびパッケージ、ロックやライブラリ・キャッシュ・ハンドルなどの制御構造が含まれています。

リスト・パーティション化

各パーティションのパーティション・キーとして離散値のリストを使用するパーティション化計画。リスト・パーティション化を使用して、特定のパーティションに対する個々の行のマップ方法を制御できます。リストを使用すると、データの識別に使用されるキーの並び順が目的にそぐわない場合に、関連するデータのセットをグループ化して整理できます。

リスナー

受信クライアント接続リクエストをリスニングし、データベースへのネットワーク・トラフィックを管理するプロセス。

リスナー登録プロセス(LREG)

データベース・インスタンスおよびディスパッチャ・プロセスに関する情報をOracle Netリスナーに登録するプロセス。

リテラル

固定データ値。

LOB

ラージ・オブジェクト。SQLのBLOB、CLOB、NCLOBおよびBFILEデータ型を含むオブジェクト。これらのデータ型は、サイズの大きいデータを格納するように設計されています。

ローカル・パーティション索引

表と同じパーティション数および同じパーティション・バウンドを使用し、同じ列でパーティション化された索引。索引パーティションと表パーティションの間に1対1のパリティが存在します。

ローカル・ロール

非CDBのロールが非CDBにのみ存在するのと同様に、CDBにおいて単一のPDBにのみ存在するロール。共通ロールとは異なり、ローカル・ロールには、そのロールが存在するコンテナ内で適用されるロールおよび権限のみを含めることができます。

ローカル一時表領域

ローカル記憶域に存在し、特定のデータベース・インスタンスによるアクセスが可能な一時表領域。対照的に、共有共有一時表領域は共有記憶域に存在し、すべてのデータベース・インスタンスによるアクセスが可能です。

ロケール

グローバリゼーション・サポートのコンテキスト内で、システムまたはプログラムが実行されている言語的および文化的な環境。

ローカル管理表領域

各データファイルに格納されたビットマップを使用してエクステントを管理する表領域。一方、ディクショナリ管理表領域は、データ・ディクショナリを使用して領域を管理します。

ロック

表、行、ユーザーには見えないシステム・オブジェクトなどの共有リソースにアクセスする複数のトランザクション間で、破損を招く相互作用を回避するためのデータベース・メカニズム。ロックの主なカテゴリとして、DMLロック、DDLロックおよびラッチと内部ロックがあります。

ロック変換

低い制限から、より高い制限への表ロックの自動的な変換。たとえば、あるトランザクションで特定の従業員に対する`SELECT ... FOR UPDATE`を発行し、その後ロックされた行を更新するとします。この場合、データベースでは、行共有表ロックが行排他表ロックに自動的に変換されます。

ロックの段階的拡大

あるレベル(行レベルなど)で多数のロックが保持されている場合に、ロックをデータベースが上位レベルの粒度(表レベルなど)の別のロックに拡大する場合に発生する状況。*Oracle Database*では、ロックの段階的拡大が発生することはありません。

ログ順序番号

REDOログ・ファイル内のREDOレコードの集合を一意に識別する番号。データベースで1つのオンラインREDOログ・ファイルがいっぱいになり、別のファイルに切り替えるとき、データベースは新しいファイルにログ順序番号を自動的に割り当てます。

ログ・スイッチ

ログ・ライター・プロセス(LGWR)がアクティブなREDOログ・ファイルへの書込みを停止し、使用可能な次のREDOログ・ファイルに切り替えるポイント。LGWRは、アクティブなREDOログ・ファイルがREDOレコードでいっぱいになったとき、または手動で切替えが開始されたときに、切替えを行います。

ログ・ライター・プロセス(LGWR)

REDOログ・バッファ管理、つまりオンラインREDOログへのREDOログ・バッファの書込みを行うバックグラウンド・プロセス。LGWRは、最後の書込み以後にバッファにコピーされたREDOエントリすべてを、REDOログ・ファイルに書き込みます。

論理I/O

データベース・バッファ・キャッシュでのバッファの読取りと書込み。

論理読取り

データベース・バッファ・キャッシュでバッファを読み取ること。

論理ROWID

索引構成表のROWID。論理ROWIDは、表の主キーをbase64でエンコードした表現です。

論理トランザクションID

アプリケーションの観点から、トランザクションを定義するグローバルに一意の識別子。論理トランザクションIDは、データベース・トランザクションIDにバインドされます。

論理ボリューム

仮想ディスク・パーティション。

論理ボリューム・マネージャ(LVM)

複数の物理ディスクの断片を、ソフトウェアの上位レイヤーに対して1つのディスクとして表示される単一の連続するアドレス空間に結合する、ほとんどのオペレーティング・システムで使用可能なソフトウェア・パッケージ。

参照表

コード列および関連付けられた値列が含まれる表。たとえば、ジョブ・コードはジョブ名に対応します。マスター/ディテール表ペアのマスター表とは対照的に、参照表は、従業員のリストなどの詳細な結果セットを取得する手段ではありません。逆に、ユーザーは従業員リストを求めてemployeesなどの表に問い合せ、結果セットを参照表に結合します。

更新内容の消失

あるデータ書込みによって、同じデータを変更する別の書込みの変更内容が上書きされるというデータ整合性の問題。

書込み欠落

データベースではブロックを永続記憶域に書き込んだと想定されていますが、実際にはブロックが書き込まれていなかったか、前のバージョンのブロックが書き込まれた場合に発生するデータ破損。

管理性モニター・プロセス(MMON)

自動ワークロード・レポジトリ(AWR)に関連する多くのタスクを実行するバックグラウンド・プロセス。たとえば、MMONはスナップショットを取得し、前回変更されたSQLオブジェクトの統計値を取得して、メトリックがそのしきい値を超えたときに書込みを行います。

仮数

有効桁数を含む浮動小数点数の一部。

手動セグメント領域管理(MSSM)

空きリストと呼ばれるリンク・リストを使用してセグメントの空き領域が管理される従来の領域管理方式。

手動UNDO管理モード

UNDOブロックがユーザー管理UNDOセグメントに格納されるデータベース・モード。自動UNDO管理モードでは、UNDOブロックはシステム管理の専用UNDO表領域に格納されます。

マスター・データベース

レプリケーションで、サブスクライバ・データベースにコピーされるデータのソース。マスター・データベースのレプリケーション・エージェントは、マスター・データベースのトランザクション・ログからレコードを読み取ります。レプリケートされた要素への変更をサブスクライバ・データベースのレプリケーション・エージェントに転送します。その後、サブスクライバ・データベースのレプリケーション・エージェントが、それらの更新を適用します。

マスター・サイト

レプリケーション環境において、マテリアライズド・ビューがデータを共有する別のデータベース。

マスター表

レプリケーション環境において、マスター・サイトのマテリアライズド・ビューに関連付けられた表。

マスター/ディテール表

ディテール表には、マスター表との外部キー関係があります。たとえば、employeesディテール表には、departmentsマスター表に対する外部キーがあります。参照表とは異なり、通常、マスター表は問い合せられてからディテール表に結合されます。たとえば、ユーザーはdepartments表で部門について問い合せてから、その結果を使用して部門内の従業員を検索します。

マテリアライズド・ビュー

問合せの結果を格納するスキーマ・オブジェクト。問合せのFROM句には、表、ビューおよびその他のマテリアライズド・ビューを指定できます。

「ビュー」も参照してください。

メディア・リカバリ

REDOまたは増分バックアップをデータ・ブロックまたはバックアップ・データファイルに適用すること。

memoptimizeプール

バッファおよびMEMOPTIMIZE FOR READと指定されたヒープ構成表の関連構造を格納するSGAプール。

メトリック

累積統計の変更率。

マウントされたデータベース

起動されており、データベース制御ファイルがオープンしているデータベース・インスタンス。

マルチテナント・コンテナ・データベース(CDB)

「CDB」を参照してください。

マルチスレッドのOracle Databaseのモデル

別のアドレス領域でオペレーティング・システム・システム・スレッドとしてOracleプロセスを実行可能にするモデル。スレッド・モードでは、UNIXおよびLinux上の一部のバックグラウンド・プロセスは、1つのスレッドを含むプロセスとして実行されることに対し、残りのOracleプロセスは、プロセス内のスレッドとして実行されます。

複数層アーキテクチャ

1つ以上のアプリケーション・サーバーがクライアントにデータを提供し、クライアントとデータベース・サーバーの間のインタフェースとして機能するようなアーキテクチャ。

マルチバージョン一貫性モデル

データベースが複数の同時ユーザーに対してデータのビューを提示する場合、それぞれ特定の時点において一貫性があるビューにすることができるモデル。

マルチバージョン

複数のバージョンのデータを同時にマテリアライズできるデータベースの機能。

相互排他オブジェクト(mutex)

メモリー内のオブジェクトが同時プロセスでアクセスされた場合にエージ・アウトまたは破損しないようにするための低レベルのメカニズム。

自然キー

表内の既存の属性から作成される、意味のある識別子。たとえば、参照表の郵便番号を自然キーとして使用できます。

ネットワーク・データベース

レコードが1対多の関係ではなく多対多の関係を持つ、階層データベースと似たデータベースのタイプ。

ネットワーク暗号化

クライアントとサーバー間のネットワークを移動するデータを暗号化すること。

非CDB

マルチテナント・コンテナ・データベース(CDB)ではないOracle Database。Oracle Database 12cまでのすべてのデータベースは非CDBでした。Oracle Database 12cから、すべてのデータベースはCDBまたは非CDBである必要があります。

非循環再利用レコード

頻繁には変更されず、上書きできない重要な情報が含まれる制御ファイル・レコード。たとえば、表領域、データファイル、オンラインREDOログ・ファイル、REDOスレッドなどの情報です。Oracle Databaseでは、対応するオブジェクトが表領域から削除されないかぎり、これらのレコードを再使用しません。

遅延不可制約

妥当性チェックをトランザクションの最後まで遅延することがない制約。かわりに、データベースでは各文の最後で制約をチェックします。制約に違反している場合、その文はロールバックします。

NULL

1つの行の1つの列に値がないこと。NULLは、データがない、不明である、または適切でないことを示します。

オブジェクト表

各行がオブジェクトを表す特殊な表。

オブジェクト型

発注書のような実社会のエンティティを抽象化したスキーマ・オブジェクト。属性はエンティティの構造をモデル化し、メソッドはアプリケーションがエンティティに対して実行できる操作を実装します。

オブジェクト・ビュー

仮想オブジェクト表。このビューの各行はオブジェクトで、これはユーザー定義データ型のインスタンスです。

オブジェクト・リレーショナル・データベース管理システム(ORDBMS)

ユーザー定義型、継承、ポリモフィズムなどのオブジェクト指向機能を実装したRDBMS。

OLAP

オンライン分析処理。OLAPは履歴データの動的なディメンション分析を特徴とします。

OLAPページ・プール

データ・ブロックと同じであるOLAPデータ・ページを管理するUGAのプール。ページ・プールは、OLAPセッションの開始時に割り当てられて、セッションの終了時に解放されます。

OLTP

オンライン・トランザクション処理。OLTPシステムは、高速で信頼性の高いトランザクション処理用に最適化されています。データ・ウェアハウス・システムに比べ、OLTPの相互作用では、扱われる行が少なく、表のグループが大きいのが通例です。

オンラインREDOログ

Oracle Databaseのデータファイルおよび制御ファイルに対して行われたすべての変更を記録する、複数のオンラインREDOログ・ファイルの集合。データベースに変更が加えられると、Oracle DatabaseではREDOバッファでREDOレコードが生成されます。ログ・ライター・プロセス(LGWR)では、REDOログ・バッファの内容をオンラインREDOログに書き込みます。

オンラインREDOログ・グループ

オンラインREDOログ・ファイルとその多重コピー。

オペレーティング・システム・ブロック

オペレーティング・システムが読取りまたは書込みできるデータの最小単位。

演算子

1. メモリー管理では、演算子はデータのフローを制御します。たとえば、ソート、ハッシュ結合、ビットマップ・マージの演算子があります。

2. SQLでは、演算子はオペランドまたは引数と呼ばれるデータ項目を操作し、結果を戻します。キーワードまたは特殊文字は、演算子を表します。たとえば、アスタリスク(*)は、乗算演算子を表します。

オプティマイザ

SQL文で参照されているオブジェクトと指定された条件に関連する要因を考慮して、SQL文を実行する最も効率的な方法を決定する組込みのデータベース・ソフトウェア。

オプティマイザ統計

各SQL文に対して最適な実行計画を選択するためにオプティマイザによって使用されるデータベースおよびデータベース・オブジェクトに関する詳細。カテゴリには、行数などの表統計、Bツリー・レベルなどの索引統計、CPUとI/Oのパフォーマンスなどのシステム統計およびNULL数などの列統計が含まれます。

Oracle Application Express

Oracle Database用のWebアプリケーション開発ツール。Oracle Application Expressのユーザー・インタフェースのテーマ、ナビゲーション・コントロール、フォーム・ハンドラ、柔軟なレポートなどの組込み機能を使用することで、効率よくアプリケーションを開発できます。

Oracleアーキテクチャ

データベースの管理でOracle Databaseによって使用されるメモリーとプロセスの構造。

Oracle ASM

Oracle Automatic Storage Management (Oracle ASM)。データベース・ファイル用のボリューム・マネージャとファイル・システム。Oracle ASMはオラクル社が推奨するストレージ管理ソリューションであり、従来のボリューム・マネージャおよびファイル・システムの代替手段となります。

Oracle ASM割当て単位

ASMディスク・グループ内の割当ての基本単位。割当て単位は、Oracle ASMによって割り当てられる最小の連続するディスク領域になります。1つ以上の割当て単位によって、1つのOracle ASMエクステントが構成されます。

Oracle ASMディスク

Oracle ASMディスク・グループにプロビジョニングされるストレージ・デバイス。Oracle ASMディスクは、物理ディスクまたはパーティション、ストレージ・アレイからの論理ユニット番号(LUN)、論理ボリュームまたはネットワーク接続ファイルのいずれでもかまいません。

Oracle ASMディスク・グループ

1つの論理単位として管理される1つ以上のOracle ASMディスク。ディスク・グループに対するI/Oは、そのグループの全ディスクに自動的に分散されます。

Oracle ASMエクステント

Oracle ASMファイル内のセクション。1つのOracle ASMファイルは1つ以上のファイル・エクステントで構成されます。それぞれのOracle ASMエクステントは、特定のディスク上にある1つ以上の割当て単位で構成されます。

Oracle ASMファイル

Oracle ASMディスク・グループに格納されているファイル。データベースには、データファイル、制御ファイル、オンラインREDOログ・ファイルなどの様々なファイルをOracle ASMファイルとして格納できます。

Oracle ASMインスタンス

Oracle ASMディスクを管理する特殊なOracleインスタンス。Oracle ASMインスタンスとデータベース・インスタンスのどちらにも、Oracle ASMディスク・グループのディスクへの共有アクセスが必要です。Oracle ASMインスタンスはディスク・グループのメタデータを管理して、ファイルのレイアウト情報をデータベース・インスタンスに提供します。

Oracle Automatic Storage Management(Oracle ASM)

「Oracle ASM」を参照してください。

Oracleベース

Oracleベース・ディレクトリは、Oracle Databaseインストール所有者のデータベース・ホーム・ディレクトリです。ORACLE_BASE環境変数で指定され、Oracleディレクトリ・ツリーのルートとなるディレクトリです。

Oracleベース構成ディレクトリ

インスタンス固有の構成ファイルを含むディレクトリ。

読取り/書込みOracleホームの場合、Oracleベース・ホーム(ORACLE_BASE_HOME)、Oracleベース構成ディレクトリ(ORACLE_BASE_CONFIG)、およびOracleホームは同じです。読取り専用Oracleホームの場合、Oracleベース・ディレクトリ構成ディレクトリにはOracleベース・ホームのすべてのOracleホームによって共有される構成ファイルが含まれます。ネーミングの競合を避けるために、各ファイル名にはSID(システム識別子)が含まれています。

Oracleベース・ホーム

Oracleホームに関連付けられたデータベース・インスタンスの構成ファイルを格納するディレクトリ。

読取り/書込みOracleホームの場合、Oracleベース・ホーム(ORACLE_BASE_HOME)、Oracleベース構成ディレクトリ(ORACLE_BASE_CONFIG)、およびOracleホームは同じです。読取り専用Oracleホームの場合、Oracleベース・ホームはホーム固有のディレクトであり、Oracleベースのhomes/home_nameサブディレクトリにあります。

Oracle Clusterware

複数のサーバーを1つのサーバーとみなし、まとめて実行できるようにするコンポーネントの集合。Oracle Clusterwareは、Oracle RACの使用に不可欠であり、Oracle RACを実行するプラットフォーム用に必要な唯一のクラスタウェアです。

Oracle Connection Manager

クライアント接続要求をその次のホップに送信するか、または直接データベース・サーバーに送信する際の経路となるルーター。

Oracle Data Redaction

権限の少ないユーザーまたはアプリケーションによって問い合されるデータを隠す(リダクションする)ことができる、Oracle Advanced Securityの機能。

Oracleデータベース

ディスク上に配置された、データを格納する一連のファイル。データベース・インスタンスとデータベースは非常に密接に関連しているため、*Oracle*データベースという用語が、インスタンスとデータベースの両方を示す場合があります。

Oracle Database Vault

データベース、データおよびアプリケーションに対するアクセスのタイミング、場所および方法を制御するデータベース・セキュリティ機能。

Oracle Developer Tools for Visual Studio

Visual Studio .NET環境に統合されたアプリケーション・ツールのセット。これらのツールを使用すると、ユーザーはGUIを介し

てOracle機能にアクセスして、様々なアプリケーション開発タスクを実行でき、生産性および操作性が向上します。

Oracle Enterprise Manager

Oracleデータベース環境を集中管理するためのシステム管理ツール。

Oracle Flashbackテクノロジ

バックアップをリストアせずに、過去のデータの状態を表示したり、特定の時間に応じてデータを前後に巻き戻すことができる機能のグループ。

Oracle Flex Clusters

Oracle ClusterwareおよびOracle Real Application Clustersを使用して構成された大規模なクラスタ。これらのクラスタには、ハブアンドスポーク・アーキテクチャに配置された2種類のノード、ハブ・ノードとリーフ・ノードが含まれます。

Oracle Globalization Development Kit (GDK)

Java用とPL/SQL用の包括的なプログラミングAPIとコード・サンプルが用意され、グローバル・アプリケーションの構築時に直面する設計、開発および配置に関する数多くの問題の対処方法を解説したドキュメントも用意されている開発ツールキット。

Oracleホーム

オペレーティング・システムのOracle Databaseインストールの位置。Oracleホームは、読取り/書込み可能な場合と読取り専用の場合があります。

Oracle JDeveloper

最新の業界標準Java、XML、Webサービス、およびSQLを使用するサービス指向アプリケーションを構築するための統合開発環境(IDE)。

Oracle JVM

Pure Javaアプリケーションを実行する標準的なJava互換環境。

Oracle Managed Files

ファイル名ではなく、データベース管理者がデータベース・オブジェクトの観点で操作を指定できるようにするためのデータベース・ファイル命名方法。Oracle Managed Filesでは、データベース内のオペレーティング・システム・ファイルを管理者が直接管理する必要がなくなります。

Oracle Multimedia

Oracle Databaseで画像、音声、動画、DICOMフォーマットの医療用画像などのオブジェクトや、異機種間のメディアを、その他企業情報とともに統合された方法で格納、管理および取得できるようにするテクノロジ。

Oracle Net

クライアント・アプリケーションとOracleデータベース間のネットワーク・セッションを可能にする通信ソフトウェア。ネットワーク・セッ

ションの確立後、Oracle Netはクライアント・アプリケーションとデータベースのためのデータ伝達手段として機能します。

Oracle Net Listener

サーバー上のプロセスであり、着信クライアント接続要求をリスニングし、サーバーへのトラフィックを管理する責任を持ちます。クライアントがデータベースとのネットワーク・セッションをリクエストするときに、Oracle Net Listener (通常はリスナーと呼ばれる)はリクエストを受け取ります。クライアント情報がリスナー情報と一致した場合、リスナーはデータベース・サーバーへの接続を許可します。

Oracle Net Services

分散された異機種間コンピューティング環境における、企業全体の接続性ソリューションを提供するネットワーキング・コンポーネントのスイート。Oracle Net Servicesは、Oracle Net、リスナー、Oracle Connection Manager、Oracle Net Configuration AssistantおよびOracle Net Managerを含みます。

Oracle Optimal Flexible Architecture (OFA)

Oracleインストールを適切に体系化するために作成された構成ガイドライン一式。

Oracleプロセス

Oracleデータベース・コードを実行する実行単位。プロセス実行のアーキテクチャは、オペレーティング・システムによって決まります。Oracleプロセスには、サーバー・プロセスとバックグラウンド・プロセスがあります。

Oracle RAC

Oracle Real Application Clusters。複数の同時データベース・インスタンスが1つの物理データベースを共有できるオプション。

Real Application Clusters

「Oracle RAC」を参照してください。

Oracle Sharding

シェアードナッシング・アーキテクチャ内のOracleデータベースのプール間でデータの分散とレプリケーションを可能にするOLTPアプリケーションの機能。アプリケーションは、シャード・データベース(SDB)と呼ばれる単一の論理的なデータベースとしてプールにアクセスします。

Oracle Spatial and Graph

空間データおよび空間分析、物理グラフ、論理グラフ、ネットワーク・グラフ、ソーシャル・グラフおよびセマンティック・グラフ・アプリケーション用の高度な機能のセット。空間フィーチャは、Oracle Databaseへの空間フィーチャの集合の格納、検索、更新および問合せを容易にするスキーマおよび関数を提供します。

Oracle SQL

SQLのANSI標準の実装の1つ。Oracle SQLでは、標準SQLを拡張した数多くの機能がサポートされています。

Oracle Text (Text)

Oracle Databaseに統合された全文検索取得テクノロジ。

Oracle Virtual Private Database(VPD)

セキュリティ・ポリシーを作成し、行および列レベルでデータベース・アクセスを制御できるセキュリティ機能。基本的には、VPDのセキュリティ・ポリシーが適用されている表、ビューまたはシノニムに対して発行されたSQL文に対して動的なWHERE句がVPDによって追加されます。

Oracle XA

Oracle Database以外のトランザクション・マネージャでグローバル・トランザクションを調整できるようにする外部インタフェース。

Oracle XML DB

高パフォーマンスのXML操作の格納と取得に関連する一連のOracle Databaseテクノロジ。Oracle XML DBは、SQLデータ・モデルとXMLデータ・モデルの相互運用が可能な方法でサポートするシステム固有のXMLを提供します。

Oracle XML Developer's Kit(XDK)

XML文書(ファイルシステム上にあるかデータベース内にあるかは不問)の読取り、操作、変換および表示のための基本ビルディング・ブロックが用意されている開発者ツールキット。APIおよびツールは、Java、CおよびC++で使用できます。本番のOracle XDKには、商用再配布ライセンスが付属します。

外部結合

結合条件を満たすすべての行を返し、一方の表の行が結合条件を満たさなくても、他方の表の一部または全部の行を返す結合。

パラレル実行

複数のCPUおよびI/Oリソースを1つのデータベース操作の実行に適用すること。

解析ロック

SQL文またはPL/SQLプログラム・ユニットは、参照する各スキーマ・オブジェクトについてロックを保持します。解析ロックを取得する目的は、参照しているオブジェクトが変更または削除された場合に、対応する共有SQL領域を無効にできるようにするためです。

部分索引

関連付けられたパーティション表の索引付けプロパティと相互に関連付けられている索引。

パーティション

同じ論理属性を共有する表または索引のピース。たとえば、1つの表内のすべてのパーティションは同じ列定義と制約定義を共有します。各パーティションは独自の名前を持つ独立したオブジェクトであり、独自の記憶特性を持つ場合があります。

パーティション絞込み

問合せ計画からのパーティションの絞込み。オプティマイザが検討対象からパーティションを排除できるかどうかは、問合せの述語に依存します。ローカル同一キー索引を使用する問合せでは、常に索引のパーティション絞込みが可能ですが、ローカル非同一キー索引を使用している問合せでは、絞込みはできない場合があります。

パーティション・キー

パーティション表の各行が属するパーティションを決定する1つ以上の列のセット。各行は、単一パーティションに明示的に割り当てられます。

パーティション索引

小さくて管理しやすい単位に分割された索引。パーティション表と同様、パーティション索引も管理性、可用性、パフォーマンスおよびスケーラビリティを改善します。

パーティション表

1つ以上のパーティションで構成されており、パーティションは個別に管理され、他のパーティションから独立して機能できる表。

パーティション化

パーティションと呼ばれる、より小さく管理しやすい部分に大規模な表や索引を分割する機能。

PDB

マルチテナント・コンテナ・データベース(CDB)内で、Oracle Netクライアントに従来のOracleデータベース(非CDB)として表示されるスキーマ、スキーマ・オブジェクトおよび非スキーマ・オブジェクトのポータブル・コレクション。

PDB管理者

1つ以上のPDBを管理するデータベース管理者。CDB管理者は、CDB全体を管理します。

PDBリストア・ポイント

CDB内で、特定のPDBについてのみ使用可能なリストア・ポイント。対照的に、CDBリストア・ポイントはすべてのPDBによって使用可能です。

PDB/非CDB互換性保障

マルチテナント・アーキテクチャにおいて、Oracle Netで接続しているクライアントから、PDBと非CDBの動作が同じように見えるようにすること。

パフォーマンス・プロファイル

PDBまたはPDBのセットのために指定された共有のシステム・リソース、CPU、パラレル実行サーバーおよびメモリー。

永続表領域

永続スキーマ・オブジェクトが格納される表領域。一時表領域以外のすべての表領域は永続表領域です。

PGA

プログラム・グローバル領域。サーバー・プロセス用のデータや制御情報を含むメモリー・バッファです。

「SGA」も参照してください。

物理推測

索引エントリの最初の作成時の物理ROWID。Oracle Databaseは、物理推測を使用して、主キー検索をバイパスし、索引構成表のリーフ・ブロックを直接検証できます。

PL/SQL

手続き型言語/SQL。SQLに対するOracle Databaseの手続き型言語拡張機能です。PL/SQLを使用すると、SQL文をプロシージャ、関数、パッケージなどのプログラム構造と混合して使用できます。

無名PL/SQLブロック

アプリケーション内に表示されるが名前が付いていない、またはデータベースに格納されていないPL/SQLブロック。多くのアプリケーションでは、SQL文を記述できるところであればどこにでもPL/SQLブロックを記述できます。

PL/SQLコレクション

すべて同じ型の要素が順序付けされたグループ。各要素には、コレクション内での位置を決定する一意の添字が付いています。

PL/SQLエンジン

PL/SQLプログラム・ユニットの定義、コンパイルおよび実行に使用するツール。このエンジンは、Oracle Databaseをはじめとする多数のOracle製品に組み込まれている特別なコンポーネントです。

PL/SQL関数

SQL文やその他のPL/SQL構成メンバーのセットで構成され、グループとしてまとめてデータベースに格納されるスキーマ・オブジェクトのことで、特定の問題を解決したり、関連する一連のタスクを実行するために、1単位として実行され、コール元に対して必ず1つの値を返します。

PL/SQL関数の結果のキャッシュ

関数の結果セットを格納するサーバー結果キャッシュのサブセット。

PL/SQLパッケージ

関連するPL/SQLタイプ、変数およびサブプログラムを論理的にグループ化すること。

PL/SQLプロシージャ

SQL文やその他のPL/SQL構成メンバーのセットで構成され、グループとしてまとめてデータベースに格納されるスキーマ・オブジェクト。特定の問題を解決したり、関連する一連のタスクを実行するために、1単位として実行されます。

PL/SQLレコード

C、C++またはJavaの構造型に類似した、異なる型のデータ値を格納できる複合変数。レコードは、表内の行データまたは表内の行の特定列を保持する場合に役立ちます。

PL/SQLサブプログラム

一連のパラメータを使用して起動できる、名前付きPL/SQLブロック。

プラン・ジェネレータ

特定の問合せブロックに対して、最小コストの計画を検出するために、様々なアクセス・パス、結合方法および結合順序を試行するオプティマイザの要素。

プラガブル・データベース(PDB)

「PDB」を参照してください。

移入

IM列ストア内へのデータの転送。移入ではデータベース内に新規のデータは挿入されません。かわりに既存のデータがメモリー内に転送され、列形式で格納されます。

プラグマ

コンパイル・オプションを実行するようにコンパイラに指示するディレクティブ。たとえば、プラグマAUTONOMOUS_TRANSACTIONによって、このプロシージャは実行時に親トランザクションから独立した新しい自律型トランザクションとして実行する必要があることが、データベースに指示されます。

精度

浮動小数点数の桁数の合計。固定小数点数は、NUMBER(*p*, *s*)の形式で指定します(*p*は精度を表します)。

プリコンパイラ

C、C++、COBOLなどの言語で記述された高水準のソース・プログラムにSQL文を埋め込むことができるプログラミング・ツール。

述語

SQL文のWHERE条件。

接頭辞圧縮

索引構成表内の主キー列値で繰り返されている値の除去。接頭辞圧縮は、以前はキー圧縮と呼ばれていました。

主キー

表の行を一意に識別する列または列の集合。各表には1つの主キーのみを定義できます。

主キー制約

列または列の集合に重複値とNULLを許可しない整合性制約。

プライベートSQL領域

解析済の文と処理に使用するその他の情報が保持されるメモリー内の領域。プライベートSQL領域には、バインド変数値、問合せ実行状況の情報、問合せ実行作業領域などのデータが格納されます。

権限

特定のタイプのSQL文を実行する権利や、別のユーザーに属するオブジェクトにアクセスする権利、PL/SQLパッケージを実行するなどの権利。権限のタイプはOracle Databaseによって定義されます。

権限分析

指定した条件に従ってデータベースの権限使用状況を取得するセキュリティ・メカニズム。たとえば、ユーザーが特定のデータベース・セッション中に行使した権限を見つけることができます。

手続き型言語

(宣言型言語のように)何を実行するかではなく、処理の実行方法を記述する言語。手続き型言語の例として、C++やJavaがあります。

プロセス

一連の処理を実行できるオペレーティング・システムのメカニズム。Oracle Databaseとデータベース・アプリケーションの作業を複数のプロセスに分割することにより、複数のユーザーやアプリケーションが同時に1つのデータベース・インスタンスに接続できます。

関連項目:「バックグラウンド・プロセス」、「Oracleプロセス」、「クライアント・プロセス」

プロセス・モニター(PMON)

他のバックグラウンド・プロセスの終了を検出するバックグラウンド・プロセス。サーバー・プロセスまたはディスパッチャ・プロセスが異常終了すると、プロセス・モニター(PMON)グループがプロセスのリカバリを実行します。

プロセス・モニター(PMON)グループ

他のプロセスの監視とクリーンアップを担当するバックグラウンド・プロセスのグループ。PMONグループには、プロセス・モニター(PMON)、クリーンアップ・メイン・プロセス(CLMN)およびクリーンアップ・ヘルパー・プロセス(CL*nn*)が含まれています。

プログラム・グローバル領域(PGA)

「PGA」を参照してください。

プログラム・インタフェース

データベース・アプリケーションとOracle Database間にあるソフトウェア・レイヤー。

保護対象データベース

リカバリ・アプライアンスでバックアップを管理するクライアント・データベース。

保護ポリシー

リカバリ・アプライアンスがバックアップ・データを保存し、維持する方法を制御する属性グループ。保護対象の各データベースは、そのクライアントのバックアップに関する処理をすべて制御する、1つの保護ポリシーに割り当てられます。

疑似列

表に格納されないが表の列のように動作する列。

問合せ

表またはビューからデータを取り出す操作。たとえば、SELECT * FROM employeesは問合せです。

関連項目:「暗黙的問合せ」、「副問合せ」

問合せブロック

最上位のSELECT文、副問合せ、またはマージされていないビュー。

問合せコーディネータ

パラレル実行において、パラレル実行サーバーを調整するユーザー・セッションまたはシャドウ・プロセス。パラレル実行サーバーは可能な場合には各操作をパラレルで実行します。パラレル・サーバーで文の実行が終了すると、問合せコーディネータは、パラレルで実行できない作業の部分を実行します。最後に問合せコーディネータが結果をユーザーに返します。

問合せの最適化

SQL文を実行する最も効率的な方法を選択するプロセス。

問合せ計画

問合せの実行に使用される実行計画。

クエリー・リライト

マスター表について記述されたユーザー要求を、マテリアライズド・ビューを含む同等の要求に変換する最適化手法。

問合せトランスフォーマ

元のSQL文を、意味的に同等でよりコストの低いSQL文にリライトすることが有利かどうかを判断するオプティマイザ・コンポーネント。

R

統計計算およびグラフィック用の言語および環境。

レンジ・パーティション化

データベースがパーティション化キーの値の範囲に基づいて、行をパーティションにマップするパーティション化の一種。レンジ・パーティション化は、最も一般的なタイプのパーティション化であり、通常は日付とともに使用されます。

RDMA

リモート・ダイレクト・メモリー・アクセス。クライアントが、サーバー側のCPUを消費せずにネットワーク経由でデータベース・サーバーのメモリー(DRAM)からデータを転送できるようにするための通信プロトコル。

コミット読取り分離レベル

トランザクションによって実行される問合せにおいて、トランザクション開始前ではなく、問合せ開始前にコミット済のデータのみが参照可能になることを保証する分離レベル。

読取り一貫性

ユーザーから見たデータの一貫したビュー。たとえば、文レベルの読取り一貫性では、SQL文から見たデータの集合は、文の実行中に不変です。

「データ同時実行性」、「データ整合性」も参照してください。

読取り/書込みデータベース・インスタンス

DMLの処理が可能で、直接クライアント接続をサポートしているデータベース・インスタンス。デフォルトでは、データベース・インスタンスは読取り/書込みです。

読取り専用データベース

問合せのみに使用でき、変更できないデータベース。

読取り専用データベース・インスタンス

DMLを処理できず、クライアント接続をサポートしていないデータベース・インスタンス。

読取り専用分離レベル

シリアライズ可能分離レベルと類似していますが、読取り専用トランザクションではSYSユーザー以外はトランザクションでデータを変更できないという違いがある分離レベル。

リアルタイムREDOトランスポート

保護対象データベースのSGAからリカバリ・アプライアンスへのREDO変更の継続的な転送。リアルタイムREDOトランスポートでは、RMANからのリカバリ・ポイントの目標を0前後で提供できます。一般に、障害発生時にRMANは1秒以内にリカバリできます。保護対象データベースは、生成時にSGAからリカバリ・アプライアンスへREDOエントリを直接書き込みます。

リカバリ可能なエラー

実行中のアプリケーション・セッション・ロジックとは関係なく、外部システムの障害のために生じるエラーの種類。リカバリ可能なエラーは、ネットワーク、ノード、記憶域およびデータベースの計画および計画外停止の後に発生します。リカバリ不可能なエラーの例は、無効なデータ値の送信です。

リカバラ・プロセス(RECO)

分散データベースで、分散トランザクションの障害を自動的に解決するバックグラウンド・プロセス。

リカバリ・アプライアンス

Zero Data Loss Recovery Applianceの短縮名。リカバリ・アプライアンスはOracleデータベースを保護するために設計された専用のOracle Engineered Systemです。RMANに統合されており、クラウド規模の耐障害性の高いハードウェアと記憶域を使用して、企業全体にわたる何百から何千ものデータベースを対象とした、一元的な永久的増分バックアップ計画が可能です。

リカバリ・アプライアンス・バックアップ・モジュール

RMANが保護対象データベースのバックアップをネットワーク経由でリカバリ・アプライアンスに送信する、Oracle提供のSBTライブラリ。ライブラリは保護対象データベースで使用するOracleホームごとにインストールする必要があります。

このモジュールは、リカバリ・アプライアンスへのバックアップのチャネル割当てまたは構成の際に、RMANが参照するSBTメディア管理ライブラリとして機能します。RMANはこのモジュールを使用して、リカバリ・アプライアンスへのすべてのバックアップ、および完了したバックアップ・セットすべてのリストアを実行します。

リカバリ・アプライアンス・メタデータ・データベース

リカバリ・アプライアンスの内部で実行されるOracleデータベース。このデータベースには、ユーザー定義、保護ポリシー定義、クライアント・データベース定義などの構成データが格納されます。メタデータ・データベースには他にも、デルタ格納のコンテンツを含むバックアップ・メタデータが格納されます。

リカバリ・アプライアンスの記憶域の場所

バックアップを格納するリカバリ・アプライアンス内のOracle ASMディスク・グループ・セット。記憶域の場所は、複数の保護対象データベースで共有できます。すべてのリカバリ・アプライアンスには、DELTAという名前のデフォルトのリカバリ・アプライアンスの記憶域の場所があります。

リカバリ・カタログ

Oracleデータベースにある集中管理されたバックアップ・レポジトリ。リカバリ・カタログにはRMANバックアップのメタデータが含まれます。

Recovery Manager (RMAN)

「RMAN」を参照してください。

リカバリ・ウィンドウ目標

ビジネス要件を満たすために、この期間内に保護されたデータベースをリカバリできる必要があります。保護ポリシー内の保護されたデータベースごとに、リカバリ・アプライアンスでは、現在の時間から遡って指定された間隔内(過去7日間など)の任意の時点までのポイント・イン・タイム・リカバリを、ディスクで最古のバックアップによってサポートできるようにしようとします。

再帰的SQL

データベース・オブジェクト用の領域を取得するため、データベースでバックグラウンドで実行されるSQL。再帰的SQLは副作用SQLと考えることができます。

REDOログ

データファイルに書き込まれていないメモリー内の変更済データベース・データを保護する一連のファイル。REDOログは、オンラインREDOログとアーカイブREDOログという2つの部分からなる場合があります。

REDOログ・バッファ

REDOエントリ(データベースに対して行われた変更のログ)を格納するSGAのメモリー構造。データベースは、REDOログ・バッファに格納されたREDOエントリをオンラインREDOログ・ファイルに書き込み、インスタンス・リカバリが必要になったときにこのファイルを使用します。

REDOレコード

変更ベクトル(データベース内の単一ブロックに加えられた変更を記述したもの)のグループを保持するオンラインREDOログ内のレコード。各REDOログ・ファイルは、REDOレコードで構成されます。

REDOスレッド

データベース・インスタンスによって生成されるREDO。

参照パーティション化

子表が親表との外部キー関係のみによって定義されるパーティション化計画。親表内の各パーティションごとに、対応する1つのパーティションが子表内に存在します。

参照キー

外部キーの関係において、外部キーが参照する主キーまたは一意キー。たとえば、共通スキーマ内で、`employees.department_id`列は外部キー、`departments.department_id`列は参照キーです。

参照オブジェクト

スキーマ・オブジェクト依存性で、別のオブジェクト定義で参照されるオブジェクト。たとえば、オブジェクトAの定義がオブジェクトB

を参照している場合、BはオブジェクトAの参照オブジェクトになります。

参照整合性

1つの表のキーに対して定義される規則であり、そのキーの値が関連する表のキーの値(参照値)と一致することを保証します。

リレーション

一連のタプル。

リレーショナル・データベース

データを一連の単純なリレーションに格納する、リレーショナル・モデルに準拠するデータベース。

リレーショナル・データベース管理システム(RDBMS)

データをリレーショナル・データベースに移動して、アプリケーションが操作できるようにそのデータを取得する管理システム。

リプレイ・コンテキスト

アプリケーション・コンティニュイティにおいて、通常のアプリケーション実行時にデータベースがクライアント・ドライバに戻す不透明な情報。

レプリケーション

複数のデータベースでデータベース・オブジェクトおよびデータを共有する処理。

予約プール

Oracle Databaseで大量の連続したメモリーのチャンクを割り当てるために使用できる共有プール内のメモリー領域。

リソース計画

リソース・コンシューマ・グループへのリソースの割当て方法を指定するリソース・プラン・ディレクティブのコンテナ。

リソース・プラン・ディレクティブ

コンシューマ・グループ内のセッションのCPU、物理I/Oまたは論理I/O使用量に対する制限と制御のセット。

リストア・ポイント

データベースのSCNに関連付けられているユーザー定義の名前。SCNはリストア・ポイントが作成された時刻に対応します。

結果セット

SELECT文の実行によって取得されるデータ・セット。

逆キー索引

列の順序は保ちながら、各索引キーのバイトを物理的に逆にするBツリー索引の一種。たとえば、索引キーが20であり、このキーに対して標準Bツリー索引で格納される16進数の2バイトがC1, 15である場合、逆キー索引では、バイトが15, C1として格納されます。

右側外部結合

結合条件が左側の表*A*のレコードに一致していない場合でも、表*A*と*B*の右側外部結合の結果には右側の表*B*のすべてのレコードが含まれます。たとえば、employees (左)からdepartments (右)への右側外部結合を実行した場合、一部の部署に従業員がいなくても、問合せではemployeesの一致なしでdepartmentsの行が返されます。

RMAN

Recovery Manager。Oracleデータベースをバックアップ、リストアおよびリカバリするOracle Databaseユーティリティ。

ロール

データベース・ユーザーまたは他のロールに付与できる一連の権限。

行

表の1つのレコードに対応する一連の列情報。データベースでは、行がデータ・ブロックに格納されます。

行連鎖

行が大きすぎて1つのブロックに収まらず、Oracle Databaseが行を一連のブロックまたは連鎖するブロックに格納することが必要になる状況。

行ロック

表の1つの行に対するロック。トランザクションでは、INSERT、UPDATE、DELETE、MERGEまたはSELECT ... FOR UPDATE文によって変更される各行で行ロックを取得します。

行優先形式

ある行のすべての列がまとめて格納され、それ以降の行のすべての列がそれに続く表記憶域のタイプ。

行の移行

行が大きくなりすぎて元のブロックに収まらなくなったためにOracle Databaseが1つのデータ・ブロックから別のデータ・ブロックへ行を移動する状況。

行断片

可変長レコードに格納される行。このレコードは、1つ以上の行断片に分割されます。各行断片は、行ヘッダーおよび列データで構成されます。

行セット

実行計画のステップによって戻される行セット。

行ソース

1組の行を処理し、行セットを生成する反復制御構造。

行ソース・ジェネレータ

オプティマイザから最適な計画を受け取り、SQL文の実行計画を出力するソフトウェア。

行トリガー

トリガー実行文によって表が影響を受けるたびに起動されるトリガー。たとえば、文が複数の行を更新する場合、UPDATEが処理する行ごとに1つずつ行トリガーが起動されます。

ROWID

データベース内の行に対するグローバルな一意のアドレス。

サンプル・スキーマ

共通のデータベース・タスクに関するOracleドキュメントおよび資料を表示できる、連結されたスキーマのセット。

セーブポイント

トランザクション内の名前付きSCN。トランザクションをそのSCNまでロールバックできます。

スケール

浮動小数点数で、小数点から最下位桁までの桁数。固定小数点数は、NUMBER(*p*, *s*)の形式で指定します(*s*はスケールを表します)。

スキーマ

表や索引などの論理構造を含む、データベース・オブジェクトの名前付きコレクション。スキーマには、そのスキーマを所有するデータベースの名前が付きます。

スキーマ・オブジェクト

スキーマに格納されたデータの論理構造。たとえば、スキーマ・オブジェクトには、表、索引、順序、データベース・リンクがあります。

スキーマ・オブジェクトの依存性

別のオブジェクトによるオブジェクトの参照。たとえば、ビューに表またはビューを参照する問合せが含まれている場合や、PL/SQLサブプログラムが他のサブプログラムを起動する場合があります。

SCN

システム変更番号。データベース順序付けプリミティブです。SCNの値は、データベースに対して変更が行われた論理的な時点です。

2次索引

索引構成表の索引。ある意味では、これは索引の索引です。

SecureFiles LOB記憶域

SecureFiles LOB記憶域はLOBのデフォルトの記憶域メカニズムです。SECUREFILE LOBパラメータでは、圧縮と複製解除(拡張圧縮オプションの一部)、および暗号化(拡張セキュリティ・オプションの一部)を含む特定の拡張機能を使用できます。

セキュリティ・ポリシー

過失または悪意によるデータの破壊、またはデータベース・インフラストラクチャの被害からデータベースを保護する一連の手段。

セグメント

表、索引、表クラスタなどの特定のデータベース・オブジェクトに割り当てられた一連のエクステント。セグメントのタイプは、ユーザー・セグメント、UNDOセグメントおよび一時セグメントで、すべてです。

選択リスト

SELECT文で、SELECTキーワードの後、FROM句の前にある式のリスト。

選択性

述語または述語の組合せによって取り出される行セットの割合を示す値。たとえば、WHERE last_name = 'Smith'などです。0を選択すると述語テストに合格する行がないことを意味し、値1を選択するとすべての行がテストに合格することを意味します。

選択性が高いという形容詞は、「選択に厳しい」と大体同じ意味です。このため、選択性の高い問合せでは戻される行の割合が低く(選択性は0に近い)、非選択性の高い問合せでは戻される行の割合は高くなります(選択性は1に近い)。

自己結合

自身への表の結合。

自己参照型整合性制約

外部キーが同じ表の親キーを参照する制約。たとえば、制約によって、employees.manager_id列のすべての値が、employees.employee_id列の既存の値に対応するようになります。

順序

表の列に対して連続する一意の数値のリストを生成するスキーマ・オブジェクト。

シリアル実行

SQL文の順次実行に必要な処理のすべてを1つのサーバー・プロセスが実行します。

シリアライズ可能性

他のユーザーがデータベースのデータを変更していないように認識される環境内でトランザクションが動作するトランザクション分離モデル。

シリアライズ可能分離レベル

トランザクションにおいて、問合せ開始時ではなく、トランザクション開始時にコミット済の変更およびこのトランザクション自体が行った変更のみが参照可能になることを保証する分離レベル。

サーバー

クライアント/サーバー・アーキテクチャで、Oracleソフトウェアを実行し、同時実行の共有データ・アクセスに必要な機能を処理するコンピュータ。サーバーは、クライアント・アプリケーションから発行されたSQL文やPL/SQL文を受け取って処理します。

サーバー・パラメータ・ファイル

データベースによって読み書きされる初期化パラメータ設定を含むサーバー側のバイナリ形式のファイル。

サーバー・プロセス

ユーザー・リクエストを満たすために、クライアント・プロセスおよびOracle Databaseと通信するOracleプロセス。サーバー・プロセスは、データベース・インスタンスと関連付けられていますが、インスタンスの一部ではありません。

サーバー結果キャッシュ

共有プール内のメモリ・プール。このメモリー・プールは、SQL問合せの結果を格納するSQL問合せの結果キャッシュと、PL/SQL関数によって戻される値を格納するPL/SQL関数の結果キャッシュで構成されています。

サービス・ハンドラ

Oracle Netでデータベースへの接続ポイントとなる、ディスパッチャまたは専用サーバー・プロセス。

サービス名

Oracle Netで、クライアント接続に使用するサービスの論理表現。

サービス登録

Oracle Netで、リスナー登録プロセス(LREG)がインスタンス情報を動的にリスナーに登録するために使用する機能。この機能によって、リスナーがクライアント接続リクエストを適切なサービス・ハンドラに送ることができます。

サービス指向アーキテクチャ(SOA)

ネットワーク上のコンピュータ間の対話をサポートするサービスに依存する複数層アーキテクチャ。

セッション

データベースへの現行ユーザー・ログインの状態を表すデータベース・インスタンス・メモリー内の論理エンティティ。1つの接続では、セッションが確立されないか、1つ以上のセッションが確立されます。

SGA

システム・グローバル領域。共有メモリー構造のグループで、1つのOracleデータベース・インスタンスに関するデータと制御情報が含まれます。

シャドウ・エクステント

シャドウ表領域内のエクステント。シャドウ・エクステントと追跡データファイルとの間には1対1のマッピングが存在します。

シャドウ書込み欠落保護

シャドウ表領域を使用して書込み欠落を自動的に任意の追跡データ・ファイルで検知する、データ保護機能。

シャドウ表領域

シャドウ書込み欠落保護に有効化されている表領域。表領域は、追跡データ・ファイルに格納されているデータ・ブロックのSCNおよび他のメタデータを追跡します。単一のシャドウ表領域に、多数の通常データ・ファイルの記述データが含まれている場合があります。

シャード

シャーディング構成に関係する単一のデータベース。

シャード・カタログ・データベース

シャード・データベース(SDB)の構成データを格納し、その他の機能(シャード間の問合せや集中管理など)を提供するデータベース。

シャード・ディレクタ

グローバル・サービス・マネージャを使用してアプリケーション層から個々のシャードへのリクエストの直接ルーティングを提供するGDSインフラストラクチャ・コンポーネント。

シャード・データベース(SDB)

シャーディング・アーキテクチャにおいて、単一の論理データベースとしてアプリケーションに示されるシャードの集合。

シャード表

シャード・データベース(SDB)間で、同じ列と異なる行のサブセットを持つ表が各シャードに含まれるように水平に分割された表。

シャーディング

独立したデータベース間でデータが水平にパーティション化されるデータ階層アーキテクチャ。シャードはCPU、メモリまたはストレージ・デバイスなどの物理リソースを共有しないため、シャーディングはシェアードナッシング・データベース・アーキテクチャです。シャードはソフトウェアの観点でも疎結合されています。クラスタウェアが実行されないためです。

シャーディング・キー

シャード表に対するパーティション化キー。

共有ロック

操作の種類に応じて、複数のトランザクションで関連するリソースの共有が可能となるロック。複数のトランザクションが同じリソースについて共有ロックを取得できます。

共有プール

共有SQL領域などの共有メモリー構成メンバーを含むSGAの一部。

共有サーバー

複数のクライアント・プロセスが少数のサーバー・プロセスを共有できるデータベース構成。

関連項目:「専用サーバー」

共有SQL領域

SQL文の解析ツリーと実行計画を含む共有プール内の領域。一意の文に対して、共有SQL領域は1つのみ存在します。

共有一時表領域

共有記憶域に存在し、すべてのデータベース・インスタンスによるアクセスが可能な一時表領域。Oracle Database 12cリリース2 (12.2)から、一時表領域は共有またはローカルのいずれかになりました。前のリリースでは、すべての一時表領域が共有一時表領域でした。

共有UNDOモード

CDB全体に対するUNDOデータ・ファイルの単一セットの使用。

単純データベース操作

単一のSQL文、あるいは単一のPL/SQLプロシージャまたは機能。

単純なトリガー

1つのタイミング・ポイントのみに対してアクションを指定できる、表のトリガー。たとえば、トリガー実行文の前にトリガーを起動する場合があります。

単一レベル・パーティション化

リスト・パーティションのみまたはレンジ・パーティションのみなど、データ配分方法を1つのみ使用するパーティション化計画。

サイト障害

アプリケーションのすべてまたは大部分で処理が停止させるか、使用できないサービス・レベルにまで低下させるイベント。

smallfile表領域

複数のデータファイルや一時ファイルを含むことができますが、bigfile表領域ほど大きなファイルは含むことができない表領域。

ソフト解析

実行依頼されたSQL文の解析済の表現が共有プールに存在し、共有できる場合に行われる、既存コードの再利用。

関連項目:「ハード解析」

ソフトウェア・コード領域

実行中または実行できるコードを保管するメモリーの部分。

ソート済のハッシュ・クラスタ

データベースがソートされた順序で効率的に行を戻すことが可能になるように、ハッシュ関数の各値に対応する行を格納するハッシュ・クラスタ。データベースは最適化されたソートを内部的に実行します。

SQL

構造化問合せ言語。リレーショナル・データベースにアクセスするための非手続き型言語です。ユーザーが処理内容をSQLで記述すると、SQL言語コンパイラはデータベースをナビゲートし、そのタスクを実行するプロシージャを自動的に生成します。
Oracle SQLには、ANSI/ISO標準SQL言語に対する拡張機能が多数含まれています。

関連項目:「SQL*Plus」、「PL/SQL」

SQL Developer

Javaで記述された、SQL*Plusのグラフィカル版ツールで、SQLおよびPL/SQLの開発を支援します。

SQLの解析

SQL文の各部分を他のルーチンで処理できるデータ構造へ分解するSQL処理の段階。

SQL計画ベースライン

SQL計画管理で、繰返し可能なSQL文に対する1つ以上の承認済の計画のセット。SQL計画ベースラインの効果は、オプティマイザによる選択がベースライン内の確認済計画に限定されることです。

SQL計画管理

データベースで確認済の計画のみが使用されるように、オプティマイザで実行計画を自動的に管理できる予防的なメカニズム。

SQLプロファイル

SQL文の自動チューニング中に作成される補助的な情報セット。SQLプロファイルとSQL文の関係は、統計と表の関係と同様です。オプティマイザは、SQLプロファイルを使用して、カーディナリティおよび選択性の見積りを改善し、より適切な計画を選択できるようになります。

SQL問合せ結果キャッシュ

問合せと問合せフラグメントの結果を格納するサーバー結果キャッシュのサブセット。

SQL*Plus

Oracle Databaseに対してSQL文を実行するために使用するOracleツール。

SQLJ

JavaプログラムにSQL文を組み込むためのANSI標準。SQLJプログラムをJDBCと組み合せることができます。

スタンバイ・データベース

高可用性環境で障害時の保護に使用できる、本番データベースの独立したコピー。

スター・スキーマ

ディメンション・データ・モデルを表すように設計されたリレーショナル・スキーマです。1つ以上のファクト表と、外部キーを介して関連付けられている1つ以上のディメンション表で構成されます。

「ディメンション表」および「ファクト表」も参照してください。

状態オブジェクト

SGAのプロセス、セッションおよびトランザクションなどのデータベース・リソースのステータスに関するメタデータが含まれるセッション・レベル構造。

文トリガー

トリガーを実行する文の影響を受ける行数とは関係なく、トリガーを実行する文の実行ごとに1回起動されるトリガー。

文レベルの原子性

SQL文がアトミックな作業単位であり、完全に成功するか、完全に失敗するかのいずれかであるという特性。

文レベルの読取り一貫性

1つの問合せによって戻されるデータは、コミット済であり、ある特定の時点での一貫性を保っているという保証。

文レベルのロールバック

実行中にエラーが発生したために失敗したSQL文の効果がロールバックされるデータベース操作。

ストアド・プロシージャ

Oracle Databaseによってデータベースに格納される名前付きPL/SQLブロックまたはJavaプログラム。アプリケーションはストアド・プロシージャを名前で呼び出すことができます。

ストリーム・プール

バッファ・キュー・メッセージが格納されるメモリー・プール。

Structured Query Language(SQL)

「SQL」を参照してください。

副問合せ

別のSQL文内にネストされた問合せ。暗黙的な問合せとは異なり、副問合せはSELECT文を使用してデータを取得します。

サマリー

データ・ウェアハウス内の集計ビューで、結合および集計操作を事前に計算して結果を表に格納することで、問合せ時間が短縮されます。

サロゲート・キー

システムで生成され、値が増加する識別子で、これによって表内での一意性が保たれます。通常、サロゲート・キーは順序によって生成されます。

シノニム

スキーマ・オブジェクトの別名。シノニムを使用してデータの独立性と位置の透過性を提供できます。

システム変更番号(SCN)

「SCN」を参照してください。

システム・イベント・トリガー

データベース・インスタンスの起動および停止またはエラー・メッセージなどのイベントによって起動されるイベント・トリガー。

システム・グローバル領域(SGA)

「SGA」を参照してください。

システム・モニター・プロセス(SMON)

インスタンス・リカバリ、インスタンス・リカバリ中にスキップされた停止後のトランザクションのリカバリ、未使用の一時セグメントのクリーンアップ、ディクショナリ管理の表領域内の連続空きエクステントの結合など、システム・レベルの様々なクリーンアップ処理を実行するバックグラウンド・プロセス。

表

Oracle Databaseにおけるデータ記憶域の基本単位。表のデータは、行と列に格納されています。

表クラスタ

すべてが1つ以上の共通する列を持つ、1つ以上の表のデータを含むスキーマ・オブジェクト。表クラスタでは、データベースは同じクラスタ・キーを共有するすべての表について、すべての行をまとめて格納します。

表の圧縮

ヒープ構成表または表パーティションのディスク領域を削減するために行うデータ・セグメントの圧縮。

表関数

行の集合(ネストした表またはVARRAY)を戻すユーザー定義のPL/SQL関数。SELECT文のTABLE句の内部で表関数を起動することで、データベース表のようにこのコレクションから要素を選択できます。

表ロック

INSERT、UPDATE、DELETE、MERGE、SELECT ... FOR UPDATEまたはLOCK TABLE文で表を変更する場合にトランザクションに取得される、表に対するロック。

表領域

関連する論理構造をグループ化するデータベース記憶域。データベースのデータファイルは表領域に格納されます。

表領域セット

Oracle Shardingにおいて、シャード・データベース(SDB)間で分散され、1つの単位として管理される表領域。

一時ファイル

一時表領域に属するファイル。一時表領域の一時ファイルには、永続的なデータベース・オブジェクトが含まれることはありません。

一時セグメント

SQL文の実行を完了するために一時的なデータベース領域が必要なときに、Oracle Databaseによって作成されるセグメント。

一時表

トランザクションまたはセッションの継続時間中に中間結果セットを保持する表。現行セッションのみが一時表のデータを参照で

きます。

一時表領域

セッションの存続期間中のみ保持される一時データのみを含むことができる表領域。永続スキーマ・オブジェクトを一時表領域に格納することはできません。

一時表領域はすべて、共有一時表領域またはローカル一時表領域のいずれかです。特に明記しないかぎり、一時表領域という用語は共有一時表領域を意味します。

一時UNDOセグメント

一時UNDOデータ専用のオプションの領域管理コンテナ。

トレース・ファイル

問題の調査に使用される診断データを含む管理用のファイル。Oracle Databaseでは、トレース・ファイルをADRに書き込みます。

追跡データ・ファイル

シャドウ表領域で追跡されるブロックSCNを保有するデータ・ファイルで、これによってシャドウ書込み欠落保護が有効になります。

トランザクション

1つ以上のSQL文を含む論理的な作業単位。トランザクション内のすべての文は、まとめてコミットまたはロールバックされます。トランザクションの使用は、データベース管理システムとファイルシステムの最も重要な違いの1つです。

トランザクション・エントリ

ブロックを更新する各トランザクションに必要な、ブロック・ヘッダー内の領域。トランザクションによる変更をサポートするセグメントに割り当てられたデータ・ブロックでは、ヘッダー領域の空きがなくなると、トランザクション・エントリが空き領域にも保持されます。

トランザクション・ガード

リカバリ可能なエラーの後にクライアント・アプリケーションが重複トランザクションを発行しないようにするための論理トランザクションIDを使用したデータベース機能。

トランザクションID

識別子はトランザクションに対して一意であり、UNDOセグメント番号、スロットおよび順序番号を表します。

トランザクションの冪等性

コミット済かどうか、コールが完了したかどうかに関係なく、トランザクションの保証された結果を戻す機能。

トランザクション名

トランザクションが実行する作業のリマインダとして機能する、ユーザー指定のオプションのタグ。トランザクションに命名するには SET TRANSACTION ... NAME文を使用します。

トランザクション・リカバリ

コミットされていないトランザクションがロールバックされるインスタンス・リカバリのフェーズ。

トランザクション表

UNDOセグメント内のデータ構造で、UNDOセグメントを使用するトランザクションのトランザクション識別子を保持します。

トランザクション・レベルの読取り一貫性

トランザクションのすべての問合せに対する読取り一貫性の保証。トランザクション内の各文には同じ時点(トランザクション開始時)のデータが表示されます。

遷移ポイント

レンジ・パーティション・キーの値によって決定される、レンジ・パーティションの上限。

透過的データ暗号化

個々の表列または表領域を暗号化するデータベース機能。ユーザーが暗号化された列にデータを挿入すると、データベースによって自動的にデータが暗号化されます。ユーザーが列を選択すると、データは復号化されます。このような形式の暗号化は透過的であり、パフォーマンスが高く、実装が容易です。

トランスポータブル表領域

データベース間でコピーまたは移動できる表領域。Oracle Data Pumpは、トランスポータブル表領域のインフラストラクチャを提供します。

トリガー

表またはビューが変更されたとき、または特定のユーザー・アクションまたはデータベース・アクションが発生したときに起動されるPL/SQLまたはJavaのプロシージャ。プロシージャが明示的に実行されるのに対し、トリガーは暗黙的に実行されます。

タプル

順不同の一連の属性値。

2フェーズ・コミット・メカニズム

分散トランザクションに関係するすべてのデータベースが、そのトランザクション内の文のすべてをコミットするか、すべてを取り消すかのどちらか一方のみになるように保証する、分散データベースのメカニズム。

UGA

ユーザー・グローバル領域。ログオン情報などのセッション変数が格納されるセッション・メモリーであり、OLAPプールを含むこともできます。

UNDOデータ

トランザクション・アクションのレコードで、基本的にはコミットされる前のもの。データベースでは、UNDOデータを使用してSQL文の効果を論理的に元に戻すことができます。UNDOデータはUNDOセグメントに格納されます。

UNDO保存期間

古いUNDOデータを上書きしないで保存する最低期間。

UNDOセグメント

UNDO表領域内のセグメント。

UNDO表領域

自動UNDO管理モードが有効になっている場合にUNDOセグメントを含む表領域。

Unicode

1つの文字セットで任意の言語の情報を格納できるユニバーサル・エンコードされた文字セット。

統合監査ポリシー

SQL文、システム権限、スキーマ・オブジェクト、ロール、管理ユーザーと非管理ユーザー、アプリケーション・コンテキスト値および様々なアプリケーションやイベントのポリシー作成の監査を構成するために使用できるポリシー。

統合監査証跡

監査証跡は、すべてのタイプの監査での監査レコードの、統一された記憶域を提供します。

一意キー

一意キー制約のある1つの列。

一意キー制約

列(または列の集合)のすべての値が一意であることを必要とする整合性制約。

ユニバーサルROWID

すべてのタイプのROWIDを格納できるデータ型。Oracleでは、索引編成表とOracle以外の表のアドレスはユニバーサルROWIDを使用して格納されます。

使用禁止索引

DML操作によって管理されず、オプティマイザにより無視される索引。すべての索引は、使用可能(デフォルト)または使用禁止です。

更新可能な結合ビュー

2つ以上の実表またはビューで定義され、DML操作が可能なビュー。

ユーザー・イベント・トリガー

ユーザーのログオンおよびログオフに関連したイベント、DDL文およびDML文によって起動されるイベント・トリガー。

ユーザー・グローバル領域(UGA)

「UGA」を参照してください。

ユーザー名

Oracle Databaseおよび他のユーザーがユーザーを識別するための名前。すべてのユーザー名にはパスワードが関連付けられており、Oracle Databaseに接続するときはユーザー名とパスワードの両方を入力する必要があります。

ユーザー権限

特定のSQL文を実行できる権利。

ユーザー・プロセス

「クライアント・プロセス」を参照してください。

ユーザー・プロファイル

ユーザーのデータベース使用とデータベース・インスタンス・リソースを制限する、リソース制限とパスワード・パラメータをセットにして名前を付けたもの。

ビュー

1つ以上の表のデータをカスタム調整したデータ表現。ビューは、実際にデータが含まれたり、格納されたりすることはなく、基になる表から導出されます。

仮想列

ディスク上に格納されない列。データベースは、必要に応じて、一連の式または関数を計算することによって仮想列の値を導出します。

仮想完全バックアップ

保護対象データベースから増分バックアップを索引付けして、リカバリ・アプライアンスで効率的に維持される、ある特定の時点の完全なデータベース・イメージ。仮想完全バックアップには、複数の増分バックアップからの個別のブロックが含まれます。たとえば、月曜日にレベル0のバックアップSCN 10000を作成し、火曜日に増分レベル1のバックアップSCN 11000を作成すると、リカバリ・アプライアンス・メタデータ・データベースでは、最新の仮想レベル0バックアップがSCN 11000になります。

ウェアハウス圧縮

COLUMN STORE COMPRESS FOR QUERYで指定されるハイブリッド列圧縮。このタイプの圧縮は、データ・ウェアハウスで有効です。

データベース全体のバックアップ

データベースに属するすべての制御ファイルおよびデータファイルのバックアップ。

作業領域

メモリー集中型の操作に使用されるPGAメモリーを個別に割り当てるための領域。

事前書込みプロトコル

データベース書込みプロセスが使用済バッファを書き込むには、バッファの変更に関連するすべてのREDOレコードがディスクに書き込まれていることを要求するプロトコル。

ゾーン

ゾーン・マップ内において、ゾーンは関連する列の最小値および最大値を格納する、一連の隣接するデータ・ブロックです。

ゾーン・マップ

属性クラスタ表内では、ゾーン・マップはデータ・ブロックをゾーンに分割する、独立したアクセス構造です。

索引

A

- アクセス・ドライバ, 外部表 2.6.3
- アクセス・パス, データ 3.1.1, 3.4.2, 7.3.1, 7.3.3
- アカウント, ユーザー 6.1
- ACIDプロパティ 10.1
- アクティブなトランザクション 10.2.2
- ADDM(自動データベース診断モニター) 18.3.4.3, 18.3.4.4
- 管理アカウント 2.1.4, 6.1.1
- 管理者権限 2.1.4, 13.2.1.1, 16.2.4.2, 17.1.1.1
- ADR 13.6.1
- 拡張索引圧縮 3.2.5.2
- アドバンスト・キューイング 17.5.2.2
- 集計ファンクション 17.4.4.1
- アラート・ログ 13.6.2, 15.4.1.6.2
- ALL_データ・ディクショナリ・ビュー 6.1.1.2
- ALTER TABLE文 2.2.3
- 分析ファンクション 17.4.4.1
- 分析ビュー 17.4.4.2
- 無名PL/SQLブロック 8.2
- ANSI/ISO規格 7.1.2
- API(Application Program Interface) 16.3
    - クライアント側 19.3.2.2
    - 埋込みSQL文 7.2.6
    - 外部表 2.6.3
    - Java 8.3.1.1, 19.3.2.1.2, 19.3.2.2.2
    - JDBC 7.2.6, 10.4.2, 19.3.2.2.2
    - ネットワーク・サービス 16.2.1
    - OCI/OCCI 19.3.2.2.1
    - ODBC 7.2.6, 19.3.2.2.2
    - Oracle Database Advanced Queuing 17.5.2.2.2
    - Oracle Data Pump 18.2.5.2
- アプリケーションおよびネットワークのアーキテクチャ 1.5.4.1, 16
- アプリケーション・コンテキスト, 監査 17.1.6.1
- アプリケーション・コンティニュイティ 10.4, 17.2.1.4
    - アーキテクチャ 10.4.2
    - 利点 10.4.1
    - 計画的メンテナンス 10.4.1.2
    - ユースケース 10.4.1.1
- アプリケーション開発者

    - 業務 19.1
    - ツール 19.2
    - トピック 19.3
- アプリケーション・ドメイン索引 3.5
- アプリケーション・プロセス 15.2
- アプリケーション・プログラム・インタフェース
    - 「API」を参照
- アプリケーション
    - アップグレード 17.2.2.3
- アプリケーション・サーバー 1.5.4.1, 8.3.1.2
    - 概要 16.1.2.2
- アーカイブREDOログ・ファイル 11.3.3, 11.4.2.3, 18.3.1.4.4
- ARCHIVELOGモード 15.4.2.1
- アーカイバ・プロセス(ARCn) 15.4.2.1
- 昇順索引 3.2.4
- ASSM表領域 12.1.2.1.1
- 非同期通知 17.5.2.2.2
- 原子性, 文レベル 10.1.3
- 属性クラスタ表
    - 利点 2.4.1
    - ディメンション階層 2.4.5
    - インターリーブされた順序付け 2.4.5
    - 結合属性クラスタリング 2.4.2
    - 線形順序付け 2.4.4
    - 概要 2.4
    - ゾーン・マップ 2.4.3, 2.4.3.1
- 監査 6.1, 6.1.3, 7.2.1, 8.4.1, 11.1.3, 13.2.1.4.1, 17.1.6.1
    - アプリケーション・コンテキスト 17.1.6.1
    - 監査管理者ロール 17.1.6.1.2
    - 監査ポリシー 17.1.5.3, 17.1.6.1.1
    - 監査レコード 17.1.6.2
    - ファイングレイン 17.1.6.1
    - Oracle Audit Vault and Database Firewall。 17.1.6.4
    - Oracle Database Vault 17.1.5.1
    - Oracle Label Security 17.1.5.3
    - 標準 17.1.6.1
    - 統合監査証跡 16.1.2.3, 17.1.5.3, 17.1.6.2
- AUDIT文 7.2.1
- 認証, データベース 7.2.4, 15.2.2, 17.1.2
- 自動索引 3.1.1, 3.1.7
- 自動ビッグ・テーブル・キャッシング 14.4.1.4.1, 14.4.1.5
- 自動データベース診断モニター。
    - 「ADDM」を参照
- 自動診断リポジトリ(ADR) 13.6.1, 13.6.2
- 自動メンテナンス・タスク 18.3.4.2

- 自動メモリー管理 18.3.2.1, 18.3.2.2
- 自動セグメント領域管理(ASSM) 12.1.2.1.1
- 自動UNDO管理 12.4.3, 12.5.1.3.1
- 自動ワークロード・リポジトリ(AWR) 18.3.4.2
- AutoTask 18.3.4.2
- AWR
  - 「自動ワークロード・リポジトリ(AWR)」を参照
- AWRレポート 18.3.4.2

B

- バックグラウンド・プロセス 1.5.3.1, 15.4
  - 必須 15.4.1
  - オプション 15.4.2
  - PMAN 15.4.1.2
- バックアップおよびリカバリ
  - 定義 18.3.1
  - 永久的増分計画 18.3.1.5.1
  - 方法 18.3.1.1
- バックアップ 18.3.1
  - バックアップ・セット 18.3.1.3.3
  - イメージ・コピー 18.3.1.3.3
  - 部分データベース 18.3.1.3.1
  - Recovery Manager 18.3.1.1
  - 方法の比較 18.3.1.1
  - データベース全体 18.3.1.3.1
- ビッグ・テーブル・キャッシュ 14.4.1.5
- BINARY_DOUBLEデータ型 2.2.4.2.2
- BINARY_FLOATデータ型 2.2.4.2.2
- ビットマップ索引 3.3
  - ビットマップ結合 3.3.2
  - ロック 3.3
  - マッピング表 3.6.3.2
  - 単一表 3.3.1
  - ストレージ 3.3.3
- ビットマップによる表領域管理 12.1.2.1
- ブロックしているトランザクション 9.2.1.2
- ブロック, データ
  - 「データ・ブロック」を参照
- BOOLEANデータ型 2.2.4, 3.2.2.3, 7.2.2.1, 8.4.4
- ブランチ・ブロック, 索引 3.2.1
- Bツリー索引 2.2.6.3, 2.3.1, 3.2
  - ブランチ・レベル 3.2.1
  - 高さ 3.2.1

    - 接頭辞圧縮 3.2.5.1
    - 逆キー 3.2.3
- バッファ・キャッシュ, データベース
    - 「データベース・バッファ・キャッシュ」を参照
- バッファ
    - 「データベース・バッファ」を参照
- ビジネス・ルール, 規定 5.1.1, 5.1.2

C

- キャッシュ・フュージョン 9.1.1.3
- カーディナリティ, 列 3.3, 7.3.2.2
- デカルト結合 7.2.2.2
- カートリッジ 3.5
- 連鎖的な削除 5.2.4.3
- catalog.sqlスクリプト 6.2.1
- CDB 1.5.1.1
    - Oracle Flashback Technology 18.3.1.4.1
- 連鎖, 行
    - 「行連鎖」を参照
- 文字データ型 2.2.4.1
    - VARCHAR2 2.2.4.1
- 文字セット 2.2.4.1
    - ASCII 2.2.4.1
    - クライアント 19.3.3.1.1
    - データベース 19.3.3.1.1
    - EBCDIC 2.2.4.1
    - Unicode 2.2.4.1, 19.3.3.1.1
- CHARデータ型 2.2.4.1.1
- チェック制約 5.2, 5.2.5
- チェックポイント・プロセス(CKPT) 15.4.1.7
- チェックポイント
    - 制御ファイル 11.3.1
    - データベース停止 13.2.2.1
    - 定義 13.3
    - 非一貫性バックアップ 18.3.1.3.2
    - 増分 13.3.2
    - 位置 13.4.4
    - スレッド 13.3.2
- チャンク 4.2.1
- クリーンアップ・ヘルパー・プロセス(CLnn) 15.4.1.1.3
- クリーンアップ・メイン・プロセス(CLMN) 15.4.1.1.2
- クライアント/サーバー・アーキテクチャ 16.1.1
    - 利点 16.1.1.2

- クライアント・プロセス 1.5.3.1, 15.2
  - 接続 15.2.2
  - セッション 15.2.2
  - 共有サーバー・プロセス 16.2.4.1.2
- クライアント結果キャッシュ 14.4.3.3
- クライアント側プログラミング 8.1
- CLMNバックグラウンド・プロセス 15.4.1.1.2
- CLnnバックグラウンド・プロセス 15.4.1.1.3
- Cloud Control 18.2.1.1
- クラスタ索引 2.3.1
- クラスタ, 表 2.3.2
  - クラスタ・キー 2.3
  - ハッシュ 2.3.2
  - 索引 2.3.2
- Codd, E. F. 1.1.2
- コード・ポイント 2.2.4.1
- コレクション, PL/SQL 8.2.5, 8.2.5.1
- 列
  - カーディナリティ 3.3, 7.3.2.2
  - 定義 1.2.1, 2.2.1
  - 非表示 2.2.1.2
  - 複数索引 3.1.4
  - 順序 2.2.6.1
  - NULLの使用禁止 5.2.1
  - 仮想 2.2.1.1, 2.2.6.1, 3.4.2
- COMMENT文 7.2.1
- COMMIT文 7.2.3
- トランザクションのコミット
  - COMMIT文 7.2.3
  - 定義 10.1
  - トランザクションの終了 10.1.2.2
  - 高速コミット 15.4.1.6.1
  - グループ・コミット 15.4.1.6.1
  - 実装 15.4.1.6.1
  - 暗黙的コミット 7.2.1
  - 失われたコミット問題 10.3.2.1
  - 2フェーズ・コミット 10.6.1
- コンパイル済PL/SQL
  - 疑似コード 8.4.6
  - 共有プール 8.2.6
  - トリガー 8.4.6
- 完全リカバリ 18.3.1.4.4
- 完全リフレッシュ 4.4.2.1
- コンポジット索引 3.1.4
- コンポジット・パーティション化 4.1.1.2.6

- 複合トリガー 8.4.3
- 圧縮
  - 拡張索引 3.2.5.2
  - アーカイブ 2.2.7.2.1
  - 基本表 2.2.7.1
  - データ・ブロック 12.2.3
  - ハイブリッド列圧縮 2.2.7.2, 2.2.7.2.1
  - 索引 3.2.5
  - OLTP表 2.2.7.1
  - 接頭辞索引 3.2.5.1
  - 表 2.2.7, 4.1.2.2
  - ハイブリッド列圧縮のユニット 2.2.7.2.2
  - ウェアハウス 2.2.7.2.1
- 連結索引 3.1.4
- 同時実行性
  - 定義 9.1
  - 内容を保証しない読取り 9.1.3
  - ファジー読取り 9.1.3
  - 仮読取り 9.1.3
  - 行ロック 9.4.1.1.1
  - トランザクション分離 9.1.3, 9.2.2, 9.2.3
- 条件, SQL 7.1.1, 7.2.2.1
- 書込みの競合 9.2.1.2
- 接続, クライアント/サーバー
  - 管理者権限 13.2.1.1
  - 定義 15.2.2
  - 埋込みSQL 7.2.6, 19.3.2.1
  - リスナー・プロセス 1.5.4.2, 16.2.2
  - セッションと対比 15.2.2
- 一貫性
  - 書込みの競合 9.2.1.2
  - 定義 9.1
  - ロック 9.1.2, 9.5
  - マルチバージョン 9.1, 9.1.1
- 制約, 整合性
  - チェック 5.2, 5.2.5
  - デフォルト値 5.3.3.1
  - 遅延可能 5.2.3, 5.3.2.2, 7.2.3
  - 有効化および無効化 5.3.1
  - 索引による規定 5.2.3
  - 外部キー 5.2, 5.2.4, 5.2.4.4
  - 規定のメカニズム 5.3.3
  - 遅延不可 5.3.2.1
  - NOT NULL 2.2.3, 5.2, 5.2.1
  - 主キー 2.2.3, 5.2, 5.2.3

    - REF 5.2
    - 参照 5.2.4.3
    - 自己参照型 5.2.4.1
    - 状態 5.3
    - 一意キー 5.2, 5.2.2
    - 外部キーおよび親キーの更新 5.3.3.2
    - 検証 5.3.1, 5.3.2
- 競合
    - デッドロック 9.3.6
- コンテキスト 2.1.1
- 制御ファイル 1.5.2.1, 11.3
    - 記録される変更内容 11.3.1
    - チェックポイント 11.3.1
    - 目次 11.3.1
    - 多重化 11.3.2
    - 概要 11.3
    - データベースのマウントでの使用 13.2.1.3
- コストベース・オプティマイザ
    - 「オプティマイザ」を参照
- CREATE CLUSTER文 2.3.1
- CREATE DIMENSION文 4.6.2
- CREATE GLOBAL TEMPORARY TABLE文 2.5.3
- CREATE INDEX文 2.5.3, 3.1.3, 3.1.4, 3.1.8, 3.2.4, 3.4.1
- CREATE MATERIALIZED VIEW文 4.4
- CREATE SEQUENCE文 4.5.1
- CREATE SYNONYM文 4.7
- CREATE TABLE文 2.2, 2.2.3, 2.2.6
    - 記憶域パラメータ 12.4.1.1
- CREATE UNIQUE INDEX文 5.2.3
- CREATE USER文
    - 一時セグメント 12.4.2.1
- カーソル
    - 埋込みSQL 7.2.6
    - 明示的 8.2.4
    - 行のフェッチ 7.2.6

D

- データ・アクセス言語 1.3
- データベース・アプリケーション 1.1.1
- データベース認証 7.2.4, 15.2.2
- データベース・バックアップ 18.3.1.3.1
- データベース・バッファ・キャッシュ 2.2.7, 14.4.1, 15.4.1.5
    - キャッシュ・ヒットとキャッシュ・ミス 14.4.1.4.3

        - コメントのキャッシュ 6.1.3.2
        - 全データベース・キャッシュ・モードの強制 14.4.1.6.5
- データベース・バッファ
        - トランザクションをコミットした後 10.2.5
        - フラッシュ・キャッシュ内のバッファ本体 14.4.1.4.3
        - バッファ・キャッシュ 14.4.1
        - チェックポイント位置 15.4.1.5
        - トランザクションのコミット 15.4.1.6.1
        - 定義 14.4.1
        - 書込み 15.4.1.5
- Database Configuration Assistant(DBCA) 18.2.3
- データベース・インスタンス 1.5.1
        - 「インスタンス」、「データベース」を参照
        - 期間 13.1.4
        - 読取り/書込み 13.1.3
        - 読取り専用 13.1.3
- データベース・オブジェクト 1.2
        - メタデータ 6.3
- データベース操作 15.2.3
- データベース常駐接続プーリング 16.2.5
- データベース・リソース・マネージャ 17.1.1.4, 17.3.1.3, 18.3.3.1
- データベース
        - 管理アカウント 2.1.4
        - クローズ 13.2.2.2
                - インスタンスの終了 13.2.2.2.2
        - 定義 1.1, 1.5.1
        - 分散 17.5.1.1
                - グローバル・データベース名の変更 14.4.3.1.3
        - 階層 1.1.1
        - 履歴 1.1.4
        - インカネーション 18.3.1.4.4
        - 概要 1
        - マウント 13.2.1.3
        - マルチプロセス 15.1.2
        - マルチユーザー 15.1.2
        - 制御ファイルに格納される名前 11.3.1
        - ネットワーク 1.1.1
        - オブジェクト・リレーショナル 1.1.3
        - オープン 13.2.1.4, 13.2.1.4.1
        - リレーショナル 1.1.2, 1.1.3, 7.1
        - 停止 13.2.2
        - 起動 2.1.4, 13.1
                - 強制 13.2.2.4
        - 構造
                - 制御ファイル 11.3

        - データ・ブロック 12, 12.2, 12.2.5
        - データ・ファイル 11
        - エクステント 12.1, 12.3
        - 論理 12
        - プロセス 15
        - セグメント 12.1, 12.4
        - 表領域 11, 12.5
- データベース・サーバー・グリッド 17.3
    - 説明 17.3.1
- データベース記憶域グリッド 17.3
    - 説明 17.3.3
- データベース構造
    - 制御ファイル 11.3
    - データ・ブロック 12, 12.2, 12.2.5
    - データ・ファイル 11
    - エクステント 12.1
    - プロセス 15
    - セグメント 12.1, 12.4
    - 表領域 11, 12.5
- Database Upgrade Assistant(DBUA) 18.2.3
- データベース・ライター・プロセス(DBW) 15.4.1.5
    - 複数のDBWnプロセス 15.4.1.5
- データベース・ライター・プロセス(DBWn)
    - チェックポイント 15.4.1.5
    - 定義 15.4.1.5
    - 最低使用頻度アルゴリズム(LRU) 14.4.1.6
    - 事前書込み 15.4.1.6
- データ・ブロック 1.5.2.2, 12.1
    - メモリーにキャッシュ 14.4.1.6
    - クラスタ化行 2.3
    - ブロック内の空き領域の結合 12.2.4.2.2
    - 圧縮 12.2.3
    - 書式 12.2.2, 12.2.2.2
    - 格納されているロック 9.4.1.1.2
    - 概要 12.1.1
    - ROWIDでの表示 12.2.2.2.3
    - バッファ・キャッシュに格納 14.4.1
    - ディスクへの書込み 14.4.1.6
- データ同時実行性
    - 定義 1.4.2
- データ整合性 1.4.3
- データ変換
    - プログラム・インタフェース 16.3
- データ破損 17.2.1.4
- データ・ディクショナリ 2.1.4, 2.2.5

    - 接頭辞がALL_のビュー 6.1.1.2
    - キャッシュ 14.4.3
    - コメント 7.2.1
    - 内容 6.1.1 14.4.3.2
    - 接頭辞がDBA_のビュー 6.1.1.1
    - ディクショナリ管理表領域 12.1.2.2
    - DUAL表 6.1.1.4
    - データベースでの使用方法 6.1.3
    - ロック 9.4.2
    - 概要 6.1
    - パブリック・シノニム 6.1.3.1
    - ストアド・サブプログラム 8.2.1.2
    - 接頭辞がUSER_のビュー 6.1.1.3
- データ・ディクショナリ・キャッシュ 6.1.3.2, 14.4.3, 14.4.3.2
- データ障害，人為的エラーに対する保護 10.3, 17.2.1.5
- データ・ファイル 1.5.2.1
    - 目次 11.2.4
    - データ・ファイル 1 12.5.1.1
    - オンラインでの移動 11.2.3
    - 制御ファイルに名前がある 11.3.1
    - ROWIDでの表示 12.2.2.2.3
    - SYSTEM表領域 12.5.1.1
    - 一時 11.2.2
- データの整合性 5.1
    - 規定 5.1, 5.1.1, 6.1.3
    - SQL 7.1
- データ操作言語
    - 「DML」を参照
- データ・オブジェクト番号，拡張ROWID 12.2.2.2.3
- データ・リカバリ・アドバイザ 18.3.1.4.2
- データ・リダクション 17.1.4
    - ランダム 17.1.4
- データ・セグメント 12.4.1
- データ型
    - BOOLEAN 2.2.4, 3.2.2.3, 7.2.2.1, 8.4.4
    - 組込み 2.2.4
    - 文字 2.2.4.1, 2.2.4.1.1
    - コンポジット型 2.2.4
    - プログラム・インタフェースによる変換 16.3
    - DATE 2.2.4.3.1
    - 日時 2.2.4.3
    - 定義 2.2.4
    - 書式モデル 2.2.4.3.1
    - 表との関連 2.2.1
    - PL/SQL 2.2.4

    - NCHAR 2.2.4.1.2
    - NUMBER 2.2.4.2.1
    - 数値型2.2.4.2
    - NVARCHAR2 2.2.4.1.2
    - オブジェクト 2.9
    - 参照型 2.2.4
    - ROWID 2.2.4.4, 2.2.4.4.1
    - TIMESTAMP 2.2.4.3.2
    - UROWID 2.2.4.4
    - ユーザー定義 2.2.4, 4.3.3
- データ・ウェアハウス
    - アーキテクチャ 17.4.2
    - ビットマップ索引 3.3
    - ディメンション 4.6
    - ディメンション表 4.6.1, 4.6.2
    - マテリアライズド・ビュー 4.4
    - パーティション 4.1
    - サマリー 4.4
- DATEデータ型 2.2.4.3, 2.2.4.3.1
- 日時データ型 2.2.4.3
- DBA_表示 6.1.1.1
- DBMS_METADATAパッケージ 6.3
- DBMS_RADMパッケージ 17.1.4
- DBMS_SERVICEパッケージ 10.4.1.2
- DBMS_SPACE_ADMINパッケージ 12.4.1.1
- DBMS_SQL_MONITORパッケージ 15.2.3
- DBMS_STATSパッケージ 7.3.4
- DBMS(データベース管理システム) 1.1.1
- DBWバックグラウンド・プロセス 15.4.1.5
- DDL(データ定義言語) 6.1
    - 説明 7.2.1
    - ロック 9.4.2
- デッドロック 7.4.1.1
    - 定義 9.3.6
- 意思決定支援システム(DSS)
    - マテリアライズド・ビュー 4.4
- デフォルト値
    - 制約の影響 5.3.3.1
- 遅延挿入 9.1.1.4, 14.4.4.2
- 定義者権限 8.2.1.1
- DELETE文 7.2.2
    - データ・ブロック内の領域の解放 12.2.4.2
- 削除，連鎖的 5.2.4.3
- 正規化されていない表 4.6.2
- 依存性，スキーマ・オブジェクト 2.1.3

- 降順索引 3.2.4
- ディクショナリ・キャッシュ・ロック 9.4.3.3
- ディクショナリ管理表領域 12.1.2.2
- ディメンション 4.6
    - 属性クラスタ表 2.4, 2.4.1, 2.4.5
    - 属性 4.6.2
    - 階層 4.6.2
        - 結合キー 4.6.2
    - 正規化された表と正規化されていない表 4.6.2
    - 表 4.6.1
- ディメンション表 4.6.1
- ディレクトリ・オブジェクト 2.1.1
- ダイレクト・パス・ロード
    - Hybrid Columnar圧縮 2.2.7.2.3
- 内容を保証しない読取り 9.1.1, 9.1.3
- ディスク領域
    - 割当てに使用されるデータファイル 11.2.4
- ディスパッチャ・プロセス(Dnnn)
    - Oracle Net Servicesを介したクライアント・プロセスの接続 16.2.4, 16.2.4.1.1
    - ネットワーク・プロトコル 16.2.4.1.1
    - 起動と停止の禁止 16.2.4.2
    - レスポンス・キュー 16.2.4.1
- 分散データベース 17.5.1.1
    - クライアント/サーバー・アーキテクチャ 16.1.1.1
    - ジョブ・キュー・プロセス 15.4.2.2
    - リカバラ・プロセス(RECO) 15.4.1.9
    - サーバーをクライアントとしても使用可能 16.1.1.1
    - トランザクション 10.6
- 分散トランザクション 10.2.1, 10.6, 17.5.1.1
    - インダウト 10.6.2
    - 命名 10.2.1
    - 2フェーズ・コミット 10.6
- DML(データ操作言語)
    - 索引付けされた列 3.3
    - 非参照用索引 3.1.2
    - ロック 9.4.1
    - 概要 7.2.2
    - 参照アクション 5.2.4.3
    - トリガー 8.4.1
- 停止時間
    - 計画メンテナンス中の回避 17.2.2
    - 計画外メンテナンス中の回避 17.2.1.2
- ドライバ 16.3.2
- DUAL表 6.1.1.4
- 動的パーティション化 15.4.3.2.3

- 動的パフォーマンス・ビュー 6.2, 6.2.1
  - データベース・オブジェクト・メタデータ 6.3
  - 記憶域 6.2.2
- 動的SQL
  - DBMS_SQLパッケージ 8.2.4
  - 埋込み 8.2.4, 19.3.2.1

E

- 埋込みSQL 7.1, 7.2.6, 8.3.2.2.2, 19.3.2.1
- EM Express18.2.1.2
- トランザクションのエンキュー 10.2.3.2
- エンタープライズ・グリッド
  - Oracle Real Application Clusters 17.3
- Enterprise Manager 13.2.2.4, 18.2.1
  - アラート・ログ 13.6.2
  - 動的パフォーマンス・ビューの使用 6.2
  - パッケージの実行 8.2.2.3
  - ロックとラッチのモニター 9.4.3
  - 停止 13.2.2.1, 13.2.2.4
- Enterprise Manager for Zero Data Loss Recovery Applianceプラグイン。
  - 「リカバリ・アプライアンス・プラグイン」を参照
- 等価結合 3.3.2
- エラー, リカバリ可能 10.3, 17.2.1.4
- 例外, PL/SQL 8.2.4
- 排他ロック 9.3.3
  - 行ロック(TX) 9.4.1.1
  - 表ロック(TM) 9.4.1.2
- EXECUTE文 8.2.1.3
- 実行計画 4.4.3, 7.3.1, 7.3.2.3, 18.3.4.5.5
  - EXPLAIN PLAN 7.2.2
- EXPLAIN PLAN文 7.2.2, 18.3.4.5.1
- 明示的ロック 9.5
- 式, SQL 3.1.3, 7.2.2.1
- エクステント 1.5.2.2
  - データ・ブロックの集合 12.3
  - 割当て解除 12.3.2
  - 定義 12.1.1
  - ディクショナリ管理 12.1.2.2
  - 増分 12.3.1
  - ローカル管理 12.1.2.1
  - 概要 12.3
- 外部プロシージャ 8.2.6
- 外部表 2.2, 2.6, 2.6.4

    - 用途 2.6.1
- 抽出、変換、ロード(ETL)
    - 概要 17.4.3

F

- ファクト表 4.6.1
- 障害
    - データベース・バッファ 13.4.1
    - 文とプロセス 15.4.1.1
- 高速コミット 15.4.1.6.1
- 高速全索引スキャン 3.2.2.2
- 高速リカバリ領域 18.3.1.2
- 高速リフレッシュ 4.4.2, 4.4.2.2
- フィールド 2.2.3
- ファイル管理ロック 9.4.3.3
- ファイル
    - アラート・ログ 15.4.1.6.2
    - 初期化パラメータ 13.2.1.2, 13.5
    - パスワード
        - 管理者権限 13.2.1.1
    - サーバー・パラメータ 13.2.1.2, 13.5
    - トレース・ファイル 15.4.1.6.2
- ファイングレイン監査 17.1.6.1
- 固定SGA 14.4.6
- フラッシュ・キャッシュ
    - バッファ読取り 14.4.1.4.3
    - 最適化物理読取り 14.4.1.4.3
- 浮動小数点数 2.2.4.2, 2.2.4.2.2
- 全データベース・キャッシュ・モードの強制 14.4.1.6.5
- 外部キー 2.2.3, 5.2.4
    - 親キー値の変更 5.2.4.3
    - コンポジット 5.2.4
    - 制約 5.2
    - 索引の作成 3.1.3, 5.2.4.4
    - NULL 5.2.4.2
    - 親キー表の更新 5.2.4.3
    - 表の更新 9.4.1.3
- 書式モデル, データ型 2.2.4.3.1, 2.2.4.5
- 空き領域
    - 自動セグメント領域管理 12.2.4
    - 管理 12.2.4
- 全索引スキャン 3.2.2.1
- 全表スキャン 3.1.1, 3.2.2.2, 7.3.3

        - デフォルト・モード 14.4.1.6.1
        - LRUアルゴリズム 14.4.1.6
        - パラレル実行 15.4.3.2.3
    - ファンクション索引 3.4, 3.4.1, 3.4.2
    - ファンクション 7.2.2.1
        - 集計 17.4.4.1
        - 分析 17.4.4.1
        - ファンクション索引 3.4
        - ハッシュ 4.1.1.2.4
        - PL/SQL 8.2.1
        - SQL 2.2.4.4.2
    - ファジー読取り 9.1.3

G

    - GDSCTLユーティリティ 10.4.1.2
    - グローバル・データベース名
        - 共有プール 14.4.3.1.3
    - グローバル索引 4.1.3, 4.1.3.2
    - グローバリゼーション・サポート 19.3.3.1
    - GRANT文 4.7, 7.2.1
    - グリッド・コンピューティング
        - データベース・サーバー・グリッド 17.3
        - データベース記憶域グリッド 17.3
    - グループ・コミット 15.4.1.6.1

H

    - SQL文のハンドル 14.3.1.1
    - ハッシュ・クラスタ 2.3.2
        - クラスタ・キー 2.3.2.1
        - ハッシュ衝突 2.3.2.4
        - ハッシュ・キー値 2.3.2.1
        - 問合せ 2.3.2.2
        - 単一表 2.3.2.3
        - ストレージ 2.3.2.4
    - ハッシュ関数 2.3.2, 4.1.1.2.4
    - ハッシュ索引 14.4.7.2
    - ハッシュ・パーティション化 4.1.1.2.4
    - ヘッダー, データ・ブロック 9.4.1.1.2, 12.2.2.1
    - 状態モニター 18.3.1.4.2
    - ヒープ構成表 2.1.2, 3.6
    - 高さ, 索引 3.2.1
    - 階層

    - 結合キー 4.6.2
    - レベル 4.6.2
- 高可用性
    - アプリケーション 17.2.2.3
    - データ破損 17.2.1.4
- ヒント, オプティマイザ 7.3.1, 7.3.5, 14.4.4.2
- ハイブリッド列圧縮 2.2.7.2, 2.2.7.2.1
    - 圧縮ユニット 2.2.7.2.2
    - DML 2.2.7.2.3
    - 行レベル・ロック 2.2.7.2.2

I

- イメージ・コピー 18.3.1.3.3
- IM列格納
    - 「インメモリー列ストア」を参照
- 不変表
    - 概要 2.8
- IM領域マネージャ
    - 「インメモリー領域マネージャ」を参照
- インカネーション, データベース 18.3.1.4.4
- 永久的増分計画 18.3.1.5.1
- 増分リフレッシュ 4.4.2, 4.4.2.2
- 索引クラスタ化係数 3.2.2.6
- 索引の圧縮 3.2.5
- 索引
    - 拡張索引圧縮 3.2.5.2
    - アプリケーション・ドメイン 3.5
    - 昇順 3.2.4
    - 自動 3.1.1, 3.1.7
    - 利点 3.1.1
    - ビットマップ 3.3, 3.3.1, 3.3.3, 3.6.3.2
    - ビットマップ結合 3.3.2
    - ブランチ・ブロック 3.2.1
    - Bツリー 2.2.6.3, 3.2
    - カーディナリティ 3.3
    - コンポジット 3.1.4
    - 圧縮 3.2.5.1
    - 連結 3.1.4
    - 定義 1.2.2, 2.1.1
    - 降順 3.2.4
    - ドメイン 3.5
    - 整合性制約の規定 5.2.3
    - 拡張可能 3.5

        - 高速全スキャン 3.2.2.2
        - ファンクション 3.4, 3.4.1, 3.4.2
        - グローバル 4.1.3, 4.1.3.2
        - 索引クラスタ化係数 3.2.2.6
        - 非表示 3.1.2
        - キー 3.1.3, 5.2.3
        - リーフ・ブロック 3.2.1
        - ローカル 4.1.3
        - ローカル・パーティション化 4.1.3.1
        - 複数 3.1.4
        - 非同一キー, ローカル 4.1.3.1.1
        - 非一意 3.1.5
        - 概要 3.1
        - パーティション化 4.1.3, 4.1.3.1.2, 4.1.4
        - パーティション化
            - 索引 4.1.3.1.2
        - 同一キー, ローカル 4.1.3.1.1
        - 接頭辞 3.2.5.1
        - レンジ・スキャン 3.2.2.3
        - 逆キー 3.2.3
        - スキャン 3.2.2, 3.2.2.5, 7.3.3
        - セカンダリ 3.6.3
        - セグメント 3.1.8, 3.3.3
        - 選択性 3.2.2.3
        - ストレージ 3.1.8, 4.1.3.1.2
        - 記憶領域 4.1.4
        - タイプ 3.1.6
        - 一意 3.1.5
        - 使用禁止 3.1.2
    - 索引, 更新 3.1.7
    - 索引構成表 3.6, 3.6.3
        - 利点 3.6
        - ビットマップ索引 3.6.3.2
        - 特性 3.6.1
        - 行オーバーフロー 3.6.2
        - 2次索引 3.6.3
    - 索引の一意スキャン 3.2.2.4
    - 処理中のトランザクション 10.3
    - 情報システム 1.1
    - INIT.ORA
        - 参照: 初期化パラメータ・ファイル
    - 初期化パラメータ・ファイル 13.2.1.2 13.5
        - 起動 13.2.1.2
    - 初期化パラメータ
        - 概要 13.5.1.2

    - 基本 13.5.1
    - OPEN_CURSORS 14.3.1.1
    - SERVICE_NAMES 16.2.2.2
- インメモリー列ストア 1.5.3.2, 2.2.7.2
    - メモリー管理 18.3.2.1
    - SQLの実行 7.4.1.4
- 内部結合 7.2.2.2
- インプレース・リフレッシュ方法 4.4.2.3
- INSERT文 7.2.2
- インスタンスPGA
    - メモリー管理 18.3.2.3
- インスタンス, データベース 1.5.1, 7.2.5, 13.1
    - データベースに対応付け 13.2.1.3
    - 期間 13.1.4
    - 障害 11.4.1
    - 障害 9.4.1.1
    - メモリー構造 14.1
    - プロセスの構造 15.1, 15.1.2
    - リカバリ
        - SMONプロセス 15.4.1.4
    - サービス名 16.2.2
    - 停止 13.2.2, 13.2.2.4
    - 終了 13.2.2.2.2
- INSTEAD OFトリガー 8.4.2
- 整合性制約 5.1, 5.1.1
    - 利点 5.1, 5.1.2
    - チェック 5.2.5
    - 定義 2.2.5
    - 遅延不可 5.3.2.1
    - 外部キーおよび親キーの更新 5.3.3.2
    - 検証 5.3.2
    - ビュー 4.3.1.1
- 内部ロック 9.4.3.3
- 内部表 2.6.3
- 時間隔パーティション表 4.1.1.2.2
- 非表示列 2.2.1.2
- 非参照用索引 3.1.2
- 実行者権限 8.2.1.1
- 分離レベル
    - シリアライズ 9.2.2
- 分離レベル, トランザクション 9.1.3
    - コミット読取り 9.2.1
    - 読取り専用 9.2.3
    - 設定 9.5

J

- Java
  - 概要 8.3
  - SQLJトランスレータ 8.3.2.2.2
  - ストアド・プロシージャ 1.3.2, 8.3.2.1
  - 仮想マシン 8.3.1
- JDBC 19.3.2.2.2
  - SQLへのアクセス 8.3.2.2
  - ドライバ 8.3.2.2.1
  - ドライバ・タイプ 8.3.2.2.1
  - 埋込みSQL 7.2.6, 19.3.2.1
- ジョブ・キュー・プロセス 15.4.2.2
- ジョブ 15.1
- 結合属性クラスタリング 2.4.2
- 結合
  - ビュー 4.3.2
- 結合, 表 3.3.2, 6.1.1, 7.2.2.2
  - デカルト 7.2.2.2
  - クラスタ化表 2.3
  - 条件 3.3.2
  - 内部結合 7.2.2.2
  - 結合条件 7.2.2.2
  - 外部結合 7.2.2.2
  - ビュー 4.3
- JSON
  - XMLとの比較 19.3.4.2.2
  - Oracle Databaseでのサポート 19.3.4.2.3
  - 概要 19.3.4.2.1

K

- キー圧縮
  - 「接頭辞圧縮」を参照
- キー
  - 索引の連結 3.2.5.1
  - 外部 5.2.4, 5.2.4.4
  - 索引 3.1.3, 3.2.5.1, 5.2.3
  - 自然 5.2.3
  - 親 5.2.4, 5.2.4.4
  - パーティション 4.1.1.1
  - 同一キー索引 3.2.1
  - 参照 5.2.4, 5.2.4.4
  - 逆 3.2.3
  - サロゲート 5.2.3

- 一意 5.2.2
- 値 5.2

L

- ラージ・プール 14.4.4
  - 遅延挿入 14.4.4.2
  - メモリー管理 14.4.4.1
- ラッチ
  - 定義 9.4.3.1
- リーフ・ブロック, 索引 3.2.1
- 最低使用頻度(LRU)アルゴリズム
  - データベース・バッファ 14.4.1.4.1
  - 全表スキャン 14.4.1.6
  - ラッチ 14.4.1.6
  - 共有SQLプール 14.4.3.1.3, 14.4.4.1
- LGWRバックグラウンド・プロセス 15.4.1.6
- ライブラリ・キャッシュ 14.4.3, 14.4.3.1
- リスナー制御ユーティリティ 18.2.4
- リスナー・プロセス 1.5.4.2, 16.2.2
  - リスナー登録プロセス(LREG) 15.4.1.3
  - サービス名 16.2.2
- リスナー登録プロセス(LREG) 15.4.1.3
- リスナー 1.5.4.2, 16.2.2
  - リスナー登録プロセス(LREG) 15.4.1.3
  - サービス名 16.2.2
- リスト・パーティション化 4.1.1.2.3
- ローカル索引 4.1.3
- ローカル管理表領域 12.1.2.1
- ローカル・パーティション索引 4.1.3.1
- ローカル一時表領域 12.5.2.1
- ロック 9.1.2
  - トランザクションをコミットした後 10.2.5
  - 自動 9.3.2, 9.4
  - ビットマップ索引 3.3
  - 変換 9.3.3, 9.3.4
  - デッドロック 7.4.1.1, 9.3.6
  - 定義 1.4.2
  - ディクショナリ 9.4.2
  - ディクショナリ・キャッシュ 9.4.3.3
  - DML 9.4.1
  - 期間 9.3.2, 9.3.5
  - エスカレーション 9.3.3
  - 排他 9.3.3

        - 排他DDL 9.4.2.1
        - 排他表 9.4.1.2
        - ファイル管理ロック 9.4.3.3
        - ハイブリッド列圧縮 2.2.7.2.2
        - ラッチ 9.4.3.1
        - ログ管理ロック 9.4.3.3
        - 手動 9.5
        - 概要 9.1.2
        - 解析 9.4.2.3
        - 制限 9.3.3
        - ロールバック・セグメント 9.4.3.3
        - 行(TX) 9.4.1.1
        - 行排他表 9.4.1.2
        - 行共有 9.4.1.2
        - 共有DDL 9.4.2.2
        - 共有ロック 9.3.3
        - 共有行排他 9.4.1.2
        - 共有表 9.4.1.2
        - システム 9.4.3
        - 表 3.1.1, 7.2.2
        - 表(TM) 9.4.1.2
        - 表領域 9.4.3.3
        - タイプ 9.4
        - 索引なしの外部キー 9.4.1.3
        - ユーザー定義 9.6
    - LOCK TABLE文 7.2.2
    - ログベースのリフレッシュ 4.4.2.2
    - 論理データベース構造
        - データ・ブロック 1.5.2.2
        - 定義 1.5.2.2
        - エクステント 1.5.2.2
        - セグメント 1.5.2.2
        - 表領域 1.5.2.2, 12.5
    - 論理ROWID 3.6.3
    - 論理トランザクションID 10.3.2.2
    - ログ管理ロック 9.4.3.3
    - ログ・スイッチ
        - アーカイバ・プロセス 15.4.2.1
        - ログ順序番号 11.4.2.1
    - ログ・ライター・プロセス(LGWR) 15.4.1.6
        - グループ・コミット 15.4.1.6.1
        - 使用可能なオンラインREDOログ 11.4.2.1
        - REDOログ・バッファ 14.4.2
        - 事前書込み 15.4.1.6
        - オンラインREDOログ・ファイルへの書込み 11.4.2.1

- LONGデータ型
  - 記憶域 2.2.6.1
- 更新内容の消失 1.4.3, 9.2.1.2
- 書込み欠落
  - シャドウ表領域 12.5.1.4.1
- LRU 14.4.1.4.1, 14.4.1.6
  - 共有SQLプール 14.4.3.1.3

M

- メンテナンス・タスク, 自動 18.3.4.2
- メンテナンス・ウィンドウ 18.3.4.2
- 手動ロック 9.5
- マッピング表 3.6.3.2
- マスター表 4.4
- マテリアライズド・ビュー 4.4
  - 特性 4.4.1
  - 完全リフレッシュ 4.4.2.1
  - 高速リフレッシュ 4.4.2.2
  - 増分リフレッシュ 4.4.2.2
  - インプレース・リフレッシュおよびホーム外リフレッシュ 4.4.2.3
  - リフレッシュ
    - ジョブ・キュー・プロセス 15.4.2.2
  - リフレッシュ 4.4.2, 4.4.2.3
  - ゾーン・マップ 2.4.3.1
- メディア・リカバリ
  - 完全 18.3.1.4.4
  - 概要 18.3.1.4.4
- memoptimizeプール 14.4.7.2, 18.3.2.1
- メモリー
  - SQL文のための割当て 14.4.3.1.1
  - 内容 14.1
  - プロセスでの使用 15.1
  - ストアド・プロシージャ 8.2.1.1
- メモリー管理
  - 概要 18.3.2
  - 自動 18.3.2.1
  - 自動共有 18.3.2.2
  - インスタンスPGA 18.3.2.3
- MERGE文 7.2.2
- メッセージ・ゲートウェイ 17.5.2.2.2
- メトリック 6.2, 18.3.4.1
- ユーザー・アクションの監視 17.1.6.1
- マルチブロック書込み 15.4.1.5

- 多重化
  - 制御ファイル 11.3.2
  - REDOログ・ファイル・グループ 11.4.2.2
  - REDOログ・ファイル 11.4.2.2
- マルチテナント・アーキテクチャ
  - 定義 1.5.1.1
- マルチテナント・コンテナ・データベース
  - 「CDB」を参照
- マルチバージョン読取り一貫性 6.2.2, 9.1, 9.1.1, 9.1.3, 12.4.3, 12.4.3.2
  - 遅延挿入 9.1.1.4
  - 内容を保証しない読取り 9.1.1
  - コミット読取り分離レベル 9.2.1.1
  - 文レベル 1.4.3, 9.1.1.1
  - トランザクション・レベル 9.1.1.2
  - UNDOセグメント 9.1.1.3
- mutex 9.4.3.2

N

- NaN(非数) 2.2.4.2
- 各国語サポート(NLS) 19.3.3.1
- 自然キー 5.2.3
- NCHARデータ型 2.2.4.1.2
- ネットワーク・リスナー・プロセス
  - 接続要求 16.2.4.1.1
- ネットワーク
  - クライアント/サーバー・アーキテクチャでの使用 16.1.1
  - 通信プロトコル 16.3.2, 16.3.3
  - ディスパッチャ・プロセス 16.2.4.1.1
  - ドライバ 16.3.2
  - リスナー・プロセス 16.2.2
  - Oracle Net Services 16.2
- NLS_DATE_FORMATパラメータ 2.2.4.3.1
- NOAUDIT文 7.2.1
- 非CDB 1.5.1.1
- 非一意索引 3.1.5
- 正規化された表 4.6.2
- NOT NULL制約 5.2 5.2.1
- NULL 2.2.3
  - 外部キー 5.2.4.2
  - 格納方法 2.2.3, 2.2.6.4
  - 索引付けされた 3.1.5
  - 禁止 5.2.1
- NUMBERデータ型 2.2.4.2.1

- 数，浮動小数点 2.2.4.2
- 数値データ型 2.2.4.2
    - 浮動小数点数 2.2.4.2.2
- NVARCHAR2データ型 2.2.4.1.2

O

- オブジェクト表 2.2, 2.9
- オブジェクト型 2.9, 4.3.3
- オブジェクト・ビュー 4.3.3
- ODBC 19.3.2.2.2
- OLAP
    - 索引構成表 3.6
    - 概要 17.4.4.3
- OLTP
    - 表圧縮 2.2.7.1
- オンライン分析処理
    - 「OLAP」を参照
- オンラインREDOログ 11.4.1, 12.4.3.2
    - アーカイバ・プロセス(ARCn) 15.4.2.1
    - バッファ管理 15.4.1.6
    - チェックポイント 11.3.1
    - コミット済データ 13.4.1
    - トランザクションのコミット 15.4.1.6.1
    - ログ・スイッチ
        - アーカイバ・プロセス 15.4.2.1
    - ログ・ライター・プロセス 14.4.2, 15.4.1.6
    - ロールフォワード 13.4.1
    - UNDOデータ 12.4.3.3
    - バッファの書込み 15.4.1.6
- オンラインREDOログ・ファイル 1.5.2.1
- OPEN_CURSORSパラメータ
    - プライベートSQL領域の管理 14.3.1.1
- オペレーティング・システム
    - 通信ソフトウェア 16.3.3
    - 管理者の権限 13.2.1.1
- 最適化物理読取り 14.4.1.4.3
- オプティマイザ 7.1.1, 7.3
    - 適応最適化 7.3.2.3
    - コンポーネント 7.3.2
    - エスティメータ 7.3.2.2
    - 実行 7.4.1.4
    - 実行計画 4.4.3, 7.3.1, 7.3.2.3, 7.4.1.3, 18.3.4.5.5
    - ファンクション索引 3.4.2

    - ヒント 7.3.1, 7.3.5, 14.4.4.2
    - 非参照用索引 3.1.2
    - 問合せ計画のパーティション 4.1
    - プラン・ジェネレータ 7.3.2.3
    - 問合せ計画 7.4.1.3
    - 問合せトランスフォーマ 4.4.3, 7.3.2.1
    - 行ソース 7.4.1.3
    - 統計 2.2.6.1, 7.3.4, 7.4.1.2, 18.3.4.2
- オプティマイザ統計アドバイザ
    - 概要 18.3.4.5.2
- Oracle Advanced Analytics 17.4.4.4
- Oracle ASM (自動ストレージ管理) 11.1.2, 11.1.2.1, 17.3.3, 18.3.1.2
- Oracle Audit Vault and Database Firewall。 17.1.6.4
- Oracle Automatic Storage Management。
    - 「Oracle ASM」を参照
- Oracleブロック 12.1.1
- Oracle Call Interface
    - 「OCI」を参照
- Oracleコード 16.3
- Oracle Connection Manager制御ユーティリティ 18.2.4
- Oracle Database
    - 履歴 1.1.4
- Oracle Database Vault 17.1.5.1, 17.1.6.2
- Oracle Data Mining 17.4.4.4.1
- Oracle Data Pump 12.5.1, 17.1.5.1, 18.2.5.2
    - ダンプ・ファイル・セット 18.2.5.2
    - 統合監査証跡 17.1.6.2
- Oracle Enterprise Manager
    - 「Enterprise Manager」を参照
- Oracle Enterprise Manager Cloud Control (Cloud Control)
    - 「Cloud Control」を参照
- Oracle Flashback Technology 1.4.3, 18.3.1.4.1
- Oracle Flex Clusters 17.3.2
- Oracle GoldenGate 17.5.2, 17.5.2.1, 17.5.2.2.1
- Oracle Internet Directory 16.2.2.2
- Oracle JDeveloper 19.2.3
- Oracle JVM
    - 主なコンポーネント 8.3.1.2
    - 概要 8.3.1.1
- Oracle Label Security 17.1.5.3, 17.1.6.2
- Oracle LogMiner 18.2.5.3
- Oracle管理エージェント 18.2.1.1
- Oracle管理リポジトリ 18.2.1.1
- Oracle管理サービス(OMS) 18.2.1.1
- Oracle Net 1.5.4.2

- Oracle Netコンフィギュレーション・アシスタント 18.2.4
- Oracle Net Listener 1.5.4.2
- Oracle Net Manager 18.2.4
- Oracle Net Services 1.5.4.2, 16.2
  - クライアント/サーバー・システムでの使用 16.2
  - 概要 16.2
  - 共有サーバーの要件 16.2.4.1.1
- Oracle Net Services Connection Manager 8.3
- Oracleプロセス 15.1.2, 15.3
- Oracleプログラム・インタフェース(OPI) 16.3.1
- Oracle RAC
  - 「Oracle Real Application Clusters」を参照
- Real Application Clusters
  - エンタープライズ・グリッド 17.3
  - Oracle Flex Clusters 17.3.2
  - 逆キー索引 3.2.3
- Oracle R Enterprise 17.4.4.4.2
- Oracle Secure Backup 18.3.1.1
- Oracle Sharding
  - 概要 4.2
  - アーキテクチャ 1.5.1.2
- Oracle Spatial and Graph 19.3.4.5
- Oracle Text 19.3.4.4
- Oracle Universal Installer(OUI) 18.2.3
- Oracle XA
  - ラージ・プール内のセッション・メモリー 14.4.4
- ORDBMS(オブジェクト・リレーショナル・データベース管理システム) 1.1.3
- 外部結合 7.2.2.2
- ホーム外リフレッシュ方法 4.4.2.3

P

- パッケージ 8.2.2
  - 利点 8.2.2.1
  - 作成 8.2.2.2
  - 実行 8.2.6
  - ロック用 9.6
  - プライベート 8.2.2.1
  - パブリック 8.2.2.1
  - 共有SQL領域 14.4.3.1.2
  - サブプログラムの実行 8.2.2.3
- ページ 12.1.1
- パラレル実行 15.4.3.2
  - コーディネータ 15.4.3.2.3

        - サーバー 15.4.3.2.3
        - サーバー 15.4.3.2.3
        - 調整 15.4.3.2
- パラレルSQL 15.4.3.2
        - コーディネータ・プロセス 15.4.3.2.3
        - サーバー・プロセス 15.4.3.2.3
- パラメータ
        - 初期化 13.5
            - ロック動作 9.4
- 解析ロック 9.4.2.3
- 解析, SQL 7.4.1, 7.4.1.1
        - 埋込みSQL 7.2.6
        - ハード解析 9.4.3.1
        - 情報の記憶域 6.1.3.2
- 部分グローバル・パーティション索引 4.1.4
- パーティション・チェンジ・トラッキング・リフレッシュ 4.4.2.2
- パーティション索引 4.1.3.1
        - グローバル 4.1.3.2
        - ストレージ 4.1.3.1.2
- パーティション表 4.1.1.2.4
        - 間隔 4.1.1.2.2
        - 参照 4.1.1.2.5
- パーティション化
        - ハッシュ 4.1.1.2.4
        - 時間隔 4.1.1.2.2
        - リスト 4.1.1.2.3
        - レンジ 4.1.1.2.1
        - 参照 4.1.1.2.5
        - コンポジット 4.1.1.2.6
        - 索引 4.1.3.2
        - 索引 4.1.3.1
        - キー 4.1.1.1
- パーティション
        - 特性 4.1.1
        - コンポジット・パーティション化 4.1.1.2
        - 定義 2.1.1
        - 動的パーティション化 15.4.3.2.3
        - 問合せからの除外 4.1.3.1.1
        - 索引 4.1.3
        - キー 4.1.1.1
        - マテリアライズド・ビュー 4.4.1
        - 概要 4.1
        - パーティション
            - コンポジット 4.1.1.2
        - レンジ 4.1.1.2

    - セグメント 12.4.1
    - 単一レベル 4.1.1.2
    - 戦略 4.1.1.2
    - 表 4.1.1.2.1, 4.1.1.2.3, 4.1.2, 4.1.4
- パスワード
    - 管理者権限 13.2.1.1
    - 接続 15.2.2
- PCTFREE記憶域パラメータ
    - 仕組み 12.2.4.1
- PDB 1.5.1.1
    - フラッシュバック 18.3.1.4.1
- パフォーマンス
    - グループ・コミット 15.4.1.6.1
    - パッケージ 8.2.2.1
- PGA
    - 「プログラム・グローバル領域(PGA)」を参照
- PGA_AGGREGATE_LIMIT初期化パラメータ 18.3.2.3
- PGA_AGGREGATE_TARGET初期化パラメータ 18.3.2.3
- 仮読取り 9.1.3
- 物理データベース構造
    - 制御ファイル 1.5.2.1, 11.3
    - データ・ファイル 1.5.2.1
    - オンラインREDOログ・ファイル 1.5.2.1
- 物理推測 3.6.3.1
- PL/SQL
    - 無名ブロック 8.2, 8.2.3
    - コレクション 8.2.5, 8.2.5.1
    - データ・タイプ 2.2.4
    - 動的SQL 8.2.4
    - 例外 8.2.4
    - 実行 8.2.6
    - サブプログラムの実行 8.2.1.3
    - 言語構造 8.2.4
    - 概要 8.2
    - パッケージの作成 8.2.2.2
    - パッケージ 6.3, 8.2.2, 8.2.2.1
    - PL/SQLエンジン 8.2.6
    - プログラム・ユニット 8.2.1.1, 14.4.3.1.2
        - 共有SQL領域 14.4.3.1.2
    - レコード 8.2.5, 8.2.5.1, 8.2.5.2
    - ストアド・プロシージャ 1.3.2, 6.1.1.2, 8.2, 8.2.1, 8.2.1.1, 8.2.2.1
    - 単位 8.2
        - コンパイル済 8.2.6
- 計画
    - SQL実行 7.2.2

- 計画停止時間
  - 停止時間の回避 17.2.2
- プラガブル・データベース
  - 「PDB」を参照
- PMANバックグラウンド・プロセス 15.4.1.2
- PMONバックグラウンド・プロセス 15.4.1.1.1
- プラグマ, PL/SQL 10.5
- プリコンパイラ 8.1
  - COBOL 19.3.2.1.1
  - 埋込みSQL 7.2.6
  - FORTRAN 19.3.2.1.1
  - Pro*C/C++ 19.3.2.1.1
- 述語, SQL 3.2.2.1, 7.1.1
- 主キー制約 5.2
- 主キー 2.2.3 3.1.1
  - 制約 5.2.3
  - ハッシュ・クラスタ 2.3.2.3
  - 索引構成表 2.2
- プライベートSQL領域 14.4.3.1.1
  - 説明 14.4.3.1.1
  - どのように管理されるか 14.3.1.1
  - 解析 7.4.1.1
- プライベート・シノニム 4.7
- 権限 6.1, 7.2.4, 17.1.1.1
  - 管理者 13.2.1.1
  - 付与 7.2.1, 17.1.1.3
  - PL/SQLプロシージャ 8.2.1.1
  - 権限プロファイラ 17.1.1.3
  - 取消し 7.2.1
- プロシージャ
  - 利点 8.2.1.1
  - 実行 8.2.1.3, 8.2.6
  - PL/SQLパッケージでの実行 8.2.2.3
  - 外部 8.2.6
  - メモリー割当て 8.2.1.1
  - セキュリティ 8.2.1.1
  - 共有SQL領域 14.4.3.1.2
  - ストアド・プロシージャ 1.3.2, 6.1.1.2, 8.2, 8.2.6
- プロセス 15.1
  - アーカイバ(ARCn) 15.4.2.1
  - バックグラウンド 1.5.3.1, 15.4
  - チェックポイント 15.4.1.5
  - クライアント 1.5.3.1, 15.2
  - 専用サーバー 16.2.4.1.2
  - 定義 1.5.3.1

        - 分散トランザクションの解決 15.4.1.9
        - ジョブ・キュー 15.4.2.2
        - リスナー 16.2.2
            - 共有サーバー 16.2.4.1.1
        - ログ・ライター(LGWR) 15.4.1.6
        - マルチスレッド 15.1.2
        - Oracle 15.3
        - パラレル実行コーディネータ 15.4.3.2.3
        - パラレル実行サーバー 15.4.3.2.3
        - プロセス・モニター(PMON) 15.4.1.1.1
        - リカバラ(RECO) 15.4.1.9
        - サーバー 1.5.3.1, 15.3
            - 共有 16.2.4.1.1, 16.2.4.1.2
        - 共有サーバー 16.2.4
            - クライアントのリクエスト 16.2.4.1
        - 構造 15.1
        - システム・モニター(SMON) 15.4.1.4
        - ユーザー
            - 障害からのリカバリ 15.4.1.1
            - サーバー・プロセスの共有 16.2.4.1.1
- 処理
    - パラレルSQL 15.4.3.2
- プロセス・マネージャ・プロセス(PMON)
    - 状態オブジェクト 15.4.1.1.4
- プロセス・モニター・プロセス(PMON)
    - 説明 15.4.1.1.1
- プログラム・グローバル領域(PGA) 1.5.3.2 14.1.1
    - 共有サーバー 16.2.4.1.2
    - 共有サーバー 16.2.4.1.2
- プログラム・インタフェース 16.3
    - Oracle側(OPI) 16.3.1
    - 構造 16.3.1
    - ユーザー側(UPI) 16.3.1
- プログラミング, サーバー側 1.3.2, 8.1
- プログラム・ユニット
    - 共有プール 14.4.3.1.2
- 保護ポリシー
    - 利点 18.3.1.5.1
- 疑似コード
    - トリガー 8.4.6
- 疑似列 2.2.4.4.2, 3.6.1
- パブリック・シノニム 4.7

Q

- 問合せ
    - ブロック 7.2.2.3
    - 定義 7.2.2.1
    - 暗黙的 9.2.1.1
    - DML 7.2.2
    - パラレル処理 15.4.3.2
    - 問合せブロック 7.3.2.3
    - 問合せトランスフォーマ 7.3.2.1
    - SQL言語 7.1
    - ストアド 4.3
    - 副問合せ 4.3, 7.2.2, 7.2.2.3
- 問合せブロック 7.3.2.3
- 問合せオプティマイザ
    - 「オプティマイザ」を参照
- 問合せ計画 7.4.1.3
    - パーティション化 4.1
- クエリー・リライト 4.6
- 問合せトランスフォーマ 4.4.3

R

- レンジ・パーティション化 4.1.1.2.1
- レンジ・パーティション 4.1.1.2
- レンジ・スキャン, 索引 3.2.2.3
- RDBMS(リレーショナル・データベース管理システム) 1.1.3
- RDMA 14.4.7.2
- コミット読取り分離 9.2.1
- 読取り一貫性
    - 「マルチバージョン読取り一貫性」を参照
- 読取り専用分離レベル 9.2.3
- 非コミット読取り 9.1.3
- Real Application Clusters
    - キャッシュ・フュージョン 9.1.1.3
    - システム・モニター・プロセス 15.4.1.4
    - オンラインREDOログのスレッド 11.4.2
- レコード, PL/SQL 8.2.5, 8.2.5.1, 8.2.5.2
- リカバリ可能なエラー 10.3, 17.2.1.4
- リカバラ・プロセス(RECO) 15.4.1.9
    - インダウト・トランザクション 10.6.2
- リカバリ
    - 完全 18.3.1.4.4
    - データベース・バッファ 13.4.1
    - 分散処理 15.4.1.9
    - インスタンス・リカバリ

        - SMONプロセス 15.4.1.4
    - メディア 18.3.1.4.4
        - ディスパッチャ・プロセス 16.2.4.2
    - プロセス・リカバリ 15.4.1.1
    - インスタンスの終了後に必要 13.2.2.2.2
    - トランザクションのロールバック 13.4.4
    - 表領域
        - ポイント・イン・タイム 18.3.1.4.4
- リカバリ・アプライアンス 18.3.1.5.1
    - バックアップ・モジュール 18.3.1.5.2
    - メタデータ・データベース 18.3.1.5.2
    - プラグイン 18.3.1.5.1
- リカバリ・カタログ 18.3.1.5.2
- Recovery Manager 6.2, 18.3.1.1
    - アーキテクチャ 18.3.1.2
    - 監査 17.1.6.2
    - バックアップ 18.3.1.3.3
    - リカバリ・アプライアンス 18.3.1.5.1
- リダクション, データ 17.1.4
    - ポリシー 17.1.4
- REDOログ・バッファ 11.4.1
- REDOログ・ファイル
    - 使用可能 11.4.2.1
    - 循環使用 11.4.2.1
    - 内容 11.4.3
    - 分散トランザクション情報 11.4.2
    - グループ・メンバー 11.4.2.2
    - グループ, 定義 11.4.2.2
    - インスタンス・リカバリでの使用 11.4
    - LGWR 11.4.2.1
    - メンバー 11.4.2.2
    - 多重化 11.4.2.2
    - オンライン, 定義 11.4
    - REDOエントリ 11.4.3
    - スレッド 11.4.2
- REDOログ・バッファ 14.4.2
- REDOレコード 11.4.3
- REF制約 5.2
- 参照パーティション表 4.1.1.2.5
- 参照整合性
    - 例 5.3.3
    - 自己参照型制約 5.3.3
- リフレッシュ
    - 増分 4.4.2
    - ジョブ・キュー・プロセス 15.4.2.2

    - マテリアライズド・ビュー 4.4.2, 4.4.2.3
- リレーショナル・データベース管理システム
    - 「RDBMS」を参照
- リレーション, 単純 1.1.2
- 予約語 7.2
- リソース管理 18.3.3.1
- レスポンス・キュー 16.2.4.1
- RESULT_CACHE句 14.4.3.3.1
- 結果キャッシュ 14.4.3.3, 14.4.3.3.1
- 結果セット, SQL 2.2, 2.2.4.4.2, 2.5, 4.3.2, 7.2.2.1
- 逆キー索引 3.2.3
- REVOKE文 7.2.1
- 権限, 定義者および実行者 8.2.1.1
- ロール 2.1.1, 6.1.1.2, 7.2.4
- ロールバック 10.2.4
    - 説明 10.2.4
    - トランザクションの終了 10.2.4
    - DDLでの暗黙的 7.2.1
    - 文レベル 9.3.6, 10.1.3
    - セーブポイントまで 10.2.3.1
    - トランザクション・レベル 7.2.3
- ロールバック・セグメント
    - ロック 9.4.3.3
    - パラレル・リカバリ 13.4.4
    - リカバリ中の使用 13.4.4
- ROLLBACK文 10.2
- ロールバック 10.1, 10.2.4
- 行連鎖 12.2.4.3
- 行データ(データ・ブロックのセクション) 12.2.2.2
- 行ディレクトリ 12.2.2.1
- ROWIDデータ型 2.2.4.4, 2.2.4.4.1
- ROWID 2.2.6.3
    - 外部 2.2.4.4
    - 索引 3.1.5
    - 論理 2.2.4.4, 3.6.3
    - 物理 2.2.4.4
    - 行の移行 12.2.4.3
    - スキャン 7.3.3
    - ユニバーサル 2.2.4.4
- 行ロック 9.4.1.1
    - 並行性 9.4.1.1.1
    - ストレージ 9.4.1.1.2
- 行断片 2.2.6.2
- 行
    - アドレス 2.2.6.3

    - ブロックにまたがる連鎖 2.2.6.2, 12.2.4.3
    - クラスタ化 2.2.6.2
    - 定義 1.2.1, 2.2.2
    - データ・ブロック内での形式 12.2.2.1
    - ロック 9.4.1.1
    - ロック 9.4.1.1
    - 新規ブロックへの移行 12.2.4.3
    - 行セット 7.4.1.3
    - 行ソース 7.4.1.3
    - ROWIDでの表示 12.2.2.2.3
    - ストレージ 2.2.6.2
    - トリガー 8.4.2
- 行ソース生成 7.4.1.3

S

- サンプル・スキーマ 2.1.5
- セーブポイント 7.2.3 10.2.3
    - 定義 10.2.3
    - 暗黙的 10.1.3
    - ロールバック 10.2.3.1
    - SAVEPOINT文 10.2
- SAVEPOINT文 7.2.3
- SBTライブラリ 18.3.1.5.2
- スキャン
    - クラスタ 7.3.3
    - 高速全索引 3.2.2.2
    - 全索引 3.2.2.1
    - 全表 3.1.1, 7.3.3, 14.4.1.6, 14.4.1.6.1
    - 索引 3.2.2, 7.3.3
    - 索引スキップ 3.2.2.5
    - 範囲 2.3.2.2
    - rowid 7.3.3
    - 一意索引 3.2.2.4
- スキーマ・オブジェクト
    - 定義 1.2, 6.1, 7.2.1
    - 依存性 2.1.3, 4.3.1
    - ディメンション 4.6
    - 索引 3.1
    - 概要 2.1
    - マテリアライズド・ビュー 4.4
    - データファイルとの関連 11.2.1
    - 順序 4.5
    - ストレージ 2.1.2

- スキーマ 1.2, 2.1
  - サンプル 2.1.5
- SCN
  - 「システム変更番号」を参照
- 2次索引 3.6.3
  - 利点 3.6.3
  - 物理推測 3.6.3.1
- SecureFiles 19.3.4.3.3
- セキュリティ
  - 管理者権限 13.2.1.1, 17.1.1.1
  - 監査 17.1.6.1
  - 定義者権限 8.2.1.1
  - プログラム・インタフェースでの規定 16.3
  - ビュー 4.3
- セグメント・アドバイザ 18.3.4.2
- セグメント 1.5.2.2, 12.4
  - データ 12.4.1
  - 定義 12.1.1
  - 索引 3.1.8, 3.3.3
  - 概要 12.4
  - 表の記憶域 2.2.6
  - 一時 2.5.2, 12.4.2
    - 割当て 12.4.2
  - ユーザー 12.4.1.1
- 選択性 3.2.2.3
- 選択リスト, SQL 7.2.2.1
- SELECT文 7.2.2
- 自己参照型整合性制約 5.2.4.1
- 順序
  - 特性 4.5.1
  - 同時アクセス 4.5.2
  - 定義 2.1.1, 4.5
  - サロゲート・キー 5.2.3
- シリアライズ可能性, トランザクション 9.1
- シリアライズ可能分離レベル 9.2.2
- サーバー・パラメータ・ファイル
  - 概要 13.5.2
  - 起動 13.2.1.2
- サーバー・プロセス 1.5.3.1, 15.3
  - 作成 15.3.3
  - リスナー・プロセス 16.2.2
- サーバー
  - クライアント/サーバー・アーキテクチャ 16.1.1
  - 共有
    - アーキテクチャ, 15.1.1, 16.2.4

        - プロセス 16.2.4, 16.2.4.1.2
- サーバー側プログラミング 8.1
    - 概要 8, 8.1
- SERVICE_NAMESパラメータ 16.2.2.2
- サービス名 16.2.2
- サービス指向アーキテクチャ 1.5.4.1, 16.1.2.4
- サービス階層
    - 「リカバリ・アプライアンス・サービス階層」を参照
- セッション制御文 7.2.4
- セッション 7.2.4
    - 接続と対比 15.2.2
    - 定義 15.2.2
    - ラージ・プール内のメモリー割当て 14.4.4
    - 順序生成 4.5.2
- SET CONSTRAINT文 7.2.3
- SET TRANSACTION文 7.2.3, 10.1.2.1
- SGA(システム・グローバル領域) 1.5.3.2, 14.1.1
    - 割当て 13.2.1.2
    - ビッグ・テーブル・キャッシュ 14.4.1.5
    - 内容 14.4
    - データベース・バッファ・キャッシュ 14.4.1
    - データ・ディクショナリ・キャッシュ 6.1.3.2, 14.4.3.2
    - 固定SGA 14.4.6
    - ラージ・プール 14.4.4
    - REDOログ・バッファ 10.2.2, 14.4.2
    - ロールバック・セグメント 10.2.2
    - 共有および書込み可能 14.4
    - 共有プール 8.2.1.1, 14.4.3
    - 可変パラメータ 13.5.1.1
- シャドウ書込み欠落保護 12.5.1.4
    - 構成 12.5.1.4.4
    - 有効化 12.5.1.4.2
    - 目的 12.5.1.4.1
- シャドウ表領域 12.5.1.4
    - 作成 12.5.1.4.4
    - 動作 12.5.1.4.2
    - 目的 12.5.1.4.1
    - ユーザー・インタフェース 12.5.1.4.3
- シャード・データベース(SDB) 4.2
- シャード表 4.2.1
- シャード表 4.2
- シャード 4.2
- 共有DDLロック 9.4.2.2
- 共有プール 8.2.6, 14.4.3, 14.4.3.1.3
    - 割当て 14.4.3.1.3

    - 依存性管理 14.4.3.1.3
    - 説明 14.4.3
    - フラッシュ 14.4.3.1.3
    - ラッチ 9.4.3.1
    - 解析ロック 9.4.2.3
- 共有サーバー
    - 説明 15.1.1
    - ディスパッチャ・プロセス 16.2.4.1.1
    - Oracle Net ServicesまたはSQL*Net V2の要件 16.2.4.1.1
    - プロセス 16.2.4.1.2
    - 必要なプロセス 16.2.4
    - 限定的運用 16.2.4.2
    - ラージ・プール内のセッション・メモリー 14.4.4
- 共有サーバー・プロセス(Snnn) 16.2.4.1.2
    - 説明 16.2.4.1.2
- 共有SQL領域 4.3.1.2, 14.4.3, 14.4.3.1.1, 14.4.3.1.3
    - 依存性管理 14.4.3.1.3
    - 説明 14.4.3.1.1
    - 解析ロック 9.4.2.3
    - プロシージャ, パッケージ, トリガー 14.4.3.1.2
- 共有一時表領域 12.5.2.1
- 共有ロック 9.3.3
- 停止 13.2.2, 13.2.2.4
    - 異常 13.2.2.4
    - ディスパッチャ・プロセスでは禁止 16.2.4.2
    - ステップ 13.2.2
- SHUTDOWN ABORT文 13.2.2.4
- Simple Object Access Protocol
    - 「SOAP」を参照
- 単純なトリガー 8.4.3
- 単一レベル・パーティション化 4.1.1.2
- SMONバックグラウンド・プロセス 15.4.1.4
- SOA 1.5.4.1, 16.1.2.4
- SOAP(Simple Object Access Protocol) 1.5.4.1
- ソフトウェア・コード領域 14.1.1
- 領域管理
    - エクステント 12.3
    - PCTFREE 12.2.4.1
    - 行連鎖 12.2.4.3
    - セグメント 12.4
- 領域管理コーディネータ・プロセス(SMCO)のバックグラウンド・プロセス14.4.4.2
- SQL 7.1, 7.2
    - 条件 7.1.1 7.2.2.1
    - データ定義言語(DDL) 7.2.1
    - データ操作言語(DML) 7.2.2

        - 定義 1.3.1
        - ディクショナリ・キャッシュ・ロック 9.4.3.3
        - 動的SQL 8.2.4
        - 埋込み 7.1, 7.2.6, 19.3.2.1
        - 実行可能ファイル 10.1.2.1
        - 実行 7.4.1.4, 10.1.3
        - 式 3.1.3, 7.2.2.1
        - ファンクション 2.2.4.4.2
        - 対話型 7.1
        - メモリー割当て 14.4.3.1.1
        - 演算子 7.1.1
        - 最適化 7.4.1.2
        - Oracle 1.3.1, 7.1.2
        - 概要 7.1
        - パラレル実行 15.4.3.2
        - 解析 7.4.1, 7.4.1.1
        - PL/SQL 8.2
        - 述語 3.2.2.1, 7.1.1
        - 処理 7.4.1
        - 予約語 7.2
        - 結果セット 2.2, 2.2.4.4.2, 2.5, 4.3.2, 7.2.2.1
        - 選択リスト 7.2.2.1
        - セッション制御文 7.2.4
        - 標準 7.1.2
        - 文 7.2
        - 副問合せ 4.3, 7.2.2.3
        - システム制御文 7.2.5
        - トランザクション制御文 7.2.3
        - トランザクション 10.1
        - 文のタイプ 7.2
    - SQL*Loader 18.2.5.1
    - SQL*Plus 18.2.2
        - アラート・ログ 13.6.2
        - パッケージの実行 8.2.2.3
        - ロックとラッチのモニター 9.4.3
    - SQL領域
        - プライベート 14.4.3.1.1
        - 共有 14.4.3.1.1
    - SQLJ規格 8.3.2.2.2
    - SQLチューニング・アドバイザ 18.3.4.2
    - SRVCTLユーティリティ 10.4.1.2
    - 標準監査 17.1.6.1
    - 規格
        - ANSI/ISO 7.1.2
            - 分離レベル 9.1.3

- スター・スキーマ 2.4.5
- 起動 13.1
  - ディスパッチャ・プロセスでは禁止 16.2.4.2
- 文レベルの原子性 10.1.3
- 文レベルの読取り一貫性 9.1.1.1
- 文レベルのロールバック 9.3.6, 10.1.3
- 文, SQL 7.2
- 統計 2.2.6.1, 6.2, 7.3.1, 14.4.3.1.3
  - ASH 18.3.4.4
  - AWR 18.3.4.2
  - 定義 7.3.4
  - オプティマイザ用に収集 18.3.4.2
  - Java関連 14.4.5
  - 結合順序 7.2.2.2
  - オプティマイザ 7.4.1.2
  - UNDO保存期間 12.5.1.3.2
- 記憶域
  - 論理構造 12.5
  - NULL 2.2.3
  - トリガー 8.4.6
- ストアド・プロシージャ
  - 「プロシージャ」を参照
- Structured Query Language(SQL) 7.1
- 構造
  - ロック 9.4.2
  - 論理 12
    - データ・ブロック 12, 12.2, 12.2.5
    - エクステント 12.1, 12.3
    - セグメント 12.1, 12.4
    - 表領域 11, 12.5
  - 物理
    - 制御ファイル 11.3
    - データ・ファイル 11
  - プロセス 15
- サブプログラム, PL/SQL
  - 「プロシージャ」を参照
- 副問合せ 4.3, 7.2.2, 7.2.2.3
- サマリー 4.4
- サロゲート・キー 5.2.3
- シノニム
  - 制約が間接的に影響する 5.3.3.2
  - データ・ディクショナリ・ビュー 6.1.3.1
  - 定義 2.1.1, 4.7
  - プライベート 4.7
  - パブリック 4.7, 6.1.1.3

    - 安全性 4.7
- SYSDBA権限 13.2.1.1
- SYSOPER権限 13.2.1.1
- システム変更番号(SCN) 10.1.4
    - コミットされたトランザクション 10.2.5
    - 定義 10.2.5
    - 定義 9.1.1.3.1
    - 割当て時 11.4.3
- システム制御文 7.2.5
- システム・グローバル領域
    - 「SGA」を参照
- システム・ロック 9.4.3
    - 内部 9.4.3.3
    - ラッチ 9.4.3.1
    - mutex 9.4.3.2
- システム・モニター・プロセス(SMON) 15.4.1.4
    - 定義 15.4.1.4
    - Real Application Clusters 15.4.1.4
    - トランザクションのロールバック 13.4.4
- SYSTEM表領域 6.1.2
    - 格納されているデータ・ディクショナリ 12.5.1.1
    - オンライン要件 12.5.3.2
- SYSTEMユーザー名 2.1.4
- SYSユーザー名 2.1.4
    - データ・ディクショナリ表 6.1.3

T

- 表クラスタ
    - クラスタ・キー 2.3
    - 定義 2.3
    - ハッシュ・クラスタ 2.3.2, 2.3.2.2, 2.3.2.3
    - ハッシュ・クラスタの記憶域 2.3.2.4
    - 索引付けされた 2.3.1
    - 概要 2
    - スキャン 7.3.3
- 表パーティション 4.1.1.2.1
- 表
    - 属性クラスタ 2.4.2
    - 自動ビッグ・テーブル・キャッシング 14.4.1.4.1, 14.4.1.5
    - 実表 4.3.1, 6.1.2
    - 特性 2.2.3
    - クラスタ化 2.3
    - 圧縮 2.2.7, 4.1.2.2

    - 定義 1.1.2, 1.2.1, 2.1.1
    - ディメンション 4.6.1
    - ディレクトリ 12.2.2.1
    - DUAL 6.1.1.4
    - 動的パーティション化 15.4.3.2.3
    - 外部 2.6, 2.6.3 2.6.4
    - ファクト 4.6.1
    - 全表スキャン 3.1.1, 14.4.1.6.1
    - ヒープ構成 2.1.2, 3.6
    - 索引構成 3.6, 3.6.1, 3.6.2, 3.6.3
    - 整合性制約 5.1
    - 結合 3.3.2
    - マスター 4.4
    - 正規化された/正規化されていない 4.6.2
    - オブジェクト 2.9
    - 概要 2 2.2
    - パーティション化 4.1.1.1, 4.1.1.2.3, 4.1.1.2.4
    - パーティション 4.1.2
    - 永続 2.2
    - ストレージ 2.2.6
    - 一時 2.5, 2.5.3, 12.4.2.2
    - トランザクション 10.1.2.1
    - 切捨て 7.2.1, 12.3.2
    - ビュー 4.3
    - 仮想 6.2.2
- 表, 属性クラスタ 2.4
    - ディメンション階層 2.4.5
    - 線形順序付け 2.4.4
    - ゾーン・マップ 2.4.3, 2.4.3.1
- 表, 実 4.3
- 表, 外部 2.2
- 表, オブジェクト 2.2
- 表, 一時 2.2
- 表領域のポイント・イン・タイム・リカバリ 18.3.1.4.4
- 表領域 1.5.2.2, 12.5
    - 説明 12.5
    - ディクショナリ管理 12.1.2.2
    - ローカル管理 12.1.2.1
    - ロック 9.4.3.3
    - オフライン 12.5.3.2
    - オンライン 12.5.3.2
    - 概要 12.5
    - リカバリ 18.3.1.4.4
    - スキーマ・オブジェクト 2.1.2
    - 領域割当て 12.1.2

    - SYSTEM 6.1.2
    - 一時セグメントに使用 12.4.2.1
- 表領域セット 4.2.1
- タスク 15.1
- 一時ファイル 11.2.2
- 一時セグメント 2.5.2, 12.4.2.2
    - 割当て 12.4.2.1
    - 問合せ用の割当て 12.4.2.1
- 一時表 2.2
    - 作成 2.5.3
    - グローバル 2.5
    - プライベート 2.5
    - 用途 2.5.1
- 一時表領域 12.5.2
    - デフォルトの作成 12.5.2.2.1
    - ローカル 12.5.2.1
    - 共有 12.5.2.1
- スレッド
    - オンラインREDOログ 11.4.2
- TIMESTAMPデータ型 2.2.4.3, 2.2.4.3.2
- タイム・ゾーン 2.2.4.3.2
    - 日付/時間列 2.2.4.3.2
- TO_CHARファンクション 2.2.4.5
- TO_DATEファンクション 2.2.4.3.1, 2.2.4.5
- トレース・ファイル
    - LGWRトレース・ファイル 15.4.1.6.2
- トランザクション制御文 7.2.3
- トランザクション・ガード 10.3, 10.4.2, 17.2.1.4
    - 利点 10.3.1
    - 仕組み 10.3.2
- トランザクション・レベルの読取り一貫性 9.1.1.2
- トランザクション管理
    - 定義 1.4
- トランザクション 10
    - ACIDプロパティ 10.1
    - アクティブ 10.2.2, 12.4.3
    - システム変更番号の割当て 10.2.5
    - 自律型 10.5
        - PL/SQLブロック内 10.5
    - 開始 10.1.2.1
    - ブロッキング 9.2.1.2
    - コミット 10.3.2.1, 15.4.1.6.1
        - グループ・コミット 15.4.1.6.1
    - 書込みの競合 9.2.1.2
    - デッドロック 9.3.6, 10.1.3

        - 定義 1.4.1, 10.1
        - 分散 10.2.1, 10.6, 17.5.1.1
            - 自動解決 15.4.1.9
        - DML文 7.2.2
        - 終了 10.1.2.2
        - エンキュー 10.2.3.2
        - 冪等性 10.3, 17.2.1.4
        - インダウト
            - 自動解決 10.6.2
        - 処理中 10.3
        - 分離レベル 9.1.3, 9.2.2, 9.2.3
        - 分離 9.1.3
        - 論理トランザクションID 10.3.2.2
        - 命名 10.2.1
        - 読取り一貫性 9.1.1.2
        - ロールバック 10.2.4
            - 部分的な 10.2.3.1
        - セーブポイント 10.2.3
        - シリアライズ可能性 9.1
        - プロパティの設定 7.2.3
        - 構造 10.1.2
        - 停止 12.4.3
        - アプリケーションの終了 10.1.2.2
        - トランザクション制御文 7.2.3
        - トランザクション履歴 9.1.1.3.2
        - トランザクションID 10.1, 10.1.2.1
    - トランザクション表 10.1.2.1
        - リカバリ時にリセット 15.4.1.1
    - トリガー 8.1
        - カスケード 8.4.1
        - コンポーネント 8.4.4
        - 複合 8.4.3
        - ロールバックの効果 10.1.3
        - 起動(実行) 8.4.5
            - 必要な権限 8.4.5
        - INSTEAD OF 8.4.2
        - 概要 8.4
        - 制約 8.4.4
        - 行 8.4.2
        - 共有SQL領域 14.4.3.1.2
        - 単純 8.4.3
        - 文 8.4.2
        - 記憶域 8.4.6
        - タイミング 8.4.3
        - UNKNOWNでは起動されない 8.4.4

    - 使用 8.4.1
- TRUNCATE文 7.2.1, 12.3.2
- タプル, 定義 1.1.2
- 2フェーズ・コミット
    - トランザクション管理 10.6

U

- UNDO管理, 自動 12.4.3
- UNDO保存期間 12.5.1.3.2
- UNDOセグメント 10.1.2.1, 12.4.3
    - 読取り一貫性 9.1.1.3
    - 一時 11.4.1, 12.4.3, 12.4.3.3
- UNDO領域の管理
    - 自動UNDO管理モード 12.5.1.3.1
- UNDO表領域 12.4.3, 12.5.1.3, 12.5.1.3.1, 12.5.1.3.2
    - UNDO保存期間 9.2.3
- Unicode 2.2.4.1
- 統合監査証跡 16.1.2.3, 17.1.5.3, 17.1.6.2
- 一意索引 3.1.5
- 一意キー制約 5.2, 5.2.2
    - コンポジット・キー 5.2.2
    - NOT NULL制約 5.2.2
- 計画外停止時間
    - 停止時間の回避 17.2.1.2
- 更新可能な結合ビュー 4.3.2
- UPDATE NO ACTION制約 5.2.4.3
- 更新
    - 消失 9.2.1.2
    - 親キーのある表 9.4.1.3
    - ビューの更新可能性 4.3.2, 8.4.2
    - 更新可能な結合ビュー 4.3.2
- UPDATE文 7.2.2
- アップグレード
    - データベース 12.5.1.2, 12.5.1.3.1, 17.2.2.3, 19.3.1
- UROWIDデータ型 2.2.4.4
- USER_ビュー 6.1.1.3
- ユーザー・グローバル領域(UGA) 14.1.1
- ユーザー・プログラム・インタフェース(UPI) 16.3.1
- ユーザー, データベース 2.1.1
    - 認証 17.1.2
    - 名前 6.1
        - セッションと接続 15.2.2
    - 権限 2.1, 17.1.1.1

    - 一時表領域 12.4.2.1
- ユーザー・セグメント 12.4.1.1
- UTL_HTTPパッケージ 8.2.2

V

- V$ ビュー 6.2.1
    - データベース・オブジェクト・メタデータ 6.3
    - 記憶域 6.2.2
- VARCHAR2データ型 2.2.4.1, 2.2.4.1.1
- 変数
    - 埋込みSQL 7.2.6
- ビュー 4.3
    - 分析 17.4.4.2
    - 実表 4.3
    - 特性 4.3.1
    - 制約が間接的に影響する 5.3.3.2
    - データ・アクセス 4.3.1.2
    - データ・ディクショナリ
        - 更新可能な列 4.3.2
    - 定義 2.1.1
    - DML 4.3.1.1
    - 動的パフォーマンス 6.2, 6.2.1
    - 索引 4.3.1.2
    - INSTEAD OFトリガー 8.4.2
    - 整合性制約 4.3.1.1
    - マテリアライズド 4.4
    - オブジェクト 4.3.3
    - スキーマ・オブジェクトの依存性 4.3.1
    - 記憶域 4.3.1
    - 更新可能性 4.3.2
    - 使用 4.3
    - V$ 6.2
- 仮想列 2.2.1.1, 2.2.6.1, 3.4.2

W

- ウェアハウス
    - マテリアライズド・ビュー 4.4
- Webサービス 1.5.4.1, 16.1.2.4
- 事前書込み 15.4.1.6

X

- XA
    - ラージ・プール内のセッション・メモリー 14.4.4
- XMLTypeデータ型 19.3.4.1
- XStream 17.5.2.2.1

Z

- Zero Data Loss Recovery Appliance。
    - 「リカバリ・アプライアンス」を参照
- ゾーン・マップ 2.4.3, 2.4.3.1