

Oracle® Database

データベース開発ガイド

19c

F16193-05(原本部品番号:E96334-06)

2023年11月

タイトルおよび著作権情報

Oracle Databaseデータベース開発ガイド, 19c

F16193-05

[Copyright ©](#) 1996, 2023, Oracle and/or its affiliates.

原著者: Jayashree Sharma

原著協力者: Amith Kumar、Louise Morin、Chuck Murray、Tom Kyte、D.Adams、L.Ashdown、S.Moore、E.Paapanen、R.Strohm、R.Ward

原著協力者: D.Alpern、G.Arora、T.Chang、B.Cheng、R.Day、R.Decker、G.Doherty、A.Ganesh、M.Hartstein、Y.Hu、J.Huang、C.Iyer、N.Jain、V.Krishnaswamy、R.Kumar、S.Kumar、C.Lei、B.Llewellyn、K.Mohan、V.Moore、J.Muller、R.Murthy、R.Pang、B.Sinha、S.Vemuri、W.Wang、D.Wong

目次

- [表一覧](#)
- [タイトルおよび著作権情報](#)
- [はじめに](#)
 - [対象読者](#)
 - [ドキュメントのアクセシビリティ](#)
 - [関連ドキュメント](#)
 - [表記規則](#)
- [Oracle Database[®]リリース19cにおける変更点](#)
 - [新機能](#)
 - [変更問合せ通知の新しいOCI登録モード](#)
 - [非推奨となった機能](#)
 - [サポート対象外機能](#)
- [第I部 データベース開発の基礎](#)
 - [1 設計の基本](#)
 - [1.1 パフォーマンスを考慮した設計](#)
 - [1.2 スケーラビリティを考慮した設計](#)
 - [1.3 拡張性を考慮した設計](#)
 - [1.3.1 データ・カートリッジ](#)
 - [1.3.2 外部プロシージャ](#)
 - [1.3.3 ユーザー定義ファンクションと集計ファンクション](#)
 - [1.3.4 オブジェクト・リレーショナル機能](#)
 - [1.4 セキュリティを考慮した設計](#)
 - [1.5 可用性を考慮した設計](#)
 - [1.6 移植性を考慮した設計](#)
 - [1.7 診断性を考慮した設計](#)
 - [1.8 特殊な環境を考慮した設計](#)
 - [1.8.1 データ・ウェアハウス](#)
 - [1.8.2 オンライン・トランザクション処理\(OLTP\)](#)
 - [1.9 特殊なシナリオのための機能](#)
 - [1.9.1 SQL分析関数](#)
 - [1.9.2 マテリアライズド・ビュー](#)
 - [1.9.3 パーティション化](#)
 - [1.9.4 時間的な有効性のサポート](#)
 - [2 データベース・アプリケーションの接続方法](#)
 - [2.1 接続プールの設計ガイドライン](#)
 - [2.1.1 接続ストーム](#)
 - [2.1.2 接続ストームを回避するためのガイドライン: 静的プールの使用](#)
 - [2.2 ログイン方法の接続ガイドライン](#)
 - [2.3 プログラム・セッション・リークを回避する設計ガイドライン](#)
 - [2.3.1 失われた接続プール](#)
 - [2.3.2 セッション・リークの確認](#)

- [2.3.3 ロック・リーク](#)
 - [2.3.4 論理的な破損](#)
 - [2.4 ランタイム接続ロード・バランシングの使用方法](#)
 - [2.4.1 ランタイム接続ロード・バランシングについて](#)
 - [2.4.2 ランタイム接続ロード・バランシングの有効化と無効化](#)
 - [2.4.2.1 OCI](#)
 - [2.4.2.2 OCCI](#)
 - [2.4.2.3 JDBC](#)
 - [2.4.2.4 ODP.NET](#)
 - [2.4.3 ロード・バランシング・アドバイザFANイベントの受信](#)
- [3 パフォーマンスとスケーラビリティ](#)
 - [3.1 パフォーマンス戦略](#)
 - [3.1.1 実行するデータ・モデルの効率的な設計](#)
 - [3.1.1.1 アプリケーションのデータ要件の分析](#)
 - [3.1.1.2 アプリケーションのデータベース設計の作成](#)
 - [3.1.1.2.1 論理設計の作成](#)
 - [3.1.1.2.2 物理設計の作成](#)
 - [3.1.1.3 データベース・アプリケーションの実装](#)
 - [3.1.1.4 データベースとデータベース・アプリケーションのメンテナンス](#)
 - [3.1.2 パフォーマンス目標\(メトリック\)の設定](#)
 - [3.1.3 アプリケーションのベンチマーク](#)
 - [3.2 パフォーマンスのためのツール](#)
 - [3.2.1 DBMS_APPLICATION_INFOパッケージ](#)
 - [3.2.2 SQLトレース機能\(SQL_TRACE\)](#)
 - [3.2.3 EXPLAIN PLAN文](#)
 - [3.3 データベース・パフォーマンスの監視](#)
 - [3.3.1 自動データベース診断モニター\(ADDM\)](#)
 - [3.3.2 データベースのパフォーマンスのリアルタイムの監視](#)
 - [3.3.3 パフォーマンス関連のアラートに対する対応](#)
 - [3.3.4 SQLアドバイザとメモリー・アドバイザ](#)
 - [3.4 パフォーマンスのテスト](#)
 - [3.5 クライアント結果キャッシュの使用](#)
 - [3.5.1 クライアント結果キャッシュの使用について](#)
 - [3.5.2 クライアント結果キャッシュの利点](#)
 - [3.5.3 クライアント結果キャッシュを使用するガイドライン](#)
 - [3.5.3.1 SQLヒント](#)
 - [3.5.3.2 表注釈](#)
 - [3.5.3.3 セッション・パラメータ](#)
 - [3.5.3.4 有効な表結果キャッシュ・モード](#)
 - [3.5.3.5 有効な表結果キャッシュ・モードの表示](#)
 - [3.5.3.6 結果キャッシュ・モードの使用例](#)
 - [3.5.3.7 クライアント結果キャッシュで結果がキャッシュされない問合せ](#)
 - [3.5.4 クライアント結果キャッシュの一貫性](#)

- [3.5.5 クライアント結果キャッシュに関するデプロイメント時設定](#)
 - [3.5.5.1 サーバー初期化パラメータ](#)
 - [3.5.5.1.1 COMPATIBLE](#)
 - [3.5.5.1.2 CLIENT_RESULT_CACHE_SIZE](#)
 - [3.5.5.1.3 CLIENT_RESULT_CACHE_LAG](#)
 - [3.5.5.2 クライアント構成パラメータ](#)
- [3.5.6 クライアント結果キャッシュの統計](#)
- [3.5.7 クライアント結果キャッシュの検証](#)
 - [3.5.7.1 実行時間の測定](#)
 - [3.5.7.2 V\\$MYSTATの問合せ](#)
 - [3.5.7.3 V\\$SQLAREAの問合せ](#)
- [3.5.8 クライアント結果キャッシュとサーバー結果キャッシュ](#)
- [3.5.9 クライアント結果キャッシュのデモ・ファイル](#)
- [3.5.10 クライアント結果キャッシュの以前のリリースとの互換性](#)
- [3.6 文キャッシング](#)
- [3.7 OCIクライアント文キャッシュの自動チューニング](#)
- [3.8 クライアント側デプロイメント・パラメータ](#)
- [3.9 問合せ変更通知の使用](#)
- [3.10 データベース常駐接続プールの使用](#)
 - [3.10.1 データベース常駐接続プールの理解](#)
 - [3.10.2 DRCPの構成](#)
 - [3.10.3 プロキシ・セッションの共有](#)
 - [3.10.4 DRCPでのJDBCの使用](#)
 - [3.10.5 DRCPでのOCIセッション・プールAPIの使用](#)
 - [3.10.6 セッション純正值および接続クラス](#)
 - [3.10.6.1 セッション純正值](#)
 - [3.10.6.2 接続クラス](#)
 - [3.10.6.3 例: 接続クラスをHRMSとして設定](#)
 - [3.10.6.4 例: 接続クラスをRECMSとして設定](#)
 - [3.10.6.5 セッション純正值および接続クラスのデフォルト](#)
 - [3.10.7 データベース常駐接続プールの開始](#)
 - [3.10.8 DRCPの有効化](#)
 - [3.10.9 OCIアプリケーションにおけるDRCPのスケーラビリティの利用](#)
 - [3.10.10 JavaアプリケーションにおけるDRCPのスケーラビリティの利用](#)
 - [3.10.11 DRCPの使用に関するベスト・プラクティス](#)
 - [3.10.12 互換性および移行](#)
 - [3.10.13 DRCPとOracle Databaseのネイティブ・ネットワーク暗号化の併用](#)
 - [3.10.14 DRCPの制限事項](#)
 - [3.10.15 カスタム・プールでのDRCPの使用](#)
 - [3.10.16 明示的なステートフルまたはステートレス・セッションのマーキング](#)
 - [3.10.17 Oracle Real Application ClustersでのDRCPの使用方法](#)
 - [3.10.18 プラガブル・データベースでのDRCPの使用](#)
 - [3.10.19 DRCPとData Guard](#)

- [3.11 Memoptimizeプール](#)
 - [3.12 Oracle RACシャーディング](#)
- [4 Oracle Real-World Performanceのアプリケーションの設計](#)
 - [4.1 バインド変数の使用](#)
 - [4.2 インストルメンテーションの使用](#)
 - [4.3 セット・ベース処理の使用](#)
 - [4.3.1 繰返しデータ処理](#)
 - [4.3.1.1 繰返しデータ処理について](#)
 - [4.3.1.2 繰返しデータ処理: 行ごと](#)
 - [4.3.1.3 繰返しデータ処理: 配列](#)
 - [4.3.1.4 繰返しデータ処理: 手動の並列度](#)
 - [4.3.2 セット・ベース処理](#)
- [5 セキュリティ](#)
 - [5.1 権限の付与、ロール、最小限の権限によるユーザー・アクセスの有効化](#)
 - [5.2 データベース・ログインの自動化](#)
 - [5.3 ファイングレイン・アクセス制御によるユーザー・アクセスの制御](#)
 - [5.4 プロシージャとファンクションに対する実行者の権限と定義者の権限の使用](#)
 - [5.4.1 実行者の権限と定義者の権限の概要](#)
 - [5.4.2 実行者の権限のプロシージャとファンクションを実行するユーザーの保護](#)
 - [5.4.3 Javaストアド・プロシージャでのデフォルト権限の処理方法](#)
 - [5.5 アプリケーションに対する外部プロシージャの管理](#)
 - [5.6 ユーザー・アクティビティの監査](#)
- [6 高可用性](#)
 - [6.1 透過的アプリケーション・フェイルオーバー\(TAF\)](#)
 - [6.1.1 透過的アプリケーション・フェイルオーバーの理解](#)
 - [6.1.2 透過的アプリケーション・フェイルオーバーの構成](#)
 - [6.1.3 透過的アプリケーション・フェイルオーバー・コールバックの使用](#)
 - [6.2 Traffic DirectorモードのOracle Connection Manager](#)
 - [6.3 高速アプリケーション通知\(FAN\)と高速接続フェイルオーバー\(FCF\)](#)
 - [6.3.1 高速アプリケーション通知\(FAN\)について](#)
 - [6.3.2 FANイベント通知の受信について](#)
 - [6.3.3 高速接続フェイルオーバー\(FCF\)について](#)
 - [6.4 アプリケーション・コンティニューイティとトランザクション・ガード](#)
 - [6.4.1 アプリケーション・コンティニューイティの概要](#)
 - [6.4.2 トランザクション・ガードの概要](#)
 - [6.5 データベース・クラウドのサービスおよびロード管理](#)
 - [6.5.1 データベース・クラウドのサービスおよびロード管理について](#)
- [7 高度なPL/SQL機能](#)
 - [7.1 PL/SQLのデータ型](#)
 - [7.2 動的SQL](#)
 - [7.3 PL/SQL最適化レベル](#)
 - [7.4 システム固有の実行のためのPL/SQLユニットのコンパイル](#)
 - [7.5 例外処理](#)

- [7.6 条件付きコンパイル](#)
 - [7.7 バルク・バインド](#)
- [第II部 アプリケーション開発者用のSQL](#)
 - [8 アプリケーション開発者用のSQL処理](#)
 - [8.1 SQL文の処理の説明](#)
 - [8.1.1 SQL文の処理のステージ](#)
 - [8.1.2 共有SQL領域](#)
 - [8.2 操作のトランザクションへのグループ化](#)
 - [8.2.1 トランザクションへの操作のグループ化方法の決定](#)
 - [8.2.2 トランザクションのパフォーマンスの改善](#)
 - [8.2.3 コミットREDO動作の管理](#)
 - [8.2.4 リカバリ可能な停止後のトランザクション結果の確認](#)
 - [8.2.4.1 トランザクション・ガードの理解](#)
 - [8.2.4.1.1 トランザクション・ガードがLTXIDを使用する方法](#)
 - [8.2.4.2 DBMS_APP_CONT.GET_LTXID_OUTCOMEの理解](#)
 - [8.2.4.2.1 CLIENT_LTXIDパラメータ](#)
 - [8.2.4.2.2 COMMITTEDパラメータ](#)
 - [8.2.4.2.3 USER_CALL_COMPLETEDパラメータ](#)
 - [8.2.4.2.4 例外](#)
 - [8.2.4.3 トランザクション・ガードの使用](#)
 - [8.3 読取り専用トランザクションでの反復可能読取りの保証](#)
 - [8.4 明示的な表のロック](#)
 - [8.4.1 表のロックを取得するために必要な権限](#)
 - [8.4.2 ロック方法の選択](#)
 - [8.4.2.1 ROW SHARE MODEおよびROW EXCLUSIVE MODEでロックする場合](#)
 - [8.4.2.2 SHARE MODEでロックする場合](#)
 - [8.4.2.3 SHARE ROW EXCLUSIVE MODEでロックする場合](#)
 - [8.4.2.4 EXCLUSIVE MODEでロックする場合](#)
 - [8.4.3 Oracle Databaseによる表ロック制御](#)
 - [8.4.4 明示的な行ロック](#)
 - [8.4.5 明示的なロックにおける同時実行性の例](#)
 - [8.5 Oracle Lock Managementサービス\(ユーザー・ロック\)の使用](#)
 - [8.5.1 ユーザー・ロックを使用する場合](#)
 - [8.5.2 ロックの表示および監視](#)
 - [8.6 シリアライズ可能トランザクションを使用した同時実行性の制御](#)
 - [8.6.1 トランザクションの相互作用と分離レベル](#)
 - [8.6.2 分離レベルの設定](#)
 - [8.6.3 シリアライズ可能トランザクションおよび参照整合性](#)
 - [8.6.4 READ COMMITTEDおよびSERIALIZABLE分離レベル](#)
 - [8.6.4.1 トランザクション集合の整合性の違い](#)
 - [8.6.4.1.1 Oracle Database](#)
 - [8.6.4.1.2 他のデータベース・システム](#)

- [8.6.4.2 トランザクション分離レベルの選択](#)
 - [8.7 非ブロック化およびブロック化のDDL文](#)
 - [8.8 自律型トランザクション](#)
 - [8.8.1 自律型トランザクションの例](#)
 - [8.8.1.1 商品の注文](#)
 - [8.8.1.2 銀行口座からの出金処理](#)
 - [8.8.1.2.1 使用例1: 十分な預金残高がある場合](#)
 - [8.8.1.2.2 使用例2: 貸越し保護があり預金残高が不十分な場合](#)
 - [8.8.1.2.3 使用例3: 貸越し保護がなく預金残高が不十分な場合](#)
 - [8.8.2 自律型ルーチンの宣言](#)
 - [8.9 記憶域割当てエラー後の実行の再開](#)
 - [8.9.1 再開可能記憶域割当てを含む操作](#)
 - [8.9.2 一時停止された記憶域割当ての処理](#)
 - [8.9.2.1 アプリケーションのAFTER SUSPENDトリガーの使用](#)
 - [8.9.2.2 一時停止された文の確認](#)
- [9 データベース・アプリケーションにおけるSQLデータ型の使用](#)
 - [9.1 正確かつ最も固有のデータ型の使用](#)
 - [9.1.1 正しいデータ型によるデータ整合性の向上](#)
 - [9.1.2 最も固有のデータ型による記憶域要件の削減](#)
 - [9.1.3 正しいデータ型によるパフォーマンスの向上](#)
 - [9.2 文字データの表現](#)
 - [9.3 数値データの表現](#)
 - [9.3.1 浮動小数点数の構成要素](#)
 - [9.3.2 浮動小数点数形式](#)
 - [9.3.2.1 浮動小数点2進形式](#)
 - [9.3.3 ネイティブ浮動小数点データ型の特殊な数値の表現](#)
 - [9.3.4 ネイティブ浮動小数点値の比較](#)
 - [9.3.5 ネイティブ浮動小数点データ型を使用した算術演算](#)
 - [9.3.6 ネイティブ浮動小数点データ型の変換ファンクション](#)
 - [9.3.7 ネイティブ浮動小数点データ型のクライアント・インタフェース](#)
 - [9.4 日時データの表現](#)
 - [9.4.1 現在の日付および時刻の表示](#)
 - [9.4.2 日付の挿入および表示](#)
 - [9.4.3 時刻の挿入および表示](#)
 - [9.4.4 日時データ型を使用した算術演算](#)
 - [9.4.5 日時データ型の変換関数](#)
 - [9.4.6 日時データ型のインポートおよびエクスポート](#)
 - [9.5 特殊なデータの表現](#)
 - [9.5.1 空間データの表現](#)
 - [9.5.2 大量のデータの表現](#)
 - [9.5.2.1 ラージ・オブジェクト\(LOB\)](#)
 - [9.5.2.2 LONGおよびLONG RAWデータ型](#)
 - [9.5.3 検索可能なテキストの表現](#)

- [9.5.4 XMLデータの表現](#)
 - [9.5.5 動的に型指定されたデータの表現](#)
 - [9.5.6 ANSI、DB2およびSQL/DSデータの表現](#)
- [9.6 アドレスによる行の識別](#)
- [9.7 SQL演算子とファンクションのメタデータの表示](#)
 - [9.7.1 ARGnデータ型](#)
 - [9.7.2 DISP_TYPEデータ型](#)
 - [9.7.3 SQLデータ型ファミリ](#)
- [10 データベース・アプリケーションでの正規表現の使用](#)
 - [10.1 正規表現の概要](#)
 - [10.2 Oracle SQLの正規表現のサポート](#)
 - [10.3 Oracle SQLおよびPOSIX正規表現標準](#)
 - [10.4 Oracle SQL正規表現内の演算子](#)
 - [10.4.1 Oracle SQL正規表現内のPOSIX演算子](#)
 - [10.4.2 Oracle SQLのPOSIX標準に対する多言語拡張機能](#)
 - [10.4.3 Oracle SQLのPOSIX標準に対するPERLの影響を受ける拡張機能](#)
 - [10.5 SQL文での正規表現の使用: 例](#)
 - [10.5.1 制約を使用した電話番号書式の規定](#)
 - [10.5.2 例: 正規表現を使用した電話番号書式の規定](#)
 - [10.5.3 例: 正しい書式と誤った書式での電話番号の挿入](#)
 - [10.5.4 後方参照を使用した文字の再配置](#)
- [11 データベース・アプリケーションでの索引の使用](#)
 - [11.1 索引を管理するためのガイドライン](#)
 - [11.2 索引の管理](#)
 - [11.3 ドメイン索引を使用する場合](#)
 - [11.4 ファンクション索引を使用する場合](#)
 - [11.4.1 ファンクション索引のメリット](#)
 - [11.4.2 ファンクション索引のデメリット](#)
 - [11.4.3 例: 算術式の事前計算用のファンクション索引](#)
 - [11.4.4 例: オブジェクト列でのファンクション索引](#)
 - [11.4.5 例: 大/小文字を区別しないより高速の検索用のファンクション索引](#)
 - [11.4.6 例: 言語依存ソート用のファンクション索引](#)
- [12 データベース・アプリケーションにおけるデータ整合性のメンテナンス](#)
 - [12.1 制約を使用したビジネス・ルールの施行](#)
 - [12.2 制約とアプリケーション・コードの両方を使用したビジネス・ルールの施行](#)
 - [12.3 制約で使用する索引の作成](#)
 - [12.4 NOT NULL制約を使用する場合](#)
 - [12.5 デフォルトの列値を使用する場合](#)
 - [12.6 表の主キーの選択\(PRIMARY KEY制約\)](#)
 - [12.7 UNIQUE制約を使用する場合](#)
 - [12.8 FOREIGN KEY制約を使用した参照整合性の施行](#)
 - [12.8.1 FOREIGN KEY制約およびNULL値](#)
 - [12.8.2 親表と子表との関連の定義](#)

- [12.8.3 複数のFOREIGN KEY制約に関する規則](#)
 - [12.8.4 制約チェックの遅延](#)
- [12.9 制約に対応付けられた索引の領域と時間のオーバーヘッドの最小化](#)
- [12.10 外部キーを索引付けするためのガイドライン](#)
- [12.11 分散データベース内の参照整合性](#)
- [12.12 CHECK制約を使用する場合](#)
 - [12.12.1 CHECK制約の制限](#)
 - [12.12.2 CHECK制約の設計](#)
 - [12.12.3 複数のCHECK制約に関する規則](#)
 - [12.12.4 CHECKおよびNOT NULL制約の選択](#)
- [12.13 制約の定義の例](#)
 - [12.13.1 制約の定義に必要な権限](#)
 - [12.13.2 制約のネーミング](#)
- [12.14 制約の使用可能および使用禁止](#)
 - [12.14.1 制約を使用禁止にする理由](#)
 - [12.14.2 制約を使用可能にする\(デフォルト\)](#)
 - [12.14.3 制約を使用禁止にする](#)
 - [12.14.4 既存の制約を使用可能にする](#)
 - [12.14.5 既存の制約を使用禁止にする](#)
 - [12.14.6 キー制約の使用可能および使用禁止に関するガイドライン](#)
 - [12.14.7 制約の例外の修正](#)
- [12.15 制約の変更](#)
- [12.16 制約名の変更](#)
- [12.17 制約の削除](#)
- [12.18 FOREIGN KEY制約の管理](#)
 - [12.18.1 外部キー列のデータ型および名前](#)
 - [12.18.2 複合外部キーにおける列の制限](#)
 - [12.18.3 デフォルトでの外部キーの主キー参照](#)
 - [12.18.4 FOREIGN KEY制約の作成に必要な権限](#)
 - [12.18.5 外部キーによる参照整合性の施行方法の選択](#)
- [12.19 制約に関する情報の表示](#)
- [第III部 アプリケーション開発者用のPL/SQL](#)
 - [13 PL/SQLサブプログラムおよびパッケージのコード化](#)
 - [13.1 PL/SQLサブプログラムの概要](#)
 - [13.2 PL/SQLパッケージの概要](#)
 - [13.3 PL/SQLユニットの概要](#)
 - [13.3.1 PLSQL_OPTIMIZE_LEVELコンパイル・パラメータ](#)
 - [13.4 PL/SQLサブプログラムおよびパッケージの作成](#)
 - [13.4.1 サブプログラムおよびパッケージの作成に必要な権限](#)
 - [13.4.2 サブプログラムおよびパッケージの作成](#)
 - [13.4.3 PL/SQLオブジェクト・サイズの制限](#)
 - [13.4.4 PL/SQLのデータ型](#)
 - [13.4.4.1 PL/SQLのスカラー・データ型](#)

- [13.4.4.1.1 SQLデータ型](#)
 - [13.4.4.1.2 BOOLEANデータ型](#)
 - [13.4.4.1.3 PLS_INTEGERおよびBINARY_INTEGERデータ型](#)
 - [13.4.4.1.4 REF CURSORデータ型](#)
 - [13.4.4.1.5 ユーザー定義のPL/SQLサブタイプ](#)
- [13.4.4.2 PL/SQLのコンポジット・データ型](#)
- [13.4.4.3 抽象データ型](#)
- [13.4.5 クライアントへの結果セットの戻し](#)
 - [13.4.5.1 カーソル変数のメリット](#)
 - [13.4.5.2 カーソル変数のデメリット](#)
 - [13.4.5.2.1 カーソル変数の解析のペナルティ](#)
 - [13.4.5.2.2 カーソル変数の複数行フェッチのペナルティ](#)
 - [13.4.5.3 問合せ結果の暗黙的戻し](#)
- [13.4.6 ファンクションからの大量のデータの戻し](#)
- [13.4.7 PL/SQLファンクション結果キャッシュ](#)
- [13.4.8 バルク・バインドの概要](#)
 - [13.4.8.1 コレクションを参照するDML文](#)
 - [13.4.8.2 コレクションを参照するSELECT文](#)
 - [13.4.8.3 コレクションを参照しDMLを戻すFORループ](#)
- [13.4.9 PL/SQLの動的SQL](#)
- [13.5 PL/SQLサブプログラムおよびパッケージの変更](#)
- [13.6 推奨されないパッケージ、サブプログラムおよびタイプ](#)
- [13.7 PL/SQLサブプログラムおよびパッケージの削除](#)
- [13.8 システム固有の実行のためのPL/SQLユニットのコンパイル](#)
- [13.9 ストアドPL/SQLサブプログラムの起動](#)
 - [13.9.1 ストアド・サブプログラムの起動に必要な権限](#)
 - [13.9.2 Oracleツールを使用した対話形式によるサブプログラムの起動](#)
 - [13.9.3 別のサブプログラムからのサブプログラムの起動](#)
 - [13.9.4 リモート・サブプログラムの起動](#)
 - [13.9.4.1 リモート・サブプログラムのシノニム](#)
 - [13.9.4.2 リモート・サブプログラムを起動するトランザクション](#)
- [13.10 SQL文からのストアドPL/SQLファンクションの起動](#)
 - [13.10.1 SQL文からPL/SQLファンクションを起動する理由](#)
 - [13.10.2 SQL文の中でPL/SQLファンクションを使用できる箇所](#)
 - [13.10.3 SQL式の中でPL/SQLファンクションを使用できる条件](#)
 - [13.10.4 SQL文から起動したPL/SQLファンクションの副作用の制御](#)
 - [13.10.4.1 SQL文から起動したファンクションの制限](#)
 - [13.10.4.2 パラレルSQL文から起動したPL/SQLファンクション](#)
 - [13.10.4.3 PRAGMA RESTRICT_REFERENCES](#)
 - [13.10.4.3.1 例: PRAGMA RESTRICT_REFERENCES](#)
 - [13.10.4.3.2 アサーションTRUSTの指定](#)
 - [13.10.4.3.3 静的SQL文と動的SQL文の違い](#)
- [13.11 ストアド・サブプログラムの分析とデバッグ](#)

- [13.11.1 PL/Scope](#)
 - [13.11.2 PL/SQL階層プロファイラ](#)
 - [13.11.3 PL/SQLおよびJavaのデバッグ](#)
 - [13.11.3.1 デバッグ用のコードのコンパイル](#)
 - [13.11.3.2 PL/SQLストアド・サブプログラムおよびJavaストアド・サブプログラムのデバッグ権限](#)
 - [13.12 パッケージの無効化とセッションの状態](#)
 - [13.13 例: ORA-04068エラーの発生](#)
 - [13.14 例: ORA-04068の検出](#)
- [14 PL/Scopeの使用](#)
 - [14.1 PL/Scopeの概要](#)
 - [14.2 PL/Scopeの使用に必要な権限](#)
 - [14.3 識別子と文の収集の指定](#)
 - [14.4 PL/Scopeデータによる領域の使用量](#)
 - [14.5 PL/Scopeデータの表示](#)
 - [14.5.1 PL/SQLおよびSQL識別子の静的データ・ディクショナリ・ビュー](#)
 - [14.5.1.1 PL/Scopeが収集するPL/SQLおよびSQL識別子の型](#)
 - [14.5.1.2 識別子の使用方法について](#)
 - [14.5.1.3 識別子の使用方法: 一意キー](#)
 - [14.5.1.4 識別子の使用方法について: コンテキスト](#)
 - [14.5.1.5 識別子のシグネチャについて](#)
 - [14.5.2 SQL文の静的データ・ディクショナリ・ビュー](#)
 - [14.5.2.1 PL/Scopeが収集するSQL文の型](#)
 - [14.5.2.2 文の場所: 一意キー](#)
 - [14.5.2.3 SQL文に使用方法について: コンテキスト](#)
 - [14.5.2.4 SQL文のシグネチャについて](#)
 - [14.5.3 SQL Developer](#)
 - [14.6 PL/SQLコードの管理に役立つデータ・ディクショナリ・ビューの概要](#)
 - [14.7 サンプルPL/Scopeセッション](#)
- [15 PL/SQL階層プロファイラの使用](#)
 - [15.1 PL/SQL階層プロファイラの概要](#)
 - [15.2 プロファイル・データの収集](#)
 - [15.3 RAWプロファイラ出力の理解](#)
 - [15.3.1 追跡されるサブプログラムのネームスペース](#)
 - [15.3.2 特別な関数名](#)
 - [15.4 プロファイル・データの分析](#)
 - [15.4.1 階層プロファイラ表の作成](#)
 - [15.4.2 階層プロファイラ表の理解](#)
 - [15.4.2.1 階層プロファイラ・データベース表の列](#)
 - [15.4.2.2 オーバーロードされたサブプログラムの区別](#)
 - [15.4.2.3 サンプルのPL/SQLプロシージャの階層プロファイラ表](#)
 - [15.4.2.4 オプションを使用したDBMS_HPROF.analyzeのコール例](#)
 - [15.5 plshprofユーティリティ](#)

- [15.5.1 plshprofのオプション](#)
- [15.5.2 単一のRAWプロファイラ出力ファイルから生成するHTMLレポート](#)
 - [15.5.2.1 レポートの最初のページ](#)
 - [15.5.2.2 関数レベルのレポート](#)
 - [15.5.2.3 モジュール・レベルのレポート](#)
 - [15.5.2.4 ネームスペース・レベルのレポート](#)
 - [15.5.2.5 関数の親子レポート](#)
 - [15.5.2.6 PL/SQL階層プロファイラのSQLレベルのレポートの理解](#)
- [15.5.3 2つのRAWプロファイラ出力ファイルから生成するHTML差分レポート](#)
 - [15.5.3.1 差分レポートの表記規則](#)
 - [15.5.3.2 差分レポートの最初のページ](#)
 - [15.5.3.3 関数レベルの差分レポート](#)
 - [15.5.3.4 モジュール・レベルの差分レポート](#)
 - [15.5.3.5 ネームスペース・レベルの差分レポート](#)
 - [15.5.3.6 関数の親子差分レポート](#)
- [16 品質を維持するためのPL/SQL基本ブロック・カバレッジの使用](#)
 - [16.1 PL/SQL基本ブロック・カバレッジの概要](#)
 - [16.2 PL/SQLコード・カバレッジ・データの収集](#)
 - [16.3 PL/SQLコード・カバレッジ表の説明](#)
- [17 PL/SQL Webアプリケーションの開発](#)
 - [17.1 PL/SQL Webアプリケーションの概要](#)
 - [17.2 PL/SQL Webアプリケーションの実装](#)
 - [17.2.1 PL/SQLゲートウェイ](#)
 - [17.2.1.1 mod_plsql](#)
 - [17.2.1.2 埋込みPL/SQLゲートウェイ](#)
 - [17.2.2 PL/SQL Web Toolkit](#)
 - [17.3 mod_plsqlゲートウェイを使用したクライアント・リクエストのPL/SQL Webアプリケーションへのマップ](#)
 - [17.4 埋込みPL/SQLゲートウェイの使用](#)
 - [17.4.1 埋込みPL/SQLゲートウェイによるクライアント・リクエストの処理方法](#)
 - [17.4.2 埋込みPL/SQLゲートウェイのインストール](#)
 - [17.4.3 埋込みPL/SQLゲートウェイの構成](#)
 - [17.4.3.1 埋込みPL/SQLゲートウェイの構成: 概要](#)
 - [17.4.3.2 埋込みPL/SQLゲートウェイのユーザー認証に関する構成](#)
 - [17.4.3.2.1 DBMS_EPGによる静的認証の構成](#)
 - [17.4.3.2.2 DBMS_EPGによる動的認証の構成](#)
 - [17.4.3.2.3 DBMS_EPGによる匿名認証の構成](#)
 - [17.4.3.2.4 DADの認証モードの決定](#)
 - [17.4.3.2.5 例: DADの作成および構成](#)
 - [17.4.3.2.6 例: DADの認証モードの決定](#)
 - [17.4.3.2.7 例: すべてのDADの認証モードの決定](#)
 - [17.4.3.2.8 例: 無効なDAD認可の表示](#)
 - [17.4.3.2.9 埋込みPL/SQLゲートウェイ構成の調査](#)

- [17.4.4 埋込みPL/SQLゲートウェイを介したPL/SQLストアード・サブプログラムの起動](#)
 - [17.4.5 埋込みPL/SQLゲートウェイによるアプリケーション・アクセスの保護](#)
 - [17.4.6 埋込みPL/SQLゲートウェイの制限事項](#)
 - [17.4.7 埋込みPL/SQLゲートウェイの使用: 使用例](#)
- [17.5 PL/SQLを使用したHTML出力の生成](#)
- [17.6 PL/SQL Webアプリケーションへのパラメータ渡し](#)
 - [17.6.1 HTMLフォームからのリスト・パラメータおよびドロップダウン・リスト・パラメータ渡し](#)
 - [17.6.2 HTMLフォームからのオプション・パラメータおよびチェック・ボックス・パラメータ渡し](#)
 - [17.6.3 HTMLフォームからの入力フィールド・パラメータ渡し](#)
 - [17.6.4 HTMLフォームからの非表示パラメータ渡し](#)
 - [17.6.5 HTMLフォームからのファイルのアップロード](#)
 - [17.6.6 記入済HTMLフォームの送信](#)
 - [17.6.7 HTMLフォームの欠落した入力処理](#)
 - [17.6.8 Webページ間での状態情報の保持](#)
- [17.7 PL/SQLサブプログラムでのネットワーク操作の実行](#)
 - [17.7.1 Internet Protocol Version 6\(IPv6\)のサポート](#)
 - [17.7.2 PL/SQLからの電子メールの送信](#)
 - [17.7.3 PL/SQLからのホスト名またはアドレスの取得](#)
 - [17.7.4 PL/SQLからのTCP/IP接続の使用](#)
 - [17.7.5 PL/SQLからのHTTP URLコンテンツの取得](#)
 - [17.7.6 PL/SQLからの表、イメージ・マップ、Cookie、およびCGI変数の使用](#)
- [18 連続問合せ通知\(CQN\)の使用](#)
 - [18.1 オブジェクト変更通知\(OCN\)について](#)
 - [18.2 問合せ結果変更通知\(QRCN\)について](#)
 - [18.2.1 保証モード](#)
 - [18.2.2 ベストエフォート・モード](#)
 - [18.2.2.1 例: 保証モードのQRCNに対して複雑すぎる問合せ](#)
 - [18.2.2.2 例: 簡略バージョンによってオブジェクトが無効になる問合せ](#)
 - [18.3 通知を生成するイベント](#)
 - [18.3.1 DMLトランザクションのコミット](#)
 - [18.3.2 DDL文のコミット](#)
 - [18.3.3 登録解除](#)
 - [18.3.4 グローバル・イベント](#)
 - [18.4 通知内容](#)
 - [18.5 CQNに適したアプリケーション](#)
 - [18.6 CQN登録の作成](#)
 - [18.7 PL/SQLを使用したCQN登録の作成](#)
 - [18.7.1 PL/SQL CQN登録インタフェース](#)
 - [18.7.2 CQN登録オプション](#)
 - [18.7.2.1 「通知タイプ」オプション](#)
 - [18.7.2.2 QRCNモード\(QRCN通知タイプのみ\)](#)
 - [18.7.2.3 ROWIDオプション](#)
 - [18.7.2.4 操作フィルタ・オプション\(OCN通知タイプのみ\)](#)

- [18.7.2.5 トランザクション・ラグ・オプション\(OCN通知タイプのみ\)](#)
 - [18.7.2.6 通知グループ化オプション](#)
 - [18.7.2.7 信頼可能オプション](#)
 - [18.7.2.8 「通知時にパージ」および「タイムアウト」オプション](#)
- [18.7.3 CQN登録を作成するための前提条件](#)
- [18.7.4 オブジェクト変更通知\(OCN\)に対して登録可能な問合せ](#)
- [18.7.5 問合せ結果変更通知\(QRCN\)に対して登録可能な問合せ](#)
 - [18.7.5.1 保証モードでQRCNに対して登録可能な問合せ](#)
 - [18.7.5.2 ベストエフォート・モードでのみQRCNに対して登録可能な問合せ](#)
 - [18.7.5.3 どちらのモードでもQRCNに対して登録不可能な問合せ](#)
- [18.7.6 PL/SQLを介したCQNに対する問合せの登録](#)
 - [18.7.6.1 PL/SQL通知ハンドラの作成](#)
 - [18.7.6.2 CQ_NOTIFICATION\\$_REG_INFOオブジェクトの作成](#)
 - [18.7.6.3 通知内の各問合せの識別](#)
 - [18.7.6.4 既存の登録への問合せの追加](#)
- [18.7.7 CQN登録のベスト・プラクティス](#)
- [18.7.8 CQN登録のトラブルシューティング](#)
- [18.7.9 登録の削除](#)
- [18.7.10 CQNの構成: 使用例](#)
 - [18.7.10.1 PL/SQL通知ハンドラの作成](#)
 - [18.7.10.2 問合せの登録](#)
- [18.8 OCIを使用したCQN登録の作成](#)
 - [18.8.1 問合せ結果セット通知でのOCI使用](#)
 - [18.8.2 連続問合せ通知の登録のためのOCI使用](#)
 - [18.8.3 クライアントが開始したCQN登録でのOCIの使用](#)
 - [18.8.4 連続問合せ通知のOCIサブスクリプション・ハンドル属性の使用](#)
 - [18.8.5 OCI_ATTR_CQ_QUERYID属性](#)
 - [18.8.6 OCI連続問合せ通知の記述子の使用](#)
 - [18.8.6.1 OCI_DTYPE_CHDES](#)
 - [18.8.6.1.1 OCI_DTYPE_CQDES](#)
 - [18.8.6.1.2 OCI_DTYPE_TABLE_CHDES](#)
 - [18.8.7 OCIサンプル・プログラムでの連続問合せ通知のデモンストレーション](#)
- [18.9 CQN登録の問合せ](#)
- [18.10 通知の解析](#)
 - [18.10.1 CQ_NOTIFICATION\\$_DESCRIPTORオブジェクトの解析](#)
 - [18.10.2 CQ_NOTIFICATION\\$_TABLEオブジェクトの解析](#)
 - [18.10.3 CQ_NOTIFICATION\\$_QUERYオブジェクトの解析](#)
 - [18.10.4 CQ_NOTIFICATION\\$_ROWオブジェクトの解析](#)
- [第IV部 アプリケーション開発者向けの高度なトピック](#)
 - [19 Oracle Flashback Technologyの使用](#)
 - [19.1 Oracle Flashback Technologyの概要](#)
 - [19.1.1 アプリケーション開発機能](#)
 - [19.1.2 データベース管理機能](#)

- [19.2 Oracle Flashback Technologyに関するデータベースの構成](#)
 - [19.2.1 自動UNDO管理に関するデータベースの構成](#)
 - [19.2.2 Oracle Flashback Transaction Queryに関するデータベースの構成](#)
 - [19.2.3 フラッシュバック・トランザクションに関するデータベースの構成](#)
 - [19.2.4 特定のLOB列に対するOracle Flashback操作の有効化](#)
 - [19.2.5 必要な権限の付与](#)
- [19.3 Oracle Flashback Query\(SELECT AS OF\)の使用](#)
 - [19.3.1 例: 過去のデータの検査およびリストア](#)
 - [19.3.2 Oracle Flashback Queryのガイドライン](#)
- [19.4 Oracle Flashback Version Queryの使用](#)
- [19.5 Oracle Flashback Transaction Queryの使用](#)
- [19.6 Oracle Flashback Transaction QueryとOracle Flashback Version Queryの併用](#)
- [19.7 DBMS_FLASHBACKパッケージの使用](#)
- [19.8 フラッシュバック・トランザクションの使用](#)
 - [19.8.1 依存トランザクション](#)
 - [19.8.2 TRANSACTION_BACKOUTパラメータ](#)
 - [19.8.3 TRANSACTION_BACKOUTレポート](#)
 - [19.8.3.1 *_FLASHBACK_TXN_STATE](#)
 - [19.8.3.2 *_FLASHBACK_TXN_REPORT](#)
- [19.9 フラッシュバック・タイム・トラベルの使用](#)
 - [19.9.1 フラッシュバック・アーカイブの作成](#)
 - [19.9.2 フラッシュバック・アーカイブの変更](#)
 - [19.9.3 フラッシュバック・アーカイブの削除](#)
 - [19.9.4 デフォルトのフラッシュバック・アーカイブの指定](#)
 - [19.9.5 フラッシュバック・アーカイブの有効化および無効化](#)
 - [19.9.6 フラッシュバック・アーカイブが有効な表に対するDDL文](#)
 - [19.9.7 フラッシュバック・アーカイブ・データの表示](#)
 - [19.9.8 フラッシュバック・タイム・トラベルのシナリオ](#)
 - [19.9.8.1 シナリオ: フラッシュバック・タイム・トラベルを使用したデジタル・シュレツディングの実行](#)
 - [19.9.8.2 シナリオ: フラッシュバック・タイム・トラベルを使用した履歴データへのアクセス](#)
 - [19.9.8.3 シナリオ: フラッシュバック・タイム・トラベルを使用したレポートの生成](#)
 - [19.9.8.4 シナリオ: フラッシュバック・タイム・トラベルの監査への使用](#)
 - [19.9.8.5 シナリオ: フラッシュバック・タイム・トラベルを使用したデータのリカバリ](#)
- [19.10 Oracle Flashback Technologyの一般ガイドライン](#)
- [19.11 Oracle Flashback Technologyのパフォーマンス・ガイドライン](#)
- [19.12 マルチテナント・コンテナ・データベースでのOracle Flashback Technologyの制限](#)
- [20 プログラミング環境の選択](#)
 - [20.1 アプリケーション・アーキテクチャの概要](#)
 - [20.1.1 クライアント/サーバー・アーキテクチャ](#)
 - [20.1.2 サーバー側のプログラミング](#)

- [20.1.3 2層アーキテクチャと3層アーキテクチャ](#)
- [20.2 プログラム・インタフェースの概要](#)
 - [20.2.1 ユーザー・インタフェース](#)
 - [20.2.2 ステートフル・ユーザー・インタフェースとステートレス・ユーザー・インタフェース](#)
- [20.3 PL/SQLの概要](#)
- [20.4 Oracle Database Javaサポートの概要](#)
 - [20.4.1 Oracle JVMの概要](#)
 - [20.4.2 Oracle JDBCの概要](#)
 - [20.4.2.1 Oracle JDBCドライバ](#)
 - [20.4.2.1.1 JDBC Thinドライバ](#)
 - [20.4.2.1.2 JDBC OCIドライバ](#)
 - [20.4.2.1.3 サーバー側JDBC内部ドライバ](#)
 - [20.4.2.2 JDBC 2.0のサンプル・プログラム](#)
 - [20.4.2.3 JDBC 2.0より前のサンプル・プログラム](#)
 - [20.4.3 Oracle SQLJの概要](#)
 - [20.4.3.1 SQLJのメリット](#)
 - [20.4.4 Oracle JDBCとOracle SQLJの比較](#)
 - [20.4.5 Javaストアド・サブプログラムの概要](#)
 - [20.4.6 Oracle Database Webサービスの概要](#)
- [20.5 PL/SQLまたはJavaの選択](#)
 - [20.5.1 PL/SQLとJavaの類似性](#)
 - [20.5.2 Javaに対するPL/SQLの利点](#)
 - [20.5.3 PL/SQLに対するJavaの利点](#)
- [20.6 プリコンパイラの概要](#)
 - [20.6.1 Pro*C/C++プリコンパイラの概要](#)
 - [20.6.2 Pro*COBOLプリコンパイラの概要](#)
- [20.7 OCIおよびOCCIの概要](#)
 - [20.7.1 OCIおよびOCCIのメリット](#)
 - [20.7.2 OCIおよびOCCI関数](#)
 - [20.7.3 OCIおよびOCCIアプリケーションの手続き型要素および非手続き型要素](#)
 - [20.7.4 OCIアプリケーションまたはOCCIアプリケーションの作成](#)
- [20.8 プリコンパイラおよびOCIの比較](#)
- [20.9 Oracle Data Provider for .NET\(ODP.NET\)の概要](#)
- [20.10 OraOLEDBの概要](#)
- [21 複数のプログラミング言語を使用したアプリケーションの開発](#)
 - [21.1 複数言語プログラムの概要](#)
 - [21.2 外部プロシージャの概要](#)
 - [21.3 外部プロシージャ用のコール仕様の概要](#)
 - [21.4 外部プロシージャのロード](#)
 - [21.4.1 Cプロシージャの定義](#)
 - [21.4.2 環境の設定](#)
 - [21.4.3 DLLの識別](#)
 - [21.4.4 外部プロシージャを公開する](#)

- [21.5 外部プロシージャの公開](#)
 - [21.5.1 Javaクラス・メソッド用のAS LANGUAGE句](#)
 - [21.5.2 外部Cプロシージャ用のAS LANGUAGE句](#)
 - [21.5.2.1 LIBRARY](#)
 - [21.5.2.2 NAME](#)
 - [21.5.2.3 LANGUAGE](#)
 - [21.5.2.4 CALLING STANDARD](#)
 - [21.5.2.5 WITH CONTEXT](#)
 - [21.5.2.6 PARAMETERS](#)
 - [21.5.2.7 AGENT IN](#)
- [21.6 Javaクラス・メソッドの発行](#)
- [21.7 外部Cプロシージャの発行](#)
- [21.8 コール仕様の位置](#)
 - [21.8.1 例: PL/SQLパッケージ内へのコール仕様の配置](#)
 - [21.8.2 例: PL/SQLパッケージ本体内部へのコール仕様の配置](#)
 - [21.8.3 例: ADT仕様内へのコール仕様の配置](#)
 - [21.8.4 例: ADT本体内部へのコール仕様の配置](#)
 - [21.8.5 例: AUTHIDを指定したJava](#)
 - [21.8.6 例: オプションのAUTHIDを指定したC](#)
 - [21.8.7 例: パッケージ内でのコール仕様の混合使用](#)
- [21.9 コール仕様による外部Cプロシージャへのパラメータの受渡し](#)
 - [21.9.1 データ型の指定](#)
 - [21.9.2 外部データ型のマッピング](#)
 - [21.9.3 BY VALUEまたはBY REFERENCEによるパラメータの受渡し](#)
 - [21.9.4 仮パラメータの宣言](#)
 - [21.9.5 デフォルトのデータ型マッピングのオーバーライド](#)
 - [21.9.6 プロパティの指定](#)
 - [21.9.6.1 INDICATOR](#)
 - [21.9.6.2 LENGTHおよびMAXLEN](#)
 - [21.9.6.3 CHARSETIDおよびCHARSETFORM](#)
 - [21.9.6.4 パラメータの再配置](#)
 - [21.9.6.5 SELF](#)
 - [21.9.6.6 BY REFERENCE](#)
 - [21.9.6.7 WITH CONTEXT](#)
 - [21.9.6.8 言語間のパラメータ・モードのマッピング](#)
- [21.10 CALL文による外部プロシージャの実行](#)
 - [21.10.1 外部プロシージャの事前条件](#)
 - [21.10.1.1 外部プロシージャの権限](#)
 - [21.10.1.2 許可の管理](#)
 - [21.10.1.3 外部プロシージャのシノニムの作成](#)
 - [21.10.2 CALL文の構文](#)
 - [21.10.3 Javaクラス・メソッドのコール](#)
 - [21.10.4 外部Cプロシージャのコール](#)

- [21.11 複数言語のプログラム・エラーおよび例外処理](#)
- [21.12 外部Cプロシージャでのサービス・ルーチンの使用](#)
 - [21.12.1 OCIEExtProcAllocCallMemory](#)
 - [21.12.2 OCIEExtProcRaiseExcp](#)
 - [21.12.3 OCIEExtProcRaiseExcpWithMsg](#)
- [21.13 外部Cプロシージャを使用したコールバックの実行](#)
 - [21.13.1 OCIEExtProcGetEnv](#)
 - [21.13.2 OCIコールバック用のオブジェクト・サポート](#)
 - [21.13.3 コールバックに関する制限事項](#)
 - [21.13.4 外部Cプロシージャのデバッグ](#)
 - [21.13.5 例: 外部Cプロシージャのコール](#)
 - [21.13.6 外部Cプロシージャのグローバル変数](#)
 - [21.13.7 外部Cプロシージャの静的変数](#)
 - [21.13.8 外部Cプロシージャに関する制限事項](#)
- [22 Oracle XAを使用したアプリケーションの開発](#)
 - [22.1 X/Open DTP](#)
 - [22.1.1 DTP用語](#)
 - [22.1.2 必須のパブリック情報](#)
 - [22.2 Oracle XAライブラリ・サブプログラム](#)
 - [22.2.1 Oracle XAライブラリ・サブプログラム](#)
 - [22.2.2 Oracle XAインタフェース拡張機能](#)
 - [22.3 XAアプリケーションの開発およびインストール](#)
 - [22.3.1 DBAまたはシステム管理者の責任](#)
 - [22.3.2 アプリケーション開発者の責任](#)
 - [22.3.3 xa_open文字列の定義](#)
 - [22.3.3.1 xa_open文字列の構文](#)
 - [22.3.3.2 xa_open文字列の必須フィールド](#)
 - [22.3.3.3 xa_open文字列のオプション・フィールド](#)
 - [22.3.4 プリコンパイラでのOracle XAの使用](#)
 - [22.3.4.1 デフォルトのデータベースでのプリコンパイラの使用](#)
 - [22.3.4.2 指定されたデータベースでのプリコンパイラの使用](#)
 - [22.3.5 OCIでのOracle XAの使用](#)
 - [22.3.6 Oracle XAを使用したトランザクション制御の管理](#)
 - [22.3.7 プリコンパイラ・アプリケーションの例](#)
 - [22.3.8 プリコンパイラまたはOCIアプリケーションのTPMアプリケーションへの移行](#)
 - [22.3.9 Oracle XAライブラリ・スレッド・セーフティの管理](#)
 - [22.3.9.1 オープン文字列でのスレッドの指定](#)
 - [22.3.9.2 Oracle XAのスレッドにおける制限事項](#)
 - [22.3.10 DBMS_XAパッケージの使用](#)
 - [22.4 XAアプリケーションのトラブルシューティング](#)
 - [22.4.1 Oracle XAトレース・ファイルへのアクセス](#)
 - [22.4.1.1 xa_open文字列DbgFl](#)
 - [22.4.1.2 トレース・ファイルの位置](#)

- [22.4.2 インダウトまたはペンディングのOracle XAトランザクションの管理](#)
 - [22.4.3 SYSアカウント表を使用したOracle XAトランザクションの監視](#)
- [22.5 Oracle XAの問題および制限事項](#)
 - [22.5.1 Oracle XAアプリケーションでのデータベース・リンクの使用](#)
 - [22.5.2 Oracle XAアプリケーションでのトランザクション・ブランチの管理](#)
 - [22.5.3 Oracle Real Application Clusters\(Oracle RAC\)でのOracle XAの使用](#)
 - [22.5.3.1 GLOBAL_TXN_PROCESSES初期化パラメータ](#)
 - [22.5.3.2 Oracle RACでのトランザクション・ブランチの管理](#)
 - [22.5.3.3 Oracle RACでのDTPサービス\(10.2\)を使用したインスタンス・リカバリの管理](#)
 - [22.5.3.4 Oracle RACのXIDのグローバル一意性](#)
 - [22.5.3.5 密結合と疎結合](#)
 - [22.5.4 SQLに基づくOracle XAの制限事項](#)
 - [22.5.4.1 ロールバックおよびコミット](#)
 - [22.5.4.2 DDL文](#)
 - [22.5.4.3 セッション状態](#)
 - [22.5.4.4 EXEC SQL](#)
 - [22.5.5 その他の制限事項](#)
- [23 パブリッシュ・サブスクライブ・モデルでのアプリケーションの開発](#)
 - [23.1 パブリッシュ・サブスクライブ・モデルの概要](#)
 - [23.2 パブリッシュ・サブスクライブのアーキテクチャ](#)
 - [23.2.1 データベース・イベント](#)
 - [23.2.2 Oracle Advanced Queuing](#)
 - [23.2.3 クライアント通知](#)
 - [23.3 パブリッシュ・サブスクライブの概念](#)
 - [23.4 パブリッシュ・サブスクライブ・メカニズムの例](#)
- [24 Oracle ODBC Driverの使用方法](#)
 - [24.1 Oracle ODBC Driverについて](#)
 - [24.2 すべてのユーザー](#)
 - [24.2.1 Oracle ODBC Driver](#)
 - [24.2.1.1 Oracle ODBC Driverとは](#)
 - [24.2.1.2 新機能と変更点](#)
 - [24.2.1.3 サポートされていない機能](#)
 - [24.2.1.4 インストールによって作成されるファイル](#)
 - [24.2.1.5 ドライバの準拠レベル](#)
 - [24.2.1.6 既知の制限事項](#)
 - [24.2.2 構成タスク](#)
 - [24.2.2.1 Oracle Net Servicesの構成](#)
 - [24.2.2.2 データソースの構成](#)
 - [24.2.2.3 「Oracle ODBCドライバ構成」ダイアログ・ボックス](#)
 - [24.2.3 oraodbc.iniファイルの変更](#)
 - [24.2.3.1 ロック・タイムアウトの短縮](#)
 - [24.2.4 データソースへの接続](#)

- [24.2.4.1 Oracleデータソースへの接続](#)
 - [24.2.5 トラブルシューティング](#)
 - [24.2.5.1 Oracle ODBC Driverの最初の使用について](#)
 - [24.2.5.2 パスワードの期限切れ](#)
- [24.3 上級ユーザー](#)
 - [24.3.1 Oracle ODBC DriverのTNSサービス名の作成](#)
 - [24.3.2 SQL文](#)
 - [24.3.3 データ型](#)
 - [24.3.4 データ型の実装](#)
 - [24.3.5 データ型に関する制限事項](#)
 - [24.3.6 エラー・メッセージ](#)
- [24.4 プログラム](#)
 - [24.4.1 接続文字列のフォーマット](#)
 - [24.4.2 SQLDriverConnectの実装](#)
 - [24.4.3 プログラムでのロック・タイムアウトの削減](#)
 - [24.4.4 odbc32.lib \(Windows\)またはlibodbc.so \(UNIX\)とのリンク](#)
 - [24.4.5 ROWIDの情報](#)
 - [24.4.6 WHERE句でのROWID](#)
 - [24.4.7 結果セットの有効化](#)
 - [24.4.8 EXEC構文の有効化](#)
 - [24.4.9 Oracle RAC環境における接続障害のイベント通知の有効化](#)
 - [24.4.10 ODBCを通じた暗黙的結果機能の使用](#)
 - [24.4.11 ODBCでのOracle TIMESTAMP WITH TIME ZONEおよびTIMESTAMP WITH LOCAL TIME ZONE列タイプのサポートについて](#)
 - [24.4.12 Oracleクライアント\(OCI、SQL*Plus、Oracle ODBC Driverなど\)でORA_SDTZを設定した場合の影響について](#)
 - [24.4.13 サポートされている機能](#)
 - [24.4.13.1 APIへの準拠](#)
 - [24.4.13.2 ODBC API関数の実装](#)
 - [24.4.13.3 ODBC SQL構文の実装](#)
 - [24.4.13.4 データ型の実装\(プログラミング\)](#)
 - [24.4.14 Unicodeのサポート](#)
 - [24.4.14.1 ODBC環境内でのUnicodeのサポート](#)
 - [24.4.14.2 ODBC APIでのUnicodeのサポート](#)
 - [24.4.14.3 ドライバ・マネージャのUnicode機能](#)
 - [24.4.14.4 SQLGetDataのパフォーマンス](#)
 - [24.4.14.5 Unicodeの例](#)
 - [24.4.15 パフォーマンスとチューニング](#)
 - [24.4.15.1 一般的なODBCプログラミングのヒント](#)
 - [24.4.15.2 データソース構成オプション](#)
 - [24.4.15.3 DATEおよびTIMESTAMPデータ型](#)
- [25 識別コード・パッケージの使用](#)
 - [25.1 識別の概念](#)

- [25.2 識別コード・パッケージの概要](#)
- [25.3 識別コード・パッケージの使用](#)
 - [25.3.1 MGD_ID ADTを使用したOracle DatabaseへのRFIDタグの格納](#)
 - [25.3.1.1 MGD_ID列型を使用した表の作成と、列へのEPCタグ・エンコーディングの格納](#)
 - [25.3.1.2 RFIDタグを表現するMGD_IDオブジェクトの構成](#)
 - [25.3.1.2.1 MGD_IDオブジェクト\(SGTIN-64\)の構成、およびカテゴリIDとコンポーネントのリストの受渡し](#)
 - [25.3.1.2.2 MGD_IDオブジェクト\(SGTIN-64\)の構成、およびカテゴリID、タグ識別子と追加の必須パラメータのリストの受渡し](#)
 - [25.3.1.2.3 MGD_IDオブジェクト\(SGTIN-64\)の構成、およびカテゴリ名、カテゴリ・バージョン\(NULLの場合、最新のバージョンを使用\)とコンポーネントのリストの受渡し](#)
 - [25.3.1.2.4 MGD_IDオブジェクト\(SGTIN-64\)の構成、およびカテゴリ名、カテゴリ・バージョン、タグ識別子と追加の必須パラメータのリストの受渡し](#)
 - [25.3.1.3 データベース表へのMGD_IDオブジェクトの挿入](#)
 - [25.3.1.4 MGD_ID列型の問合せ](#)
 - [25.3.2 MGD_ID列型のメンバー関数を使用したファンクション索引の構築](#)
 - [25.3.3 MGD_ID ADT関数の使用](#)
 - [25.3.3.1 MGD_IDオブジェクトとget_component関数の使用](#)
 - [25.3.3.2 標準的な表現からのタグ・データの解析](#)
 - [25.3.3.3 フィールドからのタグ表現の再構成](#)
 - [25.3.3.4 タグ表現間の翻訳](#)
 - [25.3.4 識別コードのカテゴリの定義と既存のカテゴリへのコード体系の追加](#)
 - [25.3.4.1 識別コードのカテゴリの作成](#)
 - [25.3.4.2 新規作成したカテゴリへの2つのメタデータ・スキームの追加](#)
- [25.4 識別コード・パッケージ・タイプ](#)
- [25.5 DBMS_MGD_ID_UTLパッケージ](#)
- [25.6 識別コード・メタデータ表およびビュー](#)
- [25.7 電子製品コード\(EPC\)の概念](#)
 - [25.7.1 RFIDテクノロジーとEPCバージョン1.1コード体系](#)
 - [25.7.2 製品コードの概念と現在の使用](#)
 - [25.7.2.1 電子製品コード\(EPC\)](#)
 - [25.7.2.1.1 EPCの純粋な識別](#)
 - [25.7.2.1.2 EPCエンコーディング](#)
 - [25.7.2.1.3 EPCタグのビットレベルのエンコーディング](#)
 - [25.7.2.1.4 EPC識別URI](#)
 - [25.7.2.1.5 EPCタグのURIエンコーディング](#)
 - [25.7.2.1.6 EPCエンコーディング・プロシージャ](#)
 - [25.7.2.1.7 EPCデコーディング・プロシージャ](#)
 - [25.7.2.2 Global Trade Identification Number \(GTIN\)およびSerializable Global Trade Identification Number \(SGTIN\)](#)

- [25.7.2.3 Serial Shipping Container Code \(SSCC\)](#)
 - [25.7.2.4 Global Location Number \(GLN\)およびSerializable Global Location Number \(SGLN\)](#)
 - [25.7.2.5 Global Returnable Asset Identifier \(GRAI\)](#)
 - [25.7.2.6 Global Individual Asset Identifier \(GIAI\)](#)
 - [25.7.2.7 RFID EPCネットワーク](#)
- [25.8 Oracle Database Tag Data Translationスキーマ](#)
- [26 スキーマ・オブジェクトの依存性の理解](#)
 - [26.1 スキーマ・オブジェクトの依存性の概要](#)
 - [26.1.1 例: 依存オブジェクト・タイプと参照オブジェクト・タイプの表示](#)
 - [26.1.2 例: 一部の依存オブジェクトを無効にするスキーマ・オブジェクトの変更](#)
 - [26.1.3 例: 複数のオブジェクトに依存するビュー](#)
 - [26.2 オブジェクトの依存性の問合せ](#)
 - [26.3 オブジェクトのステータス](#)
 - [26.4 依存オブジェクトの無効化](#)
 - [26.4.1 セッションの状態と参照パッケージ](#)
 - [26.4.2 セキュリティ認可](#)
 - [26.5 無効化を回避するためのガイドライン](#)
 - [26.5.1 パッケージの最後へのアイテムの追加](#)
 - [26.5.2 ビューを介した各表の参照](#)
 - [26.6 オブジェクトの再有効化](#)
 - [26.6.1 コンパイルされたがエラーがあるオブジェクトの再有効化](#)
 - [26.6.2 未許可オブジェクトの再有効化](#)
 - [26.6.3 無効SQLオブジェクトの再有効化](#)
 - [26.6.4 無効PL/SQLオブジェクトの再有効化](#)
 - [26.7 スキーマの範囲内での名前解決](#)
 - [26.8 ローカル依存性の管理](#)
 - [26.9 リモート依存性の管理](#)
 - [26.9.1 ローカル・データベース・プロシージャとリモート・データベース・プロシージャの間の依存性](#)
 - [26.9.2 その他のリモート・オブジェクト間の依存性](#)
 - [26.9.3 アプリケーションの依存性](#)
 - [26.10 リモート・プロシージャ・コール\(RPC\)の依存性管理](#)
 - [26.10.1 タイムスタンプ依存性モード](#)
 - [26.10.2 RPCシグネチャ依存性モード](#)
 - [26.10.2.1 パラメータの名前とデフォルト値の変更](#)
 - [26.10.2.2 パラメータ・モードINの指定の変更](#)
 - [26.10.2.3 サブプログラム本体の変更](#)
 - [26.10.2.4 パラメータのデータ型クラスの変更](#)
 - [26.10.2.5 パッケージ・タイプの変更](#)
 - [26.10.3 依存性モードの制御](#)
 - [26.10.3.1 依存性の解決](#)
 - [26.10.3.2 依存性管理の提案](#)

- [26.11 共有SQLの依存性管理](#)
- [27 エディションベースの再定義の使用](#)
 - [27.1 エディションベースの再定義の概要](#)
 - [27.2 エディション](#)
 - [27.2.1 エディション・オブジェクトと非エディション・オブジェクト](#)
 - [27.2.1.1 エディション・オブジェクトと非エディション・オブジェクトの名前解決](#)
 - [27.2.1.2 エディション・オブジェクトに依存可能な非エディション・オブジェクト](#)
 - [27.2.1.2.1 マテリアライズド・ビュー](#)
 - [27.2.1.2.2 仮想列](#)
 - [27.2.1.3 エディション対応、およびエディション非対応スキーマ・オブジェクト・タイプ](#)
 - [27.2.1.4 ユーザーに対するエディションの有効化](#)
 - [27.2.1.4.1 エディション化されていない依存を持つ潜在的なエディション・オブジェクト](#)
 - [27.2.1.4.2 エディションを有効化できないユーザー](#)
 - [27.2.1.5 EDITIONABLEおよびNONEDITIONABLEプロパティ](#)
 - [27.2.1.5.1 新規EDITIONABLEおよびNONEDITIONABLEオブジェクトの作成](#)
 - [27.2.1.5.2 EDITIONABLEおよびNONEDITIONABLEオブジェクトの置換および変更](#)
 - [27.2.1.6 エディション・オブジェクトの規則](#)
 - [27.2.2 エディションの作成](#)
 - [27.2.3 エディション・オブジェクトおよび変更時のコピー](#)
 - [27.2.3.1 例: エディション・オブジェクトおよび変更時のコピー](#)
 - [27.2.3.2 例: エディション・オブジェクトの削除](#)
 - [27.2.3.3 例: 削除済継承オブジェクトの名前を使用したオブジェクトの作成](#)
 - [27.2.4 一部のユーザーに対してエディションを使用可能にする](#)
 - [27.2.5 すべてのユーザーに対してエディションを使用可能にする](#)
 - [27.2.6 現行エディションとセッション・エディション](#)
 - [27.2.6.1 最初のセッション・エディション](#)
 - [27.2.6.2 セッション・エディションの変更](#)
 - [27.2.6.3 現行エディション名とセッション・エディション名の表示](#)
 - [27.2.6.4 現行エディションがセッション・エディションと異なる状況](#)
 - [27.2.7 エディションのリタイア](#)
 - [27.2.8 エディションの削除](#)
 - [27.3 エディションング・ビュー](#)
 - [27.3.1 エディションング・ビューの作成](#)
 - [27.3.2 パーティション拡張エディションング・ビュー名](#)
 - [27.3.3 エディションング・ビューの書き込み可能性の変更](#)
 - [27.3.4 エディションング・ビューの置換](#)
 - [27.3.5 削除または名前変更された実表](#)
 - [27.3.6 実表への索引と制約の追加](#)
 - [27.3.7 SQLオブティマイザ索引ヒント](#)

- [27.4 crosseditionトリガー](#)
 - [27.4.1 forward crosseditionトリガー](#)
 - [27.4.2 reverse crosseditionトリガー](#)
 - [27.4.3 crosseditionトリガーとエディションの相互作用](#)
 - [27.4.3.1 表示可能なトリガー](#)
 - [27.4.3.2 起動できるトリガーの種類](#)
 - [27.4.3.2.1 forward crosseditionトリガー-SQL](#)
 - [27.4.3.2.2 reverse crosseditionトリガー-SQL](#)
 - [27.4.3.2.3 アプリケーションSQL](#)
 - [27.4.3.3 起動順序](#)
 - [27.4.3.3.1 FOLLOWS句およびPRECEDES句](#)
 - [27.4.3.3.2 トリガーの種類とエディション](#)
 - [27.4.3.4 crosseditionトリガーの実行](#)
 - [27.4.4 crosseditionトリガーの作成](#)
 - [27.4.4.1 forward crosseditionトリガー本体のコーディング](#)
 - [27.4.4.1.1 データ変換における衝突の処理](#)
 - [27.4.4.1.2 他の表への変更の処理](#)
 - [27.4.5 アップグレード前の表現をアップグレード後の表現に変換](#)
 - [27.4.5.1 更新内容の消失の防止](#)
 - [27.4.6 crosseditionトリガーの削除](#)
- [27.5 EBR機能に関する情報の表示](#)
 - [27.5.1 エディションに関する情報の表示](#)
 - [27.5.2 エディショニング・ビューに関する情報の表示](#)
 - [27.5.3 crosseditionトリガーに関する情報の表示](#)
- [27.6 アプリケーションをアップグレードするためのEBRの使用](#)
 - [27.6.1 アプリケーションでのエディショニング・ビューの使用準備](#)
 - [27.6.2 エディションのみを使用するEBRの手順](#)
 - [27.6.3 エディショニング・ビューを使用するEBRの手順](#)
 - [27.6.4 crosseditionトリガーを使用するEBRの手順](#)
 - [27.6.5 アプリケーション・アップグレードのロールバック](#)
 - [27.6.6 使用されていない表の列が占める領域の再利用](#)
 - [27.6.7 例: アプリケーションをアップグレードするためのEBRの使用](#)
 - [27.6.7.1 既存のアプリケーション](#)
 - [27.6.7.1.1 例: 既存アプリケーションが作成された方法](#)
 - [27.6.7.1.2 例: 既存表のデータ表示](#)
 - [27.6.7.2 アプリケーションでのエディショニング・ビューの使用準備](#)
 - [27.6.7.3 サンプル・アプリケーションをアップグレードするためのEBRの使用](#)
 - [27.6.7.3.1 例: サンプル・アプリケーションをアップグレードするエディションの作成](#)
 - [27.6.7.3.2 例: 表の変更とエディショニング・ビューの置換](#)
 - [27.6.7.3.3 例: crosseditionトリガーの作成と有効化](#)
 - [27.6.7.3.4 例: トランスフォームの適用](#)
 - [27.6.7.3.5 例: 変更された表のデータ表示](#)

- [28 トランザクション・ガードの使用](#)
 - [28.1 トランザクション・ガードが解決する問題](#)
 - [28.2 トランザクション・ガードが提供するソリューション](#)
 - [28.3 トランザクション・ガードの概念および範囲](#)
 - [28.3.1 論理トランザクション識別子\(LTXID\)](#)
 - [28.3.2 最大1回の実行](#)
 - [28.3.3 トランザクション・ガードのカバレッジ](#)
 - [28.3.4 トランザクション・ガードとXAトランザクション](#)
 - [28.3.5 トランザクション・ガードの除外](#)
 - [28.4 トランザクション・ガードのデータベース構成](#)
 - [28.4.1 構成チェックリスト](#)
 - [28.4.2 トランザクション履歴表](#)
 - [28.4.3 サービス・パラメータ](#)
 - [28.4.3.1 例: サーバー・プール用のサービスの追加および変更](#)
 - [28.4.3.2 例: 管理者管理サービスの追加](#)
 - [28.4.3.3 例: サービスの変更\(PL/SQL\)](#)
 - [28.5 トランザクション・ガードを使用するアプリケーションの開発](#)
 - [28.5.1 トランザクション・ガードの標準使用](#)
 - [28.5.2 LTXIDの使用の詳細](#)
 - [28.5.3 トランザクション・ガードと透過的アプリケーション・フェイルオーバー](#)
 - [28.5.4 ODP.NETでのトランザクション・ガードの使用](#)
 - [28.5.5 接続プールLTXIDの使用](#)
 - [28.5.6 XA 1フェーズ最適化のコミット結果の向上](#)
 - [28.5.7 トランザクション・ガード開発の追加要件](#)
 - [28.6 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との関係](#)

表一覧

- [3-1 表注釈の結果キャッシュ・モード](#)
- [3-2 有効な結果キャッシュ表モード](#)
- [3-3 クライアント構成パラメータ\(オプション\)](#)
- [3-4 クライアント結果キャッシュとサーバー結果キャッシュの設定](#)
- [3-5 セッション純正值および接続クラスのデフォルト](#)
- [8-1 COMMIT文のオプション](#)
- [8-2 明示的なロックにおける同時実行性の例](#)
- [8-3 ロック情報の表示方法](#)
- [8-4 ANSI/ISO SQL分離レベルおよび可能なトランザクションの相互作用](#)
- [8-5 Oracle Databaseで用意されるANSI/ISO SQL分離レベル](#)
- [8-6 コミット読み込みトランザクションとシリアライズ可能トランザクションの比較](#)
- [8-7 トランザクションの結果](#)
- [9-1 SQL文字データ型](#)
- [9-2 浮動小数点データ型の範囲および精度](#)
- [9-3 浮動小数点2進形式の構成要素](#)
- [9-4 2進形式の記憶域パラメータの概要](#)
- [9-5 ネイティブ浮動小数点形式の特殊な数値](#)
- [9-6 例外により生成される値](#)
- [9-7 SQL日時データ型](#)
- [9-8 日時データ型のSQL変換関数](#)
- [9-9 ラージ・オブジェクト\(LOB\)](#)
- [9-10 SQL関数の表示タイプ](#)
- [9-11 SQLデータ型ファミリー](#)
- [10-1 Oracle SQLパターン一致条件および関数](#)
- [10-2 条件および関数のOracle SQL パターン一致オプション](#)
- [10-3 Oracle SQL正規表現内のPOSIX演算子](#)
- [10-4 POSIX演算子と多言語演算子の関係](#)
- [10-5 Oracle SQL正規表現でPERLの影響を受ける演算子](#)
- [10-6 正規表現要素の説明](#)
- [10-7 正規表現要素の説明](#)
- [14-1 PL/Scopeが収集する識別子の型](#)
- [14-2 PL/Scopeがレポートする使用方法](#)
- [15-1 RAWプロファイラ出力ファイルのインジケータ](#)
- [15-2 PL/SQL階層プロファイラによって追跡される操作の関数名](#)
- [15-3 PL/SQL階層プロファイラ・データベース表](#)
- [15-4 DBMSHP_RUNS表の列](#)
- [15-5 DBMSHP_FUNCTION_INFO表の列](#)
- [15-6 DBMSHP_PARENT_CHILD_INFO表の列](#)
- [16-1 DBMSPCC_RUNS表の列](#)
- [16-2 DBMSPCC_UNITS表の列](#)
- [16-3 DBMSPCC_BLOCKS表の列](#)

- [17-1 PL/SQL Web Toolkitで一般的に使用されるパッケージ](#)
- [17-2 mod_plsql DAD属性と埋込みPL/SQLゲートウェイDAD属性間のマッピング](#)
- [17-3 mod_plsqlグローバル属性と埋込みPL/SQLゲートウェイ・グローバル属性間のマッピング](#)
- [17-4 DADの認証の可能性](#)
- [18-1 連続問合せ通知登録オプション](#)
- [18-2 CQ_NOTIFICATION\\$_REG_INFOの属性](#)
- [18-3 サービスのクオリティ・フラグ](#)
- [18-4 CQ_NOTIFICATION\\$_DESCRIPTORの属性](#)
- [18-5 CQ_NOTIFICATION\\$_TABLEの属性](#)
- [18-6 CQ_NOTIFICATION\\$_QUERYの属性](#)
- [18-7 CQ_NOTIFICATION\\$_ROWの属性](#)
- [19-1 Oracle Flashback Version Queryにおける行データ疑似列](#)
- [19-2 フラッシュバックTRANSACTION_BACKOUTオプション](#)
- [19-3 フラッシュバック・アーカイブ・ファイルの静的データ・ディクショナリ・ビュー](#)
- [20-1 PL/SQLパッケージおよび相当するJavaでの操作](#)
- [21-1 パラメータ・データ型のマッピング](#)
- [21-2 外部データ型のマッピング](#)
- [21-3 プロパティおよびデータ型](#)
- [22-1 Oracle Databaseが発行する必要があるXA機能](#)
- [22-2 XAライブラリ・サブプログラム](#)
- [22-3 Oracle XAインタフェース拡張機能](#)
- [22-4 xa_open文字列の必須フィールド](#)
- [22-5 xa_open文字列のオプション・フィールド](#)
- [22-6 TXインタフェース関数](#)
- [22-7 TPM用に置き換える文](#)
- [22-8 サンプル・トレース・ファイルの内容](#)
- [22-9 密結合および疎結合トランザクション・ブランチ](#)
- [24-1 Oracle ODBC Driverキットでインストールされるファイル](#)
- [24-2 Oracle ODBC DriverおよびOracle Databaseデータ型に関する制限事項](#)
- [24-3 Oracle ODBC Driverが返す接頭辞のエラー・メッセージ値](#)
- [24-4 SQLDriverConnect関数コールの接続文字列引数に組み込むことができるキーワード](#)
- [24-5 SQLDriverConnect接続文字列に必要なキーワード](#)
- [24-6 Oracle ODBC Driverで特定の関数を実装する方法](#)
- [24-7 サポートされるSQLデータ型および対応するODBC SQLデータ型](#)
- [25-1 EPCエンコーディングの一般的な構造](#)
- [25-2 識別コード・パッケージADT](#)
- [25-3 MGD_ID ADTサブプログラム](#)
- [25-4 DBMS_MGD_ID_UTLパッケージ・ユーティリティ・サブプログラム](#)
- [25-5 MGD_ID_CATEGORYメタデータ・ビューの定義と説明](#)
- [25-6 USER_MGD_ID_CATEGORYメタデータ・ビューの定義と説明](#)
- [25-7 MGD_ID_SCHEMEメタデータ・ビューの定義と説明](#)
- [25-8 USER_MGD_ID_SCHEMEメタデータ・ビューの定義と説明](#)
- [26-1 データベース・オブジェクトのステータス](#)

- [26-2 高度な無効化が発生する操作](#)
- [26-3 データ型クラス](#)
- [27-1 エ디션情報を含む*_ディクショナリ・ビュー](#)
- [27-2 エディショニング・ビュー情報を含む*_ディクショナリ・ビュー](#)
- [28-1 LTXIDの状態または状況、アプリケーション・アクション、および次に使用するLTXID](#)
- [28-2 トランザクション・マネージャの状態/状況およびアクション](#)

はじめに

『Oracle Database開発ガイド』では、データベースおよびデータベース・アプリケーションの開発経験者に役立つトピックについて説明します。このマニュアルの内容は、サポートされるすべてのプラットフォーム上で同様に動作する機能に適用されるもので、システム固有の情報は含みません。

内容は次のとおりです。

対象読者

このマニュアルは主に、アプリケーションの開発や、Oracle Database環境で実行できるようにアプリケーションの変換を行うアプリケーション開発者向けです。このマニュアルは、システム・アナリストやプロジェクト・マネージャなど、データベースまたはデータベース・アプリケーションの開発に関心があるすべての方にも役立ちます。

このマニュアルでは、[『Oracle Database 2日で開発者ガイド』](#)や[『Oracle Database 2日でデータベース管理者』](#)など、各自の職務に該当する「2日」シリーズのガイドで、概念や技法を把握していることが前提となります。このマニュアルを効果的に使用するには、次のものに関する実用的な知識が必要です。

- Structured Query Language(SQL)
- オブジェクト指向プログラミング

ドキュメントのアクセシビリティ

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

Oracle Supportへのアクセス

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

関連ドキュメント

詳細は、Oracle Database 12cリリース2 (12.2)ドキュメント・セットの次のドキュメントを参照してください。

- [Oracle Database PL/SQL言語リファレンス](#)
- [Oracle Call Interfaceプログラマーズ・ガイド](#)
- [Oracle Databaseセキュリティ・ガイド](#)
- [Pro*C/C++プログラマーズ・ガイド](#)
- [Oracle Database SQL言語リファレンス](#)
- [Oracle Database管理者ガイド](#)
- [Oracle Database概要](#)
- [Oracle Database Java開発者ガイド](#)
- [Oracle Database JDBC開発者ガイド](#)

- [Oracle XML Developer's Kitプログラマーズ・ガイド](#)
- [Oracle XML DB開発者ガイド](#)
- [Oracle Database移行ガイド](#)
- [Oracle Databaseグローバル化・サポート・ガイド](#)
- [Oracle Databaseサンプル・スキーマ](#)

関連項目:

- 『Oracle PL/SQL Programming』、Steven Feuerstein、Bill Pribyl著。Sixth Edition。O'Reilly & Associates、2014年。
- 『Oracle PL/SQL Best Practices』: Steven Feuerstein著、Second Edition。O'Reilly & Associates、2007年。

表記規則

このマニュアルでは次の表記規則を使用します。

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

付記:

- *_viewは、名前がviewで終わるすべての静的データ・ディクショナリ・ビューを示します。たとえば、*_ERRORSは、ALL_ERRORS、DBA_ERRORSおよびUSER_ERRORSを示します。静的データ・ディクショナリ・ビューの詳細または概要は、[『Oracle Databaseリファレンス』](#)を参照してください。
- スキーマ名で修飾されていない表名は、サンプル・スキーマHRにあります。サンプル・スキーマの詳細は、[『Oracle Databaseサンプル・スキーマ』](#)を参照してください。

Oracle Databaseリリース19cにおける変更点

Oracle Databaseリリース19cの『Oracle Databaseデータベース開発ガイド』における変更点は次のとおりです。

新機能

『Oracle Databaseデータベース開発ガイド』のこのリリースでの新機能。

変更問合せ通知の新しいOCI登録モード

アプリケーションでは、クライアントが開始した接続モードを使用して、変更問合せ通知(CQN)を登録および受信できます。

Oracle Databaseリリース19c, バージョン19.1以降では、クラウドのアプリケーションで動作するように新しいOCI登録モードが設計されました。この新しいモードは、オンプレミスで実行されているアプリケーションでも動作します。この通知配信モードでは、クライアント・アプリケーションは通知を受信するためにOracleデータベース・サーバーへの接続を開始します。クライアントが開始した接続は、特別なネットワーク構成を必要とせず、使いやすくセキュアです。

詳細は、[「クライアントが開始したCQN登録でのOCIの使用」](#)を参照してください

非推奨となった機能

この項では、『データベース開発ガイド』に含まれていた非推奨となった機能について説明します。

Oracle Database 19cで非推奨となった機能

Oracle Fail Safeは非推奨になりました。今後のリリースではサポート対象外になり、使用できなくなる可能性があります。

関連項目:

詳細は、[『Oracle Databaseアップグレード・ガイド』](#)を参照してください。

ネイティブ暗号化のセキュリティ更新

Oracleは、Oracle Databaseリリース11.2以降における、ネイティブ・ネットワーク暗号化環境に影響を与える必要なセキュリティ機能強化に対応するために、ダウンロード可能なパッチを提供しています。

このパッチは、My Oracle Supportノート[2118136.2](#)で入手できます。

改善されたサポート対象アルゴリズムは次のとおりです。

- 暗号化アルゴリズム: AES128、AES192およびAES256
- チェックサム・アルゴリズム: SHA1、SHA256、SHA384およびSHA512

非推奨であり、使用すべきでないアルゴリズムは次のとおりです。

- 暗号化アルゴリズム: DES、DES40、3DES112、3DES168、RC4_40、RC4_56、RC4_128およびRC4_256
- チェックサム・アルゴリズム: MD5

サイトでネットワーク・ネイティブ暗号化を使用する必要がある場合は、My Oracle Supportノート[2118136.2](#)で説明されているパッチをダウンロードする必要があります。ご使用のOracle Databaseインストールの円滑な移行を可能にするために、このパッチでは、より弱いアルゴリズムを無効にしより強力なアルゴリズムの使用を開始できる、2つのパラメータが提供されます。こ

のパッチは、サーバー上とクライアント上の両方のOracle Databaseインストールにインストールする必要があります。

ネットワーク・ネイティブ暗号化の代替となるのは、TLS (トランスポート層セキュリティ)であり、中間者攻撃からの保護を実現します。

関連項目:

- [Oracle Databaseセキュリティ・ガイド](#)のネイティブ・ネットワーク暗号化とトランスポート層セキュリティの間の選択
- [Oracle Databaseセキュリティ・ガイド](#)のネイティブ・ネットワーク暗号化のセキュリティの向上

サポート対象外機能

Oracle Database 19cでサポート対象外となった機能

Oracle Multimediaは全面的にサポート対象外となりました。SecureFiles LOBにマルチメディア・コンテンツを格納し、サード・パーティ製品を使用することをお勧めします。

関連項目:

詳細は、[『Oracle Databaseアップグレード・ガイド』](#)を参照してください。

第I部 データベース開発の基礎

この部では、Oracle Databaseとデータベース・アプリケーションの開発者に必要な基本情報を示します。この部の章では、主に概要レベルで概念を説明しており、詳細な機能の説明や具体的な実装については、他の章や他のマニュアルへの参照を示しています。

関連リンク:



- Oracle Database の概念と技術の詳細は、[『Oracle Database 2 日で開発者ガイド』](#)および [『Oracle Database 2 日でデータベース管理者』](#)を参照してください。
- Tom Kyte による Oracle Database の概念に関するガイドと記事については、<http://asktom.oracle.com> を参照してください

章:

- [設計の基本](#)
- [データベース・アプリケーションの接続方法](#)
- [パフォーマンスとスケーラビリティ](#)
- [Oracle Real-World Performanceのアプリケーションの設計](#)
- [セキュリティ](#)
- [高可用性](#)
- [高度なPL/SQL機能](#)

1 設計の基本

この章では、データベース開発者にとって重要な設計上の目標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：

- [パフォーマンスを考慮した設計](#)
- [スケーラビリティを考慮した設計](#)
- [拡張性を考慮した設計](#)
- [セキュリティを考慮した設計](#)
- [可用性を考慮した設計](#)
- [移植性を考慮した設計](#)
- [診断性を考慮した設計](#)
- [特殊な環境を考慮した設計](#)
- [特殊なシナリオのための機能](#)

関連項目：

- 高レベルのデータベースの概念については、[「データベース開発の基礎」](#)を参照してください
- Oracle Databaseの概念と技術の詳細は、[『Oracle Database 2日で開発者ガイド』](#)および[『Oracle Database 2日でデータベース管理者』](#)を参照してください

1.1 パフォーマンスを考慮した設計

データベースとアプリケーションのパフォーマンスにとってキーとなるのは、チューニングではなく設計です。チューニングは非常に重要ですが、不適切な設計を補うことはできません。設計を行うには、まず効率的なデータ・モデル、適切に定義されたパフォーマンス目標とメトリック、および適切なベンチマーク戦略から開始する必要があります。そうしないと、設計が適切であれば達成できたはずの結果を、最大限のチューニングを行っても得られない場合、実装中に問題が発生します。当初の設計が不適切であれば、チューニングを実行しても後で再設計が必要になる場合があります。

関連項目：

- [パフォーマンスとスケーラビリティ](#)
- [Oracle Databaseパフォーマンス・チューニング・ガイド](#)
- [Oracle Database SQLチューニング・ガイド](#)

1.2 スケーラビリティを考慮した設計

スケーラビリティとは、負荷の増大に応じて適切に動作するシステムの能力です。負荷とは、データ量、ユーザー数およびその他の関連する要因の組合せです。スケーラビリティを考慮した設計を行うには、有効なベンチマーク戦略、適切なアプリケーション開発技法(バインド変数など)、および共有サーバー接続やクラスタリング、パーティショニング、パラレル操作など適切なOracle

Databaseアーキテクチャ機能を使用する必要があります。

関連項目:

- [パフォーマンスとスケーラビリティ](#)
- [Oracle Database 2日で開発者ガイド](#)

1.3 拡張性を考慮した設計

拡張性とは、データベースやデータベース・アプリケーションが将来的な拡張に対応できる能力のことです。データベースまたはアプリケーションの拡張性が高いほど、既存の機能に対する影響を最小限にとどめて機能を追加または変更することが容易になります。

ノート:

拡張性は上位互換性とは異なります。上位互換性とは、あるアプリケーションが、同じアプリケーションの新しいバージョンで作成されたデータを読み込むことはできるが、本来読み込めるように設計されたデータのみを使用する機能です。



たとえば、旧バージョンのアプリケーションはテキストのみを処理し、新しいバージョンの同じアプリケーションはテキストとグラフィックの両方を処理すると仮定します。旧バージョンがテキストとグラフィックの両方に対応し、グラフィックを無視してテキストを処理する場合、これは上位互換です。旧バージョンをアップグレードしてテキストとグラフィックの両方を処理できる場合、これは拡張可能です。このようなアプリケーションのアップグレードが容易であるほど、拡張性が高いことになります。

最大限の拡張性を達成するには、インフラストラクチャを大きく変更せずに強化できるメカニズムを含めることによって、データベースやアプリケーションの設計として拡張性を組み込む必要があります。旧バージョンのデータベースまたはアプリケーションではこのメカニズムが使用されていない場合もあり、どのバージョンでもまったく使用されていない可能性もありますが、メンテナンスを容易にし、短期間で廃止されないようにするには、上位互換性は必須です。

トピック:

- [データ・カートリッジ](#)
- [外部プロシージャ](#)
- [ユーザー定義ファンクションと集計ファンクション](#)
- [オブジェクト・リレーショナル機能](#)

1.3.1 データ・カートリッジ

データ・カートリッジにより、Oracle Extensibility Architectureフレームワークを利用してOracle Databaseサーバーの機能が拡張されます。このフレームワークでは、ユーザー定義データ型の特化されたデータやドメイン固有のデータに関連したビジネス・ロジックおよびプロセスを取得できます。追加の属性を必要とせずに新規の動作を提供するデータ・カートリッジは、ユーザー定義のデータ型ではなくパッケージを使用してそれを実現します。ユーザー定義のデータ型でもパッケージでも、サーバーにおけるアプリケーション・データの解析、格納、取得および索引付けの方法をユーザーが決定することになります。この機能はデータ・

カートリッジにパッケージされ、サーバーにプラグインして機能を新規ドメインへと拡張するソフトウェア・コンポーネントが作成されるため、データベース自体が拡張可能になります。

拡張可能データベース管理システムの索引付けおよび問合せ最適化メカニズムをカスタマイズし、ユーザー定義のビジネス・オブジェクトおよび様々な型向けに特化されたサービスやより効率的な処理を提供できます。拡張可能インタフェースを介して実装をサーバーに登録するときに、固有のデフォルト・プロセスのかわりにカスタマイズされた処理指示を実装するようにサーバーに対して指示します。

関連トピック

- [Oracle Databaseデータ・カートリッジ開発者ガイド](#)

1.3.2 外部プロシージャ

外部プロシージャは、呼出し側に影響することなく機能を強化できるため、高い拡張性を備えています。外部プロシージャでは、呼出しに必要なあらゆる情報を含むコール指定が、実装の詳細を含むプロシージャの本体とは分かれているためです。本体のみを変更してコール指定を変更しなければ、呼出し側には影響がありません。

関連トピック

- [複数のプログラミング言語を使用したアプリケーションの開発](#)

1.3.3 ユーザー定義ファンクションと集計ファンクション

ユーザー定義のPL/SQLファンクションをSQL文または式として使用すると、SQLの機能を拡張できます。

ユーザー定義集計ファンクションは、Oracle Extensibility Architectureフレームワークの一部です。

関連項目:

- SQL文および式でユーザー定義のPL/SQLファンクションを呼び出す方法については、[「SQL文からのストアPL/SQLファンクションの起動」](#)を参照してください
- ユーザー定義集計ファンクションの詳細は、『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

1.3.4 オブジェクト・リレーショナル機能

Oracle Databaseのオブジェクト・リレーショナル機能は、ユーザー定義の抽象データ型(ADT)です。ADTは、呼出し側に影響することなく機能を強化できるため、高い拡張性を備えています。外部プロシージャでは、呼出しに必要なあらゆる情報を含むコール指定が、実装の詳細を含むプロシージャの本体とは分かれているためです。本体のみを変更してコール指定を変更しなければ、呼出し側には影響がありません。

関連項目:

- CREATE TYPE文の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください
- CREATE TYPE BODY文の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください
- オブジェクト・リレーショナル機能の詳細は、[『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

1.4 セキュリティを考慮した設計

データベース・セキュリティには、次のように各種のアクティビティが含まれます。

- 組織のデータ、ユーザーおよびアプリケーションを不慮、不適切または不正な操作から保護するセキュリティ・ポリシーの設計および実装
- 不適切または不正な操作に対する監査とアカウントビリティに関するポリシーと手続きの作成および実施
- ユーザー・アカウント、パスワード、ロールおよび権限の作成、メンテナンス、停止
- 様々なコンピューティング・モデルに必要なサービスを安全に提供するアプリケーションの開発、およびデータベースやディレクトリ・サービスを活用した、効率と操作性の最大化

関連項目:

- セキュリティの詳細は、[『Oracle Databaseセキュリティ・ガイド』](#)を参照してください。
- セキュリティを実現する方法と、その際の考慮事項については、[「セキュリティ」](#)を参照してください。

1.5 可用性を考慮した設計

可用性は、要求に応じてアプリケーション、サービスまたは機能がどの程度使用可能であることを表しています。高可用性を考慮して設計されたシステムは、システムのハードウェアまたはソフトウェアのアップグレードなどの操作を実行する最小限の停止時間を除き、主要期間、年間を通じて毎日毎時間、割込みのない計算処理を提供します。高可用性システムには、主に次の特長があります。

- 信頼性
- リカバリ可能性
- 適切なタイミングのエラー検出
- 連続操作

関連項目:

- 高可用性を実現する方法と、その際の考慮事項については、[「高可用性」](#)を参照してください。

1.6 移植性を考慮した設計

PL/SQLの設計で、Oracle Databaseとサード・パーティ製データベースの間の移植性は考慮されていませんが、オペレーティング・システムと言語の間における移植性は確保されています。ほとんどのプログラミング言語はPL/SQLを呼び出すことができ、Macintosh、LinuxおよびWindowsを含めてOracle Databaseに対応しているすべてのプラットフォームでPL/SQLは統一的に実装されています。あるプラットフォーム上でPL/SQLアプリケーションを開発すると、他のどのプラットフォームでもそのアプリケーションの動作は一貫していると考えられます。

PL/SQLストアード・プロシージャは、複数のデータベース間で一定のアプリケーション移植性があります。特定ベンダーの言語で書かれたストアード・プロシージャを使用すると、ある程度はそのベンダーに固定される可能性があります。ストアード・プロシージャでは、アプリケーションのビジュアル・コンポーネント(ユーザー・インタフェース)とそのアプリケーション・ロジックが移植可能になります。データ・ロジックは、アプリケーションが実行されるデータベースに対して最適にエンコードされます。データ・ロジックはストアード・プロシ

ज्याに隠されるため、ベンダーの拡張機能を使用してデータ・レイヤーを最適化できます。

データベース上で開発とデプロイが終了したアプリケーションは、恒久的にデータベースにデプロイされたままになります。アプリケーションを別のデータベースに移動する場合、ビジュアル・コンポーネントとアプリケーション・ロジックはストアド・プロシージャのデータ・ロジックとは関係なく移動できるため、移動が簡単になります。(移動とともにアプリケーションの修正を行うと、移動が複雑になります。)

関連項目:

PL/SQLの概念、使用方法、リファレンス情報については、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください

1.7 診断性を考慮した設計

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。診断性インフラストラクチャの目標は、事前に問題を検出し、問題の検出後には損害と割込みを制限することによって、診断と問題解決に必要な時間を短縮し、Oracleサポートへの問合せを簡単にすることです。

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、状態モニター・レポートなど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。

関連項目:

- 診断ファイルの概要は、『Oracle Database概要』を参照してください
- Oracle Databaseの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。

1.8 特殊な環境を考慮した設計

ここでは、データベースおよびアプリケーションの次の特殊な環境を考慮した設計について説明します。

- [データ・ウェアハウス](#)
- [オンライン・トランザクション処理\(OLTP\)](#),

1.8.1 データ・ウェアハウス

データ・ウェアハウスとは、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。データ・ウェアハウスには、通常、トランザクション・データから導出された履歴データが含まれますが、他のソースからのデータを含めることもできます。データ・ウェアハウスは分析処理をトランザクション処理の負荷と分離し、組織で、様々なソースからのデータを整理統合できるようにします。この方法は、組織がビジネスをよりよく理解し改善するために、履歴レコードを保持してデータを分析するうえで便利です。

リレーショナル・データベースに加え、データ・ウェアハウス環境には次のものを含めることができます。

- 抽出、転送、変換およびロード(ETL)ソリューション
- 統計分析

- レポート
- データ・マイニング機能
- クライアント分析ツール
- データを収集し、有用かつ実用的な情報に変換して、ビジネス・ユーザーに配信するプロセスを管理するアプリケーション

データ・ウェアハウス・システムには、一般的に次の特長があります。

- 多くの索引を使用します。
- 一定数の(多くはない)結合を使用します。
- 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、問合せと分析のパフォーマンスが最適化されます。
- ユーザー導出のデータと集計を使用します。
- ワークロードは非定型問合せとデータ分析に対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、事前には不明な場合があります。このため、データ・ウェアハウスは、様々な問合せ操作および分析操作を適切に実行できるように最適化する必要があります。

- 大量データ修正の技術を使用し、ETLプロセスによって定期的に(毎晩、毎週など)更新されます。

分析ツール(データ・マイニングなど)を使用して、関連する確率から予測を作成したり、顧客を市場セグメントに当てはめたり、顧客プロファイルを作成したりする場合を除き、データ・ウェアハウスのエンド・ユーザーは、データ・ウェアハウスを直接更新することはありません。

関連項目:

オンライン・トランザクション処理(OLTP)との比較など、データ・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

1.8.2 オンライン・トランザクション処理(OLTP),

オンライン・トランザクション処理(OLTP)システムは、トランザクション処理の高速化および信頼性の向上を目的として最適化されています。データ・ウェアハウス・システムと比較して、OLTPの相互作用で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。OLTPシステムでは、パフォーマンス要件として履歴データを頻繁にアーカイブに移動する必要があります。

OLTPシステムには、一般的に次の特長があります。

- 索引はほとんど使用しません。
- 多くの結合を使用します。
- 完全に正規化されたスキーマを使用して、更新、挿入および削除のパフォーマンスを最適化し、データ整合性を保証します。
- ユーザー導出のデータと集計はほとんど使用されません。
- ワークロードは事前定義済の操作で構成されます。
- ユーザーが機械的に、個々のデータ修正のつど修正文を発行するため、OLTPデータベースには常に各トランザクション

の現在の状態が反映されます。

関連項目:

詳細情報が記載されたマニュアルへのリンクを含め、詳細は、[『Oracle Database概要』](#)を参照してください。

1.9 特殊なシナリオのための機能

ここでは、きわめて大規模なデータベースを伴い高パフォーマンスが求められるシナリオで、特に有用なOracle Database機能について説明します。

トピック:

- [SQL分析関数](#)
- [マテリアライズド・ビュー](#)
- [パーティション化](#)
- [時間的な有効性のサポート](#)

1.9.1 SQL分析関数

SQL分析関数は行のグループに基づいて集計値を計算します。SQL分析関数は、グループごとに複数の行を戻す点で集計関数と異なります。行ウィンドウは各行に対して定義されます。このウィンドウによって、カレント行の計算に使用される行の範囲が決定されます。ウィンドウの大きさは、行の物理数値または時間などのロジカル・インターバルに基づきます。

SQL分析関数は、結果セットに対して配列セマンティクスを使用する機能を、セット指向SQLに付与します。これによって、煩雑または不可能になりがちなロジックのコーディングが、簡潔で直接的なものになるため、コーディングが効率的になります。また、Oracle Databaseに統合されて内部の最適化を利用するため、処理効率も向上します。

分析関数を使用する代表例としては、最新の情報を表から取得する場合があります。たとえば、次の形式の問合せは、表にレコードがある顧客ごとに、最新の更新時刻を持つ行から情報を戻します。

```
SELECT ... FROM my_table t1
  WHERE upd_time = ( SELECT MAX(UPD _TIME)
                     FROM my_table t2
                     WHERE t2.cust_id = t1.cust_id );
```

前述の問合せは、相関副問合せを使用し、cust _idを基準としてMAX(UPD _TIME)をレコードごとに検索します。したがって、相関副問合せは表の行ごとに評価される可能性があります。表のレコードがごく少ない場合にはパフォーマンスも十分ですが、表のレコード数が万単位になると、相関副問合せを繰り返し実行する累積的な負荷は大きくなります。

次の問合せは、表でデータ渡しを1回実行するのみで、それと同時に最大のUPD_TIMEを計算します。表のサイズや戻される行数などの様々な要因によって、次の問合せが前述の問合せよりもはるかに効率的な場合があります。

```
SELECT ...
  FROM ( SELECT t1.*,
               MAX(UPD_TIME) OVER (PARTITION BY cust _id) max_time
         FROM my_table t1
       )
 WHERE upd_time = max_time;
```

次の分析関数を使用できます。

AVG
CORR
COUNT
COVAR_POP
COVAR_SAMP
CUME_DIST
DENSE_RANK
FIRST
FIRST_VALUE
LAG
LAST
LAST_VALUE
LEAD
LISTAGG
MAX
MIN
NTH_VALUE
NTILE
PERCENT_RANK
PERCENTILE_CONT
PERCENTILE_DISC
RANK
RATIO_TO_REPORT
REGR_ (Linear Regression) Functions
ROW_NUMBER
STDDEV
STDDEV_POP
STDDEV_SAMP
SUM
VAR_POP
VAR_SAMP
VARIANCE

関連項目:

- SQL分析関クションの概要は、[『Oracle Database概要』](#)を参照してください
- 構文とリファレンス情報は、『Oracle Database SQL言語リファレンス』を参照してください
- 分析とレポートに利用するSQL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

1.9.2 マテリアライズド・ビュー

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格納(マテリアライズ)された問合せ結果です。マテリアライズド・ビューは、表やビューのように問合せのFROM句に表示できます。

マテリアライズド・ビューは、データの集計、計算、レプリケートおよび分配に使用されます。このビューは、一般的な質問に事前に答える場合に役立ちます。ユーザーは、詳細なレコードを個別に集計するかわりに、マテリアライズド・ビューを問い合わせることができます。マテリアライズド・ビューが有用な環境は、データ・ウェアハウス、レプリケーション、モバイル・コンピューティングなどです。

マテリアライズド・ビューは作成と更新に時間がかかり、格納にディスク領域が必要になりますが、問合せパフォーマンスの劇的な向上というメリットでこのデメリットは相殺されます。このような観点でマテリアライズド・ビューは索引と類似しているため、「データ・ウェアハウスの索引」とも呼ばれています。索引と異なるのは、マテリアライズド・ビューは(SELECT文によって)直接問い合わせることができ、(必要な更新のタイプによっては) DML文で更新できる場合もある点です。

問合せリライトが利用可能になることは、マテリアライズド・ビューを作成しメンテナンスすることで得られる主要なメリットの1つです。問合せリライトでは、複数の表やビューに対するSQL文を、ディテール表に定義された1つ以上のマテリアライズド・ビューに対してアクセスする文に変換します。この変換はエンド・ユーザーやアプリケーションに対して透過的に処理され、SQL文内のマテリアラ

イズド・ビューに対する介入や参照は不要です。問合せリライトは透過的な処理であるため、マテリアライズド・ビューは、アプリケーション・コード内のSQLを無効にせず索引のように追加したり削除することができます。

次の文は、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;
```

関連項目:

- マテリアライズド・ビューの概要については、[『Oracle Database概要』](#)を参照してください。
- マテリアライズド・ビューとともに問合せリライトを使用する場合も含め、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
- 表などのマテリアライズド・ビューのパーティション化方法の詳細は、[「パーティション化」](#)を参照してください。

1.9.3 パーティション化

パーティショニングとは、非常に大きな表、索引またはマテリアライズド・ビューを小さい部分に物理的に分割し、個別に管理できるようにするデータベース機能です。パーティショニングは並列処理に似ており、大きなプロセスを小さな部分に分割し、個別に処理できるようにします。

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。可用性が向上する、スキーマ・オブジェクトの管理が容易になる、OLTPシステムで共有リソースの競合が減少する、データ・ウェアハウスで問合せのパフォーマンスが向上するなどのメリットがあります。

表をパーティション化するには、CREATE TABLE文でPARTITION BY句を指定します。SELECT文およびDML文では、パーティショニングを利用するために特殊な構文は必要ありません。

一般的な戦略としては、日付範囲でレコードをパーティション化できます。次の文は、4年分の売上データ(2008年から2011年)のレコードごとに1つずつ、4つのパーティションを作成します。

```
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_2008 VALUES LESS THAN (TO_DATE('01-JAN-2009','DD-MON-YYYY')),
 PARTITION SALES_2009 VALUES LESS THAN (TO_DATE('01-JAN-2010','DD-MON-YYYY')),
 PARTITION SALES_2010 VALUES LESS THAN (TO_DATE('01-JAN-2011','DD-MON-YYYY')),
 PARTITION SALES_2011 VALUES LESS THAN (MAXVALUE)
);
```

関連項目:

- パーティションの概要については、『Oracle Database概要』を参照してください
- 大規模なデータベース(VLDB)に対するパーティション化の詳細な説明と使用方法は、『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

1.9.4 時間的な有効性のサポート

時間的な有効性のサポートにより、1つ以上の有効時間ディメンションを表に関連付け、時間ベースの妥当性に基づいてデータを表示できます。時間は、特定のレコードが有効とみなされる期間の開始日と終了日またはタイムスタンプによって決定されます。時間ベースの妥当性としては、人事管理アプリケーションにおける従業員の雇用日と退職日、保険契約の保証有効日、顧客やクライアントのアドレス変更の有効日などがあります。

時間的な有効性のサポートは、通常、Oracle Flashback Technologyで、AS OF句やVERSIONS BETWEEN句によって有効期間を指定する問合せを実行する際に使用されます。

DBMS_FLASHBACK_ARCHIVE.ENABLE_AT_VALID_TIMEプロシージャを使用すれば、すべての表データ(デフォルト)、指定した時刻に有効なデータ、セッション・レベルで有効期間にある現在の有効データなど、表データの可視性に関するオプションも指定できます。

時間的な有効性のサポートは、情報ライフサイクル管理(ILM)など、特定のデータがいつ有効になったか(アプリケーションの観点から)、いつ無効になったか(無効になった場合)を把握することが重要なアプリケーションで役立ちます。

ノート:



有効時間がサポートされる表の作成と使用、および時間的な有効性のサポートを使用したデータの変更では、ユーザーには表を作成し、データ操作言語(DML)とデータ定義言語(DDL)の操作を表で実行する権限があることが想定されています。

例1-1 有効時間がサポートされる表の作成と使用

次の例では、時間的な有効性のサポートを備えた表を作成し、行を挿入して、個別の行の有効な開始日と終了日に基づいて結果が決まるいくつかの問合せを発行します。

```
CREATE TABLE my_emp(
  empno NUMBER,
  last_name VARCHAR2(30),
  start_time TIMESTAMP,
  end_time TIMESTAMP,
  PERIOD FOR user_valid_time (start_time, end_time));

INSERT INTO my_emp VALUES (100, 'Ames', '01-Jan-10', '30-Jun-11');
INSERT INTO my_emp VALUES (101, 'Burton', '01-Jan-11', '30-Jun-11');
INSERT INTO my_emp VALUES (102, 'Chen', '01-Jan-12', null);

-- Valid Time Queries --

-- AS OF PERIOD FOR queries:

-- Returns only Ames.
SELECT * from my_emp AS OF PERIOD FOR user_valid_time TO_TIMESTAMP('01-Jun-10');
-- Returns Ames and Burton, but not Chen.
SELECT * from my_emp AS OF PERIOD FOR user_valid_time TO_TIMESTAMP('01-Jun-11');
-- Returns no one.
SELECT * from my_emp AS OF PERIOD FOR user_valid_time TO_TIMESTAMP('01-Jul-11');
-- Returns only Chen.
SELECT * from my_emp AS OF PERIOD FOR user_valid_time TO_TIMESTAMP('01-Feb-12');
```

```
-- VERSIONS PERIOD FOR ... BETWEEN queries:

-- Returns only Ames.
SELECT * from my_emp VERSIONS PERIOD FOR user_valid_time BETWEEN
    TO_TIMESTAMP('01-Jun-10') AND TO_TIMESTAMP('02-Jun-10');
-- Returns Ames and Burton.
SELECT * from my_emp VERSIONS PERIOD FOR user_valid_time BETWEEN
    TO_TIMESTAMP('01-Jun-10') AND TO_TIMESTAMP('01-Mar-11');
-- Returns only Chen.
SELECT * from my_emp VERSIONS PERIOD FOR user_valid_time BETWEEN
    TO_TIMESTAMP('01-Nov-11') AND TO_TIMESTAMP('01-Mar-12');
-- Returns no one.
SELECT * from my_emp VERSIONS PERIOD FOR user_valid_time BETWEEN
    TO_TIMESTAMP('01-Jul-11') AND TO_TIMESTAMP('01-Sep-11');
```

列を明示的に追加することなく、時間的な有効性のサポートを既存の表に追加するには、ALTER TABLE文をADD PERIOD FOR句とともに使用します。たとえば、CREATE TABLE文でSTART_TIME列とEND_TIME列が作成されない場合、次の文を使用すると作成できます。

```
ALTER TABLE my_emp ADD PERIOD FOR user_valid_time;
```

前述の文は、2つの非表示列USER_VALID_TIME_STARTおよびUSER_VALID_TIME_ENDを表MY_EMPに追加します。列の値を指定して行を挿入することもできますが、SQL*PlusのDESCRIBE文の出力では列が非表示であり、SELECTリストに明示的に列名を含めた場合にのみ、SELECT文によってその列のデータが表示されます。

[例1-2](#)では、[例1-1](#)で作成した表のデータ変更に使用する有効性のサポートを使用します。[例1-2](#)では、従業員103の初期レコードで姓がDavisになっています。この従業員は、後で自分の姓をSmithに変更します。元の行のEND_TIME値がNULLから、変更が有効になる前の日に変更されます。新しい行が、新しい姓および該当するSTART_TIME値とともに挿入され、END_TIMEがNULLに設定されて、別の姓に設定されるまで有効であることが示されます。

例1-2 時間的な有効性のサポートを使用したデータの変更

```
-- Add a record for Davis.
INSERT INTO my_emp VALUES (103, 'Davis', '01-Jan-12', null);
-- To change employee 103's last name to Smith,
-- first set an end date for the record with the old name.
UPDATE my_emp SET end_time = '01-Feb-12' WHERE empno = 103;
-- Then insert another record for employee 103, specifying the new last name,
-- the appropriate valid start date, and null for the valid end date.
-- After the INSERT statement, there are two records for #103 (Davis and Smith).
INSERT INTO my_emp VALUES (103, 'Smith', '02-Feb-12', null);

-- What's the valid information for employee 103 as of today?
SELECT * from my_emp AS OF PERIOD FOR user_valid_time SYSDATE WHERE empno = 103;
-- What was the valid information for employee 103 as of a specified date?
-- First, as of a date after the change to Smith.
SELECT * from my_emp AS OF PERIOD FOR user_valid_time TO_TIMESTAMP('01-Jul-12')
    WHERE empno = 103;
-- Next, as of a date before the change to Smith.
SELECT * from my_emp AS OF PERIOD FOR user_valid_time TO_TIMESTAMP('20-Jan-12')
    WHERE empno = 103;
```

関連リンク:



- [CREATE TABLE](#)、[ALTER TABLE](#) および [SELECT](#)

- 
- [Oracle Database PL/SQL パッケージおよびタイプ・リファレンス](#)
 - [Oracle Database VLDB およびパーティショニング・ガイド](#)
 - [Oracle Flashback Technology の使用](#)
 - [マルチテナント・コンテナ・データベースでの Oracle Flashback Technology の制限](#)

2 データベース・アプリケーションの接続方法

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。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。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。

Oracle Real-World Performanceグループが調査するOLTPのパフォーマンス問題のほとんどは、アプリケーションの接続方法に関連しています。このため、健全な接続方法を設計することはアプリケーション開発において重要で、特に、規模を拡大して増加する要件を満たす必要のある企業環境では重要です。

トピック:

- [接続プールの設計ガイドライン](#)
- [ログイン方法の接続ガイドライン](#)
- [プログラム・セッション・リークを回避する設計ガイドライン](#)
- [ランタイム接続ロード・บาลancingの使用法](#)

2.1 接続プールの設計ガイドライン

接続プールは、アプリケーションから使用できるOracleデータベースへの接続のキャッシュです。

実行時に、アプリケーションはプールに接続をリクエストします。リクエストを満たすことができる接続がプールにある場合、その接続がアプリケーションに戻されます。アプリケーションは、その接続を使用してデータベースで処理を実行した後、接続をプールに返します。リリースされた接続は次の接続リクエストに使用できます。

静的接続プールでは、プールに含まれている接続の数は固定のため、要件にあわせて増やすことはできません。そのため、プールに新しいアプリケーション・リクエストを満たすアイドルな接続が見つからない場合、リクエストはキューに格納されるか、エラーが返されます。ただし、動的接続プールでは、プールが新しい接続を作成してから、それをアプリケーションに戻します。理論的には、動的接続プールではプール内の接続数を増加および減少させることができ、そのため、不要な接続の維持に浪費される場合のあるシステム・リソースの節約に役立ちます。ただし、実際には動的接続プールの方法では潜在的接続ストームを許容するため、サブスクリプションが超過する問題が発生します。

2.1.1 接続ストーム

接続ストームは、アプリケーション・サーバーがさらに多くの接続リクエストを起動するが、データベース・サーバーCPUがすぐにスケジューリングできないという競合状態で、これによりアプリケーション・サーバーはさらに多くの接続を作成します。

接続ストームの間、データベース接続の数は1分未満の間に数百から数千に急騰します。

動的接続プールは接続ストームを起こしがちです。接続リクエストの数が増えると、データベース・サーバーはCPUコアの数に対して過剰サブスクライブになります。指定の時間では、1つのCPUコアでは1つのプロセスしか実行できません。そのため、サーバー上に32個のコアがある場合、一度に32プロセスしか実行できません。アプリケーション・サーバーが数百または数千の接続を作成した場合、CPUはその数のプロセスについていこうとシステム上の時間を取り合うためにビジーになります。

データベース内では、アクティブ・セッションの数が増加すると待機するアクティビティが増加します。このアクティビティはASHレポートおよびAWRLレポートで待機イベントを参照することで確認できます。典型的な待機イベントにはエンキューへのラッチ、行キャッシュ・オブジェクト、ラッチ・フリー、enq : TX – index contentionおよびバッファ・ビジー待機が含まれます。待機イベントが増

加すると、セッションは作業を実行できないため、トランザクション・スループットが低下します。サーバーのコンピュータは過剰サブスクライブされているため、監視ツール・プロセスはCPU上の時間を勝ち取る必要があります。最も極端なケースでは、キーボードが使用できなくなり、デバッグが困難になります。

ビデオ:



[RWP #13: 大きな動的接続プール - パート 1](#)

2.1.2 接続ストームを回避するためのガイドライン: 静的プールの使用

Oracle Real-World Performanceグループでは、動的接続プールではなく、静的接続プールの使用をお勧めします。

何年にも渡る接続ストームの診断によって、Oracle Real-World Performanceグループは、動的接続プールが必要なワークロードに対して過剰なプロセスを使用することを発見しました。広く行き渡っている通説では、動的接続プールは必要に応じて接続を作成し、必要がなくなると接続を減らすとされています。実際には、接続プールが使い果たされると、アプリケーション・サーバーはデータベース接続のプールのサイズを急速に増加します。セッションの数はシステムへの小さな負荷でも増え、これによりすべてのセッションがアクティブになる際にパフォーマンスの問題を引き起こします。

動的接続プールはシステムをすぐに不安定にしてしまうため、Oracle Real-World Performanceグループでは、動的接続プールよりも静的接続プールをお勧めしています。

接続の数を減らすと、CPUのストレスが減り、これによりレスポンス時間が速くなりスループットが向上します。この結果は矛盾しているように感じるかもしれませんが、パフォーマンスは次の理由で向上します。

- セッションが減るということは、データベース内の競合に関連する待機イベントが減ることを意味します。接続を減らすと、それまでラッチのサイクルを消費し、競合を仲裁していたCPUはデータベース・トランザクションの処理に多くの時間を割くことができるようになります。
- 接続の数が減少すると、接続はCPU上に長くスケジュールされていられるようになります。結果として、これらのプロセスに関連するすべてのメモリー・ページはCPUキャッシュに存在し続けます。スケジューリングがさらに効率的になり、メモリーを消費しなくなります

経験から、Oracle Real-World Performanceグループでは、%userと%systemのCPU使用率の比率を90/10にすることと、データベース・サーバーのCPUコアごとに、割り当てるプロセスを平均10以内することをお勧めしています。接続の数は、CPUコア・スレッドではなく、CPUコアの数に基づくようにしてください。たとえば、サーバーに2つのCPUがあり、それぞれのCPUに18のコアがあるとします。各CPUコアには2つのスレッドがあります。Oracle Real-World Performanceグループのガイドラインに基づくと、アプリケーションではデータベース・インスタンスに対して36から360の接続を持つことができます。

ビデオ:



[RWP #14: 大きな動的接続プール - パート 2](#)

2.2 ログイン方法の接続ガイドライン

すべてのデータベース開発者が直面する問題は、アプリケーションがいつどのようにデータベースにログインしてトランザクションを起動するかです。

最適に次ぐ設計では、データベース・アプリケーションは、それぞれのSQLリクエストに対して次のステップを実行します。

1. データベースにログインします。
2. INSERT文またはUPDATE文などのSQLリクエストを発行します。
3. データベースからログアウトします。

ログイン/ログアウトの方法を使用するアプリケーションは機能要件を満たすことができます。また、1秒当たりのトランザクションの数が少ないときには優れたパフォーマンスを示します。ただし、データベースでのログインおよびログアウトは非常にリソース消費量の多い操作です。Oracle Real-World Performanceグループは、そのようなアルゴリズムを使用するアプリケーションは、うまくスケーリングできず、特に動的接続プールを使用している場合に重大なパフォーマンスの問題を引き起こす可能性があることを発見しました。ログイン/ログアウトの方法では、通常、接続プールは使用しません。

アプリケーションがログイン/ログアウトの設計を使用し、DBAおよび開発者が問題の原因に気がつかないと、最初に関与する症状はデータベース・スループットの低下と応答時間の不規則で極端な増大です。データベースの診断調査では、リソース競合が少ない間はアクティブなセッションがかなり少ないことを示すことがあります。

最適でないパフォーマンスに対するヒントとしては、1秒当たりのログイン数は1秒当たりのトランザクション数に近くなります。トランザクションごとのログイン/ログアウトの方法が使用されると、データベース・インスタンスおよびオペレーティング・システムは、内部的に多くの作業を実行して新しいプロセス、データベース接続および関連するメモリー領域を作成します。こうしたステップの多くはシリアル化され、CPU使用を低下させ、それによりトランザクション・スループットを低下させます。

前述の理由から、Oracle Real-World Performanceグループでは、多数のトランザクションをサポートするようにスケーリングする必要のあるアプリケーションの場合、ログイン/ログアウト設計を使用しないことを強くお勧めしています。

ビデオ:



[RWP #2 カーソルおよびログオンを使用した低パフォーマンス](#)

2.3 プログラム・セッション・リークを回避する設計ガイドライン

プログラムが接続を失うと、セッション・リークが起こりますが、データベース・インスタンス内ではセッションはアクティブなままです。リークされたセッションはプログラムのアプリケーションからみつからなくなります。

最適に設計されたアプリケーションでは、セッション・リークが起こらないようにされています。通常、セッション・リークはアプリケーションによって例外が見つかったために発生します。アプリケーションが例外を適切に処理しない場合、接続はコミットやロールバックを実行することなく停止し、セッションがリークされます。

セッション・リークは、データベース・パフォーマンスおよびデータ整合性の重大な問題を引き起こす可能性があります。一般的に、問題は次の形になります。

- 失われた接続プール
- ロック・リーク
- 論理的な破損

2.3.1 失われた接続プール

設計の不備によって接続プールが失われる場合があります。

たとえば、アプリケーション設計の不備によってセッションが常にリークするとします。リーク率が低かったとしても、動的な接続プールではかつてないほど多くのセッションがプログラムの使用できなくなります。

結果として使用できる接続プールが減り、残りの接続ではワークロードが維持できなくなります。使用できないセッションは増え続け、プール内に使用できる接続がなくなります。

2.3.2 セッション・リークの確認

セッション・リークはアプリケーションやアプリケーション・サーバーの問題が原因で発生します。これは、データベース単独で修正することはできません。この問題には、アプリケーションやアプリケーション・サーバーで対応する必要があります。セッション・リークを確認する簡単な方法は、データベースへの接続を1つ使用するように接続プールを変更し、アプリケーションをテストすることです。1つの接続を使用してテストを行うと、アプリケーションの問題の根本的な原因を簡単に見つけることができます。

2.3.3 ロック・リーク

ロック・リークは、一般的にセッション・リークの副作用です。

たとえば、バッチ更新の途中のリークされたセッションは表内の複数行に対してトランザクション・ロックを保有している可能性があります。リークされたセッションがロックを保有している場合、ロックを取得したいセッションはリークされたセッションの後にキューを作成します。

ロックを保有しているプログラムはロックを解除するためにクライアントからの対話を待ちますが、プログラムの接続が失われているため、メッセージは送信されません。そのため、データベースはセッション内でアクティブなどのセッションに対してもコミットまたはロールバックできません。

2.3.4 論理的な破損

リークされたセッションでは、データベースに対するコミットされていない変更が含まれている場合があります。データベース接続が予期せず解除された際にトランザクションが作業の途中だったなどです。

この状況では、次の問題を引き起こす可能性があります。

- アプリケーションはUIにエラーをレポートします。この場合、お客様はビジネス・オーダーやフライト・スケジュールなどの作業を失ったと苦情を言う可能性があります。
- UIは、コミットまたはロールバックが発生していなくても、コミット・メッセージを受け取ります。これは、後続のトランザクションがそのトランザクション自体の作業とリークされたセッションの途中までのトランザクションの両方をコミットする可能性があるため、最悪のケースです。このケースでは、データベースは論理的に破損しています。

2.4 ランタイム接続ロード・バランシングの使用法

トピック:

- [ランタイム接続ロード・バランシングについて](#)
- [ランタイム接続ロード・バランシングの有効化と無効化](#)
- [ロード・バランシング・アドバイザーFANイベントの受信](#)

2.4.1 ランタイム接続ロード・バランシングについて

Oracle Real Application Clusters (Oracle RAC)は、単一データベースを複数ノードにある複数インスタンスで管理する

データベース・オプションです。Oracle RACの共有ディスク方式でデータベースをクラスタ化すると、スケーラビリティが向上します。現在の必要性に応じてノードを簡単に追加または解放でき、いずれかのノードで障害が発生しても別のノードがワークロードを引き継ぐので、可用性が向上します。すべてのインスタンスがデータベース全体にアクセスできるため、Oracle RACではデータベースに高可用性機能とフェイルオーバー機能が追加されます。

作業リクエストのバランシングは、接続時(Oracle Net Servicesで提供される接続ロード・バランシング)と実行時(ランタイム接続ロード・バランシング)の両方で発生します。Oracle RAC環境の場合、アプリケーションのセッション要求を均等に分散させるために、セッション・プールは、高速アプリケーション通知(FAN)イベントを介してOracle RACロード・バランシング・アドバイザから受信されるサービス・メトリックを使用します。セッション・プールに入ってくる作業要求は、現在のサービス・パフォーマンスを使用して、サービスを提供しているOracle RACのインスタンス全体で分散できます。

接続ロード・バランシングは、アプリケーションがセッションを作成するときに発生します。プールされたセッションは、各インスタンスでセッションが作業を実行できるようにするため、セッションの作成時にOracle RACインスタンス間で適切に分散する必要があります。

ランタイム接続ロード・バランシングは、アプリケーションが既存のセッション・プールからセッションを選択する際に発生します(したがって頻度の高いアクティビティです)。ランタイム接続ロード・バランシングは、セッション・プール内で作業の処理に最も適切なセッションに作業リクエストをルーティングします。単一インスタンスでのみサービスをサポートするセッション・プールの場合は、プール内で使用可能な最初のセッションで十分です。プールが複数インスタンスにまたがるサービスをサポートしている場合は、適切なサービスを提供できる、または容量の大きいインスタンスが多数のリクエストを取得するように、作業をインスタンス間で分散する必要があります。

OCI、OCCI、JDBC、ODP.NETのクライアント・アプリケーションはすべて、ランタイム接続ロード・バランシングをサポートしています。

関連項目:

- [demoディレクトリのcdemosp.c](#)
- [データベース常駐接続プールの使用](#)
- [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](#)
- [Oracle Database JDBC開発者ガイド](#)
- [Oracle Universal Connection Pool for JDBC Java APIリファレンス](#)
- [Oracle Data Provider for .NETの開発者ガイドfor Microsoft Windows](#)

2.4.2 ランタイム接続ロード・バランシングの有効化と無効化

クライアントでのランタイム接続ロード・バランシングの有効化と無効化は、クライアント環境によって異なります。

トピック:

- [OCI](#)
- [OCCI](#)
- [JDBC](#)
- [ODP.NET](#)

2.4.2.1 OCI

OCIクライアント・アプリケーションの場合、ランタイム接続ロード・バランシングは、アプリケーションがサービス時間に基づいてサービス・メトリックを受信するように、次の操作の実行時にOracle Database 11gリリース1 (11.1)以上のクライアントとOracle Database 10gリリース2 (10.2)以上のサーバーとの通信においてデフォルトで有効になります。

- アプリケーションをスレッド・ライブラリにリンクしている。
- OCI環境をOCI_EVENTSおよびOCI_THREADEDモードで作成している。
- セッション・プールで使用するサービスに対して、ロード・バランシング・アドバイザの目標と接続ロード・バランシングの目標を構成している。

OCIクライアントのランタイム接続ロード・バランシングを無効にするには、OCISessionPoolCreate()をコールする際にmodeパラメータをOCI_SPC_NO_RLBに設定します。

OCI用のFAN HA (FCF)では、サービスがTRUEになるにはAQ_HA_NOTIFICATIONSが必要です。

関連項目:

OCISessionPoolCreate()の詳細は、『Oracle Call Interfaceプログラマーズ・ガイド』を参照してください

2.4.2.2 OCCI

OCCIクライアント・アプリケーションの場合、ランタイム接続ロード・バランシングは、次の操作の実行時にOracle Database 11gリリース1 (11.1)以上のクライアントとOracle Database 10gリリース2 (10.2)以上のサーバーとの通信においてデフォルトで有効になります。

- アプリケーションをスレッド・ライブラリにリンクしている。
- OCCI環境をEVENTSおよびTHREADED_MUTEXEDモードで作成している。
- セッション・プールで使用するサービスに対して、ロード・バランシング・アドバイザの目標と接続ロード・バランシングの目標を構成している。

OCCIクライアントのランタイム接続ロード・バランシングを無効にするには、NO_RLBオプションをStatelessConnectionPool ClassのPoolType属性に対して使用します。

OCCI用のFAN HA (FCF)では、サービスがTRUEになるにはAQ_HA_NOTIFICATIONSが必要です。

関連項目:

OCCIインタフェースを使用するランタイム・ロード・バランシングの詳細は、『Oracle C++ Call Interfaceプログラマーズ・ガイド』を参照してください

2.4.2.3 JDBC

JDBC環境で、ランタイム接続ロード・バランシングは、高速接続フェイルオーバー(FCF)の有効時にOracle Database 11gリリース1 (11.1)以上のクライアントとOracle Database 10gリリース2 (10.2)以上のサーバーとの通信においてデフォルトで有効になります。

JDBC環境で、ランタイム接続ロード・バランシングは、FCF処理で使用するのと同じバンド外ONSイベント・メカニズムを使用する、Oracle Notification Service(ONS)インフラストラクチャに依存します。ランタイム接続ロード・バランシングが機能するた

めにONSの追加設定や構成は必要ありません。

JDBC環境でランタイム接続ロード・バランシングを無効にするには、値をfalseに設定して
setFastConnectionFailoverEnabled()をコールします。

関連項目:

JDBCインタフェースを使用するランタイム・ロード・バランシングの詳細は、『Oracle Database JDBC開発者ガイド』を参照してください

2.4.2.4 ODP.NET

ODP.NETクライアント・アプリケーションの場合は、ランタイム接続ロード・バランシングがデフォルトで無効です。ランタイム接続ロード・バランシングを有効にするには、接続文字列に"Load Balancing=true"を含めて、"Pooling=true" (デフォルト)を確認します。

ODP.NET用のFAN HA (FCF)では、サービスがTRUEになるにはAQ_HA_NOTIFICATIONSが必要です。

関連項目:

ランタイム・ロード・バランシングの詳細は、[『Oracle Data Provider for .NET開発者ガイドfor Microsoft Windows』](#)を参照してください

2.4.3 ロード・バランシング・アドバイザFANイベントの受信

アプリケーションでは、次の要件がすべて満たされている場合のみロード・バランシング・アドバイザFANイベントを受信できます。

- Oracleクラスタウェアを伴うOracle RAC環境が設定され、有効化されていること。
- サーバーが、イベント通知を発行するように構成されていること。
- アプリケーションがスレッド・ライブラリとリンクされていること。
- OCI環境がOCI_EVENTSおよびOCI_THREADEDモードで作成されていること。
- OCCI環境がTHREADED_MUTEXEDおよびEVENTSモードで作成されていること。
- DBMS_SERVICEパッケージを使用してOracle RAC環境を構成または変更していること。

サービスの目的および接続ロード・バランシングの目的を設定するため、サービスを次のように変更する必要があります。

```
EXEC DBMS_SERVICE.MODIFY_SERVICE("myService",  
    DBMS_SERVICE.GOAL_SERVICE_TIME,  
    clb_goal => DBMS_SERVICE.CLB_GOAL_SHORT);
```

定数GOAL_SERVICE_TIMEは、ロード・バランシング・アドバイザがサービスでの処理の実行に要する時間およびサービスで使用可能な帯域幅に基づくことを指定します。

定数CLB_GOAL_SHORTは、接続ロード・バランシングでロード・バランシング・アドバイザを使用し、ロード・バランシング・アドバイザが有効であることを指定します。接続バランシングの目標はCLB_GOAL_LONGに設定します。ただし、CLB_GOAL_LONGはクローズされたワークロードに対して(つまり、完了した作業の割合と新しく始める作業の割合が等しいときに)最も有効です。

関連項目:

- OCIクライアントでFANイベントの受信を有効にする手順は、『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
- OCISessionPoolCreate()の詳細は、『Oracle Call Interfaceプログラマーズ・ガイド』を参照してください
- JDBCインタフェースを使用するランタイム・ロード・バランシングの詳細は、『Oracle Database JDBC開発者ガイド』を参照してください

3 パフォーマンスとスケーラビリティ

この章では、パフォーマンスとスケーラビリティをデータベースおよびデータベース・アプリケーションに組み込む設計技術について説明します。

トピック:

- [パフォーマンス戦略](#)
- [パフォーマンスのためのツール](#)
- [データベース・パフォーマンスの監視](#)
- [インストルメンテーションの使用](#)
- [パフォーマンスのテスト](#)
- [バインド変数の使用](#)
- [クライアント結果キャッシュの使用](#)
- [文キャッシング](#)
- [OCIクライアント文キャッシュの自動チューニング](#)
- [クライアント側デプロイメント・パラメータ](#)
- [問合せ変更通知の使用](#)
- [データベース常駐接続プールの使用](#)
- [Memoptimizeプール](#)
- [Oracle RACシャーディング](#)

3.1 パフォーマンス戦略

トピック:

- [実行するデータ・モデルの効率的な設計](#)
- [パフォーマンス目標\(メトリック\)の設定](#)
- [アプリケーションのベンチマーク](#)

3.1.1 実行するデータ・モデルの効率的な設計

ここでは、データ・モデリングの重要な概念について簡単に説明します。この点については各種の参考書籍があるため、詳細やガイドについてはそれらを参照してください。

次の基本手順に従って、最も頻繁に実行する重要な問合せのパフォーマンスを最適化するためのデータ・モデルを設計します。

1. [アプリケーションのデータ要件の分析](#)
2. [アプリケーションのデータベース設計の作成](#)
3. [データベース・アプリケーションの実装](#)
4. [データベースとデータベース・アプリケーションのメンテナンス](#)

3.1.1.1 アプリケーションのデータ要件の分析

アプリケーションのデータ要件を次の基本手順に従って分析します。

1. データを収集します。

インタビューを通じてビジネスの内容、アプリケーションの性質、情報の使用者と使用方法、エンドユーザーの要求を把握します。人事フォーム、請求フォーム、注文フォームなどのビジネス・ドキュメントを収集して、ビジネスにおける情報の使用方法を学びます。

2. 収集したデータを分析します。

このボトムアップ・プロセスには、データの正規化、エンティティ関連のモデリングおよびトランザクション分析が含まれます。

3. データの機能分析を行います。

このトップダウン・プロセスの結果はデータ・フロー・ダイアグラムで、メインのプロセス・ブロックと、各ブロックでの存続期間中のデータの出入りが示されます。

3.1.1.2 アプリケーションのデータベース設計の作成

データベース設計を作成するには、まず最初に論理設計を作成し、次にこの論理設計を物理設計に変換します。

トピック:

- [論理設計の作成](#)
- [物理設計の作成](#)

関連項目:

パフォーマンスを考慮した設計と開発の詳細は、[『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

3.1.1.2.1 論理設計の作成

論理設計はデータベースのグラフィカルな表現です。論理設計は、アプリケーションのデータベース・オブジェクトとトランザクション・アクティビティ間の両方の関係をモデル化します。効果的な論理設計では、データの所有、アクセス、更新を必要とするユーザーごとに要件を考慮します。

データベース・オブジェクト間の関係をモデル化するには:

1. データ要件をデータ項目(列)に変換します。
2. 関連列を表内にグループ化します。
3. 列と表の間の関係をマッピングして、各表に対する主キーおよび外部キー属性を決定します。
4. 冗長性と依存性を最小限にするために表を正規化します。

トランザクション・アクティビティをモデル化するには:

- 代表的なトランザクションと、ユーザーにとって最重要と想定されるトランザクションを理解します。
- 論理モデルを通じてトランザクション・パスを追跡します。
- SQLでトランザクションをプロトタイプ化し、データベースのサイズを示すボリューム表を開発します。
- どのユーザーがどの順序でどの表にアクセスするか、そしてどのトランザクションがデータを読み取り、どのトランザクションがデータを書き込むかを決定します。

- アプリケーションがデータの読取りと書き込みのどちらを主に行うかを決定します。

3.1.1.2.2 物理設計の作成

物理設計は、物理データベースでの論理設計の実装です。

論理設計は、表と列、表間や表内の関係、および既知のすべてのトランザクション・アクティビティに関する情報を統合するため、データベースがどのようにデータを表に格納し、索引などのその他の構造をどのように作成するかを理解できます。

この知識を使用することで、スキーマ定義の作成にSQLデータ定義言語(DDL)を使用するスクリプトを作成し、必要な順序にデータベース・オブジェクトを定義して、仕様に応じたストレージ要件などを定義します。

3.1.1.3 データベース・アプリケーションの実装

データベース・アプリケーションを次の基本手順に従って実装します。

1. アプリケーションをテスト環境で実装します。

本番環境と可能な限り一致するテスト環境で、物理データベース設計を実装するスクリプトを作成します。データを物理スキーマ内にロードします。アプリケーションを開発するためのプログラミング言語を選択し、ユーザー・インタフェースを開発して、トランザクションの作成とテストなどを行います。

2. アプリケーションが仕様に即して稼働するようにします。

すべてのコンポーネントが機能していること、アプリケーションが完全に動作すること、そしてアプリケーションで使用されるデータベース機能が適切に構成されていることを確認します。

3. アプリケーションでベンチマーク・テストを実行します。

ベンチマーク・テストでは、データおよびユーザーの追加など、リアルタイムの操作をシミュレーションすることで様々なワークロード(ピーク・アクティビティを含む)下で予定どおりにアプリケーションが稼働するかどうかを確認できます。アプリケーションが幅広く順応することを確認します。

アプリケーションがベンチマークを下回る場合、最初はワークロードがない状態から徐々にワークロードを増やすことで、最適に動作するようにSQL文を調整します。

4. アプリケーションを本番環境で実装します。

関連項目:

- SQLチューニングの詳細は、[Oracle Database SQLチューニング・ガイド](#)を参照してください
- SQLチューニングの詳細は、[『Oracle Database 2日でパフォーマンス・チューニング・ガイド』](#)を参照してください
- ワークロード・テスト、モデリングおよび実装の詳細は、[『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
- パフォーマンス・テスト用ツールの詳細は、[「パフォーマンスのためのツール」](#)を参照してください。
- アプリケーションのベンチマークの詳細は、[「アプリケーションのベンチマーク」](#)を参照してください。
- 新しいアプリケーションのデプロイの詳細は、[『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

3.1.1.4 データベースとデータベース・アプリケーションのメンテナンス

データベース、データベース・アプリケーション、オペレーティング・システムのメンテナンスは、データベース管理者、アプリケーション開発者およびシステム管理者がそれぞれ行う継続的なタスクです。ビジネスがメンテナンスに割り当てるリソースは、データベースとデータベース・アプリケーションの重要性、成長に関する潜在的な能力、対応が必要なユーザー数などによって異なります。

メンテナンスの担当者は、システムを定期的に監視し、メンテナンス期間をスケジューリングして、予定されているメンテナンス期間をユーザーに通知する必要があります。メンテナンス期間に停止時間が必要な場合は、データベース・アクティビティがないか少ない時間帯にスケジューリングします。

アプリケーションのメンテナンスには、不具合の修正、パッチの適用、アップグレードのリリースが含まれます。メンテナンス作業は、本番システムに実装する前に、テスト環境で事前にテストして問題を捕捉して解決します。

3.1.2 パフォーマンス目標(メトリック)の設定

次のようなパフォーマンス目標(メトリック)を設定して、アプリケーション開発プロジェクトを開始します。

- 予想されるアプリケーション・ユーザー数
- 予想されるピーク・ロード時の1秒当たりのトランザクション数
- 予想されるピーク・ロード時の問合せレスポンス時間
- 予想される単位時間(1日、1か月、1年など)当たりの表ごとのレコード数

これらのメトリックを使用してベンチマーク・テストを作成します。

3.1.3 アプリケーションのベンチマーク

ベンチマークは、アプリケーションの性能の局面を測定するテストです。ベンチマークの結果によって、アプリケーション設計が検証されるか、アプリケーションの本番稼働前に解決できる問題が発見されます。

通常は、他の要因による干渉を最小限にするために、分離した単一のユーザー・システムに対して最初にベンチマークを実行します。ベンチマークの結果は、アプリケーションのパフォーマンス・ベースラインになります。意味のあるベンチマーク結果を得るには、アプリケーションを実行する予定の環境でアプリケーションをテストする必要があります。

最も重要なトランザクションのパフォーマンスを測定し、パフォーマンス上の問題に対する複数のソリューションを比較して、パフォーマンスに影響し得る設計上の問題の解決を手助けする小さいベンチマークを作成できます。

ピークのユーザー負荷時やピークのトランザクション負荷時、あるいはその両方でアプリケーションのパフォーマンスを測定するには、それより大きい複雑なベンチマークを開発する必要があります。このようなベンチマークは、ユーザーまたはトランザクションの負荷が徐々に増加する場合は特に重要です。このベンチマークの予算を確保し、計画する必要があります。

アプリケーションの本番環境での稼働後、本番環境でベンチマークを定期的に行うことでその結果をデータベース表に保存します。各ベンチマークの実行後、パフォーマンスの低下を許容できないトランザクションの旧レコードと新レコードを比較します。この方式を使用することで、問題の発生時にそれを隔離できます。ユーザーがパフォーマンスに関する不満を述べるまで待っていると、問題の発生時点を判断できなくなる可能性があります。

関連項目:

アプリケーションのベンチマークの詳細は、『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

3.2 パフォーマンスのためのツール

トピック:

アプリケーションに関する実行時パフォーマンス情報をレポートするツールがいくつかあります。

- [DBMS_APPLICATION_INFOパッケージ](#)
- [SQLトレース機能\(SQL_TRACE\)](#)
- [EXPLAIN PLAN文](#)

関連項目:

データベースのチューニング用のツールに関する詳細は、[『Oracle Database Testingガイド』](#)を参照してください

3.2.1 DBMS_APPLICATION_INFOパッケージ

SQLトレース機能およびOracle TraceとともにDBMS_APPLICATION_INFOパッケージを使用して、実行しているモジュールまたはトランザクションの名前をデータベースに記録します。システム管理者およびパフォーマンス・チューニング担当者はこの記録された情報を使用して、個々のモジュールのパフォーマンスをデバッグ目的のために追跡できます。システム管理者は、モジュール別のリソース使用率をこの情報から追跡することもできます。

アプリケーションをデータベースに登録すると、その名前およびアクションがV\$SESSIONおよびV\$SQLAREAのビューに記録されます。

DBMS_APPLICATION_INFOパッケージは、V\$SESSIONビューで次の列を設定するサブプログラムを提供します。

- MODULE (アプリケーションまたはパッケージの名前)
- ACTION (トランザクションまたはパッケージされたサブプログラムの名前)
- CLIENT_INFO (現在のセッションの初期バインド変数値などの、クライアント・アプリケーションに関する追加情報)

DBMS_APPLICATION_INFOパッケージは、現在のセッションの先行するV\$SESSION列から情報を返すサブプログラムも提供します。

DBMS_APPLICATION_INFOパッケージは、結果の表示に数秒以上かかるコマンド(索引の作成や多数の行の更新など)の進行状況の追跡にも使用できます。DBMS_APPLICATION_INFOパッケージは、コマンドに関する情報をV\$SESSION_LONGOPSビューに保存するサブプログラムを提供します。V\$SESSION_LONGOPSビューには、コマンドの開始時刻、処理の進んだ段階、完了までの推定時間などが表示されます。

関連項目:

- DBMS_APPLICATION_INFO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
- SQLトレース機能があるDBMS_APPLICATION_INFOパッケージの詳細は、[『SQLトレース機能\(SQL_TRACE\)』](#)を参照してください。
- V\$SESSIONビューの詳細は、[『Oracle Databaseリファレンス』](#)を参照してください。
- V\$SESSION_LONGOPSビューの詳細は、[『Oracle Databaseリファレンス』](#)を参照してください。

- V\$SQLAREAビューの詳細は、[『Oracle Databaseリファレンス』](#)を参照してください。

3.2.2 SQLトレース機能(SQL_TRACE)

アプリケーションで実行されるすべてのSQL文とPL/SQLブロックを追跡するには、SQLのトレース機能であるSQL_TRACEを使用します。単一の文またはモジュールに対してSQLトレース機能を有効にし、文またはモジュールの実行後に無効にすると、その文またはモジュールのみのトレース情報を取得できます。

最適な結果を得るには、SQLトレース機能をTKPROFおよびEXPLAIN PLAN文とともに使用します。SQLトレース機能は、個々のSQL文に関するパフォーマンス情報を提供します。TKPROFはトレース・ファイルの内容を読み取り可能なファイルにフォーマットします。EXPLAIN PLAN文は、オプティマイザによって選択された実行計画を表示し、現在のセッションで実行された場合には指定したSQL文の実行計画を表示します。

トレースする内容と、本番稼働前のアプリケーションのパフォーマンス状況がわかるように、SQLトレース機能はアプリケーションの設計時に使用します。本番環境でパフォーマンス上の問題が大きくなるまで待つと、アプリケーション構造が原因でSQLトレース機能を使用するのがほとんど不可能になる可能性があります。

関連項目:

- [EXPLAIN PLAN文](#)
- SQL_TRACEおよびTKPROF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

3.2.3 EXPLAIN PLAN文

SQL文を実行すると、オプティマイザは可能な実行計画をいくつか生成して、各計画のコストを計算し、最もコストの低い計画を使用します。

計画のコストは、SQL文がアクセスする表、索引およびパーティションのデータ分散および記憶特性などの統計に基づいて計算されます。アクセス・パスや結合順序のコストは、I/O、CPU、メモリーなどのコンピュータ・リソースの使用見積りに基づいています。

EXPLAIN PLAN FOR statement文をstatementの実行前に使用すると、EXPLAIN PLAN文はstatementの実行計画を計画表に格納します。計画表を問い合わせると、オプティマイザで選択された実行計画を調べることができます。

計画表は実行計画を行ソース・ツリーとして表し、これはSQL文を実行するためにOracle Databaseで使用されるステップを示します。行ソース・ツリーには、文が表を参照する順序、結合操作に影響される表の結合メソッドの他に、フィルタ、ソート、集計などのデータ操作が示されます。また計画表には、最適化に関する情報(各操作のコストやカーディナリティなど)、パーティショニング情報(アクセスされるパーティションのセットなど)、およびパラレル実行情報(結合入力の分散メソッドなど)が含まれます。この情報からは、オプティマイザが実行計画を選択した経緯と理由について、重要な洞察を得ることができます。

データについてオプティマイザで認識されていない情報がある場合、オプティマイザにヒントを提供すると、実行計画が変更されることがあります。テストによってそのヒントがパフォーマンスを向上させることが確認された場合は、コードでそのヒントを維持します。

ノート:



データベースの状態の変化や以降のリリースにおける問合せパフォーマンスの向上によって、ヒントがパフォーマンスに及ぼす影響が大きく変化する場合がありますため、ヒントが含まれるコードは定期的にテストする必要があります。

EXPLAIN PLAN文は、EXPLAINされたSQL文をOracle Databaseがどのように実行するかのみを示します。実行環境ま

たはEXPLAIN PLAN環境が変化する場合、オプティマイザは別の実行計画を使用する可能性があります。したがって、SQL計画管理を使用して、SQL計画ベースライン(許容されたSQL文の一連の実行計画)を作成することをお勧めします。

最初に、EXPLAIN PLAN文を使用して文の実行計画を参照します。次に、実行計画をテストし、文の実際のリソース使用量を考慮しながらその適性を検証します。最後に、計画が最適ならば、SQL計画管理を使用します。

関連項目:

- クエリー・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
- 問合せ実行計画の詳細は、Oracle Database SQLチューニング・ガイドを参照してください
- オプティマイザに対するヒントの影響の詳細は、Oracle Database SQLチューニング・ガイドを参照してください
- SQL計画管理の詳細は、Oracle Database SQLチューニング・ガイドを参照してください

3.3 データベース・パフォーマンスの監視

Oracle Databaseには、データベースの管理とチューニングが可能なアドバイザおよび強力なツールが用意されています。

トピック:

- [自動データベース診断モニター\(ADDM\)](#)
- [データベースのパフォーマンスのリアルタイムの監視](#)
- [パフォーマンス関連のアラートに対する対応](#)
- [SQLアドバイザとメモリー・アドバイザ](#)

3.3.1 自動データベース診断モニター(ADDM)

自動データベース診断モニター(ADDM)は、自動ワークロード・リポジトリ(AWR)で取得されたデータの分析を行うアドバイザです。ADDMとAWRはOracle Diagnostic Packの一部です。

ADDMはデータベース・パフォーマンス上の問題が存在する可能性がある場所と、実際に存在する場所を判断し、その修正事項を推奨します。

ADDMは各AWRスナップショットの後に分析を実行し、結果をデータベースに保存します。デフォルトでは、AWRスナップショットの間隔は1時間で、ADDM結果の保存期間は8日間です。

Oracle Enterprise Manager Cloud Control (Cloud Control)ではADDM結果が「データベース」ホームページに表示されます。このAWRスナップショット・データではADDM結果を確認できますが、ADDMで提供される分析と推奨事項はありません。(AWRスナップショット・データは、データベース管理者がパフォーマンス分析のために使用したStatspackデータと同様です。)

関連項目:

ADDMの構成、ADDM分析の確認、ADDM結果の解釈、ADDM推奨事項の実装、およびEnterprise Managerによるスナップショット統計の表示の詳細は、『Oracle Database 2日でパフォーマンス・チューニング・ガイド』を参照してください

3.3.2 リアルタイムなデータベースのパフォーマンスの監視

Oracle Enterprise Manager Cloud Control (Cloud Control)を使用すると、「パフォーマンス」ページからリアルタイムのデータベース・パフォーマンスを監視できます。「パフォーマンス」ページから、パフォーマンス上の問題を特定するページにアクセスし、次のADDM分析を待たずにその問題を解決することができます。

関連項目:

データベースのパフォーマンスのリアルタイムの監視の詳細は、『Oracle Database 2日でパフォーマンス・チューニング・ガイド』を参照してください

3.3.3 パフォーマンス関連のアラートに対する対応

Oracle Enterprise Manager Cloud Control (Cloud Control)の「データベース」ホームページには、データベースによって生成されたパフォーマンス関連のアラートが表示されます。これらのアラートによって示された問題を解決すると、データベース・パフォーマンスが向上します。Oracle Databaseのデフォルトでは、表領域の使用状況、古すぎるスナップショット、リカバリ領域の空き容量低下、再開可能なセッションの一時停止などについてアラートが有効です。メトリックとしきい値の表示、メトリックしきい値の設定、アラートへの対応、アラートのクリア、アラートのダイレクト電子メール通知などを実行できます。このような組み込みのアラート・インフラストラクチャを使用すれば、パフォーマンス関連の特殊なアラートについて通知を受信できます。

関連項目:

- アラートを使用してデータベースの監視とチューニングを行い、アラートを管理する方法の詳細は、『Oracle Database管理者ガイド』を参照してください
- パフォーマンス・アラートの監視の詳細は、『Oracle Database 2日でパフォーマンス・チューニング・ガイド』を参照してください

3.3.4 SQLアドバイザとメモリー・アドバイザ

Oracle Databaseには、データベース・パフォーマンスの向上に効果を発揮するSQLアドバイザとメモリー・アドバイザが用意されています。

SQLアドバイザには、SQLチューニング・アドバイザとSQLアクセス・アドバイザがあり、どちらもOracle Database Tuning Packの一部です。

SQLチューニング・アドバイザは、1つ以上のSQL文を入力として受け取り、特定のチューニング推奨を返します。各推奨には論理および予測されるメリットが含まれます。推奨事項は、オブジェクト統計、新しい索引の作成、SQL文の再構成、およびSQLプロファイルの作成に基づいています。

SQLアクセス・アドバイザを使用すると、指定したSQLワークロードに対して一連のマテリアライズド・ビュー、ビュー・ログ、索引およびパーティションが推奨されるため、SQL問合せのデータ・アクセス・パスを最適化できます。

メモリー・アドバイザには、メモリー・アドバイザ、SGAアドバイザ、共有プール・アドバイザ、バッファ・キャッシュ・アドバイザおよびPGAアドバイザがあります。メモリー・アドバイザは、合計メモリーのターゲット設定、SGAおよびPGAターゲット設定、およびSGAコンポーネントのサイズ設定の分析結果をグラフィカルに表示します。この分析結果は、データベース・パフォーマンスのチューニングや可能性のある状況の計画に使用できます。

関連項目:

- 各パフォーマンス・アドバイザの簡単な説明も含め、アドバイザ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の詳細は、『Oracle Database 2日データベース管理者』を参照してください
- メモリー・アドバイザを使用してメモリー使用状況を最適化する方法の詳細は、『Oracle Database 2日データベース管理者』を参照してください
- データベースのチューニングに使用するツールの詳細は、『Oracle Database 2日パフォーマンス・チューニング・ガイド』を参照してください
- SQLチューニング・アドバイザの詳細は、Oracle Database SQLチューニング・ガイドを参照してください
- SQLチューニング・アドバイザの詳細は、Oracle Database SQLチューニング・ガイドを参照してください

3.4 パフォーマンスのテスト

アプリケーションのパフォーマンスをテストする際は、次のガイドラインに従います。

- 自動データベース診断モニター(ADDM)とSQLチューニング・アドバイザを使用した設計の検証。

ADDMはデータベース・パフォーマンス上の問題が存在する可能性がある場所を判断し、修正事項を推奨します。たとえば、ADDMが負荷の高いSQL文を発見した場合は、SQLチューニング・アドバイザを使用してその文を分析し、チューニングの推奨を提供できます。

- 現実的なデータ量とデータ配分によるテスト

テスト用データベースには、本番システムと同様のデータを入れておく必要があります。表は完全に移入され、本番システムで使用される表間のデータ量やカーディナリティを表すものである必要があります。本番と同様の索引をすべて作成し、スキーマ統計を正しく入力する必要があります。

- 本番環境で使用されるオプティマイザ・モードによるテスト。

Oracle Databaseの研究開発の取組みは問合せオプティマイザに重点が置かれているため、テスト環境および本番環境の両方で問合せオプティマイザを使用することをお勧めします。

- シングル・ユーザーのパフォーマンスの最初のテスト。

アイドル状態または使用率の低いデータベースで、シングル・ユーザーのテストから開始します。シングル・ユーザーのパフォーマンスが理想的な条件下で許容範囲に達しない場合、実際の条件下で複数のユーザーが許容可能なパフォーマンスを得ることはできません。

- 各SQL文の実行計画の取得。

オプティマイザが最適な実行計画を使用しているか、および各SQL文の相対コストがCPU時間と物理I/O数の観点から把握されているかを検証します。最もチューニングを活用する、頻繁に使用されるトランザクションを識別します。

- 複数のユーザーでのテスト。

ユーザーのワークロードやプロファイルがまだ完全に定量化されていない可能性があるため、このテストを正確に実行するのは困難です。しかし、複数ユーザー環境でDMLトランザクションをテストして、ロックの競合やシリアライズの問題がないことを確認する必要があります。

- 可能なかぎり本番システムに近いハードウェア構成でのテスト。

実際のシステムでのテストは、ネットワーク待機時間、I/Oサブシステム帯域幅およびプロセッサのタイプと速度について

てテストする場合に特に重要です。非現実的なシステムでのテストでは、潜在的なパフォーマンスの問題を見落とす可能性があります。

- 安定した状態でのパフォーマンスの測定

各ベンチマークの実行には開始フェーズが必要です。このフェーズでは、ユーザーがデータベースに接続し、アプリケーションを使用して開始します。このフェーズによって、安定した状態に達する前に、頻繁にキャッシュされるデータをキャッシュに初期化し、1回のみ実行する操作(解析など)を完了しておくことができます。同様に、各ベンチマークの実行終了時には終了フェーズが必要です。このフェーズでは、リソースをシステムから解放し、ユーザーはアプリケーションを終了してデータベースから接続を切断します。

関連項目:

- パフォーマンス・チューニング、ワークロード・テスト、モデリングおよび実装の詳細は、[『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
- ADDMおよびSQLチューニング・アドバイザの詳細は、それぞれ[「自動データベース診断モニター \(ADDM\)」](#)および[「SQLアドバイザとメモリー・アドバイザ」](#)を参照してください。
- SQLチューニング・アドバイザを使用したSQL文のチューニングの詳細は、『Oracle Database 2日でパフォーマンス・チューニング・ガイド』を参照してください

3.5 クライアント結果キャッシュの使用

トピック:

- [クライアント結果キャッシュの使用について](#)
- [クライアント結果キャッシュの利点](#)
- [クライアント結果キャッシュを使用するガイドライン](#)
- [クライアント結果キャッシュの一貫性](#)
- [クライアント結果キャッシュに関するデプロイメント時設定](#)
- [クライアント結果キャッシュの統計](#)
- [クライアント結果キャッシュの検証](#)
- [クライアント結果キャッシュとサーバー結果キャッシュ](#)
- [クライアント結果キャッシュのデモ・ファイル](#)
- [クライアント結果キャッシュの以前のリリースとの互換性](#)

3.5.1 クライアント結果キャッシュの使用について

C、C++、Java (JDBC-OCI)、PHP、Python、RubyおよびPerlを含むOCIライブラリに基づいて構築されたOracle Databaseドライバおよびアダプタを使用するアプリケーションでは、クライアント結果キャッシュを使用することで、繰返しの間合せレスポンス時間を改善できます。

クライアント結果キャッシュにより、クライアント側でのSQL問合せ(SELECT文)結果セットのクライアント・メモリーへのキャッシュが可能になります。クライアント・プロセスから結果を取り出す処理は、データベースのコールおよび問合せの再実行よりも高速で行

われるため、頻繁な問合せの実行では、結果がキャッシュされている場合パフォーマンスが大幅に向上します。クライアント結果キャッシュは、問合せの処理で使用されていたサーバーCPU時間も短縮するため、サーバーのスケーラビリティが向上します。

スキーマ、SQLテキスト、バインド変数、セッション設定が類似している場合には、複数のセッションのOCI文が、OCIプロセス・メモリにキャッシュされた同じ結果セットに一致することもあります。そうでない場合には、問合せの実行はサーバーに送られます。

クライアント結果キャッシュはアプリケーションに対して透過的であり、結果セット・データのキャッシュは、結果セット・データに影響するセッションまたはデータベースの変更内容と整合性が保たれます。

クライアント結果キャッシュを使用するアプリケーションでは、キャッシュ・ヒットがある問合せのパフォーマンスが向上します。このようなアプリケーションは、クライアントまたは中間層でキャッシュされた同じ結果セットを使用します。

クライアント結果キャッシュは、OCIセッション・プール、OCI接続プール、DRCP、およびOCI透過的アプリケーション・フェイルオーバー(TAF)などのOCI機能と連携します。

クライアント結果キャッシュを使用する場合、アプリケーション・レベルでもOCI文キャッシュまたはキャッシュ文を有効化する必要があります。

関連項目:

- 文キャッシュの詳細は、『Oracle Call Interfaceプログラマーズ・ガイド』を参照してください
- JDBC文キャッシュの詳細は、『Oracle Database JDBC開発者ガイド』を参照してください
- [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](#)
- [Oracle Database JDBC開発者ガイド](#)
- [Oracle Databaseパフォーマンス・チューニング・ガイド](#)
- [『Oracle Database概要』](#)
- ALTER TABLE およびCREATE TABLEの句は、[SQLヒント](#)および[RESULT_CACHE](#)を参照してください。
- [RESULT_CACHE_MODE](#)
- [CLIENT_RESULT_CACHE_STAT\\$](#)ビュー

3.5.2 クライアント結果キャッシュの利点

クライアント結果キャッシュの利点は次のとおりです。

- クライアント結果キャッシュはクライアント上にあるため、キャッシュ・ヒットが起こると、フェッチ(OCIStmtFetch2())と実行(OCIStmtExecute())の呼出しがローカルで処理され、サーバーのラウンド・トリップは不要になります。サーバーのラウンド・トリップが不要になることで、サーバー・リソース(サーバーCPUやサーバーI/Oなど)の使用量が減少し、パフォーマンスが大幅に向上します。
- クライアント結果キャッシュは透過的で一貫性があります。
- クライアント結果キャッシュはあらゆるプロセスに使用可能なため、複数のクライアント・セッションで一致するキャッシュ内の結果セットを同時に使用できます。
- クライアント結果キャッシュにより、各OCIアプリケーションが独自のカスタム結果キャッシュを持つ必要性が最小限に抑えられます。

- クライアント結果キャッシュは次の局面を透過的に管理します。
 - 複数のスレッド、複数の文、複数のセッションによる同時アクセス
 - データベースの変更による影響を受けている、キャッシュされた結果セットの無効化
 - キャッシュされた結果セットのリフレッシュ
 - キャッシュ・メモリー管理
- クライアント結果キャッシュは、JDBC OCI、ODP.NET、OCCI、Pro*C/C++、Pro*COBOLおよびODBCなどの、OCIライブラリを使用するアプリケーションおよびドライバで自動的に使用可能になります。
- クライアント結果キャッシュでは、サーバー・メモリーよりコストがかからないOCIクライアント・メモリーが使用されます。
- クライアントのローカル・キャッシュでは、そのクライアントにより実行される問合せについて参照の局所性が向上します。

関連項目:

『Oracle Call Interfaceプログラマーズ・ガイド』のOCIStmtExecute()とOCIStmtFetch2()を参照してください

3.5.3 クライアント結果キャッシュ使用のガイドライン

クライアント結果キャッシュは、次の3つのレベル(優先度が降順で示されています)で有効化または無効化できます。

1. 問合せ:

特定の問合せについて、SQLヒントでクライアント結果キャッシュを有効化または無効化できます。SQLヒントを追加または変更するには、アプリケーションを変更する必要があります。

2. 表

特定の表に対するすべての問合せについて、アプリケーションを変更せずに表注釈でクライアント結果キャッシュを有効化または無効化できます。

3. セッション

データベース・セッション内のすべての問合せについて、アプリケーションを変更せずにセッション・パラメータでクライアント結果キャッシュを有効化または無効化できます。

上位レベルの設定は下位レベルの設定より優先されます。

クライアント結果キャッシュは、読取り専用の表と、ほとんど更新されない表(めったに更新されない表)での問合せに対してのみ、有効化することをお薦めします。つまり、次の場合にクライアント結果キャッシュを有効化します。

- 読取り専用の表と、ほとんど読み取られない表の問合せに対する問合せレベルのみ
- 読取り専用の表と、ほとんど読み取られない表に対する表レベルのみ
- 読取り専用の表、またはほとんど読み取られない表のみを問い合わせるアプリケーションを実行中のセッション・レベルのみ

結果セットが大きい、またはバインド値のセット数が多い問合せに対してクライアント結果キャッシュを有効にすると、大量のクライアント結果キャッシュ・メモリーが使用されることがあります。バインド値の各セットでは、(同じSQLテキストに対して)異なるキャッシュ結果セットが作成されます。

クライアントの結果キャッシュが問合せに対して有効な場合、クライアント、サーバー、またはその両方で結果セットをキャッシュできます。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(デフォルトで有効)が無効化されている場合でも有効化できます。

OCIの場合、すべてのOCI文ハンドル・コールにおける最初のOCIStmtExecute()のコールは、キャッシュ内に有効な結果セットがある場合でも、必ずサーバーに対して行われます。各文ハンドルをキャッシュ内の結果セットと照合できるようにするには、OCIStmtExecute()をコールする必要があります。アプリケーションがOCI文ハンドルについて独自の文キャッシュを持つようにするか、またはOCI文キャッシュを使用し、OCIStmtPrepare2()が一度実行されたOCI文ハンドルを戻せるようにすることをお勧めします。同一または異なるセッションからの複数のOCI文ハンドルは、キャッシュ内の同一の結果セットからデータを同時にフェッチできます。

OCIに対して結果セットをキャッシュする場合は、このキャッシュ結果セットを透過的に作成するOCIStmtExecute()コールまたはOCIStmtFetch2()コールで、ORA-01403(「データが見つかりません。」)が返されるまで行をフェッチする必要があります。ローカルにキャッシュされた結果セットと一致する後続のOCIStmtExecute()またはOCIStmtFetch2()のコールでは、最後までフェッチする必要はありません。

関連項目:

- [SQLヒント](#)
- [表注釈](#)
- [セッション・パラメータ](#)
- [結果キャッシュ・モードの使用例](#)
- [OCIStmtExecute\(\)](#)
- [OCIStmtPrepare2\(\)](#)
- [OCIStmtFetch2\(\)](#)

トピック:

- [SQLヒント](#)
- [表注釈](#)
- [セッション・パラメータ](#)
- [有効な表結果キャッシュ・モード](#)
- [有効な表結果キャッシュ・モードの表示](#)
- [結果キャッシュ・モードの使用例](#)
- [クライアント結果キャッシュで結果がキャッシュされない問合せ](#)

3.5.3.1 SQLヒント

SQLヒントRESULT_CACHEまたはNO_RESULT_CACHEは単一の問合せに適用され、そのためにクライアント結果キャッシュが有効化または無効化されます。これらのヒントは、両方の表注釈とRESULT_CACHE_MODEサーバー初期化パラメータよりも優先されます。

OCIの場合、OCIStmtPrepare()コールとOCIStmtPrepare2()コールに渡されるSQLテキストで、SQLヒントの/*+ result_cache */または/*+ no_result_cache */を設定する必要があります。

JDBC OCIの場合、問合せ文字列の一部として、SQLヒント/*+ result_cache */または/*+ no_result_cache */が問合せ(SELECT文)に含まれます。

関連項目:

- SQLヒントの概要は、[『Oracle Database SQL言語リファレンス』](#)を参照してください
- 『Oracle Call Interfaceプログラマーズ・ガイド』のOCIStmtPrepare(), OCIStmtPrepare2()を参照してください
- JDBCでのSQLヒントの詳細は、[『Oracle Database JDBC開発者ガイド』](#)を参照してください

3.5.3.2 表注釈

表注釈は特定の表におけるすべての問合せのクライアント結果キャッシュを有効化または無効化します。表注釈はRESULT_CACHE_MODEサーバー初期化パラメータよりも優先されますが、SQLヒント/*+ RESULT_CACHE */および/*+ NO_RESULT_CACHE */は表注釈よりも優先されます。

RESULT_CACHE句を使用して、CREATE TABLEまたはALTER TABLE文のどちらかで表に注釈を付けます。クライアント結果キャッシュを有効にするには、RESULT_CACHE (MODE FORCE)を指定します。無効にするには、RESULT_CACHE (MODE DEFAULT)を使用します。

[表3-1](#)に、表注釈の結果キャッシュ・モードをまとめます。

表3-1 表注釈の結果キャッシュ・モード

モード	説明:
DEFAULT	デフォルト値です。結果キャッシュは表レベルでは決定されません。この値を使用して、表注釈をクリアできます。
FORCE	問合せのすべての表名がこの設定になっている場合は、問合せにNO_RESULT_CACHE ヒントが指定されていないかぎり、その問合せ結果は常にキャッシュの対象となります。問合せで指定した 1 つ以上の表がDEFAULT に設定されている場合、この問合せにおける有効な表の注釈はDEFAULT です。

関連項目:

- ALTER TABLEおよびCREATE TABLEのRESULT_CACHE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
- JDBCでの表注釈の詳細は、[『Oracle Database JDBC開発者ガイド』](#)を参照してください

3.5.3.3 セッション・パラメータ

セッション・パラメータRESULT_CACHE_MODEは、データベース・セッションにおけるすべての問合せのクライアント結果キャッシュを有効化または無効化します。RESULT_CACHE_MODEは、特定の表については表注釈より、特定の問合せについてはSQLヒントより優先度が下がります。

RESULT_CACHE_MODEは、サーバー・パラメータ・ファイル(init.ora)、あるいはALTER SESSIONまたはALTER SYSTEM文のいずれかに設定できます。クライアント結果キャッシュを有効にするにはRESULT_CACHE_MODEをFORCEに設定し、無効にするにはRESULT_CACHE_MODEをMANUALに設定します。

関連項目:

RESULT_CACHE_MODEの詳細は、『Oracle Databaseリファレンス』を参照してください

3.5.3.4 有効な表結果キャッシュ・モード

表の有効な結果キャッシュ・モードは、[表3-2](#)に示すように、表注釈の結果キャッシュ・モードとRESULT_CACHE_MODEセッション・パラメータ設定の双方によって異なります。

表3-2 有効な結果キャッシュ表モード

表注釈の結果キャッシュ・モード	RESULT_CACHE_MODE = MANUAL	RESULT_CACHE_MODE = FORCE
FORCE	FORCE	FORCE
DEFAULT	MANUAL	FORCE

有効なモードがFORCEの場合、問合せにNO_RESULT_CACHEヒントがないかぎり全問合せが結果キャッシュの対象となりますが、実際の結果キャッシュはクライアントおよびサーバーのキャッシュの内部制限、問合せの再利用可能性、およびクライアント結果キャッシュで使用可能な領域によって異なります。

有効な表結果キャッシュ・モードFORCEは、双方が単なるリクエストであるという点で、SQLヒントRESULT_CACHEに類似しています。

3.5.3.5 有効な表結果キャッシュ・モードの表示

1つ以上の表の結果キャッシュ・モードを表示するには、*_TABLES静的データ・ディクショナリ・ビューのRESULT_CACHE列を参照してください。たとえば:

- この問合せは表Tの結果キャッシュ・モードを表示します。

```
SELECT result_cache FROM all_tables WHERE table_name = 'T'
```

- この問合せは所有するすべてのリレーショナル表の結果キャッシュ・モードを表示します。

```
SELECT table_name, result_cache FROM user_tables
```

結果キャッシュ・モードがDEFAULTの場合、表に注釈は付けられません。

結果キャッシュ・モードがFORCEの場合、表はCREATE TABLEまたはALTER TABLE文のRESULT_CACHE (MODE FORCE)句で注釈が付けられます。

結果キャッシュ・モードがMANUALの場合、セッション・パラメータRESULT_CACHE_MODEは、サーバー・パラメータ・ファイル (init.ora)、あるいはALTER SESSIONまたはALTER SYSTEM文のいずれかでMANUALに設定されています。

関連項目:

静的データ・ディクショナリ・ビュー*_TABLESの詳細は、『[Oracle Databaseリファレンス](#)』を参照してください

3.5.3.6 結果キャッシュ・モードの使用例

SQLヒントが表注釈およびRESULT_CACHE_MODEセッション・パラメータより優先される場合の例を次に示します。

- emp表がALTER TABLE emp RESULT_CACHE (MODE FORCE)として注釈され、セッション・パラメータがデフォルト値のMANUALである場合は、empに対する問合せは結果キャッシュの対象となります。
ただし、問合せSELECT /*+ no_result_cache */ empno FROM empの結果はキャッシュされません。これは、SQLヒントが表注釈およびセッション・パラメータより優先されているためです。
- emp表が注釈されていないか、ALTER TABLE emp RESULT_CACHE (MODE DEFAULT)として注釈され、セッション・パラメータがデフォルト値のMANUALである場合は、empに対する問合せは結果キャッシュされません。
ただし、問合せSELECT /*+ result_cache */ * FROM empの結果はキャッシュの対象となります。これは、SQLヒントが表注釈およびセッション・パラメータより優先されているためです。
- セッション・パラメータRESULT_CACHE_MODEがFORCEに設定され、それより優先される表注釈もSQLヒントもない場合は、すべての表に対するすべての問合せが問合せキャッシュの対象となります。

3.5.3.7 クライアント結果キャッシュで結果がキャッシュされない問合せ

次の問合せの結果は、(問合せにRESULT_CACHEヒントが含まれる場合でも)クライアント結果キャッシュではキャッシュされません。

- 次のいずれかが含まれる問合せ：
 - リモート・オブジェクト
 - 選択リストの複合型
 - PL/SQL関数
- スナップショットベースの問合せ
- フラッシュバック問合せ
- シリアライズ可能の読取り専用トランザクション内で実行される問合せ
- 仮想プライベート・データベース(VPD)ポリシーが有効な表の問合せ

サーバーの結果キャッシュ機能が有効である場合、こうした問合せはデータベースにキャッシュされる可能性があります。詳細は、[『Oracle Database概要』](#)を参照してください。

3.5.4 クライアント結果キャッシュの一貫性

クライアント結果キャッシュでは、結果セットに影響を与えるすべてのデータベースまたはセッション状態の変更と一致するように、結果セットを透過的に維持します。

トランザクションにより、キャッシュされた結果セットの作成に使用されたデータベース・オブジェクトのデータまたはメタデータが変更された場合、そのキャッシュされた結果セットの無効化が、サーバーへの次のラウンド・トリップ時に送信されます。アプリケーションで一定時間にわたりデータベース・コールが行われない場合、こうした無効化の有無を確認するため、クライアント結果キャッシュの遅延設定により次のOCIStmtExecute()コールによるデータベース・コールが強制的に実行されます。

データベースの無効化に関連するキャッシュされた結果セットはただちに無効化されます。キャッシュされた結果セットからフェッチしているOCI文ハンドルは、その無効化の受信時にフェッチを続行できますが、後続のOCIStmtExecute()コールは結果セットと照合されません。

クライアント結果キャッシュに使用可能な領域がある場合、プロセスによる次のOCIStmtExecute()コールにより、新規結果セットをキャッシュできます。クライアント結果キャッシュでは、未使用のメモリーが定期的に再生されます。

セッションにオープン・トランザクションがある場合、OCIでは、このトランザクションで変更されたデータベース・オブジェクトを参照する問合せが、クライアント結果キャッシュではなくサーバーに対して行われます。

一貫性を保つこのメカニズムにより、クライアント結果キャッシュはコミットされたデータベースの変更に常に近い状態になります。キャッシュできない問合せ(DML、OCILOBコールなど)のため、データベース・ラウンド・トリップに伴う比較的頻繁なコールをアプリケーションが実行する場合、これらのコールによりクライアント結果キャッシュがデータベースの変更と整合するように透過的に保たれます。

表が変更されるとき、トリガーにより別の表が変更されることがあります。クライアント結果キャッシュは、このような変化の影響を受けやすい性質があります。

セッション状態が変わる、たとえばNLSセッション・パラメータが変更されると、問合せ結果が異なる場合があります。クライアント結果キャッシュはこのような変化の影響を受けやすく、それ以降に問合せを実行すると、正しい結果セットが返されます。ただし、プロセス内の他のセッションが一致するように、クライアント結果キャッシュは現在のキャッシュされた結果セットを維持します(そして、無効化しません)。他のプロセスが一致しない場合、これらの結果セットはしばらくして「支配」状態になります。新しいセッション状態に対応する結果セットがキャッシュされます。

アプリケーションにとって、問合せ結果セットに影響するデータベースおよびセッション状態の変化を追跡することは煩雑でエラーも起きやすい処理ですが、クライアント結果キャッシュはそのような変化と結果セットの一貫性を透過的に保ちます。

クライアント結果キャッシュでは、クライアントにおけるスレッドのサポートは必要ありません。

関連項目:

『Oracle Call Interfaceプログラマーズ・ガイド』のOCIStmtExecute()を参照してください

3.5.5 クライアント結果キャッシュに関するデプロイメント時設定

アプリケーションをデプロイする際は、(アプリケーションに変更を加えることなく)クライアント結果キャッシュについて次の要素を設定できます。

- サーバー初期化パラメータ
- クライアント構成パラメータ
- 表注釈
- RESULT_CACHE_MODEセッション・パラメータ

関連項目:

- [サーバー初期化パラメータ](#)
- [クライアント構成パラメータ](#)
- [表注釈](#)
- [セッション・パラメータ](#)
- [クライアント側デプロイメント・パラメータ](#)

3.5.5.1 サーバー初期化パラメータ

アプリケーションのデプロイ時に、クライアント結果キャッシュに設定するサーバー初期化パラメータを次に示します。

- [COMPATIBLE](#)
- [CLIENT_RESULT_CACHE_SIZE](#)
- [CLIENT_RESULT_CACHE_LAG](#)

3.5.5.1.1 COMPATIBLE

互換性を維持しておくOracle Databaseのリリースを指定します。クライアント結果キャッシュを有効にするには、COMPATIBLEが11.0.0.0以上である必要があります。ビューのクライアント結果キャッシュを有効にするには、COMPATIBLEが11.2.0.0以上である必要があります。

3.5.5.1.2 CLIENT_RESULT_CACHE_SIZE

OCIクライアント・プロセスごとにクライアント結果セット・キャッシュの最大サイズを指定します。デフォルト値は0(ゼロ)であり、クライアント結果キャッシュが無効であることを示します。クライアント結果キャッシュを有効にするには、CLIENT_RESULT_CACHE_SIZEを32768バイト(32KB)以上に設定します。

CLIENT_RESULT_CACHE_SIZEによってクライアント結果キャッシュがサーバーで有効化されている場合、その値はsqlnet.ora構成パラメータOCI_RESULT_CACHE_MAX_SIZEによって上書きできます。クライアント結果キャッシュがサーバーで無効化されている場合、OCI_RESULT_CACHE_MAX_SIZEは無視され、クライアント結果キャッシュはクライアントで有効化できません。

すべてのOracle Real Application Clusters (Oracle RAC)ノードでクライアント結果キャッシュを有効にするか、すべてのOracle RACノードでクライアント結果キャッシュを無効にすることをお勧めします。そうしないと、1つのクライアント・プロセス内で、一部のセッションではキャッシュが有効で、他のセッションではキャッシュが無効である(したがってサーバーから最新の結果を取得する)可能性があります。この組合せによって、データベースの一貫性のないビューがアプリケーションに提供されることがあります。

CLIENT_RESULT_CACHE_SIZEは静的パラメータです。したがって、ALTER SYSTEM文を使用してCLIENT_RESULT_CACHE_SIZEの値を変更する場合は、SCOPE = SPFILE句を含める必要があり、変更を有効にするにはデータベースの再起動が必要です。

CLIENT_RESULT_CACHE_SIZEの最大値は次の値以上になります。

- 使用可能なクライアント・メモリー
- ((キャッシュされる結果セットの可能な数) * (結果セット内の行の平均サイズ) * (結果セット内の行の平均数))
- 2ギガバイト(GB)
2GBより大きい値を指定すると、値は2GBになります。

ノート:



データベースの作成中は、CLIENT_RESULT_CACHE_SIZE パラメータを設定しないでください。エラーになる場合があります。

3.5.5.1.3 CLIENT_RESULT_CACHE_LAG

結果セットに影響するデータベース内の変更からクライアント結果キャッシュが遅延できる最大時間をミリ秒で指定します。デフォ

ルトは、3000ミリ秒です。

CLIENT_RESULT_CACHE_LAGは静的パラメータです。したがって、ALTER SYSTEM文を使用してCLIENT_RESULT_CACHE_LAGの値を変更する場合は、SCOPE = SPFILE句を含める必要があり、変更を有効にするにはデータベースの再起動が必要です。

3.5.5.2 クライアント構成パラメータ

クライアント構成パラメータはオプションですが、設定されていると、サーバー初期化ファイルinit.ora内の等価のパラメータを上書きします。

クライアント構成パラメータは、oraaccess.xmlファイル、sqlnet.oraファイル、あるいはその両方で設定できます。等価のパラメータを両方のファイルで設定した場合、oraaccess.xml設定の方が、対応するsqlnet.oraの設定より優先されます。パラメータがoraaccess.xmlで設定されていないと、プロセスはsqlnet.ora内の設定を検索します。

クライアント構成パラメータをoraaccess.xmlおよびsqlnet.oraの両方で設定できる場合は、oraaccess.xmlでパラメータを設定することをお勧めします。ただし、ネットワーク構成の場合は、oraaccess.xmlがネットワーク・レベル設定をサポートしないため、sqlnet.oraがプライマリ・ファイルです。

[表3-3](#)では、等価のoraaccess.xmlおよびsqlnet.oraクライアント構成パラメータについて説明します。

表3-3 クライアント構成パラメータ(オプション)

oraaccess.xmlパラメータ	sqlnet.oraパラメータ	説明:
max_size	OCI_RESULT_CACHE_MAX_SIZE	各プロセスのクライアント結果キャッシュの最大サイズ(バイト数)。 sqlnet.ora で 32768 未満のサイズを指定すると、sqlnet.ora を読み取るクライアント・プロセスのクライアント結果キャッシュが無効になります。
max_rset_size	OCI_RESULT_CACHE_MAX_RSET_SIZE	各プロセスのクライアント結果キャッシュ内の結果セットの最大サイズ(バイト数)。
max_rset_rows	OCI_RESULT_CACHE_MAX_RSET_ROWS	各プロセスのクライアント結果キャッシュ内の結果セットの最大サイズ(行数)。

クライアントでは結果キャッシュの遅延を設定できません。

関連トピック

- [Oracle Call Interfaceプログラマーズ・ガイド](#)

3.5.6 クライアント結果キャッシュの統計

OCIクライアントからサーバーへのラウンド・トリップで、OCIはクライアント結果キャッシュ統計を定期的にサーバーに送信します。この統計は、CLIENT_RESULT_CACHE_STATS\$ビューで示され、正常にキャッシュされた結果セット数、キャッシュ・ヒット数

および無効化されたキャッシュ内の結果セット数などが含まれます。問合せのキャッシュ・ミス数は、クライアント結果キャッシュ統計内の作成数の数以上になります。より正確には、キャッシュ・ミス数はサーバーの自動ワークロード・リポジトリ(AWR)レポート内のサーバー実行数と等しくなります。

関連項目:

- CLIENT_RESULT_CACHE_STAT\$ビューの詳細は、『Oracle Database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を実行しているセッションのクライアント・プロセスIDおよびセッションIDについては、『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。

3.5.7 クライアント結果キャッシュの検証

クライアント結果キャッシュの検証手段を次に示します。

- [実行時間の測定](#)
- [V\\$MYSTATの問合せ](#)
- [V\\$SQLAREAの問合せ](#)

3.5.7.1 実行時間の測定

まず、RESULT_CACHEヒントなしで問合せの実行時間を測定します。次に、RESULT_CACHEヒントを問合せに追加して、実行時間を再測定します。実行時間の差異がパフォーマンスの向上分に該当します。

3.5.7.2 V\$MYSTATの問合せ



ノート:

V\$MYSTAT ビューを問い合わせるには、そのビューでの SELECT 権限が必要です。

1. 次の問合せを5回実行します。

```
SELECT count(*) FROM table_name
```

2. V\$MYSTATを問い合わせます。

```
SELECT * FROM V$MYSTAT
```

3. 次の問合せを5回実行します。

```
SELECT /*+ result_cache */ count(*) FROM table_name
```

問合せ結果はキャッシュされているため、このステップで必要なクライアントおよびサーバー間のラウンド・トリップ数はステップ1の場合より少なくなります。

4. V\$MYSTATを問い合わせます。

```
SELECT * FROM V$MYSTAT
```

この問合せの列の値を、ステップ2の問合せの列の値と比較します。

ステップ3の問合せにヒントを追加するかわりに、ステップ3で表注釈RESULT_CACHE (MODE FORCE)を

table_nameに追加してから、ステップ1の問合せを数回実行できます。

3.5.7.3 V\$SQLAREAの問合せ

ノート:



V\$SQLAREA ビューを問い合わせるには、そのビューでの SELECT 権限が必要です。

1. 次の問合せを5回実行します。

```
SELECT count(*) FROM table_name
```

2. V\$SQLAREAを問い合わせます。

```
SELECT executions, fetches, parse_calls FROM V$SQLAREA  
WHERE sql_text LIKE '% FROM table_name'
```

3. 次の問合せを5回実行します。

```
SELECT /*+ result_cache */ count(*) FROM table_name
```

4. V\$SQLAREAを問い合わせます。

```
SELECT executions, fetches, parse_calls FROM V$SQLAREA  
WHERE sql_text LIKE '% FROM table_name'
```

この問合せの列executions、fetchesおよびparse_callsの値を、ステップ2の問合せの列の値と比較します。実行時間の差異がパフォーマンスの向上分に該当します。

ステップ3の問合せにヒントを追加するかわりに、ステップ3で表注釈RESULT_CACHE (MODE FORCE)をtable_nameに追加してから、ステップ1の問合せを数回実行できます。

3.5.8 クライアント結果キャッシュとサーバー結果キャッシュ

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。クライアント結果キャッシュでは、トップレベルのSQL問合せの結果がOCIクライアント・メモリーにキャッシュされますが、サーバー結果キャッシュでは、結果セットおよび問合せの断片がサーバーSGAメモリーにキャッシュされます。

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関係なく有効化できますが、どちらも結果キャッシュSQLヒント、表注釈、セッション・パラメータRESULT_CACHE_MODEを共有します。[表3-4](#)に、結果キャッシュ・パラメータ、PL/SQLパッケージDBMS_RESULT_CACHEおよび結果キャッシュ・ビューに対する結果キャッシュの関連付けを示します。

関連項目:

[『Oracle Database概要』](#)

表3-4 クライアント結果キャッシュとサーバー結果キャッシュの設定

パラメータ、PL/SQLパッケージおよびデータベース・ビュー	結果キャッシュの関連付け
CLIENT_RESULT_CACHE_*パラメータ:	クライアント結果キャッシュ

パラメータ、PL/SQLパッケージおよびデータベース・ビュー	結果キャッシュの関連付け
● CLIENT_RESULT_CACHE_SIZE ● CLIENT_RESULT_CACHE_LAG	
SQL ヒント: ● RESULT_CACHE ● NO_RESULT_CACHE	クライアント結果キャッシュ、 サーバー結果キャッシュ
sqlnet.ora OCI_RESULT_CACHE*パラメータ: ● OCI_RESULT_CACHE_MAX_SIZE ● OCI_RESULT_CACHE_MAX_RSET_SIZE ● OCI_RESULT_CACHE_MAX_RSET_ROWS	クライアント結果キャッシュ
CLIENT_RESULT_CACHE_STATS\$ビュー	クライアント結果キャッシュ
RESULT_CACHE_MODE パラメータ	クライアント結果キャッシュ、 サーバー結果キャッシュ
他のすべての RESULT_CACHE_*パラメータ(RESULT_CACHE_MAX_SIZE など)	サーバー結果キャッシュ
DBMS_RESULT_CACHE パッケージ	サーバー結果キャッシュ
V\$RESULT_CACHE_*および GV\$RESULT_CACHE_*ビュー (V\$RESULT_CACHE_STATISTICS および GV\$RESULT_CACHE_MEMORY など)	サーバー結果キャッシュ
CREATE TABLE 注釈	クライアント結果キャッシュ、 サーバー結果キャッシュ
ALTER TABLE 注釈	クライアント結果キャッシュ、 サーバー結果キャッシュ

3.5.9 クライアント結果キャッシュ・デモ・ファイル

OCIアプリケーションの場合、クライアント結果キャッシュのデモ用ファイルは、cdemoqc.sql、cdemoqc.cおよびcdemoqc2.cです(オペレーティング・システムのdemoディレクトリ内)。

3.5.10 クライアント結果キャッシュの以前のリリースとの互換性

クライアント結果キャッシュを使用するには、アプリケーションのリンクをOracle Database 11gリリース1 (11.1)以上のクライア

ント・ライブラリに再設定し、Oracle Database 11gリリース1 (11.1)以上のデータベース・サーバーに接続する必要があります。クライアント結果キャッシュは、JDBC Type IIドライバ、OCCI、Pro*C/C++およびODP.NETを含むすべてのOCIアプリケーションで使用できます。OCIドライバは、SQLヒントRESULT_CACHEをOCIStmtPrepare()およびOCIStmtPrepare2()コールに自動的に渡します。

関連項目:

『Oracle Call Interfaceプログラマーズ・ガイド』のOCIStmtPrepare()、OCIStmtPrepare2()を参照してください

3.6 文キャッシング

文キャッシュは、セッションごとの文のキャッシュを確立して管理する機能です。サーバーでは、文キャッシュによって文を再解析しなくてもカーソルを使用できるので、繰り返して文を解析する必要がなくなります。文キャッシュは、接続プーリングおよびセッション・プーリングの両方とともに使用でき、パフォーマンスおよびスケーラビリティが向上します。文キャッシュは、セッション・プーリングを使用せずにOCIで、あるいは接続プーリングを使用せずにOCCIで使用でき、またJDBCインタフェースやODP.NETインタフェースでも使用できます。また、動的SQL文を使用するOracleプリコンパイラ・アプリケーション、たとえばPro*C/C++やProCOBOLなどでは、動的SQL文キャッシュも使用できます。

JDBCインタフェースでは、暗黙的および明示的な文キャッシュを互いに関係なく有効または無効にできます。どちらかを使用するか、どちらも使用しないか、または両方とも使用することもできます。暗黙的文キャッシュと明示的文キャッシュは、各接続に対して1つのキャッシュを共有します。特定の文に対して暗黙的なキャッシュを無効にすることも可能です。

関連項目:

- OCIで文キャッシュを使用する方法の詳細とガイドラインは、『Oracle Call Interfaceプログラマーズ・ガイド』を参照してください
- OCCIにおける文キャッシュの詳細は、『Oracle C++ Call Interfaceプログラマーズ・ガイド』を参照してください
- 文キャッシュの使用方法の詳細は、『Oracle Database JDBC開発者ガイド』を参照してください
- ODP.NETアプリケーションで文キャッシュを使用する方法の詳細は、[『Oracle Data Provider for .NET開発者ガイドfor Microsoft Windows』](#)を参照してください
- 動的SQL文を利用するプリコンパイラ・アプリケーションで動的SQL文キャッシュを使用する方法の詳細は、[『Oracle Database Oracleプリコンパイラのためのプログラマーズ・ガイド』](#)、[『Pro*C/C++プログラマーズ・ガイド』](#)、[『Pro*COBOLプログラマーズ・ガイド』](#)を参照してください

3.7 OCIクライアント文キャッシュの自動チューニング

OCIクライアント文キャッシュの自動チューニングは、中間層アプリケーションのOCIクライアント・セッション機能を最適化することで、OCIアプリケーションを変更することなく、パフォーマンスを向上できます。

自動チューニングを使用しないと、OCIクライアントの文キャッシュ・サイズ設定は最適状態に及ばなくなる可能性があります。たとえば、ワークロードの変更によってSQL文の別のワーキング・セットが発生する場合です。サイズが小さすぎると、ネットワーク・アクティビティが過剰になり、サーバーでの解析量が増大します。サイズが大きすぎると、メモリーの使用量が超過します。クライアント・アプリケーションが、キャッシュ・サイズを最適に維持することは困難です。

自動チューニングでは、OCI文キャッシュ・サイズが定期的に自動で再構成されることで、このようなパフォーマンス上の潜在的な

問題が解決されます。

自動チューニングは、OCI文キャッシュを再構成するオプションを提供するデプロイメント時間設定を指定することで行われます。これらの設定は、クライアントのoraaccess.xmlファイルで接続文字列に基づくデプロイ設定として用意されており、OCI機能のユーザー構成に対するプログラム設定より優先されます。

中間層アプリケーションの開発者とデータベース管理者(DBA)は、それぞれのシステムでOCIクライアントの文キャッシュ・パラメータ設定を自動チューニングすることによって、パフォーマンス上の問題の診断と修正に要する時間と労力の軽減を期待できます。

関連項目:

- [Oracle Call Interfaceプログラマーズ・ガイド](#)

3.8 クライアント側デプロイメント・パラメータ

Oracle Database 12c リリース1 (12.1.0.1)以降では、OCIデプロイメント・パラメータを新しい構成ファイル(oraaccess.xml)で使用できます。

関連項目:

[Oracle Call Interfaceプログラマーズ・ガイド](#)

3.9 問合せ変更通知の使用

連続問合せ通知(CQN)を使用すると、クライアント・アプリケーションがデータベースに問合せを登録し、オブジェクトに対するDMLまたはDDL変更の通知(オブジェクト変更通知(OCN))、あるいは問合せに関連付けられた結果セット変更の通知(問合せ結果変更通知(QRCN))を受信できます。データベースは、DMLトランザクションまたはDDLトランザクションのコミット時に通知を発行します。

CQN登録により、1つ以上の問合せが通知タイプ(OCNまたはQRCN)および通知ハンドラに関連付けられます。CQN登録は、次の方法で作成できます。

- PL/SQLインタフェース

PL/SQLインタフェースを使用する場合、通知ハンドラはサーバー側PL/SQLストアド・プロシージャです。PL/SQL登録は、PHPのような非スレッド型の言語およびシステムで使用できます。PHPの場合、PL/SQLリスナーがデータベース通知を受信した段階でPHPコールバックを呼び出します。

- Oracle Call Interface(OCI)

OCIを使用する場合、通知ハンドラはクライアント側Cコールバック・プロシージャです。

- Javaデータベース接続(JDBC)インタフェース

JDBCインタフェースを使用する場合は、JDBCドライバによってサーバー上に登録が作成されます。JDBCドライバは、サーバーからの通知を(専用チャンネルを介して)リスニングする新しいスレッドを起動し、それをJavaイベントに変換してから、この登録に登録されているすべてのリスナーに通知を送ります。

関連項目:

- この機能の詳細と、PL/SQLおよびOCIインタフェースを使用してCQN登録を作成する方法については、[「連続問合せ通知\(CQN\)の使用」](#)を参照してください
- サーバーでCQN登録にJDBCを使用する方法の詳細は、『Oracle Database JDBC開発者ガイド』を参照してください

3.10 データベース常駐接続プールの使用

データベース常駐接続プール(DRCP)は、データベース接続を取得し、比較的短時間データベースで作業した後、接続を解放するような一般的なWebアプリケーション使用に対して、データベース・サーバーの接続プールを提供します。

トピック:

- [データベース常駐接続プールの理解](#)
- [DRCPの構成](#)
- [プロキシ・セッションの共有](#)
- [DRCPでのJDBCの使用](#)
- [DRCPでのOCIセッション・プールAPIの使用](#)
- [セッション純正値および接続クラス](#)
- [データベース常駐接続プールの開始](#)
- [DRCPの有効化](#)
- [OCIアプリケーションにおけるDRCPのスケラビリティの利用](#)
- [JavaアプリケーションにおけるDRCPのスケラビリティの利用](#)
- [DRCPの使用に関するベスト・プラクティス](#)
- [互換性と移行](#)
- [DRCPの制限事項](#)
- [カスタム・プールでのDRCPの使用](#)
- [明示的なステートフルまたはステートレス・セッションのマーキング](#)
- [Oracle Real Application ClustersでのDRCPの使用方法](#)
- [プラグブル・データベースでのDRCPの使用](#)
- [DRCPとData Guard](#)

3.10.1 データベース常駐接続プールの理解

DRCPプール・サーバーのプロセスは、それぞれが専用サーバー・プロセスとデータベース・セッションを組み合わせたものと同等であり、これをプール・サーバーと呼びます。プール・サーバーは、同じホストまたは複数のホストで実行されている複数のアプリケーションによる共有が可能です。接続ブローカー・プロセスは、プール・サーバーをデータベース・インスタンスのレベルで管理します。

DRCPはアプリケーション実行時に選択される構成可能な機能です。従来型とDRCPベースの接続アーキテクチャを同時に使用できるようになります。

DRCPは、中間層接続プーリングを実行できないマルチプロセス・シングル・スレッド・アプリケーション・サーバー (PHPやApache

など)を含むアーキテクチャに特に有効です。DRCPは、数百から数千単位のWebサーバーまたは中間層がデータベース・アクセスとクライアント側プールを必要とする大規模なWebデプロイメントでもきわめて効果的です(Javaのようなマルチスレッド・システムと言語でも)。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。多数の接続に伴ってデータベースWebアプリケーションのスケールアップが必要な場合、接続プーリング・ソリューションとしてDRCPが適しています。

DRCPは、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。また、DRCPを使用すると、同じ中間層ホスト上の中間層プロセス間でも、複数の中間層ホスト間でも、あるいは複数の中間層(Webサーバー、コンテナ)間でもデータベース接続を共有でき、異なる言語で書かれたアプリケーションにも対応できます。この共有によって、大量のクライアント接続をサポートするために必要となるデータベース・リソースが大幅に減少するため、データベース層のメモリ・フットプリントが縮小し、中間層とデータベース層の両方のスケーラビリティが向上します。すぐに使用できるサーバーのプールを保持することで、クライアント接続の作成と解放のコストも削減されます。

クライアントはDRCPから接続を取得し、これは接続ブローカと呼ばれるOracle Databaseバックグラウンド・プロセスに接続されます。接続ブローカはプール機能を実装し、クライアントからの永続インバウンド接続間でプール・サーバーを多重化します。

クライアントでデータベース・アクセスが必要な場合、接続ブローカがプールからサーバー・プロセスを取得してクライアントに渡します。クライアントはその後、サーバーに直接接続されます。サーバーがクライアント・プロセスを実行すると、サーバー・プロセスはプールに戻され、クライアントからの接続は、接続ブローカに対してクライアント・プロセスからの永続インバウンド接続としてリストアップされます。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。このセッションは、システム・グローバル領域(SGA)ではなくプログラム・グローバル領域(PGA)にユーザー・グローバル領域(UGA)を格納するため、クライアントはアクティビティの検出時に透過的に接続を再確立できます。

DRCPは通常、接続数の多いアプリケーションで多用されます。共有サーバーは接続数が中程度のアプリケーションに効果的であり、専用セッションは接続数が少ないアプリケーションに適しています。しきい値サイズは、データベース・ホストで利用できるメモリ量によって異なります。

DRCPには次の利点があります。

- DRCPでは複数のクライアント・アプリケーション間と、中間層アプリケーション・サーバー間でリソース共有が可能です。
- DRCPではデータベース・ホストでリソース使用量が減少するため、データベースとアプリケーションのスケラビリティが向上します。

クライアント側接続プーリングおよび共有サーバーと比較した場合は次のとおりです。

- DRCPは、クライアント側接続プーリングによって用意されるデータベース・サーバーへの直接接続を提供します(クライアント側接続プーリングと同様だが、共有サーバーとは異なる)。
- DRCPは、データベース・サーバーをプールできます(クライアント側接続プーリングおよび共有サーバーと同様)。
- DRCPは、セッションをプールできます(クライアント側接続プーリングと同様だが、共有サーバーとは異なる)。
- DRCPは、中間層の境界間で接続を共有できます(クライアント側接続プーリングとは異なる)。

DRCPは、PDB環境とPDB以外の環境の両方における共有接続をサポートします。

DRCPは、データベース・リソースの使用率を最小限に抑えたまま、多数の接続が必要となる環境でのスケラビリティ要件に対応できる、独自の接続プーリング・ソリューションです。

関連項目:

- DRCPアーキテクチャの詳細は、『Oracle Database概要』を参照してください

- 切替えサービス拡張の詳細は、[『Oracle Database管理者ガイド』](#)を参照してください。

3.10.2 DRCPの構成

DRCPはデフォルトでインストールされますが、DBAはDBMS_CONNECTION_POOLパッケージを使用してDRCPを起動および構成する必要があります。構成オプションには、プール・サーバーの最小数と最大数、接続ブローカの数、各接続ブローカが処理する接続の最大数などがあります。

Oracle Database 12c リリース2 (12.2.0.1)以降では、新しいMAX_TXN_THINK_TIMEパラメータを使用して、オープンになっているトランザクションのあるクライアントが終了する前に許可される非アクティブの最大時間を指定します。これは、トランザクションのないクライアントよりも長い時間を許可するように設定できます(MAX_THINK_TIMEパラメータ値を指定)。これにより、効率的なプールの再利用が可能ですが、不完全なトランザクションの終了に長時間かかります。

OCIセッション・プールAPIは、DRCPと相互運用するために拡張されています。

関連項目:

- DBMS_CONNECTION_POOLパッケージの詳細は、[『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』](#)を参照してください。
- データベース常駐接続プールの接続プールの構成の詳細は、[『Oracle Database管理者ガイド』](#)を参照してください。
- [Oracle Call Interfaceプログラマーズ・ガイド](#)
- [Oracle Database JDBC開発者ガイド](#)
- [『Oracle Universal Connection Pool for JDBC開発者ガイド』](#)

3.10.3 プロキシ・セッションの共有

Oracle Database 12c リリース2 (12.2.0.1)以降では、DRCPのプロキシ・セッションが同じスキーマに接続されているアプリケーション間で共有されます。

3.10.4 DRCPでのJDBCの使用

Oracle Database 12cリリース1 (12.1.0.1)から、Oracle JDBCドライバはDRCPをサポートしています。DRCPの実装によってサーバー側にプールが作成され、それが複数のクライアント・プール間で共有されます。これらのクライアント・プールは、JDBCのユニバーサル接続プールを使用します。ユニバーサル接続プールを使用することで、(サーバー上でサーバー・プロセスの数が減少するために)メモリー消費量が少なくなり、データベース・サーバーのスケーラビリティが向上します。

サーバー側接続のチェックインおよびチェックアウト操作を追跡するために、Javaアプリケーションはクライアント側プール(JDBCのユニバーサル接続プールまたは任意のサードパーティのJava接続プールなど)を使用する必要があります。

クライアント側でDRCPを有効にするには、次の操作を行う必要があります。

- NULL以外で空でないString値を接続プロパティoracle.jdbc.DRCPConnectionClassに渡します。
- 長い接続文字列で(SERVER=POOLED)を渡します。

次のように、短いURLフォームでも(SERVER=POOLED)を指定できます。

```
jdbc:oracle:thin:@//<host>:<port>/<service_name>[:POOLED]
```

たとえば:

接続プロパティ `oracle.jdbc.DRCPConnectionClass` を使用して、サーバー上のすべてのプール・サーバー・プロセスに同じ DRCP 接続クラス名を設定すると、複数の接続プール間でサーバー上のプール・サーバー・プロセスを共有できます。

DRCP では、特定の接続にタグを適用し、タグ付き接続を後から簡単に取得することもできます。

関連項目:

- [データベース常駐接続プールの開始](#)
- カスタム接続プールの実装の制御に使用する API の詳細は、[『Oracle Database JDBC 開発者ガイド』](#)を参照してください。
- クライアント側での DRCP の有効化の詳細は、[『Oracle Database JDBC 開発者ガイド』](#)を参照してください。

3.10.5 DRCPでのOCIセッション・プールAPIの使用

OCI セッション・プール API の `OCISessionPoolCreate()`、`OCISessionGet()` および `OCISessionRelease()` は、DRCP と相互運用されます。

OCI アプリケーションは、`OCISessionPoolCreate()` を呼び出して、DRCP 用の OCI セッション・プールの環境を初期化します。これは [『Oracle Call Interface プログラマーズ・ガイド』](#) に説明があります。

DRCP 用の OCI セッション・プールからセッションを取得するため、OCI アプリケーションは `OCI_SESSGET_SP00L` を mode パラメータに指定して `OCISessionGet()` を呼び出します。

DRCP 用の OCI セッション・プールにセッションを解放するため、OCI アプリケーションは `OCISessionRelease()` を呼び出します。

パフォーマンスを向上するため、OCI セッション・プールは接続ブローカへの接続を透過的にキャッシュできます。OCI アプリケーションは、`authInfo` パラメータを `OCI_ATTR_CONNECTION_CLASS` に設定して `OCISessionGet()` を呼び出し、接続クラス名を指定するか、または `OCISessionGet()` を呼び出す前に `OCIAuthInfo` ハンドルを使用することで、類似の状態ではアプリケーションがセッションを終了したセッションを再利用できます。

DRCP は、タグ付け、文キャッシュおよび TAF などの、従来のクライアント側 OCI セッション・プールにより提供されていた機能もサポートします。

関連項目:

- `OCISessionGet()` の詳細は、[『Oracle Call Interface プログラマーズ・ガイド』](#)を参照してください。
- `OCISessionRelease()` の詳細は、[『Oracle Call Interface プログラマーズ・ガイド』](#)を参照してください。
- DRCP のパフォーマンスを向上させる方法の詳細は、[「セッション純正值および接続クラス」](#)を参照してください。

3.10.6 セッション純正值および接続クラス

Oracle Database 11g リリース 1 (11.1) では、`OCISessionGet()` を使用してセッションを取得するときに指定できる 2 つの設定、つまりセッション純正值と接続クラスが OCI に導入されました。

トピック:

- [セッション純正值](#)
- [接続クラス](#)
- [例：接続クラスをHRMSとして設定](#)
- [例：接続クラスをRECMSとして設定](#)
- [セッション純正值および接続クラスのデフォルト](#)

3.10.6.1 セッション純正值

セッション純正值では、OCIアプリケーションがプールされたセッションを再利用できる(OCI_SESSGET_PURITY_SELF)か、新しいセッションを使用する必要がある(OCI_SESSGET_PURITY_NEW)かを指定します。

アプリケーションは、OCISessionGet()を呼び出す前にOCIAuthInfoハンドルで、あるいはOCISessionGet()を呼び出すときにmodeパラメータ内で、セッション純正值を設定できます。

関連項目:

- 接続プーリング・アプリケーションで新規セッションを設定する方法は、[例3-1](#)を参照してください。

例3-1 新規セッションのセッション純正值の設定

```
/* OCIAttrSet method */
ub4 purity = OCI_ATTR_PURITY_NEW;
OCIAttrSet(authInfop, OCI_HTYPE_AUTHINFO, &purity, (ub4)sizeof(purity)
OCI_ATTR_PURITY, errhp);
/* OCISessionGet mode method */
OCISessionGet (envhp, errhp, &svchp, authInfop, poolName, poolNameLen, NULL, 0, NULL,
NULL, NULL, OCI_SESSGET_SPOOL);
/* poolName is the name returned by OCISessionPoolCreate() */
```

ノート:

プールされたセッションを再利用する場合、サーバーの NLS 属性がクライアントの NLS 属性よりも優先されます。



たとえば、クライアントで NLS_LANG が french_france.us7ascii に設定されており、プールからドイツ語のセッションが割り当てられている場合、クライアント・セッションはドイツ語となります。

この問題を回避するには、接続クラスを使用して共有を制限します。

3.10.6.2 接続クラス

接続クラスには、OCIアプリケーションで必要な接続のタイプの論理名を定義します。プールされたセッションに接続クラスがある場合、OCIはセッションがその接続クラス外で共有されないように保証します。

たとえば、接続クラスはプールされたセッションを次の要素が共有することを防止します。

- 異なるユーザー
(ユーザーHRに対して最初に作成されたセッションは、ユーザーHRによる後続のリクエストのみに割り当てられます。)
- 同じユーザーの異なるセッション

- 同じユーザーによって実行中の異なるアプリケーション

(各アプリケーションは固有の接続クラスを保有できます。)

接続クラスを設定するには、OCIAuthInfoハンドルのOCI_ATTR_CONNECTION_CLASS属性を使用できます。接続クラス名は最大長1024バイトの文字列で、アスタリスク(*)は使用できません。

3.10.6.3 例: 接続クラスをHRMSとして設定

接続クラスをHRMSとして設定するには、OCISessionPoolCreate APIを使用できます。

[例3-2](#)は、HRMSアプリケーションが接続クラスHRMSのセッションを必要としていることを示します。

例3-2 接続クラスをHRMSとして設定

```
OCISessionPoolCreate (envhp, errhp, spoolhp, &poolName, &poolNameLen, "HRDB",
    strlen("HRDB"), 0, 10, 1, "HR", strlen("HR"), "HR", strlen("HR"),
    OCI_SPC_HOMOGENEOUS);

OCIAttrSet (authInfop, OCI_HTYPE_AUTHINFO, "HRMS", strlen("HRMS"),
    OCI_ATTR_CONNECTION_CLASS, errhp);
OCISessionGet (envhp, errhp, &svchp, authInfop, poolName, poolNameLen, NULL, 0,
    NULL, NULL, NULL, OCI_SESSGET_SPOOL);
```

3.10.6.4 例: 接続クラスをRECMSとして設定

接続クラスをRECMSとして作成するには、OCISessionPoolCreate APIを使用できます。

[例3-3](#)は、採用アプリケーションが接続クラスRECMSのセッションを必要としていることを示します。

例3-3 接続クラスをRECMSとして設定

```
OCISessionPoolCreate (envhp, errhp, spoolhp, &poolName, &poolNameLen, "HRDB",
    strlen("HRDB"), 0, 10, 1, "HR", strlen("HR"), "HR", strlen("HR"),
    OCI_SPC_HOMOGENEOUS);

OCIAttrSet (authInfop, OCI_HTYPE_AUTHINFO, "RECMS", strlen("RECMS"),
    OCI_ATTR_CONNECTION_CLASS, errhp);
OCISessionGet (envhp, errhp, &svchp, authInfop, poolName, poolNameLen, NULL, 0,
    NULL, NULL, NULL, OCI_SESSGET_SPOOL);
```

3.10.6.5 セッション純正值および接続クラスのデフォルト

[表3-5](#)に、OCIアプリケーションがOCIセッション・プール(OCISessionGet())を使用して)および他のソースから取得する接続の属性および設定のデフォルトを示します。

表3-5 セッション純正值および接続クラスのデフォルト

属性または設定	OCIセッション・プールからの接続のデフォルト値	OCIセッション・プール以外からの接続のデフォルト値
OCI_ATTR_PURITY	OCI_ATTR_PURITY_SELF	OCI_ATTR_PURITY_NEW
OCI_ATTR_CONNECTION_CLASS	OCI セッション・プールのすべての接続についてデフォルトの接続クラスとして使用されるクライアント側の各セッション・プールに対する、OCI で生成されたグローバル一意名	SHARED

属性または設定	OCIセッション・プールからの接続のデフォルト値	OCIセッション・プール以外からの接続のデフォルト値
セッションの共有	OCI セッション・プールからのセッションを要求するスレッド	SHARED 接続クラスを保有する特定のデータベースへの接続

3.10.7 データベース常駐接続プールの開始

DBAはSYSDBAとしてログオンし、DBMS_CONNECTION_POOL.START_POOLをデフォルト設定で使用して、デフォルト・プールSYS_DEFAULT_CONNECTION_POOLを開始する必要があります。

関連項目:

- プールの構成の詳細は、[『Oracle Database管理者ガイド』](#)を参照してください。

3.10.8 DRCPの有効化

アプリケーションでDRCPを有効にするには、簡易接続文字列で:POOLEDを指定([例3-4](#)を参照)するか、TNS接続文字列で(SERVER=POOLED)を指定([例3-5](#)を参照)します。

例3-4 簡易接続文字列で:POOLEDを指定してDRCPを有効化

```
oraclehost.company.com:1521/books.company.com:POOLED
```

例3-5 TNS接続文字列でSERVER=POOLEDを指定してDRCPを有効化

```
BOOKSDB = (DESCRIPTION=(ADDRESS=(PROTOCOL=tc)(HOST=oraclehost.company.com)
(PORT=1521))(CONNECT_DATA = (SERVICE_NAME=books.company.com)(SERVER=POOLED)))
```

3.10.9 OCIアプリケーションにおけるDRCPの拡張性による利点

次のOCIアプリケーション・シナリオと、DRCPがそれぞれにもたらすメリットについて検討します。

- アプリケーションがOCIセッション・プールを使用せず、接続クラスや純正值設定も指定(またはPURITY=NEWを指定)しない場合。

アプリケーションはDRCPから新規セッションを取得します。アプリケーションが接続をプールに戻す場合、デフォルトによりこのセッションは同じアプリケーションの他のインスタンスと共有されません。したがって、クライアント・セッションが存続する間、プールされたサーバーはクライアントに割り当てられたままになります。(SQL*Plusは、OCIセッション・プールを使用しないクライアントの一例です。SQL*Plusは接続がアイドルのときでも、接続を保持し続けます。)

アプリケーションは既存のプール・サーバーを再利用できるというメリットを得られます。

- アプリケーションがOCIセッション・プール外のOCISessionGet()を呼び出すか、接続クラスおよびPURITY=SELFを指定する場合。

アプリケーションは、DRCPプール・サーバーおよびセッションを両方とも再利用できます。ただし、OCISessionRelease()コールの後、OCIは接続ブローカへの接続を終了します。次のOCISessionGet()コールでは、アプリケーションはブローカに再接続して、DRCPは指定された接続クラスに属するプール・サーバー(およびセッション)を割り当てます。再接続では接続の確立および再認証のコストが発生します。

アプリケーションはDRCPリソース(プロセスおよびセッション)をより効率的に共有できますが、接続ブローカへの接続のキャッシュからメリットは得られません。

- アプリケーションがOCIセッション・プールAPIを使用し、接続クラスを指定し、PURITY=SELFを指定する場合。

アプリケーションは、プール・サーバーおよび関連付けられたセッションの両方を再利用して、接続ブローカへのキャッシュされた接続からメリットを得ることで、すべてのDRCP機能を使用します。キャッシュされた接続では、OCISessionGet()コールで再認証のコストが発生しません。

関連項目:

[OCISessionPoolCreate\(\)](#)

[OCISessionGet\(\)](#)

[OCISessionRelease\(\)](#),

[OCISessionPoolDestroy\(\)](#)

3.10.10 JavaアプリケーションにおけるDRCPのスケーラビリティの利用

ユニバーサル接続プール(UCP)を使用する、または接続ファクトリとしてConnectionPoolDataSourceを使用しているユーザーは、(コードではなく)構成のみを変更することでDRCPを使用するようにアップグレードできます。

関連項目:

- OCIアプリケーションとしてJavaアプリケーションでどのようにDRCPが役立つかの詳細は、[OCIアプリケーションにおけるDRCPの拡張性による利点](#)を参照してください。

3.10.11 DRCPの使用に関するベスト・プラクティス

DRCPを最大限に活用できるアプリケーションを設計するステップは、OCIセッション・プールを使用するアプリケーションに必要なステップとよく似ています。

唯一の追加ステップとして、DRCPとともに動作するようデプロイされた場合、最大限のパフォーマンスを得るために、アプリケーションで明示的に接続クラスを指定する必要があります。

最大限のパフォーマンスを得て、さらにDRCPリソースの共有機能を強化するため、同じアプリケーションの複数のインスタンスは、同じ接続クラスを指定する必要があります。アプリケーションの複数のインスタンスでデータベース・セッションを共有できることを確認してください。

[例3-6](#)は、DRCPアプリケーションを示しています。

関連項目:

- [Oracle Call Interfaceプログラマーズ・ガイド](#)

例3-6 DRCPアプリケーション

```
/* Assume that all necessary handles are allocated. */  
  
/* This middle tier uses a single database user. Create a homogeneous
```



```

    client-side session pool */
OCISessionPoolCreate (envhp, errhp, spoolhp, &poolName, &poolNameLen, "BOOKSDB",
    strlen("BOOKSDB"), 0, 10, 1, "SCOTT", strlen("SCOTT"), "password",
    strlen("password"), OCI_SPC_HOMOGENEOUS);

while (1)
{
    /* Process a client request */
    WaitForClientRequest();
    /* Application function */

    /* Set the Connection Class on the OCIAuthInfo handle that is passed as
       argument to OCISessionGet*/

    OCIAttrSet (authInfop, OCI_HTYPE_AUTHINFO, "BOOKSTORE", strlen("BOOKSTORE"),
        OCI_ATTR_CONNECTION_CLASS, errhp);

    /* Purity need not be set, as default is OCI_ATTR_PURITY_SELF
       for OCISessionPool connections */

    /* You can get a SCOTT session released by Middle-tier 2 */
    OCISessionGet(envhp, errhp, &svchp, authInfop, poolName, poolNameLen, NULL, 0,
        NULL, NULL, NULL, OCI_SESSGET_SPOOL);

    /* Database calls using the svchp obtained above */
    OCIStmtExecute(...)

    /* This releases the pooled server on the database for reuse */
    OCISessionRelease (svchp, errhp, NULL, 0, OCI_DEFAULT);
}

/* Middle tier is done - exiting */
OCISessionPoolDestroy (spoolhp, errhp, OCI_DEFAULT);

```

[例3-7](#)と[例3-8](#)では、[例3-6](#)からBOOKSTOREアプリケーションを提供する10個の中間層ホストにコードをデプロイする接続文字列を示しています。

[例3-7](#)では、データベースが専用サーバー・モードでDRCPは有効化されていないOracle Database 12c (またはそれ以下)であり、クライアントに12cライブラリがあると仮定しています。アプリケーションはデータベースから専用サーバー接続を取得します。

例3-7 専用サーバー・モードでDRCPを使用しないデプロイメントに対する接続文字列

```
BOOKSDB = (DESCRIPTION=(ADDRESS=(PROTOCOL=tcP)(HOST=oraclehost.company.com)
(PORT=1521))(CONNECT_DATA = (SERVICE_NAME=books.company.com)))
```

[例3-8](#)では、Oracle Database 12cデータベースでDRCPが有効であると仮定します。すべての中間層プロセスがDRCPのプーリング機能を利用できます。DRCPのデータベース・リソース要件は、専用サーバー・モードでの要件より大幅に少なくなります。

例3-8 DRCPを使用するデプロイメントに対する接続文字列

```
BOOKSDB = (DESCRIPTION=(ADDRESS=(PROTOCOL=tcP)(HOST=oraclehost.company.com)
(PORT=1521))(CONNECT_DATA = (SERVICE_NAME=books.company.com)(SERVER=POOLED)))
```

3.10.12 互換性と移行

Oracle Database 12cクライアント・ライブラリにリンクされたOCIアプリケーションは、次のそれぞれに対して変更なしで動作します。

- DRCPを無効にしたOracle Database 12cデータベース

- Oracle Database 12cより前のリリースのデータベース・サーバー
- DRCPを有効にしたOracle Database 12cデータベース・サーバー(DRCP接続文字列を使用してデプロイした場合)

前述のとおりOCIセッション・プールAPIを接続クラスおよび純正值設定とともに使用するようクライアントが適切に変更されていれば、そのクライアントはDRCPにより提供される高度なスケーラビリティを利用できます。

関連項目:

- Oracle JDBCドライバでのDRCPのサポートの詳細は、[『Oracle Database JDBC開発者ガイド』](#)を参照してください。

3.10.13 DRCPとOracle Databaseのネイティブ・ネットワーク暗号化の併用

DRCPをOracle Databaseネイティブ・ネットワーク暗号化とともに使用する場合、サポートされている次のチェックサム・アルゴリズムおよび暗号化アルゴリズムを使用できます。

チェックサム・アルゴリズム

- SHA1
- SHA256
- SHA384
- SHA512

暗号化アルゴリズム

- AES128
- AES192
- AES256

このリリースでは、DES、DES40、3DES112、3DES168、MD5、RC4_40、RC4_56、RC4_128およびRC4_256アルゴリズムは非推奨です。より強力なアルゴリズムを使用するようにOracle Database環境を移行するには、My Oracle Supportノート[2118136.2](#)で説明されているパッチをダウンロードしてインストールします。

3.10.14 DRCPの制限事項

プール・サーバーでは、次の操作を実行できません。

- データベースの停止
- DRCPの停止
- 接続済ユーザーのパスワードの変更
- 共有データベース・リンクを使用した、異なるインスタンスにあるDRCPへの接続
- TCPSによるアドバンスド・セキュリティ・オプション(ASO)の使用
- DRCPによるエンタープライズ・ユーザー・セキュリティの使用
- (OCI_MIGRATEオプションを使用して)直接的に、または(OCIConnectionPoolCreate()を起動して)間接的に、サーバー側の移行可能なセッションを使用

- 初期クライアント・ロールの使用
- (OCI_ATTR_APPCTX_NAMEおよびOCI_ATTR_APPCTX_VALUEなどの)アプリケーション・コンテキスト属性の使用

DDL文実行の前に作成されたセッションは、DDL文実行後にクライアントに割当てできます。したがって、プール内のデータベース・ユーザーに影響するDDL文の実行時は注意してください。たとえば、ユーザーを削除する前に、そのユーザーのセッションがプール内に存在せず、そのユーザーとして認証されたブローカへの接続がないことを確認します。

明示的なロールが有効なセッションがプールに解放される場合、後でデフォルトのログイン・ロールを必要とする(同じユーザーの)接続に割り当てることができます。したがって、明示的なロールのあるセッションは解放せずに、かわりにセッションを終了してください。

DRCPでアプリケーション・コンティニューイティを使用できますが、セッションがデフォルトのセッション状態でプールに返されることを確認する必要があります。

ノート:



暗号化や DRCP を使用した厳密認証などの Oracle Advanced Security 機能を使用できます。

ユーザーは、データの暗号化とデータの整合性の組合せを混在させることができます。ただし、ユーザーは接続クラスを使用して、このような組合せをそれぞれ分離する必要があります。たとえば、ユーザー・アプリケーションが、暗号化メカニズムとして、ある接続セットにAES256を指定し、別の接続セットにはAES128を指定する必要がある場合、アプリケーションは各セットに異なる接続クラスを指定する必要があります。

3.10.15 カスタム・プールでのDRCPの使用

すでにDRCP、FANおよびRLBと統合されている、OCIセッション・プールを使用することを強くお勧めします。ただし、次のいずれかがtrueの場合、OCIセッション・プールを使用しないアプリケーションではDRCPを引き続き使用できます。

- アプリケーションが独自のカスタム接続プールを使用して作成されている場合。
- アプリケーションでプールは使用されないが、セッションを使用しない期間があり(したがってそれをプールに解放でき)、同じセッションを取り戻すことに依存しない場合

DRCPをこのようなアプリケーションで使用するには、セッションがステートフルである必要があります。つまり、セッションにOCI_ATTR_SESSION_STATE属性が必要です。アプリケーションがステートフルでDRCPが有効な場合、OCIはDRCPプールから適切なセッションを透過的に割り当てます。アプリケーションがステートレス(OCI_SESSION_STATELESS属性がある)でDRCPが有効な場合、OCIはセッションを透過的にDRCPプールに返します。

基本のデータベース・リソースの効率的な利用のために、アプリケーションではセッション状態をできるだけ速やかに識別する必要があります。

ノート:



属性 OCI_ATTR_SESSION_STATE または OCI_SESSION_STATELESS を指定するアプリケーションは、セッション純正值および接続クラスも指定する必要があります。

関連項目:

- OCI_ATTR_SESSION_STATEおよびOCI_SESSION_STATELESSの詳細は、[「セッション純正值および接続クラス」](#)を参照してください。
- セッション状態の詳細は、[「明示的なステートフルまたはステートレス・セッションのマーキング」](#)を参照してください。
- OCI_ATTR_SESSION_STATE属性の詳細は、[『Oracle Call Interfaceプログラマーズ・ガイド』](#)を参照してください。

3.10.16 明示的なステートフルまたはステートレス・セッションのマーキング

通常、アプリケーションは、1つの作業単位の期間において特定のデータベース・セッションを必要とします。この期間中、セッションはステートフルです。この期間の後、後続の作業単位のために特定のセッションを維持することにアプリケーションが依存しない場合、セッションはステートレスです。

ステートフルからステートレス(あるいは逆)へのセッションの変化をアプリケーションまたはコール元が検出すると、アプリケーションはOCI_ATTR_SESSION_STATEまたはOCI_SESSION_STATELESS属性を使用して変化をOCIに明示的に通知できます。この情報によってOCIおよびOracle Databaseは、アプリケーションが使用していないセッションを別のユーザーに再度割り当て、必要になった場合はアプリケーションに新規セッションを割り当てるなどの、スケーラビリティ最適化を透過的に実行します。

関連項目:

[カスタム・プールでのDRCPの使用](#)

[例3-9](#)は、セッションの状態を明示的にマークするコードの一部を示しています。

例3-9 明示的なステートフルまたはステートレス・セッションのマーキング

```
wait_for_transaction_request();
do {

    ub1 state;

    /* mark database session as STATEFUL */
    state = OCI_SESSION_STATEFUL;
    checkerr(errhp, OCIAttrSet(usrhp, OCI_HTYPE_SESSION,
                              &state, 0, OCI_ATTR_SESSION_STATE, errhp));
    /* do database work consisting of one or more related calls to the database */
    ...

    /* done with database work, mark session as stateless */
    state = OCI_SESSION_STATELESS;
    checkerr(errhp, OCIAttrSet(usrhp, OCI_HTYPE_SESSION,
                              &state, 0, OCI_ATTR_SESSION_STATE, errhp));

    wait_for_transaction_request();

} while(not _done);
```

OCIセッション・プールの外部で取得されたセッションは、OCI_SESSION_STATEFULとマークされ、アプリケーションによって明示的にOCI_SESSION_STATELESSとマークされないかぎり、OCI_SESSION_STATEFULが保たれます。

OCIセッション・プールから取得されたセッションは、最初のコールがそのセッションで開始されるときに、デフォルトでOCI_SESSION_STATEFULとマークされます。セッションがプールに解放されるときは、デフォルトでOCI_SESSION_STATELESSとマークされます。したがって、OCIセッション・プールの使用時はセッションをステートフルまたはス

テートレスとして明示的にマークする必要はありません。

関連項目:

OCI_ATTR_SESSION_STATEの詳細は、『Oracle Call Interfaceプログラマーズ・ガイド』を参照してください

3.10.17 Oracle Real Application ClustersでのDRCPの使用方法

Oracle Real Application Clusters (Oracle RAC)は、単一データベースを複数ノードにある複数インスタンスで管理するデータベース・オプションです。Oracle RAC環境のデータベースでDRCPが構成されているとき、プールの構成は各データベース・インスタンスに適用されます。1つのインスタンスでプールを起動または停止すると、すべてのインスタンスのプールが起動または停止されます。

3.10.18 プラガブル・データベースでのDRCPの使用

マルチテナント・コンテナ・データベース(CDB)でのDRCPは、ルート・コンテナで構成および管理されます。プールの構成、開始および停止は、ルート・コンテナに接続しているときに実行できます。このプールは、クライアントが様々なサービス名を使用して接続する、様々なプラガブル・データベースのプール・サーバーを管理します。

3.10.19 DRCPとData Guard

Data Guard環境でDRCPが動作する場合は次のようになります。

- フィジカル・スタンバイ・データベースの場合、次のように実行します。
 - プールを起動できるのは、プールがプライマリ・データベースで実行中の場合のみです。
 - プールを停止できるのは、プールがプライマリ・データベースで停止している場合のみです。
 - プール・パラメータの構成、デフォルトへの復元または変更を行うことはできません。

物理スタンバイ・データベースがプライマリ・データベースになると、前述した制限事項は物理スタンバイ・データベースに適用されなくなります。

- ロジカル・スタンバイ・データベースでは、すべてのプール操作を行うことができます。

3.11 Memoptimizeプール

このプールでは、SELECT文の読取り操作が最適化されます

Memoptimize Rowstore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実行します。



ノート:

遅延挿入は、標準のロックと REDO のメカニズムが使用されないため、ロールバックできません

関連トピック

- [Oracle Database概要](#)
- [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 SQL言語リファレンス](#)
- [Oracle Databaseリファレンス](#)

3.12 Oracle RACシャーディング

この項では、Oracle RACのシャーディングについて説明します

Oracle RACシャーディングでは、最小限のアプリケーション変更でOracle RACデータベースのパフォーマンスおよびスケーラビリティが向上します。表パーティションをOracle RACインスタンスと結び付け、データベース・リクエストをルーティングします。これは、対応するパーティションを論理的に保持するインスタンスへのパーティション化キーを指定します。これにより、キャッシュ利用が効率的になり、インスタンスにまたがるブロックpingが削減されます。パーティション化キーは、最大パフォーマンス・クリティカル・リクエストに対してのみ追加できます。キーを指定しないリクエストは透過的に機能し、どのインスタンスにもルーティングできます。この機能は、ALTER SYSTEMコマンドを実行して使用可能にすることができ、データベース・スキーマおよびSQL文を変更する必要はありません。



ノート:

データベース接続を要求するときには、パーティション化キーの値を指定する必要があります。

関連トピック

- [Oracle Database Net Services管理者ガイド](#)
- [Oracle Call Interfaceプログラマーズ・ガイド](#)
- [Oracle Universal Connection Pool開発者ガイド](#)
- [Oracle Data Provider for .NETの開発者ガイドfor Microsoft Windows](#)
- [Oracle Real Application Clusters管理およびデプロイメント・ガイド](#)
- [Oracle Database概要](#)
- [Oracle Database管理者ガイド](#)

4 Oracle Real-World Performanceのアプリケーション の設計

Real-World Performanceのアプリケーションを設計する場合、バインド変数、インストルメンテーションおよびセット・ベース処理をどのようにコーディングするか考慮する必要があります。

トピック:

- [バインド変数の使用](#)
- [インストルメンテーションの使用](#)
- [セット・ベース処理の使用](#)

4.1 バインド変数の使用

SQL文またはPL/SQLブロックにおけるバインド変数プレースホルダは、実行時にデータを提供する場所を示します。

たとえば、次の文で作成された表にアプリケーションからデータを挿入する場合を考えます。

```
CREATE TABLE test (x VARCHAR2(30), y VARCHAR2(30));
```

データは実行時まで未知であるため、動的なSQLを使用する必要があります。

次の文は1つの行をtest表内に挿入し、x列とy列の文字列リテラルを結合します。

```
INSERT INTO test (x,y) VALUES ( ' ' || REPLACE (x, ' ', ' ') || ' ',  
                                ' ' || REPLACE (y, ' ', ' ') || ' ');
```

次の文は、x列およびy列のバインド変数: xおよび: yを使用して、1つの行をtest表内に挿入します。

```
INSERT INTO test (x,y) VALUES (:x, :y);
```

コーディングがより容易なのは、バインド変数プレースホルダを使用する文です。

次に、前述したそれぞれの方法を使用して表testに1,000行を挿入する、動的な一括ロード操作を想定します。

文字列リテラルを結合する方法は1,000のINSERT文を使用し、それぞれはハード解析、修飾、セキュリティ・チェック、最適化およびコンパイルが行われる必要があります。それぞれの文はハード解析されるため、ラッチの数も大幅に増えます。ラッチは相互排除ロック・メカニズム、つまりシリアライズ・デバイスで、同時実行が禁止されています。

バインド変数プレースホルダを使用するメソッドは、1つのINSERT文のみを使用します。文はソフト解析、修飾、セキュリティ・チェック、最適化およびコンパイルが行われてから共有プールにキャッシュされます。コンパイルされた文は共有プールから、1,000個の挿入のそれぞれに使用されます。この文キャッシュはバインド変数を使用する大きな利点です。

バインド変数プレースホルダを使用するアプリケーションは、文字列結合を使用するアプリケーションよりもスケーラブルで、サポートするユーザー数は増加し、必要なリソースは減少し、実行速度は向上します。そして、SQLインジェクション攻撃を受ける可能性も減少します。SQL文で文字列結合が使用されていると、エンド・ユーザーがその文を改変し、なんらかの不正行為にアプリケーションを利用することが可能になります。

入力変数用のバインド変数プレースホルダは、DELETE文、INSERT文、SELECT文およびUPDATE文と、式またはリテラルが使用可能なPL/SQLブロック内の任意の位置で使用できます。PL/SQLでは、出力変数にもバインド変数プレースホルダを使用できます。バインドは、問合せ以外の操作では入力変数と出力変数の両方に使用されます。

関連項目:

- SQLインジェクションからアプリケーションを保護するためのバインド変数の使用方法の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください
- OCIでバインド変数プレースホルダを使用する方法の詳細は、『Oracle Call Interfaceプログラマーズ・ガイド』を参照してください

4.2 インストルメンテーションの使用

インストルメンテーションの使用は、アプリケーションを通じてデバッグ・コードを追加することを意味します。有効な場合、このコードは、問題の特定に役立つ情報が含まれるトレース・ファイルを生成します。トレース・ファイルは、問題のある層の特定に役立つため、多層アプリケーションをデバッグする際に特に有効です。

関連項目:

詳細は、[「SQLトレース機能\(SQL_TRACE\)」](#)を参照してください

4.3 セット・ベース処理の使用

データ・ウェアハウス環境のデータベース・アプリケーションの一般的なタスクは巨大なデータ・セットを問い合わせているか変更しています。

たとえば、あるアプリケーションでは何千万もの行でデータ・セット・ナンバリングを結合し、一連の条件でフィルタリングし、集計してからユーザーに結果を表示する場合があります。一方、あるアプリケーションでは、指定した条件に基づいて行を十億行の表からフィルタリングして除外してから別の表に一致する行を挿入する場合があります。

アプリケーション開発者の課題は、これらの大きなデータ・セットを処理する際に、どうすれば高いパフォーマンスを実現できるか、という点です。処理方法は2つのカテゴリに分類されます：繰返しおよびセット・ベースです。何年にも渡るテストによって、Oracle Real-World Performanceグループはセット・ベース処理の方法の方が、大きなデータ・セットを処理するデータベース・アプリケーションにおいては桁違いに優れていることがわかりました。

トピック:

- [繰返しデータ処理](#)
- [セット・ベース処理の使用](#)

4.3.1 繰返しデータ処理

繰返しデータ処理は、行ごと、配列を使用して、または手動の並列度を使用してデータを処理します。

トピック:

- [繰返しデータ処理について](#)
- [繰返しデータ処理：行ごと](#)
- [繰返しデータ処理：配列](#)
- [繰返しデータ処理：手動の並列度](#)

4.3.1.1 繰返しデータ処理について

このタイプの処理では、アプリケーションは条件ロジックを使用して行セット全体を繰り返します。

繰返しのアプリケーションは、PL/SQL、Java、その他あらゆる手続き型言語またはオブジェクト指向言語で記述できます。この方法が「繰返し」と言われるのは、行ソースを1つ以上の行を含むサブグループに分割してから、各サブグループを処理するからです。単一プロセスがすべてのサブグループを通して繰り返すことも、複数プロセスが並列にサブグループを通して繰り返すこともできます。

必須ではありませんが、通常、繰返し処理では次のようにクライアント/サーバー・モデルを使用します。

1. データベース・サーバーからクライアント・アプリケーションに行のグループを転送します。
2. クライアント・アプリケーション内でグループを処理します。
3. 処理されたグループをデータベース・サーバーに転送して戻します。

行ごとの処理、配列処理および手動の並列度の3つの主要な方法を使用して繰返しアルゴリズムを実装できます。どの方法も結果は同じですが、パフォーマンスの観点からは、それぞれに利点と欠点があります。

4.3.1.2 繰返しデータ処理：行ごと

繰返しの方法では、行ごとの処理が最も一般的です。

単一処理はデータ・セット内をループし、一度に1つの行を処理します。典型的な実装では、アプリケーションはデータベースからそれぞれの行を取得し、中間層で処理してから行をデータベースに送って戻し、データベースでDMLが実行されコミットされます。

機能要件が、ext_scan_eventsという外部表を問い合せて、その行をstage1_scan_eventsという名前のヒープ構成表に挿入することとします。次に示すPL/SQLブロックでは、行ごとの方法を使用して、この要件を満たしています。

```
declare
  cursor c is select s.* from ext_scan_events s;
  r c%rowtype;
begin
  open c;
  loop
    fetch c into r;
    exit when c%notfound;
    insert into stage1_scan_events d values r;
    commit;
  end loop;
  close c;
end;
```

行ごとのコードは次の処理を実行するためカーソル・ループを使用します。

1. ext_scan_eventsから1行をクライアント・ホストで実行しているアプリケーションにフェッチするか、これ以上行が存在しない場合にはプログラムを終了します。
2. stage1_scan_eventsに行を挿入します。
3. 前述の挿入をコミットします。
4. ステップ1に戻ります。

行ごとの方法には次のメリットがあります。

- 小さなデータ・セットでは優れたパフォーマンスを示します。ext_scan_eventsに10,000レコードが含まれているとします。アプリケーションが1行を1ミリ秒で処理すると、合計処理時間は10秒になります。

- ループ・アルゴリズムは、プロの開発者は皆慣れているため、すぐに記述しやすく、理解しやすいです。

行ごとの方法には次のデメリットがあります。

- 大きなデータ・セットの処理時間が許容できないほど長くなる場合があります。ext_scan_eventsに10億行が含まれていて、アプリケーションが各行の処理に平均1ミリ秒かかると、合計処理時間は12日間になります。1兆行の処理には32年かかります。
- アプリケーションはシリアルに実行されるため、最新のハードウェアで稼働しているOracle Databaseのネイティブの並列処理機能を活用できません。たとえば、行ごとの方法では、マルチ・コア・コンピュータ、Oracle RACまたはOracle Exadata Machineの恩恵を受けられません。たとえば、データベース・ホストに16 CPUと32コアが含まれている場合、単独のデータベース・サーバー・プロセスが各行を読み取りまたは書き込みしている間、31コアがアイドル状態になります。複数インスタンスがOracle RACデプロイメントに存在する場合、1つのインスタンスのみがデータを処理できます。

4.3.1.3 繰返しデータ処理：配列

配列処理は、各繰返しに単一の行ではなく行のグループを処理すること以外は行ごとの処理と同じです。

行ごとの方法と同様に、配列処理はシリアルで、一度に1つのデータベース・サーバー・プロセスのみが行のグループに対して操作することを意味します。典型的な配列実装では、アプリケーションはデータベースから各行のグループを取得し、中間層で処理してからグループをデータベースに送って戻し、行のグループに対してDMLが実行され、コミットされます。

機能要件は「繰返しデータ処理：行ごと」の例と同じで、ext_scan_eventsという名前の外部表を問い合せて、その行をstage1_scan_eventsという名前のヒープ構成表に挿入することとします。データベース・サーバーとは別のホストのSQL*Plusで実行する次のPL/SQLブロックは、配列の方法を利用してこの要件を満たします。

```
declare
  cursor c is select s.* from ext_scan_events s;
  type t is table of c%rowtype index by binary_integer;
  a t;
  rows binary_integer := 0;
begin
  open c;
  loop
    fetch c bulk collect into a limit array_size;
    exit when a.count = 0;
    forall i in 1..a.count
      insert into stage1_scan_events d values a(i);
    commit;
  end loop;
  close c;
end;
```

前述のコードは、FETCH文でのBULK COLLECTオペレータの使用の点で、同等の行ごとのコードとは異なります。これは、PLS_INTEGER型のarray_size値によって制限されています。たとえば、array_sizeが100に設定されている場合、アプリケーションは100行ごとのグループで行をフェッチします。

カーソル・ループは、次の一連の処理を実行します。

1. ext_scan_eventsから行の配列をクライアント・ホストで実行しているアプリケーションにフェッチするか、ループ・カウンタが0になれば終了します。
2. 行の配列全体をループし、各行をstage1_scan_events表に挿入します。
3. 前述の挿入をコミットします。
4. ステップ1に戻ります。

PL/SQLでは、配列コードは終了条件をテストするためにカーソル属性c%not foundではなくカウンタを使用する点において、行ごとのコードとは異なります。その理由は、フェッチが表内で行の最後のグループを収集する場合、c%not foundがグループを強制的に終了してしまい、これは望ましくない動作だからです。カウンタを使用すると、各フェッチは指定した数の行を収集し、収集が空になるとプログラムが終了します。

配列の方法は行ごとの方法に対して次のメリットがあります。

- 配列では、アプリケーションが一度に行をグループで処理でき、これはクライアントとサーバー間でネットワーク・ラウンドトリップ、COMMIT時間およびコード・パスを削減することを意味します。これらの要素を組み合わせると、桁違いに合計処理時間を短縮する可能性があります
- サーバー・プロセスは挿入をバッチ処理し、挿入ごとではなく挿入のグループごとにコミットするため、データベースはさらに効率的です。コミットの数減らすと、入出力負荷を削減し、長い同期待ちイベントが発生する可能性が減少します。

この方法のデメリットは行ごとの処理と同じです。大きなデータ・セットの処理時間が、許容できないほどになる場合があります。1兆行の表の場合、処理時間を32年から3.2年に削減しても、まだ許容範囲にはなりません。また、アプリケーションは単一CPUコアでシリアルに実行される必要があり、そのためOracle Databaseのネイティブの並列度は活用できません。

関連項目:

[繰返しデータ処理: 行ごと](#)

4.3.1.4 繰返しデータ処理: 手動の並列度

手動の並列度は行ごとおよび配列処理と同じ繰返しアルゴリズムを使用しますが、複数のサーバー・プロセスが並列にジョブを操作できるようにします。

典型的な実装では、アプリケーションは複数回ソース・データをスキャンしてから、ORA_HASHファンクションを使用してデータを並列の挿入処理間で分けます。

ORA_HASHファンクションは、特定の式のハッシュ値を計算します。ファンクションは3つの引数をとります。

- expr。これは通常、列名です
- max_bucket。これはハッシュ・バケットの数を指定します
- seed_value。同じデータから複数の結果を有効にします(デフォルトは0です)

たとえば、次の文は販売表を0から9にナンバリングした10個の行のバケットに分割し、バケット1から行を戻します。

```
SELECT * FROM sales WHERE ORA_HASH(cust_id, 9) = 1;
```

アプリケーションがこの方法でORA_HASHを使用し、n個のハッシュ・バケットが存在すると、各サーバー・プロセスはデータの1/nを操作します。

機能要件が行ごとおよび配列の例と同じであるとして。ソース表からスキャン・イベントを読み取り、それをstage1_scan_events表に挿入します。主な違いを次に示します。

- スキャン・イベントは大量のフラット・ファイルに格納されます。ext_scan_events_dets表はこれらのフラット・ファイルを説明します。ext_scan_events_dets.file_seq_nbr列は数値のプライマリ・キーを格納し、ext_file_name列はファイル名を格納します。
- 32サーバー・プロセスは並行して実行する必要があり、それぞれのサーバー・プロセスは別々の外部表を問い合わせます。32個の外部表はext_scan_events_31内でext_scan_events_0という名前が付けられます。ただし、各サー

バー・プロセスは同じstage1_scan_events表に挿入します。

- 同じPL/SQLプログラムの32個のスレッドを実行することで、PL/SQLを使用して並列度を実現します。それぞれのスレッドはOracle Schedulerによって管理される別々のジョブとして並行して実行されます。ジョブはスケジュールおよびプログラムの組合せです。

データベース・サーバーとは別のホストのSQL*Plusで実行する次のPL/SQLコードは手動の並列度を使用します。

```
declare
  sqlstmt varchar2(1024) := q'[
-- BEGIN embedded anonymous block
  cursor c is select s.* from ext_scan_events_${thr} s;
  type t is table of c%rowtype index by binary_integer;
  a t;
  rows binary_integer := 0;
begin
  for r in (select ext_file_name from ext_scan_events_dets where
ora_hash(file_seq_nbr,${thrs}) = ${thr})
  loop
    execute immediate
      'alter table ext_scan_events_${thr} location' || '(' || r.ext_file_name || ')';
    open c;
    loop
      fetch c bulk collect into a limit ${array_size};
      exit when a.count = 0;
      forall i in 1..a.count
        insert into stage1_scan_events d values a(i);
      commit;
-- demo instrumentation
      rows := rows + a.count; if rows > 1e3 then exit when not
sd_control.p_progress('loading','userdefined',rows); rows := 0; end if;
    end loop;
    close c;
  end loop;
end;
-- END embedded anonymous block
]';
begin
  sqlstmt := replace(sqlstmt, '${array_size}', to_char(array_size));
  sqlstmt := replace(sqlstmt, '${thr}', thr);
  sqlstmt := replace(sqlstmt, '${thrs}', thrs);
  execute immediate sqlstmt;
end;
```

このプログラムには、外側から内側へ3つの繰返しの構成があります。

1. フラット・ファイルの名前を取得し、外部表の場所としてフラット・ファイル名を指定するのにDDLを使用する外側FOR LOOP
2. 外部表の問合せから行のグループをフェッチする中間のLOOP文。
3. 各グループを通じて繰返し、行を挿入する最も内側のFORALL文

このサンプル・プログラムでは、すべてのジョブで\$thrsを31に設定し、\$thrをすべてのジョブで0から31の間の別の値に設定します。たとえば、ジョブ1では、\$thrが0に設定され、ジョブ2では\$thrが1に設定され(続く)、となる場合があります。

\$thrが0に設定されている最初のジョブによって実行されたプログラムでは、外部FOR LOOPは次の問合せの結果を通じて繰返されます。

```
select ext_file_name
from   ext_scan_events_dets
where  ora_hash(file_seq_nbr,31) = 0
```


ORA_HASHファンクションはext_scan_events_dets表を32個の均等に分散されたバケットに分割し、SELECT文はバケット0のファイル名を取得します。たとえば、問合せ結果セットには次のファイル名が含まれている場合があります。

```
/disk1/scan_ev_101  
/disk2/scan_ev_003  
/disk1/scan_ev_077  
...  
/disk4/scan_ev_314
```

中間のLOOPはファイル名のリストを通して繰り返されます。たとえば、結果セットの最初のファイル名が/disk1/scan_ev_101であるとして。ジョブ1では、外部表はext_scan_events_0という名前で、そのため、LOOPの最初の繰返しではこの表の場所が次のように変更されます。

```
alter table ext_scan_events_0 location(/disk1/scan_ev_101);
```

最も内部のFORALL文では、BULK COLLECTオペレータがext_scan_events_0表から配列に行を取得し、stage1_scan_events表に行を挿入してからバルク挿入をコミットします。プログラムがFORALL文を終了すると、プログラムはループ内の次のアイテムに進み、外部表のファイルの場所を/disk2/scan_ev_003に変更してから前述の繰返しのように行を問合せ、挿入およびコミットを行います。Job 1はこの方法で、ハッシュ・バケット0に対応するフラット・ファイルに含まれているすべてのレコードがstage1_scan_events表に挿入されるまで処理を続行します。

ジョブ1が実行されている間、他の31個のOracle Schedulerジョブは並列に実行されます。たとえば、ジョブ2が\$thrを1に設定すると、これはカーソルを表ext_scan_events_1の問合せとして定義し、ジョブ32まで続けると、\$thrを31に設定してカーソルを表ext_scan_events_31の問合せとして定義します。この方法により、各ジョブは同時にスキャン・イベント・ファイルの別々のサブセットから読み取り、サブセットから同じstage1_scan_events表にレコードを挿入します。

手動の並列度の方法は、他の繰返し方法よりも次の点においてメリットがあります。

- サーバー・プロセスが並行して動作するため、大きなデータ・セットで優れたパフォーマンスを実現します。たとえば、32プロセスが作業を分割し、データベースに十分なCPUおよびメモリー・リソースがあって競合が発生しない場合、データベースは配列の方法が単一のジョブを実行するのにかかった時間で同時に32個の挿入ジョブを実行する可能性があります。大きなデータ・セットでのパフォーマンス向上は、シリアルな方法に対して桁違いになることが多いです。
- アプリケーションがORA_HASHを使用してワークロードを分散すると、実行の各スレッドは同じ量のデータにアクセスできます。各スレッドが同じ量のデータを読み書きすると、並行プロセスは同じ時間で終了します。これは、アプリケーションが実行されているかぎり、データベースがハードウェアを活用していることを意味します。

手動の並列度の方法には次のデメリットがあります。

- コードが比較的長く、複雑でわかりにくくなります。ワークロードを分散する作業はデータベースではなく、開発者の役割になるため、アルゴリズムが複雑になります。実質、アプリケーションは並行アルゴリズムで実行されるのではなく、シリアルなアルゴリズムが並行して実行されます。
- 通常、データを分割する開始コストは一定のオーバーヘッドを要します。並列に行を処理するメインの作業をデータベースが開始する前に一定量の準備作業を実行する必要があります。この開始時の制約は、データを分割しないその他の方法にはありません。
- データベース・オブジェクトの共通セットで複数のスレッドが同じ操作を実行すると、ロック競合およびラッチ競合が発生する可能性があります。たとえば、32個の別々のサーバー・プロセスがバッファの同じセットを更新しようとしている場合、バッファ・ビジーの待機が起こります。また、複数サーバー・プロセスがおおよそ同じ時間でCOMMIT文を発行している場合、ログ・ファイルの同期待ちが起こります。
- 並行処理は、その他の繰返しの方法と比べて甚大なCPUリソースを消費します。データベース・ホストにスレッドを同時

に処理する十分なコアがない場合、パフォーマンスが悪化します。たとえば、4つのコアのみが32個のスレッドに対して使用可能な場合、一定時間にスレッドがCPUを使用できる可能性は1/8です。

4.3.2 セット・ベース処理

セット・ベース処理は、データベースの内部でデータ・セットを処理するSQL技術です。

セット・ベース・モデルでは、SQL文が結果を定義し、データベースが結果を取得するための最も効果的な方法を判断します。対照的に、繰返しのアルゴリズムは条件ロジックを使用して、データベースからクライアント・アプリケーションに各行または行の各グループをプルし、クライアントでデータを処理してから、データベースにデータを送信して戻します。セット・ベース処理では、データがデータベースから離れないため、ネットワークのラウンドトリップおよびデータベースAPIのオーバーヘッドがなくなります。これは、COMMITの数を減らします。

前述の例と同じ機能要件だとします。次のSQL文は、セット・ベース・アルゴリズムを使用してこの要件を満たします。

```
alter session enable parallel dml;  
insert /*+ APPEND */ into stage1_scan_events d  
  select s.* from ext_scan_events s;  
commit;
```

INSERT文にext_scan_events表の副問合せがあるため、単一のSQL文がすべての行を読み書きします。また、アプリケーションはデータベースがすべての行を挿入した後、単一のCOMMITを実行します。対照的に、繰返しのアプリケーションでは各行または行の各グループの挿入の後にCOMMITを実行します。

セット・ベースの方法は、繰返しの方法に対して次に示す大きなメリットがあります。

- Oracle Real-World Performanceのデモとクラスで示すように、大きなデータ・セットでのパフォーマンスは桁違いに高速です。プログラムの実行時間が数時間から数秒に縮まることは珍しいことではありません。大きなデータ・セットでのパフォーマンスの向上があまりに大きいと、繰返しの方法の正当化が非常に難しくなります。
- 処理速度の劇的な向上の副次的な効果として、DBAが長時間実行のエラーが起きやすいバッチ・ジョブをやめることができ、ビジネス・プロセスをリアルタイムに刷新します。たとえば、6時間のバッチを毎晩実行するかわりに、日中に必要に応じて12秒のジョブを実行すればよくなります。
- アクセス・メソッドではなく、SQLが結果を定義するため、コードの長さは著しく短く(2、3行までに)なります。これは、アプリケーションではなくデータベースが行を分割、取得および操作する最適な方法を決定することを意味します。
- 手動の並列度と対照的に、アプリケーションではなくデータベースがプロセスを管理するため、並列DMLはパフォーマンスが最適化されます。したがって、クライアント・アプリケーションでワークロードを手動で分割する必要がなくなり、各プロセスが同じ時間に終了することが期待できます。
- データ・セットを結合するときには、データベースは相対的に効率の悪いアプリケーション・レベルのループではなく、効率のよいハッシュ結合を自動的に使用します。
- APPENDのヒントはダイレクト・パス・ロードを実行し、これはデータベースがREDOおよびUNDOを作成せず、そのため、I/OおよびCPUの無駄が減ることを意味します。典型的なETLワークロードでは、バッファ・キャッシュが問題の原因となります。バッファ・キャッシュ内でデータを変更し、データとそれに関連するUNDOおよびREDOを書き込んで戻すことは、多くのリソースを消費します。バッファ・キャッシュは必要な速さでブロックを管理できず、ブロックをバッファ・キャッシュに入れて再度戻す操作(通常一度に8 Kのブロック)には高いCPUコストがかかるため、バッファのボリュームについていくために、データベース・ライターおよびサーバー・プロセスの両方が懸命に取り組む必要があります。

セット・ベース処理のデメリット:

- この方法はデータベース開発者に馴染みがなく、そのため難しいものとなります。INSERTの例は比較的簡単です。ただし、複数の外部結合を必要とするような、さらに複雑な文ではさらに難しいアルゴリズムが必要になります。外部結合をパイプライン処理し、WITH句とCASE文を使用することに慣れていない開発者は、将来的に起こりうるセット・ベース・コードを書くことも理解することも考えると気が滅入る可能性があります。
- セット・ベース・モデルは繰返しモデルとはまったく異なるため、変更することはソース・コードを完全書き直すこととなります。対照的に、行ごとのコードを配列ベースのコードに変更することは、これに比べるとささいなことです。

セット・ベース処理のデメリットに関係なく、Oracle Real-World Performanceのグループは、大きなデータ・セットに対するずば抜けたパフォーマンスの向上が、その労力を妥当なものだと信じています。

ビデオ:



- [RWP #7 セット・ベース処理](#)
- [RWP #8: セット・ベース・パラレル処理](#)
- [RWP #9: セット・ベース処理--データの重複排除](#)
- [RWP #10: セット・ベース処理--データ転送](#)
- [RWP #11: セット・ベース処理--データ集約](#)

5 セキュリティ

この章では、データベースとデータベース・アプリケーションに組み込むセキュリティ設計の基礎について説明します。

トピック:

- [権限の付与、ロール、最小限の権限によるユーザー・アクセスの有効化](#)
- [データベース・ログインの自動化](#)
- [ファイングレイン・アクセス制御によるユーザー・アクセスの制御](#)
- [プロシージャとファンクションに対する実行者の権限と定義者の権限の使用](#)
- [アプリケーションに対する外部プロシージャの管理](#)
- [ユーザー・アクティビティの監査](#)

5.1 権限の付与、ロール、最小限の権限によるユーザー・アクセスの有効化

この項では、権限とロールをユーザーに付与し、データへのアクセスを制限する方法について説明します。また、最小限の権限という概念の重要性についても説明し、アプリケーションへのログインを試行するユーザーを自動的にフィルタする方法としてセキュア・アプリケーション・ロールについても紹介します。

ユーザー権限とは、データの更新や削除などのアクションを実行する権利です。それらのアクションを実行する権限をユーザーに付与できます。ロールとは、グループとしてまとめられた権限の名前付きコレクションです。通常は、ユーザーがそのジョブに関連する一連のタスクを実行できるようになります。たとえばclerkというロールでは、事務員がファイルの作成、更新および削除といった作業を実行できるようになります。clerk_mgrロールには、clerkロールが含まれるうえに、事務員の経費報告書を承認したり勤務評定を管理する権限が追加されます。

ユーザーに権限を付与する際には、最小限の権限という原則を適用します。ユーザーには、必要な権限のみを付与するようにします。可能であれば、ユーザーに権限を直接付与しないようにします。かわりに、ユーザーが必要とする一連の権限を定義したロールを作成し、ユーザーにはそのロールを付与します。たとえば、データベース・セッションにログインできるように、ユーザーfredにCREATE SESSION権限を付与します。ただし、UPDATE TABLE権限など、ユーザーがジョブで必要とする権限については、こうした権限を持つロールを付与します。

指定した条件を満たしている場合、ログインを試行するユーザーにロールが自動的に付与されるようにアプリケーションを設計できます。そのためには、セキュア・アプリケーション・ロールを作成します。これは、PL/SQLのプロシージャ(または複数のプロシージャを含むPL/SQLパッケージ)に関連付けられたロールです。このプロシージャによってユーザーが検証され、検証に失敗したユーザーはログインできません。ユーザーが検証を通過した場合は、プロシージャによってユーザーはロールを付与され、アプリケーションを使用できるようになります。ユーザーは、アプリケーションにログインしている間このロールを付与されます。ユーザーがログアウトすると、ロールは取り消されます。

関連項目:

- 権限とロール認可の詳細は、『Oracle Databaseセキュリティ・ガイド』を参照してください
- セキュア・アプリケーション・ロールの詳細は、『Oracle Databaseセキュリティ・ガイド』を参照してください

[例5-1](#)では、営業時間中(午前8時から午後5時)に特定のワークステーション群からユーザーがログインできるセキュア・アプリケーション・ロールのプロシージャを示しています。これらのチェックを通過すると、ユーザーはhr_adminというロールを付与され、

アプリケーションにログインできるようになります。

例5-1 アクセスを営業時間に制限するセキュア・アプリケーション・ロールのプロシージャ

```
CREATE OR REPLACE PROCEDURE hr_admin_role_check
AUTHID CURRENT_USER
AS
BEGIN
  IF (SYS_CONTEXT ('userenv', 'ip_address')
      BETWEEN '192.0.2.10' and '192.0.2.20'
      AND
      TO_CHAR (SYSDATE, 'HH24') BETWEEN 8 AND 17)
  THEN
    EXECUTE IMMEDIATE 'SET ROLE hr_admin';
  END IF;
END;
```

5.2 データベース・ログインの自動化

データベース・ログインを自動化するには、アプリケーションへのログインを試行するユーザーを検証できるPL/SQLプロシージャを実行するログイン・トリガーを作成します。ユーザーがログインすると、このトリガーが実行されます。ログイン・トリガーは、ユーザーが検証を通過しなかった場合にアラートを生成したり、エラー・メッセージを表示するなど、複数のアクションを実行できます。

[例5-2](#)に、PL/SQLプロシージャを実行する単純なログイン・トリガーを示します。

例5-2 ログイン・トリガーの作成

```
CREATE OR REPLACE TRIGGER run_logon_trig AFTER LOGON ON DATABASE
BEGIN
  sec_mgr.check_user_proc;
END;
```

関連項目:

- CREATE TRIGGER文に関する詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください
- データベース・セッションのアプリケーション・コンテキスト・パッケージを実行するログイン・トリガーの作成方法の詳細は、[『Oracle Databaseセキュリティ・ガイド』](#)を参照してください

5.3 ファイングレイン・アクセス制御によるユーザー・アクセスの制御

アプリケーションからのデータにユーザーがアクセスするレベルを制御するには、複数の方法があります。

- Oracle Virtual Private Database (VPD): VPDでは、データベース・アクセスを行と列のレベルで制限できるポリシーを作成できます。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。

たとえば、ユーザーがSELECT * FROM OE.ORDERS文を入力すると仮定します。VPDポリシーは、かわりにこの文を次の文に変換し、オーダーを所有する営業担当者のみがこのデータを表示できるようにします。

```
SELECT * FROM OE.ORDERS
WHERE SALES_REP_ID = SYS_CONTEXT('USERENV', 'SESSION_USER');
```

- Oracle Data Redaction: Oracle Data Redactionは、ユーザーがデータへのアクセスを試行するとき(つまり、問合せの実行時)に、実行時のデータをマスクします。この方法は、データが常に変化する動的な本番システムで効果を

発揮します。データをリダクションするときは、すべてのデータ処理が通常どおりに実行され、バックエンドの参照整合性制約が保持されます。通常、クレジット・カード番号や社会保障番号などの機密データをリダクションします。

データは次の方法でマスクできます。

- 完全なリダクション。データ全体をマスクします。たとえば、37828224という数字はゼロとして表示できます。
- 部分的なリダクション。データの一部のみをリダクションします。このタイプでは、37828224という数字を*****224として表示できます。
- ランダム・リダクション。データはランダム化されたデータとして表示されます。この場合、37828224という数字を93204857として表示できます。
- 正規表現。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。完全または一部のいずれのリダクションでも正規表現を使用できます。たとえば、ドメインのみが表示されるように電子メール・アドレスのユーザー名をリダクションできます(電子メール・アドレスjsmith@example.comのjsmithを[リダクション]に置き換えて、[リダクション]@example.comと表示するなど)。
- リダクションなし。リダクション・ポリシーが定義された表に対する問合せの結果に影響を与えずに、ポリシーの内部動作をテストできます。このオプションを使用して、リダクション・ポリシー定義を本番環境に適用する前にテストできます。
- Oracle Label Security: Oracle Label Securityを使用すると、データベース表を行レベルで保護でき、サイトのニーズに応じて行を様々なセキュリティ・レベルに割り当てることができます。非常に機密性の高いデータを含む行にはHIGHLY SENSITIVEというラベルを割り当て、機密性がそれほど高くないデータを含む行にはSENSITIVEというラベルを割り当てたりできます。すべてのユーザーがアクセスできる行にはPUBLICというラベルを割り当てます。ラベルは必要なだけ作成でき、使用する環境のセキュリティ要件に適合させることができます。

たとえば、低レベル従業員のユーザーfredがログインすると、PUBLICラベルで定義されているように、すべてのユーザーが使用できるデータのみが表示されます。ただし、そのディレクターhortensiaがログインすると、HIGHLY SENSITIVEラベルを割り当てられている機密データをすべて表示できます。

- Oracle Database Vault: Oracle Database Vaultでは、データへの管理アクセスを制限できます。デフォルトでは、管理者(SYSDBA権限を持つユーザーSYSなど)は、データベースのデータすべてにアクセスできます。管理者は通常、パフォーマンス・チューニング、バックアップとリカバリなどのタスクを実行する必要があります。ただし、ユーザーの給与レコードにアクセスする必要はありません。Database Vaultを使用すると、管理者のアクションを制限しつつ、通常の管理アクティビティの実行を妨げないようなポリシーを作成できます。

たとえば典型的なDatabase Vaultポリシーの場合、管理者はHR.EMPLOYEES表にアクセスして変更できません。管理者がログインできる時間の制限、使用できるコンピュータ、動的ツールを使用してデータベースにログインできるかどうかなどの制限を課す微調整済のポリシーを作成できます。さらに、管理者がポリシー違反を試行した場合にアラートを生成するポリシーを作成できます。

関連項目:

- Oracle Virtual Private Databaseの詳細は、『Oracle Databaseセキュリティ・ガイド』を参照してください
- Oracle Data Redactionの詳細は、『[Oracle Database Advanced Securityガイド](#)』を参照してください。
- Oracle Label Securityの詳細は、『[Oracle Label Security管理者ガイド](#)』を参照してください。
- Oracle Database Vaultの詳細は、『[Oracle Database Vault管理者ガイド](#)』を参照してください。

5.4 プロシージャとファンクションに対する実行者の権限と定義者の権限の使用

トピック:

- [実行者の権限と定義者の権限の概要](#)
- [実行者の権限のプロシージャとファンクションを実行するユーザーの保護](#)
- [Javaストアド・プロシージャでのデフォルト権限の処理方法](#)

5.4.1 実行者の権限と定義者の権限の概要

プロシージャまたはファンクション(つまりプログラム・ユニット)を作成するときには、所有者(自分)の権限と、それを呼び出す人の権限のどちらかで実行されるように設計できます。定義者の権限は、所有者の権限を使用してプログラム・ユニットを実行し、実行者の権限は実行する人の権限を使用してプログラム・ユニットを実行します。たとえばユーザーharoldが、表ordersを更新するプロシージャを作成すると仮定します。ユーザーharoldは、このプロシージャに対するEXECUTE権限をユーザーhazelに付与します。haroldが定義者の権限でプロシージャを作成した場合、そのプロシージャはorders表がharoldのスキーマにあると想定します。実行者の権限でプロシージャを作成した場合、そのプロシージャはhazelのスキーマでorders表を検索します。

プログラム・ユニットを定義者の権限または実行者の権限として指定するには、作成文でAUTHID句を使用します。AUTHID句を省略すると、プログラム・ユニットは定義者の権限で作成されます。

[例5-3](#)では、CREATE PROCEDURE文でAUTHID句を使用して定義者の権限または実行者の権限を指定する方法を示しています。

例5-3 定義者の権限または実行者の権限でのプログラム・ユニットの作成

```
CREATE PROCEDURE my_def_proc AUTHID DEFINER -- Definer's rights procedure
AS ...
CREATE PROCEDURE my_inv_proc AUTHID CURRENT_USER -- Invoker's rights procedure
AS ...
```

関連項目:

- 定義者の権限と実行者の権限のプロシージャとファンクションについては、『Oracle Database PL/SQL言語リファレンス』を参照してください
- CREATE PROCEDURE文の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください
- CREATE FUNCTION文の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください

5.4.2 実行者の権限のプロシージャとファンクションを実行するユーザーの保護

実行者の権限でプログラム・ユニットを作成するときには、実行者であるユーザーに付与する権限のレベルを考慮することが重要です。ユーザーharoldはほとんど権限を持たないランクの低い従業員であり、hazelは多くの権限を持つエグゼクティブであると仮定します。hazelがharoldの実行者権限のプロシージャを実行すると、このプロシージャは一時的にhazelの権限(のすべて)を継承します。ただし、このプロシージャの所有者はharoldであるため、haroldはプロシージャを変更でき、そのときharoldを昇格するようなhazelの権限を利用していることにhazelが気付かない可能性があります。このような事態に対する安全策として、haroldの信頼性を確認できたら、実行者権限のプロシージャおよびファンクションがその実行時にhazelの権

限を使用できるように、ユーザーhazelはharoldに権限を付与できます。そのためには、INHERIT PRIVILEGES権限を付与する必要があります。

[例5-4](#)では、実行者ユーザーであるhazelがユーザーharoldにINHERIT PRIVILEGES権限を付与する方法を示しています。

例5-4 プログラム・ユニットへのINHERIT PRIVILEGES作成権限の付与

```
GRANT INHERIT PRIVILEGES ON hazel TO harold;
```

haroldが信頼できないと判明した場合、hazelはharoldのINHERIT PRIVILEGES権限を取り消すことができます。

ユーザーSYSやSYSTEMなどの管理者はINHERIT ANY PRIVILEGES権限を持つため、これらのユーザーの実行者権限プロシージャは、どの実行者ユーザーの権限にもアクセスできます。ANY権限と同様、この権限は信頼できるユーザーにのみ付与してください。

関連項目:

定義者の権限と実行者の権限のプロシージャとファンクションに対するセキュリティ管理の詳細は、『Oracle Databaseセキュリティ・ガイド』を参照してください

5.4.3 Javaストアド・プロシージャでのデフォルト権限の処理方法

デフォルトで、Javaクラス・スキーマ・オブジェクトは、設計者の権限ではなく実行者の権限で実行されます。Javaスキーマ・オブジェクトを定義者の権限で実行する場合には、loadjavaツールを使用してロードするときに-definerオプションを指定します。

[例5-5](#)では、loadjavaコマンドで-definerオプションを使用する方法を示しています。

例5-5 定義者の権限でのJavaクラスのロード

```
loadjava -u joe -resolve -schema TEST -definer ServerObjects.jar  
Password: password
```

-definerオプションは、個々のクラスで使用できます。定義者が異なれば、権限も異なる可能性があります。目的の結果を得られるように、-definerオプションは慎重に適用してください。これにより、必要以上の権限でクラスが実行されないようにできます。

関連項目:

- loadjavaツールの詳細は、『Oracle Database Java開発者ガイド』を参照してください
- Javaアプリケーションにおける現在のユーザーの制御の詳細は、『Oracle Database Java開発者ガイド』を参照してください

5.5 アプリケーションに対する外部プロシージャの管理

セキュリティ上の理由から、Oracleの外部プロシージャはOracle Databaseから物理的に独立したプロセスで実行されます。外部プロシージャを呼び出すと、Oracle Databaseはデータベース・インスタンスのリスナーを起動したユーザーのオペレーティング・システム権限を使用してextprocオペレーティング・システム・プロセス(エージェント)を作成します。

extprocエージェントは、指定したオペレーティング・システムの資格証明として実行するように構成できます。この機能を使用

するには、extprocプロセスに関連付ける資格証明を定義し、それによって偽装ユーザーを認証(指定されたユーザー資格証明のかわりに実行)してから、ユーザー定義の共有ライブラリをロードしてファンクションを実行します。extprocユーザー資格証明を構成するには、PL/SQLパッケージDBMS_CREDENTIALとPL/SQL文CREATE LIBRARYを使用します。

関連項目:

外部プロシージャ保護の詳細は、『Oracle Databaseセキュリティ・ガイド』を参照してください

5.6 ユーザー・アクティビティの監査

データベースにおける特定のアクションを監査する監査ポリシーを作成できます。Oracle Databaseは次に、これらのアクションを監査証跡に記録します。データベースは必ずなんらかのアクションを監査し、それも監査証跡に書き込みます。作成する監査ポリシーは、特定のユーザーごとにすべてのアクションを監査するような単純なポリシーでも、特定の条件をテストして条件違反がある場合にアラートを電子メールで送信するような複雑なポリシーでもかまいません。

Oracle Databaseをインストールするとき、データベースの監査方法を選択できます。

- **統合監査:** 統合監査では、すべての監査証跡が1つの監査証跡に書き込まれ、V\$UNIFIED_AUDIT_TRAILおよびGV\$UNIFIED_AUDIT_TRAILの動的ビューによって表示されます。この監査証跡は、Oracle Database固有のアクションのみでなく、Oracle Real Application Security、Oracle Recovery Manager、Oracle Database Vault、Oracle Label Securityの各環境で実行されるアクションも対象にします。これらのソースからの監査レコードが、一貫した統一フォーマットで表示されます。名前付きの監査ポリシーを作成し、必要に応じてその有効/無効を切り替えることができます。統合監査を使用する場合は、データベースをそれに移行する必要があります。
- **統合監査と、リリース12cより前の監査の混在:** ほとんどの場合、このオプションでリリース12cより前の監査を使用できるようになり、監査レコードは独自のフォーマットを使用して別の場所には書き込まれます。この場合でも、マルチテナント環境での監査の使用など、リリース12cの機能は使用できます。このタイプの監査は、新規データベースとアップグレードしたデータベースのどちらでもデフォルトです。

どちらの場合も、データベースをOracle Database 12cリリース1 (12.1.0.1)にアップグレードするとき、前のリリースからの監査レコードは維持されます。全面的に統合監査に移行する場合は、以前のレコードをアーカイブしてから、監査証跡をパージできます。移行が完了すると、新しい監査レコードは統合監査証跡に書き込まれるようになります。

関連項目:

- 統合監査ポリシーの作成と管理の詳細は、『Oracle Databaseセキュリティ・ガイド』を参照してください
- 統合監査とリリース12cより前の監査の詳細な比較は、『Oracle Databaseセキュリティ・ガイド』を参照してください
- 統合監査への移行の詳細は、[Oracle Databaseアップグレード・ガイド](#)を参照してください。

6 高可用性

この章では、データベースとデータベース・アプリケーションに高可用性を組み込む方法について説明します。

トピック:

- [透過的アプリケーション・フェイルオーバー\(TAF\)](#)
- [Traffic DirectorモードのOracle Connection Manager](#)
- [高速アプリケーション通知\(FAN\)と高速接続フェイルオーバー\(FCF\)](#)
- [アプリケーション・コンティニューイティとトランザクション・ガード](#)
- [データベース・クラウドのサービスおよびロード管理](#)

6.1 透過的アプリケーション・フェイルオーバー(TAF)

この項では、透過的アプリケーション・フェイルオーバー(TAF)の概要とTAFの構成方法、およびTAFコールバックを使用してイベントの発生時にそのイベントをアプリケーションに通知する方法について説明します。

トピック:

- [透過的アプリケーション・フェイルオーバーの理解](#)
- [透過的アプリケーション・フェイルオーバーの構成](#)
- [透過的アプリケーション・フェイルオーバー・コールバックの使用](#)

6.1.1 透過的アプリケーション・フェイルオーバーの理解

透過的アプリケーション・フェイルオーバー(TAF)はOCI、OCCI、Java Database Connectivity (JDBC) OCIのドライバおよびODP.NETのクライアント側機能であり、インスタンスまたはネットワークの障害によりデータベース接続に失敗したときに発生するエンドユーザー・アプリケーションに対する障害を最小限にするために設計されています。TAFは、Oracle Real Application Clusters (Oracle RAC)やOracle Data Guardの物理スタンバイ・データベースなどの様々なシステム構成でも、起動後の単一インスタンス・システムでも実装できます(たとえば、修復を行うとき)。

TAFにより、クライアント・アプリケーションは、事前に構成した2次インスタンスに透過的に再接続し、最初の元のインスタンスで確立したのと新たに同じ接続を自動的に作成します。つまり、接続が失われた状態にかかわらず、接続プロパティは直前の接続のプロパティと同じです。この場合、アクティブ・トランザクションはロールバックされます。また、試行の失敗後にアプリケーションが使用しようとしたすべての文もフェイルオーバーします。

関連項目:

- OCI TAFの詳細は、『Oracle Call Interfaceプログラマーズ・ガイド』を参照してください
- OCCI TAFの詳細は、『Oracle C++ Call Interfaceプログラマーズ・ガイド』を参照してください
- JDBC TAFの詳細は、『Oracle Database JDBC開発者ガイド』を参照してください
- ODP .NET TAFの詳細は、『[Oracle Data Provider for .NET開発者ガイドfor Microsoft Windows](#)』を参照してください

- TAF (接続データ・セクション1)のクライアント側における構成の詳細は、『Oracle Database Net Servicesリファレンス』を参照してください。
- TAF(DBMS_SERVICE)のサーバー側における構成の詳細は、『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

6.1.2 透過的アプリケーション・フェイルオーバーの構成

TAFは、クライアント側でもサーバー側でも構成でき、クライアント側とサーバー側の両方に構成されている場合には、サーバー側が優先されます。クライアント側でTAFを構成するには、接続記述子のCONNECT_DATAの部分にFAILOVER_MODEパラメータを含めます。サーバー側でTAFを構成するには、DBMS_SERVICE.MODIFY_SERVICEパッケージ・プロシージャを使用してターゲット・サービスを変更します。

関連項目:

- 構成の詳細は、[「透過的アプリケーション・フェイルオーバーの理解」](#)を参照してください。

6.1.3 透過的アプリケーション・フェイルオーバー・コールバックの使用

TAFコールバックとは、フェイルオーバーが発生したときに登録され、イベントの発生時にそのイベントをアプリケーションに通知するためにフェイルオーバー時にコールされるコールバックです。ユーザーのセッションを再構築する際に何回かコールされます。

アプリケーションの開発者は、フェイルオーバーが進行中であることをユーザーに通知する必要がある場合があります。フェイルオーバーの進行時には、若干の遅延が発生するためです。この機能は、最初のコールバックへのコールによって実行されます。また、フェイルオーバーが成功して接続が再確立した場合には、フェイルオーバーが発生したことをユーザーに通知してから、ALTER SESSIONコマンドをリプレイする必要がある場合があります。このコマンドは、2つ目のインスタンスで自動的にリプレイされないためです。2回目のコールバックへのコールにより、その機能が実行されます。また、フェイルオーバーが失敗した場合には、フェイルオーバーが発生しなかったことをアプリケーションに通知します。3回目のコールバックへのコールでも、その機能が実行されます。

TAFコールバックを使用すると、次のことが可能になります。

- フェイルオーバー・プロセスを通じて、フェイルオーバーのステータスが進行中、成功、失敗のいずれであるかをユーザーに通知します
- 必要なときにALTER SESSIONコマンドをリプレイします
- 新しい接続上でセッションが開始されるたびに、プライマリ・ハンドル以外のユーザー・ハンドルを再認証します。各ユーザー・ハンドルはサーバー側セッションを表すため、クライアントでは、そのセッションのためにALTER SESSIONコマンドを再実行する場合があります。

関連項目:

各インタフェースの具体的なコールバック登録情報は、[「透過的アプリケーション・フェイルオーバーの構成」](#)を参照してください。

6.2 Traffic DirectorモードのOracle Connection Manager

この機能により、Oracle Database Connection Manager (CMAN)をTraffic Directorモードで構成し、ルーター・レベルで構成可能なHA機能とパフォーマンス機能を使用して、異なるデータベース・サービスに接続しているクライアントにサービスを

提供することができ、接続しているすべてのクライアントに利点があります。

Oracle Databaseリリース18.1以降、Traffic DirectorモードのOracle Connection Managerでは次に対するサポートを提供しています。

- 透過的パフォーマンス強化および接続多重化
 - Traffic DirectorモードのCMANインスタンスが複数ある場合、アプリケーションは、クライアント側の接続時ロード・バランシングまたはロード・バランサ(BIG-IP、NGINXなど)を通じてスケーラビリティが増加します
- 計画済データベース・メンテナンスまたはプラグブル・データベース(PDB)再配置、およびほとんどが読取りのワークロードに対する計画外データベース停止を含むアプリケーション停止時間ゼロ。
- 単一点障害を回避するTraffic DirectorモードのCMANの高可用性。
- セキュリティおよび分離: Traffic DirectorモードのCMANは次の機能を提供します。
 - Transmission Control Protocol/Transmission Control Protocol Secure (TCP/TCPs)およびプロトコル変換をサポートするデータベース・プロキシ
 - IPアドレス、サービス名、およびSecure Socket Layer/Transport Layer Security (SSL/TLS)ウォレットに基づくファイアウォール
 - マルチテナント環境でのテナント分離
 - サービス妨害およびファジング攻撃からの保護
 - オンプレミスのOracle DatabaseとOracle Cloudの間のデータベース・トラフィックの安全なトンネリング

関連トピック

- [『Oracle Databaseセキュリティ・ガイド』](#)
- [Oracle Database Net Services管理者ガイド](#)
- [Oracle Database Net Services管理者ガイド](#)
- [Oracle Database Net Services管理者ガイド](#)

6.3 高速アプリケーション通知(FAN)と高速接続フェイルオーバー(FCF)

この項では、高速アプリケーション通知(FAN)および高速接続フェイルオーバー(FCF)の概要と、高可用性環境でアプリケーションがFANイベントに応答する方法、FCFを使用してフェイルオーバー後に接続を再配置する方法について説明します。

トピック:

- [高速アプリケーション通知\(FAN\)について](#)
- [FANイベント通知の受信について](#)
- [高速接続フェイルオーバー\(FCF\)について](#)

6.3.1 高速アプリケーション通知(FAN)について

高可用性の重要な要素が、高速アプリケーション通知(FAN)と呼ばれる通知のメカニズムです。FANは、UPイベントやDOWNイベントなどのサービス・ステータス変更をはじめとして、構成とサービス・レベルの情報を他のプロセスに通知します。アプリケーションはFANイベントに응答して、すぐにアクションを実行できます。FANの UP イベントおよび DOWN イベントは、インスタンス、サービスおよびノードに適用できます。

FANには、インスタンスまたはサーバーで障害が発生した場合に、アクティブなトランザクションを即座に終了する機能が備えられています。FAN統合されているOracle Clientは、イベントおよび応答を受信します。アプリケーションによる応答は、エラーをユーザーに伝播するか、またはトランザクションを再実行しアプリケーション・ユーザーからエラーをマスクすることによって実行されます。DOWNイベントが発生すると、統合されているFANクライアントでは、終了されたデータベースへの接続が即座にクリーン・アップされます。UPイベントが発生すると、FAN統合クライアントでは新しいプライマリ・データベース・インスタンスへの新しい接続が作成されます。

FANは、一般的なOracleクライアント・ドライバの多くに統合されています。したがって、FANを使用する最も簡単な方法は、次のいずれかの統合されているOracleクライアントを使用することです。

- OCIセッション・プール
- Universal Connection Pool for Java
- Thin JDBC Driver (12.2以降)
- ODP.NET管理対象と管理対象外プロバイダ
- すべてのWebLogic Serverデータ・ソースとOracle Tuxedo

この機能は、使用可能なプライマリ・データベースにアプリケーションからいつでも一貫して接続できるようにすることを目的としています。

FANイベントは、Oracle Notification Serviceを使用して発行されます。発行メカニズムは、Oracle RACのインストール時に自動的に構成されます。ここで、Oracle RACがインストールされているということは、Oracle RACに伴うOracleクラスタウェア、Oracle RAC One Node、Oracle Data Guard (ファストスタート・フェイルオーバー)、またはOracleクラスタウェアに伴うOracle Data Guard単一インスタンスがインストールされていることを意味します。Oracle Database 12c リリース (12.1)以降では、ONSが新しいクライアント(Oracle Database 12c リリース1 (12.1))と新しいサーバー(Oracle Database 12c リリース1 (12.1))に対する第一の通知メカニズムになっています。AQ HAの通知機能は非推奨になり、古いOCIやODP.NET管理対象外クライアント(Oracle Database 12c リリース1 (12.1)より前)または古いサーバー(Oracle Database 12c リリース1 (12.1)より前)が使用されている場合の、下位互換性のためにのみ維持されます。

JDBCまたはOracle Database 12 c リリース1 (12.1.0.1) OCIまたはODP.NETのクライアントを使用する場合、Oracle Notification ServiceはユーザーのTNSを使用して自動的に構成されます。OCIベースのクライアントを使用する場合は、サービスのHA通知(-notification = TRUE)を設定し、oraccess.xmlにEVENTSを設定します。

関連項目:

- FAN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
- Oracle Restart環境でFANイベントを有効にする方法については、『Oracle Database管理者ガイド』を参照してください
- Oracleクラスタウェアを設定して有効にしたOracle RAC環境で、ランタイム接続ロード・バランシングのアドバイザ FANイベントを受信し、アプリケーションのセッション要求を均等に分散する方法の詳細は、『Oracle Call Interface プログラマーズ・ガイド』を参照してください
- Oracle RACのロード・バランシング・アドバイザによって分散されるサービス・メトリックを使用する、ステートレス接続プールのランタイム・ロード・バランシングについては、『Oracle C++ Call Interfaceプログラマーズ・ガイド』を参照してください
- 高速接続フェイルオーバーの詳細は、『Oracle Database JDBC開発者ガイド』を参照してください

6.3.2 FANイベント通知の受信について

Oracle Database 12c リリース2 (12.2)以降では、ユニバーサル接続プールまたはOracle WebLogic ServerとActive Grid Link (AGL)を使用しない場合でも、Oracle RAC FAN APIを使用すると、Oracle Databaseの高可用性(HA)機能を利用できます。この機能はOracle Notification System(ONS)メッセージ・トランスポート・メカニズムに依存しています。

ONSを使用するには、システム、サーバーおよびクライアントで構成が必要です。Oracle RAC高速アプリケーション通知を使用するには、`simplefan.jar`ファイルがCLASSPATHに存在するか、`ons.jar`ファイルがCLASSPATHに存在するか、Oracle Notification Services (ONS)クライアントがクライアント・システムにインストールされ、稼働している必要があります。

関連項目:

Oracle RAC FAN APIの詳細は、[『Oracle Database JDBC開発者ガイド』](#)を参照してください。

6.3.3 高速接続フェイルオーバー(FCF)について

スタンバイ・データベースが存在する構成では、Oracle Notification Services(ONS)をOracle Restart構成に追加した後でサービスに対してOracle Advanced Queuing (AQ) HA通知を有効にすると、クライアントで高速接続フェイルオーバー(FCF)を有効化できます。クライアントでは、FANイベントを受信し、Oracle Data Guardのフェイルオーバーの後で現在のプライマリ・データベースへの接続を再配置できます。Oracle Database 12cリリース1 (12.1)からは、ONSが新しいクライアント(Oracle Database 12cリリース1 (12.1))と新しいサーバー(Oracle Database 12cリリース1 (12.1))に対する第一の通知メカニズムです。AQ HAの通知機能は非推奨となり、古いクライアント(Oracle Database 12cリリース1 (12.1)より前)または古いサーバー(Oracle Database 12cリリース1 (12.1))が使用されている場合の下位互換性のためにのみ維持されます。

スタンバイ・データベースが構成されていないデータベースでも、クライアントのFANイベントを構成できます。停止状態(計画済みまたは計画外)が発生したとき、データベースへの接続を再試行するようにクライアントを構成できます。障害が発生したデータベースはOracle Restartによって再起動されるため、データベースが再起動した時点でクライアントは再接続できます。

Oracle Data Guard環境、またはスタンバイ・データベースのないスタンドアロン環境で、FCFに対してFAN統合クライアントのサポートを使用するには、FANイベントを有効にする必要があります。

FCFは、Oracle Databaseの接続フェイルオーバー機能を、Java Database Connectivity (JDBC)アプリケーションがドライバに依存せずに利用できる方法を提供します。FCFはUniversal Connection Pool (UCP)およびOracle RACと統合され、高可用性イベント通知の機能を提供します。

OCIクライアントは、Oracle Restart高可用性FANイベントの通知を受信し、イベント発生時に応答するように登録することで、FCFを有効にできます。これにより、OCIのセッション・フェイルオーバーのレスポンス時間が向上し、中断した接続を接続プールおよびセッション・プールから削除できます。この機能は、透過アプリケーション・フェイルオーバー(TAF)、接続プールまたはセッション・プールを使用するものも含めたOCIアプリケーションで動作します。

関連項目:

- Oracle RACのロード・バランシング・アドバイザによって分散されるサービス・メトリックを使用する、ステートレス接続プールのランタイム・ロード・バランシングについては、『Oracle C++ Call Interfaceプログラマーズ・ガイド』を参照してください

- 高速接続フェイルオーバーの詳細は、『Oracle Database JDBC開発者ガイド』を参照してください
- [Oracle Universal Connection Pool for JDBC Java APIリファレンス](#)
- JDBCクライアントに対してFCFを有効にする方法については、『Oracle Database管理者ガイド』を参照してください
- OCIクライアントに対してFCFを有効にする方法については、『Oracle Database管理者ガイド』を参照してください
- ODP.NETクライアントに対してFCFを有効にする方法については、『Oracle Database管理者ガイド』を参照してください

6.4 アプリケーション・コンティニューイティとトランザクション・ガード

アプリケーション・コンティニューイティは、フェイルオーバーのためのDBA機能です。トランザクション・ガードは、フェイルオーバーのコードを独自に作成するための、開発者用の機能です。

Oracle高可用性フレームワークでは、JDBCクライアント、OCIクライアントおよびODP.NETクライアントで高速アプリケーション通知(FAN)メッセージがサポートされます。FANは、ノード、データベース、インスタンス、サービス、およびパブリック・ネットワーク・レベルの停止を迅速にアプリケーションに通知するように設計されています。障害が通知されると、アプリケーションは障害が発生した接続を正常なインスタンスで再確立できます。

アプリケーション・コンティニューイティはアプリケーションに対して透過的です。この機能は、Oracle Database 12cとOracleドライバによって提供されます。これは、データベース・サービスの属性を設定することで有効になります。

関連項目:

- アプリケーション・コンティニューイティの詳細は、『[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。
- トランザクション・ガードの詳細は、『[トランザクション・ガードの使用](#)』を参照してください
- OCI、トランザクション・ガードおよびアプリケーション・コンティニューイティの詳細は、『[Oracle Call Interfaceプログラマーズ・ガイド](#)』を参照してください。
- JDBC、トランザクション・ガードおよびアプリケーション・コンティニューイティの詳細は、『[Oracle Database JDBC開発者ガイド](#)』を参照してください

6.4.1 アプリケーション・コンティニューイティの概要

アプリケーション・コンティニューイティは、データベース・セッションのトランザクション状態および非トランザクション状態の再構築を試行することによって、計画済停止または計画外停止(データベース・セッションが使用できない原因になる)をマスクします。そのため、停止状態になってもユーザーには実行が遅延するようには見えなくなります。

アプリケーション・コンティニューイティは、Oracle Database 12c以降のサーバーと連携して、データベース・セッションがリプレイ可能かどうかを判定します。リカバリ可能なエラーが発生し、データベース・セッションが使用不可になると、エラー・メッセージがアプリケーションに返送されます。ドライバはFANメッセージ(停止または割込み)を受信して、デッド・セッションを中断します。

最後の送信でリプレイが有効な場合、12cドライバは送信のリプレイを準備し、保存された文をリクエストのためにリプレイします。アプリケーション・コンティニューイティは、トランザクション・ガードを使用して、エラーを受信したセッションから送信された前回の操作の結果を確認することによってリプレイを準備します。送信がコミットされて完了すると、新しいセッションはこの結果をアプリケーションに返し、SESSION_STATE_CONSISTENCYモードがSTATICの場合には非トランザクション状態を確立して続行し、SESSION_STATE_CONSISTENCYモードがDYNAMICの場合には終了します。DYNAMICセッション状態の一貫性は、ほと

んだのアプリケーションに適しています。

FAILOVER_RESTOREがLEVEL1か、コールバックが設定されている場合、クライアント(JDBC replay driver, ODP.NET or OCI)は接続を初期化して、初期の非トランザクション・セッション状態(NTSS)をリストアします。リプレイするとき、権限が付与されている場合には、保持されている可変データがリストアされます。クライアントで目に見える結果が元の送信と等しくなるように、サーバーで検証が実行されます。リプレイが完了すると、アプリケーションはアプリケーション・ロジックを続行してランタイム・モードに戻るため、負荷時に発生するのと類似した実行遅延しか起こらなかったかのように見えます。

場合によっては、リプレイはクライアントが決定に使用したデータをリストアできないことがあります。その場合、リプレイは元のエラーをアプリケーションに返し、遅延エラーのように見えます。

アプリケーション・コンティニューイティは、同じDBID (時間上では先行)を持つデータベースのコピーに対して、あるいはActive Data Guardファーム内で、データベースが使用できないことによる停止をリカバリできます。これは、Active Data Guard内のOracle RAC One、Oracle Real Application Clustersであることも、1つのRACまたは複数のRAC全体、またはActive Data Guard (ADG)へのPDB再配置を使用するMultitenantであることもあります。

関連項目:

- アプリケーション・コンティニューイティの詳細は、[『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。

6.4.2 トランザクション・ガードの概要

トランザクション・ガードは信頼性の高いプロトコルとインターフェイスであり、エラーまたはタイムアウトがクライアントに返されたときに、現在進行中のトランザクションのコミット結果を返します。アプリケーションはトランザクション・ガードのインタフェースを利用して、正常にリカバリ可能なエラー処理をコード化することができます。停止中に明確なメッセージを提供することで、ユーザー・エクスペリエンスが大幅に向上します。

トランザクション・ガードでは、最大1回のトランザクション実行のセマンティクスという概念が導入されており、これはトランザクションの冪等性とも呼ばれます。アプリケーションがこのサービスを使用してデータベースへの接続をオープンすると、認証時に論理トランザクションID (LTXID)が生成され、データベースではセッション・ハンドルに格納され、クライアント・ドライバではコピーが格納されます。これはデータベース・トランザクションをアプリケーションの観点から識別する、グローバルに一意なIDです。アプリケーションはトランザクション・ガードのインタフェースを使用して、リカバリ可能なエラーに続く既知のコミット結果を取得します。

停止したとき、トランザクション・ガードを使用しているアプリケーションは以前に失敗したセッションのハンドルからLTXIDを取得し、セッション障害が起こる前にアクティブだったトランザクションの結果をそれによって判定できます。LTXIDがUNCOMMITTEDと判定された場合、アプリケーションはUNCOMMITTEDという結果をユーザーに返して実行が必要なアクションを決定するか、オプションとして、コミットされていないトランザクションをリプレイすることもできます。LTXIDがCOMMITTEDと判定された場合、トランザクションがコミットされ、アプリケーションはその結果をエンドユーザーに返すことができ、新しい接続を取得して続行することも可能です。トランザクション・ガードは、直前のユーザー・コールがCOMMITTEDかどうかのみでなく、必要な非トランザクション状態の変更を完了したかどうかをレポートします。USER_CALL_COMPLETEDを参照してください。

関連項目:

[トランザクション・ガードの使用](#)

6.5 データベース・クラウドのサービスおよびロード管理

トピック:

[データベース・クラウドのサービスおよびロード管理について](#)

6.5.1 データベース・クラウドのサービスおよびロード管理について

データベース・クラウドは、サービスおよびロード管理のフレームワークによって統合され、高いパフォーマンスと可用性、そしてリソースの最適な利用が保証される自己完結型のデータベース・システムです。このフレームワークでは、ローカルにも、また地理的に分散した地域のデータ・センターでも同期された複数のレプリカを保持する分散データベース間で、処理のワークロードが効果的に均衡されます。レプリカは、Oracle RAC環境のインスタンスの場合もあれば、Oracle Data GuardやOracle Golden Gate、あるいはレプリケーション・テクノロジーに対応する任意の組合せを使用して相互接続された単一のインスタンスの場合もあります。したがって、サービスおよびロード管理のフレームワークでは、動的なロード・バランシング、フェイルオーバー、およびレプリケートされたデータベースで一元化されたサービス管理が可能です。

グローバル・サービスは、なんらかの形のデータベース・レプリケーションによって同期される複数のデータベースによって実現され、サービスに必要なサービス品質を満たすデータベース・サービスです。これによって、サービスに対するクライアント・リクエストは、そのサービスを提供するいずれかのデータベースに転送されるようになります。

データベース・クラウド内のデータベース・プールは、同じ管理ドメインに属して同じグローバル・サービスを提供するすべてのデータベースで構成されます。データベース・クラウドは複数のデータベース・プールにパーティション化されるため、各プールを個別の管理者が管理することによってサービス管理が容易になり、セキュリティのレベルも向上します。

グローバル・サービス・マネージャ(GSM)は、サービス・レベルのロード・バランシング機能と、データベース・クラウドにおける一元的なサービス管理の機能を持つソフトウェア・コンポーネントです。ロード・バランシングは、接続時と実行時に行われます。GSMには他にも、サービスを作成、構成、開始、停止および再配置したり、カーディナリティやリージョンの局所性といったグローバル・サービスのプロパティを維持する機能もあります。リージョンは、データベース・クライアントとサーバーを含むデータ・センターとして知られる論理上の境界です。このときクライアントとサーバーは、そのデータ・センターにアクセスするアプリケーションで求められるレベルまでネットワーク待機時間を小さくできる程度に接近しているとみなされます。

GSMは、個別のホスト上で実行されていても、データベース・インスタンスと同じ場所にあってもかまいません。各リージョンでは、少なくとも1つのGSMインスタンスが実行されている必要があります。高可用性が必要な場合は、1つのリージョンに複数のGSMインスタンスをデプロイすることをお勧めします。1つのGSMインスタンスは、特定のリージョンのみに属しますが、そのリージョンに関連付けられているすべてのデータベース・プールでグローバル・サービスを管理します。

アプリケーション開発者の観点から見ると、クライアント・プログラムはリージョンのグローバル・サービス・マネージャ(GSM)に接続して、グローバル・サービスに対する接続をリクエストします。クライアントはデータベースまたはインスタンスを指定する必要はありません。GSMは、サービスを提供しているデータベース・クラウドのデータベース・プール内で最適なインスタンスに、クライアントのリクエストを転送します。

Oracle Database 12cリリース1 (12.1.0.1)以降、DBAは、Global Data Services (GDS)フレームワークでOracle Net文字列を使用して、データベース・サービスのクライアント側接続文字列を構成できます。

Oracle Database 12cリリース1 (12.1.0.1)で導入された論理トランザクションID (LTXID)は、最初に認証時に生成されてセッション・ハンドルに格納され、アプリケーションの観点からデータベース・トランザクションを識別するために使用されます。論理トランザクションIDはグローバルに一意であり、高可用性(HA)インフラストラクチャの中でトランザクションを識別します。

HAフレームワークを使用すると、クライアント・アプリケーション(JDBC、OCI、ODP.NET)で高速アプリケーション通知(FAN)メッセージがサポートされます。FANは、ノード、データベース、インスタンス、サービス、およびパブリック・ネットワーク・レベルの停止

を迅速にアプリケーションに通知するように設計されています。停止が通知されると、アプリケーションは障害が発生した接続を正常なインスタンスで再確立できます。

Oracle Database 12cリリース1 (12.1.0.1)から、DBAはアプリケーションが使用するデータベース・サービスのサーバー側設定を構成して、Java用のアプリケーション・コンティニューイティとトランザクション・ガードをサポートできるようになりました。

関連項目:

- グローバル・サービス管理の概要と、サービスおよびロード管理フレームワークの物理コンポーネントおよび論理コンポーネントの詳細は、『Oracle Database概要』を参照してください
- データベース・クラウドにおけるグローバル・サービス管理の詳細は、Oracle Database Global Data Services概要および管理ガイドを参照してください
- Global Data Services (GDS)フレームワークへの接続に関するデータベース・クライアントの構成の詳細は、[Oracle Database Global Data Services概要および管理ガイド](#)を参照してください。
- OCI、アプリケーション・コンティニューイティおよびトランザクション・ガードの詳細は、『Oracle Call Interfaceプログラマーズ・ガイド』を参照してください
- JDBC、アプリケーション・コンティニューイティおよびトランザクション・ガードの詳細は、『Oracle Database JDBC開発者ガイド』を参照してください

7 高度なPL/SQL機能

この章では、高度なPL/SQL機能について紹介し、詳細は他の章やマニュアルへの参照を示しています。

トピック:

- [PL/SQLのデータ型](#)
- [動的SQL](#)
- [PL/SQL最適化レベル](#)
- [システム固有の実行のためのPL/SQLユニットのコンパイル](#)
- [例外処理](#)
- [条件付きコンパイル](#)
- [バルク・バインド](#)

関連項目:

- [アプリケーション開発者用のPL/SQL](#)
- 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。

7.1 PL/SQLのデータ型

PL/SQLデータ型には、SQLデータ型、追加のスカラー・データ型、およびコンポジット・データ型があります。コンポジット・データ型は、ユーザーが定義します。スカラー・データ型のサブタイプも定義できます。

関連項目:

[PL/SQLのデータ型](#)

7.2 動的SQL

動的SQLは、実行時にSQL文を生成して実行するためのプログラミング方法です。この方法は、非定型の問合せシステムのような柔軟性がある汎用目的のプログラムを記述する場合、データベース定義言語(DDL)文を実行する必要があるプログラムを記述する場合、またはコンパイル時にSQL文のテキスト全体またはそのSQL文の入力変数および出力変数の数またはデータ型が不明な場合に有効です。

関連項目:

[PL/SQLの動的SQL](#)

7.3 PL/SQL最適化レベル

PL/SQL最適化レベルは、PL/SQLオプティマイザがコードを再調整してパフォーマンス向上を図ることのできる程度を決定しま

す。このレベルは、コンパイル・パラメータPLSQL_OPTIMIZE_LEVELを使用して設定します。

関連項目:

- [Oracle Databaseリファレンス](#)
- [PLSQL_OPTIMIZE_LEVELコンパイル・パラメータ](#)

7.4 システム固有の実行のためのPL/SQLユニットのコンパイル

通常、PL/SQLユニットをコンパイルして、システム固有のコード(プロセッサに依存するシステム・コード)にすると、PL/SQLユニットをスピードアップできます。システム固有のコードは、SYSTEM表領域に格納されます。

関連項目:

システム固有の実行のためのPL/SQLユニットのコンパイルの詳細は、[「システム固有の実行のためのPL/SQLユニットのコンパイル」](#)を参照してください。

7.5 例外処理

例外(PL/SQLランタイム・エラー)は、設計の失敗、コーディングの間違い、ハードウェアの障害などの多くの原因で発生する可能性があります。発生する可能性があるすべての例外を予想することはできませんが、例外が発生してもプログラムで処理を継続できるように例外ハンドラを記述できます。

関連項目:

[Oracle Database PL/SQL言語リファレンス](#)

7.6 条件付きコンパイル

条件付きコンパイルを使用すると、ソース・テキストを削除することなく、PL/SQLアプリケーションの機能をカスタマイズできます。たとえば、次のことが可能です。

- 最新のデータベース・リリースで新機能を使用し、古いデータベース・リリースでアプリケーションを実行する場合にそれらの新機能を無効化することができます。
- 開発環境でデバッグ文またはトレース文をアクティブ化し、本番サイトでアプリケーションを実行する場合にそれらの文を隠ぺいすることができます。

ただし、次の点に注意してください。

- 条件付きコンパイルを使用して型の属性構造を変更すると、依存オブジェクトが非同期になったり、依存表がアクセス不可になる場合があるため、推奨されません。
型の属性構造を変更するには、依存オブジェクトに変更を伝播するALTER TYPE文を使用することをお勧めします。
- 条件付きコンパイルには、制約があります。

関連項目:

[『Oracle Database PL/SQL言語リファレンス』](#)。

[Oracle Database SQL言語リファレンス](#)

7.7 バルク・バインド

バルク・バインドを利用すると、PL/SQLとSQLの間で通信のパフォーマンス・オーバーヘッドが最小限になり、パフォーマンスが大幅に向上します。

関連項目:

[バルク・バインドの概要](#)

第II部 アプリケーション開発者用のSQL

ここでは、アプリケーション開発者がOracle Databaseで情報を管理する場合に使用されるStructured Query Language(SQL)について必要となる情報を説明します。

章:

- [アプリケーション開発者用のSQL処理](#)
- [データベース・アプリケーションにおけるSQLデータ型の使用](#)
- [データベース・アプリケーションでの正規表現の使用](#)
- [データベース・アプリケーションでの索引の使用](#)
- [データベース・アプリケーションにおけるデータ整合性のメンテナンス](#)

関連項目:

SQLの詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。

8 アプリケーション開発者用のSQL処理

この章では、Oracle DatabaseでのSQL文の処理方法について、アプリケーション開発者が知っておく必要がある内容について説明します。

トピック:

- [SQL文の処理の説明](#)
- [操作のトランザクションへのグループ化](#)
- [読取り専用トランザクションでの反復可能読取りの保証](#)
- [明示的な表のロック](#)
- [Oracle Lock Managementサービス\(ユーザー・ロック\)の使用](#)
- [シリアライズ可能トランザクションを使用した同時実行性の制御](#)
- [非ブロック化およびブロック化のDDL文](#)
- [自律型トランザクション](#)
- [記憶域割当てエラー後の実行の再開](#)

関連項目:

[『Oracle Database概要』](#)

8.1 SQL文の処理の説明

ここでは、SQL文の実行時に処理の各ステージで何が行われるかを、例としてデータ操作言語(DML)文を使用して説明します。

ある部門の全従業員の昇給を行うためにPro*Cプログラムを使用しているとします。プログラムはOracle Databaseに接続しており、操作を行うユーザーはemployees表を所有するHRスキーマに接続しています。プログラムには次のSQL文を埋め込むことができます。

```
EXEC SQL UPDATE employees SET salary = 1.10 * salary
WHERE department_id = :department_id;
```

プログラムでは、SQL文の実行時に使用するバインド変数プレースホルダ:department_idの値が提供されます。

トピック:

- [SQL文の処理のステージ](#)
- [共有SQL領域](#)

8.1.1 SQL文の処理のステージ



ノート:

DML 文はすべてのステージを使用します。トランザクション管理、セッション管理およびシステム管理の SQL 文が使用するの、ステージ 2 およびステージ 8 のみです。

1. カーソルをオープンまたは作成します。

プログラム・インタフェース・コールにより、SQL文を予想して、カーソルがオープンまたは作成されます。ほとんどのアプリケーションではカーソルの作成は暗黙的(自動的)に行われます。プリコンパイラ・プログラムでは、カーソルを暗黙的または明示的に作成します。

2. 文を解析します。

SQL文がユーザー・プロセスからOracle Databaseに渡され、文の解析済表現が共有SQL領域にロードされます。Oracle Databaseは、解析中に多くのエラーを捕捉できます。



ノート:

データ定義言語(DDL)文では、解析にデータ・ディクショナリの参照と実行が含まれます。

3. 文が問合せかどうかを判別します。

4. 文が問合せの場合、結果を記述します。



ノート:

このステージが必要になるのは、結果の性質が不明な場合のみです。たとえば、ユーザーが対話的に問合せを入力する場合などです。

Oracle Databaseによって結果の性質(データ型、長さおよび名前)が決まります。

5. 文が問合せの場合、出力を定義します。

フェッチされる各値を受け取るために定義する変数の場所、サイズおよびデータ型を指定します。このような変数は**定義変数**と呼ばれます。必要な場合にはOracle Databaseによってデータ型の変換が行われます。

6. 変数がある場合はバインドします。

Oracle DatabaseはSQL文の意味を判別していますが、文を実行するために十分な情報は得ていません。Oracle Databaseでは、文に指定されているすべてのバインド変数プレースホルダの値が必要です。例では、:department_idの値が必要です。これらの値を獲得するプロセスは変数のバインドと呼ばれます。

値の場所(メモリー・アドレス)をプログラムで指定する必要があります。アプリケーションのエンド・ユーザーは、Oracle Databaseユーティリティから値の入力を求められることがあるため、バインド変数プレースホルダの値を指定していると認識していない場合もあります。

値の場所(つまり参照によるバインド)がプログラムによって指定されるため、値が変更された場合でも、文の再実行の前に変数を再バインドする必要はありません。Oracle Databaseは文の実行のたびにそのアドレスから変数の値を取得します。

Oracle Databaseでデータ型変換を実行する必要がある場合は、デフォルトでまたは暗黙的に指定する場合を除き、各値のデータ型と長さも指定する必要があります。

7. (オプション)文をパラレル化します。

Oracle Databaseでは、問合せと一部のデータ定義言語(DDL)操作(索引作成、副問合せによる表作成、パーティションに対する操作など)をパラレル化することができます。パラレル化では、処理が早く完了するように、複数のサーバー・プロセスでSQL文の処理を実行します。

8. 文を実行します。

Oracle Databaseは文を実行します。文が問合せまたはINSERT文の場合、データは変更されないためデータベースは行をロックする必要はありません。ただし、文がUPDATE文またはDELETE文の場合は、それにより影響を受けるすべての行を、そのトランザクションの次のCOMMIT、ROLLBACKまたはSAVEPOINTまで、データベースがロックします。これによりデータの整合性が確保されます。

一部の文では実行が複数回行われるように指定できます。これは配列処理と呼ばれます。実行回数をnと指定すると、バインドおよび定義の場所が、サイズnの配列の先頭であるとみなされます。

9. 文が問合せの場合、行をフェッチします。

Oracle Databaseは行を選択し、問合せにORDER BY句が含まれる場合は、行の順序付けを行います。後続の各フェッチにより結果セットの行が1行ずつ取得され、最後の行がフェッチされるまで続きます。

10. カーソルをクローズします。

Oracle Databaseはカーソルをクローズします。

ノート:



トランザクション管理、セッション管理またはシステム管理 SQL 文を再実行するには、別の EXECUTE 文を使用します。

関連項目:

- 索引の詳細は、[Oracle Database概要](#)を参照
- [共有SQL領域](#)
- DEFINEステージの詳細は、『[Oracle Database概要](#)』を参照してください。
- [Oracle Call Interfaceプログラマーズ・ガイド](#)
- 『[Pro*C/C++プログラマーズ・ガイド](#)』

8.1.2 共有SQL領域

Oracle Databaseは、アプリケーションが類似したSQL文をデータベースに送信すると自動的に検出します。その文の最初の処理時に使用されたSQL領域が共有されます。つまり、後で出現する同じ文の処理に使用されます。このため、一意の文に対して存在する共有SQL領域は1つのみです。共有SQL領域は共有メモリー領域であるため、すべてのOracle Databaseプロセスが共有SQL領域を使用できます。SQL領域の共有により、データベース・サーバーでのメモリー使用量が減るため、システム・スループットが向上します。

Oracle Databaseでは、文が類似しているか同一かを決定する際に、ユーザーやアプリケーションによって直接発行されたSQL文と、DDL文によって内部的に発行された再帰的SQL文の両方が比較されます。

関連項目:

- 共有SQL領域の詳細は、『Oracle Database概要』を参照してください
- 共有SQL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

8.2 操作のトランザクションへのグループ化

トピック:

- [トランザクションへの操作のグループ化方法の決定](#)
- [トランザクションのパフォーマンスの改善](#)
- [コミットREDO動作の管理](#)
- [リカバリ可能な停止後のトランザクション結果の確認](#)

関連項目:

トランザクションの基本情報は、『[Oracle Database概要](#)』を参照してください。

8.2.1 トランザクションへの操作のグループ化方法の決定

一般に、トランザクションへの操作のグループ化方法の決定は、Oracle Databaseのプログラム・インタフェースを使用するアプリケーション開発者が考慮します。トランザクションのグループ化方法を決定する際は、次の規則に従います。

- 論理単位ごとに作業が完成し、データの一貫性が保たれるように、トランザクションを定義します。
- すべての参照表の中のデータは、トランザクションが開始する前および終了した後で、一貫した状態であることを確認します。
- 各トランザクションは、データに対する一貫した変更を1つ含むSQL文またはPL/SQLブロックのみで構成されていることを確認します。

たとえば、口座間で預金を移動できるWebアプリケーションを作成すると仮定します。このトランザクションには、一方の口座の借方への記帳(1つのSQL文で実行)ともう一方の口座の貸方への記帳(もう1つのSQL文で実行)が含まれる必要があります。2つの文で1つの作業単位となり、両方が成立するか両方が不成立かのいずれかである必要があります。つまり、片方の文はもう片方のコミットなしでコミットすることはできません。一方の口座への預金など、関連のないアクションは、同じトランザクションに含めないでください。

8.2.2 トランザクションのパフォーマンスの改善

アプリケーション開発者は、パフォーマンスの改善を試みる必要があります。アプリケーションの設計および作成では、次のパフォーマンス改善のテクニックを使用することを検討してください。

- 各トランザクションに対して、次を行います。
 1. 単一のSQL文を使用できる場合、それを使用します。
 2. 単一のSQL文を使用できないが、PL/SQLを使用できる場合、できるだけ小さいPL/SQLを使用します。
 3. (ディレクトリの読み取りなど、実行する必要のあることができないため) PL/SQLを使用できない場合、Javaを

使用します。

4. Javaを使用できない場合(非常に低速な場合など)、または既存の第三世代言語(3GL)コードがある場合、外部Cサブプログラムを使用します。

- 共有SQL領域を利用できるように、SQL文の発行基準を確立します。

Oracle Databaseが同一のSQL文を認識し、SQL文がメモリー領域を共有できるようにします。これによって、データベース・サーバー上のメモリー使用量が減少し、システム・スループットが向上します。

- 統計を収集します。Oracle Databaseは統計を使用して、SQL文を最適化するコストベースの方法を実装することができます。必要に応じて、オプティマイザへの追加のヒントを使用できます。

ほとんどの統計の収集にはDBMS_STATSパッケージを使用します。このパッケージでは、パラレルでの統計の収集、パーティション化されたオブジェクトのグローバル統計の収集、その他の方法での統計収集のチューニングを行うことができます。

コストベースのオプティマイザに関係しない統計収集(空きリスト・ブロックに関する情報収集など)では、SQL文ANALYZEを使用します。

- トランザクションを開始する前にDBMS_APPLICATION_INFO.SET_ACTIONプロシージャを起動し、トランザクションの名前をデータベースに登録し、後でOracle TraceおよびSQLトレース機能を使用してパフォーマンスをトラッキングするときに使用できるようにします。
- ユーザーが作成したPL/SQLファンクションをSQL式に組み込んで、ユーザーの生産性および問合せ効率を向上させます。詳細は、次を参照してください。

関連項目:

- トランザクション管理の詳細は、[『Oracle Database概要』](#)を参照してください。
- [アプリケーション開発者用のPL/SQL](#)
- [複数のプログラミング言語を使用したアプリケーションの開発](#)
- ヒントおよびANALYZE文の詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。
- DBMS_STATSパッケージおよびDBMS_APPLICATION_INFOパッケージの詳細は、[『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』](#)を参照してください。
- [SQL文からのストアドPL/SQLファンクションの起動](#)

8.2.3 コミットREDO動作の管理

トランザクションがOracle Databaseを更新するとき、これに対応するREDOエントリが生成されます。このREDOエントリは、トランザクションが完了するまで、Oracle DatabaseによりREDOログにバッファリングされます。トランザクションがコミットされると、それに応じてログ・ライター・プロセス(LGWR)により、トランザクション内のすべての変更のバッファリングされたREDOエントリに対し、REDOレコードがディスクに書き込まれます。デフォルトでは、REDOエントリは、コールがクライアントに戻される前に、Oracle Databaseによりディスクに書き込まれます。この動作のためにアプリケーションはREDOエントリがディスクに永続的に書き込まれるのを待機する必要があり、コミットに待機時間が発生します。

Oracle Databaseでは、アプリケーションのニーズに応じてコミットREDOの処理を変更できます。アプリケーションで非常に高いトランザクション・スループットが必要とされるため、コミットでの待機時間を短くするためにコミットの永続性を放棄してもよい場合

は、デフォルトのCOMMITオプションを変更して、データベースでデータがオンラインREDOログへ書き込まれるのをアプリケーションが待機しなくて済むようにできます。

[表8-1](#) はCOMMITオプションを示します。

注意:



NOWAIT オプションを指定した場合、コミット・メッセージの受信後、REDO ログ・レコードへの書き込み前に障害が発生すると、トランザクションにその変更が永続的であると誤って示す可能性があります。

表8-1 COMMIT文のオプション

オプション	効果
WAIT (デフォルト)	対応する REDO 情報がオンライン REDO ログに永続的に書き込まれるまで、COMMIT 文が戻されないことを確認します。この COMMIT 文から成功として戻されたコミットをクライアントが受信した場合、そのトランザクションは永続的なメディアにコミットされています。 ログに正常に書き込まれた後に障害が発生すると、成功メッセージがクライアントに戻されない場合があります。その場合、クライアントはトランザクションがコミットされたかどうかを判断できません。
NOWAIT (WAIT の代替)	COMMIT 文は、REDO ログへの書き込みが完了したかどうかにかかわらず、クライアントに戻されます。この動作はトランザクションのスループットを向上させます。
BATCH (IMMEDIATE の代替)	REDO 情報は、同時に実行しているトランザクションとともに REDO ログにバッファリングされます。十分な REDO 情報が収集されると、REDO ログのディスク書き込みが開始されます。この動作では、複数のトランザクションの REDO 情報が 1 回の I/O 操作でログに書き込まれるため、グループ・コミットと呼ばれます。
IMMEDIATE (デフォルト)	LGWR がトランザクションの REDO 情報をログに書き込みます。この操作オプションではディスク I/O が発生するため、トランザクションのスループットが低下する場合があります。

COMMITのオプションを変更するには、COMMIT文、または適切な初期化パラメータを使用します。

ノート:



分散トランザクションの場合、デフォルトの IMMEDIATE および WAIT アクションは変更できません。

Oracle Call Interface(OCI)を使用するアプリケーションの場合、アプリケーション内でOCITransCommitファンクションに次のフラグを設定することにより、REDO動作を変更できます。

- OCI_TRANS_WRITEWAIT
- OCI_TRANS_WRITENOWAIT
- OCI_TRANS_WRITEBATCH
- OCI_TRANS_WRITEIMMED

注意:



OCI_TRANS_WRITE_NOWAIT の使用時には、トランザクションが強制終了、強制起動、インスタンス障害またはノード障害で警告なしに消失する可能性があります。Oracle RAC システムでは非同期にコミットした変更は、他のインスタンスですぐに読み込めない場合があります。

NOWAITおよびBATCHオプションの指定では、問題が発生しやすい期間が短くなります(このような期間には、アプリケーションがコミット済として認識するトランザクションをOracle Databaseがロールバックできます)。アプリケーションは、次のような状況に対応できる必要があります。

- データベース・ホストの障害。バッファには格納されていても、まだオンラインREDOログに書き込まれていないREDOエントリが失われる可能性があります。
- ファイルI/Oの問題で、LGWRによるバッファ済REDOエントリのディスクへの書込みが妨げられる場合。REDOログが多重化されていない場合、コミットが失われます。

関連項目:

- OCITransCommit関数の詳細は『[Oracle Call Interfaceプログラマーズ・ガイド](#)』を参照してください。
- [Oracle Database SQL言語リファレンス](#)
- 初期化パラメータについては、『[Oracle Databaseリファレンス](#)』を参照してください。

8.2.4 リカバリ可能な停止後のトランザクション結果の確認

リカバリ可能な停止は、アプリケーション(クライアント)とOracle Database (サーバー)間の接続を切断する、システム、ハードウェア、通信または記憶域の障害です。停止後、アプリケーションには切断に関するエラー・メッセージが表示されます。接続が切断されたときに実行されていたトランザクションは、進行中のトランザクションで、完了までコミットまたは実行された、またはされていない可能性があります。

停止からリカバリするには、アプリケーションが進行中のトランザクションの結果(コミットされたかどうか、およびセッションの状態が意図どおりに変更されたかどうか)を確認する必要があります。トランザクションがコミットされていなかった場合、アプリケーションはトランザクションを再発行するか、エンドユーザーに未コミット・ステータスを戻すことができます。トランザクションがコミットされていた場合、アプリケーションは切断エラーではなくコミット済ステータスをエンドユーザーに戻すことができます。トランザクションがコミットされていると同時に完了している場合、アプリケーションは新規セッションを使用してセッションの状態を再確立することによって処理を続行できる可能性があります。

進行中のトランザクションの結果をアプリケーションに提供し、トランザクションが重複していないことを確認するために使用できるOracle Database機能は、トランザクション・ガードで、そのApplication Program Interface (API)はPL/SQLプロシージャDBMS_APP_CONT.GET_LTXID_OUTCOMEです。

トピック:

- [トランザクション・ガードの理解](#)
- [DBMS_APP_CONT.GET_LTXID_OUTCOMEの理解](#)
- [トランザクション・ガードの使用](#)

8.2.4.1 トランザクション・ガードの理解

トランザクション・ガードは、停止後に進行中のトランザクションの結果をアプリケーションに提供するために使用できるOracle Databaseツールです。アプリケーションはトランザクション・ガードを使用して、停止後に既知の結果を(コミット済または未コミット)をエンドユーザーに提供できるとともに、必要に応じて、トランザクションがコミットされておらず状態が正しい場合はトランザクションをリプレイすることもできます。

トランザクション・ガードは、API、PL/SQLプロシージャDBMS_APP_CONT.GET_LTXID_OUTCOMEを介してトランザクション結果を提供します。

トランザクション・ガードは、失敗したセッション上の最後の進行中のトランザクションを識別するグローバル一意識別子である論理トランザクション識別子(LTXID)に依存します。データベースは、トランザクションがコミットされるとLTXIDを記録し、コミット・メッセージとともに新規LTXIDをクライアントに(クライアントのラウンド・トリップごとに)戻します。クライアント・ドライバは、次のCOMMIT時に使用するLTXIDを常に保持します。

ノート:



- 通信エラーを実際の結果に置き換えるために、リカバリ可能なエラーが原因で失敗したセッションの結果を検索する場合のみ、トランザクション・ガードを使用してください。
- トランザクション・ガードは独自のセッションで使用しないでください。
- トランザクション・ガードは稼働中のセッションで使用しないでください。

稼働中のトランザクションを停止するには、ローカル・インスタンスまたはリモート・インスタンスで ALTER SYSTEM KILL SESSION IMMEDIATE を使用してください。

関連項目:

[DBMS_APP_CONT.GET_LTXID_OUTCOMEの理解](#)

8.2.4.1.1 トランザクション・ガードがLTXIDを使用する方法

トランザクション・ガードは、LTXIDを次のように使用します。

- トランザクションの実行中、Oracle Database (サーバー)とアプリケーション(クライアント)の両方が次のCOMMIT時に使用するLTXIDを保持します。
- トランザクションがコミットされると、Oracle DatabaseはトランザクションとともにLTXIDを記録します。LTXIDがすでにコミットされているかブロックされている場合、データベースはエラーを発生させ、トランザクションの重複を阻止します。
- LTXIDは、RETENTION_TIMEOUTパラメータによって指定されている時間にわたってOracle Database内に存在し続けます。デフォルトは24時間です。この値を変更するには:
 - Real Application Clustersを実行している場合は、サーバー制御ユーティリティ(SRVCTL)を使用します。
 - Real Application Clustersを使用していない場合は、DBMS_SERVICEパッケージを使用します。

トランザクションがリモートであるか分散されている場合、LTXIDはローカル・データベース内に永続化されます。

LTXIDは標準REDO適用時にData GuardおよびActive Data Guardに転送されます。

- リカバリ可能なエラーの後:

- トランザクションがコミットされていない場合、Oracle DatabaseはLTXIDをブロックすることにより、同じLTXIDを持つ以前の進行中のトランザクションをコミットできないようにします。

この動作により、アプリケーションは未コミット結果をユーザーに戻すことができ、ユーザーは実行内容を決定できるようになります。また、アプリケーションは必要に応じて、アプリケーションを安全にリプレイすることもできます。

- トランザクションがコミットされている場合、アプリケーションはこの結果をエンドユーザーに戻すことができ、状態が正しい場合、アプリケーションは継続できます。
- トランザクションがロールバックされる場合、Oracle DatabaseはLTXIDを再利用します。

関連項目:

- トランザクション・ガードがコミットをサポートしているソースのリストについては、[「トランザクション・ガードのカバレッジ」](#)を参照してください
- トランザクション・ガードがコミットをサポートしていないソースのリストについては、[「トランザクション・ガードの除外」](#)を参照してください
- SRVCTL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。
- DBMS_SERVICEパッケージの詳細は、[『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』](#)を参照してください。

8.2.4.2 DBMS_APP_CONT.GET_LTXID_OUTCOMEの理解

PL/SQLプロシージャDBMS_APP_CONT.GET_LTXID_OUTCOMEは、トランザクション・ガードのAPIです。停止後、アプリケーションはOracle Databaseに再接続してから、このプロシージャを呼び出して進行中のトランザクションの結果を確認できます。

DBMS_APP_CONT.GET_LTXID_OUTCOMEには、次のパラメータがあります。

パラメータ名	データ型	パラメータ・	
		モード	値
CLIENT_LTXID	RAW	IN	進行中のトランザクションの LTXID
COMMITTED	BOOLEAN	OUT	進行中のトランザクションがコミットされた場合は TRUE、そうでない場合は FALSE
USER_CALL_COMPLETED	BOOLEAN	OUT	進行中のトランザクションが完了した場合は TRUE、そうでない場合は FALSE

トピック:

- [CLIENT_LTXIDパラメータ](#)
- [COMMITTEDパラメータ](#)
- [USER_CALL_COMPLETEDパラメータ](#)

- [例外](#)

関連項目:

DBMS_APP_CONT.GET_LTXID_OUTCOME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

8.2.4.2.1 CLIENT_LTXIDパラメータ

アプリケーション(クライアント)が進行中のトランザクションのLTXIDを使用してDBMS_APP_CONT.GET_LTXID_OUTCOMEを呼び出すには、クライアント・ドライバを使用してクライアント・ドライバで使用中の最後のLTXIDを取得する必要があります。クライアント・ドライバは、次回コミットするトランザクションのLTXIDを保持します。この場合、このLTXIDは停止時に進行中のトランザクション用です。JDBC-ThinドライバにはgetLTXID、ODP.NETにはLogicalTransactionId、OCIおよびOCCIにはLTXIDとともにOCI_ATTR_GETを使用します。

JDBC-Thinドライバには、クライアント・ドライバのLTXIDが変更されるたびにトリガーされるコールバックも用意されています。このコールバックを使用して、使用対象の現在のLTXIDを保持できます。このコールバックは、実行の繰返しをブロックする必要があるアプリケーション・サーバーおよびアプリケーションに対して特に役立ちます。

ノート:



アプリケーションは、LTXIDをDBMS_APP_CONT.GET_LTXID_OUTCOMEに渡す直前にこれを取得する必要があります。LTXIDを事前に取得すると、以前のLTXIDがDBMS_APP_CONT.GET_LTXID_OUTCOMEに渡される可能性があり、この場合、リクエストが拒否されます。

関連項目:

- [『Oracle Database JDBC開発者ガイド』](#)
- [Oracle Call Interfaceプログラマーズ・ガイド](#)
- [Oracle Data Provider for .NETの開発者ガイドfor Microsoft Windows](#)

8.2.4.2.2 COMMITTEDパラメータ

DBMS_APP_CONT.GET_LTXID_OUTCOMEからアプリケーションに制御が戻された後、アプリケーションは、仮パラメータCOMMITTEDに対応する実際のパラメータの値を確認することにより、進行中のトランザクションがコミットされたかどうかを確認できます。

実際のパラメータの値がTRUEである場合、トランザクションはコミットされています。

実際のパラメータの値がFALSEである場合、トランザクションはコミットされていません。このため、アプリケーションがコードUNCOMMITTEDをエンドユーザーに戻したり、これを使用してトランザクションをリプレイしても安全です。

アプリケーションがUNCOMMITTEDに戻した後に以前のセッションがトランザクションをコミットしないようにするために、DBMS_APP_CONT.GET_LTXID_OUTCOMEによってLTXIDがブロックされます。LTXIDをブロックすることにより、エンドユーザーが未コミット状態に基づいて決定を下したり、アプリケーションがトランザクションをリプレイできるようになり、トランザクションの重複が阻止されます。

8.2.4.2.3 USER_CALL_COMPLETEDパラメータ

一部のトランザクションは完了時に情報を戻します。たとえば、成功時にコミット(自動コミット)を使用するトランザクションは、影響を受ける行の数、またはSELECT文の場合は行自体を戻すことができます。OUTパラメータを持つPL/SQLサブプログラムを呼び出すトランザクションは、これらのパラメータの値を戻します。PL/SQLファンクションを呼び出すトランザクションは、ファンクション値を戻します。また、PL/SQLサブプログラムを呼び出すトランザクションは、COMMIT文を実行してからこれ以外の処理を行うことができます。

進行中のトランザクションが完了時に戻す情報、またはデータベースの変更のコミット後にトランザクションが行うセッション状態の変更をアプリケーションが必要とする場合、アプリケーションは、進行中のトランザクションが完了したかどうかを確認する必要があります。これを行うには、仮パラメータUSER_CALL_COMPLETEDに対応する実際のパラメータの値を確認します。

実際のパラメータの値がTRUEである場合、トランザクションは完了しており、アプリケーションには続行する必要がある情報および処理が存在します。

実際のパラメータの値がFALSEである場合、クライアントからのコールは完了していない可能性があります。このため、アプリケーションには続行する必要がある情報および処理が存在しない可能性があります。

8.2.4.2.4 例外

アプリケーション(クライアント)とOracle Database (サーバー)が同期していない場合、DBMS_APP_CONT.GET_LTXID_OUTCOMEプロシージャにより、次のいずれかの例外が発生します。

例外	説明
ORA-14950 - SERVER_AHEAD	サーバーがクライアントより進んでいます。つまり、アプリケーションがDBMS_APP_CONT.GET_LTXID_OUTCOME に渡した LTXID により、進行中のトランザクションより古いトランザクションが識別されます。 アプリケーションは、LTXID を DBMS_APP_CONT.GET_LTXID_OUTCOME に渡す直前にこれを取得する必要があります。
ORA-14951 - CLIENT_AHEAD	クライアントがサーバーより進んでいます。サーバーが以前の状態にフラッシュ・バックされたか、メディア・リカバリを使用してリカバリされたか、以前にオープンされたスタンバイ・データベースであり、データが失われています。
ORA-14906 - SAME_SESSION	LTXID を所有するセッションでの GET_LTXID_OUTCOME の実行はサポートされていません。これは、この実行によってこのセッションに対する追加処理がブロックされるためです。
ORA-14909 - COMMIT_BLOCKED	同じユーザー名を持つ別のユーザーによって GET_LTXID_OUTCOME を使用してセッションのコミットがブロックされています。GET_LTXID_OUTCOME はデッド・セッションに対してのみ呼び出す必要があります。ライブ・セッションのブロックは、ALTER SYSTEM KILL SESSION IMMEDIATE を使用した方が確実に実行できます。不明な点は、アプリケーション管理者に問い合わせてください。
ORA-14952 GENERAL ERROR	DBMS_APP_CONT.GET_LTXID_OUTCOME では、進行中のトランザクションの結果を確認できません。トランザクションの処理中にエラーが発生し、エラー・スタックにエラー詳細が表示されます。

関連項目:

- [Oracle Flashback Technologyの使用](#)
- メディア・リカバリおよびスタンバイ・データベースの管理の詳細は、『[Oracle Data Guard概要および管理](#)』を参照してください。

8.2.4.3 トランザクション・ガードの使用

アプリケーション(クライアント)は、エラー・メッセージを受け取った後、次のステップを実行してトランザクション・ガードを使用する必要があります。

1. エラーの原因が停止(リカバリ可能)であるかどうかを確認します。

手順については、クライアント・ドライバの関連ドキュメントを参照してください(OCI、OCCIおよびODP.NETはOCI_ATTRIBUTE、JDBCはisRecoverable)。

2. エラーがリカバリ可能である場合、クライアント・ドライバのAPIを使用して、進行中のトランザクションの論理トランザクション識別子(LTXID)を取得します。

手順については、クライアント・ドライバの関連ドキュメントを参照してください。

3. データベースに再接続します。

アプリケーションが取得するセッションには、新規セッションまたはプール・セッションがあります。

4. ステップ2のLTXIDを使用してDBMS_APP_CONT.GET_LTXID_OUTCOMEを呼び出します。

5. 仮パラメータCOMMITTEDに対応する実際のパラメータの値を確認します。

値がTRUEである場合、進行中のトランザクションがコミットされたことをアプリケーションに伝えます。アプリケーションはこの結果をエンドユーザーに戻すことができ、状態が正しい場合、アプリケーションは継続できます。

値がFALSEである場合、アプリケーションはUNCOMMITTEDまたは同様のメッセージをユーザーに戻すことができ、ユーザーは次のステップを選択できるようになります。オプションとして、アプリケーションはユーザーに対してトランザクションをリプレイできます。たとえば:

- a. 必要な場合、クライアント側で状態の変更をクリーン・アップします。
- b. 進行中のトランザクションを再発行します。

進行中のトランザクションを再発行しないときに、進行中のトランザクションが完了時に戻す情報や、データベースの変更のコミット後にトランザクションが実行する処理をアプリケーションが必要としない場合は、続行します。それ以外の場合、仮パラメータUSER_CALL_COMPLETEDに対応する実際のパラメータの値を確認します。

値がTRUEである場合、続行します。

値がFALSEである場合、アプリケーションを続行できないことをアプリケーション・ユーザーに伝えます。

8.3 読取り専用トランザクションでの反復可能読取りの保証

デフォルトでは、Oracle Databaseでは、文レベルの読取り一貫性は保証されますが、トランザクション・レベルの読取り一貫性は保証されません。文レベルの読取り一貫性を持つ場合、ある文に含まれる問合せは、この文の存続中、一貫性のあるデータを生成します。その他の文による変更は反映されません。トランザクション・レベルの読取り一貫性(反復可能読取り)を持つ場合、あるトランザクションに含まれる問合せは、このトランザクションの存続中、一貫性のあるデータを生成します。その他のト

ランザクションによる変更は反映されません。

DML文を含まないトランザクションに対するトランザクション・レベルの読取り一貫性を保証するには、このトランザクションを読取り専用に指定します。読取り専用トランザクションに含まれる問合せには、このトランザクションが開始される前にコミットされた編集以外見えないため、トランザクションの存続中、問合せの結果の一貫性が保たれます。

読取り専用トランザクションは、データ・ロックを追加取得することなしに、トランザクション・レベルの読取り一貫性を提供します。したがって、読取り専用トランザクションがデータを問い合わせている間、別のトランザクションが同じデータを問い合わせ、更新することができます。

ある読取り専用トランザクションは次の文で始まります。

```
SET TRANSACTION READ ONLY [ NAME string ];
```

SET TRANSACTION READ ONLY文の前に来ることができるのは、DDL文のみです。SET TRANSACTION READ ONLY文の実行が正常に終了した後で、このトランザクションはSELECT(ただし、FOR UPDATEなしで)、COMMIT、ROLLBACK、または非DML文(SET ROLE、ALTER SYSTEM、LOCK TABLEなど)のみを含むことができます。COMMIT、ROLLBACK、またはDDL文は読取り専用トランザクションで終わります。

関連項目:

SET TRANSACTION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。

実行時間の長い問合せでは、読取り一貫性(CR)操作に必要なUNDO情報が無効になったためにエラーが発生することがあります。これは、アクティブなトランザクションによって、コミット済UNDOブロックが上書きされた場合に起こります。

自動UNDO管理を行うと、データベース管理者(DBA)は、パラメータUNDO_RETENTIONを使用してUNDO情報のデータベースでの保存期間を明示的に制御できます。たとえば、UNDO_RETENTIONが30分である場合、すべてのコミット済UNDO情報は、最低30分間データベースに保持されます。この設定により、実行時間が30分以内の問合せでOERエラー「スナップショットが古すぎます。」が発生することはなくなります。

関連項目:

- 読取り一貫性の詳細は、[『Oracle Database概要』](#)を参照してください。
- 実行時間の長い問合せおよび再開可能領域割当ての詳細は、[『Oracle Database管理者ガイド』](#)を参照してください。

8.4 明示的な表のロック

Oracle Databaseには、データの同時実行性、データの整合性、および文レベルでの読取り一貫性を保持するための、デフォルト・ロック・メカニズムが用意されています。ただし、これらのメカニズムは表を明示的にロックすることによりオーバーライドできます。表の明示的なロックは次のような状況で役立ちます。

- アプリケーション内のあるトランザクションが他のトランザクションの完了を待機する必要がなくなるように、リソースに対して排他的にアクセスするトランザクションが必要な場合。
- 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(反復可能読取り)が必要な場合。

次のどのSQL文を使用しても、デフォルト・ロックをトランザクション・レベルでオーバーライドできます。

- LOCK TABLE

- FOR UPDATE句を使用したSELECT
- READ ONLYまたはISOLATION LEVEL SERIALIZABLEオプションを使用したSET TRANSACTION

これらの文によって取得されるロックは、トランザクションのコミット後またはロールバック後に解除されます。

初期化パラメータDML_LOCKSは、DMLロックの最大数を決定します。通常、デフォルト値には十分な値が設定されていますが、明示的なロックを使用する際に、この値を大きくする必要がある場合があります。

ノート:



いずれかのレベルで Oracle Database のデフォルト・ロックをオーバーライドする場合は、データ整合性が保証されていること、データ同時実行性が許容されていること、およびデッドロックの可能性がないか、デッドロックが適切に処理されていることを確認してください。

関連項目:

- [Oracle Database概要](#)
- [読取り専用トランザクションでの反復可能読取りの保証](#)
- [シリアライズ可能トランザクションを使用した同時実行性の制御](#)
- [Oracle Database SQL言語リファレンス](#)

トピック:

- [表のロックを取得するために必要な権限](#)
- [ロック方法の選択](#)
- [Oracle Databaseによる表ロック制御](#)
- [明示的な行ロック](#)
- [明示的なロックにおける同時実行性の例](#)

8.4.1 表のロックを取得するために必要な権限

自らのスキーマの表に対して表のロックを取得するためには、特別な権限は必要ありません。他スキーマ内の表に対して表ロックを取得するには、LOCK ANY TABLEシステム権限またはその表に対する(SELECTやUPDATEなどの)オブジェクト権限が必要です。

8.4.2 ロック方法の選択

LOCK TABLE文が実行されると、トランザクションは指定された表ロックを明示的に取得します。LOCK TABLE文は、デフォルト・ロックを明示的にオーバーライドします。ビューに対してLOCK TABLE文が発行されると、基礎となる実表がロックされます。次の文は、その文が含まれるトランザクションのために、employees表およびdepartments表に対する排他表ロックを取得します。

```
LOCK TABLE employees, departments IN EXCLUSIVE MODE NOWAIT;
```

ロック・モードが同じ場合は、ロックする表またはビューを複数指定できます。ただし、1つのLOCK TABLE文に指定できるロック・

モードは1つのみです。



ノート:

表がロックされると、その表のすべての行がロックされます。他のユーザーは、その表を変更できません。

LOCK TABLE文では、表ロックの待ち時間も指示できます。

- ロックを待機しない場合は、NOWAITまたはWAIT 0を指定します。
表ロックがすぐに使用可能である場合にのみ表ロックを取得します。すぐに使用可能でない場合は、その時点ではロックを使用できないことを示すエラーが戻されます。
- 表ロックを取得するためにn秒まで待機する場合は、WAIT nを指定します。ここで、nは0より大きく、100000以下です。
n秒経過しても表ロックを使用できない場合は、その時点ではロックを使用できないことを示すエラーが戻されます。
- ロックを取得するまで無限に待機する場合は、NOWAITおよびWAITのいずれも指定しないでください。
データベースは表が使用可能になるまで無限に待機し、表をロックして制御を戻します。データベースでDML文と同時にDDL文を実行すると、タイムアウトやデッドロックが発生する場合があります。このようなタイムアウトまたはデッドロックが検出されると、エラーが戻されます。

関連項目:

- [明示的な行ロック](#)
- [Oracle Database SQL言語リファレンス](#)

トピック:

- [ROW SHARE MODEおよびROW EXCLUSIVE MODEでロックする場合](#)
- [SHARE MODEでロックする場合](#)
- [SHARE ROW EXCLUSIVE MODEでロックする場合](#)
- [EXCLUSIVE MODEでロックする場合](#)

8.4.2.1 ROW SHARE MODEおよびROW EXCLUSIVE MODEでロックする場合

共有(ROW SHARE MODE)表ロック、および排他(ROW EXCLUSIVE MODE)表ロックは最も高い同時実効性を提供します。次のような場合に使用します。

- トランザクション内で表を更新する前に、別のトランザクションが共有表ロック、行共有表ロックまたは排他表ロックを割り込んで取得しないようにする必要がある場合。
別のトランザクションが共有表ロック、行共有表ロックまたは排他表ロックを割り込んで取得した場合、他のどのトランザクションも、そのロック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、表を更新できません。
- トランザクションで表を変更できるようになるまで、表の変更または削除を防止する必要がある場合。

8.4.2.2 SHARE MODEでロックする場合

SHARE MODE表ロックは非常に制限の多いデータ・ロックです。次のような場合に使用します。

- トランザクションが表を問い合わせるのみで、そのトランザクションの存続中一貫した一連の表データを必要とする場合。
- 表に対してSHARE MODEロックを保持するすべてのトランザクションがコミットまたはロールバックするまで、ロックされている表を更新しようとする他のトランザクションを阻止できる場合。
- 他のトランザクションが、ロックされている表に対して同時にSHARE MODE表ロックを取得でき、またトランザクション・レベルの読取り一貫性のオプションを使用できる場合。

注意:

同じトランザクション内で、後から表を更新しないこともあります。ただし、複数のトランザクションが同じ表に対して共有表ロックを同時に保持している場合は(SELECT FOR UPDATE 文の結果によって行ロックが保持されている場合でも)、どのトランザクションも表を更新できません。したがって、同じ表に対する同時共有表ロックがよく発生する場合は、更新処理を継続できず、デッドロックがよく発生することになります。このような場合には、かわりに共有行排他ロックまたは排他表ロックを使用してください。

シナリオ: 表employeesおよびbudget_tabは、第3の表departmentsの一貫した一連のデータを必要とします。特定の部門番号に関して、employeesおよびbudget_tabの情報を更新し、この2つのトランザクションの間にメンバーが部門に追加されないように保証するものとします。

解決方法: [例8-1](#)のようにdepartments表をSHARE MODEでロックします。departments表の更新はまれなため、ロックしても他の多くのトランザクションの待機時間が長くなることはありません。

例8-1 SHARE MODEでのLOCK TABLE

```
-- Create and populate table:

DROP TABLE budget_tab;
CREATE TABLE budget_tab (
  sal      NUMBER(8,2),
  deptno   NUMBER(4)
);

INSERT INTO budget_tab (sal, deptno)
  SELECT salary, department_id
  FROM employees;

-- Lock departments and update employees and budget_tab:

LOCK TABLE departments IN SHARE MODE;

UPDATE employees
  SET salary = salary * 1.1
  WHERE department_id IN
    (SELECT department_id FROM departments WHERE location_id = 1700);

UPDATE budget_tab
  SET sal = sal * 1.1
  WHERE deptno IN
    (SELECT department_id FROM departments WHERE location_id = 1700);

COMMIT; -- COMMIT releases lock
```

8.4.2.3 SHARE ROW EXCLUSIVE MODEでロックする場合

共有(SHARE ROW EXCLUSIVE MODE)表ロックは、次のような場合に使用します。

- トランザクションで、特定の表についてトランザクション・レベルの読取り一貫性があり、かつ、ロックされている表が更新可能であることが必要な場合。
- 他のトランザクションによる(SELECT FOR UPDATEを使用した)明示的な行ロックの可能性を考慮していない場合。ロック中のトランザクション内のUPDATEおよびINSERT文が待機させられ、デッドロックが発生する可能性があります。
- このように動作するトランザクションが1つのみ必要な場合。

8.4.2.4 EXCLUSIVE MODEでロックする場合

排他(EXCLUSIVE MODE)表ロックは、次のような場合に使用します。

- トランザクションが、ロックされている表にすぐに更新アクセスをする必要がある場合。トランザクションが排他表ロックを保持していると、他のトランザクションはロックされた表の中の特定の行をロックできません。
- 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、ロックされた表に対してトランザクション・レベルの読取り一貫性が保持される場合。
- 低レベルのデータ同時実行性を意識する必要がなく、排他表ロックを要求するトランザクションを順次待機させて表を順番に更新させる場合。

8.4.3 Oracle Databaseによる表ロック制御

Oracle Databaseに表ロック制御を任せると、アプリケーションに必要なプログラム・ロジックが少なくて済みます。ただし、表ロックを自分で管理する場合より制御範囲が小さくなります。

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE文またはALTER SESSION ISOLATION LEVEL SERIALIZABLE文を発行すると、基礎となるロック・プロトコルを変更しなくても、ANSIのシリアライズ可能性を維持できます。この手法によって、ANSIのシリアライズ可能性を維持しながら表へ同時アクセスできます。表をロックすることによって、同時実行性が大幅に減少します。

関連項目:

- SET TRANSACTION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。
- ALTER SESSION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。

これらのパラメータの設定変更は、インスタンスの停止時にのみ行ってください。複数インスタンスが単一データベースにアクセスする場合は、すべてのインスタンスでこれらのパラメータの設定を同じにする必要があります。

8.4.4 明示的な行ロック

FOR UPDATE句を含むSELECT文を使用すると、デフォルト・ロックをオーバーライドできます。この文は、選択されている行がその後の文で更新されることを想定し、(UPDATE文のように)選択された行に対する明示的な行ロックを取得します。

SELECT FOR UPDATE文を使用すると、その行を変更せずに行をロックできます。たとえば、『Oracle Database PL/SQL 言語リファレンス』では、いくつかのトリガーで参照整合性を実装する方法を示しています。EMP_DEPT_CHECKトリガーでは、参照される親キー値を含む行が、トランザクションの存続中は同じ値のままであることを保証するためにロックされます。親キーが

更新または削除された場合、参照整合性違反になります。

SELECT FOR UPDATE文は、ユーザーが1行以上の特定行のフィールドを変更できるような対話型プログラムでよく使用されます(時間がかかることがあります)。行を更新している対話型プログラム・ユーザーが常に1ユーザーのみであるように、行がロックされます。

カーソル定義にSELECT FOR UPDATE文が使用される場合は、カーソルがオープンされるとき(最初のフェッチの前)に結果セット内の行がロックされます。行は、カーソルからフェッチされるときに、個別にロックされるわけではありません。カーソルをオープンしたトランザクションがコミットまたはロールバックされたときにのみ、ロックが解除されます。カーソルがクローズされるときには、ロックは解除されません。

SELECT FOR UPDATE文の結果セット内の各行は個別にロックされます。SELECT FOR UPDATE文は、競合している行ロックを他のトランザクションが解除するまで待機します。したがって、SELECT FOR UPDATE文が表の行を多数ロックし、表に対して非常に多くの更新アクティビティが発生する場合は、EXCLUSIVE表ロックを取得する方がパフォーマンスが改善する場合があります。

ノート:



問合せの実行中に、SELECT FOR UPDATE に対する結果セットが変化する場合があります。たとえば、問合せ開始後に問合せで選択された列が更新されたり、行が削除された場合です。このような場合、SELECT FOR UPDATE は変更されなかった行でロックを取得し、このロックを使用して表の読取り一貫性スナップショットを取得してから、残りのロックを取得するために問合せを再起動します。

アプリケーションで SELECT FOR UPDATE 文を使用する場合に、競合するロック・リクエストによってユーザーが原因のデッドロックが発生しないことを保証できない場合は(たとえば、表の同時実行 DML 文が SELECT FOR UPDATE 文の問合せの結果セットに影響を与えないことを確認することによって)、このようなデッドロック(ORA-00060)をアプリケーションが常に適切に処理するようにコーディングします。

デフォルトでは、要求された行ロックが取得されるまでSELECT FOR UPDATE文は待機します。この動作を変更するには、SELECT FOR UPDATE文でNOWAIT、WAIT、またはSKIP LOCKED句を使用します。このような句の詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

関連項目:

- [Oracle Database PL/SQL言語リファレンス](#)
- [Oracle Database SQL言語リファレンス](#)

8.4.5 明示的なロックにおける同時実行性の例

[表8-2](#)に、LOCK TABLE文およびFOR UPDATE句を含むSELECT文が使用された場合に、データの同時実行性、整合性および一貫性がOracle Databaseでどのように維持されるかを示します。簡略にするため、ORA-00054のメッセージ・テキスト(「リソース・ビジー、NOWAITが指定されていました。')は記載していません。ユーザーが入力するテキストはboldで表示しています。



ノート:

ハイブリッド列圧縮(HCC)で圧縮された表では、DML 文によって行ではなく圧縮単位がロックされます。

表8-2 明示的なロックにおける同時実行性の例

タイム・ポイント	トランザクション1	トランザクション2
1	LOCK TABLE hr.departments IN ROW SHARE MODE; Statement processed.	
2		DROP TABLE hr.departments; DROP TABLE hr.departments * ORA-00054 (トランザクション1によって表が ロックされているため、排他 DDL ロッ クを取得できません。)
3		LOCK TABLE hr.departments IN EXCLUSIVE MODE NOWAIT; ORA-00054
4		SELECT location_id FROM hr.departments WHERE department_id = 20 FOR UPDATE OF location_id; LOCATION_ID - ----- DALLAS 1 row selected.
5	UPDATE hr.departments SET location_id = 'NEW YORK' WHERE department_id = 20; (トランザクション2によって同じ行 がロックされているため待機します。)	
6		ROLLBACK; (行ロックを解放します。)
7	1 row processed. ROLLBACK;	
8	LOCK TABLE hr.departments IN ROW EXCLUSIVE MODE; Statement processed.	
9		LOCK TABLE hr.departments IN EXCLUSIVE MODE NOWAIT; ORA-00054

タイム・ポイント	トランザクション1	トランザクション2
10		LOCK TABLE hr.departments IN SHARE ROW EXCLUSIVE MODE NOWAIT; ORA-00054
11		LOCK TABLE hr.departments IN SHARE ROW EXCLUSIVE MODE NOWAIT; ORA-00054
12		UPDATE hr.departments SET location_id = 'NEW YORK' WHERE department_id = 20; 1 row processed.
13		ROLLBACK;
14	SELECT location_id FROM hr.departments WHERE department_id = 20 FOR UPDATE OF location_id; LOCATION_ID - ----- DALLAS 1 row selected.	
15		UPDATE hr.departments SET location_id = 'NEW YORK' WHERE department_id = 20; 1 row processed. (トランザクション 1 によって同じ行 がロックされているため待機します。)
16	ROLLBACK;	
17		1 row processed. (競合するロックが解放されました。) ROLLBACK;
18	LOCK TABLE hr.departments IN ROW SHARE MODE Statement processed.	
19		LOCK TABLE hr.departments IN EXCLUSIVE MODE NOWAIT; ORA-00054
20		LOCK TABLE hr.departments IN SHARE ROW EXCLUSIVE MODE NOWAIT; ORA-00054

タイム・ポイント	トランザクション1	トランザクション2
21		LOCK TABLE hr.departments IN SHARE MODE; Statement processed.
22		SELECT location_id FROM hr.departments WHERE department_id = 20; LOCATION_ID ----- DALLAS 1 row selected.
23		SELECT location_id FROM hr.departments WHERE department_id = 20 FOR UPDATE OF location_id; LOCATION_ID - ----- DALLAS 1 row selected.
24		UPDATE hr.departments SET location_id = 'NEW YORK' WHERE department_id = 20; (トランザクション 1 に競合する表 ロックがあるため待機します。)
25	ROLLBACK;	
26		1 row processed. (競合する表ロックが解放されました。) ROLLBACK;
27	LOCK TABLE hr.departments IN SHARE ROW EXCLUSIVE MODE; Statement processed.	
28		LOCK TABLE hr.departments IN EXCLUSIVE MODE NOWAIT; ORA-00054
29		LOCK TABLE hr.departments IN SHARE ROW EXCLUSIVE MODE NOWAIT; ORA-00054
30		LOCK TABLE hr.departments IN SHARE MODE NOWAIT; ORA- 00054
31		LOCK TABLE hr.departments IN ROW EXCLUSIVE MODE NOWAIT;

タイム・ポイント	トランザクション1	トランザクション2
		ORA-00054
32		LOCK TABLE hr.departments IN SHARE MODE NOWAIT; ORA- 00054
33		SELECT location_id FROM hr.departments WHERE department_id = 20; LOCATION_ID ----- DALLAS 1 row selected.
34		SELECT location_id FROM hr.departments WHERE department_id = 20 FOR UPDATE OF location_id; LOCATION_ID - ----- DALLAS 1 row selected.
35		UPDATE hr.departments SET location_id = 'NEW YORK' WHERE department_id = 20; (トランザクション 1 に競合する表 ロックがあるため待機します。)
36	UPDATE hr.departments SET location_id = 'NEW YORK' WHERE department_id = 20; (トランザクション 2 によって同じ行 がロックされているため待機します。)	(デッドロック。)
37	Cancel operation. ROLLBACK;	
38		1 row processed.
39	LOCK TABLE hr.departments IN EXCLUSIVE MODE;	
40		LOCK TABLE hr.departments IN EXCLUSIVE MODE; ORA-00054
41		LOCK TABLE hr.departments IN ROW EXCLUSIVE MODE NOWAIT; ORA-00054
42		LOCK TABLE hr.departments IN SHARE MODE; ORA-00054

タイム・ポイント	トランザクション1	トランザクション2
43		LOCK TABLE hr.departments IN ROW EXCLUSIVE MODE NOWAIT; ORA-00054
44		LOCK TABLE hr.departments IN ROW SHARE MODE NOWAIT; ORA-00054
45		SELECT location_id FROM hr.departments WHERE department_id = 20; LOCATION_ID ----- DALLAS 1 row selected.
46		SELECT location_id FROM hr.departments WHERE department_id = 20 FOR UPDATE OF location_id; (トランザクション1 に競合する表 ロックがあるため待機します。)
47	UPDATE hr.departments SET department_id = 30 WHERE department_id = 20; 1 row processed.	
48	COMMIT;	
49		0 rows selected. (トランザクション1 の競合するロッ クが解放されました。)
50	SET TRANSACTION READ ONLY;	
51	SELECT location_id FROM hr.departments WHERE department_id = 10; LOCATION_ID ----- BOSTON	
52		UPDATE hr.departments SET location_id = 'NEW YORK' WHERE department_id = 10; 1 row processed.
53	SELECT location_id FROM hr.departments WHERE department_id = 10; LOCATION_ID -----	

タイム・ポイント	トランザクション1	トランザクション2
	BOSTON (トランザクション1には、未コミット・データが表示されません。)	
54		COMMIT;
55	SELECT location_id FROM hr.departments WHERE department_id = 10; LOCATION_ID ----- BOSTON (トランザクション2がコミットした後でも同じ結果になります。)	
56	COMMIT;	
57	SELECT location_id FROM hr.departments WHERE department_id = 10; LOCATION_ID ----- NEW YORK (コミット済データが表示されます。)	

関連項目:

[Oracle Database概要](#)

8.5 Oracle Lock Managementサービス(ユーザー・ロック)の使用

DBMS_LOCKパッケージのサブプログラムを起動することで、アプリケーションはOracle Lock Managementサービス(ユーザー・ロック)を使用できます。アプリケーションは、特定のモードのロックを要求し、同一または別のインスタンスの別のサブプログラムで認識できる一意の名前を付け、ロック・モードを変更し、解除できます。確保されるユーザー・ロックはOracle Databaseロックであるため、デッドロックの検出などのデータベース・ロックのすべての機能を持っています。分散トランザクションで使用されるユーザー・ロックは、COMMITと同時に解除されるようになっていることを確認してください。解除されないと、検出されないデッドロックが発生する可能性があります。

関連項目:

DBMS_LOCK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。

トピック:

- [ユーザー・ロックを使用する場合](#)

- [ロックの表示および監視](#)

8.5.1 ユーザー・ロックを使用する場合

次のような場合、ユーザー・ロックが役立ちます。

- 端末などの装置に対して排他的アクセスを可能にする場合
- アプリケーション・レベルの読み込みロックを適用する場合
- ロックの解除を検出し、アプリケーションの終了後にクリーンアップする場合
- アプリケーションを同期化して、順次処理を実行する場合

[例8-2](#)は、複数のユーザーが1つのデバイスにアクセスする必要がある場合に、競合が発生しないように保証するための Pro*COBOLプリコンパイラでのロックの使用方法を示しています。

例8-2 Pro*COBOLプリコンパイラによるロックの使用方法

```
*****
* Print Check *
* Any cashier may issue a refund to a customer returning goods. *
* Refunds under $50 are given in cash, more than $50 by check. *
* This code prints the check. One printer is opened by all *
* the cashiers to avoid the overhead of opening and closing it *
* for every check, meaning that lines of output from multiple *
* cashiers can become interleaved if you do not ensure exclusive *
* access to the printer. The DBMS_LOCK package is used to *
* ensure exclusive access. *
*****
CHECK-PRINT
* Get the lock "handle" for the printer lock.
MOVE "CHECKPRINT" TO LOCKNAME-ARR.
MOVE 10 TO LOCKNAME-LEN.
EXEC SQL EXECUTE
    BEGIN DBMS_LOCK.ALLOCATE_UNIQUE ( :LOCKNAME, :LOCKHANDLE );
    END; END-EXEC.
* Lock the printer in exclusive mode (default mode).
EXEC SQL EXECUTE
    BEGIN DBMS_LOCK.REQUEST ( :LOCKHANDLE );
    END; END-EXEC.
* You now have exclusive use of the printer, print the check.
...
* Unlock the printer so other people can use it
EXEC SQL EXECUTE
    BEGIN DBMS_LOCK.RELEASE ( :LOCKHANDLE );
    END; END-EXEC.
```

8.5.2 ロックの表示および監視

[表8-3](#)では、インスタンスで処理中のトランザクションのロック情報を表示する、Oracle Databaseの機能について説明しています。

表8-3 ロック情報の表示方法

ツール	説明:
パフォーマンス監視データ・ディクショナリ・ビュー	『Oracle Database 管理者ガイド』 を参照してください。

ツール	説明:
UTLLOCKT.SQL	UTLLOCKT.SQL スクリプトは、ツリー構造の簡単なロック待機グラフを表示します。スクリプトの実行に SQL ツール(SQL*Plus など)を使用し、システム内のロック待機中のセッション、およびそれに対応するブロッキング・ロックを出力します。このスクリプト・ファイルの位置は、オペレーティング・システムによって異なります。(UTLLOCKT.SQL を使用する前に、CATBLOCK.SQL スクリプトを実行する必要があります。)

8.6 シリアル化可能トランザクションを使用した同時実行性の制御

Oracle Databaseは、デフォルトでは、同時に実行されるトランザクションの同じ表および同じデータ・ブロック内での行の修正、追加または削除を許可しています。トランザクションAが表を変更すると、トランザクションAがコミットされるまで、同時に実行しているトランザクションでは参照できません。トランザクションAが、(DML文またはSELECT FOR UPDATE文を使用して)別のトランザクションBによってロックされている行を更新または削除しようとする、トランザクションAのDML文は、トランザクションBがコミットまたはロールバックされるまでブロックされます。この同時実行性モデルは、より高度な同時実行性が提供されて、パフォーマンスが改善されるため、ほとんどのアプリケーションに適しています。

ただし、まれにシリアル化可能なトランザクションが必要な場合もあります。シリアル化可能トランザクションは、シリアル・モードで同時に実行します。シリアル・モードでは、同時トランザクションは、それらのトランザクションが順次(つまり一度に1つずつ)実行された場合に可能となるデータベースの変更のみが可能です。シリアル化可能トランザクションの開始以降に、別のトランザクションにより変更されたデータを変更しようとする、ORA-08177が発生します。

シリアル化可能トランザクションがORA-08177で失敗した場合、アプリケーションは次のいずれかで対処できます。

- そのポイントまで実行された作業をコミットします。
- その他の様々な文を、トランザクション内の直前のセーブポイントまでロールバックした後に実行します。
- トランザクションをロールバックし再度実行します。

トランザクションがトランザクション・スナップショットを取得するため、操作が成功する可能性が高くなります。

ヒント:



トランザクションのロールバックおよび再実行によるパフォーマンスのオーバーヘッドを最小化するには、他の同時トランザクションと競合する可能性のある DML 文は、できるだけトランザクションの始めの方に置くようにしてください。

ノート:



シリアル化可能トランザクションは、遅延セグメント作成または時間隔パーティションでは動作しません。セグメントが作成されていない空の表、またはセグメントが存在しない時間隔パーティション表のパーティションにデータを挿入しようとすると、エラーになります。

トピック:

- [トランザクションの相互作用と分離レベル](#)

- [分離レベルの設定](#)
- [シリアライズ可能トランザクションおよび参照整合性](#)
- [READ COMMITTEDおよびSERIALIZABLE分離レベル](#)

8.6.1 トランザクションの相互作用と分離レベル

ANSI/ISO SQL標準は3種類のトランザクションの相互作用を定義しています。

トランザクションの相互作用	定義
内容を保証しない読み込み	トランザクション A はトランザクション B によるコミットされていない変更を読み込みます。
反復不能な読み込み	トランザクション A はデータを読み込み、トランザクション B がデータを変更して変更をコミットし、トランザクション A がデータを再度読み込み、変更を確認します。
仮読み込み	トランザクション A が問合せを実行し、トランザクション B が新しい行を挿入して変更をコミットし、トランザクション A が問合せを繰り返して新しい行を確認します。

トランザクションが可能な相互作用の種類は分離レベルによって決まります。ANSI/ISO SQL標準は4レベルの分離を定義しています。[表8-4](#)に、各分離レベルで可能な相互作用の種類を示します。

表8-4 ANSI/ISO SQL分離レベルおよび可能なトランザクションの相互作用

分離レベル	内容を保証しない読み込み	反復不能な読み込み	仮読み込み
READ UNCOMMITTED	可能性あり	可能性あり	可能性あり
READ COMMITTED	可能性なし	可能性あり	可能性あり
REPEATABLE READ	可能性なし	可能性なし	可能性あり
SERIALIZABLE	可能性なし	可能性なし	可能性なし

[表8-5](#)に、Oracle Databaseで用意されるANSI/ISO SQLトランザクション分離レベルを示します。

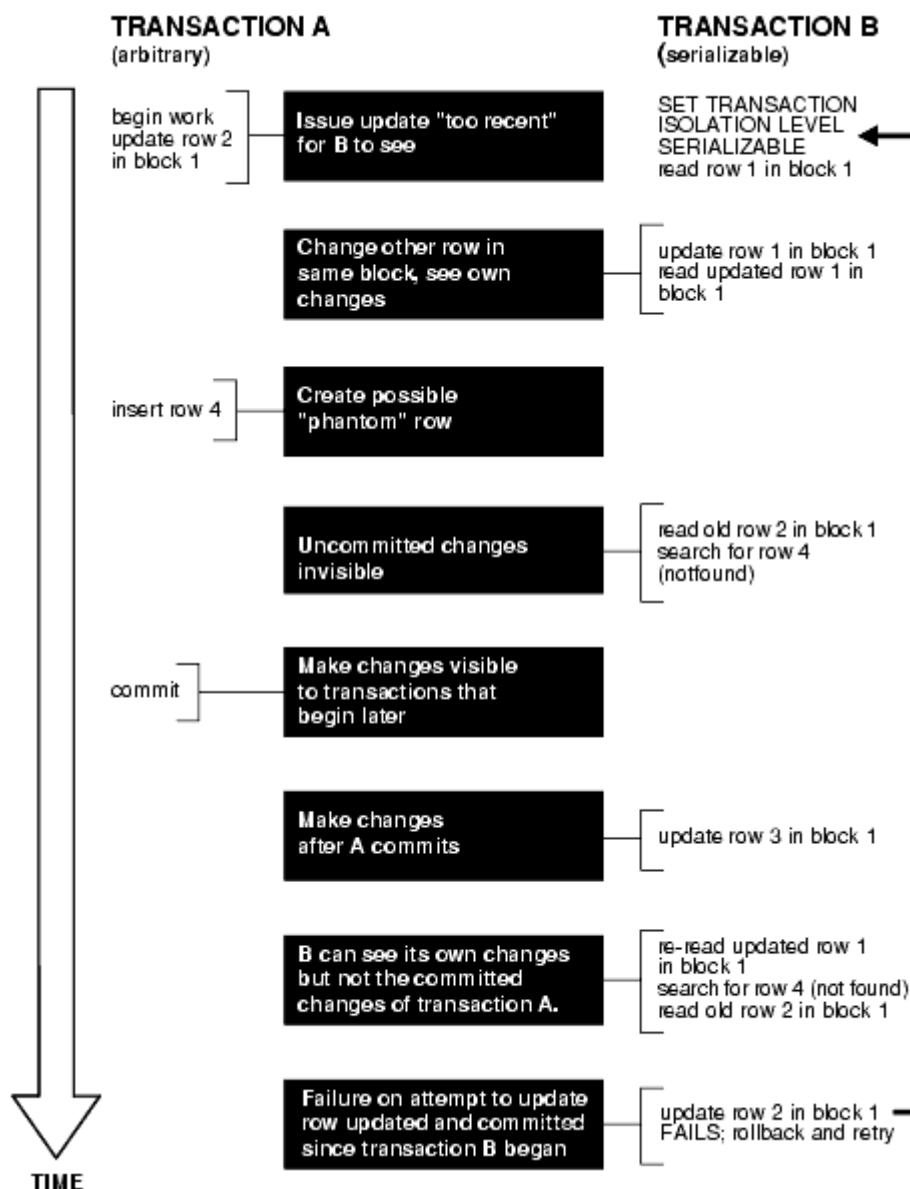
表8-5 Oracle Databaseで用意されるANSI/ISO SQL分離レベル

分離レベル	Oracle Databaseで用意される
READ UNCOMMITTED	いいえ、Oracle Database では「内容を保証しない読み込み」は許可されません。他のデータベース製品の中には、スループットの改善のために、この方法を使用するものもありますが、スループットの高い Oracle Database では不要です。
READ COMMITTED	はい(デフォルト)。Oracle Database では、問合せの始め(スナップショット時)にコミット済の

分離レベル	Oracle Databaseで用意される
	データのみを参照するため、READ COMMITTED 分離について ANSI/ISO SQL 標準以上の整合性を提供します。
REPEATABLE READ	はい、トランザクション分離レベルを SERIALIZABLE に設定する場合。
SERIALIZABLE	はい、トランザクション分離レベルを SERIALIZABLE に設定する場合。

図8-1に、任意のトランザクション(SERIALIZABLEまたはREAD COMMITTEDのいずれか)とシリアライズ可能トランザクションとの相互作用を示します。

図8-1 シリアライズ可能トランザクションと別のトランザクションの相互作用



8.6.2 分離レベルの設定

セッション内のすべてのトランザクションのトランザクション分離レベルを設定するには、ALTER SESSION文を使用します。

トランザクション分離レベルを特定のトランザクションに設定するには、SET TRANSACTION文のISOLATION LEVEL句を使

用します。SET TRANSACTION文は、トランザクション内の最初の文である必要があります。

ノート:



トランザクション分離レベルを `SERIALIZABLE` に設定する場合は、`ALTER TABLE` 文を使用して、`INITRANS` パラメータを 3 以上に設定する必要があります。多数のトランザクションが同じブロックを更新する表では、この値を大きくします。`INITRANS` の詳細は、次を参照してください。

関連項目:

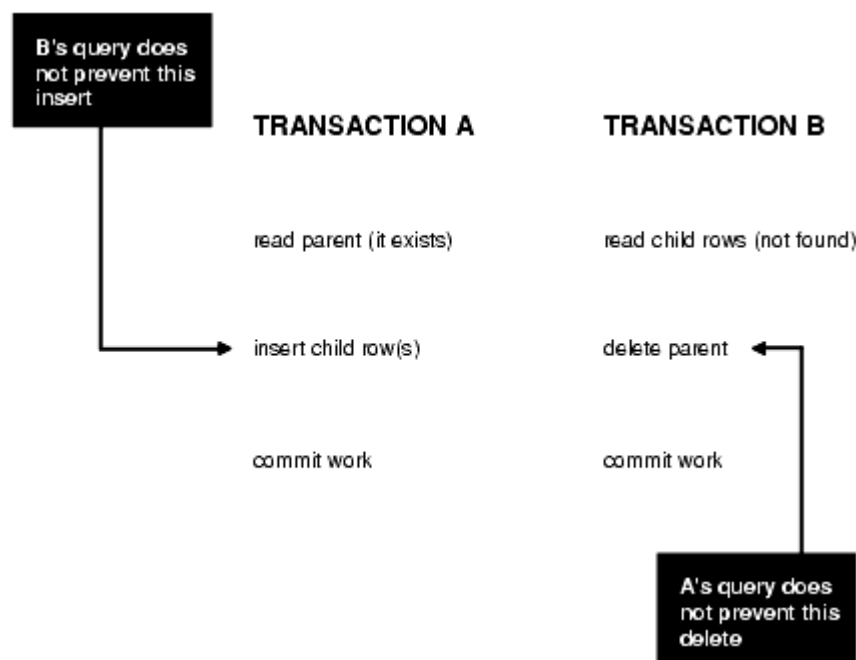
[Oracle Database SQL言語リファレンス](#)

8.6.3 シリアライズ可能トランザクションおよび参照整合性

Oracle Databaseは、`SERIALIZABLE`トランザクション内であっても読み込みロックを使用しないため、あるトランザクションによって読み込まれたデータは、別のトランザクションでオーバーライドできます。そのため、アプリケーション・レベルでデータベースの整合性チェックを実行するトランザクションでは、読み込んだデータはトランザクション中には変更されないと考えないでください(そのような変更がトランザクションからはわからない場合も)。`SERIALIZABLE`トランザクションを使用した場合でも、アプリケーション・レベルの整合性チェックのコードを十分注意して作成してください。

図8-2に、2つの表の間の参照整合性の親子関係を保持するために、アプリケーション・レベルでチェックを実行するトランザクションAとB (`READ COMMITTED`または`SERIALIZABLE`のいずれか)を示します。トランザクションAは、親表に問い合わせで特定の主キー値を持つ行が存在するかどうかをチェックした後、対応する子行を子表に挿入します。トランザクションBは、子表に問い合わせで特定の主キー値を持つ子行が存在しないことをチェックした後、親表から対応する親行を削除します。この場合、両方のトランザクションが読み込んだデータは、そのトランザクションが完了する前には変更されないものと想定しています(確認はしません)。

図8-2 参照整合性チェック



トランザクションAが実行した問合せのために、トランザクションBが親行を削除できなくなることはありません。トランザクションBが実行した問合せのために、トランザクションAが子行を挿入できなくなることもありません。したがって、次のことが起こる可能性が

あります。

1. トランザクションAは親表に問い合せて、指定した親行を確認します。
2. トランザクションBは子表に問い合せて、指定した親行が子行を持たないことを確認します。
3. 指定した親行を確認して、トランザクションAは対応する子行を子表に挿入します。
4. 指定した親行が子行を持たないことを確認して、トランザクションBは指定した親行を親表から削除します。

これで、トランザクションAがステップ3で挿入した子行には親行がなくなりました。

このような結果は、AおよびBの両方がSERIALIZABLEトランザクションであったとしても発生する可能性があります。どちらのトランザクションも、整合性チェックのために読み込んだデータに対する変更を他方が妨げないからです。

あるトランザクションが問い合せたデータが別のトランザクションにより同時に変更または削除されないようにするには、ANSI/ISO SQL標準のSERIALIZABLE分離レベルが提供するよりも高いレベルのトランザクション分離が必要です。ただしOracle Databaseでは、次のことが可能です。

- トランザクションAはSELECT FOR UPDATE文を使用して、親行を問い合せてロックし、トランザクションBがその行を削除しないようにします。
- トランザクションBは、処理ステップを逆にして、トランザクションAが親行を見つけられないようにします(その結果、子行を挿入できなくなります)。つまりトランザクションBは次のことができます。
 1. 親行を削除します。
 2. 子表を問い合せます。
 3. 削除した親行に、子表内の子行がある場合は、親行の削除をロールバックします。

または、トリガーを使用して参照整合性を施行することもできます。トランザクションAが親表を問い合せるかわりに、子表で次を行う行レベルBEFORE INSERTトリガーを定義します。

- SELECT FOR UPDATE文を使用して親表を問い合せ、親行が存在するかどうかを確認することで、子行を挿入するトランザクションの処理中に、親行がデータベース内に残るようにします。
- 親行が存在しない場合、子行の挿入は拒否されます。

トリガーによって発行されたSQL文は、そのトリガーを起動した文のコンテキスト内で実行されます(トリガー起動文とトリガー起動された文は同じデータベース状態を参照します)。そのため、READ COMMITTEDトランザクションがトリガー起動文を実行する場合、トリガー起動された文は、トリガー起動文の実行開始時点のデータベースを参照します。SERIALIZABLEトランザクションがトリガー起動文を実行する場合、トリガー起動された文は、そのトランザクションの開始時点のデータベースを参照します。いずれの場合も、トリガーで SELECT FOR UPDATEを使用すると、参照整合性が正しく施行されます。

関連項目:

- SELECT文のFOR UPDATE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。
- トリガーを使用した親表と子表の間の参照整合性の維持の詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。

8.6.4 READ COMMITTEDおよびSERIALIZABLE分離レベル

Oracle Databaseでは、READ COMMITTEDおよびSERIALIZABLEという2つのトランザクション分離レベルが用意されてい

ます。どちらのレベルも高度な一貫性および同時実行性を提供し、競合を軽減し、実社会でのアプリケーション用に設計されています。この項では、2つの分離レベルを比較し、その選択方法を説明します。

トピック:

- [トランザクション集合の整合性の違い](#)
- [トランザクション分離レベルの選択](#)

8.6.4.1 トランザクション集合の整合性の違い

どの読み込みにおいても、同じコミット済トランザクション集合によって書き込まれたデータが戻される場合、操作(問合せまたはトランザクション)は、「**トランザクション集合整合である**」といいます。トランザクション集合整合でない操作では、ある集合のトランザクションの変更が反映される読み込みと、他のトランザクションによって行われた変更が反映される読み込みが存在します。そのような操作では、そのデータベースは、コミットされたトランザクション集合が反映されていない状態のデータベースのように見えます。

トピック:

- [Oracle Database](#)
- [他のデータベース・システム](#)

8.6.4.1.1 Oracle Database

READ COMMITTED分離レベルでのOracle Databaseトランザクションは、問合せによって読み込まれたすべての行が、その問合せが始まる前にコミットされている必要があるため、文単位でのトランザクション集合整合です。

SERIALIZABLE分離レベルでのOracle Databaseトランザクションは、SERIALIZABLEトランザクション内のすべての文が、トランザクション開始時点のデータベースのイメージに対して実行されるため、トランザクション単位でのトランザクション集合整合です。

8.6.4.1.2 他のデータベース・システム

他のデータベース・システムでは、READ UNCOMMITTED分離レベルでの1回の問合せは、トランザクション集合整合ではありません。別のトランザクションによって行われた変更のサブセットしか見えないからです。たとえば、ディテール表とマスター表を結合すると、別のトランザクションによって挿入されたマスター・レコードを見ることはできますが、そのトランザクションによって挿入された対応するディテールは見えません(その逆も同じです)。READ COMMITTED分離レベルではこのような問題は回避されるため、読み込みロック・システムより高い整合性が得られます。

読み込みロック・システムでは、同時更新ができないようにするかわりに、REPEATABLE READ分離レベルによって、トランザクション・レベルではなく、文レベルでトランザクション集合の整合性が提供されます。仮読み込み保護がないため、同一のトランザクションによる2つの問合せが、トランザクションの別の集合によってコミットされたデータを参照できます。これらのシステムでは、スループットに制限がありデッドロックされやすいSERIALIZABLE分離レベルの場合のみ、トランザクション・レベルでのトランザクション集合の整合性が提供されます。

8.6.4.2 トランザクション分離レベルの選択

トランザクション分離レベルの選択は、パフォーマンスと整合性の必要性、およびアプリケーション・コーディング要件により異なります。同時実行性(トランザクションのスループット)と整合性はトレードオフの関係にあります。トランザクションの分離レベルの選択では、アプリケーションと作業負荷を考慮します。異なるトランザクションにはそれぞれ個別の分離レベルを選択できます。

多数のユーザーが、トランザクションを同時に次々に送る環境の場合、予期されるトランザクション到着頻度、応答時間要件、および必要な整合性の程度を考慮します。

READ COMMITTED分離レベルでは、一部のトランザクションについては(仮読みおよび反復不能な読みからの)一貫性のない結果が生成される可能性は多少高くなりますが、かなり高い同時実行性を提供できます。

SERIALIZABLE分離レベルの場合は、仮読みおよび反復不能な読みから保護されているため、より高い整合性が提供され、読み/書きトランザクションが問合せを2回以上実行する場合にはこの分離レベルは重要です。ただし、SERIALIZABLE分離レベルでは、アプリケーションが「このトランザクションのアクセスをシリアル化できません」というエラーの有無を確認する必要があり、多数の同時トランザクションが更新のために同じデータにアクセスする環境では、スループットはかなり低下する可能性があります。

「シリアライズ可能トランザクションと参照整合性」で説明したように、READ COMMITTEDまたはSERIALIZABLEトランザクションのいずれでも読みは書きをブロックしません。

[表8-6](#)に、READ COMMITTEDトランザクションとSERIALIZABLEトランザクションの類似点および相違点の概要を示します。

表8-6 READ COMMITTEDトランザクションとSERIALIZABLEトランザクションの比較

演算子	コミット読み	シリアライズ可能
内容を保証しない書き込み	可能性なし	可能性なし
内容を保証しない読み	可能性なし	可能性なし
反復不能な読み	可能性あり	可能性なし
仮読み	可能性あり	可能性なし
ANSI/ISO SQL 92 への準拠	あり	あり
スナップショット読み時間	文	トランザクション
トランザクション集合の整合性	文レベル	トランザクション・レベル
行レベル・ロック	あり	あり
読みが書き込みをブロック	いいえ	いいえ
書き込みが読みをブロック	いいえ	いいえ
異なる行の書き込みが書き込みをブロック	いいえ	いいえ
同じ行の書き込みが書き込みをブロック	あり	あり
阻止しているトランザクションの待機	あり	あり

演算子	コミット読み	シリアライズ可能
エラー「このトランザクションのアクセスをシリアル化できません」の発生する可能性	いいえ	あり
阻止しているトランザクションの終了後のエラー	いいえ	いいえ
阻止しているトランザクションのコミット後のエラー	いいえ	あり

関連項目:

[シリアライズ可能トランザクションおよび参照整合性](#)

8.7 非ブロック化およびブロック化のDDL文

DDL文の非ブロック化とブロック化の違いは、表または索引のどちらか(表によって異なる)を変更するDDL文についてのみ重要です。

オブジェクトXに影響するDDL文をセッションが発行すると、Xを参照するすべての同時DML文がコミットされるかロールバックされるまで、セッションは待機します。

セッションが待機している間、同時セッションが新規のDML文を発行する可能性があります。DDL文が非ブロック化の場合、新規のDML文はただちに実行されます。DDL文がブロック化の場合、新規のDML文はDDL文の完了後に実行され、成功するかエラーが発生します。

DDL_LOCK_TIMEOUTパラメータはブロック化DDL文に影響します(ただし、非ブロック化DDL文には影響しません)。したがって、ブロック化DDL文の完了にはエラーORA-00054 (「リソース・ビジー。NOWAITが指定されているか、タイムアウトしました」)が伴う場合があります。

表のパーティションに適用されるDDL文は、そのパーティションについてはブロック化、同じ表の他のパーティションについては非ブロック化です。

ノート:



サブリメンタル・ロギングがデータベース・レベルで(マルチテナント・コンテナ・データベースまたはプラグブル・データベースについて)有効になっている場合、データベースでは非ブロック化 DDL 文がブロック化 DDL 文のように扱われます。

注意:



自律型トランザクションでは非ブロック化 DDL 文を発行しないでください。自律型トランザクションの詳細は、[「自律型トランザクション」](#)を参照してください。

関連項目:

- DDL_LOCK_TIMEOUT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。
- 非ブロック化DDL文のリストは、[「B SQL操作中の自動および手動ロック・メカニズム」](#)を参照してください。
- サプリメンタル・ロギングの有効化および無効化の詳細は、[「ALTER DATABASE」](#)を参照してください。

8.8 自律型トランザクション

注意:



自律型トランザクションでは非ブロック化 DDL 文を発行しないでください。

自律型トランザクション(AT)は、別のトランザクション(**メイン・トランザクション(MT)**)によって開始される独立したトランザクションです。自律型トランザクションを使用すると、メイン・トランザクションを停止して、SQL操作を実行し、そのSQL操作をコミットまたはロールバックした後でメイン・トランザクションを再開できます。

たとえば、株式売買トランザクションでは、売買が成功するかどうかに関係なく、顧客情報のコミットが必要な場合があります。または、トランザクションがロールバックされた場合でも、エラー・メッセージをデバッグ表にログする必要がある場合があります。自律型トランザクションを使用すると、これらのタスクを実行できます。

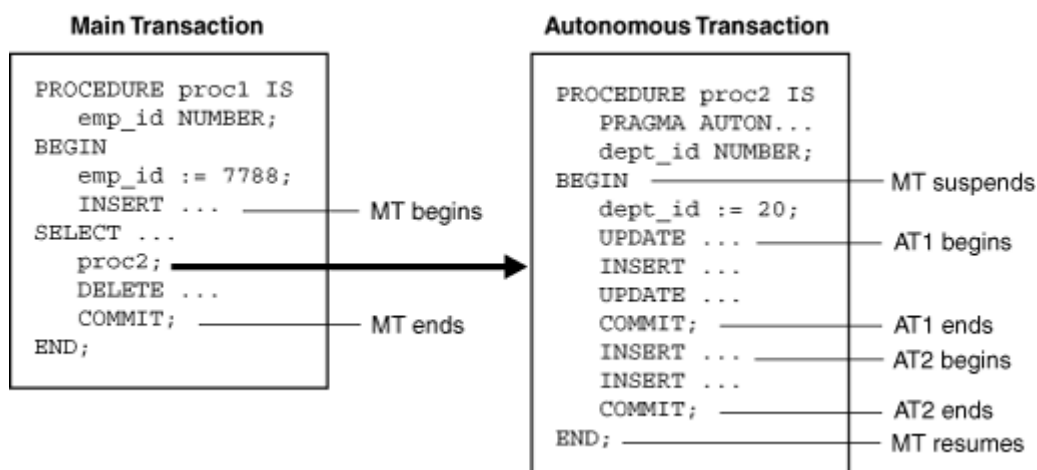
自律型トランザクションは**自律型スコープ**内、つまり、自律型ルーチン(AUTONOMOUS_TRANSACTIONプラグマでマークされるルーチン)の範囲内で実行されます。このコンテキストでは、ルーチンは次のいずれかになります。

- スキーマレベル(ネストしていない)の無名PL/SQLブロック
- スタンドアロン・サブプログラム、パッケージ・サブプログラムまたはネストしたサブプログラム
- ADTのメソッド
- 非複合トリガー

自律型ルーチンで複数の自律型トランザクションをコミットできます。

[図8-3](#)に、メイン・トランザクション(proc1)から自律型ルーチン(proc2)へ制御が行き来する様子を示します。自律型ルーチンは、制御がメイン・トランザクションに戻る前に2つのトランザクション(AT1およびAT2)をコミットします。

図8-3 トランザクション制御フロー



自律型トランザクションの実行可能セクションに入ると、メイン・トランザクションが停止します。自律型トランザクションを終了する

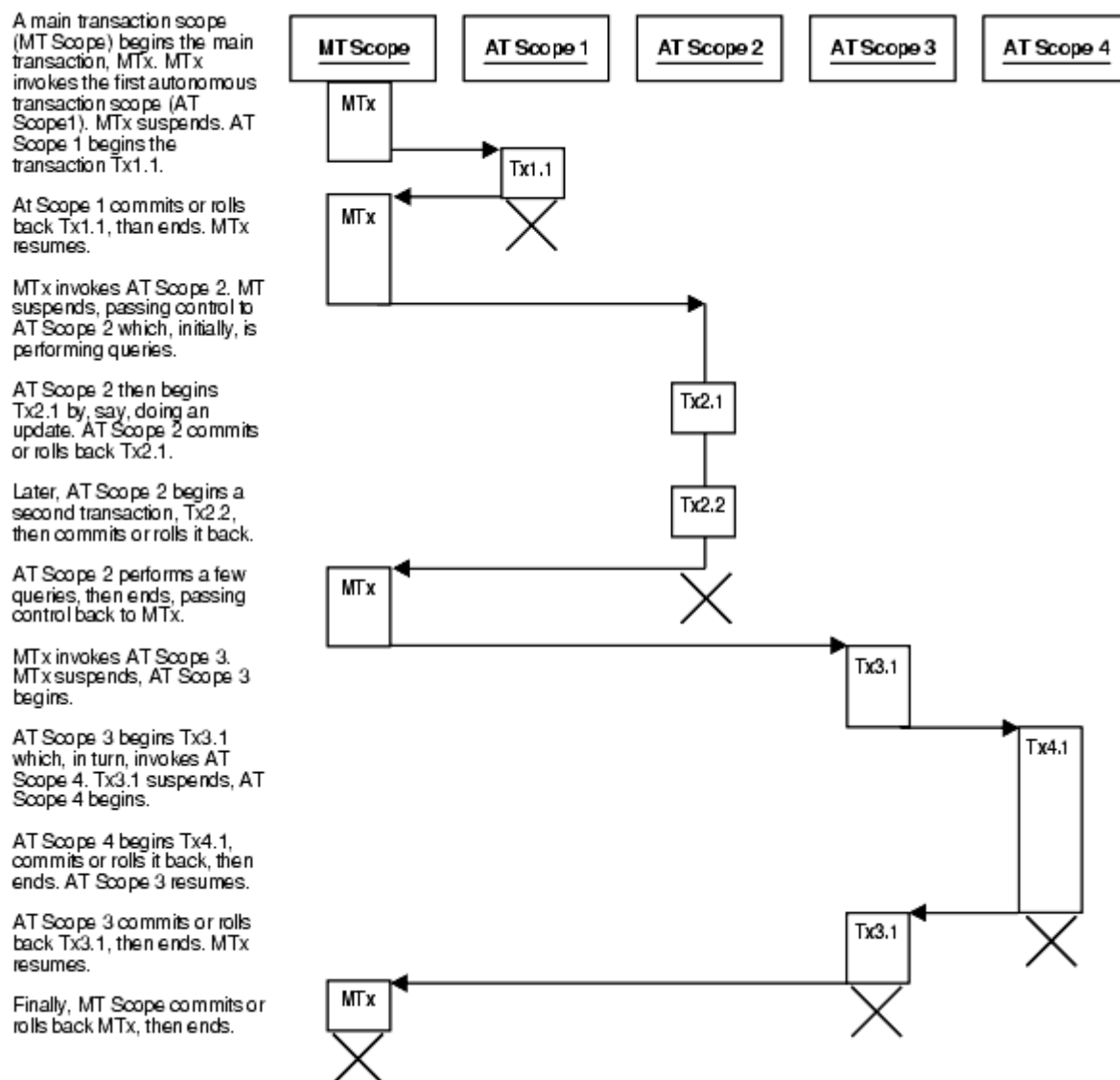
と、メイン・トランザクションが再開します。COMMITおよびROLLBACKによって、アクティブな自律型トランザクションは終了しますが、自律型トランザクションは終了しません。[図8-3](#)に示すとおり、1つのトランザクションが終了すると、次のSQL文が別のトランザクションを開始します。

自律型トランザクションの特長をさらに示します。

- 自律型トランザクションが加える変更は、メイン・トランザクションの状態または最終的な処理には依存しません。たとえば:
 - 自律型トランザクションは、メイン・トランザクションによって加えられた変更を認識しません。
 - 自律型トランザクションがコミットまたはロールバックしても、メイン・トランザクションの結果には影響しません。
- 自律型トランザクションが加える変更は、そのトランザクションがコミットした直後に他のトランザクションで参照できます。このため、メイン・トランザクションがコミットするまで待機しなくても、ユーザーは更新された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。

[図8-4](#)に、自律型トランザクションが従う実行順序の例を示します。

図8-4 自律型トランザクションの順序



トピック:

- [自律型トランザクションの例](#)

- [自律型ルーチンの宣言](#)

関連項目:

- 自律型トランザクションの詳細は、『[Oracle Database PL/SQL言語リファレンス](#)』を参照してください。
- 非ブロック化DDL文の詳細は、『[非ブロック化およびブロック化のDDL文](#)』を参照してください。

8.8.1 自律型トランザクションの例

この項では、自律型トランザクションの例を示します。

トピック:

- [商品の注文](#)
- [銀行口座からの出金処理](#)

例に示すとおり、自律型トランザクションおよびメイン・トランザクションを使用する場合は、4種類の結果が考えられます([表8-7](#)を参照)。自律型トランザクションの結果とメイン・トランザクションの結果の間に依存性はありません。

表8-7 トランザクションの結果

自律型トランザクション	メイン・トランザクション
コミット	コミット
コミット	ロールバック
ロールバック	コミット
ロールバック	ロールバック

8.8.1.1 商品の注文

[図8-5](#)に、顧客がある商品を注文する例を示します。購買契約が成立しなくても、その顧客の情報(名前、住所、電話番号など)は顧客情報表にコミットされます。

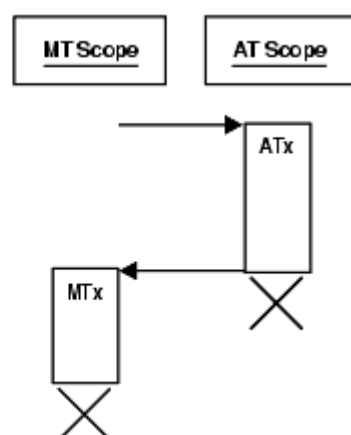
図8-5 例: 購買指示

MT Scope begins the main transaction, MTx inserts the buy order into a table.

MTx invokes the autonomous transaction scope (AT Scope). When AT Scope begins, MT Scope suspends.

ATx, updates the audit table with customer information.

MTx seeks to validate the order, finds that the selected item is unavailable, and therefore rolls back the main transaction.



8.8.1.2 銀行口座からの出金処理

この例では、顧客は銀行口座から払戻しを実行しようとしています。この処理で、メイン・トランザクションは2つの自律型トランザクション・スコープ(ATスコープ1またはATスコープ2)のいずれかを起動します。

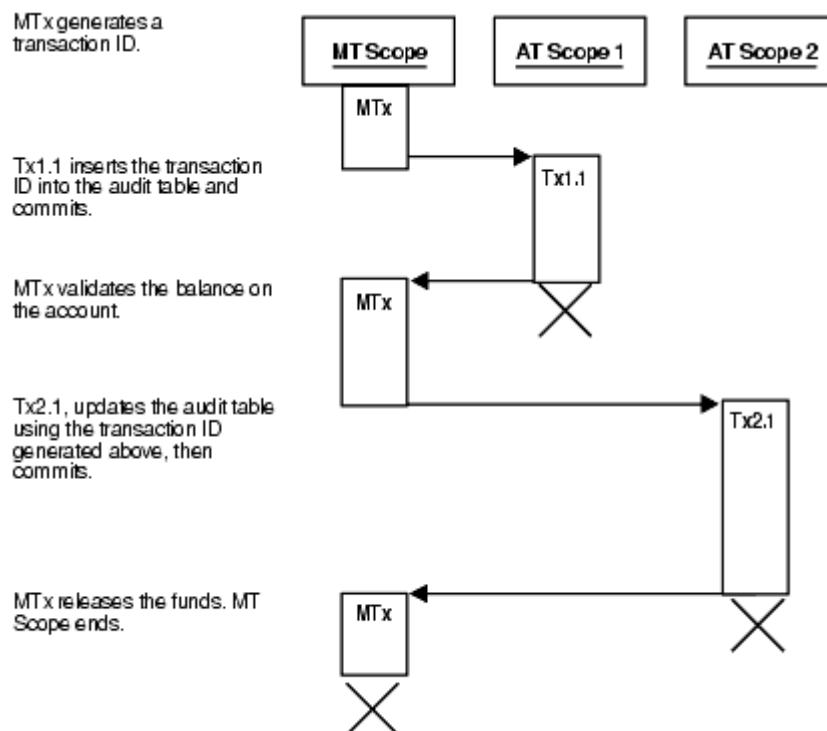
このトランザクションで考えられる使用例を次に示します。

- [使用例1: 十分な預金残高がある場合](#)
- [使用例2: 貸越し保護があり預金残高が不十分な場合](#)
- [使用例3: 貸越し保護がなく預金残高が不十分な場合](#)

8.8.1.2.1 使用例1: 十分な預金残高がある場合

払戻しに十分な預金残高があり、銀行が払戻しに応じます。(図8-6を参照)。

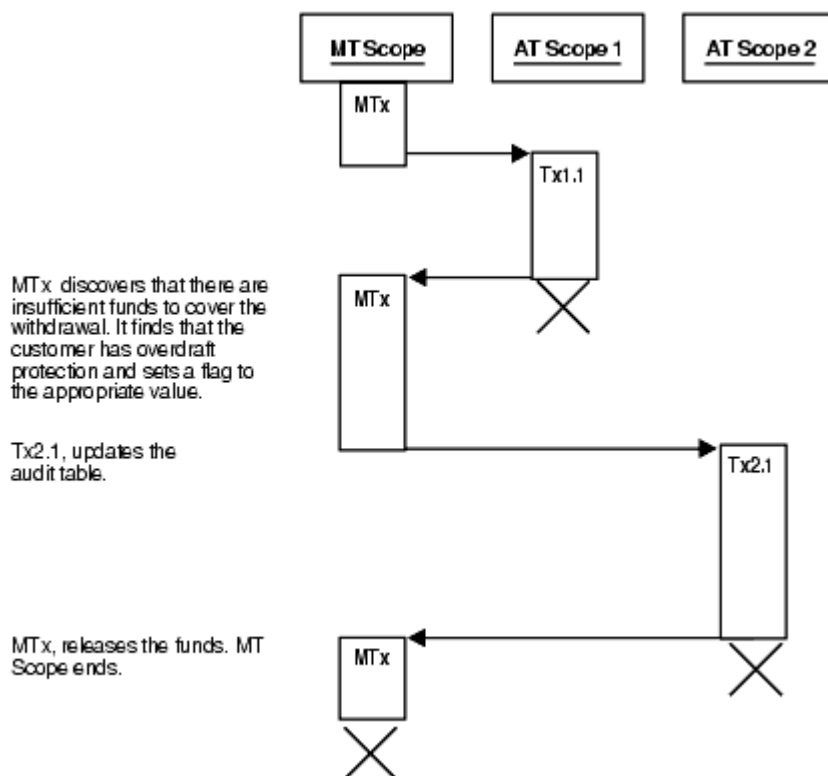
図8-6 預金払戻し - 十分な預金残高がある場合



8.8.1.2.2 使用例2: 貸越し保護があり預金残高が不十分な場合

払戻しに十分な預金残高はありませんが、この顧客には貸越し保護があるため、銀行が払戻しに応じます(図8-7を参照)。

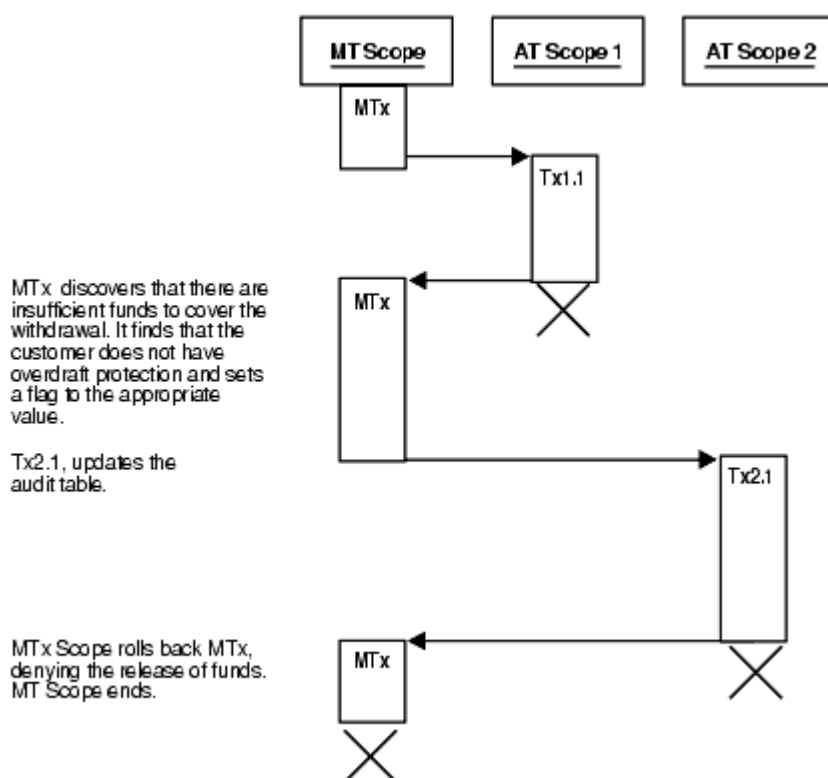
図8-7 預金払戻し - 貸越し保護があり預金残高が不十分な場合



8.8.1.2.3 使用例3: 貸越し保護がなく預金残高が不十分な場合

払戻しに十分な預金残高はなく、この顧客には貸越し保護もありません。したがって、銀行は払戻しを差し止めます(図8-8を参照)。

図8-8 預金払戻し - 貸越し保護がなく預金残高が不十分な場合



8.8.2 自律型ルーチンの宣言

自律型ルーチンを宣言するには、PRAGMA AUTONOMOUS_TRANSACTIONを使用します。これは、PL/SQLコンパイラに対して、ルーチンを自律型としてマークするように指示します。

関連項目:

PRAGMA AUTONOMOUS_TRANSACTION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。

[例8-3](#)では、ファンクションbalanceが自律型です。

例8-3 パッケージ・サブプログラムの自律型としてのマーク

```
-- Create table for package to use:

DROP TABLE accounts;
CREATE TABLE accounts (account INTEGER, balance REAL);

-- Create package:

CREATE OR REPLACE PACKAGE banking AS
  FUNCTION balance (acct_id INTEGER) RETURN REAL;
  -- Additional functions and packages
END banking;
/
CREATE OR REPLACE PACKAGE BODY banking AS
  FUNCTION balance (acct_id INTEGER) RETURN REAL IS
    PRAGMA AUTONOMOUS_TRANSACTION;
    my_bal REAL;
  BEGIN
    SELECT balance INTO my_bal FROM accounts WHERE account=acct_id;
    RETURN my_bal;
  END;
  -- Additional functions and packages
END banking;
/
```

8.9 記憶域割当てエラー後の実行の再開

長時間にわたって実行されるトランザクションが記憶域割当てエラー状態によって中断されたときに、アプリケーションによって問題が発生した文を一時停止し、問題を修正した後でその文を再開することが可能です。この機能を、**再開可能記憶域割当て**といいます。この機能によって、時間がかかるロールバックを回避できます。また、操作を小さく分割したり、処理過程を追跡するコードを作成する必要がなくなります。

関連項目:

再開可能領域割当ての詳細は、[『Oracle Database管理者ガイド』](#)を参照してください。

トピック:

- [再開可能記憶域割当てを含む操作](#)
- [一時停止された記憶域割当ての処理](#)

8.9.1 再開可能記憶域割当てを含む操作

問合せ、DML操作および特定のDDL操作には、次のエラーの後で再開可能な記憶域割当てが含まれます。

- ORA-01653などの領域不足エラー。
- ORA-01628などの領域制限エラー。

- ORA-01536などの領域割当てエラー。

再開可能記憶域割当ては、操作がSQL文によって直接実行されているか、またはSQL*Loader、ストアド・サブプログラム、無名PL/SQLブロック、OCIStmtExecuteなどのOCIコール内で実行されている場合に適用されます。

ディクショナリ管理された表領域では、ロールバック・セグメントの上限に達するか、エクステントの最大数に達した場合、索引または表の作成操作を再開できません。ローカル管理表領域および自動UNDO管理を、再開可能記憶域割当てと組み合わせて使用してください。

8.9.2 一時停止された記憶域割当ての処理

記憶域割当てエラーのためにアプリケーション内の文が一時停止された場合、アプリケーションはエラー・コードを受信しません。このため、アプリケーションでAFTER SUSPENDトリガーを使用するか、またはDBAが一時停止された文を定期的に確認する必要があります。

問題が(通常DBAによって)修正されると、一時停止された文は自動的に実行を再開します。タイムアウト周期が終了するまでに問題が修正されなかった場合は、その文によってSERVERERROR例外が発生します。

トピック:

- [アプリケーションのAFTER SUSPENDトリガーの使用](#)
- [一時停止された文の確認](#)

8.9.2.1 アプリケーションのAFTER SUSPENDトリガーの使用

アプリケーションでは、AFTER SUSPENDトリガーは、DBMS_RESUMABLEパッケージ内のサブプログラムを起動することで問題に関する情報を取得できます。次にトリガーはオペレータに電子メールなどを使用して情報を送信できます。

トリガー自体の中で領域不足エラーが発生する可能性を減らすには、トリガーを自律型トランザクションとして宣言します。自律型トランザクションとして、トリガーはSYSTEM表領域内のロールバック・セグメントを使用します。一時停止された文が保持するロックによってトリガーにデッドロック状態が発生した場合、そのトリガーは終了され、アプリケーションは文の一時停止が発生しなかった場合の本来のエラー状態を受信します。トリガーによって領域不足状態が発生した場合、そのトリガーおよび一時停止された文は両方ともロールバックされます。ロールバックを回避するには、トリガー内の例外ハンドラを使用し、文が再開されるまで待ちます。

[例8-4](#)のトリガーは、データベース内の記憶域エラーを処理します。いくつかのエラーでは、このトリガーは文を終了し、エラーに関する警告を電子メールを使用してDBAに通知します。他の一時的なエラーでは、8時間以内に記憶域の問題が解決していることを想定して、8時間後に文を再開するように指定しています。この例を実行するには、SYSDBAとしてデータベースに接続する必要があります。

関連項目:

- [Oracle Database PL/SQLパッケージおよびタイプ・リファレンス](#)
- [Oracle Database PL/SQL言語リファレンス](#)

例8-4 AFTER SUSPENDトリガーによる一時停止された記憶域割当ての処理

```
-- Create table used by trigger body

DROP TABLE rbs_error;
CREATE TABLE rbs_error (
  SQL_TEXT VARCHAR2(64),
```

```

ERROR_MSG VARCHAR2(64),
SUSPEND_TIME VARCHAR2(64)
);

-- Resumable Storage Allocation
CREATE OR REPLACE TRIGGER suspend_example
AFTER SUSPEND
ON DATABASE
DECLARE
  cur_sid          NUMBER;
  cur_inst         NUMBER;
  err_type         VARCHAR2(64);
  object_owner     VARCHAR2(64);
  object_type      VARCHAR2(64);
  table_space_name VARCHAR2(64);
  object_name      VARCHAR2(64);
  sub_object_name  VARCHAR2(64);
  msg_body         VARCHAR2(64);
  ret_value        BOOLEAN;
  error_txt        VARCHAR2(64);
  mail_conn        UTL_SMTP.CONNECTION;
BEGIN
  SELECT DISTINCT(SID) INTO cur_sid FROM V$MYSTAT;
  cur_inst := USERENV('instance');
  ret_value := DBMS_RESUMABLE.SPACE_ERROR_INFO
    (err_type,
     object_owner,
     object_type,
     table_space_name,
     object_name,
     sub_object_name);
  IF object_type = 'ROLLBACK SEGMENT' THEN
    INSERT INTO rbs_error
      (SELECT SQL_TEXT, ERROR_MSG, SUSPEND_TIME
       FROM DBA_RESUMABLE
       WHERE SESSION_ID = cur_sid
        AND INSTANCE_ID = cur_inst);
    SELECT ERROR_MSG INTO error_txt
    FROM DBA_RESUMABLE
    WHERE SESSION_ID = cur_sid
    AND INSTANCE_ID = cur_inst;
    msg_body :=
      'Space error occurred: Space limit reached for rollback segment '
      || object_name || ' on ' || to_char(SYSDATE, 'Month dd, YYYY, HH:MIam')
      || '. Error message was: ' || error_txt;
    mail_conn := UTL_SMTP.OPEN_CONNECTION('localhost', 25);
    UTL_SMTP.HELO(mail_conn, 'localhost');
    UTL_SMTP.MAIL(mail_conn, 'sender@localhost');
    UTL_SMTP.RCPT(mail_conn, 'recipient@localhost');
    UTL_SMTP.DATA(mail_conn, msg_body);
    UTL_SMTP.QUIT(mail_conn);
    DBMS_RESUMABLE.ABORT(cur_sid);
  ELSE
    DBMS_RESUMABLE.SET_TIMEOUT(3600*8);
  END IF;
  COMMIT;
END;
/

```

8.9.2.2 一時停止された文の確認

アプリケーションでAFTER SUSPENDトリガーが使用されない場合、DBAは静的データ・ディクショナリ・ビュー DBA_RESUMABLEを使用して、一時停止された文を定期的に確認する必要があります。

DBAは、動的のパフォーマンス・ビューV\$_SESSION_WAITから追加情報を入手できます。

関連項目:

- [DBA_RESUMABLE](#)
- [V\\$_SESSION_WAIT](#)

9 データベース・アプリケーションにおけるSQLデータ型の使用

この章では、データベース・アプリケーション用として作成するデータベース列に対して正しいSQLデータ型を選択する方法について説明します。

トピック:

- [正確かつ最も固有のデータ型の使用](#)
- [文字データの表現](#)
- [数値データの表現](#)
- [日時データの表現](#)
- [特殊なデータの表現](#)
- [アドレスによる行の識別](#)
- [SQL演算子と関クションのメタデータの表示](#)

ノート:



Oracle プリコンパイラでは、埋込み SQL プログラム内の SQL および PL/SQL データ型以外のデータ型が認識されます。これらの**外部データ型**は、ホスト変数に関連付けられています。

関連項目:

- データ型変換の詳細は、『Oracle Database SQL言語リファレンス』を参照してください
- [PL/SQLのデータ型](#)
- [データ型](#)
- [プリコンパイラの概要](#)

9.1 正確かつ最も固有のデータ型の使用

データベース・アプリケーション用として作成するデータベース列ごとに正確かつ最も固有のデータ型を使用することにより、データの整合性が増し、記憶域要件が減り、パフォーマンスが向上します。

トピック:

- [正しいデータ型によるデータ整合性の向上](#)
- [最も固有のデータ型による記憶域要件の削減](#)
- [正しいデータ型によるパフォーマンスの向上](#)

9.1.1 正しいデータ型によるデータ整合性の向上

データ型が正しいければ、このデータ型が制約として機能するため、データの整合性が増します。たとえば、日付列に対して日付時間データ型を使用すると、この列に格納できるのは日付のみになります。ただし、この列に対して文字または数値データ型を

使用すると、結果として、日付を表さない文字または数値がいずれかのユーザーによって格納されます。この問題を阻止するコードを作成することもできますが、正しいデータ型を使用する方が効率的です。このため、文字は文字データ型、数値は数値データ型、日付および時間は日付時間データ型に格納します。

関連項目:

データ整合性および制約の詳細は、[「データベース・アプリケーションにおけるデータ整合性のメンテナンス」](#)を参照してください

9.1.2 最も固有のデータ型による記憶域要件の削減

正しいデータ型を使用する以外にも、最も固有の長さまたは精度を使用します。たとえば:

- 最大n文字の文字列を目的としたVARCHAR2列を作成する場合、VARCHAR2(n)を指定します。
- 整数を目的とした列を作成する場合、NUMBERではなくデータ型NUMBER(38)を使用します。

制約として機能することによってデータの整合性を増すのみでなく、長さおよび精度は記憶域要件にも影響します。

すべての列でそのデータ型に対して最大の長さまたは精度を指定すると、数MBのRAMが不必要に割り当てられます。たとえば、問合せによってVARCHAR2(4000)が10列が選択され、一括フェッチ操作によって100行が戻されるとします。割り当てが必要なRAMは、 $10 \times 4,000 \times 100$ で、約4MBになります。対照的に、列の長さが80である場合、割り当てが必要なRAMは、 $10 \times 80 \times 100$ で、約78KBになります。この違いは単一の問合せにとって重要であり、アプリケーションは多くの問合せを同時に処理します。このため、アプリケーションは接続ごとに4MBまたは78KBのRAMを割り当てる必要があります。

したがって、プロパティの増加が後で必要になる可能性があるという理由のみで、列のデータ型に対して最大の長さまたは精度を指定しないようにしてください。列を作成後に変更する必要がある場合は、ALTER TABLE文を使用します。たとえば、列を長くする場合は、次を使用します。

```
ALTER TABLE table_name MODIFY column_name VARCHAR2(larger_number)
```

ノート:



MAX_STRING_SIZE 初期化パラメータが EXTENDED である場合、VARCHAR2、NVARCHAR2 および RAW データ型の最大の長さは 32,767 バイトです。

関連項目:

- ALTABLEの詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。
- 拡張データ型の詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。

9.1.3 正しいデータ型によるパフォーマンスの向上

データ型が正しいと、パフォーマンスが向上します。データ型が正しくない場合、実行計画が正しくなくなる可能性があるためです。

[例9-1](#)では、同じ概念の操作(データ型が異なる3つの列に対して、日付が2000年12月31日と2001年1月1日の間にある行を選択)を実行し、問合せごとに実行計画を表示しています。これら3つの実行計画で、行(カーディナリティ)、コストおよび操作を比較します。

例9-1 3つのデータ型のパフォーマンス比較

3つの列に同じ日付を格納する表を作成します(データ型VARCHAR2のstr_date、データ型DATEのdate_date、およびデータ型NUMBERのnumber_date)。

```
CREATE TABLE t (str_date, date_date, number_date, data)
AS
SELECT TO_CHAR(dt+rownum, 'yyyymmdd')          str_date,      -- VARCHAR2
       dt+rownum                               date_date,    -- DATE
       TO_NUMBER(TO_CHAR(dt+rownum, 'yyyymmdd')) number_date, -- NUMBER
       RPAD('*',45, '*')                       data
FROM (SELECT TO_DATE('01-jan-1995', 'dd-mm-yyyy') dt
      FROM all_objects)
ORDER BY DBMS_RANDOM.VALUE
/
```

列ごとに索引を作成します。

```
CREATE INDEX t_str_date_idx ON t(str_date);
CREATE INDEX t_date_date_idx ON t(date_date);
CREATE INDEX t_number_date_idx ON t(number_date);
```

表の統計を収集します。

```
BEGIN
  DBMS_STATS.GATHER_TABLE_STATS (
    'HR',
    'T',
    method_opt => 'for all indexed columns size 254',
    cascade => TRUE
  );
END;
/
```

後続のSQL文(SQL*Plusコマンド)の実行計画を表示します。

```
SET AUTOTRACE ON EXPLAIN
```

str_dateの日付が2000年12月31日と2001年1月1日の間にある行を選択します。

```
SELECT * FROM t WHERE str_date BETWEEN '20001231' AND '20010101'
ORDER BY str_date;
```

結果および実行計画は、次のとおりです。

```
STR_DATE DATE_DATE NUMBER_DATE DATA
-----
20001231 31-DEC-00    20001231 *****
20010101 01-JAN-01    20010101 *****
```

2 rows selected.

Execution Plan

Plan hash value: 948745535

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		236	11092	216 (8)	00:00:01
1	SORT ORDER BY		236	11092	216 (8)	00:00:01
* 2	TABLE ACCESS FULL	T	236	11092	215 (8)	00:00:01

Predicate Information (identified by operation id):

```
2 - filter("STR_DATE"<='20010101' AND "STR_DATE">='20001231')
```

number_dateの日付が2000年12月31日と2001年1月1日の間にある行を選択します。

```
SELECT * FROM t WHERE number_date BETWEEN 20001231 AND 20010101;
ORDER BY str_date;
```

結果および実行計画は、次のとおりです。

```
STR_DATE DATE_DATE NUMBER_DATE DATA
-----
20001231 31-DEC-00      20001231 *****
20010101 01-JAN-01      20010101 *****

2 rows selected.
```

Execution Plan

Plan hash value: 948745535

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		234	10998	219 (10)	00:00:01
1	SORT ORDER BY		234	10998	219 (10)	00:00:01
* 2	TABLE ACCESS FULL	T	234	10998	218 (9)	00:00:01

Predicate Information (identified by operation id):

```
2 - filter("NUMBER_DATE"<=20010101 AND "NUMBER_DATE">=20001231)
```

date_dateの日付が2000年12月31日と2001年1月1日の間にある行を選択します。

```
SELECT * FROM t WHERE date_date
  BETWEEN TO_DATE('20001231','yyyymmdd')
  AND     TO_DATE('20010101','yyyymmdd');
ORDER BY str_date;
```

結果と実行計画は次のとおりです(ページにあわせてフォーマットを変更しています)。

```
STR_DATE DATE_DATE NUMBER_DATE DATA
-----
20001231 31-DEC-00      20001231 *****
20010101 01-JAN-01      20010101 *****

2 rows selected.
```

Execution Plan

Plan hash value: 2411593187

Id	Operation	Name	Rows	Bytes
1	SORT ORDER BY		1	47
2	TABLE ACCESS BY INDEX ROWID BATCHED	T	1	47
* 3	INDEX RANGE SCAN	T_DATE_DATE_IDX	1	
0	SELECT STATEMENT		1	47

```
Cost (%CPU) | Time |
4 (25) | 00:00:01 |
4 (25) | 00:00:01 |
3 (0) | 00:00:01 |
2 (0) | 00:00:01 |
```

Predicate Information (identified by operation id):

```
-----
3 - access("DATE_DATE">=TO_DATE(' 2000-12-31 00:00:00',
'syyyy-mm-dd hh24:mi:ss') AND
"DATE_DATE"<=TO_DATE(' 2001-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
```

最後の問合せのパフォーマンスが向上しました。これは、DATEデータ型の場合、2000年12月31日と2001年1月1日の間に1日のみが存在したことを確認できたためです。このため、索引範囲スキャンが実行されましたが、これは全表スキャンより高速です。

関連項目:

- [EXPLAIN PLAN文](#)
- 全表スキャンの詳細は、『[Oracle Database SQLチューニング・ガイド](#)』を参照してください。
- 索引範囲スキャンの詳細は、『[Oracle Database SQLチューニング・ガイド](#)』を参照してください。

9.2 文字データの表現

[表9-1](#) に、文字データを格納するSQLデータ型の概要を示します。

表9-1 SQL文字データ型

データ型	格納される値
CHAR	固定長文字リテラル
VARCHAR2	可変長文字リテラル
NCHAR	固定長 Unicode 文字リテラル
NVARCHAR2	可変長 Unicode 文字リテラル
CLOB	最大(4 GB - 1) * (DBMS_LOB.GETCHUNKSIZE から取得した値)のシングルバイトおよびマルチバイト文字列
NCLOB	最大(4 GB - 1) * (DBMS_LOB.GETCHUNKSIZE から取得した値)のシングルバイトおよびマルチバイト Unicode 文字列

ノート:



VARCHAR データ型は、使用しないでください。かわりに VARCHAR2 データ型を使用してください。現在、VARCHAR データ型は VARCHAR2 データ型と同じ意味で使用されていますが、VARCHAR データ型は、異なる比較セマンティクスで比較される別の可変長文字列のデータ型に変更される予定です。

CHARとVARCHAR2を選択する場合、次を考慮してください。

- 使用領域

Oracle Databaseでは、CHAR列に格納されている値には空白を埋めますが、VARCHAR2列に格納された値にはこれを行いません。そのため、VARCHAR2列の方がCHAR列よりもより効率的に領域を使用します。

- パフォーマンス

空白を埋める違いのため、VARCHAR2列のある大規模な表の全表スキャンでは、CHAR列に同じデータが格納されている表の全表スキャンより、読み込むデータ・ブロックが少なくなる可能性があります。アプリケーションが文字データを含む大規模な表に対して全表スキャンを行う場合、CHAR列よりVARCHAR2列にデータを格納することによってパフォーマンスを改善できる可能性が高くなります。

- 比較セマンティクス

比較セマンティクスでANSI互換性が必要な場合は、CHARデータ型を使用します。文字列の比較において後続空白が重要である場合は、VARCHAR2データ型を使用します。

クライアント/サーバー・アプリケーションで、クライアント側の文字セットがサーバー側と異なる場合、Oracle DatabaseはCHAR、VARCHAR2およびLONGデータをデータベース文字セット(NLS_LANGUAGEパラメータで決定)からユーザー・セッション用に定義した文字セットに変換します。

関連項目:

- これらのデータ型の比較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。
- CLOBおよびNCLOBデータ型の詳細は、[「ラージ・オブジェクト\(LOB\)」](#)を参照してください。
- LONGデータ型の詳細は、[「LONGおよびLONG RAWデータ型」](#)を参照してください。

9.3 数値データの表現

数値データを格納するSQLデータ型は、NUMBER、BINARY_FLOATおよびBINARY_DOUBLEです。

NUMBERデータ型は、実数を固定小数点形式または浮動小数点形式で格納します。NUMBERは、最大で10進数38桁の精度を提供します。NUMBER列には、 1×10^{-130} から 9.99×10^{125} の正と負の数値および0(ゼロ)を格納できます。すべてのOracle DatabaseプラットフォームはNUMBER値をサポートします。

BINARY_FLOATデータ型およびBINARY_DOUBLEデータ型は、それぞれ単精度(32ビット)IEEE 754形式、および倍精度

(64ビット)IEEE 754形式で、浮動小数点データを格納します。高精度の値は、BINARY_FLOATおよびBINARY_DOUBLEとして格納するほうが、NUMBERとして格納するよりも必要とする領域は少なくなります。浮動小数点データの算術演算は通常、BINARY_FLOATおよびBINARY_DOUBLE値のほうが、NUMBER値よりも高速です。

Oracle Databaseでサポートされているクライアント・インタフェースでは、BINARY_FLOATデータ型およびBINARY_DOUBLE値の算術演算は、ハードウェアのベンダーが提供する固有の命令セットによって実行されます。**ネイティブ浮動小数点データ型**という用語は、データ型BINARY_FLOATおよびBINARY_DOUBLE、およびサポートされているクライアント・インタフェースで実装されているこれらの型すべてを指します。

ネイティブ浮動小数点データ型は、電気電子技術者協会(IEEE)の2進浮動小数点演算についての規格であるIEEE規格754-1985(IEEE754)に実質的に準拠しています。

ノート:



NUMBER のサブタイプとして FLOAT のかわりに BINARY_FLOAT および BINARY_DOUBLE を使用することをお勧めします。

トピック:

- [浮動小数点数の構成要素](#)
- [浮動小数点数形式](#)
- [ネイティブ浮動小数点データ型の特殊な数値の表現](#)
- [ネイティブ浮動小数点値の比較](#)
- [ネイティブ浮動小数点データ型を使用した算術演算](#)
- [ネイティブ浮動小数点データ型の変換関数](#)
- [ネイティブ浮動小数点データ型のクライアント・インタフェース](#)

関連項目:

データ・タイプの詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。

9.3.1 浮動小数点数の構成要素

浮動小数点数の値は、次の式で表されます。

$$(-1)^{\text{sign}} \cdot \text{significand} \cdot \text{base}^{\text{exponent}}$$

たとえば、4.31という値は次のように表されます。

$$(-1)^0 \cdot 431 \cdot 10^{-2}$$

この表現の構成要素は、次のとおりです。

コンポーネント名	構成要素の値

コンポーネント名	構成要素の値
符号	0
仮数	431
基数	10
指数	-2

9.3.2 浮動小数点数形式

浮動小数点数形式は、浮動小数点数の構成要素の表記方法を指定し、それにより、形式が表す値の範囲および精度が決定されます。範囲とは最小値から最大値までの間隔であり、精度とは仮数の桁数です。範囲と精度のどちらも有限です。指定された形式よりも高い精度の浮動小数点数は丸められます。

数値を丸める方法は、10進または2進のいずれかが可能な、形式の基数によって決まります。10進の形式に格納された数値は、最も近い10進の桁に丸められます(たとえば、1000、10または0.01)。2進の形式に格納された数値は、最も近い2進の桁に丸められます(たとえば、1024、512または1/64)。

NUMBERの値は、10進の形式で格納されます。10進の桁への丸めを必要とする計算の場合は、NUMBERデータ型を使用してください。

ネイティブ浮動小数点の値は、2進の形式で格納されます。

[表9-2](#)に、IEEE 754の単精度と倍精度形式およびOracle Database NUMBERの範囲と精度を示します。範囲の制限は、正数として表しますが、この制限は負数の絶対値にも適用されます(ここで使用される「数値e指数」という表記法は、数値 * 10^{指数}という意味です)。

表9-2 浮動小数点データ型の範囲および精度

範囲および精度	単精度32ビット(1)	倍精度64ビット1	Oracle Database NUMBERデータ型
正の最大正規数	3.40282347e+38	1.7976931348623157e+308	< 1.0e126
正の最小正規数	1.17549435e-38	2.2250738585072014e-308	1.0e-130
正の最大非正規数	1.17549421e-38	2.2250738585072009e-308	該当なし
正の最小非正規数	1.40129846e-45	4.9406564584124654e-324	該当なし
精度(10 進数)	6 - 9	15 - 17	38 - 40

脚注1これらの数値は『IEEE Numerical Computation Guide』から引用されています。

9.3.2.1 浮動小数点2進形式

2進形式を使用する浮動小数点数の値は、次に示す算式によって決まります。

$$(-1)^{\text{sign}} 2^E (\text{bit}_0 \text{ bit}_1 \text{ bit}_2 \dots \text{bit}_{p-1})$$

表9-3に、算式の構成要素を示します。

表9-3 浮動小数点2進形式の構成要素

コンポーネント	構成要素の値
sign	0 または 1
E (指数)	単精度(32 ビット)データ型の場合、-126 以上 127 以下の整数です。 倍精度(64 ビット)データ型の場合、-1022 以上 1023 以下の整数です。
bit _i	0 または 1。(ビットの連続は基数 2 の数値を表します)。
p (精度)	単精度データ型の場合、24。 倍精度データ型の場合、53。

仮数の先頭のビットb₀は、非正規の数値(後述)の場合を除いて、(1)に設定する必要があります。したがって、先頭のビットは格納されないため、2進形式では精度としてnビットを指定しますが、格納されるのはn-1ビットになります。IEEE 754標準は、単精度および倍精度データ型のインメモリー形式を定義します。表9-4に概要を示します。

表9-4 2進形式の記憶域パラメータの概要

データ型	符号ビット	指数ビット	仮数ビット	ビット数合計
単精度	1	8	24 (23 が格納されます)	32
倍精度	1	11	53 (52 が格納されます)	64

ノート:



IEEE 754 標準で定義されている拡張単精度および拡張倍精度形式は、Oracle Database ではサポートされていません。

仮数の先頭ビットが設定されている場合、**正規化されている**と呼ばれます。IEEE 754標準では**非正規化数(subnormal numbers, denormal numbers)**という、小さすぎて正規化された仮数で表せない数値を定義します。非正規化数の仮数を正規化すると、指数が大きくなりすぎます。非正規数では、条件が $x-y=0.0$ (浮動小数点減算を使用)、結果が $x=y$ 、というプロパティが保持されます。

9.3.3 ネイティブ浮動小数点データ型の特殊な数値の表現

[表9-5](#) に、IEEE 754標準がサポートする特殊な数値を示します。

表9-5 ネイティブ浮動小数点形式の特殊な数値

値	意味
+INF	正の無限大
-INF	負の無限大
+0	正のゼロ
-0	負のゼロ
NaN	非数

[表9-5](#)に示す各値は、NaNを除き、特殊なビット・パターンで表現されます。NaNは未定義の演算の結果であり、多数のビット・パターンで表されます。ビット・パターンには、符号ビット・セットを含むものも含まないものもありますが、符号ビットに意味はありません。

IEEE 754標準では、quiet NaN(ほとんどの演算への伝播時に追加の例外が発生しない)とsignalling NaN(発生する)が区別されます。IEEE 754標準では、例外が有効および無効の場合の動作が指定されています。

Oracle Databaseでは、例外の有効化は許可されません。Oracle Databaseの動作は、IEEE 754標準で指定されている、例外が無効化されている場合の動作になります。特に、Oracle Databaseではquietとsignaling NaNは区別されません。Oracle Call Interface(OCI)を使用して、Oracle DatabaseからNaN値を取得できますが、取得されるNaN値がsignallingかquietかは、クライアント・プラットフォームによって決まり、Oracle Databaseの制御外です。

IEEE 754標準は次の特殊な値のクラスを定義します。

- ゼロ
- 非正規
- 標準
- 無限大
- NaN

NaNを除き、このリストの各クラスの値は、小さい方から順に示されています(符号は考慮していません)。NaNは、他の特殊な値のクラスともNaN自体とも順序付けることはできません。

Oracle Databaseの場合

- NaNはすべてquietです。
- すべての非NaN value < NaN
- 任意のNaN == 他のNaN
- NaNはすべて同じビット・パターンに変換されます。

- -0は+0に変換されます。
- IEEE 754例外は発生しません。

関連項目:

浮動小数点条件の詳細は、『Oracle Database SQL言語リファレンス』を参照してください(浮動小数点条件では、式が無
限か、または演算の未定義の結果(非数値(NaN))かを判断します)。

9.3.4 ネイティブ浮動小数点値の比較

数式を比較する場合、Oracle Databaseは数値の優先順位を使用して、その条件がNUMBER、BINARY_FLOATまたは
BINARY_DOUBLEのうちどの値を比較するかを判断します。

比較では、ゼロの符号は無視されます(-0は、+0と等しいとみなされます)。

関連項目:

- 数値の優先度の詳細は、『[Oracle Database SQL言語リファレンス](#)』を参照してください。
- 比較条件の詳細は、『[Oracle Database SQL言語リファレンス](#)』を参照してください。

9.3.5 ネイティブ浮動小数点データ型を使用した算術演算

IEEE 754では、浮動小数点算術に厳密な再現性は求められません。このため、演算の結果を、演算のオペランドが使用す
るよりも大きい範囲を使用する格納先に格納できます。

倍精度乗算の結果は、拡張倍精度格納先での計算に使用できます。しかし、計算結果は、格納先が単精度または倍精度
であるかのように丸められます。結果の範囲(指数のビット数)としては、より高精度(拡張倍精度)の格納先でサポートされてい
る範囲が使用できます。ただし、二重丸め誤差(結果の最下位ビットが不正)が発生する可能性があります。

この問題は、IA-32およびIA-64命令セット・アーキテクチャを実装するハードウェアで、倍精度乗算および除算を実行した場
合のみ発生します。したがって、この例を除き、これらのデータ型に対する算術をプラットフォームを渡って再現できます。計算の
結果がNaNの場合、すべてのプラットフォームでIS_NANがTRUEの値が作成されます。ただし、すべてのプラットフォームで同じ
ビット・パターンを使用する必要はありません。

関連項目:

算術演算の詳細は、『Oracle Database SQL言語リファレンス』を参照してください

9.3.6 ネイティブ浮動小数点データ型の変換関数

Oracle Databaseでは、浮動小数点と他のデータ型(10進数精度を使用する文字列形式など)の間で変換を行う関数が
定義されています(変換中に精度が失われることがあります)。たとえば:

- [TO_BINARY_DOUBLE](#)(『[Oracle Database SQL言語リファレンス](#)』を参照)
- [TO_BINARY_FLOAT](#)(『[Oracle Database SQL言語リファレンス](#)』を参照)
- [TO_CHAR](#)(『[Oracle Database SQL言語リファレンス](#)』を参照)

- [TO_NUMBER](#)(『Oracle Database SQL言語リファレンス』を参照)

Oracle Databaseでは、変換時に例外が発生する可能性があります。IEEE 754標準では、次の例外が定義されています。

- 無効
- 不正確
- ゼロによる除算
- アンダーフロー
- オーバーフロー

ただし、Oracle Databaseでは、ネイティブ浮動小数点データ型に対して、これらの例外は発生しません。例外が発生する操作では、通常、[表9-6](#)に示す値が生成されます。

表9-6 例外により生成される値

例外	値
アンダーフロー	0
オーバーフロー	- INF、+INF
無効な演算	NaN
ゼロによる除算	- INF、+INF、NaN
不正確	任意の値 – 端数処理が実行されました。

9.3.7 ネイティブ浮動小数点データ型のクライアント・インタフェース

Oracle Databaseは、ネイティブ浮動小数点データ型を次のクライアント・インタフェースでサポートします。

- SQLおよびPL/SQL

BINARY_FLOATおよびBINARY_DOUBLEのサポートには、SQL文CREATE TYPEを使用して作成する抽象データ型(ADT)の属性としての使用が含まれます(『Oracle Database PL/SQL言語リファレンス』を参照)。

- Oracle Call Interface(OCI)

OCIを使用したBINARY_FLOATおよびBINARY_DOUBLEの使用の詳細は、『Oracle Call Interfaceプログラマーズ・ガイド』を参照してください。

- Oracle C++ Call Interface(OCCI)

OCCIを使用したBINARY_FLOATの使用の詳細は、『Oracle C++ Call Interfaceプログラマーズ・ガイド』を参照してください。

OCCIを使用したBINARY_DOUBLEの使用の詳細は、『Oracle C++ Call Interfaceプログラマーズ・ガイド』を参照してください。

- Pro*C/C++プリコンパイラ

BINARY_FLOATおよびBINARY_DOUBLEを使用するには、アプリケーションのコンパイル時に、Pro*C/C++プリコンパイラ・コマンドライン・オプションNATIVE_TYPESをYESに設定します。NATIVE_TYPESオプションの詳細は、『Pro*C/C++プログラマーズ・ガイド』を参照してください。

- Oracle JDBC

Oracle JDBCを使用したBINARY_FLOATおよびBINARY_DOUBLEの使用の詳細は、『Oracle Database JDBC開発者ガイド』を参照してください。

9.4 日時データの表現

Oracle Databaseは、世紀、年、月、日、時、分、秒、および必要に応じて小数秒とタイムゾーンを表すバイナリ形式でDATEおよびTIMESTAMP (日時)データを格納します。

[表9-7](#)に、SQL日時データ型の概要を示します。

表9-7 SQL日時データ型

日付タイプ	使用方法
DATE	日時の値を表に格納します。たとえば、ジョブの日付に使用します。
TIMESTAMP	小数秒を含む正確な日付の値を格納します。たとえば、発生順序を決定するために比較する必要があるイベントの時間に使用します。
TIME WITH TIME ZONE	複数の地域にまたがって収集または調整する必要がある日時の値を格納します。
TIMESTAMP WITH LOCAL TIME ZONE	タイムゾーンが重要でない場合に日時の値を格納します。たとえば、テレビ会議のスケジュールを立てるアプリケーションでは、参加者は自分のタイムゾーンで開始時間および終了時間を確認します。 クライアント・システムのタイムゾーンを使用して日時を表示する必要がある 2 層アプリケーションに適しています。通常、3 層アプリケーションでこのデータ型を使用するのは不適切です。これは、Web ブラウザに表示されるデータがブラウザのタイムゾーンではなく Web サーバーのタイムゾーンによって書式化されるためです。Web サーバーはデータベース・クライアントであるため、ローカル時間が使用されます。
INTERVAL YEAR TO MONTH	年および月のみが重要な場合に、2 つの日時の値の違いを格納します。たとえば、アラームを 18 か月後の日付に設定したり、特定の日付から 6 か月経過したことを確認できます。
INTERVAL DAY TO SECOND	2 つの日時の値の正確な違いを格納します。たとえば、アラームを 36 時間後に設定したり、レースの開始から終了までの時間を記録できます。長い時間を高

精度で表すには、日の部分に大きな値を使用します。

関連項目:

- Oracle Database内部日時タイプの詳細は、『[Oracle Call Interfaceプログラマーズ・ガイド](#)』を参照してください。
- 日時データ型の詳細は、『[Oracle Database SQL言語リファレンス](#)』を参照してください。

トピック:

- [現在の日付および時刻の表示](#)
- [日付の挿入および表示](#)
- [時刻の挿入および表示](#)
- [日時データ型を使用した算術演算](#)
- [日時データ型の変換関数](#)
- [日時データ型のインポート、エクスポートおよび比較](#)

9.4.1 現在の日付および時刻の表示

現在の日付および時刻を表示する最も簡単な方法は次のとおりです。

```
SELECT TO_CHAR(SYSDATE, format_model) FROM DUAL
```

デフォルトの書式モデルは、初期化パラメータNLS_DATE_FORMATに依存します。

Oracle Databaseのデフォルトの標準日付書式はDD-MON-RRです。RR日時書式要素では、年の下2桁のみを指定することにより、20世紀の日付を21世紀に格納できます。たとえば、DD-MON-YY日付書式の13-NOV-54は、1950年から2049年に発行された問合せでは1954年を指しますが、2050から2149年に発行される問合せでは2054年を指します。

ノート:

プログラムの正確さを維持し、SQL インジェクションおよび動的 SQL に関する問題を回避するために、すべての日時値に対して書式モデルを指定することをお勧めします。

書式モデルを使用して現在の日付および時刻を表示する最も簡単な方法は次のとおりです。

```
SELECT TO_CHAR(SYSDATE, format_model) FROM DUAL
```

[例9-2](#)では、TO_CHARを書式モデルを指定して使用して、SYSDATEをBCまたはAD修飾子付きの書式で表示します。(デフォルトではSYSDATEは修飾子なしで表示されます)。

例9-2 現在の日付および時刻の表示

```
SELECT TO_CHAR(SYSDATE, 'DD-MON-YYYY BC') NOW FROM DUAL;
```

結果:

NOW

18-MAR-2009 AD

1 row selected.

ヒント:



SYSDATE を使用するコードをテストするときに、SYSDATE を定数に設定できるのは便利です。これは、初期化パラメータ FIXED_DATE を使用して実行します。

関連項目:

- SYSDATEの詳細は、『[Oracle Database SQL言語リファレンス](#)』を参照してください。
- NLS_DATE_FORMAT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。
- TO_CHARの詳細は、『[Oracle Database SQL言語リファレンス](#)』を参照してください。
- 日時書式モデルの詳細は、『[Oracle Database SQL言語リファレンス](#)』を参照してください。
- RR日時書式要素の詳細は、『[Oracle Database SQL言語リファレンス](#)』を参照してください。
- FIXED_DATEの詳細は、『[Oracle Databaseリファレンス](#)』を参照してください。

9.4.2 日付の挿入および表示

日付を表示および挿入する場合、日時書式モデルとともにTO_CHARおよびTO_DATEファンクションをそれぞれ使用することをお勧めします。

[例9-3](#)では、DATE列を含む表を作成し、書式モデルを指定して日付を挿入します。その後、書式モデルを指定する場合と指定しない場合で日付を表示します。

例9-3 日付の挿入および表示

表を作成します。

```
DROP TABLE dates;  
CREATE TABLE dates (d DATE);
```

書式モデルを指定して、指定した日付を表に挿入します。

```
INSERT INTO dates VALUES (TO_DATE('OCT 27, 1998', 'MON DD, YYYY'));
```

書式モデルを指定しないで日付を表示します。

```
SELECT d FROM dates;
```

結果:

```
D  
-----  
27-OCT-98
```



```
1 row selected.
```

書式モデルを指定して日付を表示します。

```
SELECT TO_CHAR(d, 'YYYY-MON-DD') D FROM dates;
```

結果:

```
D
-----
1998-OCT-27
1 row selected.
```

注意:



YY 日時書式要素を使用して現在の世紀の年を表示する場合は注意します。たとえば、21 世紀では、書式 DD-MON-YY、31-DEC-92 は 2092 年の 12 月 31 日です(推察のとおり、1992 年 12 月 31 日ではありません)。年の末尾 2 桁のみを指定して 20 世紀の日付を 21 世紀に格納するには、RR 日時書式要素を使用します(デフォルト)。

関連項目:

- NLS_DATE_FORMAT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。
- TO_CHARの詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。
- TO_DATEの詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。
- 日時書式モデルの詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。
- RR日時書式要素の詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。

9.4.3 時刻の挿入および表示

時刻を表示および挿入する場合、日時書式モデルとともにTO_CHARおよびTO_DATEファンクションをそれぞれ使用することをお勧めします。

DATE列では

- デフォルトの時刻は12:00:00 A.M.(真夜中)です。
デフォルトの時刻は、時刻部分に何も指定されなかった、または値が切捨てられたという理由のどちらの場合でも、時刻部分のない列の値に適用されます。
- デフォルトの日は現在の月の最初の日です。
デフォルトの日付は、日付部分に何も指定されなかった場合、列の値に適用されます。

[例9-4](#)では、DATE列を含む表を作成し、日付ごとに異なる書式モデルを指定して3つの日付を挿入します。最初の書式モデルには日付および時刻部分の両方があり、2番目には時刻部分がなく、3番目には日付部分がありません。次に、日付および時刻部分の両方が含まれる書式モデルを指定して3つの日付を表示します。

例9-4 日付と時刻の挿入および表示

表を作成します。

```
DROP TABLE birthdays;
CREATE TABLE birthdays (name VARCHAR2(20), day DATE);
```

日付ごとに異なる書式モデルを指定して3つの日付を挿入します。

```
INSERT INTO birthdays (name, day)
VALUES ('Annie',
       TO_DATE('13-NOV-92 10:56 A.M.', 'DD-MON-RR HH:MI A.M. '))
);

INSERT INTO birthdays (name, day)
VALUES ('Bobby',
       TO_DATE('15-APR-02', 'DD-MON-RR'))
);

INSERT INTO birthdays (name, day)
VALUES ('Cindy',
       TO_DATE('8:25 P.M.', 'HH:MI A.M. '))
);
```

格納された日時の値の日付および時刻部分の両方を表示します。

```
SELECT name,
       TO_CHAR(day, 'Mon DD, RRRR') DAY,
       TO_CHAR(day, 'HH:MI A.M.') TIME
FROM birthdays;
```

結果:

NAME	DAY	TIME
Annie	Nov 13, 1992	10:56 A.M.
Bobby	Apr 05, 2002	12:00 A.M.
Cindy	Nov 01, 2010	08:25 P.M.

3 rows selected.

9.4.4 日時データ型を使用した算術演算

日時値に対する算術演算の結果は、『Oracle Database SQL言語リファレンス』に記載のルールによって決定されます。

SQLには、日時の式に使用できる多くの日時関数があります。たとえば、関数ADD_MONTHSは、指定した日付から指定した月数である日付を戻します。

関連項目:

- 日時関数をすべて示すリストは、[『Oracle Database SQL言語リファレンス』](#)を参照してください。
- [Oracle Database SQL言語リファレンス](#)

9.4.5 日時データ型の変換関数

[表9-8](#)に、日時データ型間で変換を行うSQL関数の概要を示します。これらの関数の詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

表9-8 日時データ型のSQL変換関数

関数	変換元	変換先
<u>NUMTODSINTERVAL</u>	NUMBER	INTERVAL DAY TO SECOND
<u>NUMTOYMINTERVAL</u>	NUMBER	INTERVAL DAY TO MONTH
<u>TO_CHAR</u>	DATE	VARCHAR2
	TIMESTAMP	
	TIME WITH TIME ZONE	
	TIMESTAMP WITH LOCAL TIME ZONE	
	INTERVAL DAY TO SECOND	
	INTERVAL YEAR TO MONTH	
<u>TO_DATE</u>	CHAR	DATE
	VARCHAR2	
	NCHAR	
	NVARCHAR2	
<u>TO_DSINTERVAL</u>	CHAR	INTERVAL DAY TO SECOND
	VARCHAR2	
	NCHAR	
	NVARCHAR2	
<u>TO_TIMESTAMP</u>	CHAR	TIMESTAMP
	VARCHAR2	
	NCHAR	
	NVARCHAR2	
<u>TO_TIMESTAMP_TZ</u>	CHAR	TIME WITH TIME ZONE
	VARCHAR2	
	NCHAR	
	NVARCHAR2	

関数	変換元	変換先
TO_YMINTERVAL	CHAR	INTERVAL DAY TO MONTH
	VARCHAR2	
	NCHAR	
	NVARCHAR2	

9.4.6 日時データ型のインポートおよびエクスポート

TIMESTAMP WITH TIME ZONEおよび TIMESTAMP WITH LOCAL TIME ZONE値はデータベースに正規化された書式で格納されるため、タイムゾーンのオフセットを考慮することなく、インポート、エクスポートおよび比較ができます。

DATEおよびTIMESTAMP値はデータベースでタイムゾーンを格納しないため、ソース・データベースとターゲット・データベース間のタイムゾーンの違いを考慮して調整する必要があります。

9.5 特殊なデータの表現

トピック:

- [空間データの表現](#)
- [大量のデータの表現](#)
- [検索可能なテキストの表現](#)
- [XMLデータの表現](#)
- [動的に型指定されたデータの表現](#)
- [ANSI、DB2およびSQL/DSデータの表現](#)

9.5.1 空間データの表現

空間データは、位置情報アプリケーション、地理情報システム(GIS)・アプリケーションおよびジオイメージング・アプリケーションによって使用されます。

関連項目:

Oracle Databaseでの空間データの表現の詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。

9.5.2 大量のデータの表現

大量のデータを表現するために、Oracle Databaseでは次のものが提供されています。

- [ラージ・オブジェクト\(LOB\)](#)
- [LONGおよびLONG RAWデータ型](#)(下位互換性のため)

9.5.2.1 ラージ・オブジェクト(LOB)

ラージ・オブジェクト(LOB)は、アプリケーションが効率的にアクセスおよび操作できるような方法で大量のデータを格納することを

目的としたデータ型です。

[表9-9](#)に、LOBの概要を示します。

表9-9 ラージ・オブジェクト(LOB)

データ型	説明:
BLOB	バイナリ・ラージ・オブジェクト すべての種類のデータがバイナリ形式で格納されます。 通常、イメージ、音声および映像などのマルチメディア・データに使用されます。
CLOB	キャラクタ・ラージ・オブジェクト 文字列データがデータベース文字セット形式で格納されます。 データベース文字セットを使用する大きい文字列またはドキュメント専用です。
NCLOB	各国語キャラクタ・ラージ・オブジェクト 文字列データが各国語文字セット形式で格納されます。 各国語文字セットを使用する大きい文字列またはドキュメントに使用します。
BFILE	外部ラージ・オブジェクト バイナリ・ファイルをホストのオペレーティング・システム・ファイル・システム内のデータベース外に格納します。アプリケーションは BFILE に対し読取り専用のアクセス権を持ちます。 イメージ・データなど、アプリケーションが操作しない静的データに使用されます。 あらゆる種類のデータ(つまりオペレーティング・システム・ファイル)は BFILE に格納できます。たとえば、キャラクタ・データを BFILE に格納し、文字セットを指定して BFILE データを CLOB にロードします。

BLOB、CLOBまたはNCLOB型のインスタンスは一時的(アプリケーションの範囲内で宣言される)か、または永続的(データベースで作成および格納される)のいずれも可能です。

関連項目:

- アプリケーション開発でのLOBの使用の詳細は、[『Oracle Database SecureFilesおよびラージ・オブジェクト開発者ガイド』](#)を参照してください。
- LOB関数の詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。

9.5.2.2 LONGおよびLONG RAWデータ型

ノート:



Oracle では LONG および LONG RAW データ型を下位互換性のためにサポートしますが、LONG 列を LOB 列に、LONG RAW 列を BLOB 列に変換することを強くお勧めします。

LONG列には最大2 GB - 1バイトの可変長文字列が格納されます。

LONG RAWデータ型(およびRAWデータ型)は、Oracle Databaseによって、異なるシステム間でデータを移動するときに明示的に変換されないデータを格納します。これらのデータ型は、2進データおよびバイト列のために用意されています。

関連項目:

データ・タイプの詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。

9.5.3 検索可能なテキストの表現

全文検索を行うには、低レベルのコードを作成するのではなく、Oracle Textを使用します。Oracle Textは、問合せアプリケーションおよびドキュメント分類アプリケーションのテキスト用の索引付け、語とテーマの検索および表示機能を提供します。また、Oracle Textを使用してXMLデータを検索することもできます。

関連項目:

Oracle Textの詳細は、[『Oracle Textアプリケーション開発者ガイド』](#)を参照してください。

9.5.4 XMLデータの表現

XML形式のファイルとして格納されている情報がある場合、またはADTをXML形式で格納する場合、Oracle提供のXMLType型を使用できます。

XMLType値を使用して、次を使用できます。

- XMLTypeメンバー・ファンクション([『Oracle XML DB開発者ガイド』](#)を参照)。
- SQL XMLファンクション([『Oracle Database SQL言語リファレンス』](#)を参照)
- PL/SQL DBMS_XMLパッケージ([『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)

関連項目:

- Oracle XML DBの詳細と、それを使用してXMLデータをデータベースに格納、生成、操作、管理および問合せする方法は、[『Oracle XML DB開発者ガイド』](#)を参照してください。
- [Oracle XML Developer's Kitプログラマーズ・ガイド](#)
- XMLTypeの詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。

9.5.5 動的に型指定されたデータの表現

いくつかの言語では、実行時にデータ型を変更したり、プログラムに変数の型を確認させることができます。たとえば、Cにはunionキーワードおよびvoid*ポインタがあり、Javaにはtypeof演算子およびNumberなどのラッパー型があります。

Oracle Databaseでは、すべての型のデータを保持できる変数および列を作成することができ、その値をテストして基礎となる表現を判断できます。たとえば、表の単一の列で、数値、文字列およびオブジェクトをそれぞれ別の行に表現できます。

Oracleが提供するSYS.ANYDATA ADTを使用すると、すべてのスカラー型またはADTの値を表現できます。SYS.ANYDATAには、任意の型のスカラー値を受け取り、スカラーまたはオブジェクトに戻すメソッドがあります。同様に、Oracleが提供するSYS.ANYDATASET ADTを使用すると、すべてのコレクション型の値を表現できます。

型の情報を確認および操作するには、[例9-5](#)に示すDBMS_TYPESパッケージを使用します。

OCIを使用するときは、OCIAnyDataおよびOCIAnyDataSetインタフェースを使用します。

例9-5 SYS.ANYDATA列の情報へのアクセス

```
CREATE OR REPLACE TYPE employee_type AS
  OBJECT (empno NUMBER, ename VARCHAR2(10));
/

DROP TABLE mytab;
CREATE TABLE mytab (id NUMBER, data SYS.ANYDATA);

INSERT INTO mytab (id, data)
VALUES (1, SYS.ANYDATA.ConvertNumber(5));

INSERT INTO mytab (id, data)
VALUES (2, SYS.ANYDATA.ConvertObject(Employee_type(5555, 'john')));

CREATE OR REPLACE PROCEDURE p IS
  CURSOR cur IS SELECT id, data FROM mytab;
  v_id          mytab.id%TYPE;
  v_data        mytab.data%TYPE;
  v_type        SYS.ANYTYPE;
  v_typecode    PLS_INTEGER;
  v_typedname   VARCHAR2(60);
  v_dummy       PLS_INTEGER;
  v_n           NUMBER;
  v_employee    employee_type;
  non_null_anytype_for_NUMBER exception;
  unknown_typedname exception;
BEGIN
  FOR x IN cur LOOP
    FETCH cur INTO v_id, v_data;
    EXIT WHEN cur%NOTFOUND;

    /* typecode signifies type represented by v_data.
       GetType also produces a value of type SYS.ANYTYPE with methods you
       can call to find precision and scale of a number, length of a
       string, and so on. */

    v_typecode := v_data.GetType (v_type /* OUT */);

    /* Compare typecode to DBMS_TYPES constants to determine type of data
       and decide how to display it. */

    CASE v_typecode
      WHEN DBMS_TYPES.TYPECODE_NUMBER THEN
        IF v_type IS NOT NULL THEN -- This condition should never happen.
          RAISE non_null_anytype_for_NUMBER;
        END IF;
```



```

-- For each type, there is a Get method.
v_dummy := v_data.GetNUMBER (v_n /* OUT */);
DBMS_OUTPUT.PUT_LINE
  (TO_CHAR(v_id) || ': NUMBER = ' || TO_CHAR(v_n) );

  WHEN DBMS_TYPES.TYPECODE_OBJECT THEN
    v_typeName := v_data.GetTypeName();
    IF v_typeName NOT IN ('HR.EMPLOYEE_TYPE') THEN
      RAISE unknown_typeName;
    END IF;
    v_dummy := v_data.GetObject (v_employee /* OUT */);
    DBMS_OUTPUT.PUT_LINE
      (TO_CHAR(v_id) || ': user-defined type = ' || v_typeName ||
       ' ( ' || v_employee.empno || ', ' || v_employee.ename || ' )' );
  END CASE;
END LOOP;
EXCEPTION
  WHEN non_null_anytype_for_NUMBER THEN
    RAISE_Application_Error (-20000,
      'Paradox: the return AnyType instance FROM GetType ' ||
      'should be NULL for all but user-defined types');
  WHEN unknown_typeName THEN
    RAISE_Application_Error( -20000, 'Unknown user-defined type ' ||
      v_typeName || ' - program written to handle only HR.EMPLOYEE_TYPE');
END;
/

SELECT t.data.gettypeName() AS "Type Name" FROM mytab t;

```

結果:

```

Type Name
-----
SYS.NUMBER
HR.EMPLOYEE_TYPE

2 rows selected.

```

関連項目:

- これらのADTの詳細は、[『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。
- OCIAnyDataおよびOCIAnyDataSetインタフェースの詳細は、[『Oracle Call Interfaceプログラマーズ・ガイド』](#)を参照してください。
- DBMS_TYPESパッケージの詳細は、[『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』](#)を参照してください。

9.5.6 ANSI、DB2およびSQL/DSデータの表現

表およびクスタを作成するSQL文は、ANSIデータ型と、IBM社の製品SQL/DSおよびDB2のデータ型を使用できます(次の段落の注意を除く)。Oracle DatabaseはANSIまたはIBMデータ型に対応するOracleデータ型に変換し、Oracleデータ型を列データ型の名前として記録し、列データをOracleデータ型で格納します。



ノート:

SQL 文では SQL/DS および DB2 の TIME、GRAPHIC、VARGRAPHIC および LONG VARGRAPHIC データ型を使用できません。対応する Oracle データ型がないからです。

関連項目:

変換の詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。

9.6 アドレスによる行の識別

データベース表の行にアクセスするには、そのアドレス、またはROWIDを使用するのが最も高速な方法です。行が1データ・ブロックよりも長い場合、ROWIDによって最初の行断片が識別されます。

ROWIDを表示するには、ROWID疑似列を問い合わせます。疑似列ROWIDの各値は、行のアドレスを表す文字列です。文字列のデータ型はROWIDまたはUROWIDです。

ノート:



行の ROWID は、様々な理由で変更される可能性があります。それらは、ユーザーが引き起こすか、データベース・エンジンによって内部的に引き起こされる可能性があります。このような操作が発生した後は、ROWID が同じ行や有効な行を指していない可能性があります。

関連項目:

- 疑似列ROWIDの概要は、『Oracle Database概要』を参照してください
- ROWIDデータ型の概要は、『Oracle Database概要』を参照してください
- ROWID疑似列の詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。
- ROWIDデータ型の詳細は、『Oracle Database SQL言語リファレンス』を参照してください
- UROWIDデータ型の詳細は、『Oracle Database SQL言語リファレンス』を参照してください
- CでのROWIDデータ型の使用方法の詳細は、『Oracle Call Interfaceプログラマーズ・ガイド』を参照してください
- Pro*C/C++プリコンパイラでのROWIDデータ型の使用方法の詳細は、『Pro*C/C++プログラマーズ・ガイド』を参照してください
- JavaでのROWIDデータ型の使用方法の詳細は、『Oracle Database JDBC開発者ガイド』を参照してください
- HCCの詳細は、[『Oracle Database概要』](#)を参照してください。

9.7 SQL演算子と関クションのメタデータの表示

動的パフォーマンス・ビューV\$SQLFN_METADATAには、SQL演算子と関クションのメタデータが表示されます。

V\$SQLFN_METADATAによって表示されるすべての関クションに対し、動的パフォーマンス・ビュー

V\$SQLFN_ARG_METADATAには、それぞれの関クションの引数に関するメタデータの1行が含まれます。関数の引数が反復可能な場合(関数LEASTおよびGREATEST内など)、V\$SQLFN_ARG_METADATAには、それぞれの反復引数向けに1行のみが含まれます。列FUNC_IDでビューV\$SQLFN_METADATAおよびV\$SQLFN_ARG_METADATAを結合できます。

これらのビューによって、サード・パーティ・ツールは、アプリケーション・レイヤーでメタデータをメンテナンスすることなくSQLファンクションを活用できます。

トピック:

- [ARGnデータ型](#)
- [DISP_TYPEデータ型](#)
- [SQLデータ型ファミリ](#)

関連項目:

- V\$SQLFN_METADATAの詳細は、[『Oracle Databaseリファレンス』](#)を参照してください。
- V\$SQLFN_ARG_METADATAの詳細は、[『Oracle Databaseリファレンス』](#)を参照してください。

9.7.1 ARGnデータ型

ビューV\$SQLFN_METADATAの中で列DATATYPEは、関数のデータ型(つまり、関数が戻すデータ型)です。このデータ型は、Oracleデータ型、データ型ファミリ、またはARGnにできます。ARGnは、関数のn番目の引数のデータ型です。たとえば:

- MAX関数は、その最初の引数のデータ型を持つ値を戻すため、MAX関数が戻すデータ型はARG1です。
- DECODE関数は、その3番目の引数のデータ型を持つ値を戻すため、DECODE関数のデータ型は、ARG3です。

関連項目:

- [SQLデータ型ファミリ](#)
- MAX関数の詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。
- DECODE関数の詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。

9.7.2 DISP_TYPEデータ型

ビューV\$SQLFN_METADATAの中では、列DISP_TYPEが任意の式を持つ引数のデータ型です。式は、単一の値、または値と単一の値を持つSQLファンクションの組合せから構成されています。

表9-10 SQLファンクションの表示タイプ

表示タイプ	説明:	例
NORMAL	FUNC(A, B, . . .)	LEAST(A, B, C)
ARITHMETIC	A FUNC B)	A+B
PARENTHESIS	FUNC()	SYS_GUID()
RELOP	A FUNC B	A IN B

表示タイプ	説明:	例
CASE_LIKE	CASE 文または DECODE デコード	
NOPAREN	FUNC	SYSDATE

9.7.3 SQLデータ型ファミリー

通常、SQL関クションの引数には、データ型ファミリーの任意のデータ型を含めることができます。[表9-11](#)に、SQLデータ型ファミリーとそのメンバーのデータ型を示します。

表9-11 SQLデータ型ファミリー

ファミリー	データ型
STRING	● CHARACTER ● VARCHAR2 ● CLOB ● NCHAR ● NVARCHAR2 ● NCLOB ● LONG
NUMERIC	● NUMBER ● BINARY_FLOAT ● BINARY_DOUBLE
DATATYPE	● DATE ● TIMESTAMP ● TIME WITH TIME ZONE ● TIMESTAMP WITH LOCAL TIME ZONE ● INTERVAL YEAR TO MONTH ● INTERVAL DAY TO SECOND
BINARY	● BLOB ● RAW

ファミリ	データ型
	● LONG RAW

10 データベース・アプリケーションでの正規表現の使用

この章では、正規表現と、データベース・アプリケーションにおけるその使用方法について説明します。

トピック:

- [正規表現の概要](#)
- [Oracle SQLの正規表現のサポート](#)
- [Oracle SQLおよびPOSIX正規表現標準](#)
- [Oracle SQL正規表現内の演算子](#)
- [SQL文での正規表現の使用: 例](#)

関連項目:

- 多言語環境でのSQL正規表現関数の使用方法の詳細は、[『Oracle Databaseグローバル化・サポート・ガイド』](#)を参照してください。
- O'Reilly & Associates社発行、Jonathan Gennick著、『Oracle Regular Expressions Pocket Reference』
- O'Reilly & Associates社発行、Jeffrey E. F. Friedl著、『Mastering Regular Expressions』

10.1 正規表現の概要

正規表現は、メタキャラクタ(演算子または演算子に属する)および文字リテラルを使用する検索パターンを指定します([『Oracle Database SQL言語リファレンス』](#)を参照)。

検索パターンは複雑なものが可能です。たとえば、次の正規表現では、fまたはhtで始まり、次にtp、オプションでそれに続くs、最後にコロン(:)で終わる文字列に一致します。

```
(f|ht)tps?:
```

前述の例でのメタキャラクタ(演算子とも言います)はカッコ、パイプ記号(|)、疑問符(?)です。文字リテラルは、f、ht、tp、sおよびコロン(:)です。

カッコは複数のパターン要素を1つの要素にグループ化します。パイプ記号(|)は両側の要素であるfとhtの間での選択を示します。疑問符(?)は、直前の要素であるsがオプションであることを示します。したがって、前述の正規表現は次の文字列に一致します。

- http:
- https:
- ftp:
- ftps:

正規表現は、PERLやJavaのようなプログラミング言語の強力なテキスト処理コンポーネントです。たとえば、PERLスクリプトでは、ディレクトリ内の各HTMLファイルの内容を1つの文字列変数に取り込み、正規表現を使用して文字列内のURLを検索できます。多くのアプリケーション開発者がPERLを使用する理由の1つは、この堅牢なパターン・マッチング機能にあります。

10.2 Oracle SQLの正規表現のサポート

Oracle SQLは正規表現をサポートしているため、開発者はデータベースに複雑なパターン一致ロジックを実装できます。この手法は、次のような理由で有用です。

- パターン一致ロジックをデータベース内に集中化させることで、中間層アプリケーションがSQL結果セットの文字列処理に集中するのを回避できます。

たとえば、ライフサイエンス分野のユーザーは、通常、DNAとタンパク質の大規模データベースに格納されている生命情報データのパターン分析にPERLを使用しています。従来、[AG].[4]GK[ST]のようなタンパク質配列の一致検索は中間層で処理されていました。SQL正規表現機能は、処理ロジックをデータに近づけることで、より有効なソリューションを提供します。

- サーバー側の正規表現を使用して制約を規定すると、検証ロジックを複数のクライアントで複製するのを回避できます。

Oracle SQLは、[表10-1](#)で概要を示すパターン一致条件および関数の正規表現をサポートします。各パターン・マッチャーは指定したパターン(正規表現を使用して記述)を含む文字列を検索し、そこでは[表10-2](#)に示すパターン一致オプションが指定されます。関数には追加のオプション(パターンを含む文字列の検索開始位置など)があります。

表10-1 Oracle SQL パターン一致条件および関数

名前	説明:
REGEXP_LIKE	条件を問合せの WHERE 句で使用すると、特定パターンと一致する行が戻されます。 例: 次の WHERE 句では、Steven または Stephen という名前を持つ従業員が識別されます。 <pre>WHERE REGEXP_LIKE((hr.employees.first_name, '^Ste(v ph)en\$')</pre>
REGEXP_COUNT	関数では、指定した文字列での特定パターンの出現回数が戻されます。 例: 次の関数呼出しでは、文字列 'Albert Einstein' における開始位置 7 からの e(E ではない)の出現回数が戻されます。 <pre>REGEXP_COUNT('Albert Einstein', 'e', 7, 'c')</pre> (c オプションは大/小文字を区別する一致を指定するので戻される値は 1 です。)
REGEXP_INSTR	関数では、指定した文字列での特定パターンの開始位置を示す整数が戻されます。または、パターンの直後の位置を示す整数を戻すこともできます。 例: 次の関数呼出しでは、hr.employees.email 列で最初に有効な電子メール・アドレスの開始位置が戻されます。 <pre>REGEXP_INSTR(hr.employees.email, '¥w+@¥w+(¥.¥w+)+')</pre> 戻された値がゼロよりも大きい場合、列には有効な電子メール・アドレスが含

名前	説明:	
	まれています。	
REGEXP_REPLACE	関数では、指定した文字列での特定パターンの出現を、置換文字列で置き換えた結果の文字列を戻します。 例: 次の関数呼び出しでは、hr.countries.country_name 列の各文字の後に空白が 1 つ挿入されます。 <pre>REGEXP_REPLACE(hr.countries.country_name, '(.)', '¥1 ')</pre>	
REGEXP_SUBSTR	関数は、REGEXP_INSTR と似ていますが、文字列での特定パターンの開始位置を戻すかわりに、一致したサブストリング自体を戻します。 例: 次の関数呼び出しでは、x オプションによりパターン内の空白が無視されるので、'Oracle' が戻されます。 <pre>REGEXP_SUBSTR('Oracle 2010', 'O r a c l e', 1, 1, 'x')</pre>	
表10-2に、表10-1の各パターン・マッチャーで使用可能なパターン一致オプションを示します。 表10-2 条件および関数のOracle SQLパターン一致オプション		
パターン一致オプション	説明:	例
i	大/小文字区別なしの一致を指定します。	次の関数呼び出しでは、3 が戻されます。 <pre>REGEXP_COUNT('Albert Einstein', 'e', 'i')</pre>
c	大/小文字区別ありの一致を指定します。	次の関数呼び出しでは、2 が戻されます。 <pre>REGEXP_COUNT('Albert Einstein', 'e', 'c')</pre>
n	ドット演算子(.)を改行文字と一致させることができますが、これはデフォルトではありません(表 10-3 を参照)。	次の関数呼び出しでは、n オプションが指定されているので、文字列および検索パターンに一致します。 <pre>REGEXP_SUBSTR('a' CHR(10) 'd', 'a.d', 1, 1, 'n')</pre>
m	複数行モードを指定します。文字列内の改行文字で行が終了します。文字列に複数の行を含めることができます。	次の関数呼び出しでは、ac が戻されます。 <pre>REGEXP_SUBSTR('ab' CHR(10) 'ac', '^a.', 1, 2, 'm')</pre>

パターン一致オプション

説明:

例

複数行モードは POSIX 演算子の行頭アンカー(^)および行末アンカー(\$) (表 10-3 を参照)に影響を与えますが、PERL の影響を受ける演算子¥A、¥Z および¥z(表 10-5 を参照)には影響しません。

x

検索パターン内の空白文字が無視されます。デフォルトでは、空白文字は空白文字として一致します。

次の関数呼び出しでは、abcd が戻されます。
REGEXP_SUBSTR('abcd', 'a b c d', 1, 1, 'x')

関連項目:

単一行関数の詳細は、『[Oracle Database SQL言語リファレンス](#)』を参照してください。

10.3 Oracle SQLおよびPOSIX正規表現標準

Oracle SQLの正規表現の実装は、次の標準に準拠しています。

- IEEEのPortable Operating System Interface(POSIX)標準ドラフト1003.2/D11.2

Oracle SQLは、ASCII(英語)データの照合に関してPOSIX標準に定義されている正規表現演算子の構文と一致セマンティクスに正確に準拠しています。

- Unicode ConsortiumのUnicode正規表現ガイドライン

Oracle SQLでは、正規表現サポートがPOSIX標準を超えて次のように拡張されています。

- 多言語データの照合機能が拡張されています。
- POSIX標準には含まれていないが競合しない(文字クラスのショートカットや最短一致修飾子(?)など)、よく使用されるPERL正規表現演算子をサポートします。

関連項目:

- [Oracle SQL正規表現内のPOSIX演算子](#)
- [POSIX正規表現仕様](#)
- [Oracle SQLのPOSIX標準に対する多言語拡張機能](#)
- [Oracle SQLのPOSIX標準に対するPERLの影響を受ける拡張機能](#)

10.4 Oracle SQL正規表現内の演算子

Oracle SQLは正規表現内で一連の共通演算子(メタキャラクタから構成)をサポートします。

注意:



メタキャラクタの解析は、正規表現をサポートするツールによって異なります。別の環境から Oracle Database へ正規表現を移植する場合は、Oracle SQL により正規表現の構文がサポートされ、予想どおりに解析されることを確認してください。

トピック:

- [Oracle SQL正規表現内のPOSIX演算子](#)
- [Oracle SQLのPOSIX標準に対する多言語拡張機能](#)
- [Oracle SQLのPOSIX標準に対するPERLの影響を受ける拡張機能](#)

10.4.1 Oracle SQL正規表現内のPOSIX演算子

[表10-3](#)に、POSIX標準拡張正規表現(ERE)構文に定義されたPOSIX演算子の概要を示します。Oracle SQLはASCII(英語)データの照合に関し、POSIX標準に定義されたこれらの演算子の構文と一致セマンティクスに完全に準拠します。Oracle SQLとPOSIX標準でアクションが異なる場合は「説明」列に示されています。

表10-3 Oracle SQL正規表現内のPOSIX演算子

演算子構文	名前	説明:	例
.	任意の文字 ドット	データベース文字セットの任意の 1 文字に一致します。n 一致オプションが指定されている場合は、改行文字にも一致します(表 10-2 を参照)。 Linux、UNIX および Windows プラットフォームでは、改行文字はラインフィード文字(¥x0a)として認識されます。 Macintosh プラットフォームでは、改行文字はキャリッジ・リターン文字(¥x0d)として認識されます。 ノート: POSIX 標準では、この演算子は NULL および改行文字以外の任意の英字に一致します。	正規表現 a.b は、文字列 abb、acb および adb に一致しますが、acc には一致しません。
+	1 つ以上 プラス数量子	直前の部分正規表現の 1 回以上の出現に一致します(最長一致 脚注 1)。	正規表現 a+は文字列 a、aa および aaa に一致しますが、ba または ab には一致しません。
*	ゼロ以上 星型数量子	直前の部分正規表現のゼロ回以上の出現に一致します(最長一致 脚注 1)。	正規表現 ab*c は、文字列 ac、abc および abbc に一致しますが、abb または bbc には一

演算子構文	名前	説明:	例
			致しません。
?	ゼロまたは 1 つ 疑問符数量子	直前の部分正規表現のゼロまたは 1 回の出現に一致します(最長一致 脚注 1)。	正規表現 <code>ab?c</code> は、文字列 <code>abc</code> および <code>ac</code> に一致しますが、 <code>abbc</code> または <code>adc</code> には一致しません。
{m}	間隔 特定数	直前の部分正規表現の m 個の連続に一致します。	正規表現 <code>a{3}</code> は、文字列 <code>aaa</code> に一致しますが、 <code>aa</code> には一致しません。
{m, }	間隔 最小数	直前の部分正規表現の m 回以上の出現に一致します(最長一致 脚注 1)。	正規表現 <code>a{3, }</code> は、文字列 <code>aaa</code> および <code>aaaa</code> に一致しますが、 <code>aa</code> には一致しません。
{m, n}	間隔 個数範囲	直前の部分正規表現の m 回以上、n 回以下の出現に一致します(最長一致 脚注 1)。	正規表現 <code>a{3, 5}</code> は、文字列 <code>aaa</code> 、 <code>aaaa</code> および <code>aaaaa</code> に一致しますが、 <code>aa</code> または <code>aaaaaa</code> には一致しません。
[char...]	一致文字リスト	カッコで囲まれたリスト内の任意の 1 文字に一致します。リスト内では、次の演算子を除くすべての演算子はリテラルとして処理されます。 ● 範囲演算子: - ● POSIX 文字クラス: [: :] ● POSIX 照合要素: [. .] ● POSIX 文字等価クラス: [= =] ハイフン(-)は、リストの先頭または最後にある場合はリテラルで、[# - -]のように範囲正規表現にある場合は範囲終了ポイントです。右カッコ(])は、リストの先頭にある場合はリテラルとして扱われます。 ノート: POSIX 標準では、範囲には現行ロケールの言語定義による範囲の開始から終了までの照合要素がすべて含まれます。したがって、範囲はバイト値の範囲ではなく言語によるもので、範囲正規表現のセマンティクスは文字セットに依存しませ	正規表現 <code>[abc]</code> は、文字列 <code>all</code> 、 <code>bill</code> および <code>cold</code> の先頭文字に一致しますが、 <code>doll</code> のどの文字とも一致しません。

演算子構文	名前	説明:	例
		ん。Oracle Database では、言語範囲は NLS_SORT 初期化パラメータにより決定されます。	
[^char...]	非一致文字リスト	カッコで囲まれたリストにない任意の 1 文字に一致します。 文字リストの演算子と範囲については、一致文字リストの演算子の説明を参照してください。	正規表現 [^abc]def は、xdef に一致しますが、adef、bdef または cdef には一致しません。 正規表現 [^a-i]x は、jx に一致しますが、ax、fx または ix には一致しません。
[alt1 alt2]	または	代替項目のいずれかを一致させます。	正規表現 a b は、文字 a または文字 b に一致します。
(expr)	部分正規表現 グループ化	カッコ内の正規表現が 1 単位として処理されます。部分正規表現は、文字列または演算子を含む複雑な正規表現にすることができます。 後方参照の部分正規表現を参照してください。	正規表現 (abc)?def は、文字列 abcdef および def に一致しますが、abcdefg または xdef には一致しません。
¥n	後方参照	直前の n 番目の部分正規表現に一致します。n は、1 から 9 の整数です。後方参照では、直前の各部分正規表現の開きカッコから右に向かって部分正規表現がカウントされます。¥n の前に含まれている部分正規表現が n 個より少ない場合、正規表現は無効です。 後方参照では、事前に実際の文字列がわからなくても、繰返し文字列を検索できます。 REGEXP_REPLACE 関数では、Oracle SQL は正規表現パターンと置換文字列の両方の後方参照をサポートします。	正規表現 (abc def)xy¥1 は、文字列 abcxyabc および defxydef に一致しますが、abcxydef または abcxy には一致しません。 正規表現 ^(.*)¥1\$ は、同じ文字列の 2 つの連続するインスタスからなる 1 行に一致します。
¥	エスケープ文字	後続の文字がリテラルとして処理されます。 円記号(¥)を使用すると、通常はメタキャラクタとして処理される 1 文字を検索できます。二重円記号(¥¥)を使用すると、円記号リテラル自体に一致	正規表現 abc¥+def は、文字列 abc+def に一致しますが、abcdef または abccdef には一致しません。

演算子構文	名前	説明:	例
		します。	
<code>^</code>	行頭アンカー	デフォルト・モード: 文字列の先頭に一致します。 複数行モード: 脚注 2 ソース文字列の任意の行の先頭に一致します。	正規表現 <code>^def</code> は、文字列 <code>defghi</code> 内の <code>def</code> に一致しますが、文字列 <code>abcdef</code> 内の <code>def</code> には一致しません。
<code>\$</code>	行末アンカー	デフォルト・モード: 文字列の末尾に一致します。 複数行モード: 脚注 2 ソース文字列の任意の行の末尾に一致します。	正規表現 <code>def\$</code> は、文字列 <code>abcdef</code> 内の <code>def</code> に一致しますが、文字列 <code>defghi</code> 内の <code>def</code> には一致しません。
<code>[:class:]</code>	POSIX 文字クラス	指定した POSIX 文字クラスに属する任意の文字 (大文字、数字、句読点文字など)に一致します。 ノート: 英語による正規表現の場合、範囲正規表現は通常は文字クラスを示します。たとえば、 <code>[a-z]</code> は任意の小文字を示します。この表記規則は多言語環境では無効であり、指定した文字クラスの最初の文字と最後の文字がすべての言語で同一であるとはかぎりません。	正規表現 <code>[:upper:]+</code> では、1 つ以上の連続する大文字が指定され、文字列 <code>abcDEFghi</code> 内の部分文字列 <code>DEF</code> に一致しますが、部分文字列 <code>abcdefghi</code> には一致しません。
<code>[.element.]</code>	POSIX 照合要素演算子	現在のロケールに定義された照合要素を指定します。NLS_SORT 初期化パラメータはサポートされる照合要素を決定します。 この構文を使用すると、他の方法では 1 文字しか許可されない場合にマルチキャラクタ照合要素を使用できます。たとえば、照合要素 <code>ch</code> が、スペイン語(トラディショナル)などのロケールで定義されている場合、文字の順序に依存する操作で 1 文字として処理されるようにすることができます。	正規表現 <code>[.ch.]</code> では、照合要素 <code>ch</code> が指定され、文字列 <code>chabc</code> 内の <code>ch</code> に一致しますが、 <code>cdefg</code> には一致しません。 正規表現 <code>[a-[.ch.]]</code> では、 <code>a</code> から <code>ch</code> の範囲が指定されます。
<code>[=char=]</code>	POSIX 文字等価クラス	現行のロケールで指定された文字と同じ POSIX 文字等価クラスに属するすべての文字に一致します。 この構文は文字リスト内に含まれる必要があるため、必ず文字リスト用の大カッコ内にネストされます。	正規表現 <code>[[=n=]]</code> では、スペイン語ロケールで <code>n</code> と等価の文字が指定され、文字列 <code>El Niño</code> 内の <code>N</code> および <code>ñ</code> の両方に一致します。

演算子構文	名前	説明:	例
		文字等価はデータベース・ロケールに対する標準 ルールの定義によって異なります。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。	

脚注1

最長一致演算子は、照合が続く限り、できるだけ多くの出現に一致します。演算子を最短一致にするには、最短一致修飾子(?)を後に付けます([表10-5](#)を参照)。

脚注2

パターン一致オプションmで複数行モードを指定します([表10-2](#)を参照)。

10.4.2 Oracle SQLのPOSIX標準に対する多言語拡張機能

Oracle SQLのPOSIX演算子を多言語データに適用すると、POSIX標準に指定されている照合機能の範囲を超えて拡張されます。

[表10-4](#)に、各POSIX演算子に対し、POSIX標準で演算子の構文が定義されているか、Oracle SQLにより演算子のセマンティクスが多言語データの処理用に拡張されるかどうかを示します。POSIX標準は基本正規表現(BRE)および拡張正規表現(ERE)です。

表10-4 POSIX演算子と多言語演算子の関係

演算子	POSIX BRE構文	POSIXのERE構文	多言語拡張
¥	あり	あり	--
*	あり	あり	--
+	--	あり	--
?	--	あり	--
	--	あり	--
^	あり	あり	あり
\$	あり	あり	あり
·	あり	あり	あり
[]	あり	あり	あり

演算子	POSIX BRE構文	POSIXのERE構文	多言語拡張
()	あり	あり	--
{m}	あり	あり	--
{m,}	あり	あり	--
{m,n}	あり	あり	--
¥n	あり	あり	あり
[..]	あり	あり	あり
[::]	あり	あり	あり
[==]	あり	あり	あり

多言語データにはマルチバイト・キャラクタが含まれている可能性があります。Oracle Databaseでは、直接入力方法でマルチバイト・キャラクタを直接入力するか、関数を使用してマルチバイト・キャラクタを構成できます。¥xxxx形式のUnicode 16進エンコーディング値は使用できません。Oracle Databaseでは、文字はグラフィカル表現ではなくエンコーディングに使用されるバイト値に基づいて評価されます。

10.4.3 Oracle SQLのPOSIX標準に対するPERLの影響を受ける拡張機能

Oracle SQLは、POSIX標準には含まれていないが競合もしない、よく使用されるPERL正規表現演算子をサポートします。

[表10-5](#)に、Oracle SQLがサポートするPERLの影響を受ける演算子の概要を示します。

注意:



PERLの文字クラス・マッチングはオペレーティング・システムのロケール・モデルに基づいていますが、Oracle SQLの正規表現はデータベースの言語固有のデータに基づいています。通常、ロケール・データを含む正規表現の場合、PERLとOracle SQLで同じ結果が生成されることは期待できません。

表10-5 Oracle SQL正規表現でPERLの影響を受ける演算子

演算子構文	説明:	例
¥d	数字 1 文字に一致します。 POSIX 正規表現[[:digit:]]と等価です。	正規表現^¥(¥d{3}¥) ¥d{3}-¥d{4}\$は、(650) 555-0100 に一致しますが 650-555-0100 には一致しません。

演算子構文	説明:	例
¥D	数字以外の 1 文字に一致します。 POSIX 正規表現 <code>[^[:digit:]]</code> と等価です。	正規表現 <code>¥w¥d¥D</code> は、<code>b2b</code> および <code>b2_</code> に一致しますが、<code>b22</code> には一致しません。
¥W	1 文字(英数字またはアンダースコア(<code>_</code>)文字)と一致します。 POSIX 正規表現 <code>[[:alnum:]]</code> と等価です。	正規表現 <code>¥w+@¥w+(¥.¥w+)+</code> は、文字列 <code>jdoe@company.co.uk</code> に一致しますが、文字列 <code>jdoe@company</code> には一致しません。
¥W	英数字およびアンダースコア以外の 1 文字に一致します。 POSIX 正規表現 <code>[^[:alnum:]]</code> と等価です。	正規表現 <code>¥w+¥W¥s¥w+</code> は、文字列 <code>to: bill</code> に一致しますが、文字列 <code>to bill</code> には一致しません。
¥S	空白文字 1 文字に一致します。 POSIX 正規表現 <code>[[:space:]]</code> と等価です。	正規表現 <code>¥(¥w¥s¥w¥s¥)</code> は、文字列 <code>(a b)</code> に一致しますが、<code>(ab)</code> または <code>(a,b.)</code> には一致しません。
¥S	空白以外の 1 文字に一致します。 POSIX 正規表現 <code>[^[:space:]]</code> と等価です。	正規表現 <code>¥(¥w¥s¥w¥s¥)</code> は、文字列 <code>(abde)</code> および <code>(a,b.)</code> に一致しますが、<code>(a b d e)</code> には一致しません。
¥A	単一行モードまたは複数行モードのいずれでも、文字列の先頭に一致します。 POSIX 演算子 <code>^</code> と等価ではありません。	正規表現 <code>¥AL</code> は、単一行モードまたは複数行モードのいずれでも、文字列 <code>Line1¥nLine2¥n</code> (<code>¥n</code> は改行文字)内の最初の <code>L</code> 文字にのみ一致します。
¥Z	単一行モードまたは複数行モードのいずれでも、文字列の最後に一致します。 POSIX 演算子 <code>\$</code> と等価ではありません。	正規表現 <code>¥s¥Z</code> は、単一行モードまたは複数行モードのいずれでも、文字列 <code>L i n e ¥n</code> (<code>¥n</code> は改行文字)内の最後の空白に一致します。
¥Z	単一行モードまたは複数行モードのいずれでも、文字列の最後に一致します。 POSIX 演算子 <code>\$</code> と等価ではありません。	正規表現 <code>¥s¥Z</code> は、単一行モードまたは複数行モードのいずれでも、文字列 <code>L i n e ¥n</code> の改行文字(<code>¥n</code>)に一致します。

演算子構文	説明:	例
+?	直前の部分正規表現の 3 回以上の出現に一致します(最短一致 脚注 3)。	正規表現 $\text{w}^+?\text{xw}$ は、文字列 <code>abxcxd</code> の <code>abxc</code> に一致します(最長一致の正規表現 w^+xw は <code>abxcxd</code> に一致します)。
*?	直前の部分正規表現のゼロ回以上の出現に一致します(最短一致 脚注 3)。可能なかぎり空の文字列に一致します。	正規表現 w^*xw は文字列 <code>xaxbxc</code> の <code>xa</code> に一致します(最長一致の正規表現 w^*xw は <code>xaxbxc</code> に一致します)。
??	直前の部分正規表現のゼロまたは 1 回の出現に一致します(最短一致 脚注 3)。可能なかぎり空の文字列に一致します。	正規表現 $\text{a}??\text{aa}$ は文字列 <code>aaaa</code> の <code>aa</code> に一致します(最長一致の正規表現 $\text{a}?\text{aa}$ は <code>aaa</code> に一致します)。
{m}?	直前の部分正規表現の m 回の出現に一致します(最短一致 脚注 3)。	正規表現 $(\text{a} \text{aa})\{2\}?$ は文字列 <code>aaaa</code> の <code>aa</code> に一致します(最長一致の正規表現 $(\text{a} \text{aa})\{2\}$ は <code>aaaa</code> に一致します)。 正規表現 $\text{b}\{2\}?$ および最長一致の正規表現 $\text{b}\{2\}$ はどちらも文字列 <code>bbbb</code> の <code>bb</code> に一致します。
{m,}?	直前の部分正規表現の m 回以上の出現に一致します(最短一致 脚注 3)。	正規表現 $\text{a}\{2, \}?$ は文字列 <code>aaaaa</code> の <code>aa</code> に一致します(最長一致の正規表現 $\text{a}\{2, \}$ は <code>aaaaa</code> に一致します)。
{m,n}?	直前の部分正規表現の m 回以上 n 回以下の出現に一致します(最短一致 脚注 3)。{0,n}? は可能なかぎり空の文字列に一致します。	正規表現 $\text{a}\{2, 4\}?$ は文字列 <code>aaaaa</code> の <code>aa</code> に一致します(最長一致の正規表現 $\text{a}\{2, 4\}$ は <code>aaaa</code> に一致します)。

脚注3

最短一致演算子は、照合が続く限り、できるだけ少ない出現に一致します。演算子を最長一致にするには、最短一致修飾子(?)を省略します。

10.5 SQL文での正規表現の使用: 例

シナリオ:

- [制約を使用した電話番号書式の規定](#)
- [例: 正規表現を使用した電話番号書式の規定](#)
- [例: 正しい書式と誤った書式での電話番号の挿入](#)
- [後方参照を使用した文字の再配置](#)

10.5.1 制約を使用した電話番号書式の規定

正規表現は、制約を規定する際に役立つ方法です。たとえば、データベースに標準書式で電話番号を入力する必要があるとします。

表10-6は「[例：正規表現を使用した電話番号書式の規定](#)」内の正規表現の要素を説明します。

表10-6 正規表現要素の説明

正規表現要素	一致
^	文字列の先頭。
¥(	左カッコ。円記号(¥)は、直後の左カッコが部分正規表現のデリミタではなくリテラルであることを示すエスケープ文字です。
¥d{3}	数字 3 文字。
¥)	右カッコ。円記号(¥)は、直後の右カッコが部分正規表現のデリミタではなくリテラルであることを示すエスケープ文字です。
空白文字	空白文字。
¥d{3}	数字 3 文字。
-	ハイフン。
¥d{4}	数字 4 文字。
\$	文字列の終わり。

10.5.2 例：正規表現を使用した電話番号書式の規定

表を作成する際に、正規表現を使用して書式を指定できます。

[例10-1](#)では、contacts表を作成し、p_number列にCHECK制約を追加して次の書式モデルを規定しています。

(XXX) XXX-XXXX

例10-1 正規表現を使用した電話番号書式の規定

```
DROP TABLE contacts;
CREATE TABLE contacts (
  l_name    VARCHAR2(30),
  p_number  VARCHAR2(30)
  CONSTRAINT c_contacts_pnf
  CHECK (REGEXP_LIKE (p_number, '^¥(¥d{3}¥) ¥d{3}-¥d{4}$'))
);
```

10.5.3 例：正しい書式と誤った書式での電話番号の挿入

書式が正しく動作したか、正しく動作しなかったかをテストするためにINSERT INTO SQL文を使用できます。

[例10-2](#)に、電話番号をcontacts表に挿入する正しい文および誤った文を示します。

例10-2 正しい書式と誤った書式での電話番号の挿入

正しい書式は次のとおりです。

```
INSERT INTO contacts (p_number) VALUES(' (650) 555-0100');
INSERT INTO contacts (p_number) VALUES(' (215) 555-0100');
```

次のコードは、CHECK制約エラーを生成します。

```
INSERT INTO contacts (p_number) VALUES('650 555-0100');
INSERT INTO contacts (p_number) VALUES('650 555 0100');
INSERT INTO contacts (p_number) VALUES('650-555-0100');
INSERT INTO contacts (p_number) VALUES('(650)555-0100');
INSERT INTO contacts (p_number) VALUES(' (650) 555-0100');
```

10.5.4 後方参照を使用した文字の再配置

後方参照([表10-3](#)を参照)では参照する部分正規表現が一時バッファに格納されます。そのため後方参照を使用して文字を再配置できます([例10-3](#)を参照)。[例10-3](#)の正規表現の要素の説明は、[表10-7](#)を参照してください。

[表10-7](#)は[例10-3](#)内の正規表現の要素を説明します。

表10-7 正規表現要素の説明

正規表現要素	説明:
^	文字列の先頭に一致します。
\$	文字列の終わりに一致します。
(¥S+)	空白以外の 1 文字以上に一致します。カッコはエスケープされていないため、グループ化表現として機能します。
¥s	空白文字 1 文字に一致します。
¥1	最初の部分正規表現、つまり、一致パターンに含まれるカッコの第 1 グループを置き換えます。
¥2	第 2 の部分正規表現、つまり、一致パターンに含まれるカッコの第 2 グループを置き換えます。
¥3	第 3 の部分正規表現、つまり、一致パターンに含まれるカッコの第 3 グループを置き換えます。

11 データベース・アプリケーションでの索引の使用

索引は表およびクラスタに関連付けられているオプション構造で、これによりSQL問合せをより迅速に実行することができます。このマニュアルで、索引の使用により、索引がない場合よりも高速に情報を検索できるように、Oracle Databaseの索引は、表データへのより高速なアクセス・パスを提供します。索引は問合せをリライトすることなく使用できます。結果は同じですが、より高速に得られます。

関連項目:

- 索引および索引構成表の詳細は、[『Oracle Database概要』](#)を参照してください。
- 索引管理の詳細は、[『Oracle Database管理者ガイド』](#)を参照してください。
- 索引およびクラスタによってどのようにパフォーマンスが向上または低下する可能性があるかの詳細は、[『Oracle Database SQLチューニング・ガイド』](#)を参照してください。

トピック:

- [索引を管理するためのガイドライン](#)
- [索引の管理](#)
- [ドメイン索引を使用する場合](#)
- [ファンクション索引を使用する場合](#)

11.1 索引を管理するためのガイドライン

索引管理のガイドラインの概要を、次に示します。

- 表データの挿入後の索引の作成
- 正しい表および列の索引付け
- パフォーマンスのための索引列の順序付け
- 表当たりの索引数の制限
- 不要な索引の削除
- 遅延セグメント作成の理解
- 索引サイズの見積りと記憶域パラメータの設定
- 索引ごとの表領域の指定
- 索引作成の並列化の検討
- NOLOGGINGを使用する索引の作成の検討
- 使用禁止または非表示の索引をいつ使用するかを理解
- 索引の結合または再構築のコストとメリットの検討
- 制約を使用禁止または削除する前のコストの検討

関連項目:

[Oracle Database管理者ガイド](#)

11.2 索引の管理

『[Oracle Database管理者ガイド](#)』には、索引の管理に関する情報が含まれています。

- [索引の作成](#)
- [索引の変更](#)
- [索引の使用領域の監視](#)
- [索引の削除](#)
- [索引の情報を表示するデータ・ディクショナリ・ビュー](#)

関連項目:

[制約で使用する索引の作成](#)

11.3 ドメイン索引を使用する場合

ドメイン索引(アプリケーション・ドメイン索引とも呼ばれます)はデータ・カートリッジ(検索エンジンまたは地理情報システムなど)を使用して実装されたアプリケーションに特有のカスタマイズされた索引です。

関連項目:

- ドメイン索引をいつ構築するかを決定するための概念的背景については、[『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。
- ドメイン索引の詳細は、[『Oracle Database概要』](#)を参照してください。

11.4 ファンクション索引を使用する場合

ファンクション索引は1つまたは複数の列を含む式の値計算し、それを索引に格納します。索引式は、算術式、あるいはSQLファンクション、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。ファンクション索引は、照合キーに基づく言語ソート、SQL文の効率的な言語依存照合、および大/小文字を区別しないソートもサポートします。

ファンクション索引は、索引式を使用する問合せのパフォーマンスを向上させます(特に式の計算が集中型の場合)。ただし、次の点に注意してください。

- データベースは索引を使用しない文の処理でも索引式を評価する必要があります。
- 頻繁に変更される列のファンクション索引は、データベースで維持するためにコストがかかります。

オプティマイザはファンクション索引をコストベースの最適化のためだけに使用できます。一方、列の索引はコストベースの最適化およびルールベースの最適化の両方のために使用できます。

ノート:



- ファンクション索引には NULL 値を含めることはできません。そのため、列に NULL を含む可能性のある索引式が含まれていないことを確認するか、または索引式に NVL 関数を使用して NULL を他の値に置き換えてください
- Oracle Database は、降順索引をファンクション索引として扱います。

トピック:

- [ファンクション索引のメリット](#)
- [ファンクション索引のデメリット](#)
- [例: 算術式の事前計算用のファンクション索引](#)
- [例: オブジェクト列でのファンクション索引](#)
- [例: 大/小文字を区別しないより高速の検索用のファンクション索引](#)
- [例: 言語依存ソート用のファンクション索引](#)

関連項目:

- ファンクション索引のその他の概念は、『[Oracle Database概要](#)』を参照してください。
- ファンクション索引の作成の詳細は、『[Oracle Database管理者ガイド](#)』を参照してください。
- ファンクション索引の言語の詳細は、『[Oracle Databaseグローバルバージョン・サポート・ガイド](#)』を参照してください。
- オプティマイザのファンクション索引の使用方法の詳細は、『[Oracle Database概要](#)』を参照してください。
- パフォーマンスのためのファンクション索引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
- NVLの詳細は、『[Oracle Database SQL言語リファレンス](#)』を参照してください。
- 索引の作成の詳細は、『[Oracle Database SQL言語リファレンス](#)』を参照してください。

11.4.1 ファンクション索引のメリット

ファンクション索引には次のようなメリットがあります。

- ファンクション索引を使用すると、データベースが全索引スキャンではなく索引範囲スキャンを実行できる機会が増加します。

WHERE句によって大きな表の15%未満の行が選択された場合、索引範囲スキャンでは通常応答時間が短くなります。式がファンクション索引に実体化されていると、オプティマイザは式で選択される行数をより正確に見積もることができます。

Oracle Databaseでは、索引式は仮想列として表されます。ここで、ANALYZE文でヒストグラムを作成できます。

- ファンクション索引は事前計算を行い、式の値を格納します。

問合せは式の値を計算するかわりに索引から得ることができます。値を必要とする問合せが増え、索引式が計算集中

型になればなるほど、索引により、アプリケーションのパフォーマンスが向上します。

- ファンクション索引はオブジェクト列またはREF列に対して作成できます。

索引式はオブジェクト・タイプを戻すメソッドを起動することができます。

- ファンクション索引により、より強力なソートを実行できます。

索引式では、大/小文字を区別しないソートのためのSQL関数UPPERおよびLOWER ([例11-3](#)を参照)、および言語ベースのソートのためのSQL関数NLSSORTを呼び出すことができます。

関連項目:

- 索引範囲スキャンおよび索引スキャンの詳細は、『[Oracle Database概要](#)』を参照してください。
- ANALYZE文の詳細は、『[Oracle Database SQL言語リファレンス](#)』を参照してください。
- ファンクション索引の詳細は、『[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。
- ファンクション索引に関連する例は、『[例: オブジェクト列でのファンクション索引](#)』および『[Oracle Database SQL言語リファレンス](#)』を参照してください。
- [例: 算術式の事前計算用のファンクション索引](#)
- NLSSORT SQL関数に関連する例は、『[例: 言語依存ソート用のファンクション索引](#)』を参照してください。

11.4.2 ファンクション索引のデメリット

ファンクション索引には次のようなデメリットがあります。

- オプティマイザはファンクション索引をコストベースの最適化のためだけに使用できます。ルールベースの最適化には使用できません。

コストベースの最適化はディクショナリに格納された統計情報を使用します。ファンクション索引の統計情報を収集するには、DBMS_STATS.GATHER_TABLE_STATSまたはDBMS_STATS.GATHER_SCHEMA_STATSを呼び出します。

- ファンクション索引は、索引自体と索引が定義される表が分析されるまで、使用されません。

索引と索引が定義される表を分析するにはDBMS_STATS.GATHER_TABLE_STATSまたはDBMS_STATS.GATHER_SCHEMA_STATSを起動します。

- ファンクション索引は、OR拡張が実行された場合、使用されません。
- 索引式が呼び出すすべてのスキーマ・レベルまたはパッケージ・レベルPL/SQLファンクションが、DETERMINISTICである(同じ入力に対して常に同じ結果を戻す)ことを確認する必要があります。

ファンクションはDETERMINISTICとして宣言する必要がありますが、Oracle Databaseはこのアサーションをチェックしないので、ファンクションが実際にDETERMINISTICであることを確認する必要があります。

DETERMINISTICファンクションのセマンティックを変更し、再コンパイルすると、依存するファンクション索引およびマテリアライズド・ビューを手動で再作成する必要があります。それ以外の場合は、旧バージョンのファンクションの結果がレポートされます。

- 索引式がファンクション起動の場合、ファンクションの戻り型は制約付きにできません。

関数型の戻り型をNOT NULLで制約できないため、索引を使用する問合せがNULL値をフェッチできないことを保証する必要があります。そうしないと、データベースは全表スキャンを実行します。

- 索引式では、集計関数は呼び出せません。
- ビットマップされた関数索引は、降順索引にはできません。
- 索引式のデータ型は、VARCHAR2、RAW、LONGRAW、または長さが不明のPL/SQLデータ型にはできません。

つまり、長さが不明な式には索引付けできません。ただし、そのような式の既知の長さのサブstringには索引付けできます。たとえば：

```
CREATE OR REPLACE FUNCTION initials (
    name IN VARCHAR2
) RETURN VARCHAR2
    DETERMINISTIC
IS
BEGIN
    RETURN('A. J. ');
END;
/
/* Invoke SUBSTR both when creating index and when referencing
   function in queries. */

CREATE INDEX func_substr_index ON
EMPLOYEES(SUBSTR(initials(FIRST_NAME),1,10));

SELECT SUBSTR(initials(FIRST_NAME),1,10) FROM EMPLOYEES;
```

関連項目:

- ファンクション索引のノートについては、[『Oracle Database SQL言語リファレンス』](#)を参照してください。
- ファンクション索引の制限事項については、[『Oracle Database SQL言語リファレンス』](#)を参照してください。
- 制限事項を含むCREATE FUNCTION文の詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。
- DBMS_STATS、GATHER_TABLE_STATS、DBMS_STATSおよびGATHER_SCHEMA_STATS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。
- 集計関数の詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。
- ファンクション索引の詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。

11.4.3 例：算術式の事前計算用のファンクション索引

算術式を計算するために複合索引を作成できます。

[例11-1](#)は列a、bおよびcのある表を作成し、表に索引を作成し、表に問合せを実行します。索引は式 $a+b*(c-1)$ を表す仮想列、列aおよび列bの3つの列に対するコンポジット索引です。問合せはそのWHERE句に索引付けされた式を使用します。そのため、全索引スキャンのかわりに索引範囲スキャンを使用できます。

例11-1 算術式の事前計算用のファンクション索引

索引を作成する表を作成します。

```
DROP TABLE Fbi_tab;
CREATE TABLE Fbi_tab (
```

```
a INTEGER,  
b INTEGER,  
c INTEGER  
);
```

索引を作成します。

```
CREATE INDEX Idx ON Fbi_tab (a+b*(c-1), a, b);
```

この問合せは全索引スキャンのかわりに索引範囲スキャンを使用できます。

```
SELECT a FROM Fbi_tab WHERE a+b*(c-1) < 100;
```



ノート:

この例では、複合索引(複数の表列に対する索引)を使用します。

関連項目:

- 高速全索引スキャンの詳細は、[『Oracle Database概要』](#)を参照してください。
- 複合索引の詳細は、[『Oracle Database概要』](#)を参照してください。

11.4.4 例: オブジェクト列でのファンクション索引

[例11-2](#)では、オブジェクト・タイプReg_objが定義され、そこに都市に関する情報が格納されると仮定します。例では、最初の列にタイプReg_objを含む表、パラメータにタイプReg_objを指定したDETERMINISTICファンクション、ファンクションを起動する2つのファンクション索引を作成します。最初の問合せは最初の索引を使用して赤道から1000マイル以上離れた都市を迅速に検索します。2番目の問合せは2番目の索引(コンポジット索引)を使用して、気温の差が20未満で最高気温が75よりも高い都市を迅速に検索します(表は例が移入されていないので問合せで行は戻りません)。

例11-2 オブジェクト列でのファンクション索引

オブジェクト列を含む表を作成します。

```
DROP TABLE Weatherdata_tab;  
CREATE TABLE Weatherdata_tab (  
  Reg_obj INTEGER,  
  Maxtemp INTEGER,  
  Mintemp INTEGER  
);
```

パラメータにタイプReg_objを指定したDETERMINISTICファンクションを作成します。

```
CREATE OR REPLACE FUNCTION Distance_from_equator (  
  Reg_obj IN INTEGER  
) RETURN INTEGER  
  DETERMINISTIC  
IS  
BEGIN  
  RETURN(3000);  
END;  
/
```

最初のファンクション索引を作成します。

```
CREATE INDEX Distance_index
ON Weatherdata_tab (Distance_from_equator (Reg_obj));
```

問合せで索引式を使用します。

```
SELECT * FROM Weatherdata_tab
WHERE (Distance_from_equator (Reg_Obj)) > '1000';
```

結果:

```
no rows selected
```

2番目のファンクション(コンポジット)索引を作成します。

```
CREATE INDEX Compare_index
ON Weatherdata_tab ((Maxtemp - Mintemp) DESC, Maxtemp);
```

問合せで索引式および索引付けされた列を使用します。

```
SELECT * FROM Weatherdata_tab
WHERE ((Maxtemp - Mintemp) < '20' AND Maxtemp > '75');
```

結果:

```
no rows selected
```

11.4.5 例: 大/小文字を区別しないより高速の検索用のファンクション索引

[例11-3](#)では、EMPLOYEES表でより高速な大/小文字を区別しない検索ができる索引を作成してから、問合せで索引式を使用します。

例11-3 大/小文字を区別しないより高速の検索用のファンクション索引

索引を作成します。

```
CREATE INDEX emp_lastname ON EMPLOYEES (UPPER(LAST_NAME));
```

問合せで索引式を使用します。

```
SELECT first_name, last_name
FROM EMPLOYEES
WHERE UPPER(LAST_NAME) LIKE 'J%S_N';
```

結果:

FIRST_NAME	LAST_NAME
Charles	Johnson

1 row selected.

11.4.6 例: 言語依存ソート用のファンクション索引

言語依存ソート用NLSSORT APIを使用できます。

[例11-4](#)では、1つの列NAMEを含む表、照合順序GERMANを使用して列をソートするファンクション索引を作成してから、表のすべての列を選択し、NAMEにより順序付けします。問合せは索引を使用できるのでより高速です(問合せはドイツ語セッションで実行され、NLS_SORTはGERMAN、NLS_COMPはANSIであると仮定します。そうでない場合、問合せでこれらのグローバル・サポートのパラメータの値を指定する必要があります)。

例11-4 言語依存ソート用のファンクション索引

索引を作成する表を作成します。

```
DROP TABLE nls_tab;  
CREATE TABLE nls_tab (NAME VARCHAR2(80));
```

索引を作成します。

```
CREATE INDEX nls_index  
ON nls_tab (NLSSORT(NAME, 'NLS_SORT = GERMAN'));
```

表のすべての列を選択し、NAMEによって順序付けします。

```
SELECT * FROM nls_tab  
WHERE NAME IS NOT NULL  
ORDER BY NAME;
```


12 データベース・アプリケーションにおけるデータ整合性のメンテナンス

データベース・アプリケーションでは、データ整合性のメンテナンスは、アプリケーションで操作される表のデータが適切なビジネス・ルールに準拠する必要があることを意味します。ビジネス・ルールは、常に真または偽になる必要がある条件および関連を指定します。たとえば、従業員の給与が\$100,000を超えないようにするビジネス・ルールや、EMPLOYEES表のすべての従業員がDEPARTMENTS表の部門に所属する必要があるビジネス・ルールなどがあります。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ているため、企業ごとにビジネス・ルールが異なります。

データ整合性を確認する方法は複数あり、可能なかぎり、整合性制約(または制約)による方法を使用する必要があります。この章は、次の情報を補足します。

ノート:



この章は、表に対する制約のみに適用されます。ビューに対する制約では、データ整合性をメンテナンスしたり、関連する索引を持つことはできません。これらは、ビューを伴う問合せをリライトできるため、マテリアライズド・ビューおよびその他のデータ・ウェアハウス機能を使用した場合のパフォーマンスが向上します。

関連項目:

- データ整合性および制約の詳細は、[『Oracle Database概要』](#)を参照してください。
- 制約管理の詳細は、[『Oracle Database管理者ガイド』](#)を参照してください。
- 制約に関する構文および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。
- ビューに関する制約の詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。
- データ・ウェアハウスでの制約の使用の詳細は、[『Oracle Databaseデータ・ウェアハウス・ガイド』](#)を参照してください。
- データ整合性においてデータ型が担う役割の詳細は、[「正しいデータ型によるデータ整合性の向上」](#)を参照してください。

トピック:

- [制約を使用したビジネス・ルールの施行](#)
- [制約とアプリケーション・コードの両方を使用したビジネス・ルールの施行](#)
- [制約で使用する索引の作成](#)
- [NOT NULL制約を使用する場合](#)
- [デフォルトの列値を使用する場合](#)
- [表の主キーの選択\(PRIMARY KEY制約\)](#)
- [UNIQUE制約を使用する場合](#)
- [FOREIGN KEY制約を使用した参照整合性の施行](#)
- [制約に対応付けられた索引の領域と時間のオーバーヘッドの最小化](#)

- [外部キーを索引付けするためのガイドライン](#)
- [分散データベース内の参照整合性](#)
- [CHECK制約を使用する場合](#)
- [制約の定義の例](#)
- [制約の使用可能および使用禁止](#)
- [制約の変更](#)
- [制約名の変更](#)
- [制約の削除](#)
- [FOREIGN KEY制約の管理](#)
- [制約に関する情報の表示](#)

12.1 制約を使用したビジネス・ルールの施行

ビジネス・ルールは、可能なかぎり制約を使用して施行します。制約には、速度の利点があります。Oracle Databaseは、表内のすべてのデータが制約に従っていることのチェックを、アプリケーション・コードで同等のチェックを行うよりも速く実行できます。

[例12-1](#)では、部門表および従業員表を作成し、部門表のすべての値が一意であるというルールを施行する制約、および各従業員が有効な部門に所属する必要があるというルールを施行する制約を作成します。

例12-1 制約を使用したビジネス・ルールの施行

部門表を作成します。

```
DROP TABLE dept_tab;
CREATE TABLE dept_tab (
  deptname VARCHAR2(20),
  deptno    INTEGER
);
```

従業員表を作成します。

```
DROP TABLE emp_tab;
CREATE TABLE emp_tab (
  empname VARCHAR2(80),
  empno    INTEGER,
  deptno   INTEGER
);
```

部門表のすべての値が一意であるというルールを施行する制約を作成します。

```
ALTER TABLE dept_tab ADD PRIMARY KEY (deptno);
```

各従業員が有効な部門に所属する必要があるというルールを施行する制約を作成します。

```
ALTER TABLE emp_tab ADD FOREIGN KEY (deptno) REFERENCES dept_tab(deptno);
```

これで、emp_tabに従業員レコードを挿入すると、そのdeptnoの値がdept_tabに存在するかどうかOracle Databaseによって確認されます。

各従業員が有効な部門に所属する必要があるというルールを施行する制約を使用せずに、dept_tabを問い合わせるトリガーを

使用して、emp_tabに挿入する従業員レコードのdeptnoの値が含まれていることを確認するとします。問合せでは読取り一貫性(CR)が使用されるため、他のトランザクションからのコミットされていない変更を見落とす場合があります。

関連項目:

- 制約に関する構文および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。
- 整合性制約の利点をすべて示すリストは、『[Oracle Database概要](#)』を参照してください。
- トリガーの使用によるビジネス・ルールの強制の詳細は、『[Oracle Database概要](#)』を参照してください。

12.2 制約とアプリケーション・コードの両方を使用したビジネス・ルールの施行

データ値が無効であることを表を問い合わせずにアプリケーション・コードで判断できる場合は、制約とアプリケーション・コードの両方を使用してビジネス・ルールを施行することをお勧めします。アプリケーション・コードには、無効なデータを表に挿入できないようにすることで、ユーザーに即座にフィードバックして負担を減らすことができます。

[例12-2](#)では、[例12-1](#)が実行され、この列が表emp_tabに追加されたことを前提としています。

```
empgender VARCHAR2(1)
```

empgenderで有効な唯一の値は'M'および'F'です。あるユーザーが行をemp_tabに挿入またはemp_tab.empgenderの値を更新しようとした場合は、emp_tab.empgenderの新しい値が有効かどうかを、表を問い合わせずにアプリケーション・コードで判断できます。値が無効の場合は、[例12-2](#)のように、無効な値を挿入することなく、アプリケーション・コードでユーザーに通知できます。

例12-2 制約とアプリケーション・コードの両方を使用したビジネス・ルールの施行

```
CREATE OR REPLACE PROCEDURE add_employee (  
    e_name    emp_tab.empname%TYPE,  
    e_gender  emp_tab.empgender%TYPE,  
    e_number  emp_tab.empno%TYPE,  
    e_dept    emp_tab.deptno%TYPE  
) AUTHID DEFINER IS  
BEGIN  
    IF UPPER(e_gender) IN ('M','F') THEN  
        INSERT INTO emp_tab VALUES (e_name, e_gender, e_number, e_dept);  
    ELSE  
        DBMS_OUTPUT.PUT_LINE('Gender must be M or F.');    END IF;  
END;  
/  
  
BEGIN  
    add_employee ('Smith', 'H', 356, 20);  
END;  
/  

```

結果:

```
Gender must be M or F.
```

12.3 制約で使用する索引の作成

一意キー制約または主キー制約が有効な場合は、Oracle Databaseによって索引が自動的に作成されますが、これらの索引を明示的に作成することをお勧めします。外部キー制約で索引を使用する場合は、索引を明示的に作成する必要があります。

す。

制約で既存の索引を使用できる場合、その制約の索引はOracle Databaseでは作成されません。次の点に注意してください。

- 一意キー制約または主キー制約では、一意索引、一意でない索引全体、または一意でない索引の最初の数個の列のいずれかを使用できます。
- 一意キー制約または主キー制約で、一意でない索引を使用している場合は、その他の一意キー制約または主キー制約では、一意でないその索引を使用できません。
- 制約の列順序と索引は一致する必要がありません。
- 一意キー制約または主キー制約が使用している索引のオブジェクト番号は、その制約のCDEF\$.ENABLEDに格納されています。この情報は、静的データ・ディクショナリ・ビューまたは動的なパフォーマンス・ビューには表示されません。

有効な一意キー制約または主キー制約が索引を使用している場合は、索引のみを削除することはできません。索引を削除するには、制約自身を削除するか、その制約を無効にしてから索引を削除する必要があります。

関連項目:

- 制約に関連付けられている索引の詳細は、『Oracle Database管理者ガイド』を参照してください。
- 制約の無効化および削除の詳細は、『[Oracle Database管理者ガイド](#)』を参照してください。
- 明示的な索引作成の詳細は、『[Oracle Database管理者ガイド](#)』を参照してください。
- [データベース・アプリケーションでの索引の使用](#)
- 明示的な索引作成の詳細は、『[Oracle Database SQL言語リファレンス](#)』を参照してください。

12.4 NOT NULL制約を使用する場合

デフォルトでは、列にNULL値を含むことができます。列にNULL値が含まれていないことを確認するには、NOT NULL制約を使用します。

NOT NULL制約は、次の状況の両方で使用します。

- 列に非NULL値を含める必要がある場合。
たとえば、表HR.EMPLOYEESでは、各従業員に従業員IDが必要です。したがって、列HR.EMPLOYEES.EMPLOYEE_IDにはNOT NULL制約が含まれるため、EMPLOYEE_IDに非NULL値を指定せずに、新しい従業員レコードをHR.EMPLOYEESに挿入することはできません。給与を指定せずに、新しい従業員レコードをHR.EMPLOYEESに挿入できるため、列HR.EMPLOYEES.SALARYには、NOT NULL制約は含まれません。
- 表の索引スキャンを実行したり、すべての行の索引付けが必要な操作を実行できるようにする場合。
Oracle Databaseの索引には、値がすべてNULLのキーは格納されません。したがって、前述の種類の操作の場合は、1つ以上の索引列にNOT NULL制約が必要です。

[例12-3](#)は、SQL*PlusコマンドDESCRIBEを使用して、DEPARTMENTS表のどの列にNOT NULL制約があるか、NOT NULL制約がある列にNULL値を挿入しようとするとうなるかを示します。

例12-3 NOT NULL制約がある列へのNULL値の挿入

```
DESCRIBE DEPARTMENTS;
```

結果:

Name	Null?	Type
-----	-----	-----
DEPARTMENT_ID	NOT NULL	NUMBER(4)
DEPARTMENT_NAME	NOT NULL	VARCHAR2(30)
MANAGER_ID		NUMBER(6)
LOCATION_ID		NUMBER(4)

DEPARTMENT_ID列にNULLを挿入してみましょう。

```
INSERT INTO DEPARTMENTS (  
  DEPARTMENT_ID, DEPARTMENT_NAME, MANAGER_ID, LOCATION_ID  
)  
VALUES (NULL, 'Sales', 200, 1700);
```

結果:

```
VALUES (NULL, 'Sales', 200, 1700)  
*  
ERROR at line 4:  
ORA-01400: cannot insert NULL into ("HR"."DEPARTMENTS"."DEPARTMENT_ID")
```

NULLにできない列の値を省略することは、その列に値NULLを割り当てることと同じです。

```
INSERT INTO DEPARTMENTS (  
  DEPARTMENT_NAME, MANAGER_ID, LOCATION_ID  
)  
VALUES ('Sales', 200, 1700);
```

結果:

```
INSERT INTO DEPARTMENTS (  
*  
ERROR at line 1:  
ORA-01400: cannot insert NULL into ("HR"."DEPARTMENTS"."DEPARTMENT_ID")
```

DEPARTMENT_IDに非NULLのデフォルト値を指定することで、前述のエラーを回避できます。

NOT NULL制約を他の制約と組み合わせて、特定の列で可能な値をさらに制限できます。たとえば、NOT NULLと一意制約を組み合せると、一意キーに必ず値が入力され、新しい行のデータと既存の行のデータが競合する可能性がなくなります。

関連項目:

- NOT NULL制約の詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。
- デフォルト列値の使用の詳細は、[「デフォルトの列値を使用する場合」](#)を参照してください。
- [外部キーのUNIQUE制約およびNOT NULL制約](#)

12.5 デフォルトの列値を使用する場合

INSERT文で特定の列の値を指定しない場合、その列はデフォルト値を受け取ります。デフォルトでは、そのデフォルト値はNULLです。表の作成中に列を定義するとき、またはALTER TABLE文を使用して列を変更するときに、デフォルト値を変更できます。

ノート:



列に非 NULL のデフォルト値を指定しても、NOT NULL 制約のように、列の値に NULL 値が含まれないようにすることはできません。

デフォルトの列値は、次の状況で使います。

- 列にNOT NULL制約がある場合。

列に非NULLのデフォルト値を指定すると、列に値を指定せずに行を挿入した場合に発生するエラーを防止できます。

- 列に最も一般的な値がある場合。

たとえば、社内のほとんどの部門がニューヨークにある場合は、列DEPARTMENTS.LOCATIONのデフォルト値を 'NEW YORK' に設定します。

- エントリなしを示す非NULL値がある場合。

たとえば、列EMPLOYEES.SALARYの値ゼロが、給与がまだ決定されていないことを意味する場合は、その列のデフォルト値をゼロに設定します。

エントリなしを示すデフォルトの列値ではテストを簡単にできます。たとえば、次のテストを変更できます。

```
IF (employees.salary IS NOT NULL) AND (employees.salary < 50000)
```

変更後のテスト:

```
IF employees.salary < 50000
```

- 表を変更するユーザーの名前を自動的に記録する場合。

たとえば、ビューを介してユーザーが行を表に挿入できるようにするとします。inserter列(ビューの定義に含める必要のない列)を実表に指定し、行を挿入したユーザーの名前を格納します。ユーザー名を自動的に記録するには、USER関数を起動するデフォルト値を定義します。たとえば:

```
CREATE TABLE audit_trail (  
  value1  NUMBER,  
  value2  VARCHAR2(32),  
  inserter VARCHAR2(30) DEFAULT USER);
```

関連項目:

- INSERT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。
- CREATE TABLE文の詳細は、[Oracle Database SQL言語リファレンス](#)を参照
- ALTER TABLE文の詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。
- NOT NULL制約の詳細は、[\[NOT NULL制約を使用する場合\]](#)を参照してください。

12.6 表の主キーの選択(PRIMARY KEY制約)

表の主キーは各行を一意に識別し、行の重複も回避できます(通常はこの目的のみ)。したがって、主キーの値をNULLにすることはできません。

表に含めることができる主キーは最大で1つですが、複数の列を含めることができます(複合キーが可能)。主キーを指定するに

は、主キー制約を使用します。

実用的なものとしては、値が順次生成される単一系列を主キーとして選択します。

主キーとして選択するもので、次に最適なのは、値が次のすべてに該当する単一系列です。

- 一意
- 変更なし
- 非NULL
- 短い数値(入力が簡単)

値が長く順次生成できない複合主キーの使用については、最小限にします。

関連項目:

- 主キー制約の詳細は、『Oracle Database概要』を参照してください。
- 制限などの主キー制約の詳細は、『Oracle Database SQL言語リファレンス』を参照してください。
- 順序の詳細は、『[Oracle Database SQL言語リファレンス](#)』を参照してください。

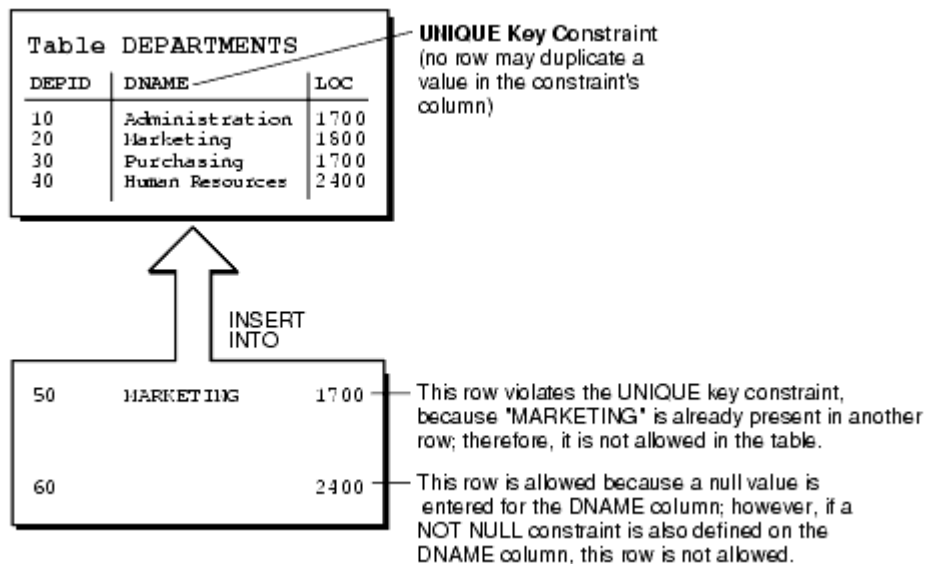
12.7 UNIQUE制約を使用する場合

非NULL値の重複が許可されていない1つの列または複数の列の組合せ(主キーを除く)では、一意制約を使用します。たとえば:

一意キー	主キー
従業員の社会保障番号	従業員番号
トラックのナンバープレートの番号	トラック番号
顧客の電話番号(国コード列、市外局番列、市内局番列)	顧客番号
部門名列および拠点列	部門番号

[図12-1](#)は、UNIQUE制約を使用した表、制約に違反している行、および制約を満たしている行を示しています。

図12-1 UNIQUE制約に違反およびUNIQUE制約を満たす行



関連項目:

- 一意制約の詳細は、『Oracle Database概要』を参照してください。
- 制限などの一意制約の詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

12.8 FOREIGN KEY制約を使用した参照整合性の施行

2つの表に1つ以上の共通の列が含まれる場合、外部キー制約を使用して、参照整合性(両方の表で共通の列に常に同じ値が含まれるようにする)を施行できます。

ノート:



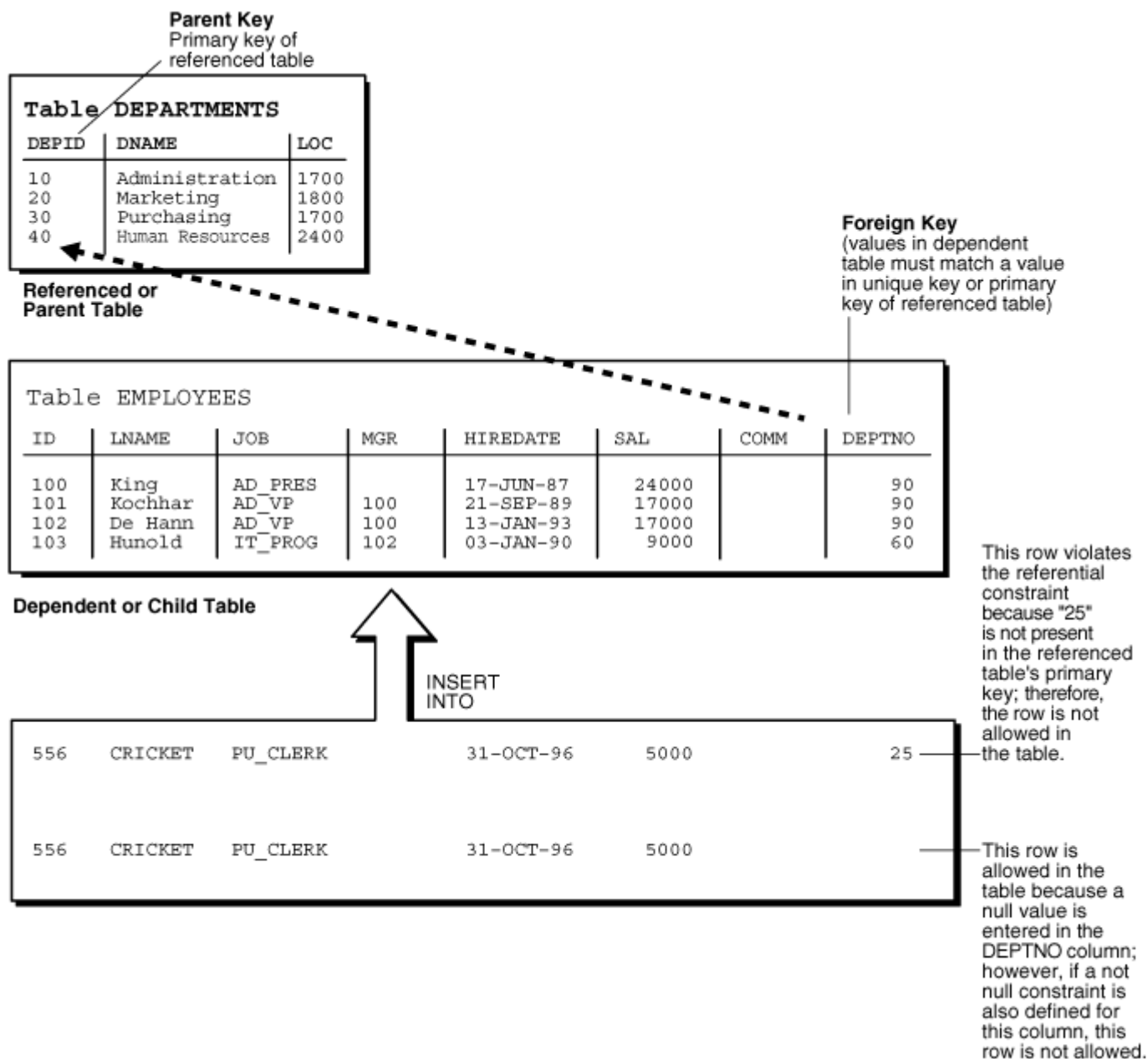
外部キー制約は参照整合性制約とも呼ばれ、静的データ・ディクショナリ・ビュー*_CONSTRAINTS では、その CONSTRAINT_TYPE は、R になります。

一方の表を参照表または親表として指定し、もう一方の表を依存または子表として指定します。親表では、共通する列に主キー制約または一意制約を定義します。子表では、共通する列に外部キー制約を定義します。これで、共通する列は外部キーで構成されます。外部キーに追加で制約を定義すると、親子関係に影響します。

[図12-2](#)に、部門番号に定義された外部キーを示します。この外部キーは、この列の値がそれぞれ部門表の主キーの値と一致することを保証します。制約によって、間違った部門番号が従業員表に入力される可能性を回避できます。

[図12-2](#)は、1つの列を共有する親表と子表、外部キー制約に違反している行、およびその制約を満たしている行を示しています。

図12-2 FOREIGN KEY制約に違反および外部キー制約を満たす行



トピック:

- [FOREIGN KEY制約およびNULL値](#)
- [親表と子表との関連の定義](#)
- [複数のFOREIGN KEY制約に関する規則](#)
- [制約チェックの遅延](#)

関連項目:

- 外部キー制約の詳細は、『Oracle Database概要』を参照してください。
- 制限などの外部キー制約の詳細は、『Oracle Database SQL言語リファレンス』を参照してください。
- [親表と子表との関連の定義](#)

12.8.1 FOREIGN KEY制約およびNULL値

外部キーには、一致する主キーまたは一意キーがない場合でも、すべてのNULLのキー値を使用できます。

- デフォルトでは(NOT NULL句またはCHECK句を指定しない場合)、外部キー制約は、ANSI/ISO規格の複合外部キーに対して不一致規則を施行します。
- 複合外部キーのNULL値に対する完全一致規則(キーのすべての構成要素がNULLまたは非NULLであることを要求する)を施行するには、すべての複合外部キーがNULLまたは非NULLであることのみを許可するCHECK制約を定義します。たとえば、列A、B、Cで構成される複合キーを次のとおり指定できます。

```
CHECK ((A IS NULL AND B IS NULL AND C IS NULL) OR  
       (A IS NOT NULL AND B IS NOT NULL AND C IS NOT NULL))
```

- 一般に、宣言型参照整合性を使用して、複合外部キーのNULL値に対する部分一致規則(キーの非NULL部分が、参照表の主キーまたは一意キー列の対応する部分にあることを要求する)を施行することはできません。多くの場合、トリガーを使用してこのような状況に対応できます。

関連項目:

トリガーの詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください。

12.8.2 親表と子表との関連の定義

親表と子表との関連のいくつかは、子表の外部キーで定義されている他のタイプの制約によって判断できます。

外部キーに制約がありません

外部キーに他の制約が定義されていない場合は、子表の行は何行でも同一の親キー値を参照できます。このモデルでは外部キーにNULLが許可されます。

このモデルは、外部キーに未定の値(NULL)を許可する親キーと外部キーとの間に1対多関連を確立します。employee(従業員)表とdepartment(部門)表の間のこのような関連の例を、[図12-2](#)に示します。各部門(親キー)には多数の従業員(外部キー)が所属しますが、一部の従業員は、部門に所属していない場合があります(外部キーでNULL)。

外部キーのNOT NULL制約

外部キーにnullが許容されていない場合、子表の各行は親キーで値を明示的に参照する必要があります(外部キーにnullが許容されていないため)。

子表の行は何行でも同一の親キー値を参照できるため、このモデルは、親キーと外部キーとの1対1関連を確立します。ただし、子表の各行は、必ず親キー値に対する参照を持っている必要があり、外部キーに値の欠如(NULL)があってははいけません。前述の項の例で、この関連を説明できます。ただし、このモデルでは、従業員は必ず特定の部門への参照を持つ必要があります。

外部キーのUNIQUE制約

外部キーでUNIQUE制約が定義されている場合は、子表の1つの行のみが指定された親キーの値を参照できます。このモデルでは外部キーにNULLが許可されます。

このモデルは、外部キーに未定の値(NULL)を許可する親キーと外部キーとの間に1対1関連を確立します。たとえば、従業員表に、企業の保険計画の従業員の会員番号を参照するMEMBERNOという名前の列があると想定します。また、INSURANCEという表には、MEMBERNOという主キーがあり、その他の列は保険証書に関連した各従業員の情報を保持しているとします。次の理由によって、従業員表のMEMBERNOは、外部キーかつ一意キーである必要があります。

- EMP_TAB表およびINSURANCE表の参照整合性規則を施行するため(外部キー制約)
- 各従業員の会員番号を一意にするため(一意キー制約)

外部キーのUNIQUE制約およびNOT NULL制約

外部キーでUNIQUE制約およびNOT NULL制約が定義されている場合は、子表の1つの行のみが指定された親キーの値を参照できます。また、外部キーにNULL値が許可されないため、子表の各行は親キーの値を明示的に参照する必要があります。

このモデルは、外部キーに未定の値(NULL)を許可しない親キーと外部キーとの間に1対1関連を確立します。前述の例を拡張して、各従業員が一意的な会員番号を持つように保証するとともに、従業員表のMEMBERNO列にNOT NULL制約を追加することで、従業員表のMEMBERNO列に未定義の値(NULL)が許可されないようにできます。

12.8.3 複数のFOREIGN KEY制約に関する規則

Oracle Databaseでは、1つの列を複数の外部キー制約で参照できます。依存キーの数に制限はありません。ある列が2つの異なる複合外部キーの一部になっている場合に、この状況が発生する可能性があります。

12.8.4 制約チェックの遅延

Oracle Databaseが制約を確認する場合に、制約が満たされないときは、エラーが表示されます。現行のトランザクションが終了するまで制約のチェックを遅延するには、SET CONSTRAINTS文を使用します。

ノート:



SET CONSTRAINTS 文は、トリガー内では使用できません。

制約のチェックを遅延するには次のようにします。

- データを適切に選択します。
データに次のいずれかの特性がある場合、一意キーおよび外部キーの制約チェックを遅延する必要があります。
 - 表がスナップショットの場合
 - 別のアプリケーションで処理される大量のデータを含む表であり、同じ順序でデータが戻されるかどうか分からない場合。

- 外部キーに対するカスケード操作を更新します。
- 制約が遅延可能かどうかを確認します。

適切な表を指定した後、表の外部キー、一意キーおよび主キーがDEFERRABLEとして作成されているかどうかを確認します。

- データを処理するアプリケーション内で、任意のデータを処理する前に、次のようにすべての制約に遅延を設定します。

```
SET CONSTRAINTS ALL DEFERRED;
```

- (オプション)トランザクションをコミットする直前に制約違反をチェックします。

COMMIT文の直前に、SET CONSTRAINTS ALL IMMEDIATE文を実行します。制約に問題があった場合、この文は失敗し、エラーの原因になった制約が識別されます。制約が違反しているときにコミットした場合、トランザクションはロールバックされ、エラー・メッセージが表示されます。

[例12-4](#)では、表empの主キーと 外部キーがDEFERRABLEとして作成され、遅延されます。

例12-4 制約チェックの遅延

```
DROP TABLE dept;
CREATE TABLE dept (
  deptno NUMBER PRIMARY KEY,
  dname  VARCHAR2 (30)
);

DROP TABLE emp;
CREATE TABLE emp (
  empno  NUMBER,
  ename  VARCHAR2(30),
  deptno NUMBER,
  CONSTRAINT pk_emp_empno PRIMARY KEY (empno) DEFERRABLE,
  CONSTRAINT fk_emp_deptno FOREIGN KEY (deptno)
  REFERENCES dept(deptno) DEFERRABLE);

INSERT INTO dept (deptno, dname) VALUES (10, 'Accounting');
INSERT INTO dept (deptno, dname) VALUES (20, 'SALES');

INSERT INTO emp (empno, ename, deptno) VALUES (1, 'Corleone', 10);
INSERT INTO emp (empno, ename, deptno) VALUES (2, 'Costanza', 20);
COMMIT;

SET CONSTRAINTS ALL DEFERRED;

UPDATE dept
SET deptno = deptno + 10
WHERE deptno = 20;
```

問合せ:

```
SELECT * from dept
ORDER BY deptno;
```

結果:

```
DEPTNO DNAME
-----
      10 Accounting
      30 SALES

2 rows selected.
```

更新:

```
UPDATE emp
SET deptno = deptno + 10
WHERE deptno = 20;
```

結果:

```
1 row updated.
```

問合せ:

```
SELECT * from emp
ORDER BY deptno;
```

結果:

EMPNO	ENAME	DEPTNO
1	Corleone	10
2	Costanza	30

2 rows selected.

SET CONSTRAINTSは現行のトランザクションのみに適用されます。設定が継続するのは、そのトランザクションの間、または次のSET CONSTRAINTS文によってモードがリセットされるまでです。ALTER SESSION SET CONSTRAINTS文は、現行セッションにのみ適用されます。制約の作成時に指定したデフォルト値は、制約の存在中は維持されます。

関連項目:

SET CONSTRAINTS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。

12.9 制約に対応付けられた索引の領域と時間のオーバーヘッドの最小化

一意キーまたは主キーを作成すると、Oracle Databaseは、既存の索引が制約の一意性を強制するかどうかを確認します。既存の索引がない場合、データベースは索引を作成します。

Oracle Databaseが一意索引を使用して制約を施行し、それに対応付けられた制約が削除または使用禁止にされる場合、索引は削除されます。索引に関連付けられた統計を保持する場合(再作成には時間がかかる)は、DROP CONSTRAINT文にKEEP INDEX句を指定します。

使用可能な外部キーが主キーまたは一意キーを参照している間は、主キー制約または一意キー制約、または索引を使用禁止にしたり削除することはできません。

ノート:



遅延可能制約を使用した一意キーおよび主キーには、非一意索引を使用してください。

一意キー制約および主キー制約の作成時に既存の索引を使用するには、CONSTRAINT句にUSING INDEXを含めます。

関連項目:

整合性制約の詳細と例は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。

12.10 外部キーを索引付けするためのガイドライン

一致する一意キーまたは主キーが決して更新または削除されない場合にかぎり、外部キーに索引を付けます。

関連項目:

外部キーの索引付けの詳細は、[『Oracle Database概要』](#)を参照してください。

12.11 分散データベース内の参照整合性

参照制約の宣言では、リモート表の主キーまたは一意キーを参照する外部キーを指定できません。

ただし、トリガーを使用すると、複数のノードにまたがる親子の表の関連をメンテナンスできます。

関連項目:

参照整合性を施行するトリガーの詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。

ノート:

トリガーを使用して分散データベースの複数のノードにまたがる参照整合性を定義する場合、ネットワーク障害が親表および子表へのアクセスを制限する可能性があることに注意してください。



たとえば、子表が SALES データベースに存在し、親表が HQ データベースに存在すると想定します。

2つのデータベース間のネットワーク接続に障害が発生すると、参照整合性トリガーが HQ データベース内の親表へアクセスする必要があるため、子表に対するデータ操作言語(DML)文が処理(行を挿入したり、外部キーの値を更新するような処理)を進めることができない場合があります。

12.12 CHECK制約を使用する場合

比較などの論理式をベースとした整合性規則を施行する必要がある場合、CHECK制約を使用します。その他のタイプの制約で必要なチェックができる場合には、CHECK制約は使用しないでください。

関連項目:

[CHECKおよびNOT NULL制約の選択](#)

CHECK制約の例を次に示します。

- 給与の値が10000を超えないように、従業員の給与にCHECK制約を定義します。
- 「BOSTON」、「NEW YORK」および「DALLAS」のみが許可されるように、部門の所在地にCHECK制約を定義します。
- コミッションの額が給与より多くならないように、給与およびコミッションの列にCHECK制約を定義します。

12.12.1 CHECK制約の制限

CHECK制約では、条件は表のすべての行に対して真または不明である必要があります。条件が偽であると評価された場合、その文はロールバックされます。CHECK制約の条件には、次のような制限があります。

- 条件は、挿入または更新が行われている行の値を使用して評価できるブール式である必要があります。
- 条件に副問合せまたは順序を含めることはできません。
- 条件にSQLファンクションSYSDATE、UID、USERまたはUSERENVを含めることはできません。
- 条件に疑似列LEVELまたはROWNUMを含めることはできません。

- 条件にPRIOR演算子を含めることはできません。
- 条件にユーザー定義ファンクションを含めることはできません。

関連項目:

- LEVEL擬似列の詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。
- ROWNUM擬似列の詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。
- PRIOR演算子(階層問合せで使用)の詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。

12.12.2 CHECK制約の設計

CHECK制約は、条件が偽であると評価される場合にのみCHECK制約に違反します。真および不明(NULLと比較して)はチェック条件には違反しません。したがって、定義するCHECK制約が、規則を施行するために十分明確であることを確認してください。

たとえば、次のCHECK制約について考えます。

```
CHECK (Sal > 0 OR Comm >= 0)
```

この規則は、「従業員の給与が0(ゼロ)より大きい場合、または従業員のコミッションが0(ゼロ)以上の場合、従業員表の行を許可しない」と解釈されます。ただし、給与にNULLの値を持つ行は、チェック条件全体が不明であると評価されるため、そのコミッションの値の評価にかかわらず、CHECK制約に違反しません。このような場合には、SAL列とCOMM列の両方にNOT NULL制約を設定することによって、このような違反を回避できます。

ノート:



どのような場合に NULL 条件の結果が不明な値となるかについては、[『Oracle Database SQL 言語リファレンス』](#)の論理条件の真理値表を参照してください。

12.12.3 複数のCHECK制約に関する規則

1つの列に、その定義で列を参照する複数のCHECK制約を指定できます。定義できるCHECK制約の数に制限はありません。制約が評価される順序は定義されません。そのため、順序に依存したり、互いに競合するような複数の制約を定義しないでください。

12.12.4 CHECKおよびNOT NULL制約の選択

ANSI/ISO規格によると、NOT NULL制約はCHECK制約の1つであり、その条件は次のとおりです。

```
CHECK (column_name IS NOT NULL)
```

このため、NOT NULL制約またはCHECK制約のいずれかを使用して、単一系列に対するNOT NULL制約を記述できます。NOT NULL制約は、CHECK制約よりも簡単に使用できます。

複合キーがすべてNULLまたはすべて非NULL値を持つ場合は、CHECK制約を使用する必要があります。たとえば、次のCHECK制約を指定すると、列C1およびC2を構成する複合キーのキー値が、すべてNULLまたはすべて値を持つことができます。

```
CHECK ((C1 IS NULL AND C2 IS NULL) OR (C1 IS NOT NULL AND C2 IS NOT NULL))
```

12.13 制約の定義の例

[例12-5](#)および[例12-6](#)に、データベース設計のプロトタイプ・フェーズでの簡単な制約の作成方法を示します。これらの例では各制約に名前が付いています。制約に名前を付けると、データ定義言語(DDL)が複数回実行された場合に、システムが生成した異なる名前で、データベースが同じ制約の複数のコピーを作成することを回避できます。

[例12-5](#)では、CREATE TABLE文を使用して表と制約を同時に作成します。

例12-5 CREATE TABLE文を使用した制約の定義

```
DROP TABLE DeptTab;
CREATE TABLE DeptTab (
  Deptno  NUMBER(3) CONSTRAINT pk_DeptTab_Deptno PRIMARY KEY,
  Dname   VARCHAR2(15),
  Loc     VARCHAR2(15),
  CONSTRAINT u_DeptTab_Dname_Loc UNIQUE (Dname, Loc),
  CONSTRAINT c_DeptTab_Loc
    CHECK (Loc IN ('NEW YORK', 'BOSTON', 'CHICAGO')));

DROP TABLE EmpTab;
CREATE TABLE EmpTab (
  Empno    NUMBER(5) CONSTRAINT pk_EmpTab_Empno PRIMARY KEY,
  Ename    VARCHAR2(15) NOT NULL,
  Job      VARCHAR2(10),
  Mgr      NUMBER(5) CONSTRAINT r_EmpTab_Mgr REFERENCES EmpTab,
  Hiredate DATE,
  Sal      NUMBER(7,2),
  Comm     NUMBER(5,2),
  Deptno   NUMBER(3) NOT NULL
  CONSTRAINT r_EmpTab_DeptTab REFERENCES DeptTab ON DELETE CASCADE);
```

[例12-6](#)では、ALTER TABLE文を使用して既存の表に制約を作成します。

制約に違反する行が表に存在している場合、VALIDATED状態の制約は作成できません。

例12-6 ALTER TABLE文を使用した制約の定義

```
-- Create tables without constraints:
DROP TABLE DeptTab;
CREATE TABLE DeptTab (
  Deptno  NUMBER(3),
  Dname   VARCHAR2(15),
  Loc     VARCHAR2(15)
);

DROP TABLE EmpTab;
CREATE TABLE EmpTab (
  Empno    NUMBER(5),
  Ename    VARCHAR2(15),
  Job      VARCHAR2(10),
  Mgr      NUMBER(5),
  Hiredate DATE,
  Sal      NUMBER(7,2),
  Comm     NUMBER(5,2),
  Deptno   NUMBER(3)
);

-- Define constraints with the ALTER TABLE statement:
ALTER TABLE DeptTab
ADD CONSTRAINT pk_DeptTab_Deptno PRIMARY KEY (Deptno);

ALTER TABLE EmpTab
ADD CONSTRAINT fk_DeptTab_Deptno
FOREIGN KEY (Deptno) REFERENCES DeptTab;
```

```
ALTER TABLE EmpTab MODIFY (Ename VARCHAR2(15) NOT NULL);
```

関連項目:

大規模な本番データベースに対する制約の作成方法およびメンテナンス方法の詳細は、[『Oracle Database管理者ガイド』](#)を参照してください。

12.13.1 制約の定義に必要な権限

CREATE TABLEまたはCREATE ANY TABLEシステム権限がある場合、作成する表の制約を定義できます。

ALTER ANY TABLEシステム権限ある場合は、既存のすべての表に制約を定義できます。

特定の表のALTERオブジェクト権限がある場合は、その表に制約を定義できます。

一意キーおよび主キー制約では、表の所有者に、UNLIMITED TABLESPACEシステム権限、または対応する索引を含む表領域の割当て制限が必要です。

外部キー制約を定義できるのは、親表またはビューが自分のスキーマ内にある場合、または親表またはビューの参照キー列に対するREFERENCES権限を持っている場合です。

関連項目:

[FOREIGN KEY制約の作成に必要な権限](#)

12.13.2 制約のネーミング

NOT NULL、一意キー、主キー、外部キーおよびCHECKの各制約に対して、制約句のCONSTRAINTオプションを使用して名前を割り当ててください。この名前は、そのユーザーが所有している他の制約名に対して一意である必要があります。制約名を指定しない場合、Oracle Databaseが名前を自動的に生成して割り当てます。

独自の名前を指定すると、制約違反のエラー・メッセージがよりわかりやすくなります。また、SQL文が複数回実行された場合に、異なる名前で重複した制約が作成されることを回避できます。

制約句のCONSTRAINTオプションの例として、前述のCREATE TABLE文およびALTER TABLE文の例を参照してください。データ・ディクショナリでは、各制約名が制約に関する他の情報に含まれます。

関連項目:

静的データ・ディクショナリ・ビューの例は、「[制約に関する情報の表示](#)」を参照してください

12.14 制約の使用可能および使用禁止

この項では、制約をユーザー自身で使用可能および使用禁止にするしくみ、および手順について説明します。

使用可能にされた制約。制約が使用可能な場合、対応する規則が対応付けられた列のデータ値に施行されます。制約の定義は、データ・ディクショナリ内に格納されます。

使用禁止にされた制約。制約が使用禁止の場合、それに対応する規則は施行されません。制約の定義は、データ・ディクショ

ナリ内に格納されたままです。

整合性制約は、データベース内のデータに関するアサーションを表します。このアサーションは、制約を使用可能にすると必ず真になります。制約を使用禁止にすると、整合性制約に違反するデータがデータベース内に存在する可能性があるため、アサーションは真でない場合もあります。

トピック:

- [制約を使用禁止にする理由](#)
- [制約を使用可能にする\(デフォルト\)](#)
- [制約を使用禁止にする](#)
- [既存の制約を使用可能にする](#)
- [既存の制約を使用禁止にする](#)
- [キー制約の使用可能および使用禁止に関するガイドライン](#)
- [制約の例外の修正](#)

12.14.1 制約を使用禁止にする理由

日常の操作では、制約を使用可能な状態にしておきます。特定の状況においては、パフォーマンス上の理由から、表の制約を一時的に使用禁止にする必要がある場合があります。たとえば:

- SQL*Loaderを使用して、表に大量のデータをロードする場合
- 表に対して大規模な変更を行うバッチ作業を実施する場合(たとえば、既存の番号に1000を加えてすべての従業員番号を変更する場合)
- 表を1つずつインポートまたはエクスポートする場合

制約を一時的に使用禁止にすると、これらの操作が高速になります。

12.14.2 制約を使用可能にする(デフォルト)

整合性制約を定義すると(CREATE TABLEまたはALTER TABLEを使用)、Oracle Databaseではデフォルトで制約が使用可能になります。コードをわかりやすくするために、[例12-7](#)のように、制約の定義にENABLE句を含めて制約を明示的に使用可能にできます。

例12-7 制約を使用可能にする

```
/* Use CREATE TABLE statement to create enabled constraint
   (ENABLE keyword is optional): */

DROP TABLE t1;
CREATE TABLE t1 (Empno NUMBER(5) PRIMARY KEY ENABLE);

/* Create table without constraint
   and then use ALTER TABLE statement to add enabled constraint
   (ENABLE keyword is optional): */
DROP TABLE t2;
CREATE TABLE t2 (Empno NUMBER(5));

ALTER TABLE t2 ADD PRIMARY KEY (Empno) ENABLE;
```

個々のトランザクションによって一度に1つずつ行が移入される表の制約を定義する場合は、ENABLE句を指定します。こうする

ことで、データが常に一貫性を保つことが保証され、各DML文のパフォーマンス・オーバーヘッドが少なくなります。

整合性制約を使用可能にするALTER TABLE文は、表の既存の行がその整合性制約に違反するとエラーになります。文はロールバックされ、制約定義は格納されず使用可能にもなりません。

関連項目:

制約に違反する行の詳細は、[「制約の例外の修正」](#)を参照してください。

12.14.3 制約を使用禁止にする

整合性制約を定義(CREATE TABLEまたはALTER TABLEを使用)して使用禁止にするには、[例12-8](#)のように定義にDISABLE句を指定します。

例12-8 制約を使用禁止にする

```
/* Use CREATE TABLE statement to create disabled constraint */

DROP TABLE t1;
CREATE TABLE t1 (Empno NUMBER(5) PRIMARY KEY DISABLE);

/* Create table without constraint
   and then use ALTER TABLE statement to add disabled constraint */

DROP TABLE t2;
CREATE TABLE t2 (Empno NUMBER(5));

ALTER TABLE t2 ADD PRIMARY KEY (Empno) DISABLE;
```

ユーザーがアクセスする前に大容量のデータが挿入される表の制約を定義するときは、DISABLE句を指定します。特に、データの挿入後にデータをクリーン・アップしたり、空の列に順序番号または親子関係を入力する必要がある場合です。

制約を定義して使用禁止にするALTER TABLE文は、その規則が施行されていないため、エラーとなることはありません。

12.14.4 既存の制約を使用可能にする

データをクリーン・アップし、空の列に値を入力した後で、データの挿入中に使用禁止にしていた制約を使用可能にできます。

既存の制約を使用可能にするには、[例12-9](#)のようにALTER TABLE文をENABLE句とともに使用します。

例12-9 既存の制約を使用可能にする

```
-- Create table with disabled constraints:

DROP TABLE DeptTab;
CREATE TABLE DeptTab (
  Deptno  NUMBER(3) PRIMARY KEY DISABLE,
  Dname   VARCHAR2(15),
  Loc     VARCHAR2(15),
  CONSTRAINT uk_DeptTab_Dname_Loc UNIQUE (Dname, Loc) DISABLE,
  CONSTRAINT c_DeptTab_Loc
  CHECK (Loc IN ('NEW YORK', 'BOSTON', 'CHICAGO')) DISABLE
);

-- Enable constraints:

ALTER TABLE DeptTab
ENABLE PRIMARY KEY
ENABLE CONSTRAINT uk_DeptTab_Dname_Loc
ENABLE CONSTRAINT c_DeptTab_Loc;
```

整合性制約を使用可能にするALTER TABLE文は、表の行がその整合性制約に違反するとエラーになります。この場合、文はロールバックされ、制約は使用可能になりません。

関連項目:

制約に違反する行の詳細は、[「制約の例外の修正」](#)を参照してください。

12.14.5 既存の制約を使用禁止にする

データが含まれている表に対して、大規模な挿入または更新を実行する必要がある場合は、整合性制約を一時的に使用禁止にして、バルク操作のパフォーマンスを向上させることができます。

既存の制約を使用禁止にするには、例12-10のようにALTER TABLE文を[DISABLE](#)句とともに使用します。

例12-10 既存の制約を使用禁止にする

```
-- Create table with enabled constraints:

DROP TABLE DeptTab;
CREATE TABLE DeptTab (
  Deptno  NUMBER(3) PRIMARY KEY ENABLE,
  Dname   VARCHAR2(15),
  Loc     VARCHAR2(15),
  CONSTRAINT uk_DeptTab_Dname_Loc UNIQUE (Dname, Loc) ENABLE,
  CONSTRAINT c_DeptTab_Loc
  CHECK (Loc IN ('NEW YORK', 'BOSTON', 'CHICAGO')) ENABLE
);

-- Disable constraints:

ALTER TABLE DeptTab
DISABLE PRIMARY KEY
DISABLE CONSTRAINT uk_DeptTab_Dname_Loc
DISABLE CONSTRAINT c_DeptTab_Loc;
```

12.14.6 キー制約の使用可能および使用禁止に関するガイドライン

一意キー、主キーおよび外部キーの各制約を使用可能または使用禁止にする場合は、いくつかの重要な問題および前提条件を認識してください。一意キー制約および主キー制約は、通常、データベース管理者が管理します。

関連項目:

[『Oracle Database管理者ガイド』](#)および[「FOREIGN KEY制約の管理」](#)

12.14.7 制約の例外の修正

表の行が整合性制約に違反する場合、この行は制約違反になり、制約に対する**例外**とされます。例外が存在する場合、制約を使用可能にはできません。制約に違反する行は、制約を使用可能にする前に更新または削除する必要があります。

制約を使用可能にするときに、特定の整合性制約に対する例外を指定できます。

関連項目:

この手順の詳細は、[「制約の例外の修正」](#)を参照してください

制約を作成または使用可能にするときに、整合性制約の例外があるために文が正常に実行されなかった場合、文はロールバックされます。この場合、すべての例外が更新または削除されるまで制約を使用可能にできません。整合性制約に違反している行を判断するには、CREATE TABLE文またはALTER TABLE文のENABLE句にEXCEPTIONSオプションを指定します。

関連項目:

制約の例外に応答する場合は、[『Oracle Database管理者ガイド』](#)を参照してください。

12.15 制約の変更

Oracle8i以降では、例12-11のようにMODIFY [CONSTRAINT](#)句を使用して既存の制約状態を変更することができます。

関連項目:

変更するパラメータの詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。

例12-11 制約の変更

```
/* Create & then modify a CHECK constraint: */

DROP TABLE X1Tab;
CREATE TABLE X1Tab (
  a1 NUMBER
  CONSTRAINT c_X1Tab_a1 CHECK (a1>3)
  DEFERRABLE DISABLE
);

ALTER TABLE X1Tab
MODIFY CONSTRAINT c_X1Tab_a1 ENABLE;

ALTER TABLE X1Tab
MODIFY CONSTRAINT c_X1Tab_a1 RELY;

ALTER TABLE X1Tab
MODIFY CONSTRAINT c_X1Tab_a1 INITIALLY DEFERRED;

ALTER TABLE X1Tab
MODIFY CONSTRAINT c_X1Tab_a1 ENABLE NOVALIDATE;

/* Create & then modify a PRIMARY KEY constraint: */

DROP TABLE t1;
CREATE TABLE t1 (a1 INT, b1 INT);

ALTER TABLE t1
ADD CONSTRAINT pk_t1_a1 PRIMARY KEY(a1) DISABLE;

ALTER TABLE t1
MODIFY PRIMARY KEY INITIALLY IMMEDIATE
USING INDEX PCTFREE = 30 ENABLE NOVALIDATE;

ALTER TABLE t1
MODIFY PRIMARY KEY ENABLE NOVALIDATE;
```

12.16 制約名の変更

変更可能な制約のプロパティの1つは名前です。制約の名前を変更する状況には次のようなものがあります。

- 表と制約を複製する場合。

制約名は、複数のスキーマに対しても一意であることが必要です。このため、複製された表の制約を、元の表の制約と同じ名前にすることはできません。

- 制約をデフォルトのシステム生成名で作成したが、使用可能または使用禁止の設定が容易になるように、覚えやすい制約名が必要な場合。

[例12-12](#)に、制約のシステム生成名を検索して変更する方法を示します。

例12-12 制約名の変更

```
DROP TABLE T;  
CREATE TABLE T (  
    C1 NUMBER PRIMARY KEY,  
    C2 NUMBER  
);
```

問合せ:

```
SELECT CONSTRAINT_NAME FROM USER_CONSTRAINTS  
WHERE TABLE_NAME = 'T'  
AND CONSTRAINT_TYPE = 'P';
```

結果(システムにより生成される制約名は様々です):

```
CONSTRAINT_NAME  
-----  
SYS_C0013059  
  
1 row selected.
```

この問合せで報告された制約名をT_C1_PKに変更します。

```
ALTER TABLE T  
RENAME CONSTRAINT SYS_C0013059  
TO T_C1_PK;
```

問合せ:

```
SELECT CONSTRAINT_NAME FROM USER_CONSTRAINTS  
WHERE TABLE_NAME = 'T'  
AND CONSTRAINT_TYPE = 'P';
```

結果:

```
CONSTRAINT_NAME  
-----  
T_C1_PK  
  
1 row selected.
```

12.17 制約の削除

制約を削除するには、ALTER TABLE文のDROP句を使用します。制約を削除する状況には次のようなものがあります。

- 制約で施行されるルールが適切でなくなった場合。
- 制約が不要になった場合。

1つの制約と、その制約が依存している他のすべての整合性制約を削除するには、CASCADEを指定します。

例12-13 制約の削除

```
-- Create table with constraints:
DROP TABLE DeptTab;
CREATE TABLE DeptTab (
  Deptno  NUMBER(3) PRIMARY KEY,
  Dname   VARCHAR2(15),
  Loc     VARCHAR2(15),
  CONSTRAINT uk_DeptTab_Dname_Loc UNIQUE (Dname, Loc),
  CONSTRAINT c_DeptTab_Loc
  CHECK (Loc IN ('NEW YORK', 'BOSTON', 'CHICAGO'))
);

-- Drop constraints:

ALTER TABLE DeptTab
DROP PRIMARY KEY
DROP CONSTRAINT uk_DeptTab_Dname_Loc
DROP CONSTRAINT c_DeptTab_Loc;
```

一意キー、主キーおよび外部キーの各制約を削除する場合は、いくつかの重要な問題および前提条件を認識してください。一意キー制約および主キー制約は、通常、データベース管理者が管理します。

関連項目:

- ALTER TABLE文のDROP句の詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。
- 制約の削除の詳細は、[『Oracle Database管理者ガイド』](#)を参照してください。
- DROP TABLE文のCASCADE CONSTRAINTS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。この方法では、削除される表の主キーと一意キーを参照するすべての参照整合性制約が削除されます。

12.18 FOREIGN KEY制約の管理

外部キー制約は、異なる表の列同士の関連を規定します。したがって、参照中の主キーまたは一意キーの制約が存在していないか、または使用可能になっていない場合は、制約を使用可能にできません。

12.18.1 外部キー列のデータ型および名前

依存する側の表と参照される側の表の対応する列は、同じデータ型である必要があります。列名が一致する必要はありません。

12.18.2 複合外部キーにおける列の制限

外部キーは、親表の主キーまたは一意キーを参照し、主キー制約および一意キー制約は索引を使用して施行されるため、複合外部キーは32列以内に制限されています。

12.18.3 デフォルトでの外部キーの主キー参照

外部キー制約(単一列または複合列)を定義するときに、REFERENCESオプションに列リストが指定されていないと、Oracle Databaseは、指定した表の主キーが参照されるものとみなします。または、カッコの中に親表で参照する列を明示的に指定で

きます。この列リストが親表の主キーまたは一意キーを参照するかどうかは、Oracle Databaseによって自動的に確認されます。参照していない場合は、エラー情報が戻されます。

12.18.4 FOREIGN KEY制約の作成に必要な権限

外部キー制約を作成するには、制約の作成者に親表および子表に対するアクセス権限が必要です。

- **親表** 参照整合性制約の作成者は、親表を所有するか、または親表の親キーを構成する列に対するREFERENCESオブジェクト権限が必要です。
- **子表** 参照整合性制約の作成者は、表を作成する権限(CREATE TABLEまたはCREATE ANY TABLEシステム権限)または子表を変更する権限(子表のALTERオブジェクト権限またはALTER ANY TABLEシステム権限)が必要です。

どちらの場合も、必要な権限をロールを介して取得することはできません。権限は、明示的に制約の作成者に付与する必要があります。

これらの制限によって、次のことが可能となります。

- 子表の所有者は、施行される制約の種類、または制約を作成できる他のユーザーを、明示的に決定できます。
- 親表の所有者は、外部キーが所有者自身の表の主キーおよび一意キーに依存可能であるかどうかを明示的に決定できます。

12.18.5 外部キーによる参照整合性の施行方法の選択

Oracle Databaseでは、外部キー制約の定義の指定どおりに、異なるタイプの参照整合性アクションを施行できます。

- **親キーの削除または更新の回避** このデフォルト設定を指定すると、キーを参照する行が子表内にある場合、親キーが削除または更新されることはありません。たとえば：

```
CREATE TABLE Emp_tab (  
  FOREIGN KEY (Deptno) REFERENCES Dept_tab);
```

- **親キー削除時の子である行の削除** ON DELETE CASCADEアクションを指定すると、子表が参照する親キーのデータを削除できます。ただし、更新はできません。親キー内のデータが削除されると、削除された親キー値に依存する子表のすべての行も削除されます。この参照アクションを指定するには、外部キー制約の定義にON DELETE CASCADEオプションを指定します。たとえば：

```
CREATE TABLE Emp_tab (  
  FOREIGN KEY (Deptno) REFERENCES Dept_tab  
  ON DELETE CASCADE);
```

- **親キー削除時の外部キーへのNULL設定** ON DELETE SET NULLアクションを指定すると、親キーを参照するデータを削除できます。ただし、更新はできません。親キー内の参照データが削除されると、削除された親キー値に依存する子表内のすべての行の外部キーがNULLに設定されます。この参照アクションを指定するには、外部キー制約の定義にON DELETE SET NULLオプションを指定します。たとえば：

```
CREATE TABLE Emp_tab (  
  FOREIGN KEY (Deptno) REFERENCES Dept_tab  
  ON DELETE SET NULL);
```

12.19 制約に関する情報の表示

制約の名前、影響がある列および制約の管理に有効な他の情報を検索するには、例12-14のように静的データ・ディクショナリ・ビュー*_CONSTRAINTSおよび*_CONS_COLUMNSに問い合わせます。

関連項目:

[*_CONSTRAINTS](#)および[*_CONS_COLUMNS](#)の詳細は、『Oracle Databaseリファレンス』を参照してください。

例12-14 制約に関する情報の表示

```
DROP TABLE DeptTab;
CREATE TABLE DeptTab (
  Deptno  NUMBER(3) PRIMARY KEY,
  Dname   VARCHAR2(15),
  Loc     VARCHAR2(15),
  CONSTRAINT uk_DeptTab_Dname_Loc UNIQUE (Dname, Loc),
  CONSTRAINT c_DeptTab_Loc
  CHECK (Loc IN ('NEW YORK', 'BOSTON', 'CHICAGO'))
);

DROP TABLE EmpTab;
CREATE TABLE EmpTab (
  Empno    NUMBER(5) PRIMARY KEY,
  Ename    VARCHAR2(15) NOT NULL,
  Job      VARCHAR2(10),
  Mgr      NUMBER(5) CONSTRAINT r_EmpTab_Mgr
          REFERENCES EmpTab ON DELETE CASCADE,
  Hiredate DATE,
  Sal      NUMBER(7,2),
  Comm     NUMBER(5,2),
  Deptno   NUMBER(3) NOT NULL
  CONSTRAINT r_EmpTab_Deptno REFERENCES DeptTab
);

-- Format columns (optional):

COLUMN CONSTRAINT_NAME  FORMAT A20;
COLUMN CONSTRAINT_TYPE  FORMAT A4 HEADING 'TYPE';
COLUMN TABLE_NAME      FORMAT A10;
COLUMN R_CONSTRAINT_NAME FORMAT A17;
COLUMN SEARCH_CONDITION  FORMAT A40;
COLUMN COLUMN_NAME       FORMAT A12;
```

DeptTabおよびEmpTabでアクセス可能な制約をリスト:

```
SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE, TABLE_NAME, R_CONSTRAINT_NAME
FROM USER_CONSTRAINTS
WHERE (TABLE_NAME = 'DEPTTAB' OR TABLE_NAME = 'EMPTAB')
ORDER BY CONSTRAINT_NAME;
```

結果:

CONSTRAINT_NAME	TYPE	TABLE_NAME	R_CONSTRAINT_NAME
C_DEPTTAB_LOC	C	DEPTTAB	
R_EMPTAB_DEPTNO	R	EMPTAB	SYS_C006286
R_EMPTAB_MGR	R	EMPTAB	SYS_C006290
SYS_C006286	P	DEPTTAB	
SYS_C006288	C	EMPTAB	
SYS_C006289	C	EMPTAB	

```
SYS_C006290      P      EMPTAB
UK_DEPTTAB_DNAME_LOC U      DEPTTAB
```

8 rows selected.

DeptTabおよびEmpTabで、制約NOT NULLとCHECKを区別:

```
SELECT CONSTRAINT_NAME, SEARCH_CONDITION
FROM USER_CONSTRAINTS
WHERE (TABLE_NAME = 'DEPTTAB' OR TABLE_NAME = 'EMPTAB')
AND CONSTRAINT_TYPE = 'C'
ORDER BY CONSTRAINT_NAME;
```

結果:

CONSTRAINT_NAME	SEARCH_CONDITION
C_DEPTTAB_LOC	Loc IN ('NEW YORK', 'BOSTON', 'CHICAGO')
SYS_C006288	"ENAME" IS NOT NULL
SYS_C006289	"DEPTNO" IS NOT NULL

3 rows selected.

DeptTabおよびEmpTabで、制約を構成する列をリスト:

```
SELECT CONSTRAINT_NAME, TABLE_NAME, COLUMN_NAME
FROM USER_CONS_COLUMNS
WHERE (TABLE_NAME = 'DEPTTAB' OR TABLE_NAME = 'EMPTAB')
ORDER BY CONSTRAINT_NAME;
```

結果:

CONSTRAINT_NAME	TABLE_NAME	COLUMN_NAME
C_DEPTTAB_LOC	DEPTTAB	LOC
R_EMPTAB_DEPTNO	EMPTAB	DEPTNO
R_EMPTAB_MGR	EMPTAB	MGR
SYS_C006286	DEPTTAB	DEPTNO
SYS_C006288	EMPTAB	ENAME
SYS_C006289	EMPTAB	DEPTNO
SYS_C006290	EMPTAB	EMPNO
UK_DEPTTAB_DNAME_LOC	DEPTTAB	LOC
UK_DEPTTAB_DNAME_LOC	DEPTTAB	DNAME

9 rows selected.

次の点に注意してください。

- 制約名には、ユーザー指定(たとえば、UK_DEPTTAB_DNAME_LOC)のものと、システム指定(たとえば、SYS_C006290)のものがあります。
- ・ 各制約タイプは、CONSTRAINT_TYPE列に別々の文字で表示されます。次の表に、各制約タイプに対応する文字を示します。

制約タイプ	文字
PRIMARY KEY	P

制約タイプ	文字
UNIQUE KEY	U
FOREIGN KEY	R
CHECK、NOT NULL	C

ノート:



その他の制約タイプで、CONSTRAINT_TYPE 列に「V」の文字で表示されるものがあります。この制約タイプは、ビューに対する WITH CHECK OPTION を使用して作成された制約に対応しています。

次の制約は、SEARCH_CONDITION列に明示的にリストされます。

- NOT NULL制約
- ユーザー定義CHECK制約の条件

第III部 アプリケーション開発者用のPL/SQL

ここでは、アプリケーション開発者が必要とする、SQLの手続き型拡張言語であるOracleのPL/SQLについて説明します。

章:

- [PL/SQLサブプログラムおよびパッケージのコード化](#)
- [PL/Scopeの使用](#)
- [PL/SQL階層プロファイラの使用](#)
- [PL/SQL Webアプリケーションの開発](#)
- [連続問合せ通知\(CQN\)の使用](#)

関連項目:

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。

13 PL/SQLサブプログラムおよびパッケージのコード化

PL/SQLサブプログラムおよびパッケージは、Oracle Databaseアプリケーションのビルディング・ブロックです。『Oracle Database PL/SQL言語リファレンス』に記載されている理由により、アプリケーションはパッケージとして実装することをお勧めします。

トピック:

- [PL/SQLサブプログラムの概要](#)
- [PL/SQLパッケージの概要](#)
- [PL/SQLユニットの概要](#)
- [PL/SQLサブプログラムおよびパッケージの作成](#)
- [PL/SQLサブプログラムおよびパッケージの変更](#)
- [推奨されないパッケージ、サブプログラムおよびタイプ](#)
- [PL/SQLサブプログラムおよびパッケージの削除](#)
- [システム固有の実行のためのPL/SQLユニットのコンパイル](#)
- [ストアードPL/SQLサブプログラムの起動](#)
- [SQL文からのストアードPL/SQLファンクションの起動](#)
- [ストアード・サブプログラムのデバッグ](#)
- [パッケージの無効化とセッションの状態](#)

関連項目:

- PL/SQLサブプログラムおよびパッケージのエラー処理の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください
- オブジェクト型、不透明型およびLOBを使用して格納されるマルチメディア・データのような複合データ型の集計ファンクションの作成の詳細は、『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
- PL/SQLコードの問題を発見する手助けとなるアプリケーションのトレース・ツールの詳細は、Oracle Database SQLチューニング・ガイドを参照してください

13.1 PL/SQLサブプログラムの概要

PL/SQLソース・プログラムの基本単位はブロックで、関連する宣言および文をグループ化します。ブロックには、オプションの宣言部、必須の実行可能部、オプションの例外処理部があります。ブロックには、匿名ブロックと名前付きブロックがあります。

PL/SQLのサブプログラムは、繰り返し起動できる名前付きブロックです。サブプログラムにパラメータが含まれている場合は、起動のたびにパラメータ値を変えることができます。

サブプログラムは、プロシージャまたはファンクションのいずれかです。通常、プロシージャはアクションを実行するために使用し、ファンクションは計算を行って値を戻すために使用します。

また、サブプログラムは、ネストしたサブプログラム(PL/SQLブロック内に作成され、別のサブプログラムである場合があります)、

パッケージ・サブプログラム(パッケージ仕様で宣言され、パッケージ本体で定義されます)、またはスタンドアロン・サブプログラム(スキーマ・レベルで作成されます)でもあります。パッケージ・サブプログラムおよびスタンドアロン・プログラムは、ストアド・サブプログラムです。ストアド・サブプログラムはデータベースでコンパイルおよび格納され、ここで多くのアプリケーションがストアド・サブプログラムを起動できます。

ストアド・サブプログラムは、AUTHIDおよびACCESSIBLE BY句の影響を受けます。AUTHID句は、実行時にサブプログラムによって発行されるSQL文の名前解決および権限チェックに影響を与えます。ACCESSIBLE BY句は、サブプログラムにアクセスできるPL/SQLユニットのホワイト・リストを指定します。

Oracle Databaseのインスタンス上で実行するPL/SQLサブプログラムは、第三世代言語(3GL)で作成された外部サブプログラムを起動できます。3GLサブプログラムは、データベースのアドレス空間とは別のアドレス空間で実行されます。

PL/SQLでは、ネストしたサブプログラム、パッケージ・サブプログラムおよび型のメソッドをオーバーロードできます。オーバーロードされたサブプログラムは名前が同じですが、仮パラメータの名前、数、順序またはデータ型のファミリが異なります。

トリガーは、ストアド・プロシージャのように、データベースに格納して繰り返し起動できる名前付きPL/SQLユニットです。ストアド・プロシージャとは異なり、トリガーは、有効にしたり無効にすることができますが、明示的に起動することはできません。トリガーは、有効の場合、データベースによって自動的に起動されます(つまり、トリガーは、トリガーを起動するイベントが発生すると、常に起動されます)。トリガーは、無効の場合、起動されません。

BEFORE UPDATEトリガーは、内部再試行によって複数回起動できます。アプリケーションを設計する際は、これを考慮してください。通常のトリガーを使用すると、トリガーによって行われたすべての作業は、再試行時にロールバックされます。ただし、自律型トランザクションを使用してトリガーを定義した場合、トリガーによって実行された作業はロールバックされません。

関連項目:

- PL/SQLサブプログラムの詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。
- PL/SQLブロックの詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。
- サブプログラムの詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。
- PL/SQLユニットの詳細は、[「PL/SQLユニットの概要」](#)を参照してください。
- 外部サブプログラムの詳細は、[「複数のプログラミング言語を使用したアプリケーションの開発」](#)を参照してください。
- オーバーロードされたサブプログラムの詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。
- トリガーの詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。

13.2 PL/SQLパッケージの概要

PL/SQLパッケージとは、論理的に関連するPL/SQLの型、変数、定数、サブプログラム、カーソルおよび例外をグループにまとめたスキーマ・オブジェクトのことです。パッケージをコンパイルしてデータベースに格納し、その内容を複数のアプリケーションで共有できます。

パッケージには必ず仕様部があり、パッケージの外から参照できるパブリック項目はここで宣言します。パブリック項目を起動または使用できるのは、パッケージに対するEXECUTE権限を持つ外部ユーザーまたはEXECUTE ANY PROCEDURE権限を持つ外部ユーザーです。

パブリック項目にカーソルまたはサブプログラムが含まれる場合、パッケージには本体も必要です。本体には、パブリック・カーソルの問合せとパブリック・サブプログラムのコードを定義する必要があります。本体では、パッケージの外からは参照できなくてもパッ

パッケージの内部動作に必要なプライベート項目を宣言および定義することもできます。本体の最後には初期化部(ここには、変数を初期化する文と、その他の1回のみの設定ステップを実行する文を入れます)、および例外処理部を配置することができます。本体は、仕様部またはパブリック項目への参照を変更せずに変更できるため、パッケージ本体はブラック・ボックスと考えることができます。

パッケージ仕様部またはパッケージ本体のいずれでも、パッケージ・サブプログラムを外部のJavaサブプログラムまたはCサブプログラムにマップできますが、その際にはコール仕様を使用して、外部サブプログラム名、パラメータ型および戻り型を対応するSQLにマップします。

パッケージ仕様部のAUTHID句は、パッケージのサブプログラムとカーソルをその定義者(デフォルト)と実行者のどちらの権限で実行するか、およびスキーマ・オブジェクトへの未修飾の参照が定義者と実行者のどちらのスキーマで解決されるかを決定します。

パッケージ仕様のACCESSIBLE BY句を使用すると、パッケージにアクセスできるPL/SQLユニットのホワイト・リストを指定できます。この句は次のような状況で使います。

- PL/SQLアプリケーションを複数のパッケージとして実装します。このうち、1つのパッケージがアプリケーション・プログラミング・インタフェース(API)を提供し、ヘルパー・パッケージが処理を実行します。このとき、クライアントがAPIにアクセスするが、ヘルパー・パッケージにはアクセスしないようにします。したがって、ACCESSIBLE BY句をAPIパッケージの仕様から省略し、ヘルパー・パッケージの仕様ごとに組み込み、ここで、APIパッケージのみがヘルパー・パッケージにアクセスできるように指定します。
- 同じスキーマ内のすべてではなく一部のPL/SQLユニットにサービスを提供するユーティリティ・パッケージを作成します。パッケージの使用対象を目的のユニットに制限するには、パッケージ仕様内のACCESSIBLE BY句にこれらをリストします。

ノート:



独自のパッケージを作成する前に、[『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、必要な機能がパッケージに用意されているかどうかを確認してください。

関連項目:

- PL/SQLユニットの詳細は、[PL/SQLユニットの概要](#)を参照してください。
- PL/SQLパッケージの詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。
- パッケージを使用する理由は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。

13.3 PL/SQLユニットの概要

PL/SQLユニットは、次のいずれかです。

- 無名PL/SQLブロック
- FUNCTION
- LIBRARY
- PACKAGE
- PACKAGE BODY

- PROCEDURE
- TRIGGER
- TYPE
- TYPE BODY

PL/SQLユニットは、PL/SQLコンパイル・パラメータ(データベース初期化パラメータのカテゴリ)によって影響を受けます。異なるPL/SQLユニット(パッケージの仕様部や本体など)に、異なるコンパイル・パラメータ設定を含めることができます。

PL/SQLユニットのAUTHIDプロパティは、実行時にユニットによって発行されるSQL文の名前解決および権限チェックに影響します。

関連項目:

- PL/SQLユニットおよびコンパイル・パラメータの詳細は、『[Oracle Database PL/SQL言語リファレンス](#)』を参照してください。
- AUTHIDプロパティの詳細は、『[Oracle Database PL/SQL言語リファレンス](#)』を参照してください。

13.3.1 PLSQL_OPTIMIZE_LEVELコンパイル・パラメータ

PL/SQL最適化レベルは、PL/SQLオプティマイザがコードを再調整してパフォーマンス向上を図ることのできる程度を決定します。このレベルは、コンパイル・パラメータPLSQL_OPTIMIZE_LEVEL (デフォルト値は2)を使用して設定します。

セッションのPL/SQL最適化レベルを変更するには、SQLコマンドALTER SESSIONを使用します。セッションのレベルを変更しても、影響を受けるのは、その後で作成されたPL/SQLユニットのみです。既存のPL/SQLユニットのレベルを変更するには、ALTERコマンドをCOMPILE句とともに使用します。

1つ以上のPL/SQLユニットのPLSQL_OPTIMIZE_LEVELの現在の値を表示するには、静的データ・ディクショナリ・ビューALL_PLSQL_OBJECT_SETTINGSを使用します。

[例13-1](#) では、2つのプロシージャを作成し、これらの最適化レベルを表示し、セッションの最適化レベルを変更し、3番目のプロシージャを作成し、3つのプロシージャすべての最適化レベルを表示します。3番目のプロシージャのみが新しい最適化レベルを持ちます。このため、この例では、1つのプロシージャの最適化レベルのみを変更し、3つのプロシージャすべての最適化レベルを再表示します。

関連項目:

- PL/SQLオプティマイザの詳細は、『[Oracle Database PL/SQL言語リファレンス](#)』を参照してください
- PLSQL_OPTIMIZE_LEVELの詳細は、『[Oracle Databaseリファレンス](#)』を参照してください
- ALTER SESSIONの詳細は、『[Oracle Database SQL言語リファレンス](#)』を参照してください
- PL/SQLユニットのALTERコマンドの詳細は、『[Oracle Database PL/SQL言語リファレンス](#)』を参照してください
- ALL_PLSQL_OBJECT_SETTINGSの詳細は、『[Oracle Databaseリファレンス](#)』を参照してください

例13-1 PLSQL_OPTIMIZE_LEVELの変更

2つのプロシージャを作成します。

```
CREATE OR REPLACE PROCEDURE p1 AUTHID DEFINER AS
BEGIN
  NULL;
```

```
END;
/
CREATE OR REPLACE PROCEDURE p2 AUTHID DEFINER AS
BEGIN
    NULL;
END;
/
```

2つのプロシーダの最適化レベルを表示します。

```
SELECT NAME, PLSQL_OPTIMIZE_LEVEL
FROM USER_PLSQL_OBJECT_SETTINGS
WHERE NAME LIKE 'P%' AND TYPE='PROCEDURE'
ORDER BY NAME;
```

結果:

NAME	PLSQL_OPTIMIZE_LEVEL
P1	2
P2	2

2 rows selected.

セッションの最適化レベルを変更し、3番目のプロシーダを作成します。

```
ALTER SESSION SET PLSQL_OPTIMIZE_LEVEL=1;

CREATE OR REPLACE PROCEDURE p3 AUTHID DEFINER AS
BEGIN
    NULL;
END;
/
```

3つのプロシーダの最適化レベルを表示します。

```
SELECT NAME, PLSQL_OPTIMIZE_LEVEL
FROM USER_PLSQL_OBJECT_SETTINGS
WHERE NAME LIKE 'P%' AND TYPE='PROCEDURE'
ORDER BY NAME;
```

結果:

NAME	PLSQL_OPTIMIZE_LEVEL
P1	2
P2	2
P3	1

3 rows selected.

プロシーダp1の最適化レベルを3に変更します。

```
ALTER PROCEDURE p1 COMPILE PLSQL_OPTIMIZE_LEVEL=3;
```

3つのプロシーダの最適化レベルを表示します。

```
SELECT NAME, PLSQL_OPTIMIZE_LEVEL
FROM USER_PLSQL_OBJECT_SETTINGS
WHERE NAME LIKE 'P%' AND TYPE='PROCEDURE'
ORDER BY NAME;
```

結果:

NAME	PLSQL_OPTIMIZE_LEVEL
P1	3
P2	2
P3	1

3 rows selected.

13.4 PL/SQLサブプログラムおよびパッケージの作成

トピック:

- [サブプログラムおよびパッケージの作成に必要な権限](#)
- [サブプログラムおよびパッケージの作成](#)
- [PL/SQLオブジェクト・サイズの制限](#)
- [PL/SQLのデータ型](#)
- [クライアントへの結果セットの戻し](#)
- [ファンクションからの大量のデータの戻し](#)
- [PL/SQLファンクション結果キャッシュ](#)
- [バルク・バインドの概要](#)
- [PL/SQLの動的SQL](#)

13.4.1 サブプログラムおよびパッケージの作成に必要な権限

独自のスキーマでスタンドアロン・サブプログラムまたはパッケージを作成するには、CREATE PROCEDUREシステム権限が必要です。別のスキーマでスタンドアロン・サブプログラムまたはパッケージを作成するには、CREATE ANY PROCEDUREシステム権限が必要です。

作成するサブプログラムまたはパッケージがスキーマ・オブジェクトを参照する場合、これらのオブジェクトに必須のオブジェクト権限が必要です。これらの権限は、ロールを介してではなく、明示的に付与される必要があります。

サブプログラムまたはパッケージの所有者の権限が変更された場合、実行前にそのサブプログラムまたはパッケージを再認証する必要があります。参照オブジェクトに必要なオブジェクト権限が、そのサブプログラムまたはパッケージの所有者から取り消されている場合、そのサブプログラムは実行できません。

サブプログラムに対してEXECUTE権限を付与すると、ユーザーはサブプログラム所有者のセキュリティ・ドメインでサブプログラムを実行できるため、サブプログラムが参照するオブジェクトに対する権限をユーザーに付与する必要がなくなります。EXECUTE権限を使用すると、データベース・アプリケーションおよびそのユーザーに対してさらに統制のとれた効率的なセキュリティ計画が可能になります。また、サブプログラムおよびパッケージをデータ・ディクショナリ(SYSTEM表領域内)に格納できるようになります。ここでは、サブプログラムおよびパッケージを作成するユーザーが使用可能な領域の量が割当てによって制御されません。

関連項目:

- システム権限およびオブジェクト権限の詳細は、『Oracle Database SQL言語リファレンス』を参照してください

- [ストアドPL/SQLサブプログラムの起動](#)

13.4.2 サブプログラムおよびパッケージの作成

このトピックでは、SQLデータ定義言語(DDL)文を使用してスタンドアロン・サブプログラムおよびスタンドアロン・パッケージを作成する方法について説明します。

スタンドアロン・サブプログラムおよびスタンドアロン・パッケージを作成するためのDDL文は、次のとおりです。

- CREATE FUNCTION
- CREATE PROCEDURE
- CREATE PACKAGE
- CREATE PACKAGE BODY

パッケージの名前およびパブリック・オブジェクトの名前は、パッケージ・スキーマ内で一意である必要があります。パッケージ仕様部およびその本体は、同じ名前である必要があります。パッケージ構成の名前は、オーバーロードされたサブプログラムを除いて、パッケージの範囲内で一意である必要があります。

前述のCREATE文には、それぞれオプションのOR REPLACE句を指定できます。OR REPLACE句は、既存のPL/SQLユニットを再作成する場合(つまり、既存のPL/SQLユニットの宣言または定義の削除または再作成およびそれらに事前に付与されているオブジェクト権限の再付与を行わずに、それらの宣言または定義を変更する場合)に指定します。PL/SQLユニットを再定義すると、そのPL/SQLユニットはデータベースによって再コンパイルされます。

注意:



CREATE OR REPLACE 文は、既存の PL/SQL ユニットの置き換える前に警告を発行しません。

任意のテスト・エディタを使用して、任意の数のサブプログラムおよびパッケージを作成するためのDDL文が含まれるテキスト・ファイルを作成します。

DDL文を実行するには、SQL*Plusなどの対話形式のツールを使用します。SQL*PlusコマンドSTARTまたは@でスクリプトを実行します。たとえば、次のSQL*Plusコマンドで、スクリプトmy_app.sqlを選択します。

```
@my_app
```

また、SQL Developerを使用してDDL文を作成および実行することもできます。

関連項目:

- SQL*Plusにおけるスクリプトの実行の詳細は、[『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 Developerの詳細は、[『Oracle SQL Developerユーザーズ・ガイド』](#)を参照してください。
- 次の関数の詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。
 - CREATE FUNCTION
 - CREATE PROCEDURE
 - CREATE PACKAGE
 - CREATE PACKAGE BODY

13.4.3 PL/SQLオブジェクト・サイズの制限

サブプログラム、トリガー、パッケージなどのPL/SQLストア・データベース・オブジェクトのサイズは、共有プール内のDescriptive Intermediate Attributed Notation for Ada (DIANA)コードのサイズ(バイト単位)に制限されています。フラット化されたDIANA/codeのサイズの制限は、LinuxやUNIXでは64KBですが、デスクトップ・プラットフォームでは32KBに制限されている場合があります。

ユーザーがアクセスできるもので最も密接に関連する数値は、静的データ・ディクショナリ・ビュー*_OBJECT_SIZEの列、PARSED_SIZEです。列PARSED_SIZEには、SYS.IDL_xxx\$表に格納されたDIANAのサイズがバイト単位で示されています。これは共有プールでのサイズではありません。(コンパイル中に使用される)PL/SQLコードのDIANA部分のサイズは、システム表内より共有プール内で非常に大きくなります。

関連項目:

- PL/SQLプログラムの制限およびPARSED_SIZE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
- *_OBJECT_SIZEの詳細は、『Oracle Databaseリファレンス』を参照してください

13.4.4 PL/SQLのデータ型

このトピックでは、PL/SQLデータ型を紹介し、詳細な情報の参照先として他の章またはドキュメントを示します。

データベース・アプリケーション内のPL/SQL変数ごとに正確かつ最も固有のPL/SQLデータ型を使用してください。

関連項目:

[正確かつ最も固有のデータ型の使用](#)

トピック:

- [PL/SQLのスカラー・データ型](#)
- [PL/SQLのコンポジット・データ型](#)
- [抽象データ型](#)

13.4.4.1 PL/SQLのスカラー・データ型

スカラー・データ型には、内部コンポーネントを持たない値が格納されます。

スカラー・データ型にはサブタイプを持たせることができます。サブタイプとは、別のデータ型のサブセットとなるデータ型のことで、その別のデータ型は、そのサブタイプのベース型となります。サブタイプには、そのベース型で有効な演算と同じ演算が含まれています。データ型とそのサブタイプでデータ型ファミリが構成されます。

PL/SQLには、多くの型やサブタイプがパッケージSTANDARDに事前定義されています。また、PL/SQLを使用して、独自のサブタイプを定義することもできます。

トピック:

- [SQLデータ型](#)
- [BOOLEANデータ型](#)

- [PLS_INTEGERおよびBINARY_INTEGERデータ型](#)
- [REF CURSORデータ型](#)
- [ユーザー定義のPL/SQLサブタイプ](#)

関連項目:

- スカラーPL/SQLデータ型の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください
- データ型ファミリでグループ化された事前定義のPL/SQLのデータ型およびサブタイプは、『Oracle Database PL/SQL言語リファレンス』を参照してください

13.4.4.1.1 SQLデータ型

PL/SQLデータ型にはSQLデータ型が含まれます。

関連項目:

- データベース・アプリケーションでSQLデータ型を使用する方法の詳細は、[「データベース・アプリケーションにおけるSQLデータ型の使用」](#)を参照してください。
- SQLデータ型の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください。

13.4.4.1.2 BOOLEANデータ型

BOOLEANデータ型には、論理値(ブール値のTRUEとFALSE、およびNULL値)が格納されます。NULLは、不明な値を表します。

関連項目:

BOOLEANデータ型の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください。

13.4.4.1.3 PLS_INTEGERおよびBINARY_INTEGERデータ型

PL/SQLのデータ型であるPLS_INTEGERとBINARY_INTEGERは同じです。わかりやすくするために、このマニュアルでは、PLS_INTEGERとBINARY_INTEGERの両方を表すためにPLS_INTEGERを使用します。

PLS_INTEGERデータ型は、32ビットで表される-2147483648から2147483647の範囲の符号付き整数を格納します。

PLS_INTEGERデータ型には、NUMBERデータ型およびNUMBERサブタイプと比べて、次のようなメリットがあります。

- PLS_INTEGER値の方が、必要な記憶域が少なくなります。
- PLS_INTEGER演算はハードウェア算術計算を使用するため、ライブラリ算術計算を使用するNUMBER演算より処理速度が速くなります。

効率のために、PLS_INTEGERの範囲内でのすべての計算にPLS_INTEGER値を使用してください。

関連項目:

PLS_INTEGERデータ型の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください。

13.4.4.1.4 REF CURSORデータ型

REF CURSORは、カーソル変数のデータ型です。

カーソル変数は、明示カーソルと似ていますが、次の点が異なります。

- 1つの問合せに限定されません。

カーソル変数を問合せに対してオープンし、結果セットを処理した後、カーソル変数を別の問合せのために使用できます。

- 値を代入できます。
- 式で使用できます。
- サブプログラム・パラメータとして使用できます。

カーソル変数は、サブプログラム間で問合せ結果セットを渡すために使用できます。

- ホスト変数として使用できます。

カーソル変数は、PL/SQLのストアド・サブプログラムとそれらのクライアントとの間で問合せ結果セットを渡すために使用できます。

- パラメータを受け入れることはできません。

カーソル変数にパラメータを渡すことはできませんが、問合せ全体を渡すことができます。

カーソル変数がこのような柔軟性を持つ理由は、それがポインタであるため(その値が項目自体ではなく項目のアドレスを示すため)です。

関連項目:

REF CURSORデータ型およびカーソル変数の詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。

13.4.4.1.5 ユーザー定義のPL/SQLサブタイプ

PL/SQLではユーザー独自のサブタイプを定義できます。ベース型には、事前に定義したユーザー定義のサブタイプを含む、任意のPL/SQLスカラー型を指定できます。

サブタイプの役割は次のとおりです。

- ANSI/ISOデータ型との互換性の提供
- その型のデータ項目の使用意図の提示
- 範囲外の値の検出

関連項目:

ユーザー定義のPL/SQLサブタイプの詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。

13.4.4.2 PL/SQLのコンポジット・データ型

コンポジット・データ型には、内部コンポーネントがあります。PL/SQLのコンポジット・データ型は、コレクションおよびレコードです。

collectionの内部コンポーネントは、常に同じデータ型であり、要素と呼ばれます。コレクション変数の各要素には、一意の索

引によってアクセスできます。PL/SQLには、連想配列、VARRAY (可変サイズの配列)およびネストした表の3つのコレクション型があります。

レコードの内部コンポーネントは、データ型が異なる場合があります、フィールドと呼ばれます。レコード変数の各フィールドには、名前によってアクセスできます。

レコードのコレクション、およびコレクションを含むレコードを作成できます。

関連項目:

PL/SQLコンポジット・データ型の詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。

13.4.4.3 抽象データ型

抽象データ型(ADT)は、データ構造と、データを操作するサブプログラムで構成されています。静的データ・ディクショナリ・ビュー*_OBJECTSでは、ADTのOBJECT_TYPEはTYPEです。静的データ・ディクショナリ・ビュー*_TYPESでは、ADTのTYPECODEはOBJECTです。

関連項目:

ADT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。

13.4.5 クライアントへの結果セットの戻し

PL/SQLでは、従来のデータベース・プログラミングと同様に、カーソルを使用して問合せ結果セットを処理できます。カーソルとは、特定のSELECT文またはDML文の処理に関する情報を格納しておく、SQLのプライベート領域を指すポインタです。

ノート:



この項で説明するカーソルは、セッション・カーソルです。セッション・カーソルはセッション・メモリに存在し、セッションが終了すると消滅します。セッション・カーソルは、プログラム・グローバル領域(PGA)のSQL プライベート領域内のカーソルとは異なります。

PL/SQLで構築され管理されるカーソルは、暗黙カーソルです。ユーザーが構築および管理するカーソルは、明示カーソルです。暗黙カーソルに勝る明示カーソルの唯一のメリットは、明示カーソルを使用すると、フェッチされる行の数を制限できる点です。

カーソル変数とは、カーソルを指すポインタです。つまり、その値は、カーソル自体ではなく、カーソルのアドレスです。したがって、カーソル変数は明示カーソルより柔軟です。ただし、カーソル変数には、明示カーソルにはないデメリットもあります。

トピック:

- [カーソル変数のメリット](#)
- [カーソル変数のデメリット](#)
- [問合せ結果の暗黙的戻し](#)

関連項目:

- カーソル変数、および暗黙カーソルと明示カーソルを使用した問合せセット結果処理の一般情報は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。
- 一括収集文での行数および収集サイズを制限する方法の詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。
- OCIでのカーソル変数の使用方法の詳細は、[『Oracle Call Interfaceプログラマーズ・ガイド』](#)を参照してください
- Pro*C/C++でのカーソル変数の使用方法の詳細は、[『Pro*C/C++プログラマーズ・ガイド』](#)を参照してください
- Pro*COBOLでのカーソル変数の使用方法の詳細は、[『Pro*COBOLプログラマーズ・ガイド』](#)を参照してください
- JDBCでのカーソル変数の使用方法の詳細は、[『Oracle Database JDBC開発者ガイド』](#)を参照してください
- [ファンクションからの大量のデータの戻し](#)
- PGAの詳細は、[『Oracle Database概要』](#)を参照してください。

13.4.5.1 カーソル変数のメリット

カーソル変数は、明示カーソルと似ていますが、次の点が異なります。

- 1つの問合せに限定されません。
カーソル変数を問合せに対してオープンし、結果セットを処理した後、カーソル変数を別の問合せのために使用できます。
- 値を代入できます。
- 式で使用できます。
- サブプログラム・パラメータとして使用できます。
カーソル変数は、サブプログラム間で問合せ結果セットを渡すために使用できます。
- ホスト変数として使用できます。
カーソル変数は、PL/SQLのストアド・サブプログラムとそれらのクライアントとの間で問合せ結果セットを渡すために使用できます。
- パラメータを受け入れることはできません。
カーソル変数にパラメータを渡すことはできませんが、問合せ全体を渡すことができます。問合せには変数を組み込むことができます。

前述の特性のため、カーソル変数には次のメリットがあります。

- カプセル化
カーソル変数をオープンするストアド・サブプログラムに問合せを集中化できます。
- メンテナンスの容易さ
カーソルの変更が必要な場合は、ストアド・サブプログラムの変更のみで済みます。ストアド・サブプログラムを起動するすべてのアプリケーションで変更を行う必要はありません。
- セキュリティの利便性
アプリケーションは、アプリケーション・ユーザーのユーザー名を持つサーバーに接続します。アプリケーション・ユーザーには、カーソルをオープンするストアド・サブプログラムに対するEXECUTE権限が必要ですが、問い合わせた表に対するREAD権

限は必要ありません。

13.4.5.2 カーソル変数のデメリット

カーソル変数を使用する必要がない場合、パフォーマンスおよびプログラミングの容易性を向上するために、暗黙または明示カーソルを使用してください。

トピック:

- [カーソル変数の解析のペナルティ](#)
- [カーソル変数の複数行フェッチのペナルティ](#)



ノート:

これらのトピックの例には、TKPROF レポートが含まれます。

関連項目:

TKPROFレポートの生成手順は、[『Oracle Database SQLチューニング・ガイド』](#)を参照してください。

13.4.5.2.1 カーソル変数の解析のペナルティ

明示カーソルを閉じた場合、カーソルはパースペクティブから閉じられますが(つまり、開いているカーソルが必要な場合にそのカーソルを使用できない)、PL/SQLは明示カーソルをオープン状態でキャッシュします。カーソルに関連付けられた文を再実行すると、PL/SQLはキャッシュされたカーソルを使用することにより、解析を回避します。

解析の回避により、CPUの使用率が大幅に削減され、明示カーソルのキャッシュが明白になります。この場合、プログラミングは影響を受けません。PL/SQLは使用可能なオープン・カーソルの供給を削減しません。プログラムが他のカーソルを開く必要があるが、開くとOPEN_CURSORSのinit.ora設定を超えるような場合、PL/SQLはキャッシュされたカーソルを閉じます。

PL/SQLはオープン状態のカーソル変数はキャッシュできません。したがって、カーソル変数には解析のペナルティがあります。

[例13-2](#)では、プロシージャは明示カーソルをオープンし、ここからフェッチし、これをクローズし、カーソル変数でも同じ処理を行います。匿名ブロックがプロシージャを10回コールします。TKPROFレポートには、両方の問合せが10回実行されたが、明示カーソルに関連付けられた問合せが解析されたのは1回のみで、カーソル変数に関連付けられた問合せが解析されたのが10回であることが示されます。

例13-2 カーソル変数の解析のペナルティ

```
CREATE OR REPLACE PROCEDURE p AUTHID DEFINER IS
  CURSOR e_c IS SELECT * FROM DUAL d1;  -- explicit cursor
  c_v SYS_REFCURSOR;                    -- cursor variable
  rec DUAL%ROWTYPE;
BEGIN
  OPEN e_c;                              -- explicit cursor
  FETCH e_c INTO rec;
  CLOSE e_c;
  OPEN c_v FOR SELECT * FROM DUAL d2;    -- cursor variable
  FETCH c_v INTO rec;
  CLOSE c_v;
END;
/
BEGIN
  FOR i IN 1..10 LOOP                    -- execute p 10 times
    p;
```

```
END LOOP;
```

TKPROFレポートは、次のようになります。

```
SELECT * FROM DUAL D1;
```

```
call      count
```

```
-----
```

```
Parse      1
```

```
Execute    10
```

```
Fetch      10
```

```
-----
```

```
total      21
```

```
*****
```

```
SELECT * FROM DUAL D2;
```

```
call      count
```

```
-----
```

```
Parse      10
```

```
Execute    10
```

```
Fetch      10
```

```
-----
```

```
total      30
```

13.4.5.2.2 カーソル変数の複数行フェッチのペナルティ

[例13-3](#)では、7,000行以上の表を作成し、これらすべての行を2回フェッチします。この場合、初回は暗黙カーソル(配列のフェッチ)を使用し、2回目はカーソル変数(個別行のフェッチ)を使用します。暗黙カーソルのコードは、カーソル変数のコードより単純であり、TKPROFレポートには、暗黙カーソルのコードの方がパフォーマンスもよいことが示されます。

カーソル変数を使用して配列をフェッチすることもできますが、より多くのコードが必要になります。特に、次の処理を行うためのコードが必要になります。

- 配列のフェッチ先のコレクションのタイプの定義
- コレクションに対する明示的な一括収集
- フェッチしたデータを処理するためのコレクションのループ
- 明示的にオープンしたカーソル変数のクローズ

例13-3 カーソル変数の配列フェッチのペナルティ

問い合わせる表を作成し、その行数を表示します。

```
CREATE TABLE t AS
  SELECT * FROM ALL_OBJECTS;
```

```
SELECT COUNT(*) FROM t;
```

結果は次のようになります。

```
COUNT(*)
```

```
-----
```

```
70788
```

暗黙カーソルとカーソル変数で同等の操作を実行します。

```
DECLARE
  c_v SYS_REFCURSOR;
  rec t%ROWTYPE;
```



```

BEGIN
  FOR x IN (SELECT * FROM t exp_cur) LOOP  -- implicit cursor
    NULL;
  END LOOP;

  OPEN c_v FOR SELECT * FROM t cur_var;    -- cursor variable

  LOOP
    FETCH c_v INTO rec;
    EXIT WHEN c_v%NOTFOUND;
  END LOOP;

  CLOSE c_v;
END;
/

```

TKPROFレポートは、次のようになります。

SELECT * FROM T EXP_CUR							
call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.00	0.00	0	0	0	0
Execute	1	0.00	0.00	0	0	0	0
Fetch	722	0.23	0.23	0	1748	0	72198

total	724	0.23	0.23	0	1748	0	72198

SELECT * FROM T CUR_VAR							
call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.00	0.00	0	0	0	0
Execute	1	0.00	0.00	0	0	0	0
Fetch	72199	0.40	0.42	0	72203	0	72198

total	72201	0.40	0.42	0	72203	0	72198

13.4.5.3 問合せ結果の暗黙的戻し

ストアド・サブプログラムは、DBMS_SQL.RETURN_RESULTプロシージャを起動することにより、問合せ結果を暗黙的にクライアント・プログラムまたはサブプログラムの直接のコール元に戻すことができます。DBMS_SQL.RETURN_RESULTによって結果が戻された後、受信者のみがこれにアクセスできます。

ノート:



動的 SQL を使用して実行された問合せの結果を暗黙的に戻すには、サブプログラムは、EXECUTE IMMEDIATE 文ではなく DBMS_SQL プロシージャを使用して問合せを実行する必要があります。この理由は、EXECUTE IMMEDIATE 文がサブプログラムに戻るカーソルは、EXECUTE IMMEDIATE 文が完了するときにクローズされるためです。

関連項目:

- DBMS_SQL.RETURN_RESULTプロシージャの詳細は、『[Oracle Database PL/SQL言語リファレンス](#)』を参照してください。
- 動的SQLのDBMS_SQLプロシージャの使用の詳細は、『[Oracle Database PL/SQL言語リファレンス](#)』を参照してください。

13.4.6 ファンクションからの大量のデータの戻し

データ・ウェアハウス環境では、大量のデータを変換するためにPL/SQLファンクションを使用します。データは、異なるファンクションによる一連の変換を経由して渡す場合があります。PL/SQL表ファンクションを使用すると、このような変換を、かなりのメモリ・オーバーヘッドを必要としたり、各変換の間でデータを表に格納する必要なく実行できます。これらのファンクションは、複数の行を受け入れて戻すことが可能で、一度ではなく準備できた順に行を戻すことができるだけでなく、パラレル化も可能です。

関連項目:

パイプライン・テーブル・ファンクションを使用して複数の変換を実行する方法の詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。

13.4.7 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を使用すると、領域と時間を大幅に節約できます。結果キャッシュPL/SQLファンクションを、異なるパラメータ値を使用して起動するたびに、それらのパラメータおよびその結果がキャッシュに格納されます。それ以降、同じファンクションが同じパラメータ値で起動されると、結果は再計算されるのではなく、キャッシュから取り出されます。キャッシュは共有グローバル領域(SGA)に格納されるため、アプリケーションが実行されるすべてのセッションで使用可能です。

キャッシュ結果を計算する際に使用したデータベース・オブジェクトが更新されると、そのキャッシュ結果は無効になり、再計算が必要になります。

結果キャッシュの対象として最良のファンクションは、頻繁に起動され、ほとんどまたはまったく変更されない情報に依存するファンクションです。

関連項目:

PL/SQLファンクション結果キャッシュの詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。

13.4.8 バルク・バインドの概要

Oracle Databaseは2つのエンジンを使用して、PL/SQLユニットを実行します。PL/SQLエンジンは手続き型の文を実行し、SQLエンジンはSQL文を実行します。すべてのSQL文により、2つのエンジン間でコンテキストの切替えが行われます。PL/SQLユニットごとに行われるコンテキストの切替えの回数を最小限に抑えることにより、データベース・アプリケーションのパフォーマンスを大幅に向上させることができます。

バインド変数としてコレクション要素を使用するループ内でSQL文が実行される場合、必要とする多数のコンテキストのスイッチングによってパフォーマンスが低下することがあります。コレクションには次のものが含まれます。

- 連想配列
- 可変サイズの配列
- ネストした表
- ホスト配列

バインドとは、SQL文内のPL/SQL変数に対して値を代入することです。**バルク・バインド**とは、コレクション全体を一度にバインドすることです。バルク・バインドは1つの操作でコレクション全体を2つのエンジン間で受け渡すことができます。

通常、バルク・バインドによって、4つ以上のデータベース行に影響するSQL文のパフォーマンスが改善されます。SQL文によって

影響される行数が多いほど、バルク・バインドによるパフォーマンスの向上率は高くなります。コレクションを参照するDMLおよびSELECT INTO文や、コレクションを参照してDMLを戻すFORループのパフォーマンスを向上させるためにバルク・バインドを使用することを検討してください。



ノート:

バルク・バインドを使用する場合、パラレル DML 文は無効になります。

トピック:

- [コレクションを参照するDML文](#)
- [コレクションを参照するSELECT文](#)
- [コレクションを参照しDMLを戻すFORループ](#)

関連項目:

- バルク・バインド操作中に発生する例外の処理方法を含むバルク・バインドの詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください
- 並列DML文の詳細は、『[Oracle Database PL/SQL言語リファレンス](#)』を参照してください。

13.4.8.1 コレクションを参照するDML文

FORALLキーワードを使用するバルク・バインドによって、コレクション要素を参照するINSERT、UPDATEまたはDELETE文のパフォーマンスが向上します。

[例13-4](#)のPL/SQLブロックは、管理職のID番号が7902、7698または7839の従業員について昇給を行うものですが、バルク・バインドを使用する場合と使用しない場合があります。PL/SQLがバルク・バインドを使用しないで、更新する各従業員についてSQLエンジンにSQL文を送信すると、コンテキストのスイッチングによってパフォーマンスが低下します。

関連項目:

FORALL文の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください

例13-4 コレクションを参照するDML文

```
DECLARE
  TYPE numlist IS VARRAY (100) OF NUMBER;
  id NUMLIST := NUMLIST(7902, 7698, 7839);
BEGIN
  -- Efficient method, using bulk bind:

  FORALL i IN id.FIRST..id.LAST
    UPDATE EMPLOYEES
    SET SALARY = 1.1 * SALARY
    WHERE MANAGER_ID = id(i);

  -- Slower method:

  FOR i IN id.FIRST..id.LAST LOOP
    UPDATE EMPLOYEES
    SET SALARY = 1.1 * SALARY
    WHERE MANAGER_ID = id(i);
```

```
END LOOP;
END;
/
```

13.4.8.2 コレクションを参照するSELECT文

BULK COLLECT句によって、コレクションを参照する問合せのパフォーマンスを改善できます。スカラー値表または%TYPE値表にBULK COLLECTを使用できます。

関連項目:

BULK COLLECT句の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください

[例13-5](#)のPL/SQLブロックは、複数の値の問合せを行ってその値をPL/SQL表に格納するものですが、バルク・バインドを使用する場合と使用しない場合があります。PL/SQLがバルク・バインドを使用しないで、選択された各従業員についてSQLエンジンにSQL文を送信すると、コンテキストのスイッチングによってパフォーマンスが低下します。

例13-5 コレクションを参照するSELECT文

```
DECLARE
  TYPE var_tab IS TABLE OF VARCHAR2(20)
    INDEX BY PLS_INTEGER;

  empno    VAR_TAB;
  ename    VAR_TAB;
  counter  NUMBER;

  CURSOR c IS
    SELECT EMPLOYEE_ID, LAST_NAME
    FROM EMPLOYEES
    WHERE MANAGER_ID = 7698;
BEGIN
  -- Efficient method, using bulk bind:

  SELECT EMPLOYEE_ID, LAST_NAME BULK COLLECT
  INTO empno, ename
  FROM EMPLOYEES
  WHERE MANAGER_ID = 7698;

  -- Slower method:

  counter := 1;

  FOR rec IN c LOOP
    empno(counter) := rec.EMPLOYEE_ID;
    ename(counter) := rec.LAST_NAME;
    counter := counter + 1;
  END LOOP;
END;
/
```

13.4.8.3 コレクションを参照しDMLを戻すFORループ

BULK COLLECTキーワードとともにFORALLキーワードを使用すると、コレクションを参照しDMLを戻すFORループのパフォーマンスを改善できます。

関連項目:

BULK COLLECT句とともにRETURNING INTOを使用する方法の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください

[例13-6](#)のPL/SQLブロックは、従業員コレクションのボーナスを計算し、EMPLOYEES表を更新します。さらに、ボーナスをbonus_list_inst列に戻します。この操作をバルク・バインドを使用して実行し、また使用せずに実行します。PL/SQLがバルク・バインドを使用しないで、更新する各従業員についてSQLエンジンにSQL文を送信すると、コンテキストのスイッチングによってパフォーマンスが低下します。

例13-6 コレクションを参照しDMLを返すFORループ

```
DECLARE
  TYPE emp_list IS VARRAY(100) OF EMPLOYEES.EMPLOYEE_ID%TYPE;
  empids emp_list := emp_list(182, 187, 193, 200, 204, 206);

  TYPE bonus_list IS TABLE OF EMPLOYEES.SALARY%TYPE;
  bonus_list_inst bonus_list;
BEGIN
  -- Efficient method, using bulk bind:

  FORALL i IN empids.FIRST..empids.LAST
    UPDATE EMPLOYEES
    SET SALARY = 0.1 * SALARY
    WHERE EMPLOYEE_ID = empids(i)
    RETURNING SALARY BULK COLLECT INTO bonus_list_inst;

  -- Slower method:

  FOR i IN empids.FIRST..empids.LAST LOOP
    UPDATE EMPLOYEES
    SET SALARY = 0.1 * SALARY
    WHERE EMPLOYEE_ID = empids(i)
    RETURNING SALARY INTO bonus_list_inst(i);
  END LOOP;
END;
/
```

13.4.9 PL/SQLの動的SQL

動的SQLは、実行時にSQL文を生成して実行するためのプログラミング方法です。この方法は、非定型の問合せシステムのような柔軟性がある汎用目的のプログラムを記述する場合、データベース定義言語(DDL)文を実行する必要があるプログラムを記述する場合、またはコンパイル時にSQL文のテキスト全体またはそのSQL文の入力変数および出力変数の数またはデータ型が不明な場合に有効です。

動的SQLが必要ない場合は、静的SQLを使用します。静的SQLには、次のようなメリットがあります。

- コンパイルが正常に完了すると、静的SQL文が有効なデータベース・オブジェクトを参照していること、およびそれらのオブジェクトへのアクセスに必要な権限が存在していることが保証されます。
- コンパイルが正常に完了すると、スキーマ・オブジェクトの依存性が作成されます。

関連項目:

- スキーマ・オブジェクトの依存性の詳細は、[「スキーマ・オブジェクトの依存性の理解」](#)を参照してください。
- 動的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。
- 静的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。

13.5 PL/SQLサブプログラムおよびパッケージの変更

ストアド・スタンダロン・サブプログラムまたはパッケージの名前を変更するには、これを削除してから、新しい名前で作成する必要があります。たとえば：

```
CREATE PROCEDURE p IS BEGIN NULL; END;  
/  
DROP PROCEDURE p  
/  
CREATE PROCEDURE p1 IS BEGIN NULL; END;  
/
```

ストアド・スタンダロン・サブプログラムまたはパッケージを名前を変更せずに変更するには、CREATE文にOR REPLACEを組み込むことにより、同じ名前でこれを新しいバージョンに置き換えます。たとえば：

```
CREATE OR REPLACE PROCEDURE p1 IS  
BEGIN  
    DBMS_OUTPUT.PUT_LINE('Hello, world!');  
END;  
/
```

ノート：



ALTER 文(ALTER FUNCTION、ALTER PROCEDURE および ALTER PACKAGE など)の場合、既存の PL/SQL ユニットの宣言または定義は変更されず、ユニットの再コンパイルのみが行われます。

関連項目：

- [PL/SQLサブプログラムおよびパッケージの削除](#)
- ALTER文の詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。

13.6 推奨されないパッケージ、サブプログラムおよびタイプ

Oracle Database 12c リリース2 (12.2)では、DEPRECATEプラグマを使用して、新しいインタフェースで非推奨または置き換えられたパッケージ、サブプログラムまたはタイプと通信できます。非推奨の要素を参照するユニットがコンパイルされると、コンパイルの警告が発行されます。PL/SQLユニットを非推奨としてマークするには、DEPRECATEプラグマを使用します。

関連項目：

DEPRECATEプラグマの詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。

13.7 PL/SQLサブプログラムおよびパッケージの削除

ストアド・スタンダロン・サブプログラムを削除するには、次の文を使用します。

- DROP FUNCTION
- DROP PROCEDURE

パッケージ(仕様部および本体)、またはその本体のみを削除するには、DROP PACKAGE文を使用します。

関連項目:

- DROP FUNCTION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。
- DROP PROCEDURE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。
- [Oracle Database PL/SQL言語リファレンス](#) DROP PACKAGE

13.8 システム固有の実行のためのPL/SQLユニットのコンパイル

通常、PL/SQLユニットをコンパイルして、システム固有のコード(プロセッサに依存するシステム・コード)にすると、PL/SQLユニット(独自のユニットまたはOracleが提供するユニット)をスピードアップできます。システム固有のコードは、SYSTEM表領域に格納されます。

ネイティブ・コードにコンパイルされたPL/SQLユニットは、共有サーバー構成(以前のマルチスレッド・サーバー)やOracle Real Application Clusters (Oracle RAC)などのすべてのサーバー環境で動作します。

PL/SQLユニットをネイティブ・コードでコンパイルするかどうかは、開発サイクルの段階およびPL/SQLユニットの内容によって決まります。

ノート:



システム固有の実行のために Java パッケージおよびクラスをコンパイルするには、ncomp ツールを使用してください。

関連項目:

- システム固有の実行のためのPL/SQLユニットのコンパイルの詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。
- [Oracle Database Java開発者ガイド](#)

13.9 ストアドPL/SQLサブプログラムの起動

ストアドPL/SQLサブプログラムは、次のように様々な環境から起動できます。たとえば:

- Oracle Databaseのツール製品を使用した対話形式による起動
- 別のサブプログラム本体からの起動
- トリガー本体からの起動
- アプリケーション(SQL*Formsやプリコンパイラなど)の内部からの起動

ストアドPL/SQLファンクション(プロシージャではない)は、SQL文から起動することもできます。

別のユーザーが所有するサブプログラムを起動する場合は、次のようになります。

- 起動の際に所有者の名前を指定する必要があります。たとえば:

```
EXECUTE jdoe.Fire_emp (1043);  
EXECUTE jdoe.Hire_fire.Fire_emp (1043);
```


- サブプログラムのAUTHIDプロパティは、実行時にサブプログラムによって発行されるSQL文の名前解決および権限チェックに影響します。

トピック:

- [ストアド・サブプログラムの起動に必要な権限](#)
- [Oracleツールを使用した対話形式によるサブプログラムの起動](#)
- [別のサブプログラムからのサブプログラムの起動](#)
- [リモート・サブプログラムの起動](#)

関連項目:

- AUTHIDプロパティ、サブプログラムの起動、パラメータ、および定義者と起動者の権限の詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。
- トリガー本体のコーディングの詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。
- [SQL文からのストアドPL/SQLファンクションの起動](#)
- OCIアプリケーションからのPL/SQLサブプログラムの起動方法の詳細は、『Oracle Call Interfaceプログラマーズ・ガイド』を参照してください
- Pro*C/C++からのPL/SQLサブプログラムの起動の詳細は、『Pro*C/C++プログラマーズ・ガイド』を参照してください
- Pro*COBOLからのPL/SQLサブプログラムの起動の詳細は、『Pro*COBOLプログラマーズ・ガイド』を参照してください
- JDBCアプリケーションからのPL/SQLサブプログラムの起動方法の詳細は、『Oracle Database JDBC開発者ガイド』を参照してください

13.9.1 ストアド・サブプログラムの起動に必要な権限

次のようなサブプログラムを起動する場合、権限は必要ありません。

- 所有しているスタンドアロン・サブプログラム
- 所有しているパッケージ内のサブプログラム
- パブリック・スタンドアロン・サブプログラム
- パブリック・パッケージ内のサブプログラム

別のユーザーが所有するストアド・サブプログラムを起動するには、スタンドアロン・サブプログラムまたはパッケージ・サブプログラムを含むパッケージのEXECUTE権限、またはEXECUTE ANY PROCEDUREシステム権限が必要です。サブプログラムがリモートにある場合は、EXECUTE権限またはEXECUTE ANY PROCEDUREシステム権限が、ロールを介してではなく直接付与されている必要があります。

関連項目:

システム権限およびオブジェクト権限の詳細は、『Oracle Database SQL言語リファレンス』を参照してください

13.9.2 Oracleツールを使用した対話形式によるサブプログラムの起動

サブプログラムは、SQL*Plusなど、Oracle Databaseのツール製品から対話形式で起動できます。

関連項目:

- EXECUTEコマンドの詳細は、『[SQL*Plusユーザズ・ガイドおよびリファレンス](#)』を参照してください。
- 開発ツールを使用して同様の操作を実行する詳細は、ご使用のツール製品のドキュメントを参照してください。

[例13-7](#)では、SQL*Plusを使用してプロシージャを作成し、2つの方法で起動します。

一部の対話形式ツールでは、セッション中に使用できるセッション変数を作成することができます。[例13-8](#)では、SQL*Plusを使用してセッション変数の作成、使用および出力を行います。

例13-7 SQL*Plusを使用した対話形式によるサブプログラムの起動

```
CREATE OR REPLACE PROCEDURE salary_raise (  
    employee EMPLOYEES.EMPLOYEE_ID%TYPE,  
    increase EMPLOYEES.SALARY%TYPE  
)  
IS  
BEGIN  
    UPDATE EMPLOYEES  
    SET SALARY = SALARY + increase  
    WHERE EMPLOYEE_ID = employee;  
END;  
/
```

無名ブロックの内部からプロシージャを起動します。

```
BEGIN  
    salary_raise(205, 200);  
END;  
/
```

結果:

```
PL/SQL procedure successfully completed.
```

EXECUTE文でプロシージャを起動します。

```
EXECUTE salary_raise(205, 200);
```

結果:

```
PL/SQL procedure successfully completed.
```

例13-8 SQL*Plusを使用したセッション変数の作成と使用

```
-- Create function for later use:  
CREATE OR REPLACE FUNCTION get_job_id (  
    emp_id EMPLOYEES.EMPLOYEE_ID%TYPE  
) RETURN EMPLOYEES.JOB_ID%TYPE  
IS  
    job_id EMPLOYEES.JOB_ID%TYPE;  
BEGIN  
    SELECT JOB_ID INTO job_id
```

```

FROM EMPLOYEES
WHERE EMPLOYEE_ID = emp_id;

RETURN job_id;
END;
/
-- Create session variable:

VARIABLE job VARCHAR2(10);

-- Run function and store returned value in session variable:

EXECUTE :job := get_job_id(204);

PL/SQL procedure successfully completed.

```

SQL*Plusコマンド:

```
PRINT job;
```

結果:

```

JOB
-----
PR_REP

```

13.9.3 別のサブプログラムからのサブプログラムの起動

ストアド・サブプログラムは、別のサブプログラムやトリガーから起動できます。[例13-9](#)では、プロシージャprint_mgr_nameがプロシージャprint_emp_nameを起動します。

サブプログラムの再帰的な起動が可能です(つまり、サブプログラムがサブプログラム自体を起動できます)。

例13-9 別のサブプログラムからのサブプログラムの起動

```

-- Create procedure that takes employee's ID and prints employee's name:

CREATE OR REPLACE PROCEDURE print_emp_name (
    emp_id EMPLOYEES.EMPLOYEE_ID%TYPE
)
IS
    fname EMPLOYEES.FIRST_NAME%TYPE;
    lname EMPLOYEES.LAST_NAME%TYPE;
BEGIN
    SELECT FIRST_NAME, LAST_NAME
    INTO fname, lname
    FROM EMPLOYEES
    WHERE EMPLOYEE_ID = emp_id;

    DBMS_OUTPUT.PUT_LINE (
        'Employee #' || emp_id || ': ' || fname || ' ' || lname
    );
END;
/

-- Create procedure that takes employee's ID and prints manager's name:

CREATE OR REPLACE PROCEDURE print_mgr_name (
    emp_id EMPLOYEES.EMPLOYEE_ID%TYPE
)
IS
    mgr_id EMPLOYEES.MANAGER_ID%TYPE;

```

```

BEGIN
  SELECT MANAGER_ID
  INTO mgr_id
  FROM EMPLOYEES
  WHERE EMPLOYEE_ID = emp_id;

  DBMS_OUTPUT.PUT_LINE (
    'Manager of employee #' || emp_id || ' is: '
  );

  print_emp_name(mgr_id);
END;
/

```

プロシージャを起動します。

```

BEGIN
  print_emp_name(200);
  print_mgr_name(200);
END;
/

```

結果:

```

Employee #200:  Jennifer Whalen
Manager of employee #200 is:
Employee #101:  Neena Kochhar

```

13.9.4 リモート・サブプログラムの起動

リモート・サブプログラムは、起動者とは別のデータベースに格納されます。リモート・サブプログラムの起動には、サブプログラム名、サブプログラムが格納されているデータベースへのデータベース・リンク、およびすべての仮パラメータの実パラメータ(仮パラメータにデフォルト値がある場合でも)が含まれる必要があります。

たとえば、次のSQL*Plus文はストアド・スタンドアロン・プロシージャfire_emp1を起動します。このプロシージャは、boston_serverという名前のローカル・データベース・リンクによって参照されています。

```
EXECUTE fire_emp1@boston_server(1043);
```

ノート:



リモート・パッケージ・サブプログラムは起動できますが、リモート・パッケージの変数および定数には直接アクセスできません。

注意:



- リモート・サブプログラムの起動では、実行時バインディングが使用されます。接続するユーザー・アカウントは、データベース・リンクに依存します。(ストアド・サブプログラムではコンパイル時バインディングが使用されます。)
- ローカル・サブプログラムがリモート・サブプログラムを起動する場合、ローカル・サブプログラムの実行中にタイ

ムスタンプの不一致が検出されると、リモート・サブプログラムは実行されず、ローカル・サブプログラムが無効になります。詳細は、[「ローカル・データベース・プロシージャとリモート・データベース・プロシージャの間の依存性」](#)を参照してください。

トピック:

- [リモート・サブプログラムのシノニム](#)
- [リモート・サブプログラムを起動するトランザクション](#)

関連項目:

- [ローカル・データベース・プロシージャとリモート・データベース・プロシージャの間の依存性](#)
- サブプログラムのエラー処理の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください

13.9.4.1 リモート・サブプログラムのシノニム

シノニムとは、スキーマ・オブジェクトの別名です。リモート・サブプログラム名とデータベース・リンクのシノニムを作成し、シノニムを使用してサブプログラムを起動することができます。たとえば:

```
CREATE SYNONYM synonym1 for fire_emp1@boston_server;  
EXECUTE synonym1(1043);
```

ノート:



パッケージ・サブプログラムのシノニムを作成することはできません。これは、パッケージ・サブプログラムはスキーマ・オブジェクトではないためです(そのパッケージはスキーマ・オブジェクトです)。

シノニムによってデータの独立性と場所の透過性の両方が実現されます。シノニムを使用すると、サブプログラムの所有者や場所を知らなくてもサブプログラムを起動できます。ただし、シノニムは権限の代替物ではありません。シノニムを使用してサブプログラムを起動するには、サブプログラムに必須の権限が必要です。

シノニムに権限を付与することは、基本オブジェクトに権限を付与することと同じです。同様に、基本オブジェクトに対して権限を付与することは、オブジェクトのすべてのシノニムに権限を付与することと同じです。

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・シノニムは自分のスキーマ内にあり、他のユーザーによるシノニムの使用を自分で制御できます。パブリック・シノニムはユーザー・グループPUBLICに属しており、すべてのデータベース・ユーザーがアクセスできます。

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。

シノニムを使用しない場合は、リモート・サブプログラムを起動するローカル・サブプログラムを作成することもできます。たとえば:

```
CREATE OR REPLACE PROCEDURE local_procedure  
  (arg IN NUMBER)  
AS  
BEGIN  
  fire_emp1@boston_server(arg);  
END;  
/  
DECLARE  
  arg NUMBER;  
BEGIN
```

```
local_procedure(arg);  
END;  
/
```

関連項目:

- シノニムの詳細は、[『Oracle Database概要』](#)を参照してください。
- パブリック・シノニムの例は、[『Oracle Database概要』](#)を参照してください。
- CREATE SYNONYM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。
- GRANT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。

13.9.4.2 リモート・サブプログラムを起動するトランザクション

リモート・サブプログラムの起動により、データベースが更新されることが想定されます。したがって、リモート・サブプログラムを起動するトランザクションには、(リモート・サブプログラムによってデータベースが更新されない場合でも) 2フェーズ・コミットが必要です。トランザクションがロールバックされると、リモート・サブプログラムによって実行された処理もロールバックされます。

文COMMIT、ROLLBACKおよびSAVEPOINTに関しては、リモート・サブプログラムは次の点においてローカル・サブプログラムと異なります。

- トランザクションがOracle Database以外のデータベースで開始された場合、リモート・サブプログラムでこれらの文は実行できません。

このような状況になるのはOracle XAアプリケーションなどの場合であり、このアプリケーションを使用することはお薦めしません。詳細は、[『Oracle XAを使用したアプリケーションの開発』](#)を参照してください。
- これらの文の1つを実行した後、リモート・サブプログラムは独自の分散トランザクションを開始できなくなります。

分散トランザクションでは複数のデータベースが更新されます。トランザクション内の文は別のデータベースに送信され、トランザクションはユニットとして成功または失敗します。トランザクションが任意のトランザクションで失敗した場合、このトランザクションをすべてのデータベースで(セーブポイントまで、または完全に)ロールバックする必要があります。分散更新を実行するサブプログラムを作成する場合は、この点を考慮する必要があります。
- リモート・サブプログラムがその作業をコミットまたはロールバックしない場合、データベース・リンクがクローズした時点で作業が暗黙的にコミットされます。それまでは、リモート・サブプログラムがトランザクションを実行しているとみなされます。したがって、リモート・サブプログラムに対するこれ以上の起動は許可されません。

13.10 SQL文からのストアドPL/SQLファクションの起動

注意:

SQL は命令型(または手続き型)言語ではなく宣言型言語であるため、ファンクションが命令型言語の PL/SQL で記述されていても、SQL 文によって起動されるファンクションが何回実行されるかはわかりません。

アプリケーションでファンクションが特定の回数実行されることが必要な場合には、SQL 文からファンクションを起動しないでください。かわりにカーソルを使用します。

たとえば、選択した行ごとにファンクションをコールすることがアプリケーションで必要な場合は、カーソルをオープンして、

カーソルから行を選択し、各行に対してファンクションをコールします。この技術により、ファンクションをコールする回数が、カーソルからフェッチされる行数と同じになります。

カーソルの詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

PL/SQLストアド・ファンクションを起動できるSQL文は次のとおりです。

- INSERT
- UPDATE
- DELETE
- SELECT

(SELECTは、WITH句で宣言および定義されているPL/SQLファンクションを起動することもできます。

- CALL

(CALLは、PL/SQLストアド・プロシージャを起動することもできます。)

PL/SQLファンクションをSQL文から起動するには、ファンクションに対するEXECUTE権限を所有している必要があります。PL/SQLファンクションで定義されたビューを検索するには、そのビューのREADまたはSELECT権限が必要です。ビューを検索するには個々のEXECUTE権限は必要ありません。

ノート:



AUTHID はユニットが実行時に発行する SQL 文の名前の解決と権限チェックに影響するため、PL/SQL ファンクションの AUTHID プロパティは、SQL 文からファンクションを呼び出すのに必要な権限にも影響します。詳細は、[「Oracle Database PL/SQL 言語リファレンス」](#)を参照してください。

トピック:

- [SQL文からPL/SQLファンクションを起動する理由](#)
- [SQL文の中でPL/SQLファンクションを使用できる箇所](#)
- [SQL式の中でPL/SQLファンクションを使用できる条件](#)
- [SQL文から起動したPL/SQLファンクションの副作用の制御](#)

関連項目:

- [SELECT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。](#)
- パラメータの受渡しも含めた、サブプログラムの起動の一般情報は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。

13.10.1 SQL文からPL/SQLファンクションを起動する理由

SQL文でPL/SQLファンクションを起動することにより、次の効果があります。

- SQLの拡張によって、ユーザーの生産性が向上します。

実行する内容がSQL文のみで表現するには複雑すぎたり、非常に扱いにくかったり、不可能な場合に、SQL文の表

現機能が強化されます。

- 問合せの効率を向上します。

問合せのWHERE句にファンクションを指定すると、条件を使用してデータをフィルタできます。ファンクションを使用できない場合は、アプリケーションで評価する必要があります。

- 特殊なデータ型(緯度、経度、温度など)を表すための文字列を操作できます
- パラレル問合せを実行できます。

問合せがパラレル化されると、PL/SQLサブプログラム内のSQL文も(パラレル問合せオプションを使用して)パラレルに実行できます。

13.10.2 SQL文の中でPL/SQLファンクションを使用できる箇所

SQL文では、PL/SQLファンクションや式を使用できる任意の箇所でPL/SQLファンクションを使用できます。たとえば:

- SELECT文の選択リスト
- WHERE句またはHAVING句の条件
- CONNECT BY句、START WITH句、ORDER BY句またはGROUP BY句
- INSERT文のVALUES句
- UPDATE文のSET句

SELECT文では、次のかわりにPL/SQL表ファンクション(複数の行をまとめて戻す)を使用できます。

- SELECTリストの列名
- FROM句の表名

不変の定義が必要な次のようなコンテキストには、PL/SQLファンクションは使用できません。

- CREATE文またはALTER TABLE文のCHECK制約句
- 列に対するデフォルト値の指定

13.10.3 SQL式の中でPL/SQLファンクションを使用できる条件

PL/SQLファンクションをSQL式から起動するには、そのファンクションが次の要件を満たしている必要があります。

- ユーザー定義の集計ファンクションまたは行ファンクションであること。
- 仮パラメータが、OUTパラメータやIN OUTパラメータではなく、INパラメータであること。

[例13-10](#)のファンクションは、これらの要件を満たしています。

例13-10 SQL式でのPL/SQLファンクション(規則に従った場合)

```
DROP TABLE payroll; -- in case it exists
CREATE TABLE payroll (
  srate NUMBER,
  orate NUMBER,
  acctno NUMBER
);

CREATE OR REPLACE FUNCTION gross_pay (
  emp_id IN NUMBER,
  st_hrs IN NUMBER := 40,
```

```

    ot_hrs IN NUMBER := 0
) RETURN NUMBER
IS
    st_rate NUMBER;
    ot_rate NUMBER;
BEGIN
    SELECT srate, orate
    INTO st_rate, ot_rate
    FROM payroll
    WHERE acctno = emp_id;

    RETURN st_hrs * st_rate + ot_hrs * ot_rate;
END gross_pay;
/

```

13.10.4 SQL文から起動したPL/SQLファクションの副作用の制御

サブプログラムは、独自のローカル変数の値以外のものを変更する場合、副作用を伴います。たとえば、次のいずれかを変更するサブプログラムには、副作用があります。

- 独自のOUTまたはIN OUTパラメータ
- グローバル変数
- パッケージ内のパブリック変数
- データベース表
- データベース
- 外部の状態(たとえば、DBMS_OUTPUTの起動や電子メールの送信などによる変更)

副作用によって、問合せの並列処理が妨害されたり、処理順序に依存する(したがって、不確定な)結果が発生したり、ユーザー・セッションにまたがったパッケージ状態のメンテナンスが必要になります。

ファンクションがSQL問合せまたはDML文から起動される場合は、一部の副作用は受け入れられません。

Oracle Database 8gリリース1 (8.1)より前では、アプリケーション開発者はPRAGMA RESTRICT_REFERENCESを使用してファンクションの純粋度(副作用からの自由度)をアサートしていました。このプラグマは、下位互換用として使用可能のままですが、新しいアプリケーションでは使用しないでください。かわりに、ファンクションの作成時にはオプティマイザ・ヒント DETERMINISTICおよびPARALLEL_ENABLEを指定してください。

トピック:

- [SQL文から起動したファンクションの制限](#)
- [並列SQL文から起動したPL/SQLファンクション](#)
- [PRAGMA RESTRICT_REFERENCES](#) (非推奨)

関連項目:

DETERMINISTICおよびPARALLEL_ENABLEオプティマイザ・ヒントの詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。

13.10.4.1 SQL文から起動したファンクションの制限



ノート:

SQL 文から起動したファンクションの制限は、SQL 文によって起動されるトリガーにも適用されます。

SQL文がファンクションを起動し、ファンクションが新規SQL文を実行すると、新しい文の実行は、ファンクションを起動した文のコンテキストに論理的に埋め込まれます。このコンテキストで新しい文が安全であるようにするために、Oracle Databaseはファンクションに対して次の制限を課します。

- ファンクションを起動するSQL文が問合せまたはDML文である場合、ファンクションは、現行のトランザクションの終了、セーブポイントの作成またはセーブポイントまでのロールバック、あるいはシステムまたはセッションの変更(ALTER)を実行できません。
- ファンクションを起動するSQL文が問合せまたはパラレル化されたパラレルDML文である場合、ファンクションは、DML文を実行できないか、またはデータベースを変更できません。
- ファンクションを起動するSQL文がDML文である場合、ファンクションは、ファンクションを起動したSQL文によって変更される表の読取りまたは変更を実行できません。

これらの制限は、ファンクションが新規SQL文を実行する方法とは関係なく適用されます。たとえば、これらは、ファンクションが次の処理を行う新規SQL文に適用されます。

- ファンクションの本体に直接埋め込まれているかどうかにかかわらず、PL/SQLから起動し、EXECUTE IMMEDIATEを使用して実行するか、またはDBMS_SQLパッケージを使用して実行
- JDBCを使用した実行
- 外部Cファンクション内からコールバック・コンテキストを使用してOCIで実行

これらの制約を回避するには、新しいSQL文の実行が、ファンクションを起動するSQL文のコンテキストに論理的に埋め込まれていない場合を確認してください。たとえば、新規SQL文を自律型トランザクションに配置するか、OCIで、OCIExtProcContext引数によって提供されるハンドルを使用するかわりに外部Cファンクションに対する新規接続を作成します。

関連項目:

[自律型トランザクション](#)

13.10.4.2 パラレルSQL文から起動したPL/SQLファンクション

Oracle Databaseがパラレル化されたSQL文を実行する場合、同時に多数のプロセスが連携して1つのSQL文を実行します。パラレル化されたSQL文がファンクションを起動すると、各プロセスにより、プロセスが処理する行のサブセットのみに対して、そのファンクションのコピーが起動される場合があります。

各プロセスには、そのプロセス専用のパッケージ変数のコピーがあります。パラレル実行が開始されると、ユーザーがシステムにログインしたときのように、プロセスごとにパッケージ変数が初期化されます。パッケージ変数内の値は、元のログイン・セッションからはコピーされません。1つのプロセスがパッケージ変数に対して行う変更が他のプロセスまたは元のログイン・セッションに自動的に伝播されることはありません。JavaのSTATICクラス属性は、同様に、各プロセス内で独立して初期化され変更されます。ファンクションは、パッケージおよびJava STATIC変数を使用して、発生する様々な行にわたって値を累積できます。このため、Oracle Databaseはデフォルトでは、ユーザー定義ファンクションの実行をパラレル化しません。

Oracle Database 8gリリース1 (8.1)より前:

- パラレル化された問合せによってユーザー定義ファンクションが起動されたときに、PRAGMA RESTRICT_REFERENCESによってファンクションに対してRNPSとWNPSの両方がアサートされた(つまり、ファンクションがパッケージ変数を参照、またはその値を変更しなかった)場合、ファンクションの実行をパラレル化できました。
このアサーションがなく、ファンクションがパッケージ変数を参照、またはその値を変更しなかったことをOracle Databaseが確認した場合、スタンドアロンPL/SQLファンクション(ただし、CまたはJava関数ではありません)の実行をパラレル化できました。
- パラレル化されたDML文によってユーザー定義ファンクションが起動されたときに、PRAGMA RESTRICT_REFERENCESによってファンクションに対してRNDS、WNDS、RNPSおよびWNPSがアサートされた(つまり、ファンクションがパッケージ変数またはデータベース表を参照、またはその値を変更しなかった)場合、ファンクションの実行をパラレル化できました。
このアサーションがなく、ファンクションがパッケージ変数またはデータベース表を参照、またはその値を変更しなかったことをOracle Databaseが確認した場合、スタンドアロンPL/SQLファンクション(ただし、CまたはJava関数ではありません)の実行をパラレル化できました。

Oracle Database 8gリリース1 (8.1)以降、パラレル化されたSQL文によってユーザー定義ファンクションが起動された場合、ファンクションの実行は次の状況下でパラレル化できます。

- ファンクションがPARALLEL_ENABLEを使用して作成された場合。
- Oracle Database 8gリリース1 (8.1)より前では、データベースがファンクションをパラレル化可能として識別した場合。

13.10.4.3 PRAGMA RESTRICT_REFERENCES

ノート:



PRAGMA RESTRICT_REFERENCES は非推奨です。新規アプリケーションでは、RESTRICT_REFERENCES のかわりに DETERMINISTIC および PARALLEL_ENABLE(『Oracle Database SQL 言語リファレンス』を参照)を使用することをお勧めします。

既存のPL/SQLアプリケーションでは、PRAGMA RESTRICT_REFERENCESを削除することも、新しいファンクションでも既存コードとの統合を容易にするために、これを引き続き使用することもできます。たとえば:

- 既存コードからPRAGMA RESTRICT_REFERENCESを完全に削除することが不可能かつ非現実的な場合。
たとえば、サブプログラムS1がサブプログラムS2に依存し、S1からプラグラマを削除しない場合、S1のコンパイルにS2のプラグラマが必要になる場合があります。
- 既存コードでPRAGMA RESTRICT_REFERENCESをPARALLEL_ENABLEおよびDETERMINISTICに置き換えると、新しい依存コードのアクションに悪影響を与える可能性がある場合。

PRAGMA RESTRICT_REFERENCESを使用してファンクションの純粋度をアサートする場合、(パッケージ本体ではなく)パッケージ仕様で、ファンクション宣言後の任意の場所で、次の構文を使用します。

```
PRAGMA RESTRICT_REFERENCES (function_name, assertion [, assertion]... );
```

この場合、assertionは次のいずれかです。

アサーション	意味
RNPS	ファンクションはパッケージ状態を読み込みません(パッケージ変数の値を参照しないということです)
WNPS	ファンクションはパッケージ状態を書き込みません(パッケージ変数の値を変更しないということです)。
RNDS	ファンクションはデータベース状態を読み込みません(データベース表を問い合わせないということです)。
WNDS	ファンクションはデータベース状態を書き込みません(データベース表を変更しないということです)。
TRUST	ファンクションに対して行われた任意のアサーションにファンクション本体内の SQL 文が違反していないことを信頼します。詳細は、 「アサーション TRUST の指定」 を参照してください。

TRUSTを指定せず、指定したアサーションにファンクション本体内のSQL文が違反すると、PL/SQLコンパイラは、違反している文の解析中にエラー・メッセージを表示します。

PL/SQLコンパイラによってファンクションが不要に拒否されることがないように、ファンクションで許可される最高の純粋度レベル(最上位アサーション)をアサートしてください。

ノート:



ファンクションがサブプログラムを起動する場合、これらのサブプログラムに対しても PRAGMA RESTRICT_REFERENCES を指定するか、起動する側のファンクションまたは起動される側のサブプログラムに TRUST を指定してください。

関連項目:

PRAGMA RESTRICT_REFERENCES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

トピック:

- [アサーションTRUSTの指定](#)
- [静的SQL文と動的SQL文の違い](#)

13.10.4.3.1 例: PRAGMA RESTRICT_REFERENCES

PL/SQLパッケージを作成する際には、PRAGMA RESTRICT_REFERENCES句を使用できます。

[例13-11](#)では、データベースまたはパッケージ状態の読取りおよび書込みを行わず、それが最大の純粋度レベルであることをアサートするファンクションを作成します。

例13-11 PRAGMA RESTRICT_REFERENCES

```
DROP TABLE accounts; -- in case it exists
CREATE TABLE accounts (
    acctno    INTEGER,
    balance   NUMBER
);
```

```

INSERT INTO accounts (acctno, balance)
VALUES (12345, 1000.00);

CREATE OR REPLACE PACKAGE finance AS
    FUNCTION compound_ (
        years IN NUMBER,
        amount IN NUMBER,
        rate IN NUMBER
    ) RETURN NUMBER;
    PRAGMA RESTRICT_REFERENCES (compound_, WNDS, WNPS, RNDS, RNPS);
END finance;
/

CREATE PACKAGE BODY finance AS
    FUNCTION compound_ (
        years IN NUMBER,
        amount IN NUMBER,
        rate IN NUMBER
    ) RETURN NUMBER
    IS
    BEGIN
        RETURN amount * POWER((rate / 100) + 1, years);
    END compound_;
    -- No pragma in package body
END finance;
/

DECLARE
    interest NUMBER;
BEGIN
    SELECT finance.compound_(5, 1000, 6)
    INTO interest
    FROM accounts
    WHERE acctno = 12345;
END;
/

```

13.10.4.3.2 アサーションTRUSTの指定

PRAGMA RESTRICT REFERENCESによってTRUSTが指定されると、PL/SQLコンパイラはサブプログラム本体に違反があるかどうかを確認しません。

TRUSTを使用すると、PRAGMA RESTRICT REFERENCESを使用するサブプログラムが、これを使用しないサブプログラムを起動しやすくなります。

PL/SQLサブプログラムがCまたはJavaサブプログラムを起動する場合、PL/SQLサブプログラム([例13-12](#)を参照)またはCまたはJavaサブプログラム([例13-13](#)を参照)に対してTRUSTを指定する必要があります。これは、PL/SQLコンパイラは実行時にCまたはJavaサブプログラムに違反があるかどうかを確認できないためです。

例13-12 起動する側にTRUSTが指定されたPRAGMA RESTRICT REFERENCES

```

CREATE OR REPLACE PACKAGE p IS
    PROCEDURE java_sleep (milli_seconds IN NUMBER)
    AS LANGUAGE JAVA NAME 'java.lang.Thread.sleep(long)';

    FUNCTION f (n NUMBER) RETURN NUMBER;
    PRAGMA RESTRICT_REFERENCES(f, WNDS, TRUST);
END p;
/

CREATE OR REPLACE PACKAGE BODY p IS
    FUNCTION f (
        n NUMBER
    ) RETURN NUMBER
    IS
    BEGIN
        java_sleep(n);
    END f;
END p;
/

```

```

        RETURN n;
    END f;
END p;
/

```

例13-13 起動される側にTRUSTが指定されたPRAGMA RESTRICT REFERENCES

```

CREATE OR REPLACE PACKAGE p IS
    PROCEDURE java_sleep (milli_seconds IN NUMBER)
    AS LANGUAGE JAVA NAME 'java.lang.Thread.sleep(long)';
    PRAGMA RESTRICT_REFERENCES(java_sleep,WNDS,TRUST);

    FUNCTION f (n NUMBER) RETURN NUMBER;
END p;
/
CREATE OR REPLACE PACKAGE BODY p IS
    FUNCTION f (
        n NUMBER
    ) RETURN NUMBER
    IS
    BEGIN
        java_sleep(n);
        RETURN n;
    END f;
END p;
/

```

13.10.4.3.3 静的SQL文と動的SQL文の違い

静的なINSERT、UPDATEまたはDELETE文は、データベース状態(表の列など)を明示的に読み込まない場合は、RNDSには違反しません。動的なINSERT、UPDATEまたはDELETE文の場合は、データベース状態を明示的に読み込むかどうかにかかわらず、常にRNDSに違反します。

次のINSERT文は、動的に実行される場合はRNDSに違反しますが、静的に実行される場合は違反しません。

```
INSERT INTO my_table values(3, 'BOB');
```

次のUPDATE文は、my_tableの列nameを明示的に読み込むため、静的に実行された場合も動的に実行された場合もRNDSに違反します。

```
UPDATE my_table SET id=777 WHERE name='BOB';
```

13.11 ストアド・サブプログラムの分析とデバッグ

ストアド・サブプログラムをコンパイルするには、コードの構文エラーを修正する必要があります。サブプログラムが正常に実行され、パフォーマンスも最適であり、エラーが修正されていることを確認するには、追加のデバッグを行う必要があります。次のようなデバッグが考えられます。

- 出力文を追加して、実行処理の検証およびサブプログラム内の任意の点でのデータ値のチェックをする。
変数または式の値を出力するには、OracleパッケージDBMS_OUTPUT内のPUTおよびPUT_LINEサブプログラムを使用します。
- PL/Scope、PL/SQL階層プロファイラまたはデバッガの実行によって、プログラムとその実行をより詳細に分析する

トピック:

- [PL/Scope](#)
- [PL/SQL階層プロファイラ](#)

- [PL/SQLおよびJavaのデバッグ](#)

関連項目:

- PL/SQLサブプログラムおよびパッケージのエラー処理の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください
- DBMS_OUTPUTパッケージの詳細は、『[Oracle Database PL/SQLパッケージおよびタイプ・リファレンス](#)』を参照してください。

13.11.1 PL/Scope

PL/Scopeによって、強力で効率的なPL/Scopeソース・コード・ツールの開発が可能になります。このツールは、ソース・コードの参照および理解に費やされる時間を最小限にすることによって、PL/SQL開発者の生産性を向上させます。

PL/Scopeの詳細は、『[PL/Scopeの使用](#)』を参照してください。

13.11.2 PL/SQL階層プロファイラ

PL/SQL階層プロファイラは、サブプログラム・コール別に編成されたPL/SQLプログラムの動的実行プロファイルレポートします。SQL実行時間およびPL/SQL実行時間が個別に説明されます。動的実行プロファイルにおけるサブプログラム・レベルの各サマリーには、サブプログラムへのコール数、サブプログラム自体に要した時間、サブプログラムのサブツリー（つまり依存サブプログラム）に要した時間、詳細な親子情報などが表示されます。

生成されたHTMLレポートは任意のブラウザで参照できます。ブラウザのナビゲーション機能と厳選したリンクを組み合わせた効率的な手段により、大規模なアプリケーションのパフォーマンスを分析し、アプリケーションのパフォーマンスを向上させ、開発コストを削減できます。

関連項目:

PL/SQL階層プロファイラの詳細は、『[PL/SQL階層プロファイラの使用](#)』を参照してください。

13.11.3 PL/SQLおよびJavaのデバッグ

データベース内のPL/SQLとJavaコードは、Oracle SQL Developer、Oracle JDeveloperおよび各種のサード・パーティ・ツールを使用してデバッグできます。DBMS_DEBUG_JDWPパッケージを使用すると、データベース・セッションとこれらのデバッグ・プログラム間の接続を確立できます。

セッションに接続して別のセッションからデバッグすることで、長時間実行しているテストや運用環境で発生する問題を調査できます。セッションのデバッグ中に、有効範囲内変数の状態を調べることができます。また、デバッグされるセッションでコミットされていないトランザクション中にデータベースの状態を参照する際に、データベースの状態を調べることができます。ブレークポイントで停止すると、デバッグを実行しているユーザーはSQLコマンドを発行して、必要に応じて匿名ブロックのストアードPL/SQLサブプログラムを呼び出すPL/SQLコードを実行できます。

関連項目:

- 関数とプロシージャの実行とデバッグの詳細は、『[Oracle SQL Developerユーザーズ・ガイド](#)』を参照してください。

- Java Debug Wire Protocol (JDWP) PL/SQLデバッガの使用の詳細は、[『Oracle Database Java開発者ガイド』](#)を参照してください。
- DBMS_DEBUG_JDWP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。
- V\$PLSQL_DEBUGGABLE_SESSIONSビューの詳細は、[『Oracle Databaseリファレンス』](#)を参照してください。

13.11.3.1 デバッグ用のコードのコンパイル

デバッガは、生成されたデバッグ情報とともにコンパイルされたコードでのみ、個々のコード行で停止したり、変数にアクセスできます。

生成されたデバッグ情報とともにPL/SQLユニットをコンパイルするには、コンパイル・パラメータPLSQL_OPTIMIZE_LEVEL (デフォルト値は2)を1に設定します。

ノート:



PL/SQL コンパイラでは、PL/SQL の wrap ユーティリティにより非表示になっているコードに対しては、デバッグ情報は生成されません。

関連項目:

- [wrapユーティリティの詳細](#)は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。
- PL/SQLユニットの詳細は、[PL/SQLユニットの概要](#)を参照してください。
- PLSQL_OPTIMIZE_LEVELの詳細は、[『Oracle Databaseリファレンス』](#)を参照してください。

13.11.3.2 PL/SQLストアド・サブプログラムおよびJavaストアド・サブプログラムのデバッグ権限

デバッガに接続するセッションでは、DEBUG CONNECT SESSION、DEBUG CONNECT ANYまたは適切なDEBUG CONNECT ON USER権限を所有するユーザーが接続操作を実行する必要があります。接続コールに関係するDRサブプログラムの所有者であるユーザーも、この操作を実行できます。

セッションがデバッガに接続されると、セッション・ログイン・ユーザーおよび有効なセッション・レベルのロールが、このデバッグ用の接続の権限環境として決定されます。デバッグに必要なすべての権限を、関連コードでこのユーザーとロールの組合せに付与する必要があります。これらの権限は、次のとおりです。

- PL/SQLパッケージ仕様で宣言された変数またはJavaパブリック変数の表示および変更: EXECUTEまたはDEBUG。
- プライベート変数の表示および変更、またはコード行をブレークポイントまでおよびステップごとに実行するには: DEBUG

注意:



DEBUG 権限を付与することにより、デバッグ対象のサブプログラムが実行するようにコーディングされている処理をデバッグ・セッションで実行することが可能になります。

DEBUG ANY PROCEDUREシステム権限の付与は、データベース内のすべてのオブジェクトにDEBUG権限を付与することと同じです。SYSが所有するオブジェクトは含まれません。

注意:



SYS が所有するオブジェクトに DEBUG ANY PROCEDURE 権限または DEBUG 権限を付与すると、すべての権限がデータベースに付与されます。

関連項目:

- システム権限およびオブジェクト権限の詳細は、『Oracle Database SQL言語リファレンス』を参照してください
- Javaサブプログラムのデバッグ権限の詳細は、『Oracle Database Java開発者ガイド』を参照してください
- DBMS_DEBUG_JDWPパッケージのセキュリティ・モデルの詳細は、[『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。

13.12 パッケージの無効化とセッションの状態

パッケージ・オブジェクトを参照する各セッションには、対応するパッケージの独自のインスタンスがあり、この中には、パブリック変数、プライベート変数、カーソルおよび定数に対する持続状態が含まれます。セッションのインスタンス化されたパッケージ(仕様部または本体)のいずれかが無効化されると、そのセッションのすべてのパッケージ・インスタンスが無効になり、再コンパイルされます。これにより、そのセッションのすべてのパッケージ・インスタンスに関するセッション状態は失われます。

セッションのパッケージが無効である場合、セッションが無効なパッケージ・インスタンスのオブジェクトを最初に使用しようとしたときに、ORA-04068が戻されます。2度目にセッションがこのようなパッケージ・コールを行うと、エラーは発生せずに、パッケージはセッションに対して再インスタンス化されます。ただし、このエラーをアプリケーションで処理する場合は、次の点に注意してください。

- パフォーマンスを最適な状態に保つため、Oracle Databaseがこのエラー・メッセージを戻すのは、パッケージ状態が破棄されるとき1回のみです。あるパッケージ内のサブプログラムにより別のパッケージ内のサブプログラムが起動される場合、両方のパッケージに対するセッション状態が失われます。
- サーバー・セッションがORA-04068をトラップした場合、クライアント・セッションに対してORA-04068は発生しません。したがって、このクライアント・セッションがパッケージ内のオブジェクトを使用しようとする、このパッケージは再インスタンス化されません。パッケージを再インスタンス化するには、クライアント・セッションをデータベースに再接続するか、またはクライアント・セッションでパッケージを再コンパイルする必要があります。

本番環境の多くでは、パッケージが無効になるDDL操作は、通常、業務時間外に行われるため、エンドユーザー・アプリケーションでは、このような状況は問題にならない可能性もあります。しかし、パッケージが業務時間中に無効になることがよくある場合は、パッケージ・コールが行われたときにこのエラーを処理するように、アプリケーションを作成することが必要になります。

13.13 例: ORA-04068エラーの呼出し

RAISE句を使用して例外を呼び出すことができます。

[例13-14](#)では、RAISE文により、処理中の例外ORA-06508の原因である現在の例外ORA-04068が呼び出されます。ORA-04068は検出されません。

例13-14 ORA-04068の呼出し

```
PROCEDURE p IS
  package_exception EXCEPTION;
  PRAGMA EXCEPTION_INIT (package_exception, -6508);
```

```

BEGIN
    ...
EXCEPTION
    WHEN package_exception THEN
        RAISE;
END;
/

```

13.14 例: ORA-04068の検出

パッケージ定義でRAISE文を使用してエラーを検出できます。

[例13-15](#)では、RAISE文により、ORA-06508に応じて、現在の例外ORA-04068ではなく、例外ORA-20001が呼び出されます。ORA-04068が検出されます。このような場合、ORA-04068エラーはマスクされ、これにより、パッケージの再インスタンス化が停止します。

例13-15 ORA-04068の検出

```

PROCEDURE p IS
    package_exception EXCEPTION;
    other_exception     EXCEPTION;
    PRAGMA EXCEPTION_INIT (package_exception, -6508);
    PRAGMA EXCEPTION_INIT (other_exception, -20001);
BEGIN
    ...
EXCEPTION
    WHEN package_exception THEN
        ...
        RAISE other_exception;
END;
/

```

14 PL/Scopeの使用

PL/Scopeによって、強力で効率的なPL/Scopeソース・コード・ツールの開発が可能になります。このツールは、ソース・コードの参照および理解に費やされる時間を最小限にすることによって、PL/SQL開発者の生産性を向上させます。

PL/Scopeはアプリケーション開発者向けのものであり、通常は開発データベース環境で使用されます。

ノート:



PL/Scope では、ソース・コードがラップされている PL/SQL ユニットのデータを収集できません。PL/SQL ソース・コードのラップの詳細は、[『Oracle Database PL/SQL 言語リファレンス』](#)を参照してください。

トピック:

- [PL/Scopeの概要](#)
- [PL/Scopeの使用に必要な権限](#)
- [識別子と文の収集の指定](#)
- [PL/Scopeデータによる領域の使用量](#)
- [PL/Scopeデータの表示](#)
- [PL/SQLコードの管理に役立つデータ・ディクショナリ・ビューの概要](#)
- [サンプルPL/Scopeセッション](#)

14.1 PL/Scopeの概要

PL/Scopeはコンパイラ駆動方式のツールであり、PL/SQLとSQL識別子およびPL/SQLソース・コード内のSQL文の使用方法を収集します。

PL/Scopeはプログラム・ユニットのコンパイル時にPL/SQL識別子、SQL識別子およびSQL文のメタデータを収集し、静的データ・ディクショナリ・ビューでできるようにします。収集されたデータには識別子の型、使用方法(DECLARATION、DEFINITION、REFERENCE、CALL、ASSIGNMENT)およびソース・コードにおける各使用場所に関する情報が含まれています。

Oracle Database 12c リリース2 (12.2)以降では、PL/ScopeはPL/SQLユニット内の静的SQLと動的SQLの呼出し位置の発生をレポートするように拡張されています。ネイティブ動的SQL (EXECUTE IMMEDIATE、OPEN CURSOR FOR)とDBMS_SQLコールの呼出し位置が収集されます。動的SQL文は実行時に生成されるため、コンパイル時には呼出し位置のみを収集できます。新しいDBA_STATEMENTSビューで収集されたデータは他のデータ・ディクショナリ・ビューとともに問合せを行い、プロジェクトのプログラミングに必要な変更の対象範囲やコード分析の実行についての疑問に答えを出すために役立ちます。また、正しく実行されないSQL文のソースを特定するためにも有用です。PL/Scopeは、表、ビューおよびPL/SQLユニット間の依存関係についての洞察を提供します。このレベルの詳細は、必要な変更の程度を判断する移行アセスメント・ツールとして使用できます。

PL/Scopeは、次に示す疑問に答えを出す上で役立ちます。

- 表yの列xはPL/SQLコードのどこで、どのように使用されるのか。

- 自分のアプリケーションのPL/SQLコードのSQLに、TimesTenとの互換性があるかどうか。
- 宣言されてはいるが使用されていない定数、変数および例外が、自分のアプリケーションの場合どれなのか。
- 自分のコードにSQLインジェクションの危険性があるかどうか。
- アプリケーションでコード化されたオプティマイザ・ヒントを使用するSQL文は何なのか。
- BULK COLLECT句を持つSQLはどれなのか。SQLがどこから呼び出されるのか。

14.2 PL/Scopeの使用に必要な権限

PUBLICにはデフォルトで、様々なシステム表とビューに対するSELECT権限と、様々なPL/SQLオブジェクトに対するEXECUTE権限があります。

PL/Scopeデータは、DBA_IDENTIFIERSおよびDBA_STATEMENTSデータ・ディクショナリ・ビューで使用できます。ユーザーは、これらのビューでデータを問い合わせる権限を持っている必要があります。

これらの関連ビューでは、次に示す権限が付与されています。

ビュー名	ロールに付与された権限
USER_IDENTIFIERS	PUBLIC への READ
ALL_IDENTIFIERS	PUBLIC への READ
DBA_IDENTIFIERS	SELECT_CATALOG_ROLE への SELECT
USER_STATEMENTS	PUBLIC への READ
ALL_STATEMENTS	PUBLIC への READ
DBA_STATEMENTS	SELECT_CATALOG_ROLE への SELECT

データベース管理者は次のような問合せを使用して、これらのビューに対する権限のリストを検証できます。

```
SELECT*
FROMSYS.DBA_TAB_PRIVS
WHEREGRANTEE='PUBLIC'
AND TABLE_NAME IN
('ALL_IDENTIFIERS','USER_IDENTIFIERS','ALL_STATEMENTS','USER_STATEMENTS');
```

14.3 識別子と文の収集の指定

デフォルトでは、PL/ScopeはPL/SQLソース・プログラムの識別子と文のデータを収集しません。収集対象を有効化し制御するには、PL/SQLコンパイル・パラメータPLSCOPE_SETTINGSを設定します。

Oracle Database 12c リリース2 (12.2)以降では、PLSCOPE_SETTINGSに新しい構文があります。この構文は、識別子とSQL文のメタデータを収集するためのコントロールとオプションを、さらに提供します。メタデータは、静的データ・ディクショナリ・ビューDBA_IDENTIFIERSとDBA_STATEMENTSに収集されます。

パッケージ本体にある識別子など、PL/SQLソース・プログラムのすべての識別子用のデータをPL/Scopeによって収集するには、PL/SQLコンパイル・パラメータPLSCOPE_SETTINGSを'IDENTIFIERS:ALL'に設定します。IDENTIFIERS句に使用できる値は、ALL、NONE (デフォルト)、PUBLIC、SQLおよびPLSQLです。新しいSQL識別子は、ALIAS、COLUMN、MATERIALIZED VIEW、OPERATOR、TABLEおよびVIEWです。拡張されたメタデータ収集により、アプリケーションを理解する上で役立つレポートを生成できます。PL/Scopeは、PL/SQLアプリケーションのコード化プロジェクトの複雑さを、これま

でよりも細かく評価できるツールとして使用できるようになりました。

PL/SQLソース・プログラムで使用される、すべてのSQL文のPL/Scopeデータを収集するには、PL/SQLコンパイル・パラメータ PLSCOPE_SETTINGSを 'STATEMENTS:ALL' に設定します。デフォルト値はNONEです。



ノート:

すべての識別子と文を収集すると、大量のデータが生成され、コンパイル時間が長くなる可能性があります。

PL/Scopeでは、収集するデータをSYSAUX表領域に格納します。プログラム・ユニットのコンパイル中に、PL/Scope収集が有効でSYSAUX表領域が使用不可の場合、PL/Scopeはコンパイルされたオブジェクトのデータを収集しません。コンパイラから警告は発行されませんが、USER_ERRORS内に警告が保存されます。

関連項目:

- PLSCOPE_SETTINGSの詳細は、[『Oracle Databaseリファレンス』](#)を参照してください。
- PL/SQLのコンパイル・パラメータの詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。

14.4 PL/Scopeデータによる領域の使用量

PL/ScopeのデータはSYSAUX表領域に格納されます。SYSDBAとしてログオンしている場合は、[例14-1](#)の問合せを使用してPL/Scopeデータが使用している領域の容量を表示できます。

例14-1 PL/Scopeデータによる領域の使用量

問合せ:

```
SELECT SPACE_USAGE_KBYTES
FROM V$SYSAUX_OCCUPANTS
WHERE OCCUPANT_NAME= 'PL/SCOPE' ;
```

結果:

```
SPACE_USAGE_KBYTES
-----
                1920

1 row selected.
```

関連項目:

SYSAUX表領域の管理の詳細は、[『Oracle Database管理者ガイド』](#)を参照してください。

14.5 PL/Scopeデータの表示

PL/Scopeで収集したデータを表示するには、次のいずれかを使用できます。

- [PL/SQLおよびSQL識別子の静的データ・ディクショナリ・ビュー](#)
- [SQL文の静的データ・ディクショナリ・ビュー](#)

- [SQL Developer](#)

14.5.1 PL/SQLおよびSQL識別子の静的データ・ディクショナリ・ビュー

静的データ・ディクショナリ・ビューのファミリーDBA_IDENTIFIERSでは、その型や使用方法などのPL/Scope識別子に関する情報が表示されます。

トピック:

- [PL/Scopeが収集するPL/SQLおよびSQL識別子の型](#)
- [識別子の使用方法について](#)
- [識別子の使用方法: 一意キー](#)
- [識別子の使用方法について: コンテキスト](#)
- [識別子のシグネチャについて](#)

関連項目:

ディクショナリ・ビューDBA_IDENTIFIERSビューの詳細は、『[Oracle Databaseリファレンス](#)』を参照してください。

14.5.1.1 PL/Scopeが収集するPL/SQLおよびSQL識別子の型

[表14-1](#)に、PL/Scopeが収集する識別子の型をアルファベット順に示します。[表14-1](#)の識別子の型は、DBA_IDENTIFIERS静的データ・ディクショナリ・ビューのファミリーのTYPE列に表示されます。

ノート:

PLSCOPE_SETTINGS=' IDENTIFIERS:ALL 'でコンパイルされていないコンパイル・ユニットに宣言された識別子は、DBA_IDENTIFIERS 静的データ・ディクショナリ・ビューに表示されません。

ROWNUM などの疑似列は、ユーザー定義の識別子ではないため、サポートされていません。

PL/Scope はリテラル文字列の列名を無視します。

表14-1 PL/Scopeが収集する識別子の型

TYPE列の値	コメント
ALIAS	Oracle Database 12c リリース 2 (12.2)で導入された、新しいSQL 識別子です。
ASSOCIATIVE ARRAY	
COLUMN	Oracle Database 12c リリース 2 (12.2)で導入された、新しいSQL 識別子です。

TYPE列の値	コメント
CONSTANT	
CURSOR	
BFILE DATATYPE	各 DATATYPE は、STANDARD パッケージで宣言されたベース型です。
BLOB DATATYPE	
BOOLEAN DATATYPE	
CHARACTER DATATYPE	
CLOB DATATYPE	
DATE DATATYPE	
INTERVAL DATATYPE	
NUMBER DATATYPE	
TIME DATATYPE	
TIMESTAMP DATATYPE	
EXCEPTION	
FORMAL IN	
FORMAL IN OUT	
FORMAL OUT	
FUNCTION	
INDEX TABLE	
ITERATOR	イテレータは FOR ループの索引です。
LABEL	ラベル宣言もコンテキストとして機能します。
LIBRARY	
MATERIALIZED VIEW	Oracle Database 12c リリース 2 (12.2)で導入された、新しい SQL 識別子です。
NESTED TABLE	

TYPE列の値	コメント
OBJECT	
OPAQUE	内部不透明型には ANYDATA および XMLType があります。
OPERATOR	Oracle Database 12c リリース 2 (12.2)で導入された、新しい SQL 識別子です。
PACKAGE	
PROCEDURE	
RECORD	
REFCURSOR	
SEQUENCE	Oracle Database 12c リリース 2 (12.2)で導入された、新しい SQL 識別子です。
SUBTYPE	
SYNONYM	PL/Scope では、シノニムのベース・オブジェクト名は解決しません。シノニムのベース・オブジェクト名を調べるには、*_SYNONYMS を問い合わせます。
TABLE	Oracle Database 12c リリース 2 (12.2)で導入された、新しい SQL 識別子です。
TRIGGER	
UROWID	
VARIABLE	オブジェクト属性、ローカル変数、パッケージ変数、レコード・フィールドのいずれにもできます。
VARRAY	
VIEW	この型は、ビューの編集に使用されます。 Oracle Database 12c リリース 2 (12.2)で導入された、新しい SQL 識別子です。

関連項目:

ストアド・オブジェクト内の識別子の詳細は、[『Oracle Databaseリファレンス』](#)を参照してください。

14.5.1.2 識別子の使用方法について

PL/Scopeは、識別子のかわりに実行されるアクションを説明する動詞として使用します。

[表14-2](#)に、PL/Scopeがレポートする使用方法をアルファベット順に示します。[表14-2](#)の識別子のアクションは、DBA_IDENTIFIERS静的データ・ディクショナリ・ビューのファミリのUSAGE列に表示されます。このビューについては、[『Oracle Databaseリファレンス』](#)で説明されています。

表14-2 PL/Scopeがレポートする使用方法

USAGE列の値	説明:
ASSIGNMENT	代入は、VARIABLE などの値を保持できる識別子に対してのみ可能です。 代入には次のものがあります。 ● 代入演算子の左側での識別子の使用 ● FETCH 文の INTO 句での識別子の使用 ● サブプログラムへの識別子の参照渡し(OUT モード) ● OUT または IN OUT モードでの、EXECUTE IMMEDIATE 文の USING 句における識別子のバインド変数としての使用 IN OUT モードでサブプログラムに渡される識別子には、REFERENCE 使用方法(IN に対応)と ASSIGNMENT 使用方法(OUT に対応)の両方があります。 式およびネストした副問合せは、割当てソースとしてサポートされていません。
CALL	PL/Scope のコンテキストでは、CALL はコール・スタックの上にコールをプッシュする操作です。つまり、次のようになります。 ● FUNCTION または PROCEDURE へのコール ● カーソル識別子の実行またはフェッチ(SQL への論理コール) GOTO 文または例外の発生はどちらもコール・スタックの上にコールをプッシュしないため、CALL ではありません。
DECLARATION	DECLARATION は、識別子が存在し、それぞれの識別子に 1 つずつ DECLARATION があることをコンパイラに通知します。各 DECLARATION は、関連データ型を持つことができます。 ループ索引宣言の場合、(*_IDENTIFIERS ビューの)LINE および COL は、暗黙的にループ索

USAGE列の値	説明:
	引を宣言する FOR 句の行および列です。 ラベル索引宣言の場合、LINE および COL は、デリミタ<<および>>内でラベルが出現する(および暗黙的に宣言される)行および列です。
DEFINITION	DEFINITION は、以前に宣言した識別子を実装または使用方法をコンパイラに通知します。 次の各識別子の型にそれぞれ DEFINITION があります。 ● EXCEPTION(複数定義が可能) ● FUNCTION ● OBJECT ● PACKAGE ● PROCEDURE ● TRIGGER 最上位の識別子の場合のみ、DEFINITION および DECLARATION は同じ場所にあります。
REFERENCE	REFERENCE は、その値を変更せずに識別子を使用します。 参照には次のものがあります。 ● 例外識別子の発生。 ● 変数または仮パラメータの宣言での型識別子の使用。 ● あるフィールドにアクセスするための複数のフィールドが型に含まれている変数識別子の使用。たとえば、myrecordvar.myfield := 1 では、myrecordvar に対して参照が行われ、myfield に対して代入が行われます。 ● FETCH を除く任意の目的でのカーソルの使用。 ● サブプログラムへの識別子の値渡し(IN モード)。 ● IN または IN OUT モードでの、EXECUTE IMMEDIATE 文の USING 句における識別子のバインド変数としての使用 IN OUT モードでサブプログラムに渡される識別子には、REFERENCE 使用方法(IN に対応)と ASSIGNMENT 使用方法(OUT に対応)の両方があります。

14.5.1.3 識別子の使用方法: 一意キー

すべての識別子の使用方法には、コード・ユニット内で一意の数値IDが指定されます。これは識別子のUSAGE_IDです。

*_IDENTIFIERSビューの各行は、PL/SQLユニットの識別子の一意の使用法を表します。これらの各ビューでは、次のものがコンパイル・ユニット内で同等の一意キーです。

- LINE、COL、USAGE
- USAGE_ID

ノート:



IN OUT モードでサブプログラムに渡される識別子では、*_IDENTIFIERS に REFERENCE 使用方法(IN に対応)および ASSIGNMENT 使用方法(OUT に対応)の 2 つの行があります。

この例は、PROCEDURE p1に対して生成されたUSAGE_IDを示しています。

```
CREATE OR REPLACE PROCEDURE p1 (a OUT VARCHAR2)
IS
  b VARCHAR2(100) := 'hello FROM p1';
BEGIN
  a := b;
END;
/
SELECT USAGE_ID, USAGE, NAME
FROM ALL_IDENTIFIERS
WHERE OBJECT_NAME = 'P1'
ORDER BY USAGE_ID;
```

```
1 DECLARATION P1
2 DEFINITION  P1
3 DECLARATION A
4 REFERENCE   VARCHAR2
5 DECLARATION B
6 REFERENCE   VARCHAR2
7 ASSIGNMENT  B
8 ASSIGNMENT  A
9 REFERENCE   B
```

関連項目:

*_IDENTIFIERSビューの使用法は、[「識別子の使用方法について」](#)を参照してください。

14.5.1.4 識別子の使用方法について: コンテキスト

識別子の使用法は、他の識別子の使用法のコンテキストにすることができます。これは、使用法間の1対多の親子関係を作成します。親コンテキストの使用法でのUSAGE_IDは、子の使用法ではUSAGE_CONTEXT_IDになります。

コンテキストは、複数の使用法の間の関係を検出する場合に役立ちます。最上位のスキーマ・オブジェクト宣言および定義を除き、識別子のすべての使用法は別の使用法のコンテキストで発生します。

デフォルトのトップレベルのコンテキスト(すべてのトップレベルのオブジェクトが含まれる)は、0のUSAGE_CONTEXT_IDによって識別されます。

たとえば:

- ローカル変数宣言は、最上位のプロシージャ宣言のコンテキスト内で発生します。
- x VARCHAR2(10)など、識別子の変数として宣言される場合、VARCHAR2型参照のUSAGE_CONTEXT_IDには、x宣言のUSAGE_IDが挿入され、変数宣言をその型に関連付けることができます。

つまり、[例14-2](#)に示すように、USAGE_CONTEXT_IDはUSAGE_IDに対する再帰外部キーです。

例14-2 USAGE_CONTEXT_IDおよびUSAGE_ID

```
ALTER SESSION SET PLSCOPE_SETTINGS = 'IDENTIFIERS:ALL';
CREATE OR REPLACE PROCEDURE a (p1 IN BOOLEAN) AUTHID DEFINER IS
    v PLS_INTEGER;
BEGIN
    v := 42;
    DBMS_OUTPUT.PUT_LINE(v);
    RAISE_APPLICATION_ERROR (-20000, 'Bad');
EXCEPTION
    WHEN Program_Error THEN NULL;
END a;
/

CREATE OR REPLACE PROCEDURE b (
    p2 OUT PLS_INTEGER,
    p3 IN OUT VARCHAR2
) AUTHID DEFINER
IS
    n NUMBER;
    q BOOLEAN := TRUE;
BEGIN
    FOR j IN 1..5 LOOP
        a(q); a(TRUE); a(TRUE);
        IF j > 2 THEN
            GOTO z;
        END IF;
    END LOOP;
<<z>> DECLARE
    d CONSTANT CHAR(1) := 'X';
    BEGIN
        SELECT COUNT(*) INTO n FROM Dual WHERE Dummy = d;
    END z;
END b;
/

WITH v AS (
    SELECT      Line,
               Col,
               INITCAP(NAME) Name,
               LOWER(TYPE)   Type,
               LOWER(USAGE)   Usage,
               USAGE_ID,
               USAGE_CONTEXT_ID
    FROM USER_IDENTIFIERS
       WHERE Object_Name = 'B'
          AND Object_Type = 'PROCEDURE'
)
SELECT RPAD(LPAD(' ', 2*(Level-1)) ||
           Name, 20, '.') || ' ' ||
       RPAD(Type, 20) ||
       RPAD(Usage, 20)
       IDENTIFIER_USAGE_CONTEXTS
FROM v
START WITH USAGE_CONTEXT_ID = 0
CONNECT BY PRIOR USAGE_ID = USAGE_CONTEXT_ID
ORDER SIBLINGS BY Line, Col
/
```

結果:

IDENTIFIER_USAGE_CONTEXTS

```

-----
B..... procedure      declaration
  B..... procedure      definition
    P2..... formal out   declaration
      Pls_Integer... subtype reference
    P3..... formal in out declaration
      Varchar2..... character datatype reference
  N..... variable        declaration
    Number..... number datatype reference
  Q..... variable        declaration
    Q..... variable        assignment
    Boolean..... boolean datatype reference
  J..... iterator        declaration
    A..... procedure      call
      Q..... variable        reference
    A..... procedure      call
    A..... procedure      call
    J..... iterator        reference
    Z..... label          reference
  Z..... label          declaration
    D..... constant        declaration
      D..... constant        assignment
    Char..... subtype      reference

```

14.5.1.5 識別子のシグネチャについて

識別子のシグネチャは、プログラム・ユニット内および複数のプログラム・ユニット間で一意のものです。つまり、シグネチャは複数の識別子が同じ名前であっても、同じプログラム・ユニット内に定義されているか、または異なるプログラム・ユニットに定義されているかを区別します。

[例14-3](#)のプログラム・ユニットの場合、p5という名前の2つの識別子があり、静的データ・ディクショナリ・ビュー

USER_IDENTIFIERSにNAMEがp5という行が複数ありますが、これらの行のSIGNATUREは異なるものです。外部プロシージャp5に関連付けられた行に1つのシグネチャがあり、内部プロシージャp5に関連付けられた行には別のシグネチャがあります。プログラム・ユニットqがプロシージャp5をコールする場合、qのUSER_IDENTIFIERSビューには、NAMEがp5でSIGNATUREが外部プロシージャp5のシグネチャの行があります。

例14-3 p5という名前の2つの識別子を持つプログラム・ユニット

この例はp5という名前の2つの識別子を持つプログラム・ユニットを示して、シグネチャの一意性を明示しています。

```

CREATE OR REPLACE PROCEDURE p5 AUTHID DEFINER IS
  PROCEDURE p5 IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE('Inner p5');
  END p5;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Outer p5');
  p5();
END p5;
/
SELECT LINE || ' > ' || TEXT
FROM   ALL_SOURCE
WHERE  NAME = 'P5'
      AND TYPE = 'PROCEDURE'
ORDER BY LINE;

```

```

1 > PROCEDURE p5 AUTHID DEFINER IS
2 >   PROCEDURE p5 IS
3 >   BEGIN
4 >     DBMS_OUTPUT.PUT_LINE('Inner p5');
5 >   END p5;

```

```

6 > BEGIN
7 >   DBMS_OUTPUT.PUT_LINE('Outer p5');
8 >   p5();
9 > END p5;

```

次の問合せは、PL/SQLユニットのSIGNATUREが、内部と外部のp5のDECLARATIONとDEFINITIONで同じであることを示しています。

```

SELECT SIGNATURE, USAGE, LINE, COL, USAGE_ID, USAGE_CONTEXT_ID
FROM   ALL_IDENTIFIERS
WHERE  OBJECT_NAME = 'P5'
ORDER BY LINE, COL, USAGE_ID;

```

75CD5986BA2EE5C61ACEED8C7162528F	DECLARATION	1	11	1	0
75CD5986BA2EE5C61ACEED8C7162528F	DEFINITION	1	11	2	1
33FB9F948F526C4B0634C0F35DFA91F6	DECLARATION	2	13	3	2
33FB9F948F526C4B0634C0F35DFA91F6	DEFINITION	2	13	4	3
33FB9F948F526C4B0634C0F35DFA91F6	CALL	8	3	7	2

```

CREATE OR REPLACE PROCEDURE q AUTHID DEFINER IS
BEGIN
  p5();
END q;
/

EXEC q;
Outer p5
Inner p5

SELECT SIGNATURE, USAGE, LINE, COL, USAGE_ID, USAGE_CONTEXT_ID
FROM   ALL_IDENTIFIERS
WHERE  OBJECT_NAME = 'Q'
AND NAME = 'P5'
ORDER BY LINE, COL, USAGE_ID;

```

75CD5986BA2EE5C61ACEED8C7162528F	CALL	3	3	3	2

例14-4 VARCHAR2のすべての使用方法の検索

識別子のシグネチャは、グローバルに一意です。これは、データベースのすべてのユニット内で識別子のすべての使用方法を検索するために役立ちます。この例は、VARCHAR2のすべての使用方法を検索する問合せを示しています。

```

SELECT UNIQUE OBJECT_NAME uses_varchar2
FROM   ALL_IDENTIFIERS
WHERE  SIGNATURE = (SELECT SIGNATURE
                     FROM   ALL_IDENTIFIERS
                     WHERE  OBJECT_NAME = 'STANDARD'
                           AND OWNER = 'SYS'
                           AND USAGE = 'DECLARATION'
                           AND NAME = 'VARCHAR2')
ORDER BY OBJECT_NAME;

```

14.5.2 SQL文の静的データ・ディクショナリ・ビュー

DBA_STATEMENTS静的ディクショナリ・ビュー・ファミリーは、PL/Scopeによって収集されたSQL文を説明します。

Oracle Database 12c リリース2 (12.2.0.1)以降では、新しいビューDBA_STATEMENTSが、PL/SQLユニット内の静的SQLの発生をレポートします。これは、SQL_ID、正規文のテキスト、文タイプ、有用な文使用の属性、シグネチャおよびPL/SQLコード内の場所についての情報を提供します。各行は、PL/SQLコード内のSQL文のインスタンスを表します。

トピック:

- [PL/Scopeが収集するSQL文の型](#)
- [文の場所: 一意キー](#)
- [SQL文に使用方法について: コンテキスト](#)
- [SQL文のシグネチャについて](#)

関連項目:

DBA_STATEMENTSビューの詳細は、[『Oracle Databaseリファレンス』](#)を参照してください。

14.5.2.1 PL/Scopeが収集するSQL文の型

PL/Scope文タイプは、PL/SQLで使用されるSQL文を表します。

PL/SQLでの実行や、それ以外のSQL操作に使用できる文に対応するSQL文の型。文タイプは、DBA_STATEMENTS静的データ・ディクショナリ・ビュー・ファミリのTYPE列に表示されます。

このメタデータを収集するには、PL/SQLユニットをPLSCOPE_SETTINGS='STATEMENTS:ALL'でコンパイルする必要があります。

PL/Scopeが収集するSQL文の型を次に示します。

- SELECT
- UPDATE
- INSERT
- DELETE
- MERGE
- EXECUTE IMMEDIATE
- SET TRANSACTION
- LOCK TABLE
- COMMIT
- SAVEPOINT
- ROLLBACK
- OPEN
- CLOSE
- FETCH

14.5.2.2 文の場所: 一意キー

DBA_STATEMENTSビューの各行は、PL/SQLユニット内のSQL文の、一意の場所を表します。これは、コンパイル・ユニット内の一意のキーに相当します。

次に示す列は、PL/SQLコード内の文の位置を判別するために使用されます。

- OWNER、OBJECT_NAME、OBJECT_TYPE、LINE、COL

● USAGE_ID

USAGE_IDは、PL/SQLユニット内で一意に定義されます。識別子とは異なり、SQL文には別の使用方法(DECLARATION、ASSIGNMENTまたはREFERENCEなど)がありません。すべての文は、SQLエンジンに対する暗黙のCALLとみなされます。そのため、DBA_STATEMENTSビューにUSAGE列はありませんが、USAGE_IDを使用します。

例14-5 USAGE_ID列を使用したSQL識別子と文の問合せ

```
PROCEDURE p1 (p_cust_id  NUMBER,
              p_cust_name OUT VARCHAR2)
IS
BEGIN
    SELECT (SELECT CUST_FIRST_NAME
            FROM  CUSTOMERS)
    INTO   p_cust_name
    FROM   CUSTOMERS
    WHERE  CUSTOMER_ID = p_cust_id;
END;
SELECT USAGE_ID, TYPE, NAME, USAGE, LINE, COL
FROM ( SELECT USAGE_ID, TYPE, NAME, USAGE, LINE, COL
      FROM ALL_IDENTIFIERS
      WHERE OBJECT_NAME = 'P1'
      UNION
      SELECT USAGE_ID, TYPE, 'SQL STATEMENT', " ", LINE, COL
      FROM ALL_STATEMENTS
      WHERE OBJECT_NAME = 'P1')
ORDER BY USAGE_ID;
```

USAGE_ID	TYPE	NAME	USAGE	LINE	COL
1	PROCEDURE	P1	DECLARATION	1	11
2	PROCEDURE	P1	DEFINITION	1	11
3	FORMAL IN	P_CUST_ID	DECLARATION	1	15
4	NUMBER DATATYPE	NUMBER	REFERENCE	1	25
5	FORMAL OUT	P_CUST_NAME	DECLARATION	1	33
6	CHARACTER DATATYPE	VARCHAR2	REFERENCE	1	49
7	SQL STATEMENT	SELECT		3	3
8	TABLE	CUSTOMERS	REFERENCE	4	10
9	FORMAL IN	P_CUST_ID	REFERENCE	4	38
10	COLUMN	CUSTOMER_ID	REFERENCE	4	26
11	FORMAL OUT	P_CUST_NAME	ASSIGNMENT	3	31
12	COLUMN	CUST_FIRST_NAME	REFERENCE	3	10

14.5.2.3 SQL文に使用方法について: コンテキスト

文は、他の文や識別子のコンテキストとして機能することができます。文は、他の文や識別子のコンテキスト内に置くこともできます。

USAGE_CONTEXT_ID列は、文のコンテキストを判別するために使用されます。文の中に現れるすべての識別子は、その文のコンテキストの中に存在することになります。

式およびネストした副問合せは、割当てソースとしてサポートされていません。

例14-6 DBA_STATEMENTS USAGE_CONTEXT_IDを使用した識別子の問合せ

この例では、USAGE_CONTEXT_ID列を使用して SELECT文のコンテキスト内にある識別子を取得する方法を示しています。

```
PROCEDURE p1 (p_cust_id  NUMBER,
              p_cust_name OUT VARCHAR2)
IS
BEGIN
    SELECT (SELECT CUST_FIRST_NAME
```

```

        FROM    CUSTOMERS)
    INTO    p_cust_name
    FROM    CUSTOMERS
    WHERE    CUSTOMER_ID = p_cust_id;
END;
SELECT USAGE_ID, LPAD(' ', 2*(level-1)) || TO_CHAR(USAGE) || ' ' || NAME usages, LINE,
COL
FROM ( SELECT OBJECT_NAME, USAGE, USAGE_ID, USAGE_CONTEXT_ID, NAME, LINE, COL
      FROM ALL_IDENTIFIERS
      WHERE OBJECT_NAME = 'P1'
      UNION
      SELECT OBJECT_NAME, TYPE usage, USAGE_ID, USAGE_CONTEXT_ID, 'Statement' name,
LINE, COL
      FROM ALL_STATEMENTS
      WHERE OBJECT_NAME = 'P1'
      )
START WITH USAGE_CONTEXT_ID = 0 AND OBJECT_NAME = 'P1'
CONNECT BY PRIOR USAGE_ID = USAGE_CONTEXT_ID AND OBJECT_NAME = 'P1';

```

USAGE_ID	USAGES	LINE	COL
1	DECLARATION P1	1	11
2	DEFINITION P1	1	11
3	DECLARATION P_CUST_ID	1	15
4	REFERENCE NUMBER	1	27
5	DECLARATION P_CUST_NAME	2	33
6	REFERENCE VARCHAR2	2	49
7	SELECT STATEMENT	5	5
8	REFERENCE CUSTOMERS	8	12
9	REFERENCE P_CUST_ID	9	26
10	REFERENCE CUSTOMER_ID	9	12
11	REFERENCE CUSTOMERS	6	20
12	REFERENCE CUST_FIRST_NAME	5	20
13	ASSIGNMENT P_CUST_NAME	7	12

14.5.2.4 SQL文のシグネチャについて

すべてのSQL文には、すべてのPL/SQLユニット内の文のインスタンスを識別する一意のPL/Scopeシグネチャがあります。

SQL文のシグネチャは、同じSQL_IDを使用するSQLのPL/SQLユニットからのコールと、他のPL/SQLユニットからの別のコールを区別します。

ネストされた副問合せは、ALL_STATEMENTS内の個別のSQL文ではありません。

例14-7 異なるPL/SQLユニットからコールされた場合の同じSQL文の個別のSQLシグネチャ

この例では、PROCEDURE p1およびp2からコールされたときの、同じSQL文の2つの異なるシグネチャを示しています。ネストされた副問合せに別のSQL_IDが割り当てられていないため、ALL_STATEMENTS内の個別のSQL文ではないことがわかります。

```

CREATE OR REPLACE PROCEDURE p1 (p_cust_id    NUMBER,
                                p_cust_name OUT VARCHAR2)
IS
BEGIN
    SELECT (SELECT CUST_FIRST_NAME
            FROM    CUSTOMERS)
    INTO    p_cust_name
    FROM    CUSTOMERS
    WHERE    CUSTOMER_ID = p_cust_id;
END;
/
CREATE OR REPLACE PROCEDURE P2 (p_cust_id    NUMBER,
                                p_cust_name OUT VARCHAR2)
IS

```

```

BEGIN
    SELECT (SELECT CUST_FIRST_NAME
            FROM CUSTOMERS)
    INTO   p_cust_name
    FROM   CUSTOMERS
    WHERE  CUSTOMER_ID = p_cust_id;
END;
/
ACCEPT nam CHAR PROMPT "Enter OBJECT_NAME : "
SELECT *
FROM   ALL_STATEMENTS
WHERE  OBJECT_NAME = '&&nam'
ORDER BY LINE, COL;

```

P1とP2のALL_STATEMENTSを選択して、同じSQL_IDの別のSQLシグネチャを確認します。

```

new      3: WHERE OBJECT_NAME = 'P1'
OE
138835D3A2EBBA76A7A064E4DC14B466 SELECT
P1
PROCEDURE          7          5          5          2 c02b6yppqb46p NO  NO
NO NO NO NO
YES SELECT (SELECT CUST_FIRST_NAME FROM CUSTOMERS) FROM CUSTOMERS WHERE CUSTOMER_ID
= :B1
SELECT (SELECT CUST_FIRST_NAME FROM CUSTOMERS) FROM CUSTOMERS WHERE CUSTOMER_ID
0
new      3: WHERE OBJECT_NAME = 'P2'
OE
E6A5E27E5E90997A169C5C25393FAB35 SELECT
P2
PROCEDURE          7          5          5          2 c02b6yppqb46p NO  NO
NO NO NO NO
YES SELECT (SELECT CUST_FIRST_NAME FROM CUSTOMERS) FROM CUSTOMERS WHERE CUSTOMER_ID
= :B1
SELECT (SELECT CUST_FIRST_NAME FROM CUSTOMERS) FROM CUSTOMERS WHERE CUSTOMER_ID
0

```

14.5.3 SQL Developer

PL/ScopeはSQL Developerの機能の1つです。SQL DeveloperからのPL/Scopeの使用の詳細は、SQL Developerのオンライン・ヘルプを参照してください。

関連項目:

[『Oracle SQL Developerユーザズ・ガイド』](#)

14.6 PL/SQLコードの管理に役立つデータ・ディクショナリ・ビューの概要

PL/Scopeデータ・ディクショナリ・ビューの他に、次に示す静的ディクショナリ・ビューがPL/SQLのプログラマにとって最も有用であり、PL/SQLコード管理レポートに関連する問合せで最も頻繁に参照されます。これは、すべての静的データ・ディクショナリ・ビューを網羅したリストではありません。

PL/SQLコードの管理に役立つデータ・ディクショナリ・ビューのサマリー

ビュー名	説明:
ALL_ARGUMENTS	現在のユーザーがアクセスできる関数およびプロシージャの引

ビュー名	説明:
	数をリストします
ALL_DEPENDENCIES	現在のユーザーがアクセスできるプロシージャ、パッケージ、関数、パッケージ本体およびトリガー間の依存関係を説明します。
ALL_ERRORS	現行のユーザーがアクセスできるストアド・オブジェクトでの現状のエラーを説明します
ALL_IDENTIFIERS	現行のユーザーがアクセスできるストアド・オブジェクトの識別子に関する情報を説明します
USER_OBJECT_SIZE	現行のユーザーが所有する PL/SQL オブジェクトのサイズ(バイト単位)を説明します。この情報はコンパイラとランタイム・エンジンで使用するものですが、この情報を使用すると、環境内の大きなプログラムを識別できます。
ALL_OBJECTS	現行ユーザーがアクセス可能なすべてのオブジェクトが表示されます。
ALL_PLSQL_OBJECT_SETTINGS	現行のユーザーがアクセスできるストアド・オブジェクトのコンパイラ設定の情報を説明します
ALL_PROCEDURES	現在のユーザーがアクセスできるすべての PL/SQL 関数とプロシージャ、および関連するプロパティを説明します
ALL_SEQUENCES	現行ユーザーがアクセス可能な順序を説明します
ALL_SOURCE	現行のユーザーがアクセスできるストアド・オブジェクトのテキスト・ソースを説明します
ALL_STATEMENTS	ユーザーがアクセスできるストアド PL/SQL オブジェクト内のすべての SQL 文を説明します
ALL_STORED_SETTINGS	現行のユーザーが実行権限を持つストアド PL/SQL ユニットの永続パラメータ設定を説明します
ALL_SYNONYMS	現行のユーザーがアクセスできるシノニムを説明します
ALL_TAB_COLUMNS	現行のユーザーがアクセスできる表、ビューおよびクラスタの列を説明します
ALL_TABLES	現行のユーザーがアクセスできるリレーショナル表を説明します
ALL_TRIGGERS	現行のユーザーがアクセスできる表のトリガーを説明します
ALL_VIEWS	現行のユーザーがアクセスできるビューを説明します

14.7 サンプルPL/Scopeセッション

次のサンプル・セッションでは、HRとしてログインしているものとします。

1. セッション・パラメータを設定します。

```
ALTER SESSION SET PLSCOPE_SETTINGS='IDENTIFIERS:ALL';
```

2. 次のパッケージを作成します。

```
CREATE OR REPLACE PACKAGE pack1 AUTHID DEFINER IS
  TYPE r1 IS RECORD (rf1 VARCHAR2(10));
  FUNCTION f1(fp1 NUMBER) RETURN NUMBER;
  PROCEDURE p1(pp1 VARCHAR2);
END PACK1;
/
CREATE OR REPLACE PACKAGE BODY pack1 IS
  FUNCTION f1(fp1 NUMBER) RETURN NUMBER IS
    a NUMBER := 10;
  BEGIN
    RETURN a;
  END f1;
  PROCEDURE p1(pp1 VARCHAR2) IS
    pr1 r1;
  BEGIN
    pr1.rf1 := pp1;
  END;
END pack1;
/
```

3. PL/Scopeがそのパッケージ本体のすべての識別子を収集していることを確認します。

```
SELECT PLSCOPE_SETTINGS
FROM USER_PLSQL_OBJECT_SETTINGS
WHERE NAME='PACK1' AND TYPE='PACKAGE BODY'
```

結果:

```
PLSCOPE_SETTINGS
-----
IDENTIFIERS:ALL
```

4. すべてのDECLARATIONの使用方法について問い合わせ、HRの一意の識別子を表示します。たとえば、%1のような名前の一意の識別子をすべて確認するには、次のSQL*Plus書式設定コマンドと問合せを使用します。

```
COLUMN NAME FORMAT A6
COLUMN SIGNATURE FORMAT A32
COLUMN TYPE FORMAT A9
SELECT NAME, SIGNATURE, TYPE
FROM USER_IDENTIFIERS
WHERE NAME LIKE '%1' AND USAGE='DECLARATION'
ORDER BY OBJECT_TYPE, USAGE_ID;
```

結果は次のようになります。

NAME	SIGNATURE	TYPE
PACK1	41820FA4D5EF6BE707895178D0C5C4EF	PACKAGE
R1	EEBB6849DEE31BC77BF186EBAE5D4E2D	RECORD
RF1	41D70040337349634A7F547BC83517C7	VARIABLE
F1	D51E825FF334817C977174423E3D0765	FUNCTION
FP1	CAC3474C112DBEC67AB926978D9A16C1	FORMAL IN
P1	B7C0576BA4D00C33A65CC0C64CADAB89	PROCEDURE
PP1	6B74CF95A5B7377A735925DFAA280266	FORMAL IN
FP1	98EB63B8A4AFEB5EF94D50A20165D6CC	FORMAL IN
PP1	62D8463A314BE1F996794723402278CF	FORMAL IN
PR1	BDB1CB26C97562CCC20CD1F32D341D7C	VARIABLE

10 rows selected.

*_IDENTIFIERS静的データ・ディクショナリ・ビューでは、基本的な型名のみが表示されます。たとえば、ローカル変数やレコード・フィールドのTYPEはVARIABLEです。VARIABLEの正確な型を確認するには、そのUSAGE_CONTEXT_IDを使用する必要があります。

5. すべてのローカル変数を検索します。

```
COLUMN VARIABLE_NAME FORMAT A13
COLUMN CONTEXT_NAME FORMAT A12
SELECT a.NAME variable_name,
       b.NAME context_name,
       a.SIGNATURE
FROM USER_IDENTIFIERS a, USER_IDENTIFIERS b
WHERE a.USAGE_CONTEXT_ID = b.USAGE_ID
AND a.TYPE = 'VARIABLE'
AND a.USAGE = 'DECLARATION'
AND a.OBJECT_NAME = 'PACK1'
AND a.OBJECT_NAME = b.OBJECT_NAME
AND a.OBJECT_TYPE = b.OBJECT_TYPE
AND (b.TYPE = 'FUNCTION' or b.TYPE = 'PROCEDURE')
ORDER BY a.OBJECT_TYPE, a.USAGE_ID;
```

結果は次のようになります。

VARIABLE_NAME	CONTEXT_NAME	SIGNATURE
A	F1	1691C6B3C951FCAA2CBEEB47F85CF128
PR1	P1	BDB1CB26C97562CCC20CD1F32D341D7C

2 rows selected.

6. ローカル変数Aに対して実行されたすべての使用方法を検索します。

```
COLUMN USAGE FORMAT A11
COLUMN USAGE_ID FORMAT 999
COLUMN OBJECT_NAME FORMAT A11
COLUMN OBJECT_TYPE FORMAT A12
SELECT USAGE, USAGE_ID, OBJECT_NAME, OBJECT_TYPE
FROM USER_IDENTIFIERS
WHERE SIGNATURE='1691C6B3C951FCAA2CBEEB47F85CF128' -- signature of A
ORDER BY OBJECT_TYPE, USAGE_ID;
```

結果:

USAGE	USAGE_ID	OBJECT_NAME	OBJECT_TYPE
DECLARATION	6	PACK1	PACKAGE BODY
ASSIGNMENT	8	PACK1	PACKAGE BODY
REFERENCE	9	PACK1	PACKAGE BODY

3 rows selected.

ローカル識別子Aに対して実行された使用方法は、識別子宣言(USAGE_ID 6)、代入(USAGE_ID 8)、および参照(USAGE_ID 9)です。

7. ローカル識別子Aの宣言から、その型を調べます。

```
COLUMN NAME FORMAT A6
COLUMN TYPE FORMAT A15
SELECT a.NAME, a.TYPE
FROM USER_IDENTIFIERS a, USER_IDENTIFIERS b
WHERE a.USAGE = 'REFERENCE'
AND a.USAGE_CONTEXT_ID = b.USAGE_ID
AND b.USAGE = 'DECLARATION'
AND b.SIGNATURE = 'D51E825FF334817C977174423E3D0765' -- signature of F1
```

```
AND a.OBJECT_TYPE = b.OBJECT_TYPE
AND a.OBJECT_NAME = b.OBJECT_NAME;
```

結果:

```
NAME    TYPE
-----
NUMBER  NUMBER DATATYPE
1 row selected.
```

8. ローカル識別子Aに対する代入が行われた場所を調べます。

```
SELECT LINE, COL, OBJECT_NAME, OBJECT_TYPE
FROM USER_IDENTIFIERS
WHERE SIGNATURE='1691C6B3C951FCAA2CBEEB47F85CF128' -- signature of A
AND USAGE='ASSIGNMENT';
```

結果:

```
LINE      COL OBJECT_NAME OBJECT_TYPE
-----
3         5 PACK1      PACKAGE BODY
1 row selected.
```

15 PL/SQL階層プロファイラの使用

PL/SQL階層プロファイラを使用すると、PL/SQLアプリケーションのボトルネックおよびパフォーマンスのチューニング機会を特定できます。

このプロファイラにより、ファンクション・コール別に編成されたPL/SQLプログラムの動的実行プロファイルがレポートされ、SQL実行時間とPL/SQL実行時間が個別に明確になります。特別なソースやコンパイル時間の準備は必要なく、あらゆるPL/SQLプログラムをプロファイリングできます。

この章では、PL/SQL階層プロファイラについて説明するとともに、これを使用してPL/SQLプログラムのプロファイル・データを収集して分析する方法を示します。

トピック:

- [PL/SQL階層プロファイラの概要](#)
- [プロファイル・データの収集](#)
- [RAWプロファイラ出力の理解](#)
- [プロファイル・データの分析](#)
- [plshprofユーティリティ](#)

15.1 PL/SQL階層プロファイラの概要

非階層(フラット)プロファイラにより、プログラムによって各サブプログラム内で費やされた時間(サブプログラムの関数時間または自己時間)が記録されます。関数時間は役に立ちますが、たいていこれでは十分ではありません。たとえば、サブプログラムINSERT_ORDERでプログラム全体の40%が費やされたことがわかれば参考になりますが、INSERT_ORDERを頻繁にコールするプログラムや、INSERT_ORDERの下(子サブプログラムを含む)で費やされた合計時間がわかればさらに便利です。階層プロファイラは、このような情報を提供します。

PL/SQL階層プロファイラには、次の機能があります。

- サブプログラム・コール別に編成されたPL/SQLプログラムの動的実行プロファイルのレポート
- SQL実行時間およびPL/SQL実行時間の個別の説明
- 特殊なソースやコンパイル時間の準備が不要
- カスタム・レポートを生成するために統合開発環境(IDE)ツール(SQL Developerやサード・パーティ・ツールなど)によって処理された結果のデータベース表(階層プロファイラ表)への格納
- 次のような、サブプログラム・レベルの実行サマリー情報の提供
 - サブプログラムへのコール回数
 - サブプログラム自体で費やされた時間(関数時間または自己時間)
 - サブプログラム自体およびその子サブプログラムで費やされた時間(サブツリー時間)
 - 次のような、詳細な親子情報
 - 特定のサブプログラムのすべてのコール元(親)
 - 特定のサブプログラムによってコールされたすべてのサブプログラム(子)

- サブプログラムxがyからコールされたときに費やされた時間
- サブプログラムxがyからコールされた回数

PL/SQL階層プロファイラは、DBMS_HPROFパッケージによって実装され、次の2つのコンポーネントを持ちます。

- データ収集

データ・コレクション・コンポーネントは、PL/SQL Virtual Machine固有のパーツです。DBMS_HPROFパッケージには、階層プロファイリングのオンとオフを切り替えるAPIが用意されています。RAWプロファイラ出力はファイルまたはRAWプロファイラ・データ表に書き込まれます。

- アナライザ

アナライザ・コンポーネントにより、RAWプロファイラ出力が処理され、分析の結果が生成されます。アナライザ・コンポーネントによって次が分析されます。

- RAWプロファイラ・データ・ファイルおよびHTML CLOBレポートへのRAWプロファイラ・データ表に存在するRAWプロファイラ・データ、分析されたレポート・ファイル、および階層プロファイラ分析表。



ノート:

アナライザを使用せずに、RAW プロファイラ出力から直接簡単な HTML レポートを生成するには、plshprof コマンドライン・ユーティリティを使用できます。

15.2 プロファイル・データの収集

PL/SQL階層プロファイラ用としてPL/SQLプログラムからプロファイル・データを収集するステップは、次のとおりです。

1. 次の権限があることを確認します。
 - DBMS_HPROFパッケージに対するEXECUTE権限
 - DBMS_HPROF.START_PROFILINGをコールするときに指定するディレクトリに対するWRITE権限
2. DBMS_HPROF.START_PROFILING PL/SQL APIを使用して、階層プロファイラのデータ・コレクションをセッションで開始します。
3. 十分な範囲のコードを取得できるまでPL/SQLプログラムを実行します。

経過時間を最大限正確に測定するために、PL/SQLプログラムが実行されているシステム上で関係のないアクティビティは実行しないようにしてください。
4. DBMS_HPROF.STOP_PROFILING PL/SQL APIを使用して、階層プロファイラのデータ・コレクションを終了します。

例15-1 PL/SQLプロシージャのプロファイリング

```
BEGIN
  /* Start profiling.
   Write raw profiler output to file test.trc in a directory
   that is mapped to directory object PLSHPROF_DIR
   (see note after example). */
  DBMS_HPROF.START_PROFILING('PLSHPROF_DIR', 'test.trc');
END;
/
-- Run procedure to be profiled
BEGIN
  test;
END;
```

```

/
BEGIN
  -- Stop profiling
  DBMS_HPROF.STOP_PROFILING;
END;
/

```

例15-2 RAWプロファイラ・データ表でのプロファイル

```

DECLARE
analyze_runid number;
trace_id number;
BEGIN
  -- create raw profiler data and analysis tables
  -- call create_tables with force_it =>FALSE ( default) when
  -- raw profiler data and analysis tables do not exist already
  DBMS_HPROF . CREATE_TABLES ;
  -- Start profiling .
  -- Write raw profiler data in raw profiler data table
  trace_id := DBMS_HPROF . START_PROFILING ;
  -- Run procedure to be profiled
  test ;
  -- Stop profiling
  DBMS_HPROF . STOP_PROFILING ;
  -- analyzes trace_id entry in raw profiler data table and writes
  -- hierarchical profiler information in hprof 's analysis tables.
  analyze_runid := DBMS_HPROF . ANALYZE(trace_id );
END;
/

```

次のPL/SQLプロシージャtestについて検討してみます。

```

CREATE OR REPLACE PROCEDURE test AUTHID DEFINER IS
  n NUMBER;
  PROCEDURE foo IS
  BEGIN
    SELECT COUNT(*) INTO n FROM EMPLOYEES;
  END foo;
BEGIN -- test
  FOR i IN 1..3 LOOP
    foo;
  END LOOP;
END test;
/

```

RAWプロファイラ・データ表を分析し、HTML CLOBレポートを生成するPL/SQLプロシージャについて検討してみます

```

declare
reportclob clob ;
trace_id number;
begin
  -- create raw profiler data and analysis tables
  -- force_it =>TRUE will dropped the tables if table existed
  DBMS_HPROF . CREATE_TABLES (force_it =>TRUE );
  -- Start profiling .
  -- Write raw profiler data in raw profiler data table
  trace_id := DBMS_HPROF . START_PROFILING ;
  -- Run procedure to be profiled
  test ;
  -- Stop profiling
  DBMS_HPROF . STOP_PROFILING ;
  -- analyzes trace_id entry in raw profiler data table and produce
  -- analyzed HTML report in reportclob .
  DBMS_HPROF .ANALYZE (trace_id , reportclob );
end;
/

```

[例15-1](#)のSQLスクリプトは、PL/SQLプロシージャtestの実行をプロファイリングしたものです。

ノート:

ディレクトリ・オブジェクトは、ファイル・システムのパス名の別名です。たとえば、データベース AS SYSDBA に接続している場合、次の CREATE DIRECTORY 文では、ディレクトリ・オブジェクト PLSHPROF_DIR が作成され、これがファイル・システム・ディレクトリ/private/plshprof/results にマップされます。



```
CREATE DIRECTORY PLSHPROF_DIR as '/private/plshprof/results';
```

[例 15-1](#) の SQL スクリプトを実行するには、PLSHPROF_DIR とマップ先のディレクトリの両方に対する READ および WRITE 権限が必要です。データベースに AS SYSDBA で接続している場合、次の GRANT 文では、PLSHPROF_DIR に対する READ および WRITE 権限が HR に付与されます。

```
GRANT READ, WRITE ON DIRECTORY PLSHPROF_DIR TO HR;
```

関連項目:

- ディレクトリ・オブジェクトの使用の詳細は、[『Oracle Database SecureFilesおよびラージ・オブジェクト開発者ガイド』](#)を参照してください。
- DBMS_HPROF.START_PROFILINGおよびDBMS_HPROF.STOP_PROFILING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。

15.3 RAWプロファイラ出力の理解

RAWプロファイラ出力は、PL/SQL階層プロファイラのアナライザ・コンポーネントによって処理されることを目的としています。ただし、このように処理されなくても、プログラムの簡単な関数レベルのトレースは提供できます。ここでは、RAWプロファイラ出力について説明します。

ノート:



この章で示す RAW プロファイラの形式は、RAW プロファイラ出力の概念的な機能を示すことのみを目的としています。形式の詳細は、Oracle Database リリースごとに異なる可能性があります。

[例15-1](#)のSQLスクリプトでは、次のRAWプロファイラ出力がファイルtest.trcに書き込まれました。

```
P#V PLSHPROF Internal Version 1.0
P#! PL/SQL Timer Started
P#C PLSQL."". ""."__plsql_vm"
P#X 4
P#C PLSQL."". ""."__anonymous_block"
P#X 77
P#C PLSQL."HR"."TEST": :7."TEST"#980980e97e42f8ec #1
P#X 4
P#C PLSQL."HR"."TEST": :7."TEST.F00"#980980e97e42f8ec #4
P#X 47
P#C SQL."HR"."TEST": :7."__static_sql_exec_line6" #6."3r6qf2qhr3cm1"
P#! SELECT COUNT(*) FROM EMPLOYEES
P#X 279
```



```

P#R
P#X 58
P#R
P#X 3
P#C PLSQL."HR"."TEST"::7."TEST.F00"#980980e97e42f8ec #4
P#X 4
P#C SQL."HR"."TEST"::7."__static_sql_exec_line6" #6."3r6qf2qhr3cm1"
P#! SELECT COUNT(*) FROM EMPLOYEES
P#X 121
P#R
P#X 5
P#R
P#X 2
P#C PLSQL."HR"."TEST"::7."TEST.F00"#980980e97e42f8ec #4
P#X 3
P#C SQL."HR"."TEST"::7."__static_sql_exec_line6" #6."3r6qf2qhr3cm1"
P#! SELECT COUNT(*) FROM EMPLOYEES
P#X 117
P#R
P#X 4
P#R
P#X 2
P#R
P#X 2
P#R
P#X 3
P#R
P#C PLSQL."".""."__plsql_vm"
P#X 3
P#C PLSQL."".""."__anonymous_block"
P#X 86
P#C PLSQL."SYS"."DBMS_HPROF"::11."STOP_PROFILING"#980980e97e42f8ec #453
P#R
P#R
P#R
P#! PL/SQL Timer Stopped

```

表15-1 RAWプロファイラ出力ファイルのインジケータ

インジケータ	意味
P#V	PLSHPROF バナーとバージョン番号
P#C	サブプログラムへのコール(コール・イベント)
P#R	サブプログラムからの戻り値(戻りイベント)
P#X	前のイベントと次のイベント間の経過時間
P#!	コメント

コール・イベント(P#C)と戻りイベント(P#R)は、(一対のカッコのように)正しくネストされます。未処理の例外が原因でコール先のサブプログラムが終了しても、対応する戻りイベントはレポートされます。

RAWプロファイラ出力の各コール・イベント(P#C)のエントリには、次の情報が含まれます。

- コール先のサブプログラムが属するネームスペース

- コール先のサブプログラムが定義されているPL/SQLモジュールの名前
- コール先のサブプログラムが定義されているPL/SQLモジュールのタイプ
- コール先のサブプログラムの名前

この名前は、「[特別なファンクション名](#)」の特別なファンクション名の1つである場合があります。

- コール先のサブプログラムのシグネチャのハッシュ・アルゴリズムを表す16進値

DBMS_HPROF.analyze PL/SQL APIにより、ハッシュ値が階層プロファイラ表に格納されます。これにより、ユーザーとDBMS_HPROF.analyzeの両方が、オーバーロードされたサブプログラム(同じ名前を持つサブプログラム)を区別できるようになります。

- コール先のサブプログラムがPL/SQLモジュールに定義されている行番号

PL/SQL階層プロファイラでは、モジュール内の個々の行で費やされた時間は測定されませんが、行番号を使用すると、(IDE/ツールのように)モジュール内のサブプログラムのソースの場所を特定し、オーバーロードされたサブプログラムを区別できるようになります。

たとえば、前述の例のRAWプロファイラ出力の次のエントリについて検討してみます。

```
P#C PLSQL."HR"."TEST": : 7."TEST.F00"#980980e97e42f8ec #4
```

前述のエントリの構成要素には、次のような意味があります。

コンポーネント	意味
PLSQL	PLSQL は、コール先のサブプログラムが属するネームスペースです。
"HR"."TEST"	HR.TEST は、コール先のサブプログラムが定義されている PL/SQL モジュールの名前です。
7	7 は、HR.TEST のモジュール・タイプの内部列挙子です。モジュール・タイプの例としては、プロシージャ、パッケージおよびパッケージ本体があります。
"TEST.F00"	TEST.F00 は、コール先のサブプログラムの名前です。
#980980e97e42f8ec	#980980e97e42f8ec は、TEST.F00 のシグネチャのハッシュ・アルゴリズムを表す 16 進値です。
#4	4 は、TEST.F00 が定義されている PL/SQL モジュール HR.TEST 内の行番号です。

ノート:



サブプログラムがインラインされている場合、プロファイラ出力ではレポートされません。

DETERMINISTIC ファンクションのコールが最適化されている場合、プロファイラ出力ではレポートされません。

関連項目:

- [追跡されるサブプログラムのネームスペース](#)
- analyze PL/SQL APIの詳細は、「[プロファイル・データの分析](#)」を参照してください。
- サブプログラムのインライン記述の詳細は、『[Oracle Database PL/SQL言語リファレンス](#)』を参照してください。
- DETERMINISTIC関数の詳細は、『[Oracle Database PL/SQL言語リファレンス](#)』を参照してください。

15.3.1 追跡されるサブプログラムのネームスペース

PL/SQL階層プロファイラが追跡するサブプログラムは、次のようにネームスペースPLSQLおよびSQLに分類されます。

- ネームスペースPLSQLには、次が含まれます。
 - PL/SQLサブプログラム・コール
 - PL/SQLトリガー
 - 無名PL/SQLブロック
 - リモート・サブプログラム・コール
 - パッケージ初期化ブロック
- ネームスペースSQLには、問合せ、データ操作言語(DML)文、データ定義言語(DDL)文およびネイティブ動的SQL文など、PL/SQLから実行されるSQL文が含まれます。

15.3.2 特別な関数名

PL/SQL階層プロファイラでは、特定の操作が[表15-2](#)に示す名前とネームスペースを持つ関数であるかのように追跡されます。

表15-2 PL/SQL階層プロファイラによって追跡される操作の関数名

追跡される操作	関数名	ネームスペース
PL/SQL Virtual Machine へのコール	__plsqli_vm	PL/SQL
無名 PL/SQL ブロック	__anonymous_block	PL/SQL
パッケージ初期化ブロック	__pkg_init	PL/SQL
行 line#の静的 SQL 文	__static_sql_exec_lineline#	SQL
行 line#の動的 SQL 文	__dyn_sql_exec_lineline#	SQL
行 line#の SQL FETCH 文	__sql_fetch_lineline#	SQL

15.4 プロファイル・データの分析

PL/SQL階層プロファイラのアナライザ・コンポーネントDBMS_HPROF.analyzeにより、RAWプロファイラ出力が処理され、表15-3に示す[階層データベース表](#)に結果が格納されます。

表15-3 PL/SQL階層プロファイラ・データベース表

表	説明:
DBMSHP_RUNS	この DBMS_HPROF.analyze の実行時のトップレベル情報。列の詳細は、 表 15-4 を参照してください。
DBMSHP_FUNCTION_INFO	この DBMS_HPROF.analyze の実行時にプロファイリングされる各サブプログラムの情報。列の詳細は、 表 15-5 を参照してください。
DBMSHP_PARENT_CHILD_INFO	この DBMS_HPROF.analyze の実行時にプロファイリングされる各サブプログラムの親子情報。列の詳細は、 表 15-6 を参照してください。

トピック:

- [階層プロファイラ表の作成](#)
- [階層プロファイラ表の理解](#)

ノート:



アナライザを使用せずに、RAW プロファイラ出力から直接簡単な HTML レポートを生成するには、plshprof コマンドライン・ユーティリティを使用できます。詳細は、[「plshprof ユーティリティ」](#)を参照してください。

15.4.1 階層プロファイラ表の作成

次のステップでは、[表15-3](#)にある階層プロファイラ表と、プロファイル・データを永続的に格納するために必要な他のデータ構造を作成する方法、DBMS_HPROFパッケージの実行に必要な権限、カスタム・レポートの生成について説明します。

1. [表15-3](#)にある階層プロファイラ表と、プロファイル・データを永続的に格納するために必要な他のデータ構造は、次の方法で作成できます。
 - a. DBMS_HPROF.CREATE_TABLESプロシージャをコールします。
 - b. スクリプトdbmshptab.sql(ディレクトリrdbms/adminにあります)を実行します。

ノート:



スクリプト dbmshptab.sql を実行すると、以前に作成されたすべての階層プロファイラ表が削除されます。

ノート:



dbmshtab.sql(ディレクトリ rdbms/admin 内)は非推奨になりました。このスクリプトには、表および順序を削除する文および非推奨のノートが含まれています。

2. 次の権限があることを確認します。

- a. DBMS_HPROFパッケージに対するEXECUTE権限
- b. DBMS_HPROF.analyzeで指定されるディレクトリに対するREAD権限

3. PL/SQL API DBMS_HPROF.analyzeを使用して、単一のRAWプロファイラ出力ファイルを分析し、結果を階層プロファイラ表に格納します。

(RAWプロファイラ出力ファイルの例は、[「RAWプロファイラ出力の理解」](#)にあるtest.trcを参照してください。)

DBMS_HPROF.analyze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。

4. 階層プロファイラ表を使用して、カスタム・レポートを生成します。

例15-3 DBMS_HPROF.analyzeの起動

```
DECLARE
  runid NUMBER;
BEGIN
  runid := DBMS_HPROF.analyze(LOCATION=>'PLSHPROF_DIR',
                              FILENAME=>'test.trc');
  DBMS_OUTPUT.PUT_LINE('runid = ' || runid);
END;
/
```

[例15-3](#)の無名ブロックでは、次の処理を行います。

- DBMS_HPROF.analyze機能の起動により、次が実行されます。
 - ディレクトリ・オブジェクトPLSHPROF_DIRにマップされたディレクトリにあるRAWプロファイラ出力ファイルtest.trc([「RAWプロファイラ出力の理解」](#)を参照)のプロファイル・データを分析し、データを[表15-3](#)の階層プロファイラ表に格納します。
 - [表15-3](#)の階層プロファイラ表に問い合わせるために使用できる一意の識別子を戻します。(これらの階層プロファイラ表に問い合わせることにより、カスタマイズ・レポートを作成できます。)
- 一意の識別子を、出力のための変数runidに保存します。

15.4.2 階層プロファイラ表の理解

次の各項では、[表15-3](#)の階層プロファイラ表を使用する方法について説明します。

- [階層プロファイラ・データベース表の列](#)
- [オーバーロードされたサブプログラムの区別](#)
- [サンプルのPL/SQLプロシージャの階層プロファイラ表](#)
- [オプションを使用したDBMS_HPROF.analyzeのコール例](#)

15.4.2.1 階層プロファイラ・データベース表の列

[表15-4](#)は、階層プロファイラ表DBMSHP_RUNSの列を示します。この表には、DBMS_HPROF.analyzeの実行時ごとに1行のトップレベル情報が含まれます。

階層プロファイラ表DBMSHP_RUNSの主キーはRUNIDです。

表15-4 DBMSHP_RUNS表の列

列名	列データ型	列の内容
RUNID	NUMBER	DBMSHP_RUNNUMBER 順序から生成された、この DBMS_HPROF.analyze の実行時の一意の識別子。
RUN_TIMESTAMP	TIMESTAMP(6)	この DBMS_HPROF.analyze の実行時のタイムスタンプ。
RUN_COMMENT	VARCHAR2(2047)	この DBMS_HPROF.analyze の実行時に指定したコメント。
TOTAL_ELAPSED_TIME	NUMBER(38)	この DBMS_HPROF.analyze の実行時の合計経過時間。

[表15-5](#)は、階層プロファイラ表DBMSHP_FUNCTION_INFOの列を示します。この表には、このDBMS_HPROF.analyzeの実行時にプロファイリングされたサブプログラムごとに1行の情報が含まれます。サブプログラムがオーバーロードされた場合、[表15-5](#)には、このサブシステムのバリエーションごとに1行が含まれます。各バリエーションには、独自のLINE#およびHASHがあります([「オーバーロードされたサブプログラムの区別」](#)を参照)。

階層プロファイラ表DBMSHP_FUNCTION_INFOの主キーはRUNID、SYMBOLIDです。

表15-5 DBMSHP_FUNCTION_INFO表の列

列名	列データ型	列の内容
RUNID	NUMBER	DBMSHP_RUNS 表の RUNID 列を参照します。この列の詳細は、 表 15-4 を参照してください。
SYMBOLID	NUMBER	サブプログラムの記号識別子(この DBMS_HPROF.analyze の実行時に固有)。
OWNER	VARCHAR2(128)	各サブプログラムが定義されているモジュールの所有者 (SYS または HR など)。
MODULE	VARCHAR2(128)	サブプログラムが定義されているモジュール (DBMS_LOB、UTL_HTTP または MY_PACKAGE など)。

列名	列データ型	列の内容
TYPE	VARCHAR2(32)	サブプログラムが定義されているモジュールのタイプ (PACKAGE、PACKAGE_BODY または PROCEDURE など)。
FUNCTION	VARCHAR2(4000)	サブプログラムの名前(必ずしも関数であるとはかぎりません)(INSERT_ORDER、PROCESS_ITEMS または TEST など)。 この名前は、「特別なファンクション名」の特別なファンクション名の 1 つである場合があります。 サブプログラム A 内で定義されているサブプログラム B の場合、この名前は A.B になります。 関数 X に対する再帰的コールは X@n として表されます。n は再帰の深さです。たとえば、X@1 は、X に対する第 1 再帰的コールです。
LINE#	NUMBER	FUNCTION が定義されている OWNER.MODULE 内の行番号。
HASH	RAW(32)	サブプログラムのシグネチャのハッシュ・コード(この DBMS_HPROF.analyze の実行時に固有)。
NAMESPACE	VARCHAR2(32)	サブプログラムのネームスペース。詳細は、 「追跡されるサブプログラムのネームスペース」 を参照してください。
SUBTREE_ELAPSED_TIME	NUMBER(38)	子サブプログラムで費やされた時間を含む、サブプログラムの経過時間(マイクロ秒単位)。
FUNCTION_ELAPSED_TIME	NUMBER(38)	子サブプログラムで費やされた時間を除く、サブプログラムの経過時間(マイクロ秒単位)。
CALLS	NUMBER(38)	サブプログラムに対するコールの数。
SQL_ID	VARCHAR2(13)	SQL 文の SQL 識別子。
SQL_TEXT	VARCHAR2(4000)	SQL 文の最初の 50 文字。

[表15-6](#)は、階層プロファイラ表DBMSHP_PARENT_CHILD_INFOの列を示します。この表には、この DBMS_HPROF.analyzeの実行時にプロファイリングされた一意の親子サブプログラムの組合せごとに1行の親子情報が含ま

れます。

表15-6 DBMSHP_PARENT_CHILD_INFO表の列

列名	列データ型	列の内容
RUNID	NUMBER	DBMSHP_FUNCTION_INFO 表の RUNID 列を参照します。この列の詳細は、 表 15-5 を参照してください。
PARENTSYMID	NUMBER	親記号 ID。 RUNID、PARENTSYMID は、DBMSHP_FUNCTION_INFO(RUNID、SYMBOLID)を参照します。
CHILDSYMID	NUMBER	子記号 ID。 RUNID、CHILDSYMID は、DBMSHP_FUNCTION_INFO(RUNID、SYMBOLID)を参照します。
SUBTREE_ELAPSED_TIME	NUMBER(38)	子サブプログラムで費やされた時間を含む、RUNID、PARENTSYMID からコールされたときの RUNID、CHILDSYMID の経過時間(マイクロ秒単位)。
FUNCTION_ELAPSED_TIME	NUMBER(38)	子サブプログラムで費やされた時間を除く、RUNID、PARENTSYMID からコールされたときの RUNID、CHILDSYMID の経過時間(マイクロ秒単位)。
CALLS	NUMBER(38)	RUNID、PARENTSYMID から RUNID、CHILDSYMID へのコールの数。

15.4.2.2 オーバーロードされたサブプログラムの区別

オーバーロードされたサブプログラムとは、同じ名前を持つ異なるサブプログラムです。

プログラムによってcomputeという名前の3つのサブプログラム(1番目は50行目、2番目は76行目、3番目は100行目)が宣言されるとします。DBMSHP_FUNCTION_INFO表では、これらのサブプログラムごとにFUNCTION列にcomputeがあります。これらの3つのサブプログラムを区別するには、LINE#列(1番目のサブプログラムは50、2番目は76、3番目は100)またはHASH列(サブプログラムごとに一意の値)を使用します。

2回の異なる実行時のプロファイル・データでは、関数のLINE#およびHASH値が異なる可能性があります。関数のLINE#値は、関数定義の前で行を挿入または削除したときに変更されます。HASH値は、パラメータ・リストを変更した場合など、関数のシグ

ネチャが変更されたときのみ変更されます。

関連項目:

オーバーロードされたサブプログラムの詳細は、『[Oracle Database PL/SQL言語リファレンス](#)』を参照してください。

15.4.2.3 サンプルのPL/SQLプロシージャの階層プロファイル表

「[プロファイル・データの収集](#)」のPL/SQLプロシージャtestの階層プロファイル表を[例15-4](#)から[例15-6](#)に示します。

DBMSHP_PARENT_CHILD_INFO表([例15-6](#))の第3行を検討してみます。RUNID列は、この行が3回目の実行時のものであることを示します。列PARENTSYMIDおよびCHILDSYMIDは、親(コール元)と子(コール先サブプログラム)の記号IDがそれぞれ2と1であることを示します。表DBMSHP_FUNCTION_INFO ([例15-5](#))は、3回目の実行時については記号ID 2と1がそれぞれ、プロシージャ__plsql_vmと__anonymous_blockに対応することを示します。したがって、この行の情報は、プロシージャ__plsql_vmから、モジュールHR.TESTの__anonymous_block (__plsql_vm内で定義)へのコールに関するものです。この行には、プロシージャ__plsql_vmからコールされたとき、プロシージャ__anonymous_blockの関数時間が44マイクロ秒で、__anonymous_blockサブツリー(子孫を含む)で費やされた時間が121マイクロ秒であると表示されます。

例15-4 サンプルのPL/SQLプロシージャのDBMSHP_RUNS表

RUNID	RUN_TIMESTAMP	TOTAL_ELAPSED_TIME	RUN_COMMENT
1	20-DEC-12 11.37.26.688381 AM	7	
2	20-DEC-12 11.37.26.700523 AM	9	
3	20-DEC-12 11.37.26.706824 AM	123	

例15-5 サンプルのPL/SQLプロシージャのDBMSHP_FUNCTION_INFO表

RUNID	SYMBOLID	OWNER	MODULE	TYPE	FUNCTION
1	1	HR	PKG	PACKAGE BODY	MYPROC
2	1	HR	PKG	PACKAGE BODY	MYFUNC
2	2	HR	PKG	PACKAGE BODY	MYPROC
3	1				__anonymous_block
3	2				__plsql_vm
3	3	HR	PKG	PACKAGE BODY	MYFUNC
3	4	HR	PKG	PACKAGE BODY	MYPROC
3	5	HR	TEST2	PROCEDURE	TEST2
LINE#	CALLS	HASH			
NAMESPACE					
2	1	9689BA467A19CD19			PLSQL
7	1	28DC3402BAEB2B0D			PLSQL
2	1	9689BA467A19CD19			PLSQL
0	1				PLSQL
0	1				PLSQL
7	1	28DC3402BAEB2B0D			PLSQL
2	2	9689BA467A19CD19			PLSQL
1	1	980980E97E42F8EC			PLSQL
NAMESPACE			SUBTREE_ELAPSED_TIME		FUNCTION_ELAPSED_TIME
PLSQL			7		7
PLSQL			9		8
PLSQL			1		1
PLSQL			121		44
PLSQL			123		2
PLSQL			9		8
PLSQL			8		8
PLSQL			77		61

例15-6 サンプルのPL/SQLプロシージャのDBMSHP_PARENT_CHILD_INFO表

CALLS	RUNID	PARENTSYMID	CHILDSYMID	SUBTREE_ELAPSED_TIME	FUNCTION_ELAPSED_TIME
1	2	1	2	1	1
1	3	1	5	77	61
1	3	2	1	121	44
1	3	3	4	1	1
1	3	5	3	9	8
1	3	5	4	7	7

15.4.2.4 オプションを使用したDBMS_HPROF.analyzeのコール例

オプションを使用しないDBMS_HPROF.analyzeのコール例は、[例15-3](#)を参照してください。

[例15-7](#)では、パッケージを作成し、パッケージ内のサブプログラムを起動するプロシージャを作成し、プロシージャをプロファイリングし、RAWプロファイル出力ファイルの分析にDBMS_HPROF.analyzeを使用します。RAWプロファイル出力ファイルは、PLSHPROF_DIRディレクトリ・オブジェクトに対応するディレクトリにあります。

例15-7 オプションを使用したDBMS_HPROF.analyzeの起動

```
-- Create package
CREATE OR REPLACE PACKAGE pkg AUTHID DEFINER IS
  PROCEDURE myproc (n IN out NUMBER);
  FUNCTION myfunc (v VARCHAR2) RETURN VARCHAR2;
  FUNCTION myfunc (n PLS_INTEGER) RETURN PLS_INTEGER;
END pkg;
/

CREATE OR REPLACE PACKAGE BODY pkg IS
  PROCEDURE myproc (n IN OUT NUMBER) IS
  BEGIN
    n := n + 5;
  END;
  FUNCTION myfunc (v VARCHAR2) RETURN VARCHAR2 IS
    n NUMBER;
  BEGIN
    n := LENGTH(v);
    myproc(n);
    IF n > 20 THEN
      RETURN SUBSTR(v, 1, 20);
    ELSE
      RETURN v || '...';
    END IF;
  END;
  FUNCTION myfunc (n PLS_INTEGER) RETURN PLS_INTEGER IS
    i PLS_INTEGER;
    PROCEDURE myproc (n IN out PLS_INTEGER) IS
    BEGIN
      n := n + 1;
    END;
  BEGIN
    i := n;
    myproc(i);
    RETURN i;
  END;
END pkg;
/
```

```

-- Create procedure that invokes package subprograms
CREATE OR REPLACE PROCEDURE test2 AUTHID DEFINER IS
  x NUMBER := 5;
  y VARCHAR2(32767);
BEGIN
  pkg.myproc(x);
  y := pkg.myfunc('hello');
END;
-- Profile test2
BEGIN
  DBMS_HPROF.START_PROFILING('PLSHPROF_DIR', 'test2.trc');
END;
/
BEGIN
  test2;
END;
/
BEGIN
  DBMS_HPROF.STOP_PROFILING;
END;
/
-- If not done, create hierarchical profiler tables (see Creating Hierarchical Profiler Tables).
-- Call DBMS_HPROF.analyze with options
DECLARE
  runid NUMBER;
BEGIN
  -- Analyze only subtrees rooted at trace entry "HR"."PKG"."MYPROC"
  runid := DBMS_HPROF.analyze('PLSHPROF_DIR', 'test2.trc',
                             trace => 'HR"."PKG"."MYPROC');
  -- Analyze up to 20 calls to subtrees rooted at trace entry
  -- "HR"."PKG"."MYFUNC". Because "HR"."PKG"."MYFUNC" is overloaded,
  -- both overloads are considered for analysis.
  runid := DBMS_HPROF.analyze('PLSHPROF_DIR', 'test2.trc',
                             collect => 20,
                             trace => 'HR"."PKG"."MYFUNC');
  -- Analyze second call to PL/SQL virtual machine
  runid := DBMS_HPROF.analyze('PLSHPROF_DIR', 'test2.trc',
                             skip => 1, collect => 1,
                             trace => '""."".""__plsql_vm');
END;
/

```

15.5 plshprofユーティリティ

plshprofコマンドライン・ユーティリティ(ディレクトリ\$ORACLE_HOME/bin/にあります)により、1つまたは2つのRAWプロファイラ出力ファイルから簡単なHTMLレポートが生成されます。(RAWプロファイラ出力ファイルの例は、[「プロファイル・データの収集」](#)にあるtest.trcを参照してください。)

生成されたHTMLレポートは任意のブラウザで参照できます。ブラウザのナビゲーション機能と厳選したリンクを組み合わせた効率的な手段により、大規模なアプリケーションのパフォーマンスを分析し、アプリケーションのパフォーマンスを向上させ、開発コストを削減できます。

トピック:

- [plshprofのオプション](#)
- [単一のRAWプロファイラ出力ファイルから生成するHTMLレポート](#)
- [2つのRAWプロファイラ出力ファイルから生成するHTML差分レポート](#)

15.5.1 plshprofのオプション

plshprofユーティリティを実行するコマンドは、次のとおりです。

```
plshprof [option...] profiler_output_filename_1 profiler_output_filename_2
```

optionは、次のいずれかを使用します。

オプション	説明:	デフォルト
-skip count	最初の count コールをスキップします。-trace symbol とともにのみ使用します。	0
-collect count	count コールに関する情報を収集します。-trace symbol とともにのみ使用します。	1
-output filename	出力ファイルの名前を指定します。	symbol.html または tracefile1.html
-summary	経過時間のみを出力します。	なし
-trace symbol	ツリー・ルートの関数名を指定します。	該当なし

RAWプロファイラ出力ファイルtest.trcが現行のディレクトリ内にあるとします。また、HTMLレポートを分析および生成し、HTMLレポートのルート・ファイルの名前をreport.htmlにするとします。次のコマンドを使用します(%はプロンプトです)。

```
% plshprof -output report test.trc
```

15.5.2 単一のRAWプロファイラ出力ファイルから生成するHTMLレポート

単一のRAWプロファイラ出力ファイルからPL/SQL階層プロファイラのHTMLレポートを生成するには、次のコマンドを使用します。

```
% cd target_directory
% plshprof -output html_root_filename profiler_output_filename
```

target_directoryは、HTMLファイルを作成するディレクトリです。

html_root_filenameは、作成するルートHTMLファイルの名前です。

profiler_output_filenameは、RAWプロファイラ出力ファイルの名前です。

前述のplshprofコマンドにより、一連のHTMLファイルが生成されます。html_root_filename.htmlからこれらの参照を開始します。

トピック:

- [レポートの最初のページ](#)
- [関数レベルのレポート](#)
- [PL/SQL階層プロファイラのSQLレベルのレポートの理解](#)
- [モジュール・レベルのレポート](#)

- [ネームスペース・レベルのレポート](#)
- [関数の親子レポート](#)

15.5.2.1 レポートの最初のページ

単一のRAWプロファイラ出力ファイルから生成するHTMLレポートの最初のページには、サマリー情報やレポートの他のページへのハイパーリンクが含まれます。

最初のページのサンプル

PL/SQLの経過時間(マイクロ秒)の分析

824マイクロ秒(経過時間)および12のファンクション・コール

PL/SQL階層プロファイラにより、様々な形式でプロファイラの出力ログから導出された情報が示されたレポートの集合が作成されます。一般的に、最初の参照レポートとしては次のレポートが最も役に立つことがわかっています。

- サブツリーの合計経過時間(マイクロ秒)別にソートした関数の経過時間(マイクロ秒)データ
- 関数の合計経過時間(マイクロ秒)別にソートした関数の経過時間(マイクロ秒)データ
- SQL ID(マイクロ秒)別にソートしたSQL IDの経過時間(マイクロ秒)データ

また、次のレポートも使用できます。

- 関数名別にソートした関数の経過時間(マイクロ秒)データ
- 子の合計経過時間(マイクロ秒)別にソートした関数の経過時間(マイクロ秒)データ
- 関数の合計コール件数別にソートした関数の経過時間(マイクロ秒)データ
- サブツリーの平均経過時間(マイクロ秒)別にソートした関数の経過時間(マイクロ秒)データ
- 関数の平均経過時間(マイクロ秒)別にソートした関数の経過時間(マイクロ秒)データ
- 子の平均経過時間(マイクロ秒)別にソートした関数の経過時間(マイクロ秒)データ
- 関数の合計経過時間(マイクロ秒)別にソートしたモジュールの経過時間(マイクロ秒)データ
- モジュール名別にソートしたモジュールの経過時間(マイクロ秒)データ
- 関数の合計コール件数別にソートしたモジュールの経過時間(マイクロ秒)データ
- 関数の合計経過時間(マイクロ秒)別にソートしたネームスペースの経過時間(マイクロ秒)データ
- ネームスペース別にソートしたネームスペースの経過時間(マイクロ秒)データ
- 関数の合計コール件数別にソートしたネームスペースの経過時間(マイクロ秒)データ
- 親子の経過時間(マイクロ秒)データ

15.5.2.2 関数レベルのレポート

関数レベルのレポートは、プロファイル情報のフラット・ビューを提供します。関数レベルの各レポートには、関数ごとに次の情報が含まれます。

- 関数時間(関数自体で費やされた時間で、自己時間とも呼ばれます)
- 子の時間(関数の子で費やされた時間)
- サブツリー時間(関数のサブツリーで費やされた時間—関数時間と子の時間の合計)

- 関数に対するコールの数

- 関数名

関数名からは、関数の親子レポートへのハイパーリンクが作成されます。

- SQL ID

- SQLテキスト(SQLテキストの最初の50文字)。

関数レベルの各レポートは、関数時間またはサブツリー時間などの特定の属性によってソートされます。

次のサンプル・レポートは、関数のサブツリーの合計経過時間の降順でソートされています。「サブツリー」および「Ind%」列の情報がbold typeであるのはこのためです。

サンプル・レポート

サブツリーの合計経過時間(マイクロ秒)別にソートした関数の経過時間(マイクロ秒)データ

824マイクロ秒(経過時間)および12のファンクション・コール

サブツリー	Ind %	関数	子	Ind %	コール	Ind %	関数名	SQL ID	SQL TEXT
824	100 %	10	814	98.8 %	2	16.7 %	__plsq_vm		
814	98.8 %	165	649	78.8 %	2	16.7 %	__anonymous_block		
649	78.8 %	11	638	77.4 %	1	8.3 %	HR.TEST.TEST (1 行目)		
638	77.4 %	121	517	62.7 %	3	25.0 %	HR.TEST.TEST.FOO (4 行目)		
517	62.7 %	517	0	0.0 %	3	25.0 %	HR.TEST.__static_sql__exec_line5(6 行目)	3r6qf2qhr3cm1	SELECT COUNT(*) FROM EMPLOYEES
0	0.0 %	0	0	0.0 %	1	8.3 %	SYS.DBMS_HPROF.STOP_PROFILING(453 行目)		

15.5.2.3 モジュール・レベルのレポート

モジュール・レベルの各レポートには、モジュールごとに次の情報が含まれます(パッケージまたはタイプなど)。

- モジュール時間(モジュールで費やされた時間—モジュール内のすべての関数の関数時間の合計)

- モジュール内の関数に対するコールの数

モジュール・レベルの各レポートは、モジュール時間またはモジュール名などの特定の属性によってソートされます。

次のサンプル・レポートは、モジュール時間でソートされています。「モジュール」、「Ind%」および「Cum%」列の情報がbold typeであるのはこのためです。

サンプル・レポート

関数の合計経過時間(マイクロ秒)別にソートしたモジュールの経過時間(マイクロ秒)データ

166878マイクロ秒(経過時間)および1099のファンクション・コール

モジュール	Ind%	Cum%	コール	Ind%	モジュール名
84932	50.9%	50.9%	6	0.5%	HR.P
67749	40.6%	91.5%	216	19.7%	SYS.DBMS_LOB
13340	8.0%	99.5%	660	60.1%	SYS.UTL_FILE
839	0.5%	100%	214	19.5%	SYS.UTL_RAW
18	0.0%	100%	2	0.2%	HR.UTILS
0	0.0%	100%	1	0.1%	SYS.DBMS_HPROF

15.5.2.4 ネームスペース・レベルのレポート

ネームスペース・レベルの各レポートには、ネームスペースごとに次の情報が含まれます。

- ネームスペース時間(ネームスペースで費やされた時間—ネームスペース内のすべての関数の関数時間の合計)
- ネームスペース内の関数に対するコールの数

ネームスペース・レベルの各レポートは、ネームスペース時間またはネームスペース内の関数に対するコールの数などの、特定の属性によってソートされます。

次のサンプル・レポートは、関数時間でソートされています。「関数」、「Ind%」および「Cum%」列の情報がbold typeであるのはこのためです。

サンプル・レポート

関数の合計経過時間(マイクロ秒)別にソートしたネームスペースの経過時間(マイクロ秒)データ

166878マイクロ秒(経過時間)および1099のファンクション・コール

関数	Ind%	Cum%	コール	Ind%	ネームスペース
93659	56.1%	56.1%	1095	99.6%	PLSQL

関数	Ind%	Cum%	コール	Ind%	ネームスペース
73219	43.9%	100%	4	0.4%	SQL

15.5.2.5 関数の親子レポート

親子レポートは、プロファイラによって追跡される関数ごとに、親(コール元の関数)と子(コール先の関数)に関する情報を提供します。このレポートは、親ごとに、関数の実行プロファイル(サブツリー時間、関数時間、子の時間、コール数)を提供します。また、子ごとに、この関数からコールされたときの子の実行プロファイルを提供します(ただし、他の関数からコールされたときは除きます)。

関数の実行プロファイルには、関数レベルのレポートで各関数について含まれる情報と同じ情報が含まれます。

後述の「[サンプル・レポート](#)」は、HR.P.UPLOADという名前の関数に対応する親子レポートの断片です。最初の行には、次のサマリー情報があります。

- 関数HR.P.UPLOADに対するコールは2つあります。
- 関数の合計サブツリー時間は166,860マイクロ秒で、そのうち11,713マイクロ秒(7.0%)は関数自体、155,147マイクロ秒(93.0%)は子の時間です。

行「親」の後には、HR.P.UPLOADの2つの親の実行プロファイル行であるHR.UTILS.COPY_IMAGEとHR.UTILS.COPY_FILEがあります。

HR.UTILS.COPY_IMAGEの最初の親実行プロファイル行は、次を示しています。

- HR.UTILS.COPY_IMAGEはHR.P.UPLOADを1回コールしており、これはHR.P.UPLOADに対するコールの50%を占めます。
- HR.UTILS.COPY_IMAGEからコールされたときのHR.P.UPLOADのサブツリー時間は106,325マイクロ秒で、これはHR.P.UPLOADの合計サブツリー時間の63.7%を占めます。
- HR.UTILS.COPY_IMAGEからコールされたときのHR.P.UPLOADの関数時間は6,434マイクロ秒で、これはHR.P.UPLOADの合計関数時間の54.9%を占めます。

行「子」の後には、HR.P.UPLOADからコールされたときのHR.P.UPLOADの子の実行プロファイル行があります。

SYS.UTL_FILE.GET_RAWに対応する3番目の子の実行プロファイル行は、次を示しています。

- HR.P.UPLOADは、SYS.UTL_FILE.GET_RAWを216回コールしています。
- HR.P.UPLOADからコールされたときのSYS.UTL_FILE.GET_RAWのサブツリー時間、関数時間および子の時間はそれぞれ12,487マイクロ秒、3,969マイクロ秒および8,518マイクロ秒です。
- HR.P.UPLOADの子の合計時間(155,147マイクロ秒)のうち、子SYS.UTL_FILE.GET_RAWは12,487マイクロ秒(8.0%)を占めます。

サンプル・レポート

HR.P.UPLOAD (3行目)

サブツリー	Ind %	関数	Ind %	子	Ind %	コール	Ind %	関数名
166860	100	11713	7.0%	155147	93.0	2	0.2%	HR.P.UPLOAD (3 行目)

サブツリー	Ind %	関数	Ind %	子	Ind %	コール	Ind %	関数名
	%				%			
親:								
106325	63.7 %	6434	54.9 %	99891	64.4 %	1	50.0 %	HR.UTILS.COPY_IMAGE (3 行目)
60535	36.3 %	5279	45.1 %	55256	35.6 %	1	50.0 %	HR.UTILS.COPY_FILE (8 行目)
子:								
71818	46.3 %	71818	100 %	0	N/A	2	100 %	HR.P.__static_sql_exec_line38(38 行目)
67649	43.6 %	67649	100 %	0	N/A	214	100 %	SYS.DBMS_LOB.WRITEAPPEND (926 行目)
12487	8.0%	3969	100 %	8518	100 %	216	100 %	SYS.UTL_FILE.GET_RAW(1089 行目)
1401	0.9%	1401	100 %	0	N/A	2	100 %	HR.P.__static_sql_exec_line39(39 行目)
839	0.5%	839	100 %	0	N/A	214	100 %	SYS.UTL_FILE.GET_RAW(246 行目)
740	0.5%	73	100 %	667	100 %	2	100 %	SYS.UTL_FILE.FOPEN (422 行目)
113	0.1%	11	100 %	102	100 %	2	100 %	SYS.UTL_FILE.FCLOSE (585 行目)
100	0.1%	100	100 %	0	N/A	2	100 %	SYS.DBMS_LOB.CREATETEMPORARY (536 行目)

関連項目:

15.5.2.6 PL/SQL階層プロファイラのSQLレベルのレポートの理解

DBMS_HPROF.ANALYZEのSQLレベルのレポートについて理解します。

PL/SQL階層プロファイラSQLレベル・レポートは、プロファイリング中に収集されたすべてのSQLのリストと略記されたSQLテキストおよびSQL IDでソートされた経過時間(マイクロ秒)を提供します。SQL IDは他のSQL統計を他の表で取得する必要がある場合(たとえば、SQLチューニング目的)に便利です。レポートには、SQLテキストの最初の50文字が含まれます。SQLがコールされる場所の周囲の詳細および必要に応じてソース・コード内の場所を取得するには、ファンクション・レベルのレポートを使用できます。

サンプル・レポート

SQL ID(マイクロ秒)別にソートしたSQL IDの経過時間(マイクロ秒)データ

824マイクロ秒(経過時間)および12のファンクション・コール

SQL ID	SQL TEXT	関数	Ind%	コール	Ind%
3r6qf2qhr3cm 1	SELECT COUNT(*) FROM EMPLOYEES	679	82.4 %	3	25.0 %

15.5.3 2つのRAWプロファイラ出力ファイルから生成するHTML差分レポート

2つのRAWプロファイラ出力ファイルからPL/SQL階層プロファイラのHTML差分レポートを生成するには、次のコマンドを使用します。

```
% cd target_directory
% plshprof -output html_root_filename profiler_output_filename_1
profiler_output_filename_2
```

target_directoryは、HTMLファイルを作成するディレクトリです。

html_root_filenameは、作成するルートHTMLファイルの名前です。

profiler_output_filename_1およびprofiler_output_filename_2は、RAWプロファイラ出力ファイルの名前です。

前述のplshprofコマンドにより、一連のHTMLファイルが生成されます。html_root_filename.htmlからこれらの参照を開始します。

トピック:

- [差分レポートの表記規則](#)
- [差分レポートの最初のページ](#)
- [関数レベルの差分レポート](#)
- [モジュール・レベルの差分レポート](#)
- [ネームスペース・レベルの差分レポート](#)
- [関数の親子差分レポート](#)

15.5.3.1 差分レポートの表記規則

差分レポートでは、次の表記規則が使用されます。

- レポート・タイトルでは、デルタは差分または変更を意味します。
- 正数は、初回実行時から2回目の実行時にかけて数値が増加(低下)したことを示します。
- 差分の負数は、初回実行時から2回目の実行時にかけて数値が減少(向上)したことを示します。
- 関数名の後ろの記号#は、関数が1回の実行のみでコールされたことを意味します。

15.5.3.2 差分レポートの最初のページ

2つのRAWプロファイラ出力ファイルから生成するHTML差分レポートの最初のページには、サマリー情報およびレポートの他のページへのハイパーリンクが含まれます。

最初のページのサンプル

PL/SQLの経過時間(マイクロ秒)の分析 - サマリー・ページ

この分析により、正味2709589マイクロ秒(経過時間)または80%(3393719対6103308)の回帰が判明します。これは、最も重要な7つの個別関数の回帰および向上状況をまとめたものです。

回帰: 3399382マイクロ秒(経過時間)

関数	相対%	Ind%	コール	相対%	関数名
2075627	+941%	61.1%	0		HR.P.G (35 行目)
1101384	+54.6%	32.4%	5	+55.6%	HR.P.H (18 行目)
222371		6.5%	1		HR.P.J (10 行目)

向上: 689793マイクロ秒(経過時間)

関数	相対%	Ind%	コール	相対%	関数名
-467051	-50.0%	67.7%	-2	-50.0%	HR.P.I (25 行目)
-222737		32.3%	-1		HR.P.I (2 行目)#
-5	-21.7%	0.0%	0		HR.P.TEST (46 行目)

PL/SQLタイミング・アナライザにより、様々な形式でプロファイラの出力ログから導出された情報が示されたレポートの集合が作成されます。一般的に、最初の参照レポートとしては次のレポートが最も役に立つことがわかっています。

- パフォーマンス回帰に関する関数の経過時間(マイクロ秒)データ
- パフォーマンス向上に関する関数の経過時間(マイクロ秒)データ

また、次のレポートも使用できます。

- 関数名別にソートした関数の経過時間(マイクロ秒)データ
- サブツリーの合計経過時間(マイクロ秒)デルタ別にソートした関数の経過時間(マイクロ秒)データ
- 関数の合計経過時間(マイクロ秒)デルタ別にソートした関数の経過時間(マイクロ秒)データ
- 子の合計経過時間(マイクロ秒)デルタ別にソートした関数の経過時間(マイクロ秒)データ
- 関数の合計コール件数デルタ別にソートした関数の経過時間(マイクロ秒)データ
- モジュール名別にソートしたモジュールの経過時間(マイクロ秒)データ
- 関数の合計経過時間(マイクロ秒)デルタ別にソートしたモジュールの経過時間(マイクロ秒)データ
- 関数の合計コール件数デルタ別にソートしたモジュールの経過時間(マイクロ秒)データ
- ネームスペース別にソートしたネームスペースの経過時間(マイクロ秒)データ
- 関数の合計経過時間(マイクロ秒)別にソートしたネームスペースの経過時間(マイクロ秒)データ
- 関数の合計コール件数別にソートしたネームスペースの経過時間(マイクロ秒)データ
- 親子のファイル経過時間(マイクロ秒)データ比較

15.5.3.3 関数レベルの差分レポート

関数レベルの各差分レポートには、各関数の、初回実行時から2回目の実行時にかけての次の値の変更が含まれます。

- 関数時間(関数自体で費やされた時間で、自己時間とも呼ばれます)
- 子の時間(関数の子で費やされた時間)
- サブツリー時間(関数のサブツリーで費やされた時間—関数時間と子の時間の合計)
- 関数に対するコールの数
- 平均関数時間

平均関数時間は、関数時間を関数へのコール数で割った値です。

- 関数名

関数名からは、関数の親子差分レポートへのハイパーリンクが作成されます。

[「サンプル・レポート1」](#)のレポートは、2回目の実行時より初回実行時の方がパフォーマンスがよかったすべての関数に関する差分情報を示します。次の点に注意してください。

- HR.P.Gの場合、関数時間は2,075,627マイクロ秒(941%)増加し、これはすべての回帰の61.1%を占めます。
- HR.P.Hの場合、関数時間およびコール数はそれぞれ1,101,384マイクロ秒(54.6%)および5(55.6%)増加しましたが、平均関数時間は-1,346マイクロ秒(-0.6%)向上しました。
- HR.P.Jは、1回の実行でのみコールされました。

サンプル・レポート 1

パフォーマンス回帰に関する関数の経過時間(マイクロ秒)データ

サブツリー	関数	Ind		Cum		子	コール	相対%	平均関数	相対%	関数名
		相対%	%	%	%						

サブツリー	関数	相対%	Ind %	Cum %	子	コール	相対%	平均関数	相対%	関数名
4075787	2075627	+941 %	61.1 %	61.1 %	2000160	0		2075627	+941 %	HR.P.G (35 行目)
1101384	1101384	+54.6 %	32.4 %	93.5 %	0	5	+55.6 %	-1346	- 0.6%	HR.P.H (18 行目)
222371	222371		6.5 %	100%	0	1				HR.P.J (10 行目)#

[「サンプル・レポート2」](#)のレポートは、初回実行時より2回目の実行時の方がパフォーマンスがよかったすべての関数に関する差分情報を示します。

サンプル・レポート 2

パフォーマンス向上に関する関数の経過時間(マイクロ秒)データ

サブツリー	関数	相対%	Ind %	Cum %	子	コール	相対%	平均関数	相対%	関数名
-1365827	-467051	- 50.0 %	67.7 %	67.7 %	-898776	-2	- 50.0 %	-32	0.0%	HR.P.I (25 行目)
-222737	-222737		32.3 %	100%	0	-1				HR.P.I (2 行目)
2709589	-5	- 21.7 %	0.0 %	100%	2709594	0		-5	-20.8	HR.P.TEST (46 行目)#

[「サンプル・レポート3」](#)のレポートには、すべての関数の差分情報をまとめます。

サンプル・レポート 3

関数の合計コール件数デルタ別にソートした関数の経過時間(マイクロ秒)データ

サブツリー	関数	相対%	Ind %	子	コール	相対%	平均関数	相対%	関数名
1101384	1101384	+54.6 %	32.4 %	0	5	+55.6 %	-1346	- 0.6%	HR.P.H (18 行目)

サブツリー	関数	相 対%	Ind %	子	コール	相 対%	平均関数	相 対%	関数名
- 1365827	-467051	+50.0 %	67.7 %	-898776	-2	- 50.0 %	-32	- 0.0%	HR.P.I (25 行目)
-222377	-222377		32.3 %	0	-1				HR.P.I (2 行目)#
222371	222371		6.5 %	0	1				HR.P.J (10 行目)#
4075787	2075627	+941 %	61.1 %	2000160	0		2075627	+941 %	HR.P.G (35 行目)
2709589	-5	- 21.7 %	0.0 %	2709594	0		-5	- 20.8 %	HR.P.TEST (46 行 目)
0	0			0	0				SYS.DBMS_HPROF. STOP_PROFILING(53 行目)

15.5.3.4 モジュール・レベルの差分レポート

モジュール・レベルの各レポートには、各モジュールの、初回実行時から2回目の実行時にかけての次の値の変更が含まれます。

- モジュール時間(モジュールで費やされた時間ーモジュール内のすべての関数の関数時間の合計)
- モジュール内の関数に対するコールの数

サンプル・レポート

関数の合計経過時間(マイクロ秒)デルタ別にソートしたモジュールの経過時間(マイクロ秒)データ

モジュール	コール	モジュール名
2709589	3	HR.P
0	0	SYS.DBMS_HPROF

15.5.3.5 ネームスペース・レベルの差分レポート

ネームスペース・レベルの各レポートには、各ネームスペースの、初回実行時から2回目の実行時にかけての次の値の変更が含まれます。

- ネームスペース時間(ネームスペースで費やされた時間—ネームスペース内のすべての関数の関数時間の合計)
- ネームスペース内の関数に対するコールの数

サンプル・レポート

ネームスペース別にソートしたネームスペースの経過時間(マイクロ秒)データ

関数	コール	ネームスペース
2709589	3	PLSQL

15.5.3.6 関数の親子差分レポート

関数の親子差分レポートは、初回実行時から2回目の実行時にかけての次の実行プロファイルの変更を示します。

- 親(関数のコール元の関数)
- 子(関数のコール先の関数)

子の実行プロファイルには、他の関数によってこれらがコールされたときではなく、この関数によってこれらがコールされたときの情報のみが含まれます。

関数の実行プロファイルには、次の情報が含まれます。

- 関数時間(関数自体で費やされた時間で、自己時間とも呼ばれます)
- 子の時間(関数の子で費やされた時間)
- サブツリー時間(関数のサブツリーで費やされた時間—関数時間と子の時間の合計)
- 関数に対するコールの数
- 関数名

サンプル・レポートは、HR.P.Xという名前の関数に対応する親子差分レポートの断片です。

初回実行時と2回目の実行時の差分のサマリーである最初の行は、回帰を示します。関数時間は1,094,099マイクロ秒増加しています(おそらく、関数が5回余計にコールされたことが原因です)。

「親」行は、HR.P.GによってHR.P.Xが初回実行時より2回目の実行時の方が9回多くコールされたが、HR.P.Fによっては4回少なくコールされたことを示しています。

「子」行は、HR.P.Xによって各子が初回実行時より2回目の実行時の方が5回多くコールされたことを示しています。

サンプル・レポート

HR.P.X (11行目)

サブツリー	関数	子	コール	関数名
3322196	1094099	2228097	5	HR.P.X (11 行目)

親:

サブツリー	関数	子	コール	関数名
6037490	1993169	4044321	9	HR.P.G (38 行目)
-2715294	-899070	-1816224	-4	HR.P.F (28 行目)

子:

1125489	1125489	0	5	HR.P.J (10 行目)
1102608	1102608	0	5	HR.P.I (2 行目)

関数の親子差分レポートには、関数比較レポートが付随しています。このレポートは、関数の初回実行時と2回目の実行時およびこれらの差分に関する実行プロファイルを示します。次の例は、関数HR.P.Xの関数比較レポートです。

サンプル・レポート

HR.P.X(11行目)の経過時間(マイクロ秒)(回帰合計の20.1%)

HR.P.X (11行目)	初回トレース	Ind%	2回目のトレース	Ind%	差分	差分%
関数の経過時間(マイクロ秒)	1999509	26.9%	3093608	24.9%	1094099	+54.7%
子の経過時間(マイクロ秒)	4095943	55.1%	6324040	50.9%	2228097	+54.4%
サブツリーの経過時間(マイクロ秒)	6095452	81.9%	9417648	75.7%	3322196	+54.5%
関数のコール	9	25.0%	14	28.6%	5	+55.6%
関数の平均経過時間(マイクロ秒)	222167.7		220972.0		-1195.7	-0.5%
子の平均経過時間(マイクロ秒)	455104.8		451717.1		-3387.6	-0.7%
サブツリーの平均経過時間(マイクロ秒)	677272.4		672689.1		-4583.3	-0.7%

16 品質を維持するためのPL/SQL基本ブロック・カバレッジの使用

PL/SQL基本ブロック・カバレッジのインタフェースを使用して、テストがコードをどれだけ適切にテストするかを評価することで、品質、予測可能性および整合性が保証されます。

コード・カバレッジの測定テストは、通常テスト環境で実行され、本番データベースでは実行されません。目的は、複数のPL/SQLコード・リリースのライフサイクルにおいて回帰テスト・スイートの品質を維持して改善することです。PL/SQLコード・カバレッジは、次に示す疑問に答えを出す上で役立ちます。

- テスト・スイート開発は新しいコードの開発に追いついていますか。
- テストはさらに必要ですか。

PL/SQLの基本ブロック・カバレッジ・インタフェースは、テスト実行時に実行されるPL/SQLユニットのカバレッジ・データを収集します。

トピック:

- [PL/SQL基本ブロック・カバレッジの概要](#)
- [PL/SQLコード・カバレッジ・データの収集](#)
- [PL/SQLコード・カバレッジ表の説明](#)

関連項目:

- COVERAGE PRAGMAの構文およびセマンティクスについては、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。
- DBMS_PLSQL_CODE_COVERAGEパッケージの使用方法の詳細は、[『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』](#)を参照してください。
- PLSQL_OPTIMIZE_LEVELコンパイル・パラメータの詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。

16.1 PL/SQL基本ブロック・カバレッジの概要

DBMS_PLSQL_CODE_COVERAGEパッケージは、基本ブロック・レベルでデータを収集できるようにします。PL/SQL開発者およびテスト・エンジニアはコード・カバレッジのテスト結果を標準品質保証メトリックの一部として使用します。

コード・カバレッジは、自動テストによってカバーされるコードの割合の尺度です。コード・カバレッジの高いプログラムでは、コード・カバレッジの低いプログラムよりバグ保有の可能性が下がります。最も重要なのは、テスト・スイートによって実行される基本ブロックの割合です。基本ブロックはブランチのないコードの線形セグメントです。基本ブロックには、単一のエントリ・ポイント(基本ブロック内にコードがない場合はジャンプ命令の宛先です)および単一の終了ポイント(最後の命令のみ、または例外で、実行のポイントを異なる基本ブロックに移動できます)があります。基本ブロックの境界は、コードを目で見て調査しても予想できません。コンパイラは、実行時に実行されるブロックを生成します。

ユニット・レベルのカバレッジ情報は、基本ブロック・レベルでカバレッジを収集することで正確に導出できます。ユーティリティを生成して、テスト・カバレッジの結果をレポートおよび視覚化できます。また、コードがテストでカバーされているか、部分的にカバーさ

れているか、カバーされていないかを判別するために役立ちます。

基本ブロックを練習するためにテスト・ケースを書くことが常に可能なわけではありません。これらのブロックは、COVERAGEプラグマを使用してマーク付けすることにより、カバレッジ計算から除外できます。ソースコードは、単一の基本ブロックまたは基本ブロックの範囲のいずれかで、カバレッジが不可であるとマーク付けされる可能性があります。

16.2 PL/SQLコード・カバレッジ・データの収集

この例は、DBMS_PLSQL_CODE_COVERAGEパッケージを使用したPL/SQL基本ブロック・コード・カバレッジ・データを収集して分析する、基本的なステップを示します。

デフォルトでは、すべてのユニットのカバレッジ・データ(ラップされたユニットを除く)が収集されます。

次のステップを実行して、DBMS_PLSQL_CODE_COVERAGEパッケージを使用したPL/SQL基本ブロック・コード・カバレッジ・データを収集し、分析します。

1. プロシージャCREATE_COVERAGE_TABLESを実行し、パッケージに必要な表を作成して、カバレッジ・データを格納します。このステップはセットアップの一環として一度のみ実行する必要があります。

```
EXECUTE DBMS_PLSQL_CODE_COVERAGE.CREATE_COVERAGE_TABLES;
```

2. カバレッジ実行を起動します。

```
DECLARE    testsuite_run NUMBER;
BEGIN
    testsuite_run := DBMS_PLSQL_CODE_COVERAGE.START_COVERAGE(RUN_COMMENT
=> 'Test Suite ABC');
END;
/
```

3. テストを実行します。
4. カバレッジの収集を停止して、データを収集表に書き込みます。

```
EXECUTE DBMS_PLSQL_CODE_COVERAGE.STOP_COVERAGE;
```

5. カバレッジ表を問合せ、結果を調査します。

16.3 PL/SQLコード・カバレッジ表の説明

次の表はPLSQL_CODE_COVERAGE.CREATE_COVERAGE_TABLESプロシージャによって作成され、コード・カバレッジ・データを収集します。

DBMSPCC_RUNS表には、DBMS_PLSQL_CODE_COVERAGE.START_COVERAGE関数の実行ごとに1つの行が含まれています。主キーは RUN_ID。

表16-1 DBMSPCC_RUNS表の列

列名	列データ型	説明:
RUN_ID	NUMBER(38)	DBMS_PLSQL_CODE_COVERAGE.START_COVERAGE のこの実行で自動的に生成された一意の識別子
RUN_COMMENT	VARCHAR2(4000)	実行を判断するユーザーのコメント

列名	列データ型	説明:
RUN_OWNER	VARCHAR2(128)	実行を開始したユーザー。
RUN_TIMESTAMP	DATE	実行が開始される日時のタイムスタンプ。

DBMSPCC_UNITS表には、実行時に実行されるPL/SQLユニット情報が含まれています。主キーはRUN_IDとOBJECT_IDです。OBJECT_IDとLAST_DDL_TIMEにより、静的データ・ディクショナリ・ビューALL_OBJECTSの内のオブジェクトLAST_DDL_TIMEと比較することで実行が開始された後にユニットが変更されたかどうかを判断できます。

表16-2 DBMSPCC_UNITS表の列

列名	列データ型	説明:
RUN_ID	NUMBER	DBMSPCC_RUNS 表の RUN_ID 列を参照します。
OBJECT_ID	NUMBER	ユニットの一意的識別子。
OWNER	VARCHAR2(128)	ユニットの所有者。
NAME	VARCHAR2(128)	ユニット名
TYPE	VARCHAR2(12)	ユニット・タイプ
LAST_DDL_TIME	DATE	実行時に取得された DDL 文で生じたユニットに対する変更の最新日時のタイムスタンプ。

DBMSPCC_BLOCKS表は、ユニット内のすべてのブロックを識別します。ブロックの場所は、ソースコード(LINE, COL)内の開始位置によって示されます。主キーはRUN_ID、OBJECT_IDおよびBLOCKです。1つのブロックは、暗黙的に次の文字の開始位置の直前の文字位置で終了します。同じ場所で複数のブロックを開始できます。実行の開始後にユニットが変更されていない場合は、静的データ・ディクショナリ・ビューALL_SOURCEからソース・コードの行を抽出できます。

表16-3 DBMSPCC_BLOCKS表の列

列名	列データ型	説明:
RUN_ID	NUMBER(38)	DBMSPCC_UNITS 表の RUN_ID 列を参照します。
OBJECT_ID	NUMBER(38)	DBMSPCC_UNITS 表の OBJECT_ID 列を参照します。
BLOCK	NUMBER(38)	基本ブロックの番号
LINE	NUMBER(38)	基本ブロックの先頭行の番号

列名	列データ型	説明:
COL	NUMBER(38)	基本ブロックの先頭列の番号
COVERED	NUMBER(1)	基本ブロックがカバーされている場合は 1、それ以外の場合は 0 に設定します
NOT_FEASIBLE	NUMBER(1)	基本ブロックが NOT_FEASIBLE としてマーク付けされている場合は 1、それ以外の場合は 0 に設定します

17 PL/SQL Webアプリケーションの開発

ノート:



この章で説明するテクノロジーの使用は、HTTP 通信および HTML 生成を厳密に制御する必要があるアプリケーションに適しています。他のアプリケーションの場合、アプリケーション開発を容易にするためにより多くの機能や便利なグラフィカル・インタフェースを備えた Oracle Application Express を使用することをお勧めします。

この章では、データベースをイントラネットで利用可能にする、PL/SQL Webアプリケーションの開発方法について説明します。

トピック:

- [PL/SQL Webアプリケーションの概要](#)
- [PL/SQL Webアプリケーションの実装](#)
- [mod_plsqlゲートウェイを使用したクライアント・リクエストのPL/SQL Webアプリケーションへのマップ](#)
- [埋込みPL/SQLゲートウェイの使用](#)
- [PL/SQLを使用したHTML出力の生成](#)
- [PL/SQL Webアプリケーションへのパラメータ渡し](#)
- [PL/SQLサブプログラムでのネットワーク操作の実行](#)

関連項目:

Oracle Application Expressの使用の詳細は、[『Oracle Application Expressアプリケーション・ビルダー・ユーザーズ・ガイド』](#)を参照してください

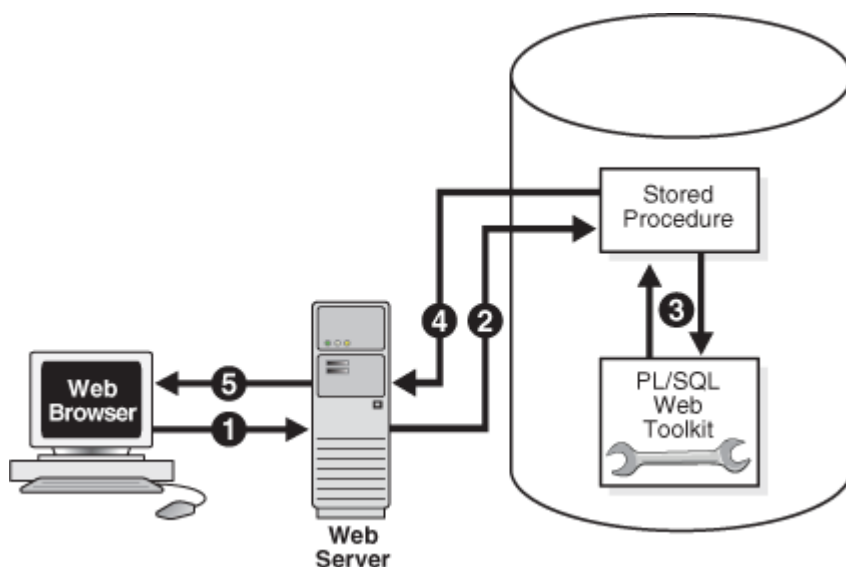
17.1 PL/SQL Webアプリケーションの概要

一般的に、PL/SQLで作成されるWebアプリケーションは、HTTPを介してWebブラウザと対話するストアド・サブプログラムの集合です。相互リンクされた一連の動的に作成されたHTMLページが、Webアプリケーションのユーザー・インタフェースになります。

PL/SQL Webアプリケーションのプログラム・フローは、CGI PERLスクリプトのプログラム・フローに似ています。通常、開発者はCGIスクリプトを使用してWebページを動的に作成しますが、多くの場合それらのスクリプトはデータベースへのアクセスに適していません。PL/SQLストアド・サブプログラムでWebコンテンツを配信することによって、強力な柔軟なデータベース処理が実現されます。たとえば、データ操作言語(DML)、動的SQL文およびカーソルを使用できます。各HTTPリクエストを処理する新しいCGIプロセスをforkするためのプロセス・オーバーヘッドを解消することもできます。

[図17-1](#)に、PL/SQL Webアプリケーションの一般的なプロセスを示します。

図17-1 PL/SQL Webアプリケーション



17.2 PL/SQL Webアプリケーションの実装

PL/SQL GatewayまたはPL/SQL Web Toolkitを使用して、完全にWebブラウザ・ベースのアプリケーションをPL/SQLに実装できます。

トピック:

- [PL/SQLゲートウェイ](#)
- [PL/SQL Webツールキット](#)

17.2.1 PL/SQLゲートウェイ

PL/SQLゲートウェイは、WebブラウザがHTTPリスナーを介してPL/SQLストアド・サブプログラムを起動するのを可能にします。ゲートウェイとは、PL/SQLユーザーがPL/SQL Webアプリケーションを開発およびデプロイするプラットフォームのことです。

17.2.1.1 mod_plsql

mod_plsqlは、PL/SQLゲートウェイの実装の一例です。モジュールとはOracle HTTP Serverのプラグインのことで、WebブラウザがPL/SQLストアド・サブプログラムを起動するのを可能にします。Oracle HTTP Serverは、Oracle Application ServerおよびOracle Databaseのコンポーネントです。

mod_plsqlプラグインを使用すると、PL/SQLストアド・サブプログラムを使用して、HTTPリクエストを処理し、レスポンスを作成できます。このコンテキストでは、HTTPリクエストはストアド・サブプログラムに渡されるパラメータ値を含むURLです。PL/SQLゲートウェイではURLを変換し、そのパラメータでストアド・サブプログラムを起動し、クライアントに出力(通常はHTML)を返します。

PL/SQLゲートウェイの埋込み形式と比較して、mod_plsqlを使用することのメリットには、次のものがあります。

- ファイアウォール環境で実行できます。この環境では、Oracle HTTP Serverがファイアウォールに対応するホスト上で実行され、データベースはファイアウォールの内側にホストされます。埋込みゲートウェイではこの構成を使用できません。
- 埋込みゲートウェイは、動的HTMLキャッシュ、システム監視、共通ログ形式でのロギングなどのmod_plsql機能をサポートしていません。

17.2.1.2 埋込みPL/SQLゲートウェイ

データベースのXML DB HTTP Listenerで稼働するPL/SQLゲートウェイの埋込みバージョンを使用できます。これにより、

mod_plsqlのコア機能がデータベースに提供されますが、Oracle HTTP Serverは必要ありません。PL/SQL Web ToolkitにあるDBMS_EPGパッケージを使用して埋込みPL/SQLゲートウェイを構成します。

mod_plsqlと比較して、埋込みゲートウェイを使用することのメリットには次のものがあります。

- Oracle HTTP ServerをインストールせずにApplication ExpressなどのPL/SQL Webアプリケーションを起動できるため、PL/SQLベースのWebアプリケーションのインストール、構成および管理を簡素化できます。
- 使用されているものと同じ構成手法を使用して、FTPリクエストおよびHTTPリクエストに応じて、Oracle XML DBからコンテンツを提供します。

17.2.2 PL/SQL Webツールキット

このPL/SQLパッケージのセットは、実行時にmod_plsqlによって起動されるストアード・サブプログラムを使用可能にする汎用インタフェースです。

ブラウザのリクエストに対するレスポンスで、PL/SQLサブプログラムはユーザー入力に従ってOracle Databaseのデータを更新または取得します。続いて、ストアード・プロシージャはブラウザに対するHTTPレスポンスを生成します。通常はダウンロード・ファイルまたはHTML表示になります。PL/SQL Web Toolkit APIを使用すると、ストアード・サブプログラムで次のような処理を実行できます。

- HTTPリクエストの情報の取得
- コンテンツタイプおよびMIMEタイプなどのHTTPヘッダーの生成
- ブラウザのCookieの設定
- HTMLページの生成

表17-1に、一般的に使用されるPL/SQL Web Toolkitパッケージを示します。

表17-1 PL/SQL Web Toolkitで一般的に使用されるパッケージ

パッケージ	内容の説明
HTF	http パッケージ内のサブプログラムのファンクション・バージョン。ファンクション・バージョンでは、出力が Web ページに直接生成されません。かわりに、出力が戻り値として、ファンクションを起動する文に渡されます。ファンクション・コールをネストする必要がある場合、これらのファンクションを使用します。
HTP	HTML タグを生成するサブプログラム。たとえば、プロシージャ http.anchor では、HTML アンカー・タグ<A>が生成されます。
OWA_CACHE	PL/SQL ゲートウェイのキャッシュ機能を使用可能にし、PL/SQL Web アプリケーションのパフォーマンスを向上させるサブプログラム。 このパッケージを使用して、PL/SQL ゲートウェイ・ファイル・システムを使用した期限切れベースおよび検証ベースのキャッシュを使用可能にできます。
OWA_COOKIE	クライアント Web ブラウザとの間で HTTP Cookie を送受信するサブプログラム。Cookie は、HTTP コール間で状態を保持するためにブラウザで使われる文字列です。状態を保持できるのは、クライアン

パッケージ	内容の説明
	ト・セッションの終わりまでか、Cookie の期日が含まれる場合はそれ以上の期間です。
OWA_CUSTOM	Cookie で使用される権限ファンクション。
OWA_IMAGE	ユーザーがイメージをクリックした座標を取得するサブプログラム。宛先リンクにより PL/SQL ゲートウェイが起動されるイメージ・マップがある場合、このパッケージを使用します。
OWA_OPT_LOCK	更新内容が失われるのを防ぐために、データベース・コミット時ロック方法を強制実行するサブプログラム。このサブプログラムを使用しないと、あるユーザーが、それまでに別のユーザーによって値が変更されている行を選択して更新しようとした場合、更新内容が失われる場合があります。
OWA_PATTERN	正規表現により、文字列一致検索および文字列操作を実行するサブプログラム。
OWA_SEC	PL/SQL ゲートウェイでリクエストの認証に使用されるサブプログラム。
OWA_TEXT	OWA_PATTERN パッケージで文字列の操作に使用されるサブプログラム。直接使用することもできます。
OWA_UTIL	次のタイプのユーティリティ・サブプログラム - 動的に生成された SQL コードによりページを生成する動的 SQL ユーティリティ。 - CGI 環境変数の値を取得し、URL リダイレクトを実行する HTML ユーティリティ。 - 正しい日付処理を行うための日付ユーティリティ。日付の値は HTML では単純な文字列ですが、Oracle Database データ型として適切に処理される必要があります。
WPG_DOCLOAD	DAD 構成を使用して定義したドキュメント・リポジトリからドキュメントをダウンロードするサブプログラム。

関連項目:

PL/SQL Web Toolkitパッケージの構文、説明および例は、[『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。

17.3 mod_plsqlゲートウェイを使用したクライアント・リクエストのPL/SQL Webアプリケーションへのマップ

『Oracle HTTP Server mod_plsqlユーザーズ・ガイド』で説明しているように、mod_plsqlは、HTTP上でWebクライアント・リクエストをPL/SQLストアド・サブプログラムにマップします。詳細は、このマニュアルを参照してください。

関連項目:

- `mod_plsql`の構成および使用方法を学習するには、[『Oracle HTTP Server mod_plsqlユーザズ・ガイド』](#)を参照してください。
- `mod_plsql`モジュールの詳細は、[『Oracle Fusion Middleware Oracle HTTP Server管理者ガイド』](#)を参照してください。

17.4 埋込みPL/SQLゲートウェイの使用

埋込みゲートウェイ機能は、`mod_plsql`ゲートウェイに類似しています。

トピック:

- [埋込みPL/SQLゲートウェイによるクライアント・リクエストの処理方法](#)
- [埋込みPL/SQLゲートウェイのインストール](#)
- [埋込みPL/SQLゲートウェイの構成](#)
- [埋込みPL/SQLゲートウェイを介したPL/SQLストアド・サブプログラムの起動](#)
- [埋込みPL/SQLゲートウェイによるアプリケーション・アクセスの保護](#)
- [埋込みPL/SQLゲートウェイの制限事項](#)
- [埋込みPL/SQLゲートウェイの使用: 使用例](#)

関連項目:

[Oracle HTTP Server mod_plsqlユーザズ・ガイド](#)

17.4.1 埋込みPL/SQLゲートウェイによるクライアント・リクエストの処理方法

[図17-2](#)に、埋込みゲートウェイによるクライアントHTTPリクエストの処理プロセスを示します。

図17-2 埋込みPL/SQLゲートウェイによるクライアント・リクエストの処理

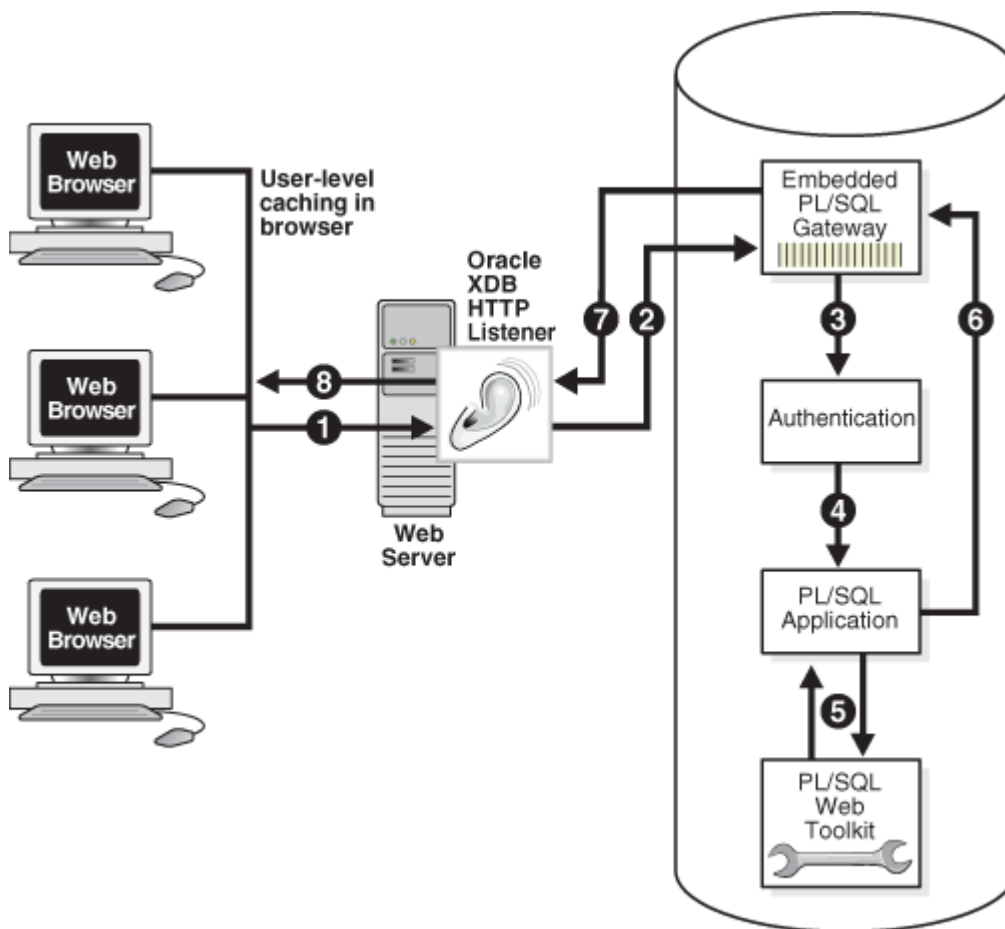


図17-2の各ステップの内容は次のとおりです。

1. Oracle XML DB HTTP Listenerは、クライアント・ブラウザからのリクエストを受け取り、PL/SQLサブプログラムを起動するようリクエストします。サブプログラムは、PL/SQLに直接記述されたもの、またはPL/SQL Server Pagesがデータベースにアップロードされ、コンパイルされたときに間接的に生成されたものを使用できます。
2. XML DB HTTP Listenerは、その仮想パス・マッピング構成に指定されているとおり、埋込みPL/SQLゲートウェイにリクエストをルーティングします。
3. 埋込みゲートウェイではHTTPリクエスト情報およびゲートウェイ構成を使用して、認証にどのデータベース・アカウントを使用するかを決定します。
4. 埋込みゲートウェイはコール・パラメータを準備して、アプリケーション内でPL/SQLサブプログラムを起動します。
5. PL/SQLサブプログラムは、リレーショナル・データおよびデータベースからアクセスされたPL/SQL Web ToolkitからHTMLページを生成します。
6. アプリケーションから埋込みゲートウェイにページを送信します。
7. 埋込みゲートウェイからXML DB HTTP Listenerにページを送信します。
8. XML DB HTTP Listenerからクライアント・ブラウザにページを送信します。

mod_plsqlとは異なり、埋込みゲートウェイはOracle XML DB HTTP Listenerを使用してHTTPリクエストを処理します。このリスナーは、Oracle Net Listenerと同じサーバー側プロセスであり、Oracle Net Services、HTTPおよびFTPをサポートします。

XML DBインタフェースを介して一般HTTPリスナー設定を構成します(手順については、『[Oracle XML DB開発者ガイド](#)』を参照してください)。Oracle Enterprise Manager Cloud Control (Cloud Control)を使用するか、xdbconfig.xmlファイルを編集することでHTTPリスナーを構成します。DADの属性の作成または設定など、すべての埋込みPL/SQLゲートウェイ

イ構成で、DBMS_EPGパッケージを使用します。

17.4.2 埋込みPL/SQLゲートウェイのインストール

埋込みゲートウェイでは次のコンポーネントが必要です。

- XML DB HTTP Listener
- PL/SQL Webツールキット

埋込みPL/SQLゲートウェイは、Oracle XML DBの一部としてインストールします。インストール時、またはDatabase Configuration Assistant(DBCA)によって作成された事前構成済データベースを使用している場合、Oracle XML DBがインストールされ、構成されます。

PL/SQL Web Toolkitはデータベースの標準インストールの一部であるため、追加でインストールする必要はありません。

関連項目:

既存のデータベースにOracle XML DBを手動で追加する方法の詳細は、[『Oracle XML DB開発者ガイド』](#)を参照してください。

17.4.3 埋込みPL/SQLゲートウェイの構成

Oracle HTTP Serverの構成ファイルを編集して、mod_plsqlを構成します。埋込みゲートウェイはOracle XML DB HTTP Listenerの一部としてインストールするため、Oracle XML DBサーブレット管理インタフェースを介してサーブレットとして管理します。

埋込みゲートウェイへの構成インタフェースは、PL/SQLパッケージDBMS_EPGです。このパッケージは、XML DBで使用される、基礎となるxdbconfig.xml構成ファイルを変更します。ほとんどのユーザーにとって、埋込みゲートウェイ構成パラメータはデフォルト値で問題ありません。

トピック:

- [埋込みPL/SQLゲートウェイの構成: 概要](#)
- [埋込みPL/SQLゲートウェイのユーザー認証に関する構成](#)

17.4.3.1 埋込みPL/SQLゲートウェイの構成: 概要

mod_plsqlと同様、PL/SQLストアド・サブプログラムの各リクエストは、データベース・アクセス記述子(DAD)に関連付けられています。DADとは、データベース・アクセスに使用する構成値の集合です。DADでは次のような情報を指定します。

- 認証に使用するデータベース・アカウント
- ドキュメントのアップロードおよびダウンロードに使用するサブプログラム

埋込みPL/SQLゲートウェイでは、DADは、XML DB HTTP LISTENER構成のサーブレットとして表されます。各DAD属性は、構成ファイルxdbconfig.xmlのXML要素にマップされます。DAD属性の値は、要素の内容に一致します。たとえば、database-username DAD属性は、<database-username> XML要素に一致し、DAD属性の値がHRである場合、<database-username>HR<database-username>に一致します。DAD属性名は、大/小文字を区別します。

次の埋込みPL/SQLゲートウェイ構成を実行するには、DBMS_EPGパッケージを使用します。

1. DBMS_EPG.CREATE_DADプロシージャを使用してDADを作成します。

2. DBMS_EPG.SET_DAD_ATTRIBUTEプロシージャを使用してDAD属性を設定します。

すべてのDAD属性は、オプションです。属性を指定しない場合は、初期値が使用されます。

[表17-2](#)に、埋込みPL/SQLゲートウェイ属性および対応するmod_plsql DADパラメータを示します。「有効な値」の列挙値は大/小文字を区別します。

表17-2 mod_plsql DAD属性と埋込みPL/SQLゲートウェイDAD属性間のマッピング

mod_plsql DAD属性	埋込みPL/SQLゲートウェイDAD属性	重複	有効な値
PlsqlAfterProcedure	after-procedure	なし	文字列
PlsqlAlwaysDescribeProcedure	always-describe-procedure	なし	On、Off の列挙
PlsqlAuthenticationMode	authentication-mode	なし	Basic、Single GlobalOwa、 PerPackageC
PlsqlBeforeProcedure	before-procedure	なし	文字列
PlsqlBindBucketLengths	bind-bucket-lengths	あり	符号なし整数
PlsqlBindBucketWidths	bind-bucket-widths	あり	符号なし整数
PlsqlCGIEnvironmentList	cgi-environment-list	あり	文字列
PlsqlCompatibilityMode	compatibility-mode	なし	符号なし整数
PlsqlDatabaseUsername	database-username	なし	文字列
PlsqlDefaultPage	default-page	なし	文字列
PlsqlDocumentPath	document-path	なし	文字列
PlsqlDocumentProcedure	document-procedure	なし	文字列
PlsqlDocumentTablename	document-table-name	なし	文字列
PlsqlErrorStyle	error-style	なし	ApacheStyle DebugStyle
PlsqlExclusionList	exclusion-list	あり	文字列

mod_plsql DAD属性	埋込みPL/SQLゲートウェイDAD属性	重複	有効な値
PlsqlFetchBufferSize	fetch-buffer-size	なし	符号なし整数
PlsqlInfoLogging	info-logging	なし	Enumeration
PlsqlInputFilterEnable	input-filter-enable	なし	文字列
PlsqlMaxRequestsPerSession	max-requests-per-session	なし	符号なし整数
PlsqlNLSLanguage	nls-language	なし	文字列
PlsqlOWADebugEnable	owa-debug-enable	なし	On、Off の列挙
PlsqlPathAlias	path-alias	なし	文字列
PlsqlPathAliasProcedure	path-alias-procedure	なし	文字列
PlsqlRequestValidationFunction	request-validation-function	なし	文字列
PlsqlSessionCookieName	session-cookie-name	なし	文字列
PlsqlSessionStateManagement	session-state-management	なし	StatelessWithCookie、StatelessWithoutCookie、StatelessWithCookie の列挙
PlsqlTransferMode	transfer-mode	なし	Char、Raw の列挙
PlsqlUploadAsLongRaw	upload-as-long-raw	なし	文字列

埋込みゲートウェイのほとんどのユーザーにとって、DAD属性はデフォルト値で問題ありません。mod_plsqlユーザーには、次の属性は不要です。

- PlsqlDatabasePassword
- PlsqlDatabaseConnectString(埋込みゲートウェイで外部データベースへのログオンがサポートされていないため)

DAD属性と同様、グローバル構成パラメータはオプションです。[表17-3](#)に、DBMS_EPGグローバル属性および対応するmod_plsqlグローバル・パラメータを示します。

表17-3 mod_plsqlグローバル属性と埋込みPL/SQLゲートウェイ・グローバル属性間のマッピング

mod_plsql DAD属性	埋込みPL/SQLゲートウェイ DAD属性	重複	有効な値
PlsqlLogLevel	log-level	なし	符号なし整数
PlsqlMaxParameters	max-parameters	なし	符号なし整数

関連項目:

- mod_plsql DAD属性の詳細は、[『Oracle Fusion Middleware Oracle HTTP Server管理者ガイド』](#)を参照してください。デフォルト値と使用上のノートについては、このマニュアルを参照してください。
- DBMS_EPG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。
- xdbconfig.xmlファイルのアカウントについては、[『Oracle XML DB開発者ガイド』](#)を参照してください。

17.4.3.2 埋込みPL/SQLゲートウェイのユーザー認証に関する構成

XML DB認証スキームを使用するため、埋込みゲートウェイではデータベースの認証方式がmod_plsqlとは異なります。特に、DADにはデータベースのパスワードは格納されません。

ノート:



PL/SQL Web アプリケーションをインターネット上で提供しても、データベースをファイアウォールの内側に保持する場合は、アプリケーションの実行に埋込み PL/SQL ゲートウェイを使用せずに、mod_plsql を使用してください。

データベース認証を構成するには、DBMS_EPGパッケージを使用します。

トピック:

- [DBMS_EPGによる静的認証の構成](#)
- [DBMS_EPGによる動的認証の構成](#)
- [DBMS_EPGによる匿名認証の構成](#)
- [DADの認証モードの決定](#)
- [例: DADの作成および構成](#)
- [例: DADの認証モードの決定](#)
- [例: すべてのDADの認証モードの決定](#)
- [例: 無効なDAD認可の表示](#)
- [埋込みPL/SQLゲートウェイ構成の調査](#)

17.4.3.2.1 DBMS_EPGによる静的認証の構成

静的認証はデータベースのユーザー名およびパスワードをDADに格納するmod_plsqlユーザー用であるため、ブラウザ・ユーザーがデータベース認証情報を入力する必要はありません。

静的認証を設定するステップは次のとおりです。

1. XML DB管理者、つまりXDBADMINのロールが割り当てられたユーザーとしてデータベースにログオンします。
2. DADを作成します。たとえば、次のプロシージャでは、DADによって起動されるHR_DADを作成し、仮想パスを/hrweb/にマップします。

```
EXEC DBMS_EPG.CREATE_DAD('HR_DAD', '/hrweb/*');
```

3. このステップでは、ALTER ANY USERシステム権限が必要です。DADにより権限が使用される必要があるデータベース・アカウントに、DAD属性database-usernameを設定します。たとえば、次のプロシージャでは、DAD名HR_DADがHRアカウントの権限を保持すると指定します。

```
EXEC DBMS_EPG.SET_DAD_ATTRIBUTE('HR_DAD', 'database-username', 'HR');
```

DAD属性database-usernameは、大/小文字を区別します。

4. 前のステップでユーザーが指定したデータベースの権限をDADに割り当てます。この認可によって、エンド・ユーザーはプロシージャを起動し、認可されたアカウントの権限のある埋込みPL/SQLゲートウェイを介してドキュメント表にアクセスできます。たとえば：

```
EXEC DBMS_EPG.AUTHORIZE_DAD('HR_DAD', 'HR');
```

あるいは、XDBADMIN権限のあるユーザーとしてログオフし、その権限がDADで使用される必要があるデータベース・ユーザーとしてログオンしてから、これらの権限をDADに割り当てる次のコマンドを使用します。

```
EXEC DBMS_EPG.AUTHORIZE_DAD('HR_DAD');
```

ノート：



複数のユーザーが同一のDADを認可できます。DADのdatabase-username属性設定によって、どのユーザーの権限を使用するかが決定されます。

mod_plsqlとは異なり、埋込みゲートウェイは特殊ユーザーANONYMOUSとしてデータベースに接続しますが、データベース・オブジェクトにはDADに割り当てられたユーザー権限でアクセスします。ブラウザ・ユーザーがHTTP Authorizationヘッダーで明示的に接続しようとする、データベースはアクセスを拒否します。

ノート：



アカウントANONYMOUSは、XML DBのインストール後にロックされます。埋込みPL/SQLゲートウェイで静的認証を使用する場合は、最初にこのアカウントのロックを解除します。

17.4.3.2.2 DBMS_EPGによる動的認証の構成

動的認証は、DADにデータベースのユーザー名およびパスワードを格納しないmod_plsqlユーザーのためのものです。

動的認証では、データベース・ユーザーは、データベース・オブジェクトへのアクセス権限を使用するため、埋込みゲートウェイを認

可する必要がありません。かわりに、ブラウザ・ユーザーがHTTP Basic認証スキームを介してデータベース認証情報を指定する必要があります。

埋込みゲートウェイのアクションは、DADに対してdatabase-username属性が設定されているかどうかによって異なります。この属性が設定されていない場合、埋込みゲートウェイはブラウザ・クライアントによって指定されたユーザーとして、データベースに接続します。属性が設定されている場合、データベースでは、database-username属性で指定されたユーザーにアクセスを制限します。

動的認証を設定するステップは次のとおりです。

1. XML DB管理者、つまりXDBADMINのロールのあるユーザーとしてデータベースにログオンします。
2. DADを作成します。たとえば、次のプロシージャでは、DADによって起動されるDYNAMIC_DADを作成し、仮想パスを/hrweb/にマップします。

```
EXEC DBMS_EPG.CREATE_DAD('DYNAMIC_DAD', '/hrweb/*');
```

3. オプションで、DADにより権限が使用される必要があるデータベース・アカウントに、DAD属性database-usernameを設定します。ブラウザ・ユーザーはDADアクセスの際、このアカウントのユーザー名およびパスワードを入力するよう要求されます。たとえば、次のプロシージャでは、DAD名DYNAMIC_DADがHRアカウントの権限を保持すると指定します。

```
EXEC DBMS_EPG.SET_DAD_ATTRIBUTE('DYNAMIC_DAD', 'database-username', 'HR');
```

属性database-usernameは、大/小文字を区別します。

警告:



HTTP Basic 認証スキームを介して送信されるパスワードは、暗号化されていません。ブラウザ・クライアントによって送信されるパスワードを保護するため、HTTPS プロトコルを使用するよう埋込みゲートウェイを構成します。

17.4.3.2.3 DBMS_EPGによる匿名認証の構成

匿名認証はmod_plsqlユーザー用であり、このユーザーはデータベース・ログオン用の特殊なDADデータベースを作成しますが、アプリケーション・プロシージャおよびドキュメント表を別のスキーマに格納し、プロシージャおよびドキュメント表へのアクセス権をPUBLICに付与します。

匿名認証を設定するステップは次のとおりです。

1. XML DB管理者、つまりXDBADMINのロールが割り当てられたユーザーとしてデータベースにログオンします。
2. DADを作成します。たとえば、次のプロシージャでは、DADによって起動されるHR_DADを作成し、仮想パスを/hrweb/にマップします。

```
EXEC DBMS_EPG.CREATE_DAD('HR_DAD', '/hrweb/*');
```

3. DAD属性database-usernameをANONYMOUSに設定します。たとえば:

```
EXEC DBMS_EPG.SET_DAD_ATTRIBUTE('HR_DAD', 'database-username', 'ANONYMOUS');
```

database-usernameおよびANONYMOUSはともに、大/小文字を区別します。

データベース・オブジェクトへのアクセス用ANONYMOUS権限の使用のために埋込みゲートウェイを認可する必要はありません。ANONYMOUSにはシステム権限がなく、データベース・オブジェクトを所有していません。

17.4.3.2.4 DADの認証モードの決定

DADの名前がわかっている場合は、このDADの認証モードは次によって決まります。

- DADが存在するかどうか。
- database-username属性がDAD用に設定されているかどうか。
- DADにdatabase-usernameユーザーの権限の使用が認可されているかどうか。
- database-username属性がDADの使用を認可されているユーザーの名前であるかどうか。

[表17-4](#)に、これらの質問の回答に応じた認証モードの決定方法を示します。

表17-4 DADの認証の可能性

DADの有無	database-username設定の有無		モード
	無	ユーザー認可の有無	
あり	あり	あり	静的
あり	あり	なし	動的(制限付き)
あり	なし	どちらでもよい	動的
あり	あり(ANONYMOUS に対して)	どちらでもよい	匿名
なし			N/A

たとえば、DAD名MY_DADを作成するとします。MY_DADのdatabase-username属性がHRに設定されていても、HRユーザーがMY_DADを認可しない場合、MY_DADの認証モードは制限付きで動的になります。ブラウザ・ユーザーがMY_DADを介してPL/SQLサブプログラムを実行しようとすると、HRデータベースのユーザー名およびパスワードを入力するよう要求されます。

DBA_EPG_DAD_AUTHORIZATIONビューでは、どのユーザーがDADの使用を認可しているかを示します。DAD_NAME列にはDADの名前が表示され、USERNAME列にはDADに権限が割り当てられたユーザーが表示されます。権限が認められたDADは存在しないことがあります。

関連項目:

DBA_EPG_DAD_AUTHORIZATIONビューの詳細は、[『Oracle Databaseリファレンス』](#)を参照してください。

17.4.3.2.5 例: DADの作成および構成

[例17-1](#)では次の処理が行われます。

- データベース・ユーザーHR用に静的認証のDADを作成し、これを認可するHRアカウントの権限を割り当てます。
- ユーザーを限定しない、動的認証のDADを作成します。
- HRアカウントに限定される、動的認証のDADを作成します。

DADの作成および認証は独立しているため、次のことができます。

- 存在していないDADの認証(後で作成が可能)
- 自分がユーザーでないDADの認証(ただし、この認証はDAD database-user属性が自分のユーザー名に変更されるまで反映されません)

[例17-2](#)では、データベース・ユーザーHR用に静的認証のDADを作成し、これにHRアカウントの権限を割り当てます。その後、次の処理が行われます。

- データベース・ユーザーHRがそのDADではなく、存在しないDADを認証します。
ユーザーが誤ってこれを実行した場合でも、存在しないDADは後で作成できるため、エラーは発生しません。
- データベース・ユーザーOEが、自分のdatabase-user属性がHRに設定されているDADを認証します。
エラーは発生しませんが、DADのdatabase-user属性がOEに変更されるまで認証は有効になりません。

例17-1 DADの作成および構成

```
-----
--- DAD with static authentication
-----
CONNECT SYSTEM AS SYSDBA
PASSWORD: password
EXEC DBMS_EPG.CREATE_DAD('Static_Auth_DAD', '/static/*');
EXEC DBMS_EPG.SET_DAD_ATTRIBUTE('Static_Auth_DAD', 'database-username', 'HR');
GRANT EXECUTE ON DBMS_EPG TO HR;
-- Authorization
CONNECT HR
PASSWORD: password
EXEC DBMS_EPG.AUTHORIZE_DAD('Static_Auth_DAD');
-----
-- DAD with dynamic authentication
-----
CONNECT SYSTEM AS SYSDBA
PASSWORD: password
EXEC DBMS_EPG.CREATE_DAD('Dynamic_Auth_DAD', '/dynamic/*');
-----
-- DAD with dynamic authentication restricted
-----
EXEC DBMS_EPG.CREATE_DAD('Dynamic_Auth_DAD_Restricted', '/dynamic/*');
EXEC DBMS_EPG.SET_DAD_ATTRIBUTE
('Dynamic_Auth_DAD_Restricted', 'database-username', 'HR');
```

例17-2 後で作成または変更されるDADの認証

```
REM Create DAD with static authentication for database user HR
CONNECT SYSTEM AS SYSDBA
PASSWORD: password
EXEC DBMS_EPG.CREATE_DAD('Static_Auth_DAD', '/static/*');
EXEC DBMS_EPG.SET_DAD_ATTRIBUTE('Static_Auth_DAD', 'database-username', 'HR');
GRANT EXECUTE ON DBMS_EPG TO HR;
REM Database user HR authorizes DAD that does not exist
CONNECT HR
PASSWORD: password
EXEC DBMS_EPG.AUTHORIZE_DAD('Static_Auth_DAD_Typo');
REM Database user OE authorizes DAD with database-username 'HR'
CONNECT OE
PASSWORD: password
EXEC DBMS_EPG.AUTHORIZE_DAD('Static_Auth_DAD');
```

17.4.3.2.6 例: DADの認証モードの決定

[例17-3](#)では、指定されたDADの名称を許容し、その認証モードをレポートするPL/SQLプロシージャ

show_dad_auth_statusを作成します。指定したDADが存在しない場合、プロシーダはエラー状態で終了します。

[例17-1](#)のスクリプトを実行して、各種DADを作成および構成するとします。出力は次のとおりです。

```
SET SERVEROUTPUT ON;
BEGIN
  show_dad_auth_status('Static_Auth_DAD');
END;
/
'Static_Auth_DAD' is set up for static authentication for user 'HR'.
```

例17-3 DADの認証モードの決定

```
CREATE OR REPLACE PROCEDURE show_dad_auth_status (p_dadname VARCHAR2) IS
  v_daduser VARCHAR2(128);
  v_cnt      PLS_INTEGER;
BEGIN
  -- Determine DAD user
  v_daduser := DBMS_EPG.GET_DAD_ATTRIBUTE(p_dadname, 'database-username');
  -- Determine whether DAD authorization exists for DAD user
  SELECT COUNT(*)
    INTO v_cnt
    FROM DBA_EPG_DAD_AUTHORIZATION da
    WHERE da.DAD_NAME = p_dadname
    AND da.USERNAME = v_daduser;
  -- If DAD authorization exists for DAD user, authentication mode is static
  IF (v_cnt > 0) THEN
    DBMS_OUTPUT.PUT_LINE (
      ''' || p_dadname ||
      ''' is set up for static authentication for user ''' ||
      v_daduser || '''.'');
    RETURN;
  END IF;
  -- If no DAD authorization exists for DAD user, authentication mode is dynamic
  -- Determine whether dynamic authentication is restricted to particular user
  IF (v_daduser IS NOT NULL) THEN
    DBMS_OUTPUT.PUT_LINE (
      ''' || p_dadname ||
      ''' is set up for dynamic authentication for user ''' ||
      v_daduser || ''' only.'');
  ELSE
    DBMS_OUTPUT.PUT_LINE (
      ''' || p_dadname ||
      ''' is set up for dynamic authentication for any user.'');
  END IF;
END;
/
```

17.4.3.2.7 例: すべてのDADの認証モードの決定

[例17-4](#)の無名ブロックでは、登録したすべてのDADの認証モードがレポートされます。これによって、[例17-3](#)のshow_dad_auth_statusプロシーダが起動されます。

[例17-1](#)のスクリプトを実行して各種DADを作成および構成した場合、[例17-4](#)の出力は次のとおりです。

```
----- Authorization Status for All DADs -----
'Static_Auth_DAD' is set up for static auth for user 'HR'.
'Dynamic_Auth_DAD' is set up for dynamic auth for any user.
'Dynamic_Auth_DAD_Restricted' is set up for dynamic auth for user 'HR' only.
```

例17-4 全DADの認証モードの表示

```
DECLARE
  v_dad_names DBMS_EPG.VARCHAR2_TABLE;
BEGIN
```

```

DBMS_OUTPUT.PUT_LINE
('----- Authorization Status for All DADs -----');
DBMS_EPG.GET_DAD_LIST(v_dad_names);
FOR i IN 1..v_dad_names.count LOOP
    show_dad_auth_status(v_dad_names(i));
END LOOP;
END;
/

```

17.4.3.2.8 例: 無効なDAD認可の表示

[例17-5](#)の無名ブロックでは、無効なDAD認証がレポートされます。DAD認証は、次の場合は無効です。

- DADを認可するユーザーが、DADのdatabase-username属性で指定されたユーザーではない。
- DADを認可するユーザーが存在しない。

[例17-2](#)のスクリプトを実行して各種DADを作成および構成した場合、[例17-5](#)の出力は次のとおりです(ページ内に収まるように再フォーマット済)。

```

----- DAD Authorizations Not in Effect -----
DAD authorization of 'Static_Auth_DAD' by user 'OE' is not in effect
because DAD user is 'HR'.
DAD authorization of 'Static_Auth_DAD_Typo' by user 'HR' is not in effect
because DAD does not exist.

```

例17-5 無効なDAD認可の表示

```

DECLARE
    v_dad_names DBMS_EPG.VARCHAR2_TABLE;
    v_dad_user  VARCHAR2(128);
    v_dad_found BOOLEAN;
BEGIN
    DBMS_OUTPUT.PUT_LINE
    ('----- DAD Authorizations Not in Effect -----');
    DBMS_EPG.GET_DAD_LIST(v_dad_names);
    FOR r IN (SELECT * FROM DBA_EPG_DAD_AUTHORIZATION) LOOP -- Outer loop
        v_dad_found := FALSE;
        FOR i IN 1..v_dad_names.count LOOP -- Inner loop
            IF (r.DAD_NAME = v_dad_names(i)) THEN
                v_dad_user :=
                    DBMS_EPG.GET_DAD_ATTRIBUTE(r.DAD_NAME, 'database-username');
                -- Is database-username the user for whom DAD is authorized?
                IF (r.USERNAME <> v_dad_user) THEN
                    DBMS_OUTPUT.PUT_LINE (
                        'DAD authorization of ''' || r.dad_name ||
                        ''' by user ''' || r.username || ''' ||
                        ' is not in effect because DAD user is ' ||
                        ''' || v_dad_user || '''.');
                END IF;
                v_dad_found := TRUE;
                EXIT; -- Inner loop
            END IF;
        END LOOP; -- Inner loop
        -- Does DAD exist?
        IF (NOT v_dad_found) THEN
            DBMS_OUTPUT.PUT_LINE (
                'DAD authorization of ''' || r.dad_name ||
                ''' by user ''' || r.username ||
                ''' is not in effect because the DAD does not exist. ');
        END IF;
    END LOOP; -- Outer loop
END;
/

```

17.4.3.2.9 埋込みPL/SQLゲートウェイ構成の調査

システム権限のあるユーザーとしてデータベースに接続している場合、次のスクリプトにより、埋込みPL/SQLゲートウェイ構成を調査します。

```
$ORACLE_HOME/rdbms/admin/epgstat.sql
```

例17-6に、ANONYMOUSアカウントがロックされている場合の例17-1のepgstat.sqlスクリプトの出力を示します。

例17-6 例17-1のepgstat.sqlスクリプトの出力

スクリプトを実行するコマンドは次のとおりです。

```
@$ORACLE_HOME/rdbms/admin/epgstat.sql
```

結果:

```
+-----+
| XDB protocol ports: |
| XDB is listening for the protocol |
| when the protocol port is nonzero. |
+-----+
HTTP Port FTP Port
-----
0 0
1 row selected.
+-----+
| DAD virtual-path mappings |
+-----+
Virtual Path DAD Name
-----
/dynamic/* Dynamic_Auth_DAD_Restricted
/static/* Static_Auth_DAD
2 rows selected.
+-----+
| DAD attributes |
+-----+
DAD Name DAD Param DAD Value
-----
Dynamic_Auth_DAD_Restricted database-username HR
Static_Auth_DAD database-username HR
2 rows selected.
+-----+
| DAD authorization: |
| To use static authentication of a user in a DAD, |
| the DAD must be authorized for the user. |
+-----+
DAD Name User Name
-----
Static_Auth_DAD HR
OE
Static_Auth_DAD_Typo HR
3 rows selected.
+-----+
| DAD authentication schemes |
+-----+
DAD Name User Name Auth Scheme
-----
Dynamic_Auth_DAD Dynamic
Dynamic_Auth_DAD_Restricted Dynamic Restricted
Static_Auth_DAD HR Static
```

```

3 rows selected.
+-----+
| ANONYMOUS user status:                |
| To use static or anonymous authentication in any DAD, |
| the ANONYMOUS account must be unlocked.             |
+-----+
Database User      Status
-----
ANONYMOUS          EXPIRED & LOCKED
1 row selected.
+-----+
| ANONYMOUS access to XDB repository:      |
| To allow public access to XDB repository without authentication, |
| ANONYMOUS access to the repository must be allowed.             |
+-----+
Allow repository anonymous access?
-----
false
1 row selected.

```

17.4.4 埋込みPL/SQLゲートウェイを介したPL/SQLストアド・サブプログラムの起動

埋込みPL/SQLゲートウェイを介してPL/SQLサブプログラムを起動する基本ステップは、mod_plsqlゲートウェイに対するものと同じです。詳細は、『[Oracle HTTP Server mod_plsqlユーザズ・ガイド](#)』を参照してください。mod_plsql命令は埋込みゲートウェイで使用するため、若干の調整が必要です。たとえば、次の形式でURLを入力し、ブラウザで埋込みゲートウェイを起動します。

```
protocol://hostname[:port]/virt-path/[!][schema.][package.]proc_name[?query_str]
```

プレースホルダvirt-pathは、DBMS_EPG.CREATE_DADで構成する仮想パスを表します。mod_plsqlのドキュメントでは、virt-pathのかわりにDAD_locationを使用します。

関連項目:

- 次のトピックについては、『[Oracle HTTP Server mod_plsqlユーザズ・ガイド](#)』を参照してください。
 - トランザクション・モード
 - サポートされるデータ型
 - パラメータ渡しスキーム
 - ファイルのアップロード/ダウンロードのサポート
 - パスの別名
 - Common Gateway Interface(CGI)環境変数

17.4.5 埋込みPL/SQLゲートウェイによるアプリケーション・アクセスの保護

埋込みゲートウェイは、mod_plsqlと同じ保護メカニズムを使用します。

関連項目:

[Oracle HTTP Server mod_plsqlユーザズ・ガイド](#)

17.4.6 埋込みPL/SQLゲートウェイの制限事項

mod_plsqlの制限は、埋込みゲートウェイにも同様に適用されます。また、ゲートウェイの埋込みバージョンでは、次の機能はサポートされません。

- 動的HTMLキャッシング
- システムの監視
- Basic以外の認証モード

関連項目:

- 制限の詳細は、『[Oracle HTTP Server mod_plsqlユーザーズ・ガイド](#)』を参照してください。
- 認証モードの詳細は、『[Oracle HTTP Server mod_plsqlユーザーズ・ガイド](#)』を参照してください。

17.4.7 埋込みPL/SQLゲートウェイの使用: 使用例

この項では、hr.employees表に対して問合せを行い、PL/SQLゲートウェイを介してWebブラウザにHTML出力を供給する簡易アプリケーションを作成する方法について説明します。XML DBおよびサンプル・スキーマの両方がインストールされているものとします。

プログラムを作成して実行するステップは次のとおりです。

1. ALTER USER権限のあるユーザーとしてデータベースにログオンし、データベース・アカウントANONYMOUSのロックが解除されていることを確認します。ANONYMOUSアカウントはデフォルトでロックされており、静的認証を必要とします。アカウントがロックされている場合は、次のSQL文を使用してそのロックを解除します。

```
ALTER USER anonymous ACCOUNT UNLOCK;
```

2. XML DB管理者、つまりXDBADMINのロールのあるユーザーとしてデータベースにログオンします。

データ・ディクショナリを問い合わせ、どのユーザーおよびロールにXDBADMINのロールが付与されているかを確認します。

```
SELECT *  
FROM DBA_ROLE_PRIVS  
WHERE GRANTED_ROLE = 'XDBADMIN';
```

3. DADを作成します。たとえば、次のプロシージャでは、DADによって起動されるHR_DADを作成し、仮想パスを/plsql/にマップします。

```
EXEC DBMS_EPG.CREATE_DAD('HR_DAD', '/plsql/*');
```

4. DADにより権限が使用される必要があるデータベース・ユーザーに、DAD属性database-usernameを設定します。たとえば、次のプロシージャでは、DAD HR_DADがユーザーHR権限によってデータベース・オブジェクトにアクセスすることを指定します。

```
EXEC DBMS_EPG.SET_DAD_ATTRIBUTE('HR_DAD', 'database-username', 'HR');
```

属性database-usernameは、大/小文字を区別します。

5. DADにより権限が使用される必要のあるデータベース・ユーザーにEXECUTE権限を付与します(これによって、ユーザーはDADを認証できます)。たとえば:

```
GRANT EXECUTE ON DBMS_EPG TO HR;
```

6. XML DB管理者としてログオフし、DADにより権限が使用される必要があるデータベース・ユーザーとして、データベースにログオンします(たとえば、HR)。

7. 埋込みPL/SQLゲートウェイを認可してプロシージャを起動し、DADを介してドキュメント表にアクセスします。たとえば:

```
EXEC DBMS_EPG.AUTHORIZE_DAD('HR_DAD');
```

8. サンプルのPL/SQLストアド・プロシージャによって起動されるprint_employeesを作成します。次のプログラムでは、hr.employeesの間合せの結果セットを含むHTMLページを作成します。

```
CREATE OR REPLACE PROCEDURE print_employees IS
  CURSOR emp_cursor IS
    SELECT last_name, first_name
    FROM hr.employees
    ORDER BY last_name;
BEGIN
  HTP.PRINT('<html>');
  HTP.PRINT('<head>');
  HTP.PRINT('<meta http-equiv="Content-Type" content="text/html">');
  HTP.PRINT('<title>List of Employees</title>');
  HTP.PRINT('</head>');
  HTP.PRINT('<body TEXT="#000000" BGCOLOR="#FFFFFF">');
  HTP.PRINT('<h1>List of Employees</h1>');
  HTP.PRINT('<table width="40%" border="1">');
  HTP.PRINT('<tr>');
  HTP.PRINT('<th align="left">Last Name</th>');
  HTP.PRINT('<th align="left">First Name</th>');
  HTP.PRINT('</tr>');
  FOR emp_record IN emp_cursor LOOP
    HTP.PRINT('<tr>');
    HTP.PRINT('<td>' || emp_record.last_name || '</td>');
    HTP.PRINT('<td>' || emp_record.first_name || '</td>');
  END LOOP;
  HTP.PRINT('</table>');
  HTP.PRINT('</body>');
  HTP.PRINT('</html>');
END;
/
```

9. Oracle NetリスナーがHTTPリクエストを受け入れられることを確認します。システム・プロンプトで次のコマンドを実行して、LinuxおよびUNIXでのリスナーのステータスを確認できます。

```
lsnrctl status | grep HTTP
```

出力は次のとおりです(ページ・サイズによる制限のため単一行から複数行に再フォーマット済)。

```
(DESCRIPTION=
  (ADDRESS=(PROTOCOL=tcp)(HOST=example.com)(PORT=8080))
  (Presentation=HTTP)
  (Session=RAW)
)
```

HTTPサービスが起動していないことを確認できた場合は、初期化パラメータ・ファイルに次の行を追加

(listener_nameにOracle Netローカル・リスナーを指定)し、データベースおよびリスナーを再起動します。

```
dispatchers="(PROTOCOL=TCP)"
local_listener=listener_name
```

10. Webブラウザからprint_employeesプログラムを実行します。たとえば、hostに自分のホスト・コンピュータ名を指定し、portに前のステップのPORTの値を指定すると、次のURLを使用できます。

```
http://host:port/plsql/print_employees
```

たとえば、ホストがtest.comで、HTTPポートが8080である場合は、次のように入力します。

```
http://example.com:8080/plsql/print_employees
```

Webブラウザに、hr.employees表に全従業員の姓名が記載された表のあるHTMLページが戻されます。

17.5 PL/SQLを使用したHTML出力の生成

一般的に、PL/SQL Webアプリケーションでは、ファンクション・コールを使用して出力用の各HTMLタグを生成します。このファンクションは、Oracle Databaseに付属するPL/SQL Web Toolkitパッケージの一部です。[例17-7](#)に、各HTMLタグと対応するHTPファンクションをコールして簡単なHTMLページを生成する方法を示します。

各タグに対応するファンクション・コールを作成するかわりに、HTP.PRINTファンクションを使用して、テキストとタグを両方とも出力します。[例17-8](#)に、その方法を示します。

例17-7 HTPファンクションを使用したHTMLタグの生成

```
CREATE OR REPLACE PROCEDURE html_page IS
BEGIN
    HTP.HTMLOPEN;                -- generates <HTML>
    HTP.HEADOPEN;                -- generates <HEAD>
    HTP.TITLE('Title');          -- generates <TITLE>Hello</TITLE>
    HTP.HEADCLOSE;               -- generates </HTML>
    -- generates <BODY TEXT="#000000" BGCOLOR="#FFFFFF">
    HTP.BODYOPEN( cattributes => 'TEXT="#000000" BGCOLOR="#FFFFFF"');
    -- generates <H1>Heading in the HTML File</H1>
    HTP.HEADER(1, 'Heading in the HTML File');
    HTP.PARA;                    -- generates <P>
    HTP.PRINT('Some text in the HTML file.');
```

例17-8 HTP.PRINTを使用したHTMLタグの生成

```
CREATE OR REPLACE PROCEDURE html_page2 IS
BEGIN
    HTP.PRINT('<html>');
    HTP.PRINT('<head>');
    HTP.PRINT('<meta http-equiv="Content-Type" content="text/html">');
    HTP.PRINT('<title>Title of the HTML File</title>');
    HTP.PRINT('</head>');
    HTP.PRINT('<body TEXT="#000000" BGCOLOR="#FFFFFF">');
    HTP.PRINT('<h1>Heading in the HTML File</h1>');
    HTP.PRINT('<p>Some text in the HTML file.');
```

17.6 PL/SQL Webアプリケーションへのパラメータ渡し

Webアプリケーションを様々な状況で有効に活用するには、対話性を十分に高め、ユーザーによる選択を可能にする必要があります。迅速さを求めるWeb閲覧者に対応するには、対話の効率をよくなり、ユーザーが選択項目を簡単に指定できるように、過大な決定事項やデータ入力を避ける必要があります。

PL/SQL Webアプリケーションにパラメータを渡す主な方法は、次のとおりです。

- HTMLフォーム・タグを使用します。ユーザーが1つのWebページ上のフォームに記入して、「Submit」ボタンをクリックすると、すべてのデータおよび選択項目がストア・サブプログラムに転送されます。
- URLにハードコード化します。ユーザーがリンクをクリックすると、一連の事前定義パラメータがストア・サブプログラムに転送されます。通常、ユーザーが選択する可能性があるすべての選択項目には、Webページ上に個別のリンクを挿入します。

トピック:

- [HTMLフォームからのリスト・パラメータおよびドロップダウン・リスト・パラメータ渡し](#)
- [HTMLフォームからのオプション・パラメータおよびチェック・ボックス・パラメータ渡し](#)
- [HTMLフォームからの入力フィールド・パラメータ渡し](#)
- [HTMLフォームからの非表示パラメータ渡し](#)
- [HTMLフォームからのファイルのアップロード](#)
- [記入済HTMLフォームの送信](#)
- [HTMLフォームの欠落した入力処理](#)
- [Webページ間での状態情報の保持](#)

17.6.1 HTMLフォームからのリスト・パラメータおよびドロップダウン・リスト・パラメータ渡し

リスト・ボックスおよびドロップダウン・リストは、HTMLタグ(<SELECT>)を使用して実装されます。

選択項目の数を多くする、または複数の選択を可能にする必要がある場合は、リスト・ボックスを使用します。リスト・ボックスは、項目をアルファベット順に表示する場合に適しています。これによって、ユーザーは、すべての項目を読まなくても目的の項目を短時間で検索できます。

次の場合にドロップダウン・リストを使用します。

- 選択項目が少ない
- 画面のスペースがかぎられている。
- 選択項目の順序が特殊。

ドロップダウン・リストは、初めてアクセスするユーザーの注意を引き、選択項目を読ませることができます。選択項目および順序を固定することで、ユーザーはドロップダウン・リストから項目を選択するときの動作に慣れ、素早く選択できるようになります。

[例17-9](#)に、簡単なドロップダウン・リストを示します。

例17-9 HTMLドロップダウン・リスト

```
<form>
<select name="seasons">
<option value="winter">Winter
<option value="spring">Spring
<option value="summer">Summer
<option value="fall">Fall
</select>
```

17.6.2 HTMLフォームからのオプション・パラメータおよびチェック・ボックス・パラメータ渡し

オプション・ボタンによって、NULL値(グループ内のいずれのオプションもチェックされていない場合)、またはチェックされたオプションに指定されている値のいずれかが渡されます。

一連のオプションのデフォルト値を指定するには、INPUTタグにCHECKED属性を含めるか、ストアド・サブプログラム内のパラメータにDEFAULT句を含めます。オプションのグループを設定するときは、「どちらでもない」ことを示す選択項目を含める必要があります。これは、オプションを選択した後では、別のボタンを選択することはできませんが、選択を完全にクリアすることができないためです。たとえば、あるユーザーが選択を行った後に、それが間違っていたことに気付くという状況を考慮して、「はい」および「いいえ」とともに「興味なし」または「わからない」という選択項目を含めます。

チェック・ボックスでは、ストアド・サブプログラムがNULL値、単一の値、または複数の値を受け取る場合があるため、特別な処理が必要です。

同じNAME属性を持つチェック・ボックスはすべて、チェック・ボックス・グループを構成します。グループ内のいずれのチェック・ボックスもチェックされていない場合、ストアド・サブプログラムは、対応するパラメータに対してNULL値を受け取ります。

グループ内のチェック・ボックスの1つがチェックされている場合、ストアド・サブプログラムは単一のVARCHAR2パラメータを受け取ります。

グループ内の複数のチェック・ボックスがチェックされている場合、ストアド・サブプログラムはPL/SQLのTABLE OF VARCHAR2データ型のパラメータを受け取ります。TABLE OF VARCHAR2のようなデータ型を宣言するか、OWA_UTIL.IDENT_ARRのような事前定義のデータ型を使用する必要があります。値を検索するには、次のようにループを使用します。

```
CREATE OR REPLACE PROCEDURE handle_checkboxes (  
    checkboxes owa_util.ident_arr  
) AS  
BEGIN  
    FOR i IN 1..checkboxes.count  
    LOOP  
        htp.print('<p>Check Box value: ' || checkboxes(i));  
    END LOOP;  
END;  
/
```

17.6.3 HTMLフォームからの入力フィールド・パラメータ渡し

入力フィールドでは、ユーザーが誤った形式のデータや範囲外のデータなどを入力する可能性があるため、最も厳密な妥当性チェックが必要です。可能な場合、クライアント側JavaScript関数を使用してクライアント側でデータの妥当性チェックを行い、ユーザーにかわって正しいフォーマットにするか、ユーザーに再入力を促します。

たとえば:

- ユーザーが数値入力フィールドにアルファベット文字を入力したり、文字数制限を超えた後に文字を入力することができないようにします。
- クレジット・カード番号から空白およびハイフンを暗黙的に削除します(ストアド・サブプログラムでこの形式の値が要求されている場合)。
- ユーザーが最大値を超える数値を入力した場合、すぐに通知して、ユーザーに再入力を促します。

これらの妥当性チェックが成功するとはかぎらないため、このような状況に対応できるようにストアド・サブプログラムをコーディングします。ユーザーが誤ったデータを入力したときに「Back」ボタンを押さなくて済むように、エラー・メッセージと他のすべての値が記入

された元のフォームを、1つのページに表示します。

パスワードなどの機密性の高い情報の場合、入力フィールドの<INPUT TYPE=PASSWORD>という特殊なフォームによって、入力されたテキストが非表示になります。

[例17-10](#)のプロシージャは、入力として2つの文字列を受け入れます。このプロシージャが初めて起動された場合、値を入力するための単純なフォームのプロンプトが表示されます。ユーザーが情報を送ると、同じプロシージャが再起動され、入力が正しいかどうかを確認されます。入力が正しい場合、プロシージャは入力を処理します。正しくない場合、プロシージャは、再入力するためのプロンプトを、最初の値が入力された状態で表示します。

例17-10 HTMLフォームからの入力フィールド・パラメータ渡し

```
DROP TABLE name_zip_table;
CREATE TABLE name_zip_table (
  name      VARCHAR2(100),
  zipcode   NUMBER
);
-- Store a name and associated zip code in the database.
CREATE OR REPLACE PROCEDURE associate_name_with_zipcode
  (name VARCHAR2 := NULL,
   zip  VARCHAR2 := NULL)
AS
BEGIN
  -- Each entry field must contain a value. Zip code must be 6 characters.
  -- (In a real program you perform more extensive checking.)
  IF name IS NOT NULL AND zip IS NOT NULL AND length(zip) = 6 THEN
    INSERT INTO name_zip_table (name, zipcode) VALUES (name, zip);
    HTP.PRINT('<p>The person ' || HTP.ESCAPE_SC(name) ||
              ' has the zip code ' || HTP.ESCAPE_SC(zip) || '.');
    -- If input was OK, stop here. User does not see form again.
    RETURN;
  END IF;
  -- If user entered incomplete or incorrect data, show error message.
  IF (name IS NULL AND zip IS NOT NULL)
    OR (name IS NOT NULL AND zip IS NULL)
    OR (zip IS NOT NULL AND length(zip) != 6)
  THEN
    HTP.PRINT('<p><b>Please reenter data. Fill all fields,
              and use 6-digit zip code.</b>');
  END IF;
  -- If user entered no data or incorrect data, show error message
  -- & make form invoke same procedure to check input values.
  HTP.FORMOPEN('HR.associate_name_with_zipcode', 'GET');
  HTP.PRINT('<p>Enter your name:</td>');
  HTP.PRINT('<td valign=center><input type=text name=name value="' ||
            HTP.ESCAPE_SC(name) || '">');
  HTP.PRINT('<p>Enter your zip code:</td>');
  HTP.PRINT('<td valign=center><input type=text name=zip value="' ||
            HTP.ESCAPE_SC(zip) || '">');
  HTP.FORMSUBMIT(NULL, 'Submit');
  HTP.FORMCLOSE;
END;
/
```

17.6.4 HTMLフォームからの非表示パラメータ渡し

ユーザーが同じ選択項目を毎回指定しなくても、一連のストアド・サブプログラムを介して情報を渡すには、ストアド・サブプログラムを起動するフォームに非表示パラメータを含める方法があります。最初のストアド・サブプログラムによって、このストアド・サブプログラムで生成されたHTMLフォームにユーザー名などの情報が入力されます。非表示パラメータの値は、ユーザーがその値をオプションまたは入力フィールドを介して入力した場合と同様に、次のストアド・サブプログラムに渡されます。

1つのストアド・サブプログラムから別のサブプログラムに情報を渡すには、次の方法もあります。

- 永続情報を含む「Cookie」をブラウザに送信します。ブラウザは、同じサイトから他のWebページにアクセスするときに、同じ情報をサーバーに戻します。Cookieは、各WebページのHTMLテキストの前に、ブラウザとWebサーバーの間で転送されるHTTPヘッダーを介して設定され、検索されます。
- データベース自体に情報を格納し、後でストアド・サブプログラムが検索できるようにします。この方法ではデータベース・サーバーに余分なオーバーヘッドが発生し、複数のユーザーがサーバーに同時アクセスしたときに、各ユーザーを追跡する方法を見つける必要があります。

17.6.5 HTMLフォームからのファイルのアップロード

HTMLフォームを使用して、クライアント・システム上のファイルを選択し、これをサーバーに転送できます。ストアド・サブプログラムにより、CLOB、BLOBまたは大量のデータを保持できるその他のデータ型として、データベースにファイルを挿入できます。

PL/SQL Web ToolkitおよびPL/SQLゲートウェイには、アップロードされたファイルを保持する「ドキュメント表」の概念がありません。

関連項目:

mod_plsqlユーザズ・ガイド

17.6.6 記入済HTMLフォームの送信

デフォルトでは、HTMLフォームには、データをHTMLフォームからストアド・サブプログラムまたはCGIプログラムに転送するための送信(Submit)ボタンが必要です。このボタンには、「検索」や「登録」など任意の名前を指定できます。

1つのページに複数のフォームを挿入し、各フォームに独自のフォーム要素と送信(Submit)ボタンを設定できます。非表示パラメータのみで構成されるフォームも作成できます。このフォームでは、ユーザーはボタンをクリックする以外の選択は行いません。

JavaScriptなどのスクリプト言語を使用すると、送信ボタンを削除して、ドロップダウン・リストからの選択など、別のアクションに応答してフォームが送られるようにすることができます。この方法は、ユーザーが選択する項目が1つで、送信(Submit)ボタンで確認するステップが必要でない場合に最適です。

17.6.7 HTMLフォームの欠落した入力の処理

HTMLフォームが送られると、ストアド・サブプログラムは、記入されていないフォーム要素に対してNULLパラメータを受け取ります。NULLパラメータの原因には、入力フィールドが空である、一連のチェック・ボックス、オプションまたはリスト項目がチェックされていない、またはVALUEパラメータの値が""(空の引用符)である、などがあります。

クライアント側で実行する妥当性チェックの内容にかかわらず、ストアド・サブプログラムのコードを使用して、いくつかのパラメータがNULLになる状況に対処します。

- すべてのパラメータ宣言に初期値を指定して、フォーム・パラメータが欠落した状態でストアド・サブプログラムが起動されたときに発生する例外を回避します。数値に対する初期値は0(ゼロ)に設定できます(適切な場合)。ユーザーが値を指定したかどうか確認するには、NULLを使用します。
- 初期値NULLを持つ入力パラメータ値を使用する前に、その値がNULLかどうかを確認します。
- 一部の入力パラメータが指定されていないときでも、サブプログラムが有効な結果を生成するようにします。レポートから一部のセクションを除外するか、パラメータが指定されていない場所を示すテキスト文字列またはイメージをレポート内

に表示します。

- 欠落している値を結果ページから直接入力することで、ストアド・サブプログラムを再実行できるようにします。たとえば、追加パラメータを指定して同じストアド・サブプログラムを起動するリンクを含めたり、出力の一部として値を挿入して元のフォームを表示させることができます。

17.6.8 Webページ間での状態情報の保持

Webアプリケーションでは、**状態**(特定の時点における現行のデータ・セット)の概念が特に重要です。Webページを切り替えると状態情報が簡単に失われ、ユーザーに何度も同じ選択をさせることになる場合があります。

状態情報は、HTMLフォームを使用して動的Webページ間で渡されます。情報は一連の名前/値ペアとして渡され、ストアド・サブプログラムのパラメータになります。

ユーザーが複数の選択をする必要がある場合、多くの選択項目から1つを選択する必要がある場合、または偶然に誤って選択することを回避することが重要な場合は、HTMLフォームを使用します。ユーザーは、すべての選択を行って、選択した項目を再確認すると、「Submit」ボタンを押して選択を決定します。それ以降のページでは、非表示パラメータ(<INPUT TYPE=HIDDEN>タグ)を含むフォームを使用して、これらの選択項目をページからページへと渡すことができます。

ユーザーが1つまたは2つの選択項目に関心がある場合、または選択事項がWebページ内に散在している場合、アクションをハイパーリンクとして表し、必要な名前/値ペアを問合せ文字列(URL内の「?」の後に続く部分)に含めることによって、ユーザーが送信(Submit)ボタンを簡単に見つけられるようになります。

状態情報は、Oracle Application Serverおよびそのmod_oseモジュールを使用して保持することもできます。この方法では、状態情報をパッケージ変数に格納して、ユーザーがWebサイトを移動しても、情報を使用可能のままにできます。

関連項目:

Oracle Application Serverのマニュアル・セット:

<http://www.oracle.com/technetwork/indexes/documentation/index.html>

17.7 PL/SQLサブプログラムでのネットワーク操作の実行

OracleはPL/SQLサブプログラムがPL/SQLを使用してネットワーク操作のセットを実行できるようにするパッケージを提供します(電子メールの送信、ホスト名またはアドレスの取得、TCP/IP接続の使用、HTTP URLコンテンツの取得、および表、イメージ・マップ、Cookie、CGI変数の使用)。

トピック:

- [Internet Protocol Version 6\(IPv6\)のサポート](#)
- [PL/SQLからの電子メールの送信](#)
- [PL/SQLからのホスト名またはアドレスの取得](#)
- [PL/SQLからのTCP/IP接続の使用](#)
- [PL/SQLからのHTTP URLコンテンツの取得](#)
- [PL/SQLからの表、イメージ・マップ、Cookie、およびCGI変数の使用](#)

17.7.1 Internet Protocol Version 6(IPv6)のサポート

Oracle Database 11gリリース2(11.2.0.1)では、PL/SQLネットワーク・ユーティリティ・パッケージがIPv6アドレスをサポートします。

パッケージ・インタフェースは変更されていません。ネットワーク・ホストを指定できるすべてのインタフェース・パラメータはIPv6アドレスを文字列形式で受け入れ、IPアドレスを返すすべてのインタフェースはIPv6アドレスを返します。

ただし、ネットワーク・アドレスを使用するアプリケーションでは、IPv6アドレスを処理するために、わずかな変更と再コンパイルが必要になる場合があります。IPv6アドレスは128ビットですが、IPv4アドレスは32ビットです。IPv6アドレスは、URL内では大カッコで囲む必要があります。たとえば：

```
http://[2001:0db8:85a3:08d3:1319:8a2e:0370:7344]/
```

関連項目:

- Oracle DatabaseでのIPv6サポートの詳細は、[『Oracle Database Net Services管理者ガイド』](#)を参照してください。
- 特定のPL/SQLネットワーク・ユーティリティ・パッケージでのIPv6サポートの詳細は、[『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』](#)を参照してください。

17.7.2 PL/SQLからの電子メールの送信

[例17-11](#)のように、PL/SQLサブプログラムはUTL_SMTPパッケージを使用して電子メールを送信できます。

関連項目:

UTL_SMTPパッケージの詳細は、[『Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス』](#)を参照してください。

例17-11 PL/SQLからの電子メールの送信

```
CREATE OR REPLACE PROCEDURE send_test_message
IS
    mailhost    VARCHAR2(64) := 'mailhost.example.com';
    sender       VARCHAR2(64) := 'me@example.com';
    recipient    VARCHAR2(64) := 'you@example.com';
    mail_conn    UTL_SMTP.CONNECTION;
BEGIN
    mail_conn := UTL_SMTP.OPEN_CONNECTION(mailhost, 25); -- 25 is the port
    UTL_SMTP.HELO(mail_conn, mailhost);
    UTL_SMTP.MAIL(mail_conn, sender);
    UTL_SMTP.RCPT(mail_conn, recipient);

    UTL_SMTP.OPEN_DATA(mail_conn);
    UTL_SMTP.WRITE_DATA(mail_conn, 'This is a test message.' || chr(13));
    UTL_SMTP.WRITE_DATA(mail_conn, 'This is line 2.' || chr(13));
    UTL_SMTP.CLOSE_DATA(mail_conn);

    /* If message were in single string, open_data(), write_data(),
       and close_data() could be in a single call to data(). */

    UTL_SMTP.QUIT(mail_conn);
EXCEPTION
    WHEN OTHERS THEN
```

```
-- Insert error-handling code here
RAISE;
END;
/
```

17.7.3 PL/SQLからのホスト名またはアドレスの取得

PL/SQLサブプログラムは、UTL_INADDRパッケージを使用して、ローカル・システムのホスト名または特定のホスト名のIPアドレスを判別できます。

関連項目:

UTL_INADDR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。

17.7.4 PL/SQLからのTCP/IP接続の使用

PL/SQLサブプログラムは、UTL_TCPパッケージを使用して、ネットワーク上のシステムにTCP/IP接続をオープンし、対応するソケットへの読み込みまたは書き込みを行います。

関連項目:

UTL_TCP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。

17.7.5 PL/SQLからのHTTP URLコンテンツの取得

PL/SQLサブプログラムは、UTL_HTTPパッケージを使用して次の操作を実行できます。

- HTTP URLのコンテンツの取得
通常、コンテンツはHTMLタグ・テキストの形式ですが、Webサーバーからダウンロード可能な種類のファイル(たとえば、プレーン・テキストまたはJPEGイメージ)であることもあります。
- HTTPセッションの詳細(ヘッダー、cookie、リダイレクト、プロキシ・サーバー、保護サイトのIDとパスワード、CGIパラメータなど)の管理
- HTTP 1.1永続接続を使用した同じWebサイトへの同時アクセスの高速化

PL/SQLサブプログラムは、ファンクションUTL_URL.ESCAPEやUTL_URL.UNESCAPEを使用して、UTL_HTTPパッケージで使用するためにURLの構築と解釈を行うことができます。

[例17-12](#)のPL/SQLプロシージャは、UTL_HTTPパッケージを使用してHTTP URLのコンテンツを取得します。

次のブロックでは、[例17-12](#)のプロシージャのコールの例を示していますが、そのURLは存在しないページのもので、自分のWebブラウザからURLを代入します。

```
BEGIN
  show_url('http://www.oracle.com/no-such-page.html');
  show_url('http://www.oracle.com/protected-page.html');
  show_url
    ('http://www.oracle.com/protected-page.html', 'username', 'password');
END;
/
```

関連項目:

- UTL_HTTP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。
- UTL_URL.ESCAPEおよびUTL_URL.UNESCAPEの詳細は、『[Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス](#)』を参照してください。

例17-12 PL/SQLからのHTTP URLコンテンツの取得

```
CREATE OR REPLACE PROCEDURE show_url
(url          IN VARCHAR2,
username IN VARCHAR2 := NULL,
password IN VARCHAR2 := NULL)
AS
  req          UTL_HTTP.REQ;
  resp         UTL_HTTP.RESP;
  name_        VARCHAR2(256);
  value_       VARCHAR2(1024);
  data_        VARCHAR2(255);
  my_scheme    VARCHAR2(256);
  my_realm     VARCHAR2(256);
  my_proxy     BOOLEAN;
BEGIN
  -- When going through a firewall, pass requests through this host.
  -- Specify sites inside the firewall that do not need the proxy host.

  UTL_HTTP.SET_PROXY('proxy.example.com', 'corp.example.com');

  -- Ask UTL_HTTP not to raise an exception for 4xx and 5xx status codes,
  -- rather than just returning the text of the error page.

  UTL_HTTP.SET_RESPONSE_ERROR_CHECK(FALSE);

  -- Begin retrieving this web page.
  req := UTL_HTTP.BEGIN_REQUEST(url);

  -- Identify yourself.
  -- Some sites serve special pages for particular browsers.
  UTL_HTTP.SET_HEADER(req, 'User-Agent', 'Mozilla/4.0');

  -- Specify user ID and password for pages that require them.
  IF (username IS NOT NULL) THEN
    UTL_HTTP.SET_AUTHENTICATION(req, username, password);
  END IF;

  -- Start receiving the HTML text.
  resp := UTL_HTTP.GET_RESPONSE(req);

  -- Show status codes and reason phrase of response.
  DBMS_OUTPUT.PUT_LINE('HTTP response status code: ' || resp.status_code);
  DBMS_OUTPUT.PUT_LINE
    ('HTTP response reason phrase: ' || resp.reason_phrase);

  -- Look for client-side error and report it.
  IF (resp.status_code >= 400) AND (resp.status_code <= 499) THEN

    -- Detect whether page is password protected
    -- and you didn't supply the right authorization.

    IF (resp.status_code = UTL_HTTP.HTTP_UNAUTHORIZED) THEN
      UTL_HTTP.GET_AUTHENTICATION(resp, my_scheme, my_realm, my_proxy);
      IF (my_proxy) THEN
```

```

    DBMS_OUTPUT.PUT_LINE('Web proxy server is protected.');
```

```

    DBMS_OUTPUT.PUT('Please supply the required ' || my_scheme ||
        ' authentication username for realm ' || my_realm ||
        ' for the proxy server.');
```

```

ELSE
    DBMS_OUTPUT.PUT_LINE('Web page ' || url || ' is protected.');
```

```

    DBMS_OUTPUT.PUT('Please supplied the required ' || my_scheme ||
        ' authentication username for realm ' || my_realm ||
        ' for the web page.');
```

```

END IF;
ELSE
    DBMS_OUTPUT.PUT_LINE('Check the URL.');
```

```

END IF;

UTL_HTTP.END_RESPONSE(resp);
RETURN;

-- Look for server-side error and report it.
ELSIF (resp.status_code >= 500) AND (resp.status_code <= 599) THEN
    DBMS_OUTPUT.PUT_LINE('Check if the website is up.');
```

```

    UTL_HTTP.END_RESPONSE(resp);
    RETURN;
END IF;

-- HTTP header lines contain information about cookies, character sets,
-- and other data that client and server can use to customize each
-- session.

FOR i IN 1..UTL_HTTP.GET_HEADER_COUNT(resp) LOOP
    UTL_HTTP.GET_HEADER(resp, i, name_, value_);
    DBMS_OUTPUT.PUT_LINE(name_ || ': ' || value_);
END LOOP;

-- Read lines until none are left and an exception is raised.
LOOP
    UTL_HTTP.READ_LINE(resp, value_);
    DBMS_OUTPUT.PUT_LINE(value_);
END LOOP;
EXCEPTION
    WHEN UTL_HTTP.END_OF_BODY THEN
        UTL_HTTP.END_RESPONSE(resp);
END;
/
```

17.7.6 PL/SQLからの表、イメージ・マップ、Cookie、およびCGI変数の使用

PL/SQLサブプログラムは、Oracle提供のパッケージやOracle HTTP Server(OHS)のmod_plsqlプラグインを使用して、問合せ結果のHTML表としての書式設定、イメージ・マップの生成、HTTP cookieの取得、CGI変数の値の確認、その他の一般的なWeb操作の実行を行います。

これらのパッケージのドキュメントは、データベース・ドキュメント・ライブラリの一部ではありません。ドキュメントの場所は、アプリケーション・サーバーによって異なります。これらのパッケージを使用するには、次に示すSQL*PlusのDESCRIBE文を使用して、サブプログラム名およびパラメータを確認します。

```

DESCRIBE HTTP;
DESCRIBE HTF;
DESCRIBE OWA_UTIL;
```

18 連続問合せ通知(CQN)の使用

連続問合せ通知(CQN)を使用すると、問合せをオブジェクト変更通知用(デフォルト)または問合せ結果変更通知用としてアプリケーションからデータベースに登録できます。登録済の問合せで参照されるオブジェクトは、登録済オブジェクトです。

オブジェクト変更通知(OCN)に対する問合せが登録されると、その問合せで参照されるオブジェクトがトランザクションによって変更されてコミットされるたびに、問合せ結果が変更されたかどうかにかかわらず、データベースからアプリケーションに通知されます。

問合せ結果変更通知(QRCN)に対して問合せが登録されると、その問合せの結果がトランザクションによって変更されてコミットされるたびに、データベースからアプリケーションに通知されます。

CQN登録により、1つ以上の問合せのリストが通知タイプ(OCNまたはQRCN)および通知ハンドラに関連付けられます。CQN登録を作成するには、PL/SQLインタフェースまたはOracle Call Interface(OCI)を使用できます。PL/SQLインタフェースを使用する場合、通知ハンドラはサーバー側のPL/SQLストアド・プロシージャになり、OCIを使用する場合、通知ハンドラはクライアント側のCコールバック・プロシージャになります。

この章では、CQNの一般的な概念と、PL/SQL CQNインタフェースの使用方法について説明します。

トピック:

- [オブジェクト変更通知\(OCN\)について](#)
- [問合せ結果変更通知\(QRCN\)について](#)
- [通知を生成するイベント](#)
- [通知内容](#)
- [CQNに適したアプリケーション](#)
- [CQN登録の作成](#)
- [PL/SQLを使用したCQN登録の作成](#)
- [OCIを使用したCQN登録の作成](#)
- [CQN登録の問合せ](#)
- [通知の解析](#)

ノート:



OCN および QRCN という用語は、通知タイプと通知自体の両方を指します。アプリケーションにより OCN に対する問合せが登録されると、データベースからアプリケーションに OCN が送信され、アプリケーションにより QRCN に対する問合せが登録されると、データベースからアプリケーションに QRCN が送信されます。

関連項目:

CQNのためのOCIの使用の詳細は、[『Oracle Call Interfaceプログラマーズ・ガイド』](#)を参照してください。

18.1 オブジェクト変更通知(OCN)について

アプリケーションでオブジェクト変更通知(OCN)に対する問合せが登録されると、その問合せに関連付けられているオブジェクトがトランザクションによって変更されてコミットされるたびに、問合せ結果が変更されたかどうかにかかわらず、データベースからアプリケーションにOCNが送信されます。

たとえば、アプリケーションで[例18-1](#)のOCNに対する問合せが登録され、EMPLOYEES表を変更するトランザクションがユーザーによってコミットされると、変更された行が問合せの条件句を満たしていない場合でも(DEPARTMENT_ID = 5の場合など)、データベースからアプリケーションにOCNが送信されます。

例18-1 変更通知として登録される問合せ

```
SELECT EMPLOYEE_ID, SALARY
FROM EMPLOYEES
WHERE DEPARTMENT_ID = 10;
```

18.2 問合せ結果変更通知(QRCN)について

ノート:



QRCNを使用するには、データベースの COMPATIBLE 初期化パラメータが 11.0.0 以上であり、自動 UNDO 管理(AUM)が有効(デフォルト設定)である必要があります。

アプリケーションで問合せ結果変更通知(QRCN)に対する問合せが登録されると、その問合せの結果がトランザクションによって変更されてコミットされるたびに、データベースからアプリケーションにQRCNが送信されます。

たとえば、アプリケーションで[例18-1](#)のQRCNに対する問合せが登録されると、問合せ結果セットが変更された場合のみ、つまり、次のデータ操作言語(DML)文のいずれかがコミットされた場合にのみ、データベースからアプリケーションにQRCNが送信されます。

- 問合せの条件句(DEPARTMENT_ID = 10)を満たす行のINSERTまたはDELETE。
- 問合せの条件句(DEPARTMENT_ID = 10)を満たしている行のEMPLOYEE_IDまたはSALARY列へのUPDATE。
- 行の値が10から10以外の値に変更され、その行が結果セットから削除される原因となったDEPARTMENT_ID列へのUPDATE。
- 行の値が10以外の値から10に変更され、その行が結果セットに追加される原因となったDEPARTMENT_ID列へのUPDATE。

デフォルトの通知タイプはOCNです。QRCNの場合は、CQ_NOTIFICATION\$_REG_INFOオブジェクトのQOSFLAGS属性にQOS_QUERYを指定します。

QRCNでは、保証モード(デフォルト)またはベストエフォート・モードを選択できます。

トピック:

- [保証モード](#)
- [ベストエフォート・モード](#)

関連項目:

- COMPATIBLE初期化パラメータの詳細は、[『Oracle Database管理者ガイド』](#)を参照してください。
- AUMの詳細は、[『Oracle Database管理者ガイド』](#)を参照してください。

18.2.1 保証モード

保証モードでは、誤検出はありません。つまり、問合せ結果セットの変更が保証される場合にのみ、データベースからアプリケーションにQRCNが送信されます。

たとえば、アプリケーションで[例18-1](#)のQRCNに対する問合せが登録され、従業員201が部門10に所属しており、次の文が実行されるとします。

```
UPDATE EMPLOYEES
SET SALARY = SALARY + 10
WHERE EMPLOYEE_ID = 201;
UPDATE EMPLOYEES
SET SALARY = SALARY - 10
WHERE EMPLOYEE_ID = 201;
COMMIT;
```

このトランザクションの各UPDATE文によって問合せ結果セットは変更されますが、各文を合計すると問合せ結果セットへの影響はないため、このトランザクションに対してはデータベースからアプリケーションにQRCNは送信されません。

保証モードでは、CQ_NOTIFICATION\$_REG_INFOオブジェクトのQOSFLAGS属性にQOS_BEST_EFFORTではなくQOS_QUERYを指定します。

問合せの中には、保証モードのQRCNに対して複雑すぎるものがあります。

関連項目:

保証モードで登録可能な問合せの特性は、[「保証モードでQRCNに対して登録可能な問合せ」](#)を参照してください。

18.2.2 ベストエフォート・モード

保証モードでは複雑すぎる問合せの中には、ベストエフォート・モードでQRCNに対して登録できるものがあります。このモードでは、簡略バージョンの問合せがCQNによって作成および登録されます。

次の2つの例で、この動作を示します。

- [例: 保証モードのQRCNに対して複雑すぎる問合せ](#)
- [例: 簡略バージョンによってオブジェクトが無効になる問合せ](#)

18.2.2.1 例: 保証モードのQRCNに対して複雑すぎる問合せ

[例18-2](#)の問合せは、集計関数SUMを含んでおり、保証モードのQRCNに対しては複雑すぎます。

例18-2 保証モードのQRCNに対して複雑すぎる問合せ

```
SELECT SUM(SALARY)
FROM EMPLOYEES
WHERE DEPARTMENT_ID = 20;
```

ベストエフォート・モードでは、この例の問合せの次のような簡略バージョンがCQNによって登録されます。

```
SELECT SALARY
```



```
FROM EMPLOYEES
WHERE DEPARTMENT_ID = 20;
```

元の問合せの結果が変更されるたびに、その簡略バージョンの結果も変更されるため、簡略化が原因で通知が失われることはありません。ただし、簡略バージョンの結果は元の問合せの結果が変更されなくても変更される場合があるため、簡略化によって誤検出が発生することがあります。

ベストエフォート・モードでは、データベースは次の処理を行います。

- 通知関連の処理によるOLTPレスポンスのオーバーヘッドを次のように最小化します。
 - 単一表問合せの場合、データベースでは、どの列が変更されたか、および変更された行によってどの条件句が満たされたかに基づいて、問合せ結果が変更されたかどうかを判断します。
 - 複数表問合せ(結合)の場合、データベースでは、複数の表の間の主キー/外部キー制約関係を使用して、問合せ結果が変更されたかどうかを判断します。
- DML文によって問合せ結果セットが変更されるたびに、最初のDML文によって行われた変更が後続のDML文によって無効になった場合でも、アプリケーションにQRCNを送信します。

ベストエフォート・モードでのオーバーヘッドを最小化により、CQNで簡略化されない問合せに対しても、誤検出がまれに発生します。たとえば、この例の問合せと、[「保証モード」](#)のトランザクションを想定します。ベストエフォート・モードでは、CQNによって問合せは簡略化されませんが、トランザクションによって誤検出が発生します。

18.2.2.2 例：簡略バージョンによってオブジェクトが無効になる問合せ

一部のタイプの問合せは非常に簡略化されているため、無効化がオブジェクト・レベルで発生します。つまり、該当の問合せで参照されるいずれかのオブジェクトが変更されるたびに発生します。このような問合せの例としては、サポートされていない列型を使用する問合せや、副問合せを含んでいる問合せがあります。この問題を解決するには、元の問合せを書き換えます。

たとえば、[例18-3](#)の問合せは、副問合せを含んでいるため、保証モードのQRCNに対しては複雑すぎます。

例18-3 簡略バージョンによってオブジェクトが無効になる問合せ

```
SELECT SALARY
FROM EMPLOYEES
WHERE DEPARTMENT_ID IN (
  SELECT DEPARTMENT_ID
  FROM DEPARTMENTS
  WHERE LOCATION_ID = 1700
);
```

ベストエフォート・モードでは、この例の問合せがCQNによって次のように簡略化されます。

```
SELECT * FROM EMPLOYEES, DEPARTMENTS;
```

問合せの簡略化により、オブジェクトが無効になる場合があります。ただし、元の問合せを次のように書き換えると、その問合せを保証モードまたはベストエフォート・モードで登録できます。

```
SELECT SALARY
FROM EMPLOYEES, DEPARTMENTS
WHERE EMPLOYEES.DEPARTMENT_ID = DEPARTMENTS.DEPARTMENT_ID
AND DEPARTMENTS.LOCATION_ID = 1700;
```

ベストエフォート・モードでのみ登録可能な問合せについては、[「ベストエフォート・モードでのみQRCNに対して登録可能な問合せ」](#)で説明します。

デフォルトのQRCNモードは保証モードです。ベストエフォート・モードの場合は、CQ_NOTIFICATION\$_REG_INFOオブジェ

クトのQOSFLAGS属性にQOS_BEST_EFFORTを指定します。

18.3 通知を生成するイベント

通知を生成するイベントは次のとおりです。

- [DMLトランザクションのコミット](#)
- [DDL文のコミット](#)
- [登録解除](#)
- [グローバル・イベント](#)

18.3.1 DMLトランザクションのコミット

通知タイプがOCNの場合、1つ以上の登録済オブジェクトを変更するあらゆるDMLトランザクションで、コミット時に各オブジェクトに対して通知が1つ生成されます。

通知タイプがQRCNの場合、1つ以上の登録済問合せの結果を変更するあらゆるDMLトランザクションで、コミット時に通知が生成されます。この通知には、結果が変更された問合せのIDが含まれます。

どちらの通知タイプでも、通知には次の内容が含まれます。

- 変更された各表の名前
- 操作タイプ(INSERT、UPDATEまたはDELETE)
- 変更された各行のROWID(ROWIDオプションを使用して登録が作成され、変更された行の数があまり多くない場合)。

関連項目:

[ROWIDオプション](#)

18.3.2 DDL文のコミット

OCNとQRCNの両方について、次のデータ定義言語(DDL)文により、コミット時に通知が生成されます。

- ALTER TABLE
- TRUNCATE TABLE
- FLASHBACK TABLE
- DROP TABLE

ノート:

通知タイプがOCNの場合、DROP TABLE 文のコミットによって DROP NOTIFICATION が生成されます。



削除された表に対する問合せの OCN 登録は、その表(すでに存在しない表)から関連付けが解除されますが、登録自体は存続します。これらの登録のいずれかが削除された表以外のオブジェクトに関連付けられている場合、それらのオブジェクトに対する変更がコミットされると、通知は引き続き生成されます。削除された表にのみ関連付けられていた登録も存続し、その作成者は問合せ(およびその参照オブジェクト)をその登録に追加できます。

OCN 登録は、問合せの登録時点におけるオブジェクトのバージョンと定義に基づきます。オブジェクトが削除されると、そのオブジェクトの登録の関連付けはオブジェクトから永続的に解除されます。オブジェクトが削除されたオブジェクトと同じ名前および同じスキーマで作成された場合、作成されたオブジェクトは削除されたオブジェクトに関連付けられていた OCN 登録には関連付けられません。

通知タイプがQRCNの場合、次のようになります。

- 通知には次の内容が含まれます。
 - 結果が変更された問合せのID
 - 変更された表の名前
 - DDL操作のタイプ
- 登録済問合せを無効にするDDL操作の中には、その問合せを登録解除する原因となるものもあります。

たとえば、次の問合せがQRCNに対して登録されるとします。

```
SELECT COL1 FROM TEST_TABLE
WHERE COL2 = 1;
```

TEST_TABLEのスキーマは次のとおりとします。

```
(COL1 NUMBER, COL2 NUMBER, COL3 NUMBER)
```

この場合、次のDDL文により、コミット時に問合せが無効になり、問合せが登録から削除されます。

```
ALTER TABLE DROP COLUMN COL2;
```

18.3.3 登録解除

OCNとQRCNの両方について、登録解除(データベースからの登録の削除)によって通知が生成されます。データベースで登録が削除される原因は次のとおりです。

- タイムアウト

問合せの登録時にTIMEOUTがゼロ以外の値で指定された場合、指定の時間間隔後にデータベースによって登録がパージされます。

問合せの登録時にQOS_DEREG_NFYが指定された場合、最初の通知の生成後にデータベースによって登録がパージされます。
- 権限の消失

登録済問合せに関連付けられているオブジェクトに対する権限が消失し、通知タイプがOCNの場合、データベースによって登録がパージされます。(通知タイプがQRCNの場合は、データベースによってその問合せは登録から削除されますが、登録はパージされません。)

クライアント・アプリケーションによる明示的な登録解除の実行時には、通知は生成されません。

関連項目:

問合せの登録に必要な権限は、[「CQN登録を作成するための前提条件」](#)を参照してください。

18.3.4 グローバル・イベント

グローバル・イベントEVENT_STARTUPおよびEVENT_SHUTDOWNにより、通知が生成されます。

Oracle RAC環境では、次のイベントによって通知が生成されます。

- EVENT_STARTUP(データベースの最初のインスタンスの起動時)
- EVENT_SHUTDOWN(データベースの最終インスタンスの停止時)
- EVENT_SHUTDOWN_ANY(データベースのいずれかのインスタンスの停止時)

前述のグローバル・イベントは、DBMS_CQ_NOTIFICATIONパッケージで定義されている定数です。

関連項目:

DBMS_CQ_NOTIFICATIONパッケージの詳細は、[『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』](#)を参照してください。

18.4 通知内容

通知には、次の情報の一部またはすべてが含まれます。

- イベントのタイプ(次のいずれか)
 - 起動
 - オブジェクト変更
 - 問合せ結果変更
 - 登録解除
 - 停止
- 影響を受けた登録の登録ID
- 変更されたオブジェクトの名前
- ROWIDオプションが指定されている場合の、変更された行のROWID
- 通知タイプがQRCNの場合の、結果が変更された問合せの問合せID
- 通知がDMLまたはDDL文の結果として生成された場合
 - 変更された表の名前の配列
 - 操作タイプ(INSERT、UPDATEなど)

通知には、変更されたデータ自体は含まれません。たとえば、5000から6000への月給の増額があっても、通知には示されません。変更されたオブジェクト、行または問合せ結果の最新の値を取得するには、アプリケーションからデータベースに問い合わせる必要があります。

18.5 CQNに適したアプリケーション

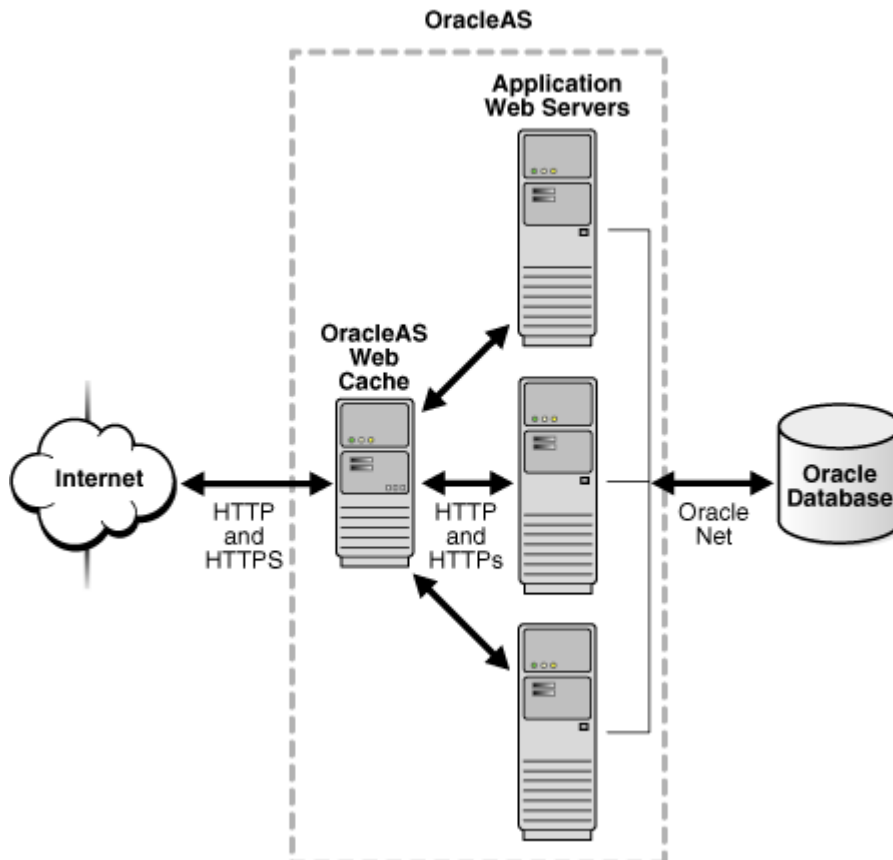
CQNに適したアプリケーションは、データベースへのネットワーク・ラウンドトリップを回避するために、変更の頻度の低いオブジェクトに対する問合せの結果セットを中間層にキャッシュするアプリケーションです。これらのアプリケーションでは、CQNを使用して、

キャッシュされる問合せを登録できます。このようなアプリケーションは、通知を受信すると、登録済問合せを再実行してキャッシュをリフレッシュできます。

このようなアプリケーションの例としては、Webフォーラムがあります。ユーザーは、コンテンツがデータベースに挿入されても、挿入後すぐにそのコンテンツを参照する必要はないため、このアプリケーションでは情報を中間層にキャッシュし、キャッシュをいつリフレッシュするかをCQNで指示できます。

[図18-1](#)に、データがデータベースから提供され、中間にキャッシュされた後、インターネット経由でアクセスされる一般的な使用例を示します。

図18-1 中間層キャッシュ



中間層のアプリケーションは、キャッシュをデータベースと相対でできるかぎり最新の状態に保ちつつ、データベース・オブジェクトのキャッシュ・コピーに迅速にアクセスする必要があります。キャッシュ・データは、トランザクションでデータが変更されてコミットされると不要になるため、アプリケーションが不正な結果にアクセスする危険性があります。アプリケーションがCQNを使用している場合、データベースは、登録済オブジェクトに変更が発生した時点で変更内容の詳細を使用して通知を公開できます。この通知に回答して、アプリケーションではキャッシュ・データをバックエンド・データベースからフェッチしてリフレッシュできます。

[図18-2](#)に、中間層のWebクライアントが通知を受信して処理するプロセスを示します。

図18-2 連続問合せ通知(CQN)の基本プロセス

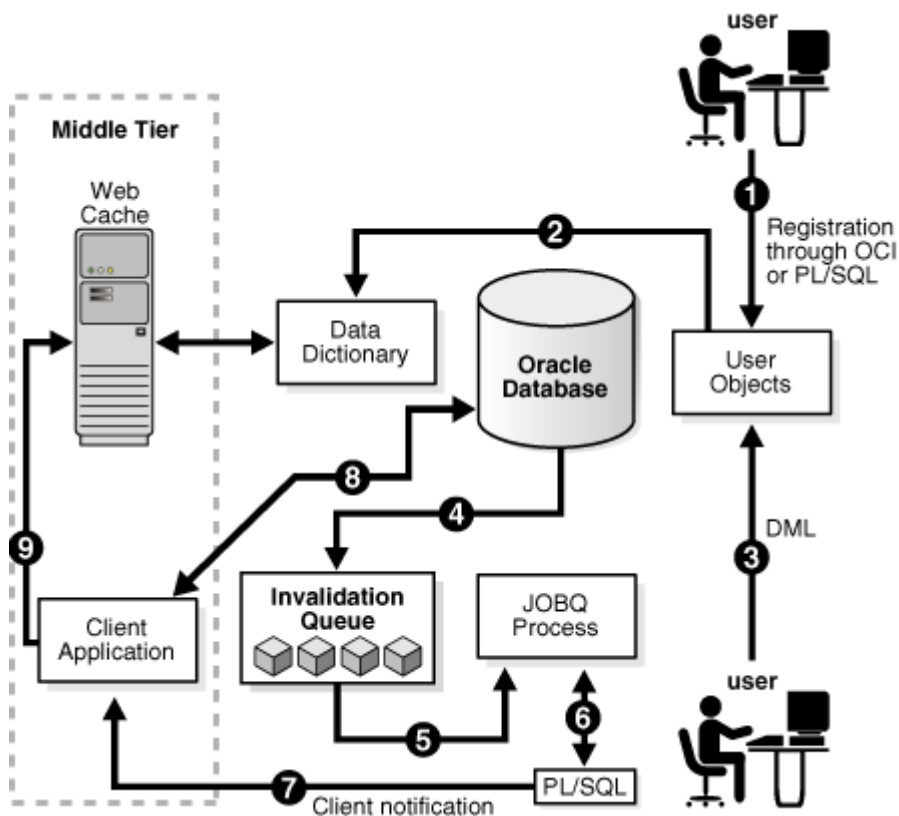


図18-2の各ステップは次のとおりです(PL/SQLを使用して登録が作成され、アプリケーションによりHR.EMPLOYEESに対する問合せの結果セットがキャッシュされている場合)。

1. 開発者は、PL/SQLを使用して、問合せに対するCQN登録を作成します。このプロセスでは、通知を処理するストアードPL/SQLプロシージャを作成した後、PL/SQL CQNインタフェースを使用して問合せに対する登録を作成し、作成したPL/SQLプロシージャを通知ハンドラとして指定します。
2. データベースでは、データ辞書に登録情報が移入されます。
3. ユーザーがバックエンド・データベースのHR.EMPLOYEES表の行を更新し、その更新をコミットすると、問合せ結果が変更されます。これで、中間層でキャッシュされていたHR.EMPLOYEESのデータが無効になります。
4. データベースにより、内部キューに変更を説明するメッセージが追加されます。
5. データベースからJOBQバックグラウンド・プロセスに、通知メッセージが通知されます。
6. JOBQプロセスにより、クライアント・アプリケーションから指定されたストアード・プロシージャが実行されます。この例では、JOBQからサーバー側PL/SQLプロシージャにデータが渡されます。PL/SQL通知ハンドラの実装により、通知の処理方法が決まります。
7. 開発者は、サーバー側PL/SQLプロシージャ内に、登録済オブジェクトに対する変更を中間層のクライアント・アプリケーションに通知するための論理を実装できます。たとえば、HR.EMPLOYEES内で変更された行のROWIDをアプリケーションに通知します。
8. 中間層のクライアント・アプリケーションでは、バックエンド・データベースを問い合せて、変更があった行のデータを取得します。
9. クライアント・アプリケーションにより、キャッシュがこのデータで更新されます。

18.6 CQN登録の作成

CQN登録により、1つ以上の問合せのリストが通知タイプおよび通知ハンドラに関連付けられます。

通知タイプは、OCNまたはQRCNです。

CQN登録を作成するには、2つのインタフェースのいずれかを使用します。

- **PL/SQLインタフェース**

PL/SQLインタフェースを使用する場合、通知ハンドラはサーバー側PL/SQLストアド・プロシージャです。

- **Oracle Call Interface(OCI)**

OCIを使用する場合、通知ハンドラはクライアント側Cコールバック・プロシージャです。

作成された登録は、データベースに格納されます。Oracle RAC環境では、この登録はすべてのデータベース・インタフェースから参照できます。いずれかのデータベース・インスタンスの問合せ結果がトランザクションによって変更されると、通知が生成されます。

デフォルトでは、登録は、それを作成したアプリケーションによって明示的に登録解除されるまで、または(権限の消失などが原因で)データベースによって暗黙的にパージされるまで存続します。

関連項目:

- [オブジェクト変更通知\(OCN\)について](#)
- [問合せ結果変更通知\(QRCN\)について](#)
- [PL/SQLを使用したCQN登録の作成](#)
- [OCIを使用したCQN登録の作成](#)

18.7 PL/SQLを使用したCQN登録の作成

この項では、PL/SQLを使用したCQN登録の作成について説明します。PL/SQLインタフェースを使用する場合、通知ハンドラはサーバー側PL/SQLストアド・プロシージャです。

トピック:

- [PL/SQL CQN登録インタフェース](#)
- [CQN登録オプション](#)
- [CQN登録を作成するための前提条件](#)
- [オブジェクト変更通知\(OCN\)に対して登録可能な問合せ](#)
- [問合せ結果変更通知\(QRCN\)に対して登録可能な問合せ](#)
- [PL/SQLを介したCQNに対する問合せの登録](#)
- [CQN登録のベスト・プラクティス](#)
- [CQN登録のトラブルシューティング](#)
- [登録の削除](#)
- [CQNの構成: 使用例](#)

18.7.1 PL/SQL CQN登録インタフェース

PL/SQL CQN登録インタフェースは、DBMS_CQ_NOTIFICATIONパッケージを使用して実装されます。登録ブロックを開くに

は、DBMS_CQ_NOTIFICATION.NEW_REG_STARTファンクションを使用します。通知タイプや通知ハンドラなどの登録詳細は、CQ_NOTIFICATION\$_REG_INFOオブジェクトの一部として指定し、このオブジェクトは引数としてNEW_REG_STARTプロシージャに渡されます。登録ブロックがオープンの実行したすべての問合せは、CQNに登録されます。通知タイプとしてQRCNを指定すると、データベースによって各問合せに問合せIDが割り当てられます。この問合せIDは、DBMS_CQ_NOTIFICATION.CQ_NOTIFICATION_QUERYIDファンクションを使用して取得できます。登録ブロックをクローズするには、DBMS_CQ_NOTIFICATION.REG_ENDファンクションを使用します。

関連項目:

- DBMS_CQ_NOTIFICATIONパッケージの詳細は、『[Oracle Database PL/SQLパッケージおよびタイプ・リファレンス](#)』を参照してください。
- [PL/SQLを介したCQNに対する問合せの登録](#)

18.7.2 CQN登録オプション

CQN登録のデフォルトは、[表18-1](#)に示すオプションを使用して変更できます。

表18-1 連続問合せ通知登録オプション

オプション	説明:
通知のタイプ	QRCN を指定します(デフォルトは OCN)。
QRCN モード 脚注 1	ベストエフォート・モードを指定します(デフォルトは保証モード)。
ROWID	変更された各行の ROWID 疑似列の値を通知に含めます。
操作フィルタ 脚注 2	指定したフィルタ条件を操作タイプが満たした場合のみ、通知を公開します。
トランザクション・ラグ 脚注 2	非推奨。かわりに通知グループ化を使用してください。
通知グループ化	通知のグループ化方法を指定します。
信頼性	通知を(デフォルトの共有メモリーではなく)永続データベース・キューに格納します。
通知時にページ	最初の通知後に登録をページします。
タイムアウト	指定の時間間隔後に登録をページします。

脚注1

通知タイプがQRCNの場合にのみ適用されます。

脚注2

通知タイプがOCNの場合にのみ適用されます。

トピック:

- [「通知タイプ」オプション](#)
- [QRCNモード\(QRCN通知タイプのみ\)](#)
- [ROWIDオプション](#)
- [操作フィルタ・オプション\(OCN通知タイプのみ\)](#)
- [トランザクション・ラグ・オプション\(OCN通知タイプのみ\)](#)
- [通知グループ化オプション](#)
- [信頼可能オプション](#)
- [「通知時にページ」および「タイムアウト」オプション](#)

18.7.2.1 「通知タイプ」オプション

通知タイプは、OCNおよびQRCNです。

関連項目:

- [オブジェクト変更通知\(OCN\)について](#)
- [問合せ結果変更通知\(QRCN\)について](#)

18.7.2.2 QRCNモード(QRCN通知タイプのみ)

QRCNモード・オプションは、通知タイプがQRCNの場合にのみ適用されます。通知タイプをQRCNに設定する手順は、[「通知タイプ」オプション](#)を参照してください。

次に、QRCNモードを示します。

- 保証
- ベスト・エフォート

デフォルトは保証モードです。ベストエフォート・モードの場合は、CQ_NOTIFICATION\$_REG_INFOオブジェクトのQOSFLAGS属性にQOS_BEST_EFFORTを指定します。

関連項目:

- [保証モード](#)
- [ベストエフォート・モード](#)

18.7.2.3 ROWIDオプション

ROWIDオプションにより、変更された行それぞれのROWID疑似列の値(その行のrowid)が通知に含まれるようになります。変更された各行のROWIDオプションを通知に含めるには、CQ_NOTIFICATION\$_REG_INFOオブジェクトのQOSFLAGS属性にQOS_ROWIDSを指定します。

ノート:



ハイブリッド列圧縮(HCC)で圧縮された表の行を更新すると、行の ROWID が変更されます。特定の Oracle ストレージ・システムの機能である HCC の詳細は、[『Oracle Database 概要』](#)を参照してください。

アプリケーションでは、次の形式の問合せを実行すると、変更された行の内容を通知のROWID情報から取得できます。

```
SELECT * FROM table_name_from_notification
WHERE ROWID = rowid_from_notification;
```

ROWIDは、外部文字列形式で公開されます。通常のヒープ表では、ROWIDの長さは18バイトです。索引構成表(IOT)では、ROWIDの長さは主キーのサイズによって異なり、18バイトより長い場合があります。

ROWIDに十分なメモリーがサーバーにない場合、通知はFULL - TABLE - NOTIFICATIONにロールアップされることがあります(これは、通知記述子内の特殊フラグで示されます)。FULL - TABLE - NOTIFICATIONに対して考えられる原因は次のとおりです。

- ROWIDによって消費される共有メモリーの合計が動的共有プール・サイズの1%を超えています。
- 1回のトランザクションで1つの登録済オブジェクト内で変更された行が多すぎます(上限は約80行)。
- IOTについて変更された行の論理ROWIDの合計長が長すぎます(上限は約1800バイト)。
- 通知グループ化オプションNTFN_GROUPING_TYPEに値DBMS_CQ_NOTIFICATION.NTFN_GROUPING_TYPE_SUMMARYが指定されています([「通知グループ化オプション」](#)を参照)。

FULL - TABLE - NOTIFICATIONにはROWIDは含まれないため、これを受信したアプリケーションでは表全体(つまり、すべての行)が変更された可能性があると考えられます。

18.7.2.4 操作フィルタ・オプション(OCN通知タイプのみ)

操作フィルタ・オプションは、通知タイプがOCNの場合にのみ適用されます。

操作フィルタ・オプションを使用すると、通知を生成する操作のタイプを指定できます。

デフォルトは全操作です。一部の操作でのみ通知が生成されるように指定するには、CQ_NOTIFICATION\$_REG_INFOオブジェクトのOPERATIONS_FILTER属性を使用します。OPERATIONS_FILTER属性では、操作のタイプを表す定数を使用してタイプを指定し、この定数は、DBMS_CQ_NOTIFICATIONパッケージで次のように定義されています。

演算子	定数
INSERT	DBMS_CQ_NOTIFICATION.INSERTOP
UPDATE	DBMS_CQ_NOTIFICATION.UPDATEOP
DELETE	DBMS_CQ_NOTIFICATION.DELETEOP
ALTEROP	DBMS_CQ_NOTIFICATION.ALTEROP
DROPOP	DBMS_CQ_NOTIFICATION.DROPOP

演算子	定数
UNKNOWNOP	DBMS_CQ_NOTIFICATION.UNKNOWNOP
すべて(デフォルト)	DBMS_CQ_NOTIFICATION.ALL_OPERATIONS

複数の操作を指定するには、ビット単位のORを使用します。たとえば:

```
DBMS_CQ_NOTIFICATION.INSERTOP + DBMS_CQ_NOTIFICATION.DELETEOP
```

QOS_QUERYは通知タイプとしてQRCNを指定するため、QOSFLAGS属性にQOS_QUERYも指定した場合はOPERATIONS_FILTERは無効です。

関連項目:

DBMS_CQ_NOTIFICATIONパッケージの詳細は、『[Oracle Database PL/SQLパッケージおよびタイプ・リファレンス](#)』を参照してください。

18.7.2.5 トランザクション・ラグ・オプション(OCN通知タイプのみ)

トランザクション・ラグ・オプションは、通知タイプがOCNの場合にのみ適用されます。

ノート:



このオプションは非推奨です。フロー制御の通知を実装するには、『[通知グループ化オプション](#)』を使用してください。

トランザクション・ラグ・オプションでは、クライアント・アプリケーションがデータベースから遅れることのできるトランザクション数を指定します。数が0の場合、登録済オブジェクトを変更するトランザクションごとに通知が送信されます。数が5の場合、登録済オブジェクトを変更するトランザクションのうち5番目ごとに通知が送信されます。データベースでは、クライアントで中間の変更が失われるように、中間の変更をオブジェクトの粒度で追跡して通知に含めます。

0より大きいトランザクション・ラグが役立つのは、アプリケーションでフロー制御の通知を実装している場合のみです。アプリケーションでは、通知がラグに達する程度に十分な頻度で生成されるようにし、無限に遅延されないようにしてください。

TRANSACTION_LAGを指定すると、QOS_ROWIDSも指定した場合でも、通知にROWIDは含まれません。

18.7.2.6 通知グループ化オプション

デフォルトでは、通知はその原因となるイベントの直後に生成されます。

通知グループ化オプションは、CQ_NOTIFICATION\$_REG_INFOオブジェクトの属性であり、次の種類があります。

属性	説明:
NTFN_GROUPING_CLASS	通知をグループ化するクラスを指定します。使用可能な値は、DBMS_CQ_NOTIFICATION.NTFN_GROUPING_CLASS_TIME(通知を時間でグループ化)と、デフォルトのゼロ(通知をその原因となるイベント

属性	説明:
	の直後に生成)のみです。
NTFN_GROUPING_VALUE	グループを定義する時間間隔を秒単位で指定します。たとえば、この値が 900 の場合、同じ 15 分間隔内に生成された通知がグループ化されます。
NTFN_GROUPING_TYPE	次のいずれかのグループ化のタイプを指定します。 ● DBMS_CQ_NOTIFICATION.NTFN_GROUPING_TYPE_SUMMARY: グループ内のすべての通知は、単一の通知に要約されます。 ノート: ROWID オプションを指定した場合でも、この単一の通知には ROWID は含まれません。 ● DBMS_CQ_NOTIFICATION.NTFN_GROUPING_TYPE_LAST: グループ内の最後の通知のみが公開され、それより前の通知は破棄されます。
NTFN_GROUPING_START_TIME	通知生成の開始時間を指定します。NULL が指定されると、システム生成の現在の時間がデフォルトになります。
NTFN_GROUPING_REPEAT_COUNT	通知を繰り返す回数を指定します。登録の存続中に通知を永続的に受信するには、 DBMS_CQ_NOTIFICATION.NTFN_GROUPING_FOREVER に設定します。登録の存続中に最大で n の通知を受信するには、n に設定します。

ノート:



タイムアウト、権限の消失およびグローバル・イベントによって生成された通知は、指定したグループ化間隔が終了する前に公開される場合があります。この場合、グループ化された保留中の通知が存在すると、間隔が終了する前にその通知も公開されます。

18.7.2.7 信頼可能オプション

デフォルトでは、CQN登録は共有メモリーに格納されます。これをかわりに永続データベース・キューに格納するには(つまり、信頼できる通知を生成するには)、CQ_NOTIFICATION\$_REG_INFOオブジェクトのQOSFLAGS属性にQOS_RELIABLEを指定します。

信頼できる通知のメリットは、その生成後にデータベースで障害が発生しても、再起動後にその通知を配信できることです。Oracle RAC環境では、正常なデータベース・インスタンスがこの通知を配信できます。

信頼できる通知のデメリットは、デフォルトの通知よりもCPUおよびI/Oのコストが高いことです。

18.7.2.8 「通知時にパージ」および「タイムアウト」オプション

デフォルトでは、CQN登録は、それを作成したアプリケーションによって明示的に登録解除されるまで、または(権限の消失などが原因で)データベースによって暗黙的にパージされるまで存続します。

登録の最初の通知の生成後にその登録をパージするには、CQ_NOTIFICATION\$_REG_INFOオブジェクトのQOSFLAGS属性にQOS_DEREG_NFYを指定します。

n秒後に登録をパージするには、CQ_NOTIFICATION\$_REG_INFOオブジェクトのTIMEOUT属性にnを指定します。

「通知時にパージ」と「タイムアウト」の2つのオプションは、同時に使用できます。

18.7.3 CQN登録を作成するための前提条件

CQN登録を作成するための前提条件は次のとおりです。

- 次の権限が必要です。
 - DBMS_CQ_NOTIFICATIONパッケージのEXECUTE権限(登録の作成にこのパッケージのサブプログラムを使用)
 - CHANGE NOTIFICATIONシステム権限
 - 登録する各オブジェクトでのREADまたはSELECT権限

登録済の問合せに関連付けられているオブジェクトに対する権限を失うと、通知が生成されます。

- SYS以外のユーザーとして接続する必要があります。
- コミットされていないトランザクションの途中でない必要があります。
- dml_locks_init.oraパラメータの値がゼロ以外である必要があります(デフォルト値はゼロ以外です)。

(これは、通知を受信するための前提条件でもあります。)

ノート:



QRCN を使用するには、データベースの COMPATIBLE 設定が 11.0.0 以上である必要があります。

関連項目:

[登録解除](#)

18.7.4 オブジェクト変更通知(OCN)に対して登録可能な問合せ

ストアド・プロシージャやREFカーソルの一部として実行される問合せなど、ほとんどの問合せをOCNに対して登録できます。

OCNに対して登録不可能な問合せは次のとおりです。

- 固定表または固定ビューに対する問合せ
- ユーザー・ビューに対する問合せ

- データベース・リンク(dblink)を含む問合せ
- マテリアライズド・ビューでの問合せ

ノート:



OCN 登録ではシノニムを使用できますが、QRCN 登録では使用できません。

18.7.5 問合せ結果変更通知(QRCN)に対して登録可能な問合せ

QRCNに対して保証モードで登録可能な問合せ、QRCNに対してベストエフォート・モードでのみ登録可能な問合せ、およびQRCNに対してどちらのモードでも登録不可能な問合せがあります。

トピック:

- [保証モードでQRCNに対して登録可能な問合せ](#)
- [ベストエフォート・モードでのみQRCNに対して登録可能な問合せ](#)
- [どちらのモードでもQRCNに対して登録不可能な問合せ](#)

関連項目:

- [保証モード](#)および
- [ベストエフォート・モード](#)

18.7.5.1 保証モードでQRCNに対して登録可能な問合せ

問合せを保証モードでQRCNに対して登録するには、問合せを次のルールに準拠させる必要があります。

- 参照するすべての列がNUMBERまたはVARCHAR2データ型である必要があります。
- 列の式に含まれる算術演算子が次のバイナリ演算子に制限され、そのオペランドが数値データ型の列である必要があります。
 - + (加算)
 - - (減算、単項マイナスではない)
 - * (乗算)
 - / (除算)
- 条件句の比較演算子は次の演算子に制限されます。
 - < (未満)
 - <=(以下)
 - = (等しい)
 - >=(以上)
 - > (より大きい)
 - <>または!= (等しくない)

- IS NULL
- IS NOT NULL
- 条件句のBoolean演算子はAND、ORおよびNOTに制限されます。
- 問合せに集計関数(SUM、COUNT、AVERAGE、MIN、MAXなど)は含められません。

保証モードでは、単一表および一部の内部等価結合に対するほとんどの問合せが可能です(次の例を参照)。

```
SELECT SALARY FROM EMPLOYEES, DEPARTMENTS
WHERE EMPLOYEES.DEPARTMENT_ID = DEPARTMENTS.DEPARTMENT_ID
AND DEPARTMENTS.LOCATION_ID = 1700;
```

ノート:



- 問合せオプティマイザで使用する実行計画により、問合せが保証モードでは矛盾する場合があります (OR 拡張など)。
- 保証モードで登録可能な問合せは、ベストエフォート・モードでも登録可能ですが、ベストエフォート・モードでは CQN で簡略化されない問合せに対しても誤検出が発生する可能性があるため、結果は異なる場合があります。

関連項目:

- SQL集計関数のリストは、[『Oracle Database SQL言語リファレンス』](#)を参照してください。
- 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。
- [ベストエフォート・モード](#)

18.7.5.2 ベストエフォート・モードでのみQRCNに対して登録可能な問合せ

次のいずれかの条件を満たす問合せは、ベストエフォート・モードでのみQRCNに対して登録可能であり、その簡略バージョンでは通知がオブジェクトの粒度で生成されます。

- 暗号化が有効にされている列を参照する問合せ
- 同じ型の項目がSELECT構文のリストに11以上ある問合せ
- 次のいずれかを使用した式を含む問合せ
 - 文字列関数(SUBSTR、LTRIM、RTRIMなど)
 - 算術関数(TRUNC、ABS、SQRTなど)
 - パターン一致条件LIKEおよびREGEXP_LIKE
 - EXISTSまたはNOT EXISTS条件
- 別の表の列で定義されている条件句を使用した論理和を含む問合せ。たとえば:

```
SELECT EMPLOYEE_ID, DEPARTMENT_ID
FROM EMPLOYEES, DEPARTMENTS
WHERE EMPLOYEES.EMPLOYEE_ID = 10
OR DEPARTMENTS.DEPARTMENT_ID = 'IT';
```

- ユーザーROWIDアクセスを含む問合せ。たとえば:

```
SELECT DEPARTMENT_ID
FROM DEPARTMENTS
WHERE ROWID = 'AAANKdAABAAALnAAF';
```

- 内部結合以外の結合を含む問合せ
- 次のいずれかを使用した実行計画を含む問合せ
 - ビットマップ結合索引、ドメイン索引またはファンクション索引
 - UNION ALLまたはCONCATENATION
(問合せ自体の内部、または問合せオプティマイザで選択されたOR拡張実行計画の結果内で使用。)
 - ORDER BYまたはGROUP BY
(問合せ自体の内部、または問合せオプティマイザで選択された実行計画内のORDER BYオプションを使用したSORT操作の結果内で使用。)
 - オーバーフロー・セグメントがあり、パーティション化された索引構成表(IOT)
 - クラスタ・オブジェクト
 - パラレル実行

関連項目:

SQL関数のリストは、[『Oracle Database SQL言語リファレンス』](#)を参照してください。

18.7.5.3 どちらのモードでもQRCNに対して登録不可能な問合せ

次のいずれかを参照する問合せは、保証モードとベストエフォート・モードのどちらでもQRCNに対して登録不可能です。

- ビュー
- 固定表、リモート表または仮想プライベート・データベース(VPD)ポリシーが有効化されている表
- DUAL(SELECT構文のリスト内)
- シノニム
- ユーザー定義のPL/SQLサブプログラムのコール
- [『保証モードでQRCNに対して登録可能な問合せ』](#)にリストされていない演算子
- 集計関数COUNT
(他の集計関数は、ベストエフォート・モードでは登録可能ですが、保証モードでは登録不可能です。)
- アプリケーション・コンテキスト。たとえば:

```
SELECT SALARY FROM EMPLOYEES
WHERE USER = SYS_CONTEXT('USERENV', 'SESSION_USER');
```

- SYSDATE、SYSTIMESTAMPまたはCURRENT_TIMESTAMP

また、マテリアライズド・ビューを使用して問合せオプティマイザがリライトした問合せはQRCNに対して登録できません。

関連項目:

問合せオブ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。

18.7.6 PL/SQLを介したCQNに対する問合せの登録

PL/SQLを使用してCQN登録を作成するステップは、次のとおりです。

1. 通知ハンドラとして機能するストアードPL/SQLプロシージャを作成します。
[『PL/SQL通知ハンドラの作成』](#)を参照してください。
2. 通知ハンドラの名前、通知タイプ、および登録のその他の属性を指定するCQ_NOTIFICATION\$_REG_INFOオブジェクトを作成します。
[『CQ_NOTIFICATION\\$_REG_INFOオブジェクトの作成』](#)を参照してください。
3. クライアント・アプリケーションで、DBMS_CQ_NOTIFICATION.NEW_REG_STARTファンクションを使用して登録ブロックをオープンします。

CQ_NOTIFICATION\$_REG_INFOオブジェクトとファンクションNEW_REG_STARTおよびREG_END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。それらすべてはDBMS_CQ_NOTIFICATIONパッケージで定義されています。

4. 登録する問合せを実行します(DML操作またはDDL操作は実行しないでください)。
詳細は次のトピックを参照してください。
 - [通知内の各問合せの識別](#)
 - [既存の登録への問合せの追加](#)

5. DBMS_CQ_NOTIFICATION.REG_ENDファンクションを使用して登録ブロックをクローズします。

18.7.6.1 PL/SQL通知ハンドラの作成

通知ハンドラとして使用するPL/SQLストアード・プロシージャの作成時には、次のシグネチャを含める必要があります。

```
PROCEDURE schema_name.proc_name(ntfnds IN CQ_NOTIFICATION$_DESCRIPTOR)
```

このシグネチャで、schema_nameはデータベース・スキーマ名、proc_nameはストアード・プロシージャ名、ntfndsは通知記述子を示します。

通知記述子はCQ_NOTIFICATION\$_DESCRIPTORオブジェクトであり、その属性によって変更の詳細(トランザクションID、変更のタイプ、影響を受けた問合せ、変更された表など)が記述されます。

通知記述子ntfndsはJOBQプロセスから通知ハンドラproc_nameに渡され、このハンドラがアプリケーション要件に従って通知を処理します。(これは[図18-2](#)のステップ6に示します。)

ノート:



通知ハンドラは、ジョブ・キュー・プロセス内で実行されます。JOB_QUEUE_PROCESSES 初期化パラメータは、ジョブの実行用に作成できる最大プロセス数を指定します。PL/SQL 通知を受信するには、JOB_QUEUE_PROCESSES をゼロ以外の値に設定する必要があります。

関連項目:

18.7.6.2 CQ_NOTIFICATION\$_REG_INFOオブジェクトの作成

タイプCQ_NOTIFICATION\$_REG_INFOのオブジェクトでは、登録済オブジェクトの変更時にデータベースで実行される通知ハンドラを指定します。SQL*Plusでは、次の文を実行してそのタイプ属性を表示できます。

```
DESC CQ_NOTIFICATION$_REG_INFO
```

[表18-2](#)に、SYS.CQ_NOTIFICATION\$_REG_INFOの属性を示します。

表18-2 CQ_NOTIFICATION\$_REG_INFOの属性

属性	説明:
CALLBACK	通知の生成時に実行される PL/SQL プロシージャ(通知ハンドラ)の名前を指定します。この名前は、hr.dcn_callback のように schema_name.procedure_name 形式で指定する必要があります。
QOSFLAGS	1 つ以上のサービスのクオリティ・フラグ(DBMS_CQ_NOTIFICATION パッケージ内の定数)を指定します。この名前と説明は、表 18-3 を参照してください。 複数のサービスのクオリティ・フラグを指定するには、ビット単位の OR を使用します。たとえば、DBMS_CQ_NOTIFICATION.QOS_RELIABLE + DBMS_CQ_NOTIFICATION.QOS_ROWIDS などです。
TIMEOUT	登録のタイムアウト期間を指定します。0(ゼロ)以外の値に設定する場合は、データベースにより登録がパージされるまでの秒数を指定します。0 または NULL の場合、登録はクライアントにより明示的に解除されるまで存続します。 QOSFLAGS 属性およびその QOS_DEREG_NFY フラグと併用できます。
OPERATIONS_FILTER	OCN(「オブジェクト変更通知(OCN)について」を参照)にのみ適用されます。QOS_FLAGS 属性に QOS_QUERY フラグが指定されている場合は無効です。 SQL 文の型に基づいてメッセージをフィルタリングします。 DBMS_CQ_NOTIFICATION パッケージ内の次の定数を指定できます。 ● ALL_OPERATIONS。すべての変更時に通知します。 ● INSERTOP。INSERT 時に通知します。 ● UPDATEOP。UPDATE 時に通知します。

属性	説明:
	● DELETEOP。DELETE 時に通知します。 ● ALTEROP。ALTER TABLE 操作時に通知します。 ● DROPPOP。DROP TABLE 操作時に通知します。 ● UNKNOWNOP。不明な操作時に通知します。 ビット単位の OR を使用すると、操作の組合せを指定できます。たとえば、 DBMS_CQ_NOTIFICATION.INSERTOP + DBMS_CQ_NOTIFICATION.DELETEOP などです。
TRANSACTION_LAG	非推奨。フロー制御の通知を実装するには、NTFN_GROUPING_* 属性を使用してください。 OCN(「オブジェクト変更通知(OCN)について」を参照)にのみ適用されます。QOS_FLAGS 属性に QOS_QUERY フラグが指定されている場合は無効です。 クライアントがデータベースから遅れることのできるトランザクション数またはデータベース変更数を指定します。0 の場合、クライアントは無効化メッセージを生成直後に受信します。5 の場合、登録済オブジェクトを変更するトランザクションのうち 5 番目ごとに通知が送信されます。データベースでは、中間の変更がオブジェクトの粒度で追跡され、変更が通知とともにバンドルされます。そのため、クライアントから中間の変更が失われることはありません。 トランザクションのコミット時にオブジェクトの変更通知をそれ以上の遅延なしで受信する必要のあるほとんどのアプリケーションでは、トランザクション・ラグ 0(ゼロ)を選択するように期待されます。ゼロ以外のトランザクション・ラグが役立つのは、アプリケーションで通知に対するフロー制御を実装している場合のみです。ゼロ以外のトランザクション・ラグを使用する場合は、アプリケーションのワークロードのプロパティを、通知が妥当な頻度で生成されるように設定することをお勧めします。そうしないと、ラグに達するまで通知が無限に遅延する可能性があります。 TRANSACTION_LAG を指定すると、登録時に QOS_ROWIDS を指定した場合にも、通知メッセージには ROWID レベルの粒度を使用できなくなります。
NTFN_GROUPING_CLASS	通知をグループ化するクラスを指定します。使用可能な値は DBMS_CQ_NOTIFICATION.NTFN_GROUPING_CLASS_TIME(通知を時間でグループ化)のみです。

属性	説明:
NTFN_GROUPING_VALUE	グループを定義する時間間隔を秒単位で指定します。たとえば、この値が 900 の場合、同じ 15 分間隔内に生成された通知がグループ化されます。
NTFN_GROUPING_TYPE	次のいずれかのグループ化のタイプを指定します。 ● DBMS_CQ_NOTIFICATION.NTFN_GROUPING_TYPE_SUMMARY: グループ内のすべての通知は、単一の通知に要約されます。 ● DBMS_CQ_NOTIFICATION.NTFN_GROUPING_TYPE_LAST: グループ内の最後の通知のみが公開され、それより前の通知は破棄されます。
NTFN_GROUPING_START_TIME	通知生成の開始時間を指定します。NULL が指定されると、システム生成の現在の時間がデフォルトになります。
NTFN_GROUPING_REPEAT_COUNT	通知を繰り返す回数を指定します。登録の存続中に通知を永続的に受信するには、DBMS_CQ_NOTIFICATION.NTFN_GROUPING_FOREVER に設定します。登録の存続中に最大で n の通知を受信するには、n に設定します。

表18-3のサービスのクオリティ・フラグは、DBMS_CQ_NOTIFICATIONパッケージ内の定数です。このフラグは、CQ_NOTIFICATION\$_REG_INFOのQOS_FLAGS属性を使用して指定できます([表18-2](#)を参照)。

表18-3 サービスのクオリティ・フラグ

フラグ	説明:
QOS_DEREG_NFY	最初の通知後に登録をページします。
QOS_RELIABLE	通知を永続データベース・キューに格納します。 Oracle RAC 環境では、データベース・インスタンスで障害が発生すると、正常なデータベース・インスタンスがキューにある通知メッセージを配信できます。 デフォルト: 通知はより効率のよい共有メモリーに格納されます。
QOS_ROWIDS	変更された各行の ROWID を通知に含めます。
QOS_QUERY	QRCN に対する問合せを登録します(「問合せ結果変更通知(QRCN)について」 を参照)。 QRCN に対する問合せを登録できない場合は、QOS_BEST_EFFORT も指定していないかぎり、登録時にエラーが生成されます。

フラグ	説明:
	デフォルト: OCN に対する問合せが登録されます(「オブジェクト変更通知(OCN)について」 を参照)
QOS_BEST_EFFORT	QOS_QUERY と併用されます。問合せ結果変更の評価には複雑すぎる問合せの簡略バージョンを登録します。つまり、QRCN に対する問合せをベストエフォート・モードで登録します(「ベストエフォート・モード」 を参照)。 どの問合せが簡略化されているかを確認するには、静的データ・ディクショナリ・ビュー DBA_CQ_NOTIFICATION_QUERIES または USER_CQ_NOTIFICATION_QUERIES を問い合わせます。これらのビューでは、登録済の各問合せの QUERYID とテキストが表示されます。 デフォルト: QRCN に対して問合せが保証モードで登録されます(「保証モード」 を参照)

登録済オブジェクトに変更があるたびに、プロシージャHR.dcn_callbackを起動する必要があるとします。[例18-4](#)では、HR.dcn_callbackが通知を受け取るように指定するCQ_NOTIFICATION\$_REG_INFOオブジェクトを作成します。このオブジェクトを作成するには、DBMS_CQ_NOTIFICATIONパッケージに対するEXECUTE権限が必要です。

例18-4 CQ_NOTIFICATION\$_REG_INFOオブジェクトの作成

```
DECLARE
  v_cn_addr CQ_NOTIFICATION$_REG_INFO;
BEGIN
  -- Create object:
  v_cn_addr := CQ_NOTIFICATION$_REG_INFO (
    'HR.dcn_callback',          -- PL/SQL notification handler
    DBMS_CQ_NOTIFICATION.QOS_QUERY -- notification type QRCN
  + DBMS_CQ_NOTIFICATION.QOS_ROWIDS, -- include rowids of changed objects
    0,                          -- registration persists until unregistered
    0,                          -- notify on all operations
    0                           -- notify immediately
  );
  -- Register queries: ...
END;
/
```

18.7.6.3 通知内の各問合せの識別

問合せの登録済リストに含まれる問合せにより、連続問合せ通知が生成される場合があります。特定の問合せによっていつ通知が生成されるかを確認するには、その問合せのSELECT構文のリストで

DBMS_CQ_NOTIFICATION.CQ_NOTIFICATION_QUERYIDファンクションを使用します。たとえば:

```
SELECT EMPLOYEE_ID, SALARY, DBMS_CQ_NOTIFICATION.CQ_NOTIFICATION_QUERYID
FROM EMPLOYEES
WHERE DEPARTMENT_ID = 10;
```

結果:

EMPLOYEE_ID	SALARY	CQ_NOTIFICATION_QUERYID
200	2800	0

1 row selected.

問合せによって通知が生成される場合、その通知には問合せIDが含まれます。

18.7.6.4 既存の登録への問合せの追加

問合せを既存の登録に追加するステップは、次のとおりです。

1. 既存の登録の登録IDを取得します。

これは、保存済の出力または*_CHANGE_NOTIFICATION_REGSの問合せから取得できます。

2. 登録IDをパラメータとして使用し、プロシージャDBMS_CQ_NOTIFICATION.ENABLE_REGをコールして既存の登録をオープンします。
3. 登録する問合せを実行します(DML操作またはDDL操作は実行しないでください)。
4. DBMS_CQ_NOTIFICATION.REG_ENDファンクションを使用して登録をクローズします。

[例18-5](#)では、登録IDが21の既存の登録に問合せを追加します。

例18-5 既存の登録への問合せの追加

```
DECLARE
  v_cursor SYS_REFCURSOR;
BEGIN
  -- Open existing registration
  DBMS_CQ_NOTIFICATION.ENABLE_REG(21);
  OPEN v_cursor FOR
    -- Run query to be registered
    SELECT DEPARTMENT_ID
      FROM HR.DEPARTMENTS; -- register this query
  CLOSE v_cursor;
  -- Close registration
  DBMS_CQ_NOTIFICATION.REG_END;
END;
/
```

18.7.7 CQN登録のベスト・プラクティス

CQNに最適なパフォーマンスを得るには、次の登録ガイドラインに従ってください。

- 登録する問合せの数を少なくします。可能であれば、ほとんど変更されないオブジェクトを参照する問合せを登録します。
オブジェクトの揮発性が非常に高いと、多数の通知が送信されることになり、そのオーバーヘッドによってOLTPスループットが低下します。
- 同じ通知メッセージが複数の受信者宛に複製されないように、特定のオブジェクトについて重複登録の数を最小限に抑えます。

18.7.8 CQN登録のトラブルシューティング

登録を作成できない場合、または登録を作成しても予期した通知を受信していない場合は、次のような原因が考えられます。

- JOB_QUEUE_PROCESSESパラメータがゼロ以外の値に設定されていません。
このため、通知ハンドラを介してPL/SQL通知を受信できません。
- 登録の作成時にSYSユーザーとして接続しました。

CQN登録を作成するには、SYS以外のユーザーとして接続する必要があります。

- 登録済オブジェクトを変更した後で、そのトランザクションをコミットしていません。

通知は、トランザクションのコミット時にのみ生成されます。

- 登録がデータベースに正常に作成されていません。

確認するには、静的データ・ディクショナリ・ビュー *_CHANGE_NOTIFICATION_REGSを問い合わせます。たとえば、次の文により、現行ユーザーに関するすべての登録と登録済オブジェクトが表示されます。

```
SELECT REGID, TABLE_NAME FROM USER_CHANGE_NOTIFICATION_REGS;
```

- 通知ハンドラの実行中に実行時エラーが発生しました。

その場合は、プロシージャの実行を試行したJOBQプロセスのトレース・ファイルに記録されます。通常、トレース・ファイルの名前は次の形式になります。

```
ORACLE_SID_jnumber_PID.trc
```

たとえば、ORACLE_SIDがdbs1で、JOBQプロセスのプロセスID(PID)が12483の場合、トレース・ファイルの名前は通常、dbs1_j000_12483.trcとなります。

通知ハンドラに'chnf_callback'、登録IDに100を指定して登録が作成されているとします。ここで、'chnf_callback'はデータベースに定義されていないとします。この場合、JOBQトレース・ファイルには次の書式のメッセージが含まれます。

```
*****
Runtime error during execution of PL/SQL cbk chnf_callback for reg CHNF100.
Error in PLSQL notification of msgid:
Queue :
Consumer Name :
PLSQL function :chnf_callback
Exception Occured, Error msg:
ORA-00604: error occurred at recursive SQL level 2
ORA-06550: line 1, column 7:
PLS-00201: identifier 'CHNF_CALLBACK' must be declared
ORA-06550: line 1, column 7:
PL/SQL: Statement ignored
*****
```

通知ハンドラの実行中に実行時エラーが発生した場合は、通知を受信していることを確認するために、簡略バージョンの通知ハンドラを作成してから、アプリケーション・ロジックを徐々に追加してください。

非常に単純な通知ハンドラの例は次のとおりです。

```
REM Create table in HR schema to hold count of notifications received.
CREATE TABLE nfcount(cnt NUMBER);
INSERT INTO nfcount (cnt) VALUES(0);
COMMIT;
CREATE OR REPLACE PROCEDURE chnf_callback
(ntfnds IN CQ_NOTIFICATION$_DESCRIPTOR)
IS
BEGIN
    UPDATE nfcount SET cnt = cnt+1;
    COMMIT;
END;
/
```

- トランザクションがコミットしてからエンド・ユーザーが通知を受信するまでに、タイム・ラグが発生しています。

18.7.9 登録の削除

登録を削除するには、登録IDをパラメータとして使用してプロシージャDBMS_CQ_NOTIFICATION.DEREGISTERをコールします。たとえば、次の文により、登録IDが21の登録が解除されます。

```
DBMS_CQ_NOTIFICATION.DEREGISTER(21);
```

登録を作成したユーザーまたはSYSユーザーのみが登録を解除できます。

18.7.10 CQNの構成：使用例

この使用例では、開発者として従業員データ(名前、所在地、電話番号など)を提供するWebアプリケーションを管理しているとします。このアプリケーションはOracle Application Serverで実行され、使用量が多く、バックエンド・データベースにあるHR.EMPLOYEES表およびHR.DEPARTMENTS表の頻繁な問合せを処理します。この2つの表はあまり変更されないため、アプリケーションでは問合せ結果をキャッシュすることでパフォーマンスを改善できます。キャッシュにより、バックエンド・データベースへのラウンド・トリップ、およびサーバー側での実行待機時間が回避されます。

DBMS_CQ_NOTIFICATIONパッケージを使用すると、HR.EMPLOYEES表およびHR.DEPARTMENTS表に基づく問合せを登録できます。CQNを構成するステップは、次のとおりです。

1. [「PL/SQL通知ハンドラの作成」](#)の説明に従って、通知を処理するサーバー側PL/SQLストアド・プロシージャを作成します。
2. [「問合せの登録」](#)の説明に従って、HR.EMPLOYEES表およびHR.DEPARTMENTS表へのQRCNに対する問合せを登録します。

これらのステップの完了後は、ステップ2で登録した問合せの結果に対する変更がコミットされると、ステップ1で作成した通知ハンドラによってWebアプリケーションに変更が通知され、この直後に、バックエンド・データベースの問合せによりWebアプリケーションでキャッシュがリフレッシュされます。

18.7.10.1 PL/SQL通知ハンドラの作成

次の説明に従って、通知を処理するサーバー側ストアドPL/SQLプロシージャを作成します。

1. AS SYSDBAでデータベースに接続します。
2. 必要な権限をHRに付与します。

```
GRANT EXECUTE ON DBMS_CQ_NOTIFICATION TO HR;
GRANT CHANGE NOTIFICATION TO HR;
```
3. 通知を受信するためにJOB_QUEUE_PROCESSESパラメータを有効化します。

```
ALTER SYSTEM SET "JOB_QUEUE_PROCESSES"=4;
```
4. SYS以外のユーザー(HRなど)としてデータベースに接続します。
5. 受信した通知イベントのレコードを保持するデータベース表を作成します。

```
-- Create table to record notification events.
DROP TABLE nfevents;
CREATE TABLE nfevents (
    regid      NUMBER,
    event_type NUMBER
);
-- Create table to record notification queries:
DROP TABLE nfqueries;
CREATE TABLE nfqueries (
    qid NUMBER,
```



```

    qop NUMBER
);
-- Create table to record changes to registered tables:
DROP TABLE nftablechanges;
CREATE TABLE nftablechanges (
    qid          NUMBER,
    table_name   VARCHAR2(100),
    table_operation NUMBER
);
-- Create table to record ROWIDs of changed rows:
DROP TABLE nfrowchanges;
CREATE TABLE nfrowchanges (
    qid          NUMBER,
    table_name   VARCHAR2(100),
    row_id       VARCHAR2(2000)
);

```

6. [例18-6](#)に示すように、プロシージャHR.chnf_callbackを作成します。

例18-6 サーバー側PL/SQL通知ハンドラの作成

```

CREATE OR REPLACE PROCEDURE chnf_callback (
    ntfnds IN CQ_NOTIFICATION$_DESCRIPTOR
)
IS
    regid          NUMBER;
    tbname         VARCHAR2(60);
    event_type     NUMBER;
    numtables      NUMBER;
    operation_type  NUMBER;
    numrows        NUMBER;
    row_id         VARCHAR2(2000);
    numqueries     NUMBER;
    qid            NUMBER;
    qop            NUMBER;
BEGIN
    regid := ntfnds.registration_id;
    event_type := ntfnds.event_type;
    INSERT INTO nfevents (regid, event_type)
    VALUES (chnf_callback.regid, chnf_callback.event_type);
    numqueries := 0;
    IF (event_type = DBMS_CQ_NOTIFICATION.EVENT_QUERYCHANGE) THEN
        numqueries := ntfnds.query_desc_array.count;
        FOR i IN 1..numqueries LOOP -- loop over queries
            qid := ntfnds.query_desc_array(i).queryid;
            qop := ntfnds.query_desc_array(i).queryop;
            INSERT INTO nfqueries (qid, qop)
            VALUES(chnf_callback.qid, chnf_callback.qop);
            numtables := 0;
            numtables := ntfnds.query_desc_array(i).table_desc_array.count;
            FOR j IN 1..numtables LOOP -- loop over tables
                tbname :=
                    ntfnds.query_desc_array(i).table_desc_array(j).table_name;
                operation_type :=
                    ntfnds.query_desc_array(i).table_desc_array(j).Opflags;
                INSERT INTO nftablechanges (qid, table_name, table_operation)
                VALUES (
                    chnf_callback.qid,
                    tbname,
                    operation_type
                );
            END LOOP;
            IF (bitand(operation_type, DBMS_CQ_NOTIFICATION.ALL_ROWS) = 0) THEN
                numrows := ntfnds.query_desc_array(i).table_desc_array(j).numrows;
            ELSE
                numrows := 0; -- ROWID info not available
            END IF;
        END LOOP;
    END IF;

```



```

        -- Body of loop does not run when numrows is zero.
        FOR k IN 1..numrows LOOP -- loop over rows
            Row_id :=
ntfnds.query_desc_array(i).table_desc_array(j).row_desc_array(k).row_id;
            INSERT INTO nfrowchanges (qid, table_name, row_id)
            VALUES (chnf_callback.qid, tbname, chnf_callback.Row_id);
        END LOOP; -- loop over rows
    END LOOP; -- loop over tables
END LOOP; -- loop over queries
END IF;
COMMIT;
END;
/

```

18.7.10.2 問合せの登録

通知ハンドラの作成後に、[例18-7](#)に示すように、HR.chnf_callbackを通知ハンドラとして指定して、通知を受信する対象となる問合せを登録します。

例18-7 問合せの登録

```

DECLARE
    reginfo    CQ_NOTIFICATION$_REG_INFO;
    mgr_id     NUMBER;
    dept_id    NUMBER;
    v_cursor   SYS_REFCURSOR;
    regid      NUMBER;
BEGIN
    /* Register two queries for QRNC: */
    /* 1. Construct registration information.
       chnf_callback is name of notification handler.
       QOS_QUERY specifies result-set-change notifications. */
    reginfo := cq_notification$_reg_info (
        'chnf_callback',
        DBMS_CQ_NOTIFICATION.QOS_QUERY,
        0, 0, 0
    );
    /* 2. Create registration. */
    regid := DBMS_CQ_NOTIFICATION.new_reg_start(reginfo);
    OPEN v_cursor FOR
        SELECT dbms_cq_notification.CQ_NOTIFICATION_QUERYID, manager_id
        FROM HR.EMPLOYEES
        WHERE employee_id = 7902;
    CLOSE v_cursor;
    OPEN v_cursor FOR
        SELECT dbms_cq_notification.CQ_NOTIFICATION_QUERYID, department_id
        FROM HR.departments
        WHERE department_name = 'IT';
    CLOSE v_cursor;
    DBMS_CQ_NOTIFICATION.reg_end;
END;
/

```

新しく作成した登録を表示:

```

SELECT queryid, regid, TO_CHAR(querytext)
FROM user_cq_notification_queries;

```

結果は次のようになります。

QUERYID	REGID	TO_CHAR(QUERYTEXT)
22	41	SELECT HR.DEPARTMENTS.DEPARTMENT_ID FROM HR.DEPARTMENTS WHERE HR.DEPARTMENTS.DEPARTMENT_NAME = 'IT'

```

21      41 SELECT HR.EMPLOYEES.MANAGER_ID
        FROM HR.EMPLOYEES
        WHERE HR.EMPLOYEES.EMPLOYEE_ID = 7902

```

次のトランザクションを実行すると、QUERYIDが22の問合せの結果が変更されます。

```

UPDATE DEPARTMENTS
SET DEPARTMENT_NAME = 'FINANCE'
WHERE department_name = 'IT';

```

通知プロシージャchnf_callback([例18-6](#)で作成したプロシージャ)が実行されます。

通知イベントが記録される表を問合せ:

```
SELECT * FROM nfevents;
```

結果は次のようになります。

REGID	EVENT_TYPE
61	7

EVENT_TYPE 7は、EVENT_QUERYCHANGE(問合せ結果変更)に対応します。

登録済の表への変更が記録される表を問い合わせます。

```
SELECT * FROM nftablechanges;
```

結果は次のようになります。

REGID	TABLE_NAME	TABLE_OPERATION
42	HR.DEPARTMENTS	4

TABLE_OPERATION 4は、UPDATEOP(更新操作)に対応します。

変更された行のROWIDが記録される表を問い合わせます。

```
SELECT * FROM nfrowchanges;
```

結果は次のようになります。

REGID	TABLE_NAME	ROWID
61	HR.DEPARTMENTS	AAANKdAABAAALinAAF

18.8 OCIを使用したCQN登録の作成

この項では、OCIを使用したCQN登録の作成について説明します。OCIを使用する場合、通知ハンドラはクライアント側コールバック・プロシージャです。

トピック

- [問合せ結果セット通知でのOCI使用](#)
- [連続問合せ通知の登録のためのOCI使用](#)
- [連続問合せ通知のOCIサブスクリプション・ハンドル属性の使用](#)
- [クライアントが開始したCQN登録でのOCIの使用](#)

- [OCI_ATTR_CQ_QUERYID属性](#)
- [OCI連続問合せ通知の記述子の使用](#)
- [OCIサンプル・プログラムでの連続問合せ通知のデモンストレーション](#)

関連項目:

[OCIのパブリッシュ・サブスクライブ通知](#)の詳細は、『Oracle Call Interfaceプログラマーズ・ガイド』を参照してください

18.8.1 問合せ結果セット通知でのOCI使用

連続問合せ(CQ)通知固有のQOS(サービス品質フラグ)を記録するには、サブスクリプション・ハンドルOCI_HTYPE_SUBSCRに属性OCI_ATTR_SUBSCR_CQ_QOSFLAGSを設定します。オブジェクトの細分化レベルではなく問合せの細分化レベルで登録するようにリクエストするには、属性OCI_ATTR_SUBSCR_CQ_QOSFLAGSにOCI_SUBSCR_CQ_QOS_QUERYフラグ・ビットを設定します。

オプションで擬似列CQ_NOTIFICATION_QUERY_IDを指定して、登録済問合せの問合せIDを取得できます。この場合、細分化レベルは自動的に問合せレベルに変換されません。戻り時の擬似列の値は、問合せに割り当てられた一意の問合せIDに設定されます。OCIベースの登録の場合は問合せIDの擬似列を省略できますが、その場合は問合せIDが文ハンドルのREAD属性として送信されます。(この属性はOCI_ATTR_CQ_QUERYIDです)。

通知中に、クライアントが指定したコールバックが起動され、トップレベルの通知記述子が引数として渡されます。

変更された問合せの問合せIDに関する情報は、OCI_DTYPE_CQDESという特別な記述型を使用して送られます。問合せ記述子の集合(OCIColl)は、トップレベルの通知記述子内に埋め込まれます。各記述子はOCI_DTYPE_CQDES型です。この問合せ記述子には、次の属性が含まれています。

- OCI_ATTR_CQDES_OPERATION - OCI_EVENT_QUERYCHANGEまたはOCI_EVENT_DEREGです。
- OCI_ATTR_CQDES_QUERYID - 変更された問合せの問合せIDです。
- OCI_ATTR_CQDES_TABLE_CHANGES - 問合せ結果セットが変更される原因となった表に対するDML操作を記述する表記述子の配列です。各表記述子はOCI_DTYPE_TABLE_CHDES型です。

関連項目:

[OCI_DTYPE_CHDES](#)

18.8.2 連続問合せ通知の登録のためのOCI使用

コール側セッションには、登録対象のすべてのオブジェクトに対するCHANGE NOTIFICATIONシステム権限およびSELECT権限が必要です。登録はデータベースに記録される永続エンティティであり、Oracle RACのすべてのインスタンスに表示されます。登録が問合せ精度の場合、Oracle RACインスタンスで問合せ結果セットが変更され、コミットされる原因となるトランザクションによって通知が生成されます。登録がオブジェクト精度の場合、Oracle RACインスタンスに登録済のオブジェクトを変更するトランザクションによって通知が生成されます。

マテリアライズド・ビューまたは非マテリアライズド・ビューが関係する問合せは、サポートされません。

登録インタフェースは、コールバック、およびAQのネームスペース拡張(DBCHANGE)を使用して、問合せ対象オブジェクトの変更を通知します。

登録を作成するステップは、次のとおりです。

1. OCI_EVENTSおよびOCI_OBJECTモードで環境を作成します。
2. サブスクリプション・ハンドル属性OCI_ATTR_SUBSCR_NAMESPACEをネームスペースOCI_SUBSCR_NAMESPACE_DBCHANGEに設定します。
3. サブスクリプション・ハンドル属性OCI_ATTR_SUBSCR_CALLBACKを、問合せハンドルに関連したOCIコールバックを格納するように設定します。このコールバックには、次のプロトタイプがあります。

```
void notification_callback (void *ctx, OCISubscription *subscrhp,  
                             void *payload, ub4 paylen, void *desc, ub4 mode);
```

パラメータの詳細は、『Oracle Call Interfaceプログラマーズ・ガイド』のOCIでの通知コールバックに関する項を参照してください。

4. 必要に応じて、OCI_ATTR_SUBSCR_CTX属性を使用してクライアント固有のコンテキストを関連付けます。
5. OCI_ATTR_SUBSCR_TIMEOUT属性を設定して、ub4タイムアウト間隔を秒単位で指定します。設定しない場合、タイムアウトはありません。
6. OCI_ATTR_SUBSCR_QOSFLAGS属性に次の値を使用して、QOS (サービス品質)レベルを設定します。
 - OCI_SUBSCR_QOS_PURGE_ON_NTFNフラグでは、最初の通知で登録がパージされます。
 - OCI_SUBSCR_QOS_RELIABLEフラグでは、通知が持続されます。Oracle RACの残存インスタンスを使用すると、ノードが失敗した後でも連続問合せ通知メッセージを送信および取得できます。これは、この登録に関連する無効化が、データベースに永続的にキューされるためです。FALSEである場合、無効化されたメッセージは高速インメモリ・キューにエンキューされます。このオプションは、登録の永続性ではなく、通知の永続性を記述します。デフォルトでは、登録は自動的に持続されます。
7. OCISubscriptionRegister()をコールし、DBCHANGEネームスペースに新しい登録を作成します。クライアントが開始したCQNには、OCISubscriptionRegister()を使用します。

関連項目:

クライアントが開始したCQNの詳細は、[「クライアントが開始したCQN登録でのOCIの使用」](#)を参照してください。

8. 複数の問合せ文をサブスクリプション・ハンドルに関連付けるには、文ハンドルOCI_HTYPE_STMTのOCI_ATTR_CHNF_REGHANDLE属性を設定します。問合せが実行されると、登録が完了します。

関連項目:

[OCI_ATTR_CHNF_REGHANDLE](#)の詳細は、『Oracle Call Interfaceプログラマーズ・ガイド』を参照してください

9. 必要に応じて、サブスクリプションの登録を解除します。クライアントはパラメータとしてサブスクリプション・ハンドルを使用して、OCISubscriptionRegister()ファンクションをコールできます。

サブスクリプション・ハンドルへの文ハンドルのバインドが有効なのは、問合せの初回の実行時のみです。アプリケーションで後続の実行時に同じOCI statement文ハンドルを使用する必要がある場合は、文ハンドルの登録ハンドル属性に値を再移入する必要があります。文ハンドルをサブスクリプション・ハンドルにバインドできるのは、文が問合せである場合のみです(実行時に判別されます)。実行の一部としてDML文が実行されると、例外が発行されます。

18.8.3 クライアントが開始したCQN登録でのOCIの使用

アプリケーションでは、クライアントが開始した接続モードを使用して、変更問合せ通知(CQN)を登録および受信できます。

このモードは、クラウドのアプリケーションで動作するように設計されていますが、オンプレミスで実行されているアプリケーションにも使用できます。この通知配信モードでは、クライアント・アプリケーションは通知を受信するためにOracleデータベース・サーバーへの接続を開始します。データベース・サーバーとアプリケーションを接続する必要はありません。クライアントが開始した接続は、特別なネットワーク構成を必要とせず、使いやすくセキュアです。

CQN登録を開始するには、アプリケーション・クライアントでOCIインタフェースを使用できます。モードの値がOCI_SECURE_NOTIFICATIONに設定されたOCISubscriptionRegister()関数をコールし、アプリケーション・クライアントを登録します。登録を削除するには、モードの値がOCI_SECURE_NOTIFICATIONに設定されたOCISubscriptionUnRegister()を使用します。

関連項目:

[OCISubscriptionRegister\(\)](#)および[OCISubscriptionUnRegister\(\)](#)の詳細は、『Oracle Call Interfaceプログラマーズ・ガイド』を参照してください

18.8.4 連続問合せ通知のOCIサブスクリプション・ハンドル属性の使用

連続問合せ通知のサブスクリプション・ハンドル属性は、汎用属性(すべてのサブスクリプションに対して共通)と、連続問合せ通知に固有のネームスペース特有属性に分けられます。

文ハンドルのWRITE属性を変更できるのは、登録の作成前のみです。

汎用属性 - すべてのサブスクリプションに共通

OCI_ATTR_SUBSCR_NAMESPACE(WRITE) - サブスクリプション・ハンドルの場合、OCI_SUBSCR_NAMESPACE_DBCHANGEに、この属性を設定します。

OCI_ATTR_SUBSCR_CALLBACK(WRITE) - サブスクリプション・ハンドルに関連するコールバックを格納するために、この属性を使用します。通知を受信すると、コールバックが実行されます。

新しい連続問合せ通知メッセージが作成されると、無効化に関する詳細情報が含まれるOCI_DTYPE_CHDES型の記述子がdescに設定されてコールバックがリスナー・スレッド内で呼び出されます。

OCI_ATTR_SUBSCR_QOSFLAGS: この属性は次の値を含む汎用フラグです。

```
#define OCI_SUBSCR_QOS_RELIABLE      0x01          /* reliable */
#define OCI_SUBSCR_QOS_PURGE_ON_NTFN 0x10          /* purge on first ntfn */
```

- OCI_SUBSCR_QOS_RELIABLE - このビットを設定し、通知を持続させます。このため、Oracle RACクラスタの残存インスタンスを使用して、ノードの失敗後も無効化メッセージを送信および取得できます。これは、この登録IDに関連する無効化が、データベースに永続的にキューされるためです。このビットがFALSEの場合、無効化は高速インメモリ・キューにエンキューされます。このオプションは、登録の永続性ではなく、通知の永続性を記述します。デフォルトでは、登録は自動的に持続されます。
- OCI_SUBSCR_QOS_PURGE_ON_NTFN - このビットを設定し、最初の通知で登録をページできるようにします。

パラレルのサンプルは、『Oracle Call Interfaceプログラマーズ・ガイド』のOCIのパブリッシュ・サブスクライブ登録機能に関する項を参照してください。

OCI_ATTR_SUBSCR_CQ_QOSFLAGS: この属性では、連続問合せ通知固有の次のQOSフラグを記述します(モードはWRITE、データ型はub4)。

- 0x1 OCI_SUBSCR_CQ_QOS_QUERY - このフラグを設定し、問合せレベルの細分性が必須であることを示します。通知は、問合せ結果セットが変更される場合にのみ生成します。デフォルトでは、このQOSレベルにFalseの正数はありません。
- 0x2 OCI_SUBSCR_CQ_QOS_BEST_EFFORT - このフラグを設定し、ベスト・エフォート・フィルタリングが許容されることを示します。アプリケーションをキャッシュすることで使用できます。データベースでは、評価のコストに基づく経験則を使用し、完全プルーニングを回避できる場合があります。

OCI_ATTR_SUBSCR_TIMEOUT - この属性を使用して、秒単位で定義したub4タイムアウト値を指定します。タイムアウト値が0(ゼロ)または未指定の場合、登録は明示的に解除されるまで有効です。

ネームスペース固有または機能固有の属性

次の属性は、ネームスペース固有または連続問合せ通知機能固有のものです。

OCI_ATTR_CHNF_TABLENAMES(データ型は(OCICOLL *)) - これらは、登録された表名のリストを取得するための属性です。これらの属性は、問合せの実行後にサブスクリプション・ハンドルから使用できます。

OCI_ATTR_CHNF_ROWIDS - ブール型属性(デフォルトはFALSE)です。TRUEの場合、連続問合せ通知メッセージには、操作タイプやROWIDなどの行レベルの詳細が含まれます。

OCI_ATTR_CHNF_OPERATIONS - ub4フラグを使用して、操作タイプに基づいて通知をフィルタで選択します。登録が問合せレベルの細分性を持つ場合、このオプションは無視されます。用意されているフラグは、次のとおりです。

- OCI_OPCODE_ALL - すべての操作
- OCI_OPCODE_INSERT - 表に対する挿入操作
- OCI_OPCODE_UPDATE - 表に対する更新操作
- OCI_OPCODE_DELETE - 表に対する削除操作

OCI_ATTR_CHNF_CHANGELAG - クライアントはこのub4値を使用して、遅延させるトランザクション数を指定できます。クライアントは、このオプションを連続問合せ通知メッセージのスロットル・メカニズムとしても使用できます。このオプションを選択すると、OCI_ATTR_CHNF_ROWIDSがTRUEであっても、通知にROWIDレベルの細分性を持つ情報が含まれなくなります。登録が問合せレベルの細分性を持つ場合、このオプションは無視されます。

OCISubscriptionRegister()のコールが起動されると、先行する属性(汎用、名前固有、または機能固有)はすべて、作成済の登録では変更不可になります。これらの属性への変更は、すでに作成されている登録には反映されませんが、新しく作成された、同じ登録ハンドルを使用する登録には反映されます。

関連項目:

[連続問合せ通知の記述子の属性](#)の詳細は、『Oracle Call Interfaceプログラマーズ・ガイド』を参照してください

NTFNグループ化オプションを使用して、通知の間隔を置くことができます。関連する汎用通知属性は次のとおりです。

```
OCI_ATTR_SUBSCR_NTFN_GROUPING_VALUE
OCI_ATTR_SUBSCR_NTFN_GROUPING_TYPE
OCI_ATTR_SUBSCR_NTFN_GROUPING_START_TIME
OCI_ATTR_SUBSCR_NTFN_GROUPING_REPEAT_COUNT
```


関連項目:

[データベースへの直接パブリッシュ・サブスクライブ登録](#)のこれらの属性の詳細は、『Oracle Call Interfaceプログラマーズ・ガイド』を参照してください

18.8.5 OCI_ATTR_CQ_QUERYID属性

OCIStmtExecute()のコールによる登録後、文ハンドルOCI_HTYPE_STMTのOCI_ATTR_CQ_QUERYID属性が登録済問合せの問合せIDを取得します。

関連項目:

[OCI_ATTR_CQ_QUERYID](#)の詳細は、『Oracle Call Interfaceプログラマーズ・ガイド』を参照してください

18.8.6 OCI連続問合せ通知の記述子の使用

連続問合せ通知の記述子は、アプリケーションによって指定された通知コールバックのdescパラメータに渡されます。次の属性は、連続問合せ通知に固有のものです。連続問合せ通知の記述子のOCI型定数は、OCI_DTYPE_CHDESです。

通知コールバックは、トップレベルの通知記述子OCI_DTYPE_CHDESを引数として受け取ります。この記述子には、イベント・タイプがOCI_EVENT_QUERYCHANGEであるかOCI_EVENT_OBJCHANGEであるかに基づいて、OCI_DTYPE_CQDESまたはOCI_DTYPE_TABLE_CHDES記述子のコレクションが含まれます。OCI_EVENT_QUERYCHANGE型の通知の場合、連続問合せ記述子には表連続問合せ記述子の配列が埋め込まれます。ROWIDレベルの細分性を持つ情報がリクエストされた場合、各OCI_DTYPE_TABLE_CHDESには変更された各ROWIDに対応する行レベルの連続問合せ記述子(OCI_DTYPE_ROW_CHDES)の配列が含まれます。

18.8.6.1 OCI_DTYPE_CHDES

これはトップレベルの連続問合せ通知記述子型です。

OCI_ATTR_CHDES_DBNAME (oratext *) - データベースの名前(連続問合せ通知のソース)

OCI_ATTR_CHDES_XID (RAW(8)) - メッセージのメッセージID

OCI_ATTR_CHDES_NFYTYPE - 通知タイプを示すフラグ。

- 0x0 OCI_EVENT_NONE - 連続問合せ通知に関する追加情報なし
- 0x1 OCI_EVENT_STARTUP - インスタンスの起動
- 0x2 OCI_EVENT_SHUTDOWN - インスタンスの停止
- 0x3 OCI_EVENT_SHUTDOWN_ANY - 任意のインスタンスの停止 - Oracle Real Application Clusters (RAC)
- 0x5 OCI_EVENT_DEREG - 登録解除またはタイムアウト
- 0x6 OCI_EVENT_OBJCHANGE - オブジェクト変更通知
- 0x7 OCI_EVENT_QUERYCHANGE - 問合せ変更通知

OCI_ATTR_CHDES_TABLE_CHANGES - 表またはデータ型(OCIColl *)の操作を記述するコレクション型。この属性は、OCI_ATTR_CHDES_NFYTYPE属性がOCI_EVENT_OBJCHANGE型の場合のみ表示され、そうでない場合はNULLに

なります。コレクションの各要素は、OCI_DTYPE_TABLE_CHDES型の連続問合せ記述子の表です。

OCI_ATTR_CHDES_QUERIES - 無効化された問合せを記述するコレクション型。コレクションの各メンバーはOCI_DTYPE_CQDES型です。この属性は、OCI_ATTR_CHDES_NFTYPE属性がOCI_EVENT_QUERYCHANGE型の場合のみ表示され、そうでない場合はNULLになります。

18.8.6.1.1 OCI_DTYPE_CQDES

この通知記述子は、通常はDMLトランザクションまたはDDLトランザクションのコミットに応じて、無効化された問合せを記述します。次の属性が含まれます。

- OCI_ATTR_CQDES_OPERATION (ub4, READ) - 問合せで発生した操作。これは次の値のいずれかです。
 - OCI_EVENT_QUERYCHANGE - 問合せ結果セットの変更
 - OCI_EVENT_DEREG - 問合せの登録解除
- OCI_ATTR_CQDES_TABLE_CHANGES (OCIColl *, READ) - 問合せ結果セットの変更原因となった表に対するDMLまたはDDL操作を記述する、表連続問合せ記述子のコレクション。コレクションの各要素はOCI_DTYPE_TABLE_CHDES型です。
- OCI_ATTR_CQDES_QUERYID (ub8, READ) - 無効化された問合せの問合せID。

18.8.6.1.2 OCI_DTYPE_TABLE_CHDES

この通知記述子は、登録済問合せに関係する表の変更内容に関する情報を送信します。次の属性が含まれます。

- OCI_ATTR_CHDES_TABLE_NAME (oratext *) - スキーマ注釈付き表名。
- OCI_ATTR_CHDES_TABLE_OPFLAGS (ub4)- 表への操作を示すフラグのフィールド。次の各フラグ・フィールドは、属性内の個別のビット位置にあります。
 - 0x1 OCI_OPCODE_ALLROWS - 表は完全に無効化されています。
 - 0x2 OCI_OPCODE_INSERT - 表に対する挿入操作。
 - 0x4 OCI_OPCODE_UPDATE - 表に対する更新操作。
 - 0x8 OCI_OPCODE_DELETE - 表に対する削除操作。
 - 0x10 OCI_OPCODE_ALTER - 表は変更されました(スキーマ変更)。これには、行の移行を引き起こしたDDL文と内部操作が含まれます。
 - 0x20 OCI_OPCODE_DROP - 表は削除されました。
- OCI_ATTR_CHDES_TABLE_ROW_CHANGES - これは、表内の行の変更を示す埋込みコレクションです。コレクションの各要素は、次の属性を持つOCI_DTYPE_ROW_CHDES型の行連続問合せ記述子です。
 - OCI_ATTR_CHDES_ROW_ROWID (OraText *)- ROWIDの文字列表現。
 - OCI_ATTR_CHDES_ROW_OPFLAGS - 操作タイプ(INSERT、UPDATE、DELETEまたはOTHER)を反映します。

関連項目:

[連続問合せ通知の記述子の属性](#)の詳細は、『Oracle Call Interfaceプログラマーズ・ガイド』を参照してください

18.8.7 OCIサンプル・プログラムでの連続問合せ通知のデモンストレーション

[例18-8](#)は単純なOCIプログラム、demoquery.cです。リストにあるコメントを参照してください。コール側セッションには、登録対象のすべてのオブジェクトに対するCHANGE NOTIFICATIONシステム権限およびSELECT権限が必要です。

例18-8 連続問合せ通知を示すプログラム・リスト

```
/* Copyright (c) 2010, Oracle. All rights reserved. */

#ifdef S_ORACLE
# include <oratypes.h>
#endif

/*****
 *This is a DEMO program. To test, compile the file to generate the executable
 *demoquery. Then demoquery can be invoked from a command prompt.
 *It will have the following output:

Initializing OCI Process
Registering query : select last_name, employees.department_id, department_name
                    from employees, departments
                    where employee_id = 200
                    and employees.department_id = departments.department_id
Query Id 23
Waiting for Notifications
*Then from another session, log in as HR/<password> and perform the following
* DML transactions. It will cause two notifications to be generated.

update departments set department_name = 'Global Admin' where department_id=10;
commit;
update departments set department_name = 'Administration' where department_id=10;
commit;
*The demoquery program will now show the following output corresponding
*to the notifications received.

Query 23 is changed
Table changed is HR.DEPARTMENTS table_op 4
Row changed is AAAMBoAABAAAKX2AAA row_op 4
Query 23 is changed
Table changed is HR.DEPARTMENTS table_op 4
Row changed is AAAMBoAABAAAKX2AAA row_op 4

*The demo program waits for exactly 10 notifications to be received before
*logging off and unregistering the subscription.

*****/

/*-----
PRIVATE TYPES AND CONSTANTS
-----*/
/*-----
STATIC FUNCTION DECLARATIONS
-----*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>

#define MAXSTRLLENGTH 1024
#define bit(a,b) ((a)&(b))

static int notifications_processed = 0;
```

```

static OCISubscription *subhandle1 = (OCISubscription *)0;
static OCISubscription *subhandle2 = (OCISubscription *)0;
static void checker(/*_ OCIError *errhp, sword status _*/);
static void registerQuery(/*_ OCISvcCtx *svchp, OCIError *errhp, OCISstmt *stmthp,
                           OCIEnv *envhp _*/);
static void myCallback (/*_ dvoid *ctx, OCISubscription *subscrhp,
                        dvoid *payload, ub4 *payl, dvoid *descriptor,
                        ub4 mode _*/);
static int NotificationDriver(/*_ int argc, char *argv[] _*/);
static sword status;
static boolean logged_on = FALSE;
static void processRowChanges(OCIEnv *envhp, OCIError *errhp, OCISstmt *stmthp,
                             OCIColl *row_changes);
static void processTableChanges(OCIEnv *envhp, OCIError *errhp,
                                OCISstmt *stmthp, OCIColl *table_changes);
static void processQueryChanges(OCIEnv *envhp, OCIError *errhp, OCISstmt *stmthp,
                                OCIColl *query_changes);
static int nonractests2(/*_ int argc, char *argv[] _*/);

int main(int argc, char **argv)
{
    NotificationDriver(argc, argv);
    return 0;
}

int NotificationDriver(argc, argv)
int argc;
char *argv[];
{
    OCIEnv *envhp;
    OCISvcCtx *svchp, *svchp2;
    OCIError *errhp, *errhp2;
    OCISession *authp, *authp2;
    OCISstmt *stmthp, *stmthp2;
    OCIDuration dur, dur2;
    int i;
    dvoid *tmp;
    OCISession *usrhp;
    OCIServer *srvhp;

    printf("Initializing OCI Process\n");
    /* Initialize the environment. The environment must be initialized
       with OCI_EVENTS and OCI_OBJECT to create a continuous query notification
       registration and receive notifications.
    */
    OCIEnvCreate( (OCIEnv **) &envhp, OCI_EVENTS|OCI_OBJECT, (dvoid *)0,
                  (dvoid * (*)(dvoid *, size_t)) 0,
                  (dvoid * (*)(dvoid *, dvoid *, size_t))0,
                  (void (*)(dvoid *, dvoid *)) 0,
                  (size_t) 0, (dvoid **) 0 );

    OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &errhp, OCI_HTYPE_ERROR,
                    (size_t) 0, (dvoid **) 0);
    /* server contexts */
    OCIHandleAlloc((dvoid *) envhp, (dvoid **) &srvhp, OCI_HTYPE_SERVER,
                    (size_t) 0, (dvoid **) 0);
    OCIHandleAlloc((dvoid *) envhp, (dvoid **) &svchp, OCI_HTYPE_SVCCTX,
                    (size_t) 0, (dvoid **) 0);
    checker(errhp, OCI_SrvAttach(srvhp, errhp, (text *) 0, (sb4) 0,
                                (ub4) OCI_DEFAULT));
    /* set attribute server context in the service context */
    OCIAttrSet( (dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *)srvhp,
                (ub4) 0, (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

```

```

    /* allocate a user context handle */
    OCIHandleAlloc((dvoid *)envhp, (dvoid **)&usrhp, (ub4) OCI_HTYPE_SESSION,
                    (size_t) 0, (dvoid **) 0);

    OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
                (dvoid *)((text *)"HR"), (ub4)strlen((char *)"HR"),
                OCI_ATTR_USERNAME, errhp);

    OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
                (dvoid *)((text *)"HR"), (ub4)strlen((char *)"HR"),
                OCI_ATTR_PASSWORD, errhp);
    checker(errhp, OCISessionBegin (svchp, errhp, usrhp, OCI_CRED_RDBMS,
                                    OCI_DEFAULT));
    /* Allocate a statement handle */
    OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &stmthp,
                    (ub4) OCI_HTYPE_STMT, 52, (dvoid **) &tmp);

    OCIAttrSet((dvoid *)svchp, (ub4)OCI_HTYPE_SVCCTX, (dvoid *)usrhp, (ub4)0,
                OCI_ATTR_SESSION, errhp);

    registerQuery(svchp, errhp, stmthp, envhp);
    printf("Waiting for Notifications\n");
    while (notifications_processed !=10)
    {
        sleep(1);
    }
    printf ("Going to unregister HR\n");
    fflush(stdout);
    /* Unregister HR */
    checker(errhp,
            OCISubscriptionUnRegister(svchp, subhandle1, errhp, OCI_DEFAULT));
    checker(errhp, OCISessionEnd(svchp, errhp, usrhp, (ub4) 0));
    printf("HR Logged off.\n");

    if (subhandle1)
        OCIHandleFree((dvoid *)subhandle1, OCI_HTYPE_SUBSCRIPTION);
    if (stmthp)
        OCIHandleFree((dvoid *)stmthp, OCI_HTYPE_STMT);
    if (srvhp)
        OCIHandleFree((dvoid *) srvhp, (ub4) OCI_HTYPE_SERVER);
    if (svchp)
        OCIHandleFree((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX);
    if (authp)
        OCIHandleFree((dvoid *) usrhp, (ub4) OCI_HTYPE_SESSION);
    if (errhp)
        OCIHandleFree((dvoid *) errhp, (ub4) OCI_HTYPE_ERROR);
    if (envhp)
        OCIHandleFree((dvoid *) envhp, (ub4) OCI_HTYPE_ENV);

    return 0;
}

void checker(errhp, status)
OCIError *errhp;
sword status;
{
    text errbuf[512];
    sb4 errcode = 0;
    int retval = 1;

    switch (status)
    {
    case OCI_SUCCESS:
        retval = 0;
        break;

```

```

case OCI_SUCCESS_WITH_INFO:
    (void) printf("Error - OCI_SUCCESS_WITH_INFO\n");
    break;
case OCI_NEED_DATA:
    (void) printf("Error - OCI_NEED_DATA\n");
    break;
case OCI_NO_DATA:
    (void) printf("Error - OCI_NODATA\n");
    break;
case OCI_ERROR:
    (void) OCIErrorGet((dvoid *)errhp, (ub4) 1, (text *) NULL, &errcode,
                      errbuf, (ub4) sizeof(errbuf), OCI_HTYPE_ERROR);
    (void) printf("Error - %.s\n", 512, errbuf);
    break;
case OCI_INVALID_HANDLE:
    (void) printf("Error - OCI_INVALID_HANDLE\n");
    break;
case OCI_STILL_EXECUTING:
    (void) printf("Error - OCI_STILL_EXECUTE\n");
    break;
case OCI_CONTINUE:
    (void) printf("Error - OCI_CONTINUE\n");
    break;
default:
    break;
}
if (retval)
{
    exit(1);
}
}

void processRowChanges(OCIEnv *envhp, OCIError *errhp, OCISmt *stmthp,
                      OCIColl *row_changes)
{
    dvoid **row_descp;
    dvoid *row_desc;
    boolean exist;
    ub2 i, j;
    dvoid *elemind = (dvoid *)0;
    oratext *row_id;
    ub4 row_op;

    sb4 num_rows;
    if (!row_changes) return;
    checker(errhp, OCICollSize(envhp, errhp,
                              (CONST OCIColl *) row_changes, &num_rows));
    for (i=0; i<num_rows; i++)
    {
        checker(errhp, OCICollGetElem(envhp,
                                      errhp, (OCIColl *) row_changes,
                                      i, &exist, &row_descp, &elemind));

        row_desc = *row_descp;
        checker(errhp, OCIAttrGet (row_desc,
                                   OCI_DTYPE_ROW_CHDES, (dvoid *)&row_id,
                                   NULL, OCI_ATTR_CHDES_ROW_ROWID, errhp));
        checker(errhp, OCIAttrGet (row_desc,
                                   OCI_DTYPE_ROW_CHDES, (dvoid *)&row_op,
                                   NULL, OCI_ATTR_CHDES_ROW_OPFLAGS, errhp));

        printf ("Row changed is %s row_op %d\n", row_id, row_op);
        fflush(stdout);
    }
}

```

```

void processTableChanges(OCIEnv *envhp, OCIError *errhp, OCISmt *stmthp,
                        OCIColl *table_changes)
{
    dvoid **table_descp;
    dvoid *table_desc;
    dvoid **row_descp;
    dvoid *row_desc;
    OCIColl *row_changes = (OCIColl *)0;
    boolean exist;
    ub2 i, j;
    dvoid *elemind = (dvoid *)0;
    oratext *table_name;
    ub4 table_op;

    sb4 num_tables;
    if (!table_changes) return;
    checker(errhp, OCICollSize(envhp, errhp,
                              (CONST OCIColl *) table_changes, &num_tables));
    for (i=0; i<num_tables; i++)
    {
        checker(errhp, OCICollGetElem(envhp,
                                      errhp, (OCIColl *) table_changes,
                                      i, &exist, &table_descp, &elemind));

        table_desc = *table_descp;
        checker(errhp, OCIAttrGet (table_desc,
                                   OCI_DTYPE_TABLE_CHDES, (dvoid *)&table_name,
                                   NULL, OCI_ATTR_CHDES_TABLE_NAME, errhp));
        checker(errhp, OCIAttrGet (table_desc,
                                   OCI_DTYPE_TABLE_CHDES, (dvoid *)&table_op,
                                   NULL, OCI_ATTR_CHDES_TABLE_OPFLAGS, errhp));
        checker(errhp, OCIAttrGet (table_desc,
                                   OCI_DTYPE_TABLE_CHDES, (dvoid *)&row_changes,
                                   NULL, OCI_ATTR_CHDES_TABLE_ROW_CHANGES, errhp));

        printf ("Table changed is %s table_op %d\n", table_name, table_op);
        fflush(stdout);
        if (!bit(table_op, OCI_OPCODE_ALLROWS))
            processRowChanges(envhp, errhp, stmthp, row_changes);
    }
}

void processQueryChanges(OCIEnv *envhp, OCIError *errhp, OCISmt *stmthp,
                        OCIColl *query_changes)
{
    sb4 num_queries;
    ub8 queryid;
    OCINumber qidnum;
    ub4 queryop;
    dvoid *elemind = (dvoid *)0;
    dvoid *query_desc;
    dvoid **query_descp;
    ub2 i;
    boolean exist;
    OCIColl *table_changes = (OCIColl *)0;

    if (!query_changes) return;
    checker(errhp, OCICollSize(envhp, errhp,
                              (CONST OCIColl *) query_changes, &num_queries));
    for (i=0; i < num_queries; i++)
    {
        checker(errhp, OCICollGetElem(envhp,
                                      errhp, (OCIColl *) query_changes,
                                      i, &exist, &query_descp, &elemind));
    }
}

```

```

    query_desc = *query_descp;
    checker(errhp, OCIAttrGet (query_desc,
        OCI_DTYPE_CQDES, (dvoid *)&queryid,
        NULL, OCI_ATTR_CQDES_QUERYID, errhp));
    checker(errhp, OCIAttrGet (query_desc,
        OCI_DTYPE_CQDES, (dvoid *)&queryop,
        NULL, OCI_ATTR_CQDES_OPERATION, errhp));
    printf(" Query %d is changed\n", queryid);
    if (queryop == OCI_EVENT_DEREG)
        printf("Query Deregistered\n");
    checker(errhp, OCIAttrGet (query_desc,
        OCI_DTYPE_CQDES, (dvoid *)&table_changes,
        NULL, OCI_ATTR_CQDES_TABLE_CHANGES, errhp));
    processTableChanges(envhp, errhp, stmthp, table_changes);

}
}

void myCallback (ctx, subscrhp, payload, payl, descriptor, mode)
dvoid *ctx;
OCISubscription *subscrhp;
dvoid *payload;
ub4 *payl;
dvoid *descriptor;
ub4 mode;
{
    OCIColl *table_changes = (OCIColl *)0;
    OCIColl *row_changes = (OCIColl *)0;
    dvoid *change_descriptor = descriptor;
    ub4 notify_type;
    ub2 i, j;
    OCIEnv *envhp;
    OCIError *errhp;
    OCIColl *query_changes = (OCIColl *)0;
    OCIServer *srvhp;
    OCISvcCtx *svchp;
    OCISession *usrhp;
    dvoid *tmp;
    OCISmt *stmthp;

    (void)OCIEnvInit( (OCIEnv **) &envhp, OCI_DEFAULT, (size_t)0, (dvoid **)0 );

    (void) OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &errhp, OCI_HTYPE_ERROR,
        (size_t) 0, (dvoid **) 0);
    /* server contexts */
    (void) OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &srvhp, OCI_HTYPE_SERVER,
        (size_t) 0, (dvoid **) 0);

    (void) OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &svchp, OCI_HTYPE_SVCCTX,
        (size_t) 0, (dvoid **) 0);

    OCIAttrGet (change_descriptor, OCI_DTYPE_CHDES, (dvoid *) &notify_type,
        NULL, OCI_ATTR_CHDES_NFYTYPE, errhp);
    fflush(stdout);
    if (notify_type == OCI_EVENT_SHUTDOWN ||
        notify_type == OCI_EVENT_SHUTDOWN_ANY)
    {
        printf("SHUTDOWN NOTIFICATION RECEIVED\n");
        fflush(stdout);
        notifications_processed++;
        return;
    }
    if (notify_type == OCI_EVENT_STARTUP)
    {
        printf("STARTUP NOTIFICATION RECEIVED\n");

```

```

        fflush(stdout);
        notifications_processed++;
        return;
    }

    notifications_processed++;
    checker(errhp, OCIServerAttach( srvhp, errhp, (text *) 0, (sb4) 0,
                                   (ub4) OCI_DEFAULT));

    OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &svchp, (ub4) OCI_HTYPE_SVCCTX,
                    52, (dvoid **) &tmp);
    /* set attribute server context in the service context */
    OCIAttrSet( (dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *)srvhp,
                (ub4) 0, (ub4) OCI_ATTR_SERVER, (OCIError *) errhp);

    /* allocate a user context handle */
    OCIHandleAlloc((dvoid *)envhp, (dvoid **)&usrhp, (ub4) OCI_HTYPE_SESSION,
                   (size_t) 0, (dvoid **) 0);

    OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
                (dvoid *)"HR", (ub4)strlen("HR"), OCI_ATTR_USERNAME, errhp);

    OCIAttrSet((dvoid *)usrhp, (ub4)OCI_HTYPE_SESSION,
                (dvoid *)"HR", (ub4)strlen("HR"),
                OCI_ATTR_PASSWORD, errhp);

    checker(errhp, OCISessionBegin (svchp, errhp, usrhp, OCI_CRED_RDBMS,
                                   OCI_DEFAULT));

    OCIAttrSet((dvoid *)svchp, (ub4)OCI_HTYPE_SVCCTX,
                (dvoid *)usrhp, (ub4)0, OCI_ATTR_SESSION, errhp);

    /* Allocate a statement handle */
    OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &stmthp,
                    (ub4) OCI_HTYPE_STMT, 52, (dvoid **) &tmp);

    if (notify_type == OCI_EVENT_OBJCHANGE)
    {
        checker(errhp, OCIAttrGet (change_descriptor,
                                   OCI_DTYPE_CHDES, &table_changes, NULL,
                                   OCI_ATTR_CHDES_TABLE_CHANGES, errhp));
        processTableChanges(envhp, errhp, stmthp, table_changes);
    }
    else if (notify_type == OCI_EVENT_QUERYCHANGE)
    {
        checker(errhp, OCIAttrGet (change_descriptor,
                                   OCI_DTYPE_CHDES, &query_changes, NULL,
                                   OCI_ATTR_CHDES_QUERIES, errhp));
        processQueryChanges(envhp, errhp, stmthp, query_changes);
    }
    checker(errhp, OCISessionEnd(svchp, errhp, usrhp, OCI_DEFAULT));
    checker(errhp, OCIServerDetach(srvhp, errhp, OCI_DEFAULT));
    if (stmthp)
        OCIHandleFree((dvoid *)stmthp, OCI_HTYPE_STMT);
    if (errhp)
        OCIHandleFree((dvoid *)errhp, OCI_HTYPE_ERROR);
    if (srvhp)
        OCIHandleFree((dvoid *)srvhp, OCI_HTYPE_SERVER);
    if (svchp)
        OCIHandleFree((dvoid *)svchp, OCI_HTYPE_SVCCTX);
    if (usrhp)
        OCIHandleFree((dvoid *)usrhp, OCI_HTYPE_SESSION);
    if (envhp)
        OCIHandleFree((dvoid *)envhp, OCI_HTYPE_ENV);
}

```

```

void registerQuery(svchp, errhp, stmthp, envhp)
OCISvcCtx *svchp;
OCIError *errhp;
OCISmt *stmthp;
OCIEnv *envhp;
{
    OCISubscription *subscrhp;
    ub4 namespace = OCI_SUBSCR_NAMESPACE_DBCHANGE;
    ub4 timeout = 60;
    OCIDefine *defnp1 = (OCIDefine *)0;
    OCIDefine *defnp2 = (OCIDefine *)0;
    OCIDefine *defnp3 = (OCIDefine *)0;
    OCIDefine *defnp4 = (OCIDefine *)0;
    OCIDefine *defnp5 = (OCIDefine *)0;
    int mgr_id = 0;
    text query_text1[] = "select last_name, employees.department_id, department_name ¥
    from employees,departments where employee_id = 200 and employees.department_id =¥
    departments.department_id";

    ub4 num_prefetch_rows = 0;
    ub4 num_reg_tables;
    OCIColl *table_names;
    ub2 i;
    boolean rowids = TRUE;
    ub4 qosflags = OCI_SUBSCR_CQ_QOS_QUERY ;
    int empno=0;
    OCINumber qidnum;
    ub8 qid;
    char outstr[MAXSTRLENGTH], dname[MAXSTRLENGTH];
    int q3out;

    fflush(stdout);
    /* allocate subscription handle */
    OCIHandleAlloc ((dvoid *) envhp, (dvoid **) &subscrhp,
                    OCI_HTYPE_SUBSCRIPTION, (size_t) 0, (dvoid **) 0);

    /* set the namespace to DBCHANGE */
    checker(errhp, OCIAttrSet (subscrhp, OCI_HTYPE_SUBSCRIPTION,
                              (dvoid *) &namespace, sizeof(ub4),
                              OCI_ATTR_SUBSCR_NAMESPACE, errhp));

    /* Associate a notification callback with the subscription */
    checker(errhp, OCIAttrSet (subscrhp, OCI_HTYPE_SUBSCRIPTION,
                              (void *)myCallback, 0, OCI_ATTR_SUBSCR_CALLBACK, errhp));
    /* Allow extraction of rowid information */
    checker(errhp, OCIAttrSet (subscrhp, OCI_HTYPE_SUBSCRIPTION,
                              (dvoid *)&rowids, sizeof(ub4),
                              OCI_ATTR_CHNF_ROWIDS, errhp));

    checker(errhp, OCIAttrSet (subscrhp, OCI_HTYPE_SUBSCRIPTION,
                              (dvoid *)&qosflags, sizeof(ub4),
                              OCI_ATTR_SUBSCR_CQ_QOSFLAGS, errhp));

    /* Create a new registration in the DBCHANGE namespace */
    checker(errhp,
            OCISubscriptionRegister(svchp, &subscrhp, 1, errhp, OCI_DEFAULT));

    /* Multiple queries can now be associated with the subscription */

    subhandle1 = subscrhp;

    printf("Registering query : %s¥n", (const signed char *)query_text1);
    /* Prepare the statement */
    checker(errhp, OCISmtPrepare (stmthp, errhp, query_text1,
                                  (ub4)strlen((const signed char *)query_text1), OCI_V7_SYNTAX,
                                  OCI_DEFAULT));

```



```

    checker(errhp,
        OCIDefineByPos(stmthp, &defnp1,
            errhp, 1, (dvoid *)outstr, MAXSTRLENGTH * sizeof(char),
            SQLT_STR, (dvoid *)0, (ub2 *)0, (ub2 *)0, OCI_DEFAULT));
    checker(errhp,
        OCIDefineByPos(stmthp, &defnp2,
            errhp, 2, (dvoid *)&empno, sizeof(empno),
            SQLT_INT, (dvoid *)0, (ub2 *)0, (ub2 *)0, OCI_DEFAULT));
    checker(errhp,
        OCIDefineByPos(stmthp, &defnp3,
            errhp, 3, (dvoid *)&dname, sizeof(dname),
            SQLT_STR, (dvoid *)0, (ub2 *)0, (ub2 *)0, OCI_DEFAULT));

    /* Associate the statement with the subscription handle */
    OCIAttrSet (stmthp, OCI_HTYPE_STMT, subscrhp, 0,
        OCI_ATTR_CHNF_REGHANDLE, errhp);

    /* Execute the statement, the execution performs object registration */
    checker(errhp, OCISstmtExecute (svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
        (CONST OCISnapshot *) NULL, (OCISnapshot *) NULL ,
        OCI_DEFAULT));
    fflush(stdout);

    OCIAttrGet(stmthp, OCI_HTYPE_STMT, &qid, (ub4 *)0,
        OCI_ATTR_CQ_QUERYID, errhp);
    printf("Query Id %d¥n", qid);

    /* commit */
    checker(errhp, OCITransCommit(svchp, errhp, (ub4) 0));
}

static void cleanup(envhp, svchp, srvhp, errhp, usrhp)
OCIEnv *envhp;
OCISvcCtx *svchp;
OCIError *errhp;
OCIError *errhp;
OCISession *usrhp;
{
    /* detach from the server */
    checker(errhp, OCISessionEnd(svchp, errhp, usrhp, OCI_DEFAULT));
    checker(errhp, OCIServerDetach(srvhp, errhp, (ub4)OCI_DEFAULT));

    if (usrhp)
        (void) OCIHandleFree((dvoid *) usrhp, (ub4) OCI_HTYPE_SESSION);
    if (svchp)
        (void) OCIHandleFree((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX);
    if (srvhp)
        (void) OCIHandleFree((dvoid *) srvhp, (ub4) OCI_HTYPE_SERVER);
    if (errhp)
        (void) OCIHandleFree((dvoid *) errhp, (ub4) OCI_HTYPE_ERROR);
    if (envhp)
        (void) OCIHandleFree((dvoid *) envhp, (ub4) OCI_HTYPE_ENV);
}

```

18.9 CQN登録の問合せ

登録のQOSオプションなど、すべての登録に関するトップレベルの情報を確認するには、静的データ・ディクショナリ・ビュー *_CHANGE_NOTIFICATION_REGSを問い合わせます。

たとえば、クライアントの登録IDと通知を受信するオブジェクトのリストを取得できます。HRの登録IDと表名を表示するには、次

の問合せを使用します。

```
SELECT regid, table_name FROM USER_CHANGE_NOTIFICATION_REGS;
```

どの問合せがQRCNに対して登録されているかを確認するには、静的データ・ディクショナリ・ビュー `USER_CQ_NOTIFICATION_QUERIES` または `DBA_CQ_NOTIFICATION_QUERIES` を問い合わせます。これらのビューには、問合せで使用されるバインド値に関する情報が含まれています。これらのビューでは、元の問合せのバインド値が、定数として問合せテキストに含まれます。問合せテキストは、登録済の元の問合せと同等ですが、同一ではない場合があります。

関連項目:

静的データ・ディクショナリ・ビュー [USER_CHANGE_NOTIFICATION_REGS](#) および [DBA_CQ_NOTIFICATION_QUERIES](#) の詳細は、『Oracle Database リファレンス』を参照してください。

18.10 通知の解析

トランザクションのコミット時に、データベースでは登録済オブジェクトがトランザクション中に変更されたかどうかが判別されます。変更されたと判別された場合、登録で指定されている通知ハンドラが実行されます。

トピック:

- [CQ_NOTIFICATION\\$_DESCRIPTOR オブジェクトの解析](#)
- [CQ_NOTIFICATION\\$_TABLE オブジェクトの解析](#)
- [CQ_NOTIFICATION\\$_QUERY オブジェクトの解析](#)
- [CQ_NOTIFICATION\\$_ROW オブジェクトの解析](#)

18.10.1 CQ_NOTIFICATION\$_DESCRIPTOR オブジェクトの解析

CQN登録で通知が生成されると、`CQ_NOTIFICATION$_DESCRIPTOR` オブジェクトがデータベースから通知ハンドラに渡されます。通知ハンドラでは、データベース変更の詳細を `CQ_NOTIFICATION$_DESCRIPTOR` オブジェクトの属性から検出できます。

SQL*Plusでは、SYSとして接続し、次の文を実行するとこれらの属性を表示できます。

```
DESC CQ_NOTIFICATION$_DESCRIPTOR
```

[表18-4](#)に、`CQ_NOTIFICATION$_DESCRIPTOR`の属性の要約を示します。

表18-4 `CQ_NOTIFICATION$_DESCRIPTOR`の属性

属性	説明:
REGISTRATION_ID	登録時に戻された登録 ID。
TRANSACTION_ID	変更を行ったトランザクションの ID。
DBNAME	通知が生成されたデータベースの名前。

属性	説明:
EVENT_TYPE	通知をトリガーするデータベース・イベント。たとえば、属性には様々なデータベース・イベントに対応する次の定数を含めることができます。 ● EVENT_NONE ● EVENT_STARTUP(インスタンスの起動) ● EVENT_SHUTDOWN(インスタンスの停止 - Oracle RAC の場合は最後のインスタンスの停止) ● EVENT_SHUTDOWN_ANY(Oracle RAC の場合は任意のインスタンスの停止) ● EVENT_DEREG(登録の削除) ● EVENT_OBJCHANGE(登録済の表に対する変更) ● EVENT_QUERYCHANGE(登録済の結果セットに対する変更)
NUMTABLES	変更された表の数。
TABLE_DESC_ARRAY	このフィールドは、OCN 登録の場合にのみ存在します。QRCN 登録の場合は NULL になります。 EVENT_TYPE が EVENT_OBJCHANGE の場合は、タイプ CQ_NOTIFICATION\$_TABLE の表変更記述子の VARRAY となり、それぞれが変更された表に対応します。CQ_NOTIFICATION\$_TABLE の属性は、表 18-5を参照してください。 前述以外の場合は NULL になります。
QUERY_DESC_ARRAY	このフィールドは、QRCN 登録の場合にのみ存在します。OCN 登録の場合は NULL になります。 EVENT_TYPE が EVENT_QUERYCHANGE の場合は、タイプ CQ_NOTIFICATION\$_QUERY の結果セット変更記述子の VARRAY となり、それぞれが変更された結果セットに対応します。CQ_NOTIFICATION\$_QUERY の属性は、表 18-6を参照してください。 前述以外の場合は NULL になります。

18.10.2 CQ_NOTIFICATION\$_TABLEオブジェクトの解析

CQ_NOTIFICATION\$_DESCRIPTORタイプには、タイプCQ_NOTIFICATION\$_TABLEの表記述子のVARRAYを保持する属性TABLE_DESC_ARRAYが含まれています。

SQL*Plusでは、SYSとして接続し、次の文を実行するとこれらの属性を表示できます。

```
DESC CQ_NOTIFICATION$_TABLE
```

[表18-5](#)に、CQ_NOTIFICATION\$_TABLEの属性の要約を示します。

表18-5 CQ_NOTIFICATION\$_TABLEの属性

属性	指定内容
OPFLAGS	変更があった表に対して実行された操作のタイプ。たとえば、属性には様々なデータベース操作に対応する次の定数を含めることができます。 ● ALL_ROWS は、DELETE *のように表全体に変更があるか、行レベルの粒度の情報が要求されないか、通知に使用できず、受信者は表全体が変更されたとみなす必要があることを示します。 ● UPDATEOP は更新を示します。 ● DELETEOP は削除を示します。 ● ALTEROP は ALTER TABLE を示します。 ● DROPPOP は DROP TABLE を示します。 ● UNKNOWNOP は不明な操作を示します。
TABLE_NAME	変更があった表の名前。
NUMROWS	変更があった行数。
ROW_DESC_ARRAY	タイプ CQ_NOTIFICATION\$_ROW の行記述子の VARRAY(表 18-7 を参照)。opflags に ALL_ROWS が設定されていた場合、ROW_DESC_ARRAY メンバーは NULL です。

18.10.3 CQ_NOTIFICATION\$_QUERYオブジェクトの解析

CQ_NOTIFICATION\$_DESCRIPTORタイプには、タイプCQ_NOTIFICATION\$_QUERYの結果セット変更記述子のVARRAYを保持する属性QUERY_DESC_ARRAYが含まれています。

SQL*Plusでは、SYSとして接続し、次の文を実行するとこれらの属性を表示できます。

```
DESC CQ_NOTIFICATION$_QUERY
```

[表18-6](#)に、CQ_NOTIFICATION\$_QUERYの属性の要約を示します。

表18-6 CQ_NOTIFICATION\$_QUERYの属性

属性	指定内容

属性	指定内容
QUERYID	変更された問合せの ID。
QUERYOP	問合せを変更した操作(EVENT_QUERYCHANGE または EVENT_DEREG)。
TABLE_DESC_ARRAY	タイプ CQ_NOTIFICATION\$_TABLE の表変更記述子の VARRAY(それぞれが結果セットの変更の原因となった変更済の表に対応)。CQ_NOTIFICATION\$_TABLE の属性は、 表 18-5 を参照してください。

18.10.4 CQ_NOTIFICATION\$_ROWオブジェクトの解析

登録中にROWIDオプションが指定されると、CQ_NOTIFICATION\$_TABLEタイプにはROW_DESC_ARRAY属性、つまり、変更された行のROWIDを含むタイプCQ_NOTIFICATION\$_ROWのVARRAYが設定されます。

CQ_NOTIFICATION\$_TABLEオブジェクトのOPFLAGSフィールドにALL_ROWSが設定されていた場合、ROWID情報は使用できません。

[表18-7](#)に、CQ_NOTIFICATION\$_ROWの属性の要約を示します。

表18-7 CQ_NOTIFICATION\$_ROWの属性

属性	指定内容
OPFLAGS	変更があった表に対して実行された操作のタイプ。表 18-5 の OPFLAGS の説明を参照してください。
ROW_ID	変更された行の ROWID。

第IV部 アプリケーション開発者向けの高度なトピック

ここでは、高度なテクノロジーが必要なアプリケーションや、一部の開発者のみが使用するテクノロジーに関するアプリケーション開発の情報について説明します。

章:

- [Oracle Flashback Technologyの使用](#)
- [プログラミング環境の選択](#)
- [複数のプログラミング言語を使用したアプリケーションの開発](#)
- [Oracle XAを使用したアプリケーションの開発](#)
- [パブリッシュ・サブスクライブ・モデルでのアプリケーションの開発](#)
- [Oracle ODBC Driverの使用方法](#)
- [識別コード・パッケージの使用](#)
- [スキーマ・オブジェクトの依存性の理解](#)
- [エディションベースの再定義の使用](#)
- [トランザクション・ガードの使用](#)

関連項目:

アプリケーションの開発時に検討する必要があるパフォーマンス上の問題の詳細は、[『Oracle Database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

19 Oracle Flashback Technologyの使用

この章では、データベース・アプリケーションにおけるOracle Flashback Technologyの使用について説明します。

トピック:

- [Oracle Flashback Technologyの概要](#)
- [Oracle Flashback Technologyに関するデータベースの構成](#)
- [Oracle Flashback Query\(SELECT AS OF\)の使用](#)
- [Oracle Flashback Version Queryの使用](#)
- [Oracle Flashback Transaction Queryの使用](#)
- [Oracle Flashback Transaction QueryとOracle Flashback Version Queryの併用](#)
- [DBMS_FLASHBACKパッケージの使用](#)
- [フラッシュバック・トランザクションの使用](#)
- [フラッシュバック時間移動の使用](#)
- [Oracle Flashback Technologyの一般ガイドライン](#)
- [Oracle Flashback Technologyのパフォーマンス・ガイドライン](#)
- [マルチテナント・コンテナ・データベースでのOracle Flashback Technologyの制限](#)

19.1 Oracle Flashback Technologyの概要

Oracle Flashback TechnologyとはOracle Database機能のグループの1つであり、Point-in-Timeメディア・リカバリを使用しなくても、データベース・オブジェクトの過去の状態を表示したり、データベース・オブジェクトを前の状態に戻すことができます。

フラッシュバック機能を使用すると、次のことができます。

- 過去のデータを戻す問合せを実行します。
- データベースに対する変更の詳細履歴を示したメタデータを戻す問合せを実行します。
- 表または行を前の時点にリカバリします。
- トランザクション・データの変更を自動的に追跡およびアーカイブします。
- データベースがオンラインである間にトランザクションおよびその依存トランザクションをロールバックします。

Oracle Flashback機能では、自動UNDO管理(AUM)システムにより、トランザクションに関するメタデータおよび履歴データが取得されます。フラッシュバック機能は**UNDOデータ**に依存します。UNDOデータは、個々のトランザクションの結果のレコードです。たとえば、ユーザーが給与を1000から1100に変更するUPDATE文を実行すると、Oracle DatabaseによってUNDOデータに値1000が格納されます。

UNDOデータは永続的であり、データベース停止時にも失われません。これが保持されるのは、undo_retentionに指定された期間か、自動UNDO管理(AUM)が存在する場合はチューニングされたUNDO保存期間までです。フラッシュバック機能を使用すると、UNDOデータを使用して過去のデータを問い合わせたり、論理的な損害をリカバリしたりすることができます。UNDOデータは、フラッシュバック機能以外でも、Oracle Databaseによって次の処理に使用されます。

- アクティブなトランザクションのロールバック
- データベースまたはプロセス・リカバリを使用した終了済トランザクションのリカバリ
- SQL問合せに対する読取り一貫性の提供

ノート:



CREATE TABLE 文の実行後、トランザクションをコミットするために少なくとも 15 秒待機して、Oracle Flashback 機能(特に Oracle Flashback Version Query)にそのトランザクションが反映されるようにします。

ノート:



Oracle Database では、versions_starttime 列、versions_endtime 列または scn_to_timestamp 列を

VERSIONS 問合せ(CTAS 問合せを含む)で使用するのを避けてパフォーマンスの向上を図ることをお勧めします。

トピック:

- [アプリケーション開発機能](#)
- [データベース管理機能](#)

関連項目:

フラッシュバック機能の詳細は、『[Oracle Database概要](#)』を参照してください。

19.1.1 アプリケーション開発機能

アプリケーション開発では、次のフラッシュバック機能を使用して、履歴データに関するレポートを作成したり、誤った変更を元に戻すことができます。(この機能は、データベース・ユーザーまたは管理者として対話形式でも使用できます。)

Oracle Flashback Query

この機能を使用して、SELECT文のAS OF句で指定した過去のある時点のデータを取得できます。

関連項目:

[Oracle Flashback Query\(SELECT AS OF\)の使用](#)

Oracle Flashback Version Query

この機能を使用して、特定の時間間隔内のメタデータおよび履歴データを取得します(たとえば、ある表の、特定の時間間隔内に存在していたすべての行を表示します)。各行バージョンに関するメタデータには、開始時間と終了時間、変更処理のタイプおよび行バージョンを作成したトランザクションの識別情報が含まれます。Oracle Flashback Version Queryを作成する

には、SELECT文でVERSIONS BETWEEN句を使用します。

関連項目:

[Oracle Flashback Version Queryの使用](#)

Oracle Flashback Transaction Query

この機能を使用して、特定の時間間隔内における特定のトランザクションまたはすべてのトランザクションのメタデータおよび履歴データを取得できます。Oracle Flashback Transaction Queryを実行するには、静的データ・ディクショナリ・ビューFLASHBACK_TRANSACTION_QUERYから選択します。

関連項目:

[Oracle Flashback Transaction Queryの使用](#)

通常は、Oracle Flashback Transaction QueryとOracle Flashback Version Queryを併用することで、対象となる行のトランザクションIDを提供します。

関連項目:

[Oracle Flashback Transaction QueryとOracle Flashback Version Queryの併用](#)

DBMS_FLASHBACKパッケージ

この機能は、ある時点の最新データを検査できるよう、Oracle Database内部の時計を過去のある時点まで戻す場合、あるいはデータベースがオンラインである間にトランザクションおよびその依存トランザクションをロールバックする場合に使用できます。

関連項目:

- [フラッシュバック・トランザクション](#)
- [DBMS_FLASHBACKパッケージの使用](#)

フラッシュバック・トランザクション

フラッシュバック・トランザクションは、データベースがオンラインである間にトランザクションおよびその依存トランザクションをロールバックする場合に使用します。このリカバリ操作では、UNDOデータを使用して対応する補正トランザクションを作成および実行します。(フラッシュバック・トランザクションは、DBMS_FLASHBACKパッケージの一部です)。

関連項目:

[DBMS_FLASHBACKパッケージの使用](#)

フラッシュバック・タイム・トラベル

フラッシュバック・タイム・トラベルを使用して、フラッシュバック・アーカイブが有効化された表に対する変更バージョンの履歴を自動的に追跡およびアーカイブすると、スナップショットが古すぎるというエラーが発生することなくデータベース・オブジェクトの各バージョンへのSQLレベルのアクセスが保証されます。

関連項目:

[フラッシュバック時間移動の使用。](#)

19.1.2 データベース管理機能

次のフラッシュバック機能は、主としてデータ・リカバリ用です。通常、この機能は、データベース管理者としてのみ使用します。

この章では、[「アプリケーション開発機能」](#)に重点を置いています。

関連項目:

- [Oracle Database管理者ガイド](#)
- [Oracle Databaseバックアップおよびリカバリ・アドバンスド・ユーザズ・ガイド](#)

Oracle Flashback Table

表の状態を以前の特定の時点までリカバリする場合に、この機能を使用します。データベースがオンラインの間は表のリストアが可能であり、指定した表に対する変更のみ元に戻すことができます。

Oracle Flashback Drop

削除された表をリカバリする場合に、この機能を使用します。この機能によって、DROP TABLE文の影響が無効になります。

Oracle Flashback Database

リカバリ領域を使用してデータベースをその時点まで迅速に戻す場合に、この機能を使用します。データベースのバックアップをリストアする必要がないため、処理が高速です。

19.2 Oracle Flashback Technologyに関するデータベースの構成

アプリケーションでフラッシュバック機能を使用するには、まず、ユーザーまたはデータベース管理者は次の各項で説明する構成タスクを実行する必要があります。

トピック:

- [自動UNDO管理に関するデータベースの構成](#)
- [Oracle Flashback Transaction Queryに関するデータベースの構成](#)
- [フラッシュバック・トランザクションに関するデータベースの構成](#)
- [特定のLOB列に対するOracle Flashback操作の有効化](#)
- [必要な権限の付与](#)

19.2.1 自動UNDO管理に関するデータベースの構成

自動UNDO管理(AUM)に関してデータベースを構成するには、ユーザーまたはデータベース管理者は次を実行する必要があります。

- フラッシュバック操作に必要なデータを保持するために十分な領域を持つUNDO表領域を作成します。

データの更新回数が増加するほど、より多くの領域が必要になります。データベース管理者は通常、必要な領域を計

算します。

- [『Oracle Database管理者ガイド』](#)の説明に従ってAUMを有効にします。次のデータベース初期化パラメータを設定します。
 - UNDO_MANAGEMENT
 - UNDO_TABLESPACE
 - UNDO_RETENTION

固定サイズのUNDO表領域の場合、Oracle Databaseでは自動的にシステムを調整して、UNDO表領域に最適なUNDOを保持します。

自動的に拡張可能なUNDO領域の場合、Oracle Databaseでは、最長の問合せ期間よりも長いUNDOデータ、およびUNDO_RETENTIONパラメータで指定されたUNDO保持の低いしきい値が保持されます。



ノート:

V\$UNDOSTAT.TUNED_UNDORETENTION を問い合せて、現在の UNDO 表領域に対して UNDO が保持される期間を決定できます。

UNDO_RETENTIONの設定は、期限切れになっていないUNDOデータが廃棄されないことを保証するものではありません。システムの領域が不十分な場合、Oracle Databaseでは期限切れになっていないUNDOが新しく生成されたUNDOによって上書きされる場合があります。

- UNDO表領域に対しRETENTION GUARANTEE句を指定して、期限切れになっていないUNDOデータが廃棄されないようにします。

関連項目:

- UNDO表領域の作成とAUMの有効化の詳細は、[『Oracle Database管理者ガイド』](#)を参照してください。
- V\$UNDOSTATの詳細は、[『Oracle Databaseリファレンス』](#)を参照してください。

19.2.2 Oracle Flashback Transaction Queryに関するデータベースの構成

Oracle Flashback Transaction Query機能についてデータベースを構成するには、ユーザーまたはデータベース管理者は次を実行する必要があります。

- Oracle Databaseがバージョン10.0と互換性があることを確認します。
- サプリメンタル・ロギングを有効化します。

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;
```

19.2.3 フラッシュバック・トランザクションに関するデータベースの構成

フラッシュバック・トランザクション機能についてデータベースを構成するには、ユーザーまたはデータベース管理者は次を実行する必要があります。

- データベースがマウントされているが、開かれていない状態で、ARCHIVELOGを有効にします。

```
ALTER DATABASE ARCHIVELOG;
```

- 1つ以上のアーカイブ・ログを開きます。

```
ALTER SYSTEM ARCHIVE LOG CURRENT;
```

- 必要最低限の主キーのサブリメンタル・ロギングが有効になっていない場合、これを有効にします。

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;  
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
```

- 外部キーの依存性を追跡する場合は、外部キーのサブリメンタル・ロギングを有効にします。

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (FOREIGN KEY) COLUMNS;
```

ノート:



外部キーの制約が非常に多い場合、外部キーのサブリメンタル・ロギングを有効にしても、パフォーマンス・ペナルティに見合う効果は得られない可能性があります。

19.2.4 特定のLOB列に対するOracle Flashback操作の有効化

表の特定のLOB列に対するフラッシュバック操作を可能にするには、ALTER TABLE文をRETENTIONオプションとともに使用します。

LOB列のUNDOデータは膨大になる場合があるため、フラッシュバック操作で使用するLOB列を定義する必要があります。

関連項目:

LOB記憶域およびRETENTIONパラメータについて学習するには、[『Oracle Database SecureFilesおよびラージ・オブジェクト開発者ガイド』](#)を参照してください。

19.2.5 必要な権限の付与

ユーザーまたはデータベース管理者は、次のフラッシュバック機能を使用する必要があるユーザー、ロールまたはアプリケーションに対して権限を付与する必要があります。

関連項目:

GRANT文の詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。

Oracle Flashback QueryおよびOracle Flashback Version Query

問合せ中に特定のオブジェクトへのアクセスを許可するには、そのオブジェクトに関するFLASHBACK権限と、READまたはSELECT権限を付与します。

すべての表に対する問合せを許可するには、FLASHBACK ANY TABLE権限を付与します。

Oracle Flashback Transaction Query

SELECT ANY TRANSACTION権限を付与します。

Oracle Flashback Transaction Queryによって取得したUNDO SQLコードの実行を許可するには、特定の表に対するSELECT、UPDATE、DELETEおよびINSERT権限を付与します。

DBMS_FLASHBACKパッケージの機能にアクセスできるようにするには、DBMS_FLASHBACKに対するEXECUTE権限を付与します。

フラッシュバック・タイム・トラベル

特定のユーザーが特定のフラッシュバック・アーカイブを使用して、表に対するフラッシュバック・タイム・トラベルを有効化するには、フラッシュバック・アーカイブに対するFLASHBACK ARCHIVEオブジェクト権限をユーザーに付与します。FLASHBACK ARCHIVEオブジェクト権限を付与するには、SYSDBAとしてログオンしているか、FLASHBACK ARCHIVE ADMINISTERシステム権限が必要です。

次の文の実行を許可するには、FLASHBACK ARCHIVE ADMINISTERシステム権限を付与します。

- CREATE FLASHBACK ARCHIVE
- ALTER FLASHBACK ARCHIVE
- DROP FLASHBACK ARCHIVE

FLASHBACK ARCHIVE ADMINISTERシステム権限を付与するには、SYSDBAとしてログオンしている必要があります。

CREATE FLASHBACK ARCHIVEまたはALTER FLASHBACK ARCHIVE文を使用して、デフォルトのフラッシュバック・アーカイブを作成するには、SYSDBAとしてログオンしている必要があります。

フラッシュバック・アーカイブで有効にされている表のフラッシュバック・アーカイブを無効にするには、SYSDBAとしてログオンしているか、FLASHBACK ARCHIVE ADMINISTERシステム権限が必要です。

19.3 Oracle Flashback Query(SELECT AS OF)の使用

Oracle Flashback Queryを使用するには、SELECT文でAS OF句を使用します。Oracle Flashback Queryにより、過去のある時点で存在していたデータを取得できます。問合せでは、タイムスタンプまたはシステム変更番号(SCN)を使用することで、過去の時点が明示的に参照されます。その時点で最新であったコミット済データが戻されます。

Oracle Flashback Queryは、次のような場合に使用します。

- 失われたデータのリカバリや、誤ったコミット済の変更の取消しを行います。
たとえば、誤って行を削除または更新し、コミットしていても、すぐに誤りを元に戻すことができます。
- 現在のデータを過去の特定の時点の対応するデータと比較する場合。
たとえば、前日からのデータの変更を示す日報を作成できます。表データの個々の行の比較、または行セットの共通部分や結合の検索が可能です。
- 特定の時点でのトランザクション・データの状態を確認する場合。
たとえば、特定の日の預金残高を確認できます。
- 特定の時点、またはユーザー定義の有効期間内に有効なデータを選択します。
たとえば、特定のタイムスタンプの時点、または指定された有効期間内で指定の開始時刻から終了時刻の間の、有効な従業員情報を持つ従業員を見つけることができます。(詳細は、[「時間的な有効性のサポート」](#)を参照してください。)
- アプリケーションの設計を簡略化する場合。一時データの類を格納する必要性をなくすことにより簡略化します。

Oracle Flashback Queryにより、データベースから直接過去のデータを取得できます。

- レポート生成ツールなどのパッケージ・アプリケーションを過去のデータに適用します。
- アプリケーションにセルフサービス・エラー修正を提供し、ユーザーが自分のエラーを元に戻して修正できるようにする場合。

トピック:

- [例: 過去のデータの検査およびリストア](#)
- [Oracle Flashback Queryのガイドライン](#)

関連項目:

SELECT AS OF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。

19.3.1 例: 過去のデータの検査およびリストア

たとえば、午後12時30分に従業員Chungの行がemployees表から削除されていることに気づき、午前9時30分にはこのデータがデータベース内に正しく格納されていたことを知っているとします。この場合、Oracle Flashback Queryを使用して午前9時30分の表の内容を調べ、失われたデータを見つけることができます。必要に応じて失われたデータをリストアできます。

[例19-1](#)では、2004年4月4日午前9時30分におけるChungのレコードの状態が取得されます。

例19-1 Oracle Flashback Queryを使用した失われた行の取得

```
SELECT * FROM employees
AS OF TIMESTAMP
TO_TIMESTAMP('2004-04-04 09:30:00', 'YYYY-MM-DD HH:MI:SS')
WHERE last_name = 'Chung';
```

[例19-2](#)では、Chungの情報がemployees表にリストアされます。

例19-2 Oracle Flashback Query後の失われた行のリストア

```
INSERT INTO employees (
  SELECT * FROM employees
  AS OF TIMESTAMP
  TO_TIMESTAMP('2004-04-04 09:30:00', 'YYYY-MM-DD HH:MI:SS')
  WHERE last_name = 'Chung'
);
```

19.3.2 Oracle Flashback Queryのガイドライン

- 各表に対してAS OF句を指定または省略できます。また、異なる表に異なる時間を指定できます。



ノート:

表がフラッシュバック・タイム・トラベルであり、この表の作成日時よりも過去の日時を指定すると、問合せはこの表に対して、エラーではなく0行を返します。

- 問合せでAS OF句を使用すると、Oracle Flashback Queryと同じセッションで表の作成、切捨てなどのデータ定義言語(DDL)操作、またはINSERT、DELETEなどのデータ操作言語(DML)操作を実行できます。
- Oracle Flashback Queryの結果を、データベースの現在の状態に影響するDDL文またはDML文内で使用するには、INSERT文またはCREATE TABLE AS SELECT文内でAS OF句を使用します。

- 3秒の違い(最大)がアプリケーションにおけるOracle Flashback Queryで重要となる場合、タイムスタンプではなくSCNを使用します。[「Oracle Flashback Technologyの一般ガイドライン」](#)を参照してください。

- ビューを定義するSELECT文のAS OF句を使用して、過去のデータを参照するビューを作成できます。

データベース・ホストの現在の時間から引いて相対時間を指定すると、過去の時間が各問合せで再計算されます。たとえば:

```
CREATE VIEW hour_ago AS
  SELECT * FROM employees
  AS OF TIMESTAMP (SYSTIMESTAMP - INTERVAL '60' MINUTE);
```

SYSTIMESTAMPは、データベース・ホスト環境のタイムゾーンを参照します。

- 自己結合または集合演算(INTERSECT、MINUSなど)のAS OF句は、2つの異なる時間のデータを抽出または比較するために使用できます。

Oracle Flashback Queryの前にCREATE TABLE AS SELECT文またはINSERT INTO TABLE SELECT文を使用すると、結果を格納できます。たとえば、次の問合せでは、1時間前に存在していた行がemployees表に再挿入されます。

```
INSERT INTO employees
  (SELECT * FROM employees
   AS OF TIMESTAMP (SYSTIMESTAMP - INTERVAL '60' MINUTE)
  MINUS SELECT * FROM employees;
```

SYSTIMESTAMPは、データベース・ホスト環境のタイムゾーンを参照します。

- 問合せにAS OF句を使用して、特定の時点に有効であったデータをチェックできます。

関連項目:

- [時間的な有効性のサポート](#)
- [フラッシュバック時間移動の使用](#)(フラッシュバック時間移動の詳細)

19.4 Oracle Flashback Version Queryの使用

特定の時間間隔内に存在していた、特定の行の様々なバージョンを取得するには、Oracle Flashback Version Queryを使用します。COMMIT文が実行されると、そのたびに行バージョンが作成されます。

ノート:



CREATE TABLE 文の実行後、トランザクションをコミットするために少なくとも 15 秒待機して、Oracle Flashback Version Query にそのトランザクションが反映されるようにします。

SELECT文でVERSIONS BETWEEN句を使用して、Oracle Flashback Version Queryを指定します。構文は次のいずれかになります。

```
VERSIONS BETWEEN { SCN | TIMESTAMP } start AND end
```

ここで、startおよびendはそれぞれ、問合せ対象の時間間隔の開始点と終了点を表す式です。時間間隔にはstartおよびendが含まれます。

または

```
VERSIONS PERIOD FOR user_valid_time [ BETWEEN TIMESTAMP start AND end ]
```

ここで、user_valid_timeはユーザー指定の有効時間を示しています。これについては「[時間的な有効性のサポート](#)」で説明しています。

Oracle Flashback Version Queryでは、特定の行について、指定した時間間隔内に存在していたすべてのバージョンの各1行を含む表が戻されます。表の各行には、行バージョンに関するメタデータの疑似列が含まれます。これらの列には、データベースに対して特定の変更(誤りの可能性がある)がいつどのように行われたかが示されます。

[表19-1](#)は、行バージョンに関するメタデータの疑似列を示しています。VERSIONS_*疑似列には、トランザクション時間のFlashback Version Query (BETWEEN TIMESTAMP start AND end句による問合せ)のみに関する値が含まれます。

表19-1 Oracle Flashback Version Queryにおける行データ疑似列

疑似列名	説明:
VERSIONS_STARTSCN	行バージョンが作成されたときの開始システム変更番号(SCN)または TIMESTAMP。この疑似列によって、データの値が最初に行バージョンに反映された時間が識別されます。この疑似列を使用し
VERSIONS_STARTTIME	て、Oracle Flashback Table または Oracle Flashback Query における過去のターゲット時間を指定できます。
	この疑似列が NULL の場合、その行バージョンは、start より前に作成されています。
VERSIONS_ENDSCN	行バージョンが期限切れとなったときの SCN または TIMESTAMP。
VERSIONS_ENDTIME	この疑似列が NULL の場合、問合せの時点で行バージョンが最新であったか、またはその行が DELETE 操作に対応しています。
VERSIONS_XID	その行バージョンが作成されたトランザクションの識別子。
VERSIONS_OPERATION	トランザクションにより実行された操作。I は挿入、D は削除、U は更新を表します。バージョンは、挿入、削除または更新された行のバージョンです。つまり、INSERT 操作後の行、DELETE 操作前の行、または UPDATE 操作の影響を受ける行です。
	ユーザーによる索引キーの更新の場合、Oracle Flashback Version Query では、UPDATE 操作が DELETE および INSERT の 2 つの操作として処理される場合があります。D の後に I の VERSIONS_OPERATION が続く、2 つのバージョンの行で表されます。

指定された行バージョンは、VERSIONS_START*の時間からVERSIONS_END*の時間(ただしこの時間は含まれない)まで有効です。つまり、VERSIONS_START* <= t < VERSIONS_END*である場合、時間tではいつでも有効です。たとえば、次の出力は、2002年9月9日(この日付を含む)から2003年11月25日(この日付は含まない)まで、給与が10243であったことを示します。

VERSIONS_START_TIME	VERSIONS_END_TIME	SALARY
-----	-----	-----
09-SEP-2003	25-NOV-2003	10243

次に、Oracle Flashback Version Queryの一般的な使用方法を示します。

```
SELECT versions_startscn, versions_starttime,
       versions_endscn, versions_endtime,
       versions_xid, versions_operation,
       last_name, salary
FROM employees
VERSIONS BETWEEN TIMESTAMP
   TO_TIMESTAMP('2008-12-18 14:00:00', 'YYYY-MM-DD HH24:MI:SS')
  AND TO_TIMESTAMP('2008-12-18 17:00:00', 'YYYY-MM-DD HH24:MI:SS')
WHERE first_name = 'John';
```

Oracle Flashback Transaction QueryでVERSIONS_XIDを使用して、行の変更を元に戻すために必要なSQL、およびその変更の担当ユーザーなど、このトランザクションのメタデータを検索できます。

フラッシュバック・バージョン問合せでは、IOT(索引構成表)のみを使用した索引専用アクセスが可能です。高速全索引スキャンは使用できません。

関連項目:

- Oracle Flashback Version Queryの疑似列、およびVERSIONS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
- [Oracle Flashback Transaction Queryの使用](#)

19.5 Oracle Flashback Transaction Queryの使用

Oracle Flashback Transaction Queryを使用して、特定の時間間隔内における特定のトランザクションまたはすべてのトランザクションのメタデータおよび履歴データを取得できます。Oracle Flashback Transaction Queryは、静的データ・ディクショナリ・ビューFLASHBACK_TRANSACTION_QUERYの問合せを行います。この表の列の詳細は、[『Oracle Databaseリファレンス』](#)を参照してください。

列UNDO_SQLには、トランザクションで実行されたDML操作の論理的に逆の操作となるSQLコードがあります。通常、このコードを使用して、トランザクション中に実行された論理ステップを元に戻すことができます。ただし、UNDO_SQLのコードが、元のトランザクションの厳密な逆操作ではない場合があります。たとえば、UNDO_SQL INSERT操作で表に行を戻す場合、その行が削除された行と同じROWIDではないことがあります。

次の文では、トランザクションID、操作、操作の開始SCNおよび終了SCN、操作の担当ユーザー、および操作の論理的な逆操作を示すSQLコードなどのトランザクション情報について、FLASHBACK_TRANSACTION_QUERYビューを問い合わせています。

```
SELECT xid, operation, start_scn, commit_scn, logon_user, undo_sql
FROM flashback_transaction_query
WHERE xid = HEXTORAW('0002000300000002D');
```

次の文では、Oracle Flashback Version Queryを副問合せとして使用し、各行バージョンを、行データの変更を担当するLOGON_USERに関連付けています。

```
SELECT xid, logon_user
FROM flashback_transaction_query
WHERE xid IN (
  SELECT versions_xid FROM employees VERSIONS BETWEEN TIMESTAMP
    TO_TIMESTAMP('2003-07-18 14:00:00', 'YYYY-MM-DD HH24:MI:SS') AND
    TO_TIMESTAMP('2003-07-18 17:00:00', 'YYYY-MM-DD HH24:MI:SS')
);
```

ノート:



XID を WHERE 句に指定せずに FLASHBACK_TRANSACTION_QUERY を問い合わせると、問合せによって多数の無関係の行がスキャンされるため、パフォーマンスが低下します。

関連項目:

- データベース管理者がフラッシュバック表を使用して、個々の行ではなく表全体をリストアする方法の詳細は、『[Oracle Databaseバックアップおよびリカバリ・ユーザズ・ガイド](#)』を参照してください。
- データベース管理者がフラッシュバック表を使用して、個々の行ではなく表全体をリストアする方法の詳細は、『[Oracle Database管理者ガイド](#)』を参照してください。

19.6 Oracle Flashback Transaction QueryとOracle Flashback Version Queryの併用

この例では、データベース管理者は次のコマンドを実行します。

```
DROP TABLE emp;  
CREATE TABLE emp (  
    empno    NUMBER PRIMARY KEY,  
    empname  VARCHAR2(16),  
    salary   NUMBER  
);  
INSERT INTO emp (empno, empname, salary) VALUES (111, 'Mike', 555);  
COMMIT;  
DROP TABLE dept;  
CREATE TABLE dept (  
    deptno   NUMBER,  
    deptname VARCHAR2(32)  
);  
INSERT INTO dept (deptno, deptname) VALUES (10, 'Accounting');  
COMMIT;
```

この時点で、empおよびdeptにはそれぞれ1行があります。行バージョンに関しては、各表に1つのバージョンを示す1行が含まれます。誤ったトランザクションにより、empno 111が表empから削除されたとします。

```
UPDATE emp SET salary = salary + 100 WHERE empno = 111;  
INSERT INTO dept (deptno, deptname) VALUES (20, 'Finance');  
DELETE FROM emp WHERE empno = 111;  
COMMIT;
```

次に、トランザクションにより、新規従業員名とともにempno 111が表empに再挿入されます。

```
INSERT INTO emp (empno, empname, salary) VALUES (111, 'Tom', 777);  
UPDATE emp SET salary = salary + 100 WHERE empno = 111;  
UPDATE emp SET salary = salary + 50 WHERE empno = 111;  
COMMIT;
```

データベース管理者はアプリケーション・エラーを検出したため、問題を診断する必要があります。データベース管理者は次の問合せを発行し、empno 111に対応するemp表の行のバージョンを取得します。この問合せでは、Oracle Flashback Version Queryの疑似列が使用されます。

```
SELECT versions_xid XID, versions_startscn START_SCN,
```

```
versions_endscn END_SCN, versions_operation OPERATION,
empname, salary
FROM emp
VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
WHERE empno = 111;
```

次に類似した結果が得られます。

XID	START_SCN	END_SCN	O	EMPNAME	SALARY
09001100B2200000	10093466		I	Tom	927
030002002B210000	10093459		D	Mike	555
0800120096200000	10093375	10093459	I	Mike	555

3 rows selected.

結果表の行は降順の時系列で示しています。3行目は、表が作成されたときに表に挿入された、表empの行のバージョンに対応しています。2行目は、誤ったトランザクションにより削除されたempの行に対応します。1行目は、新規従業員名とともに再挿入されたempの行のバージョンに対応します。

データベース管理者は、トランザクション030002002B210000を誤ったトランザクションとして識別し、Oracle Flashback Transaction Queryを使用して、このトランザクションによるすべての変更を監査します。

```
SELECT xid, start_scn, commit_scn, operation, logon_user, undo_sql
FROM flashback_transaction_query
WHERE xid = HEXTORAW('0002000300000002D');
```

次に類似した結果が得られます。

XID	START_SCN	COMMIT_SCN	OPERATION	LOGON_USER	UNDO_SQL
030002002B210000	10093452	10093459	DELETE	HR	insert into "HR"."EMP"("EMPNO","EMPNAME","SALARY") values ('111','Mike','655');
030002002B210000	10093452	10093459	INSERT	HR	delete from "HR"."DEPT" where ROWID = 'AAATjuAAEAAAAJrAAB';
030002002B210000	10093452	10093459	UPDATE	HR	update "HR"."EMP" set "SALARY" = '555' where ROWID = 'AAATjsAAEAAAAJ7AAA';
030002002B210000	10093452	10093459	BEGIN	HR	

4 rows selected.

次に行われる問合せの結果を読みやすくするために、データベース管理者は次のSQL*Plusコマンドを使用します。

```
COLUMN operation FORMAT A9
COLUMN table_name FORMAT A10
COLUMN table_owner FORMAT A11
```

誤ったトランザクションおよびそれ以降のすべてのトランザクションを確認するには、データベース管理者は次の問合せを実行します。

```
SELECT xid, start_scn, commit_scn, operation, table_name, table_owner
FROM flashback_transaction_query
WHERE table_owner = 'HR'
AND start_timestamp >=
TO_TIMESTAMP ('2002-04-16 11:00:00', 'YYYY-MM-DD HH:MI:SS');
```

次に類似した結果が得られます。

XID	START_SCN	COMMIT_SCN	OPERATION	TABLE_NAME	TABLE_OWNER
02000E0074200000	10093435	10093446	INSERT	DEPT	HR
030002002B210000	10093452	10093459	DELETE	EMP	HR
030002002B210000	10093452	10093459	INSERT	DEPT	HR
030002002B210000	10093452	10093459	UPDATE	EMP	HR
0800120096200000	10093374	10093375	INSERT	EMP	HR
09001100B2200000	10093462	10093466	UPDATE	EMP	HR
09001100B2200000	10093462	10093466	UPDATE	EMP	HR
09001100B2200000	10093462	10093466	INSERT	EMP	HR

8 rows selected.

ノート:



この問合せでは XID が WHERE 句に指定されていないため、多数の無関係の行をスキャンし、パフォーマンスが低下します。

19.7 DBMS_FLASHBACKパッケージの使用

DBMS_FLASHBACKパッケージでもOracle Flashback Queryと同じ機能が提供されますが、Oracle Flashback Queryのほうが便利な場合があります。

DBMS_FLASHBACKパッケージはタイムマシンのように機能します。つまり、時計を特定の時点に戻して、過去のその時点にいるかのように通常の問合せを実行し、その後現在に戻ることができます。DBMS_FLASHBACKパッケージを使用して、AS OFやVERSIONS BETWEENなどの特別な句を使用せずに過去のデータに問合せを実行できるため、既存のPL/SQLコードを再使用して、過去の時点のデータベースを問い合わせることができます。

DBMS_FLASHBACKパッケージのEXECUTE権限が必要です。

PL/SQLコードでDBMS_FLASHBACKパッケージを使用するには:

1. DBMS_FLASHBACK.ENABLE_AT_TIMEまたは
DBMS_FLASHBACK.ENABLE_AT_SYSTEM_CHANGE_NUMBERを起動して、過去の時間を指定します。
2. 通常の問合せ(AS OFなど、特別なフラッシュバック機能の構文を使用しない)を実行します。DDL操作やDML操作は実行しません。

指定された過去の時間におけるデータベースに対し、問合せが行われます。

3. DBMS_FLASHBACK.DISABLEを起動して現在に戻ります。

再度DBMS_FLASHBACK.ENABLE_AT_TIMEまたは
DBMS_FLASHBACK.ENABLE_AT_SYSTEM_CHANGE_NUMBERを起動する前に、
DBMS_FLASHBACK.DISABLEを起動する必要があります。ENABLE/DISABLEのペアはネストできません。

カーソルを使用して問合せの結果を格納するには、カーソルをオープンしてからDBMS_FLASHBACK.DISABLEを起動します。結果を格納し、DBMS_FLASHBACK.DISABLEを起動した後、次のことができます。

- INSERTまたはUPDATE操作を実行し、過去のデータベースの格納結果を使用して現在のデータベースの状態を変更します。
- 現在のデータを過去のデータと比較します。DBMS_FLASHBACK.DISABLEを起動した後、2番目のカーソルをオーブ

ンします。過去のデータを取得するには最初のカーソルからフェッチし、現在のデータを取得するには2番目のカーソルからフェッチします。過去のデータを一時表に格納し、その後MINUSやUNIONなどの集合演算子を使用して、過去のデータと現在のデータを比較または結合できます。

DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBERを起動して、いつでも現在のシステム変更番号(SCN)を取得できます。事前のDBMS_FLASHBACK.ENABLEの起動とは関係なく、DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBERでは、現在のSCNが戻されます。

関連項目:

- DBMS_FLASHBACKパッケージの詳細は、[『Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス』](#)を参照してください。

19.8 フラッシュバック・トランザクションの使用

DBMS_FLASHBACK.TRANSACTION_BACKOUTプロシージャでは、データベースがオンラインである間にトランザクションおよびその依存トランザクションをロールバックします。このリカバリ操作では、UNDOデータを使用して対応する補正トランザクションを作成および実行します。このトランザクションによって影響のあったデータが元の状態に戻ります。

ロールバック中のトランザクションは、次の制約を受けます。

- トランザクションは、データベース表の論理構造を変更するDDL操作を実行できません。
- 次のラージ・オブジェクト(LOB)データ型を使用できません。
 - BFILE
 - BLOB
 - CLOB
 - NCLOB
- LogMinerがサポートしていない機能を使用できません。

LogMinerがサポートする機能は、トランザクションをロールバックしているデータベースのCOMPATIBLE初期化パラメータの値によって異なります。デフォルト値は、最新メジャー・リリースのリリース番号です。

トランザクションのフラッシュバックは、LogMinerからSQLデータ型サポートを継承します。したがって、トランザクションにサポートされていないSQLデータ型が含まれていたことが原因でLogMinerで障害が発生した場合、トランザクションのフラッシュバックでも障害が発生します。

一部のデータ型は、LogMinerではサポートされていますが、このような型の列を修正する操作の一端として、Undo情報を生成しません。したがって、トランザクションのフラッシュバックは、このようなデータ型を含む表をサポートしません。これには、BLOB型、CLOB型、およびXML型を含む表が該当します。

関連項目:

- ロジカル・スタンバイ・データベースでのデータ型およびDDLサポートに関する詳細は、[『Oracle Data Guard概要および管理』](#)を参照してください。
- LOBデータ型の詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。
- LogMinerの詳細は、[『Oracle Databaseユーティリティ』](#)を参照してください。

- COMPATIBLE初期化パラメータの詳細は、[『Oracle Database管理者ガイド』](#)を参照してください。

トピック:

- [依存トランザクション](#)
- [TRANSACTION_BACKOUTパラメータ](#)
- [TRANSACTION_BACKOUTレポート](#)

19.8.1 依存トランザクション

トランザクションのフラッシュバックのコンテキストでは、トランザクション2は、次のいずれかの方法で、トランザクション1に依存します。

- Write-after-Write依存性

トランザクション1は表の行を変更し、後からトランザクション2が同じ行を変更します。

- 主キーの依存性

ある表の列cには主キー制約が含まれます。この表のある行の列cには値vが入っています。トランザクション1はこの行を削除し、その後、トランザクション2は同じ表に行を1つ挿入し、値vを列cに割り当てます。

- 外部キーの依存性

表bの列b1には、表aの列a1にある外部キー制約が含まれます。トランザクション1はa1の値を変更し、後に、トランザクション2がb1の値を変更します。

19.8.2 TRANSACTION_BACKOUTパラメータ

TRANSACTION_BACKOUTプロシージャのパラメータは次のとおりです。

- バックアウトするトランザクションの数
- 名前またはXIDで識別されたバックアウトするトランザクションのリスト
- 時間ヒント(名前でトランザクションを識別する場合)
任意のトランザクションの開始時間より早い時間を指定します。
- [表19-2](#)のバックアウト・オプション

表19-2 フラッシュバックTRANSACTION_BACKOUTオプション

オプション	説明:
CASCADE	後行順で指定トランザクションおよびすべての依存トランザクションをバックアウトします(つまり、親がバックアウトされる前に子がバックアウトされます)。 CASCADE がない場合、依存トランザクションが指定されていないとエラーが発生します。
NOCASCADE	デフォルトです。依存トランザクションを持たないと予想される指定トランザクションをバックアウトします。最初の依存トランザクションによってエラーが発生し、*_FLASHBACK_TXN_REPORT に表示されます。

オプション	説明:
NOCASCADE_FORCE	依存トランザクションを無視して、指定トランザクションをバックアウトします。サーバーでは、コミット時間とは逆の順序で指定トランザクションの UNDO SQL 文を実行します。 制約に違反せず、その結果に満足している場合は、変更をコミットできます。それ以外の場合は、ロールバックできます。
NONCONFLICT_ONLY	指定トランザクションの競合しない行に変更をバックアウトします。データベースの一貫性は保たれますが、トランザクションは完全に失われます。

TRANSACTION_BACKOUTではトランザクションの依存性を分析してDML操作を実行し、レポートを生成します。TRANSACTION_BACKOUTではトランザクション・バックアウトの一環として実行されるDML操作はコミットしませんが、正しい形式で行および表に対する必要なロックを保持し、他の依存関係がシステムに介入しないようにします。トランザクション・バックアウトを永続的なものにするには、明示的にトランザクションをコミットする必要があります。

関連項目:

TRANSACTION_BACKOUTプロシージャの構文、およびパラメータの詳細な説明は、[『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。

19.8.3 TRANSACTION_BACKOUTレポート

TRANSACTION_BACKOUTによって生成されたレポートを参照するには、静的データ・ディクショナリ・ビュー *_FLASHBACK_TXN_STATEおよび*_FLASHBACK_TXN_REPORTを問い合わせます。

19.8.3.1 *_FLASHBACK_TXN_STATE

静的データ・ディクショナリ・ビュー *_FLASHBACK_TXN_STATEでは、トランザクションがアクティブであるか、またはバックアウトされているかが示されます。このビューにトランザクションが表示される場合は、バックアウトされています。

*_FLASHBACK_TXN_STATEは、補正トランザクションに対してアトミックにメンテナンスされます。補正トランザクションがバックアウトされている場合、加えられたすべての変更もバックアウトされ、*_FLASHBACK_TXN_STATEにこれが反映されます。たとえば、補正トランザクションctがトランザクションt1およびt2をバックアウトする場合、t1およびt2は

*_FLASHBACK_TXN_STATEに表示されます。ct自身が後でバックアウトされる場合、t1およびt2による影響は元に戻され、t1およびt2は*_FLASHBACK_TXN_STATEに表示されなくなります。

関連項目:

*_FLASHBACK_TXN_STATEの詳細は、[『Oracle Databaseリファレンス』](#)を参照してください。

19.8.3.2 *_FLASHBACK_TXN_REPORT

静的データ・ディクショナリ・ビュー *_FLASHBACK_TXN_REPORTでは、各バックアウト・トランザクションの詳細レポートを生成します。

関連項目:

*_FLASHBACK_TXN_REPORTの詳細は、『[Oracle Databaseリファレンス](#)』を参照してください。

19.9 フラッシュバック・タイム・トラベルの使用

フラッシュバック時間移動により、その存続期間中、表に対するすべてのトランザクションによる変更を追跡および格納できます。これは、レコード・ステージ・ポリシーおよび監査レポートへの準拠に役立ちます。

フラッシュバック・アーカイブは、1つ以上の表領域またはその一部で構成されています。フラッシュバック・アーカイブは複数所有できます。SYSDBAとしてログオンしている場合は、システムに対してデフォルトのフラッシュバック・アーカイブを指定できます。フラッシュバック・アーカイブは、保有時間を使用して構成されます。フラッシュバック・アーカイブにアーカイブされたデータは、フラッシュバック・アーカイブ作成時に指定した保有時間まで保持されます。

デフォルトでは、すべての表についてフラッシュバック・アーカイブは有効になっていません。次のすべてに該当する場合は表のフラッシュバック・アーカイブを有効にすることができます。

- 表に対して使用するフラッシュ・アーカイブのFLASHBACK ARCHIVEオブジェクト権限がある。
- 表が、ネスト表、一時表、リモート表または外部表でない。
- 表にLONG列またはネストされた列が含まれない。
- 表で、次のいずれかのフラッシュバック・アーカイブ予約語が列名として使用されていない。
 - STARTSCN
 - ENDSCN
 - RID
 - XID
 - OP
 - OPERATION

ネスト表、一時表、外部表、マテリアライズド・ビュー、AQ (Advanced Query)表または表以外のオブジェクトで、フラッシュバック・アーカイブを有効にすることはできません。

表に対してフラッシュバック・アーカイブを有効にした後は、FLASHBACK ARCHIVE ADMINISTERシステム権限がある場合、またはSYSDBAとしてログオンしている場合のみ、これを無効にできます。

特定の表に対してフラッシュバック・アーカイブを選択する場合は、その表のデータ保有要件およびFLASHBACK ARCHIVEオブジェクト権限のあるフラッシュバック・アーカイブの保有時間を考慮してください。

Oracle Database 12cリリース1 (12.1.0.1)以降、フラッシュバック・アーカイブは拡張されており、次のものを備えています。

- ユーザー・コンテキストのトラッキング。トランザクションのトラッキングに関するメタデータ情報は、ユーザー・コンテキストを含むことができます(機能が有効になっている場合)。これによって、どのユーザーが何の変更を表に加えたかを確認しやすくなります。

ユーザー・コンテキスト・レベル(保存するユーザー・コンテキストの量を決定)を設定するには、

[DBMS_FLASHBACK_ARCHIVE.SET_CONTEXT_LEVEL](#)プロシージャを使用します。コンテキスト情報にアクセスするには、[DBMS_FLASHBACK_ARCHIVE.GET_SYS_CONTEXT](#)ファンクションを使用します。

- データベース強化。この機能によって、アプリケーション内で一連の表を関連付けて、それらすべての表に対して1つのコマンドでフラッシュバック・アーカイブを有効にすることができます。また、データベース強化により、すべての表を1つのコマンドでロックできます。後でロック解除するまでは、それらの表に対するDMLを防ぐことができます。データベース強化は、ア

アプリケーション内のセキュリティ上の重要な表を、フラッシュバック時間移動で容易に追跡および保護することを目的としています。

データベース強化のためにアプリケーションを登録するには、

[DBMS_FLASHBACK_ARCHIVE.REGISTER_APPLICATION](#) プロシージャを使用します。この詳細は、『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。

トピック:

- [フラッシュバック・アーカイブの作成](#)
- [フラッシュバック・アーカイブの変更](#)
- [フラッシュバック・アーカイブの削除](#)
- [デフォルトのフラッシュバック・アーカイブの指定](#)
- [フラッシュバック・アーカイブの有効化および無効化](#)
- [フラッシュバック・アーカイブが有効な表に対するDDL文](#)
- [フラッシュバック・アーカイブ・データの表示](#)
- [フラッシュバック時間移動のシナリオ](#)

関連項目:

DBMS_FLASHBACK_ARCHIVEパッケージの詳細は、『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』を参照してください。

19.9.1 フラッシュバック・アーカイブの作成

フラッシュバック・アーカイブは、CREATE FLASHBACK ARCHIVE文を使用して作成できます。

次を指定し、CREATE FLASHBACK ARCHIVE文を使用して、フラッシュバック・アーカイブを作成します。

- フラッシュバック・アーカイブの名前
- フラッシュバック・アーカイブの最初の表領域の名前
- (オプション)フラッシュバック・アーカイブが最初の表領域で使用できる最大領域量
デフォルトは無制限。最初の表領域の領域割当て制限も無制限でないかぎり、この値を指定する必要があります。指定しない場合、ORA-55621のエラーが発生します。
- 保有時間(表のフラッシュバック・アーカイブのデータの格納が保証されている日数)
- (オプション)フラッシュバック・アーカイブで管理する履歴表のデータの記憶域を最適化するかどうか([NO] OPTIMIZE DATAを使用)。

デフォルトはNO OPTIMIZE DATAです。

SYSDBAとしてログオンしている場合は、システムに対してこれをデフォルトのフラッシュバック・アーカイブとして指定することもできます。このオプションを省略した場合でも、後でこのフラッシュバック・アーカイブをデフォルトにできます。

フラッシュバック・アーカイブを使用する必要があるすべてのユーザーに対しては、フラッシュバック・アーカイブ表領域の割当て制限を無制限にすることをお薦めします。このようにできない場合は、それらのユーザーに対して十分な表領域の割当て制限を与え

る必要があります。

例

- デフォルトのフラッシュバック・アーカイブfla1を作成します。これは最大10GBの表領域tbs1を使用し、そのデータは1年間保持されます。

```
CREATE FLASHBACK ARCHIVE DEFAULT fla1 TABLESPACE tbs1
QUOTA 10G RETENTION 1 YEAR;
```

- フラッシュバック・アーカイブfla2を作成します。これは表領域tbs2を使用し、そのデータは2年間保持されます。

```
CREATE FLASHBACK ARCHIVE fla2 TABLESPACE tbs2 RETENTION 2 YEAR;
```

関連項目:

- CREATE FLASHBACK ARCHIVE文の構文の詳細は、『[Oracle Database SQL言語リファレンス](#)』を参照してください
- [デフォルトのフラッシュバック・アーカイブの指定](#)

19.9.2 フラッシュバック・アーカイブの変更

フラッシュバック・アーカイブを変更するには、ALTER FLASHBACK ARCHIVE文を使用します。

ALTER FLASHBACK ARCHIVE文を使用して、次を実行できます。

- フラッシュバック・アーカイブの保有時間を変更します
- データの一部またはすべてをページします。
- 表領域を追加、変更、削除します。



ノート:

フラッシュバック・アーカイブのすべての表領域を削除すると、エラーが発生します。

SYSDBAとしてログオンしている場合は、ALTER FLASHBACK ARCHIVE文を使用して、特定のファイルをシステムに対するデフォルトのフラッシュバック・アーカイブにすることもできます。

例

- フラッシュバック・アーカイブfla1を、デフォルトのフラッシュバック・アーカイブにします。

```
ALTER FLASHBACK ARCHIVE fla1 SET DEFAULT;
```

- フラッシュバック・アーカイブfla1に対して、最大5GBの表領域tbs3を追加します。

```
ALTER FLASHBACK ARCHIVE fla1 ADD TABLESPACE tbs3 QUOTA 5G;
```

- フラッシュバック・アーカイブfla1に対して、必要に応じてtbs4と同じサイズの表領域を追加します。

```
ALTER FLASHBACK ARCHIVE fla1 ADD TABLESPACE tbs4;
```

- フラッシュバック・アーカイブfla1が表領域tbs3で利用できる最大領域を20GBに変更します。

```
ALTER FLASHBACK ARCHIVE fla1 MODIFY TABLESPACE tbs3 QUOTA 20G;
```

- フラッシュバック・アーカイブfla1に対して、必要に応じてtbs1と同じサイズの表領域を使用できるようにします。

```
ALTER FLASHBACK ARCHIVE fla1 MODIFY TABLESPACE tbs1;
```

- フラッシュバック・アーカイブfla1の保有時間を2年に変更します。

```
ALTER FLASHBACK ARCHIVE fla1 MODIFY RETENTION 2 YEAR;
```

- フラッシュバック・アーカイブfla1から表領域tbs2を削除します。

```
ALTER FLASHBACK ARCHIVE fla1 REMOVE TABLESPACE tbs2;
```

(表領域tbs2は削除されません。)

- フラッシュバック・アーカイブfla1からすべての履歴データをパージします。

```
ALTER FLASHBACK ARCHIVE fla1 PURGE ALL;
```

- フラッシュバック・アーカイブfla1から1日以上経過したすべての履歴データをパージします。

```
ALTER FLASHBACK ARCHIVE fla1  
PURGE BEFORE TIMESTAMP (SYSTIMESTAMP - INTERVAL '1' DAY);
```

- フラッシュバック・アーカイブfla1からSCN 728969より古いすべての履歴データをパージします。

```
ALTER FLASHBACK ARCHIVE fla1 PURGE BEFORE SCN 728969;
```

関連項目:

ALTER FLASHBACK ARCHIVE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。

19.9.3 フラッシュバック・アーカイブの削除

フラッシュバック・アーカイブを削除するには、DROP FLASHBACK ARCHIVE文を使用します。

フラッシュバック・アーカイブを削除するとその履歴データは削除されますが、その表領域は削除されません。

例

表領域を削除せず、フラッシュバック・アーカイブfla1およびそのすべての履歴データを削除します。

```
DROP FLASHBACK ARCHIVE fla1;
```

DROP FLASHBACK ARCHIVE文の構文の詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください

19.9.4 デフォルトのフラッシュバック・アーカイブの指定

デフォルトのフラッシュバック・アーカイブを指定するには、CREATEまたはALTER FLASHBACK ARCHIVE文を使用します。

システムのデフォルト・フラッシュバック・アーカイブは、自分自身のデフォルト・フラッシュバック・アーカイブを持たないすべてのユーザーに対するデフォルト・フラッシュバック・アーカイブです。

デフォルトでは、システムにはデフォルト・フラッシュバック・アーカイブがありません。SYSDBAとしてログオンしている場合は、次のいずれかの方法でデフォルト・フラッシュバック・アーカイブを指定できます。

- ALTER FLASHBACK ARCHIVE文のSET DEFAULT句を使用して、既存のフラッシュバック・アーカイブの名前を指

定します。たとえば:

```
ALTER FLASHBACK ARCHIVE fla1 SET DEFAULT;
```

fla1がない場合は、エラーが発生します。

- フラッシュバック・アーカイブを作成するときに、CREATE FLASHBACK ARCHIVE文にDEFAULTを含めます。たとえば:

```
CREATE FLASHBACK ARCHIVE DEFAULT fla2 TABLESPACE tbs1  
QUOTA 10G RETENTION 1 YEAR;
```

関連項目:

- CREATE FLASHBACK ARCHIVE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。
- ALTER DATABASE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。

19.9.5 フラッシュバック・アーカイブの有効化および無効化

デフォルトでは、フラッシュバック・アーカイブはすべての表に対して無効になっています。その表に対して使用するフラッシュバック・アーカイブのFLASHBACK ARCHIVEオブジェクト権限がある場合は、表のフラッシュバック・アーカイブを有効にできます。

表に対してフラッシュバック・アーカイブを有効にするには、CREATE TABLE文またはALTER TABLE文にFLASHBACK ARCHIVE句を含めます。FLASHBACK ARCHIVE句には、その表の履歴データを格納するフラッシュバック・アーカイブを指定できます。デフォルトは、システムのデフォルト・フラッシュバック・アーカイブです。存在しないフラッシュバック・アーカイブを指定すると、エラーが発生します。

表でフラッシュバック・アーカイブが有効になっている場合は、別のフラッシュバック・アーカイブを再度有効にしようとするとエラーが発生します。

表に対してフラッシュバック・アーカイブを有効にした後は、FLASHBACK ARCHIVE ADMINISTERシステム権限がある場合、またはSYSDBAとしてログオンしている場合のみ、これを無効にできます。表に対してフラッシュバック・アーカイブを無効にするには、ALTER TABLE文にNO FLASHBACK ARCHIVEを指定します。(CREATE TABLE文にNO FLASHBACK ARCHIVEを指定する必要はありません。これはデフォルトです。)

関連項目:

CREATE TABLE文のFLASHBACK ARCHIVE句の詳細(使用制限も含む)は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。

例

- 表employeeを作成してデフォルトのフラッシュバック・アーカイブに履歴データを格納します。

```
CREATE TABLE employee (EMPNO NUMBER(4) NOT NULL, ENAME VARCHAR2(10),  
JOB VARCHAR2(9), MGR NUMBER(4)) FLASHBACK ARCHIVE;
```

- 表employeeを作成してフラッシュバック・アーカイブfla1に履歴データを格納します。

```
CREATE TABLE employee (EMPNO NUMBER(4) NOT NULL, ENAME VARCHAR2(10),  
JOB VARCHAR2(9), MGR NUMBER(4)) FLASHBACK ARCHIVE fla1;
```

- 表employeeのフラッシュバック・アーカイブを有効にして、デフォルトのフラッシュバック・アーカイブに履歴データを格納します。

```
ALTER TABLE employee FLASHBACK ARCHIVE;
```

- 表employeeのフラッシュバック・アーカイブを有効にして、フラッシュバック・アーカイブfla1に履歴データを格納します。

```
ALTER TABLE employee FLASHBACK ARCHIVE fla1;
```

- 表employeeのフラッシュバック・アーカイブを無効にします。

```
ALTER TABLE employee NO FLASHBACK ARCHIVE;
```

19.9.6 フラッシュバック・アーカイブが有効な表に対するDDL文

フラッシュバック・アーカイブは、次のDDL文のみをサポートしています。

- ALTER TABLE文では、次のいずれかが実行されます。
 - 列の追加、削除、名前変更、または修正
 - 制約の追加、削除、または名前変更
 - パーティション、またはサブパーティション操作の削除、または切捨て
- TRUNCATE TABLE文
- 表の名前を変更するRENAME文

フラッシュバック・アーカイブは、パーティションまたはサブパーティションを移動、分割、マージまたは結合したり、表を移動したり、LONG列をLOB列に変換するDDL文はサポートしていません。

たとえば、フラッシュバック・アーカイブを有効にした表に対して次のDDL文を使用すると、エラーORA-55610が発生します。

- UPGRADE TABLE句を含むALTER TABLE文。INCLUDING DATA句を持つことも持たないこともあります。
- パーティション、またはサブパーティション操作を移動、または交換するALTER TABLE文。
- DROP TABLE文

フラッシュバック・アーカイブが有効な表でサポートされていないDDL文を使用する必要がある場合は、DBMS_FLASHBACK_ARCHIVE.DISASSOCIATE_FBAプロシージャを使用して、そのフラッシュバック・アーカイブから実表の関連付けを解除します。その後、フラッシュバック・アーカイブと実表を再度関連付けるには、DBMS_FLASHBACK_ARCHIVE.REASSOCIATE_FBAプロシージャを使用します。また、フラッシュバック・アーカイブが有効な表を削除するには、最初にALTER TABLE ... NO FLASHBACK ARCHIVE句を使用して、表でフラッシュバック・アーカイブを無効にする必要があります。

関連項目:

- ALTER TABLE文の詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。
- TRUNCATE TABLE文の詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。
- RENAME文の詳細は、[Oracle Database SQLリファレンス](#)を参照
- DROP TABLE文の詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。
- DBMS_FLASHBACK_ARCHIVE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。

19.9.7 フラッシュバック・アーカイブ・データの表示

フラッシュバック・アーカイブ・ファイルに関する情報は、静的データ・ディクショナリ・ビューで表示できます。

表19-3 フラッシュバック・アーカイブ・ファイルの静的データ・ディクショナリ・ビュー

ビュー	説明:
*_FLASHBACK_ARCHIVE	フラッシュバック・アーカイブ・ファイルに関する情報を表示します
*_FLASHBACK_ARCHIVE_TS	フラッシュバック・アーカイブ・ファイルの表領域を表示します
*_FLASHBACK_ARCHIVE_TABLE S	フラッシュバック・アーカイブが有効な表に関する情報を表示します

関連項目:

- *_FLASHBACK_ARCHIVEの詳細は、『[Oracle Databaseリファレンス](#)』を参照してください。
- *_FLASHBACK_ARCHIVE_TSの詳細は、『[Oracle Databaseリファレンス](#)』を参照してください。
- *_FLASHBACK_ARCHIVE_TABLESの詳細は、『[Oracle Databaseリファレンス](#)』を参照してください。

19.9.8 フラッシュバック・タイム・トラベルのシナリオ

- [シナリオ: フラッシュバック・タイム・トラベルを使用したデジタル・シュレディングの実行](#)
- [シナリオ: フラッシュバック・タイム・トラベルを使用した履歴データへのアクセス](#)
- [シナリオ: フラッシュバック・タイム・トラベルを使用したレポートの生成](#)
- [シナリオ: フラッシュバック・タイム・トラベルの監査への使用](#)
- [シナリオ: フラッシュバック・タイム・トラベルを使用したデータのリカバリ](#)

19.9.8.1 シナリオ: フラッシュバック・タイム・トラベルを使用したデジタル・シュレディングの実行

企業でTaxes表に対する履歴データの変更内容を10年後に「シュレッド」(削除)とします。Taxesのフラッシュバック・アーカイブを作成する際、10年の保有時間を指定します。

```
CREATE FLASHBACK ARCHIVE taxes_archive TABLESPACE tbs1 RETENTION 10 YEAR;
```

Taxesのトランザクションからの履歴データは、10年を超えるとページされます。(Taxesの表自体、および10年未満のトランザクションからの履歴データはページされません。)

19.9.8.2 シナリオ: フラッシュバック・タイム・トラベルを使用した履歴データへのアクセス

表inventoryから年初にすべての品目の在庫を取得できるように、また、表stock_dataからその年の任意の日付の業務終了時点での自社の有価証券における、各シンボルの株価を取得できるようにとします。

デフォルトのフラッシュバック・アーカイブfla1を作成します。これは最大10GBの表領域tbs1を使用し、そのデータは5年間保持されます(SYSDBAとしてログオンしている必要があります)。

```
CREATE FLASHBACK ARCHIVE DEFAULT fla1 TABLESPACE tbs1
```



```
QUOTA 10G RETENTION 5 YEAR;
```

表inventoryおよびstock_dataのフラッシュバック・アーカイブを有効にして、デフォルトのフラッシュバック・アーカイブに履歴データを格納します。

```
ALTER TABLE inventory FLASHBACK ARCHIVE;  
ALTER TABLE stock_data FLASHBACK ARCHIVE;
```

2007年の年初にすべての品目の在庫を取得するには、次の問合せを使用します。

```
SELECT product_number, product_name, count FROM inventory AS OF  
TIMESTAMP TO_TIMESTAMP ('2007-01-01 00:00:00', 'YYYY-MM-DD HH24:MI:SS');
```

2007年7月23日の業務の終了時に有価証券の各シンボルの株価を取得するには、次の問合せを使用します。

```
SELECT symbol, stock_price FROM stock_data AS OF  
TIMESTAMP TO_TIMESTAMP ('2007-07-23 16:00:00', 'YYYY-MM-DD HH24:MI:SS')  
WHERE symbol IN my_portfolio;
```

19.9.8.3 シナリオ: フラッシュバック・タイム・トラベルを使用したレポートの生成

ユーザーが、過去5年間格納されているデータについて、表investmentsからレポートを生成できるようにするとします。

デフォルトのフラッシュバック・アーカイブfla2を作成します。これは最大20GBの表領域tbs1を使用し、そのデータは5年間保持されます(SYSDBAとしてログオンしている必要があります)。

```
CREATE FLASHBACK ARCHIVE DEFAULT fla2 TABLESPACE tbs1  
QUOTA 20G RETENTION 5 YEAR;
```

表investmentsのフラッシュバック・アーカイブを有効にして、デフォルトのフラッシュバック・アーカイブに履歴データを格納します。

```
ALTER TABLE investments FLASHBACK ARCHIVE;
```

Lisaは、2006年12月31日の業務終了時点での投資の運用成績に関するレポートを必要としています。次の問合せを使用します。

```
SELECT * FROM investments AS OF  
TIMESTAMP TO_TIMESTAMP ('2006-12-31 16:00:00', 'YYYY-MM-DD HH24:MI:SS')  
WHERE name = 'LISA';
```

19.9.8.4 シナリオ: フラッシュバック・タイム・トラベルの監査への使用

医療保険会社では、診療所を監査する必要があります。医療保険会社は表Billingsにその債権があり、デフォルトのフラッシュバック・アーカイブfla4を作成します。これは最大100GBの表領域tbs1を使用し、そのデータは10年間保持されます。

```
CREATE FLASHBACK ARCHIVE DEFAULT fla4 TABLESPACE tbs1  
QUOTA 100G RETENTION 10 YEAR;
```

この会社では表Billingsのフラッシュバック・アーカイブを有効にして、デフォルトのフラッシュバック・アーカイブに履歴データを格納します。

```
ALTER TABLE Billings FLASHBACK ARCHIVE;
```

2007年5月1日に、クライアントは診断と検査について誤った金額を請求されました。2007年5月1日現在の記録を確認するには、会社は次の問合せを使用します。

```
SELECT date_billed, amount_billed, patient_name, claim_Id,  
test_costs, diagnosis FROM Billings AS OF TIMESTAMP  
TO_TIMESTAMP('2007-05-01 00:00:00', 'YYYY-MM-DD HH24:MI:SS');
```

19.9.8.5 シナリオ: フラッシュバック・タイム・トラベルを使用したデータのリカバリ

エンドユーザーが、データベースですでにコミット済の誤ったトランザクションからリカバリします。誤ったトランザクションのUNDOデータは今後利用できなくなりますが、必要な履歴情報がフラッシュバック・アーカイブにあるため、フラッシュバック問合せはシームレスに機能します。

Lisaはソフトウェア開発グループを管理しており、そのグループの製品販売が好調です。このため、2007年11月3日に、2年を超える経験を有するレベル3の全従業員に10%の昇給とレベル4への昇進を与えることを決定します。Lisaはこの変更を人事部(HR)代表のBobに依頼します。

BobはHRのWebアプリケーションを使用して、employee表を更新し、Lisaのレベル3の部下に10%の昇給とレベル4への昇進を与えます。次に、Bobは当日の自分の業務を終了して退社しますが、トランザクション中に2年の経験という要件を省略したことに気づいていません。数日後、Bobが更新を完了したかどうかをLisaが確認した際、昇給がグループの全員に与えられたことが判明します。彼女はただちにBobに電話し、誤りを修正するよう要求します。

Bobは最初は、バックアップにアクセスしなければ従業員表を以前の状態に戻せないと考えます。次に、employee表ではフラッシュバック・アーカイブが有効になっていることを思い出します。

Bobは最初に、彼の変更後にemployee表を変更した他のトランザクションがないことを検証します。トランザクション問合せからのコミット・タイムスタンプは、2日前のBobのトランザクションと一致します。

次に、Bobは次の文を使用してemployee表を誤った変更以前の状態に戻します。

```
DELETE EMPLOYEE WHERE MANAGER = 'LISA JOHNSON';
INSERT INTO EMPLOYEE
  SELECT * FROM EMPLOYEE
    AS OF TIMESTAMP (SYSTIMESTAMP - INTERVAL '2' DAY)
  WHERE MANAGER = 'LISA JOHNSON';
```

それからBobはLisaに要求された更新を再度実行します。

19.10 Oracle Flashback Technologyの一般ガイドライン

- DBMS_FLASHBACK.ENABLEおよびDBMS_FLASHBACK.DISABLEプロシージャを、制御しないSQLコード、または複数の連続した問合せに対して過去の同じ時刻を使用する場合のSQLコードの周辺で使用します。
- 記述するSQLコードで、簡便性のためにOracle Flashback Query、Oracle Flashback Version QueryまたはOracle Flashback Transaction Queryを使用します。たとえば、Oracle Flashback Queryは柔軟性が高く、単一の問合せで比較と結果の格納を行うことができます。
- 後でフラッシュバック機能で使用するためのSCNを取得するには、DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBERを使用します。
- 問合せで使用する過去の時間を計算または取得するには、タイムスタンプまたはSCN引数としてファンクション戻り値を使用します。たとえば、INTERVAL値をSYSTIMESTAMP関数の値に加算または減算します。
- Oracle Flashback Query、Oracle Flashback Version QueryおよびOracle Flashback Transaction Queryをローカルまたはリモートで使用します。リモートのOracle Flashback Queryの例は次のとおりです。

```
SELECT * FROM employees@some_remote_host AS OF
  TIMESTAMP (SYSTIMESTAMP - INTERVAL '60' MINUTE);
```

- データベースの整合性を保証するには、過去のデータを問い合わせる前に、COMMITまたはROLLBACK操作を実行します。

- すべてのフラッシュバック処理は、問合せ対象の時点に有効であった設定ではなく、各国語および文字セットなどの現在のセッション設定を使用することに注意してください。
- 表の構造を変更するいくつかのDDL(列の削除/変更、表の移動、パーティションの削除、表/パーティションの切捨て、制約の追加など)では、表に対する既存のUNDOデータは無効になります。DDL実行以前の時間のデータを取得しようとすると、ORA-01466のエラーが発生します。表の記憶域属性(PCTFREE、INITRANS、MAXTRANSなど)を変更するDDL操作では、UNDOデータは無効にされません。
- 正確な時間の過去のデータを問い合わせするには、SCNを使用します。タイムスタンプを使用した場合、問合せの実際の時間は、指定した時間よりも最大3秒早くなる場合があります。内部的には、Oracle DatabaseではSCNが使用され、このSCNは、3秒の間隔でタイムスタンプにマップされます。
たとえば、SCN値1000および1005が、タイムスタンプ午前8時41分および8時46分にそれぞれマップされているとします。午前8時41分から8時45分59秒の間に行われた問合せはSCN 1000にマップされます。午前8時46分に行われたOracle Flashback QueryはSCN 1005にマップされます。したがって、表の作成などのDDL操作の直後の時点を指定した場合、Oracle DatabaseではDDL操作の直前のSCNが使用されることがあり、エラーORA-01466が発生します。
- 動的パフォーマンス(v\$)ビューから過去のデータは取得できません。このようなビューに問合せを実行すると、現在のデータが戻されます。
- 静的データ・ディクショナリ・ビュー(*_TABLESなど)では、過去のデータに問合せを実行できます。

ノート:



静的データ・ディクショナリ・ビューを使用して過去のデータを取得できない場合は、対応する実表を問い合わせてデータを取得できます。ただし、実表は正規化され、ほとんどのデータは暗号化形式で格納されるため、Oracle では直接使用しないことをお勧めします。

- フラッシュバック・アーカイブの作成時または変更時にOPTIMIZE DATAを指定することで、フラッシュバック・アーカイブで管理される履歴表のデータ記憶域を最適化できます。

OPTIMIZE DATAを指定すると、次のいずれかの機能で履歴表のデータ記憶域が最適化されます。

- 高度な行圧縮
- 高度なLOB圧縮
- 高度なLOB重複除外
- セグメント・レベルの圧縮層
- 行レベルの圧縮層

デフォルトは、履歴表のデータ記憶域は最適化されません。

注意:



ユーザーが生成した履歴をインポートすると、不正確で信頼性のない結果となる可能性があります。このプロシージャを使用する場合は、事前に Oracle サポートに連絡してください。

19.11 Oracle Flashback Technologyのパフォーマンス・ガイドライン

- Oracle Flashback Queryに関連するすべての表の統計情報を生成するには、DBMS_STATSパッケージを使用します。統計情報を最新の状態に維持します。Oracle Flashback Queryは、これらの統計情報に依存するコストベース・オプティマイザを使用します。
- アクセスする必要のあるUNDOデータの量を最小限にします。問合せを使用する際に、表全体をスキャンするのではなく、索引を使用して少量の過去データを選択します。全表をスキャンする必要がある場合は、問合せにパラレル・ヒントを追加します。

I/Oのパフォーマンス・コストは、バッファ・キャッシュに存在していないデータおよびUNDOブロックにおけるページングのコストです。CPUの使用におけるパフォーマンス・コストは、影響を受けるデータ・ブロックにUNDO情報を適用する際のコストです。最近の過去における変更を操作する場合、フラッシュバック操作にはCPUの限界があります。

アーカイバのバージョン問合せがバッファ・キャッシュでUNDOデータを見つけられるように、十分なバッファ・キャッシュを設定することをお勧めします。バッファ・キャッシュのアクセスは、ディスク・アクセスよりも大幅に高速です。

- 追跡対象の表に非常に大きなトランザクション(1,000,000行超に影響)が実行される場合は、パラレル問合せが新たなチャンクをSGAから割り当てる必要がないように、十分な大容量のプール・サイズ(1GB以上)を設定します。
- Oracle Flashback Version Queryの場合は、索引構造を使用します。Oracle Databaseでは、データの変更および索引変更に関するUNDOデータが保持されます。索引参照ベースのOracle Flashback Version Queryのパフォーマンスは、全表スキャン(索引を使用しない場合は必要)よりも桁違いに高速です。
- Oracle Flashback Transaction Queryでは、xid列の型がRAW(8)です。xid列に対して作成された索引を利用するには、HEXTORAW変換ファンクションHEXTORAW(xid)を使用します。
- マテリアライズド・ビューに対するOracle Flashback Queryには、問合せリライト最適化の効果はありません。

関連項目:

ラージ・プール・サイズの設定の詳細は、[『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。

19.12 マルチテナント・コンテナ・データベースでのOracle Flashback Technologyの制限

次のOracle Flashback Technology機能は、マルチテナント・コンテナ・データベース(CDB)に対して使用不可です。

- Oracle Database 12cリリース1 (12.1.0.1)の場合、フラッシュバック・アーカイブはCDBでサポートされていません。Oracle Database 12cリリース1 (12.1.0.2)では、この制限がなくなります。
- トランザクションのフラッシュバック問合せはCDBではサポートされません。
- トランザクションのフラッシュバックのバックアウトはCDBではサポートされません。

20 プログラミング環境の選択

開発プロジェクト用のプログラミング環境を選択するときは、次のドキュメントを読むことをお勧めします。

- この章の各項、およびそれらの項で参照しているドキュメント。
- プラットフォームでサポートされるコンパイラや開発ツールを示すプラットフォーム固有のドキュメント。

次のようにプログラミング環境の選択が明白な場合もあります。

- Pro*COBOLでは、ADTやコレクション型がサポートされませんが、Pro*C/C++ではサポートされています。

必要な機能をすべて備えたプログラミング言語がない場合は、次の理由により、複数のプログラミング言語を使用することができます。

- この章のすべてのプログラミング言語は、PL/SQLおよびJavaのストアド・サブプログラムを起動できます。(ストアド・サブプログラムには、トリガーおよびADTメソッドが含まれます。)
- PL/SQL、Java、SQLおよびOracle Call Interface(OCI)は、外部のCサブプログラムを起動できます。
- 外部Cサブプログラムは、SQL、OCIまたはPro*C(C++ではなく)を使用してOracle Databaseにアクセスできます。

トピック:

- [アプリケーション・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](#)
- [PL/SQLの概要](#)
- [Oracle Database Javaサポートの概要](#)
- [PL/SQLまたはJavaの選択](#)
- [プリコンパイラの概要](#)
- [OCIおよびOCCIの概要](#)
- [プリコンパイラおよびOCIの比較](#)
- [Oracle Data Provider for .NET\(ODP.NET\)の概要](#)
- [OraOLEDBの概要](#)

関連項目:

複数言語プログラミングの詳細は、[「複数のプログラミング言語を使用したアプリケーションの開発」](#)を参照してください。

20.1 アプリケーション・アーキテクチャの概要

ここで、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 Databaseに接続するコンピューティング環境を意味します。

トピック:

- [クライアント/サーバー・アーキテクチャ](#)

- [サーバー側のプログラミング](#)
- [2層アーキテクチャと3層アーキテクチャ](#)

関連項目:

アプリケーション・アーキテクチャの詳細は、[『Oracle Database概要』](#)を参照してください。

20.1.1 クライアント/サーバー・アーキテクチャ

従来のクライアント/サーバー・プログラムでは、アプリケーションのコードはデータベース・サーバー以外のシステムである、クライアント・システム上で実行されます。データベース・コールは、クライアント・システムからデータベース・サーバーに送信されます。挿入および更新操作時には、データがクライアントからサーバーに送信され、問合せ操作時には、データがサーバーからクライアントに戻されます。データは、クライアント・システム上で処理されます。クライアント/サーバー・プログラムは、通常、プリコンパイラを使用して作成されますが、SQL文はC、C++、COBOLなどの別の言語のコードに埋め込まれています。

関連項目:

クライアント/サーバー・アーキテクチャの詳細は、[『Oracle Database概要』](#)を参照してください。

20.1.2 サーバー側のプログラミング

データベースで変更が発生した場合に自動的に実行されるトリガー、または明示的に起動されるストアド・サブプログラムを使用して、完全にデータベース内に常駐するアプリケーション・ロジックを開発できます。アプリケーションから作業をオフロードすると、検証およびクリーン・アップの実行に使用されるコードを再利用できたり、様々なクライアントからのデータベース処理を制御できます。たとえば、Webサーバーを介したストアド・サブプログラムの起動を可能にすることで、クライアント/サーバー・アプリケーションと同じ機能を実行するWebベースのユーザー・インタフェースを構成できます。

関連項目:

サーバー側のプログラミングの詳細は、[『Oracle Database概要』](#)を参照してください。

20.1.3 2層アーキテクチャと3層アーキテクチャ

クライアント/サーバー・コンピューティングは、**2層モデル**と呼ばれることがあります。このモデルでは、アプリケーションはデータベース・サーバーと直接通信します。**3層モデル**では、別個のアプリケーション・サーバーが要求を処理します。アプリケーション・サーバーには、基本的なWebサーバーや、キャッシュやロード・バランスなどの拡張機能を実行するサーバーがあります。この中間層の処理能力を向上させると、クライアント・システムに必要なリソースを少なくできるため、Thinkクライアント構成になり、この構成では、クライアント・システムが必要とするのはWebブラウザか、またはTCP/IPやHTTPプロトコルを介して要求を送信する他の手段のみです。

関連項目:

複数層アーキテクチャの詳細は、[『Oracle Database概要』](#)を参照してください。

20.2 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・インタフェースの機能は次のとおりです。

- クライアント・ユーザー・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で、または外部ユーザー・プログラムのデータ型に対して、データを変換します。

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。

関連項目:

プログラム・インタフェースの詳細は、[『Oracle Database概要』](#)を参照してください。

トピック:

- [ユーザー・インタフェース](#)
- [ステートフル・ユーザー・インタフェースとステートレス・ユーザー・インタフェース](#)

20.2.1 ユーザー・インタフェース

ユーザー・インタフェースとは、アプリケーションがエンドユーザーに対して表示するものです。これはそのアプリケーションの背後にあるテクノロジーおよびユーザー自身のニーズによって異なります。経験豊富なユーザーの場合は、データベースに渡すSQL文を入力できます。初心者ユーザーの場合は、クライアント・システム(Windows、X-Windowなど)のグラフィック・ライブラリを使用したグラフィカル・ユーザー・インタフェースを表示できます。また、従来のユーザー・インタフェースは、HTMLおよびJavaを使用してWebブラウザで提供することもできます。

20.2.2 ステートフル・ユーザー・インタフェースとステートレス・ユーザー・インタフェース

従来のクライアント/サーバー・アプリケーションでは、アプリケーションはユーザーのアクションを記録し、この情報を1つ以上のセッションで使用できます。たとえば、前に選択した項目を再入力する必要がないように、その項目をメニュー内に表示できます。アプリケーションがこの方法で情報を保存できる場合、そのアプリケーションは**ステートフル**であるといえます。

WebアプリケーションまたはThinクライアント・アプリケーションは、**ステートレス**であると開発が容易です。ステートレス・アプリケーションは、必要なすべての情報を収集し、データベースを使用してその情報を処理し、ユーザーが変わるとそれらの操作を再実行します。これは、顧客登録などの単一画面要求を処理する一般的な方法です。

デフォルトでステートレスであるWebアプリケーションにステートフルのアクションを追加する方法は、数多くあります。たとえば、Webページ上のエントリ・フォームから次のWebページに情報が渡されるように、複数の異なるステップを通してユーザーの選択内容を記憶するウィザードのようなインタフェースを構成できます。Cookieを使用してクライアント・システムに関する少量の情報項目を格納でき、ユーザーがWebサイトに再度アクセスしたときはそれらの項目を取得できます。また、サーブレットを使用して、データベース・セッションを開いたままにし、同じクライアントからの要求間で、変数の格納もできます。

20.3 PL/SQLの概要

SQLの手続き型拡張言語であるOracleのPL/SQLは、完全な移植性を備えた高パフォーマンスのトランザクション処理言語です。PL/SQLでは、SQL文によるデータ操作、条件選択やループによるプログラム・フローの制御、定数や変数の宣言、サブプログラムの定義、型、サブタイプ、およびADTの定義とこれらの型を持つ変数の宣言、ランタイム・エラーのトラップが可能です。

Oracle Databaseのプログラム・インタフェースのいずれかを使用して作成したアプリケーションであれば、PL/SQLストアド・サブプログラムを起動したり、PL/SQLコードのブロックをOracle Databaseに送信して実行できます。第三世代言語(3GL)のアプリケーションでは、ホスト変数および暗黙的データ型変換を使用して、PL/SQLのスカラー・データ型およびコンポジット・データ型にアクセスできます。3GL言語は、アセンブラ言語に比べて理解しやすく、名前付き変数などの機能が含まれています。第四世代言語(4GL)とは異なり、アプリケーション・ドメイン固有ではありません。

PL/SQLを使用すると、Webクライアントで起動できるストアド・プロシージャを開発することができます。

関連項目:

- PL/SQLの利点、主要な機能およびアーキテクチャの詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。

20.4 Oracle Database Javaサポートの概要

この項では、JavaアプリケーションをサポートするOracle Databaseの機能の概要を説明します。データベースには、java.lang、java.ioなどのコアJDKライブラリが含まれています。データベースでは、JDBCおよびSQLJなどのクライアント側Java標準がサポートされています。また、データベース内でデータ処理集中型のJavaコードを実行可能な、サーバー側のJDBCドライバが提供されています。

トピック:

- [Oracle JVMの概要](#)
- [Oracle JDBCの概要](#)
- [Oracle SQLJの概要](#)
- [Oracle JDBCとOracle SQLJの比較](#)
- [Javaストアド・サブプログラムの概要](#)
- [Oracle Database Webサービスの概要](#)

関連項目:

- [Oracle Database Java開発者ガイド](#)
- [『Oracle Database JDBC開発者ガイド』](#)

20.4.1 Oracle JVMの概要

Oracle Databaseとともに提供されるJava仮想マシンであるOracle JVMはJ2SEバージョン1.5.x仕様に準拠しており、データベース・セッション・アーキテクチャをサポートしています。

任意のデータベース・セッションにより、専用JVMをアクティブにできます。すべてのセッションで同じJVMコードおよびJVM静的値

を共有しますが、個々のセッション領域では、指定したセッションのプライベート状態が保持され、ガベージが収集されます。

この設計のメリットは次のとおりです。

- JavaアプリケーションにSQL操作と同じセッションの分離性およびデータの整合性が備わります。
- データの整合性を保つために別のプロセスでJavaを実行する必要がありません。
- Oracle JVMはメモリー・フットプリントが小さい堅牢なJVMです。
- Oracle JVMにはデータベースと同じ線形対称マルチプロセッシング(SMP)スケーラビリティがあるため、何千もの同時Javaセッションをサポートできます。

Oracle JVMは、Oracle Databaseがサポートする全プラットフォームで一貫して動作します。Oracle JVMを使用して開発したJavaアプリケーションは、サポートされるどのプラットフォームにも簡単に移植できます。

Oracle JVMには、デプロイ時のネイティブ・コンパイラが含まれています。このコンパイラでは、Javaコードは1回のコンパイルで実行可能形式で格納され、複数のユーザーがこれを共有できるため、より早く効率的な起動が可能になります。

Oracle JVMでは、データベースのセキュリティ機能も使用できます。Javaクラスをコール可能にする前に、(Oracle JDeveloper、サード・パーティIDE、SQL*Plusまたはloadjavaユーティリティを使用して)データベース・スキーマにロードする必要があります。Javaクラスのコールは、データベース認証とデータベース認可、Java 2セキュリティ、および実行者権限(IR)や定義者権限(DR)を介して、保護および制御されます。

Oracle Database 12cリリース1 (12.1.0.1)から、Oracle JVMでは、最新のJava Standard Editionを完全にサポートしています。最新のJava標準との互換性により、アプリケーションのポータビリティが向上し、クライアント側Javaクラスをデータベースで直接実行できるようになりました。

関連項目:

- Oracle JVMのその他の概要は、[『Oracle Database概要』](#)を参照してください。
- 最新のJava Standard Editionのサポートの詳細は、『Oracle Database Java開発者ガイド』を参照してください

20.4.2 Oracle JDBCの概要

Java Database Connectivity(JDBC)は、Oracle Databaseなどのオブジェクト・リレーショナル・データベースに対してJavaからSQL文を送信するApplication Program Interface(API)です。

Oracle Databaseには、JDBC 1.22標準に対する次の拡張機能が含まれます。

- Oracleデータ型のサポート
- 行のプリフェッチによるパフォーマンス強化
- バッチ処理の実行によるパフォーマンス強化
- ラウンド・トリップを削減する問合せ列型の指定
- DatabaseMetaDataコールの制御

Oracle Databaseは、JDBC 2.0標準のすべてのAPI(コアAPI、オプションのパッケージおよび多数の拡張機能を含む)をサポートしています。その拡張機能のいくつかには、データソース、JTAおよび分散トランザクションが含まれます。

Oracle Databaseは、JDBC 3.0標準の次の機能をサポートしています。

- JDK 1.5のサポート。
- ローカル・トランザクションとグローバル・トランザクション間の切替え。
- トランザクションのセーブポイント。
- 接続プールによるプリペアド文の再利用

ノート:



JDBC コードと SQLJ コードは相互運用が可能です。

トピック:

- [Oracle JDBCドライバ](#)
- [JDBC 2.0のサンプル・プログラム](#)
- [JDBC 2.0より前のサンプル・プログラム](#)

関連項目:

- Oracle DatabaseでのJavaサポートのその他の概要は、『[Oracle Database概要](#)』を参照してください。
- [Oracle JDBCとOracle SQLJの比較](#)

20.4.2.1 Oracle JDBCドライバ

JDBC標準では、次の4種類のJDBCドライバが定義されています。

タイプ	説明:
1	JDBC-ODBC ブリッジ。ソフトウェアはクライアント・システム上にインストールする必要があります。
2	ネイティブ・メソッド(C または C++をコール)および Java メソッド。ソフトウェアはクライアント上にインストールする必要があります。
3	Pure Java。クライアントはソケットを使用してサーバー上のミドルウェアをコールします。
4	最も Pure Java の度合の高いソリューション。Java ソケットを使用してデータベースと直接対話します。

JDBCはSQL規格のPart 3「Call-Level Interface」に基づいています。

動的SQLを実行するにはJDBCを使用します。動的SQLでは、実行される埋込みSQL文が何であるかはアプリケーションが実行されるまでわからず、SQL文の作成に入力が必要とします。

Oracleによって実装されたドライバには、Sun社が定義したJDBC標準機能に対する拡張が含まれています。

トピック:

- [JDBC Thinドライバ](#)

- [JDBC OCIドライバ](#)
- [サーバー側JDBC内部ドライバ](#)

関連項目:

- JDBCドライバのその他の概要は、[『Oracle Database概要』](#)を参照してください。
- JDBCの詳細は、[『Oracle Database JDBC開発者ガイドおよびリファレンス』](#)を参照してください。

20.4.2.1.1 JDBC Thinドライバ

JDBC Thinドライバはタイプ4(100% Pure Java)のドライバで、Javaソケットを使用してデータベース・サーバーに直接接続します。このドライバには、独自のTwo-Task Common(TTC)実装(Oracle NetのTCP/IPを軽量小型用に実装したもの)が含まれています。このドライバは、完全にJavaで作成されているため、プラットフォームには依存しません。

Thinドライバでは、クライアント側にOracleソフトウェアは必要ありません。サーバー側にはTCP/IPリスナーが必要です。このドライバは、WebブラウザをダウンロードするJavaアプレット内、またはOracleクライアント・ソフトウェアをインストールしないアプリケーション内で使用します。Thinドライバは自己完結型ですが、Javaソケットをオープンするため、ソケットをサポートするブラウザでのみ実行できます。

20.4.2.1.2 JDBC OCIドライバ

JDBC OCIドライバはタイプ2のJDBCドライバです。Cで作成されているOCIをコールして、Oracle Databaseと対話します。したがって、ネイティブ・メソッドおよびJavaメソッドの両方を使用します。

OCIドライバを使用すると、Thinドライバより多くの機能(透過的アプリケーション・フェイルオーバー、高度なセキュリティ、高度なLOB操作など)にアクセスできます。

OCIドライバは、異なるOracle Databaseバージョン間の互換性を提供します。また、IPC、Named Pipes、TCP/IPおよびIPX/SPXを含むすべてのインストール済Oracle Netアダプタもサポートします。

OCIドライバは、ネイティブ・メソッド(JavaとCの組合せ)を使用するため、プラットフォーム固有のドライバです。Oracle Net、OCIライブラリ、COREライブラリなどのすべての依存ファイルとともに、Oracle8i以降のバージョンをクライアントにインストールする必要があります。OCIドライバは、通常、Thinドライバより実行速度が速くなります。

OCIドライバは、プラットフォーム固有のCライブラリを使用しており、Webブラウザにはダウンロードできないため、Javaアプレット用には適していません。Oracle Application Serverなどの中間層アプリケーション・サーバー内で実行中のJ2EEコンポーネントでは使用できます。Oracle Application Serverは、アプリケーションおよびブラウザ間のアクセスをサポートするミドルウェア・サービスおよびツール製品を提供します。

20.4.2.1.3 サーバー側JDBC内部ドライバ

サーバー側JDBC内部ドライバは、タイプ2のドライバで、データベース・サーバー内で実行されます。このため、大量データへのアクセスに必要なラウンド・トリップの数が削減されます。このドライバと、Javaサーバー・VM、データベース、実行速度を最大で10倍に上げるJavaネイティブ・コンパイラ、およびSQLエンジンは、すべて同じアドレス空間内で実行されます。

このドライバは、データベースで使用するすべてのJavaプログラムのサーバー側サポートを提供します。PL/SQLストアド・サブプログラムおよびトリガーをコールすることもできます。

サーバー側ドライバは、クライアント側ドライバと同じ機能および拡張機能を完全にサポートします。

20.4.2.2 JDBC 2.0のサンプル・プログラム

次の例では、JDBC 2.0のJNDIを使用してデータソースを検索する場合の推奨される方法を示します。

```
// import the JDBC packages
import java.sql.*;
import javax.sql.*;
import oracle.jdbc.pool.*;
...
InitialContext ictx = new InitialContext();
DataSource ds = (DataSource)ictx.lookup("jdbc/OracleDS");
Connection conn = ds.getConnection();
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("SELECT last_name FROM employees");
while ( rs.next() ) {
    out.println( rs.getString("ename") + "<br>");
}
conn.close();
```

20.4.2.3 JDBC 2.0より前のサンプル・プログラム

次のソース・コードは、Oracle JDBC Thinドライバの登録、データベースへの接続、Statementオブジェクトの作成、問合せの実行および結果セットの処理を行います。

SELECT文は、hr.employees表のlast_name列を取得してその内容をリストします。

```
import java.sql.*;
import java.math.*;
import java.io.*;
import java.awt.*;
class JdbcTest {
    public static void main (String args []) throws SQLException {
        // Load Oracle driver
        DriverManager.registerDriver (new oracle.jdbc.OracleDriver());

        // Connect to the local database
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:thin:@myhost:1521:orcl",
                                         "hr", "password");

        // Query the employee names
        Statement stmt = conn.createStatement ();
        ResultSet rset = stmt.executeQuery ("SELECT last_name FROM employees");
        // Print the name out
        while (rset.next ())
            System.out.println (rset.getString (1));
        // Close the result set, statement, and the connection
        rset.close();
        stmt.close();
        conn.close();
    }
}
```

Propertiesオブジェクトを使用するgetConnection()メソッドによって、JDBCドライバに対するOracle Databaseの拡張を実行します。Propertiesオブジェクトを使用すると、ユーザー、パスワードおよびデータベース情報の他に、行プリフェッチおよびバッチ実行を指定できます。

このコード内でOCIドライバを使用するには、Connection文を次のコードで置換します。MyHostStringは、tnsnames.oraファイル内のエントリです。

```
Connection conn = DriverManager.getConnection ("jdbc:oracle:oci8:@MyHostString",
        "hr", "password");
```

アプレットを作成する場合の`getConnection()`および`registerDriver()`の文字列の指定は異なります。

20.4.3 Oracle SQLJの概要



ノート:

このマニュアルでは、SQLJ は Oracle SQLJ およびその拡張機能を意味します。

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、JDBCにかわり、Javaからのクライアント側SQLデータ・アクセスを、より簡単なものにします。

SQLJのソース・ファイルには、SQL文が埋め込まれたJavaソースが含まれています。Oracle SQLJは、動的SQLおよび静的SQLをサポートします。動的SQLのサポートは、SQLJ標準に対するOracleの拡張機能です。

Oracle SQLJトランスレータは、次の作業を行います。

- SQLJソースを、SQLJランタイム・ドライバへのコールを持つJavaコードに変換します。SQLJトランスレータは、ソース・コードをPure Javaのソース・コードに変換し、データベース・スキーマに対して静的SQL文の構文およびセマンティクスを確認し、ホスト変数とSQLとで型に互換性があるかどうかを検証します。
- Javaコンパイラを使用して生成済Javaコードをコンパイルします。
- ターゲット・データベース用にプロファイルを作成します(オプション)。SQLJで、Oracle Database固有にカスタマイズされたプロファイル・ファイルが生成されます。

SQLJは、JDeveloperと統合されています。SQLJに対するソース・レベルのデバッグ・サポートはJDeveloperで使用できます。

次に、単純で実行可能なSQLJ文の例を示します。employee表ではemployee_idが一意であるため、この文は値を1つ戻します。

```
String name;  
#sql { SELECT first_name INTO :name FROM employees WHERE employee_id=112 };  
System.out.println("Name is " + name + ", employee number = " + employee_id);
```

SQL式に含まれる各ホスト変数(または、修飾名が複雑なJavaホスト式)の前にはコロン(:)が付きます。その他のSQLJ文はJavaの型を宣言します。たとえば、多数の値を取得する問合せのイテレータ(データベース・カーソルに関連する構造体)を宣言できます。

```
#sql iterator EmpIter (String EmpNam, int EmpNumb);
```

関連項目:

Oracle SQLJ構文の例および詳細は、『Oracle Database SQLJ開発者ガイド』を参照してください

トピック:

- [SQLJのメリット](#)

関連項目:

SQLJのその他の概要は、『[Oracle Database概要](#)』を参照してください。

20.4.3.1 SQLJのメリット

Javaに対するOracle SQLJ拡張機能を使用すると、埋込みSQLを介してデータベース操作を実行するアプリケーションを迅速に開発したり、簡単にメンテナンスできます。

特に、Oracle SQLJには次の特長があります。

- 静的SQLからデータベースにアクセスする簡潔でわかりやすいメカニズムを提供します。アプリケーション内のSQLは、ほとんどの場合静的です。SQLJは、JDBCよりさらに簡潔で、エラーが発生しにくい静的SQL構文を提供します。
- 変換時に構文およびセマンティクスの検証を行うSQLチェッカー・モジュールを提供します。
- 柔軟なデプロイメント構成を提供します。これにより、クライアント側または中間層でSQLJを実装できるようになります。
- ソフトウェア標準をサポートします。SQLJはベンダー・グループの努力の集積であり、携わったすべてのベンダーは、今後、SQLJをサポートしていきます。アプリケーションで複数ベンダーのデータベースにアクセスできます。
- ソース・コード・レベルの移植性を提供します。各ベンダーのDBMSコードがベンダー固有の機能に依存していない場合は、実行可能ファイルをすべてのベンダーのDBMSで使用できます。
- クライアントおよびサーバーで統一されたプログラミング・スタイルを施行します。
- SQLJトランスレータと、グラフィカルな統合開発環境である**Oracle JDeveloper**を統合します。この開発環境では、SQLJ変換、Javaコンパイル、プロファイルのカスタマイズ、ソース・コード・レベルのデバッグがすべて1つのステップで提供されます。
- Oracle Databaseデータ型の拡張機能が含まれています。

20.4.4 Oracle JDBCとOracle SQLJの比較

JDBCコードおよびSQLJコードは相互運用が可能なため、JDBCの動的SQL文をSQLJの静的および動的SQL文とともに使用できます。SQLJのイテレータ・クラスが、JDBCの結果セットに対応します。

JDBCとSQLJには、次のような相違点があります。

- JDBCは、Javaからデータベースに対する完全な動的SQLインタフェースを提供します。これにより、開発者はデータベースの操作を完全に制御できるようになります。SQLJを使用すると、Javaデータベース・プログラミングを簡略化して、開発の生産性を向上できます。
- JDBCは、Javaからの動的SQL実行を詳細に制御します。SQLJは、特定のデータベース・スキーマ内でのSQL操作に対して、高レベルの静的バインディングを提供します。
- SQLJのソース・コードは同等のJDBCソース・コードより簡潔です。
- SQLJでは、データベース接続を使用して静的SQLコードの型チェックを行います。JDBCは完全に動的なAPIであるため、型チェックを行いません。
- SQLJでは、問合せ出力および戻りパラメータに強い型指定を行えるため、コール時に型チェックを実行できます。JDBCは、SQLとの間でやりとりされる値についてコンパイル時の型チェックを行いません。
- SQLJプログラムでは、SQL文の中にJavaバインド式を直接埋め込むことができます。JDBCでは、各バインド変数ごとに取得するか、設定用の文が個別に必要なため、バインドは位置番号によって指定されます。
- SQLJは、SQLストアド・サブプログラムのコールに関して簡潔な規則を提供します。

たとえば、次の連続する4つの例は、JDBCエスケープ構文、またはOracle JDBC構文を使用したストアド・プロシー

ジャまたはストアド・ファンクションをコールする方法を示しています。

```
prepStmt.prepareCall("{call fun(?,?)}"); //stored proc. JDBC esc.  
prepStmt.prepareCall("{? = call fun(?,?)}"); //stored func. JDBC esc.  
prepStmt.prepareCall("begin fun(:1,:2);end;"); //stored proc. Oracle  
prepStmt.prepareCall("begin :1 := fun(:2,:3);end;"); //stored func. Oracle
```

SQLJでは、次のコードに相当します。

```
#sql {call fun(param_list) }; //Stored procedure  
// Declare x  
...  
#sql x = {VALUES(fun(param_list)) }; // Stored function  
// where VALUES is the SQL construct
```

次に、SQLJおよびJDBCに共通するメリットを示します。

- SQLJソース・ファイルにはJDBCコールを含めることができます。SQLJおよびJDBCは相互運用が可能です。
- PL/SQLとJavaのストアド・サブプログラムは完全に互換性があります。

20.4.5 Javaストアド・サブプログラムの概要

Javaストアド・サブプログラムを使用すると、中間層で実行するプログラムから独立した、データベース・サーバーで実行するプログラムを実装できます。この方法でアプリケーションを構造化すると複雑にならず、再利用率、セキュリティ、パフォーマンスおよびスケーラビリティが向上します。

たとえば、永続性のあるデータが必要な操作を実行するJavaストアド・サブプログラム、およびプレゼンテーションやビジネス・ロジック操作を実行する別のプログラムを作成できます。

Javaストアド・サブプログラムのSQLに対するインタフェースは、PL/SQLの実行モデルと類似した実行モデルを使用します。

関連項目:

- Javaストアド・サブプログラムのその他の概要は、[『Oracle Database概要』](#)を参照してください。
- Javaストアド・サブプログラムの詳細は、[『Oracle Database Java開発者ガイド』](#)を参照してください。

20.4.6 Oracle Database Webサービスの概要

Webサービスは、Javaアプリケーション開発用の分散コンピューティング・パラダイムを提供し、JDBCなどの初期のJavaプロトコルのかわりとなり、アプリケーションがXMLおよびWebプロトコルを介して相互作用できるようにします。たとえば、電子部品のベンダーは、在庫管理のためにWebベースのプログラム・インタフェースを仕入先に提供できます。ベンダーは、プログラムの一部としてWebサービスを起動し、戻されたデータに基づいて在庫を自動的に発注できます。

Webサービスで使用される主要なテクノロジーは次のとおりです。

- Webサービス記述言語(WSDL): XML文書作成のための標準書式。WSDLでは、Webサービスで実行できる操作、Webサービスの位置および起動方法が記述されます。特に、Webサービスが提供する操作およびパラメータ(パラメータ型を含む)が記述されます。また、WSDL文書には、Webサービスの位置、転送プロトコルおよび起動スタイルも記述されます。
- Simple Object Access Protocol(SOAP)メッセージ: Webサービスで使用するXMLベースのメッセージ・プロトコル。SOAPでは、HTTP、FTP、SMTPまたはJMSなどの特定の転送メカニズムは指定しませんが、ほとんどのWebサービスはHTTPまたはHTTPSを使用するメッセージを受信します。

- Universal Description, Discovery, and Integration(UDDI)ビジネス・レジストリ: Webサービスをインターネット上でリスト表示するディレクトリ。UDDIレジストリはよく電話帳と比較されます。たとえば、一意の識別子のリスト表示(ホワイト・ページ)、ビジネス・カテゴリのリスト表示(イエロー・ページ)およびサービス・プロトコルへのバインド指示(グリーン・ページ)という具合です。

Webサービスでは、様々な方法およびプロトコルを使用できます。たとえば:

- 同期ディスパッチ(通常)または非同期ディスパッチが可能です。
- Webサービスは、引数が送信されて応答が戻されるRPCスタイルの操作、または一方向のSOAP文書交換などのメッセージ・スタイルで起動できます。
- 様々なエンコーディング・ルール(リテラルまたはエンコード)を使用できます。

Webサービスは、必要な情報が事前にすべてわかっている場合は静的に起動でき、操作およびトランスポート・エンドポイントをその場で検出できる場合は動的に起動できます。

Oracle Databaseは、Webサービス・プロバイダまたはWebサービス・コンシューマのいずれかとして機能できます。プロバイダとしてデータベースを使用すると、ストアド・サブプログラム、データ、メタデータ、およびキューイング・システムやメッセージング・システムなどのデータベース・リソースに対するアクセスを共有および切断できます。

Webサービス・プロバイダとしてOracle Databaseを使用すると、次のような非接続環境および異機種間環境が提供されます。

- 作成に使用された言語に依存しないストアド・サブプログラムの公開
- SQL問合せおよびXQueryの公開

関連項目:

Webサービス・プロバイダとしてのOracle Databaseのその他の概要は、『Oracle Database概要』を参照してください。

20.5 PL/SQLまたはJavaの選択

PL/SQLおよびJavaは、サーバー内で相互運用します。JavaからPL/SQLパッケージを実行するか、またはJavaからPL/SQLを起動して、分散CORBAおよびEnterprise Java Beansクライアントからいずれかを起動できるようにできます。

[表20-1](#)に、PL/SQLパッケージ、およびそれぞれの相当するJavaでの操作を示します。

表20-1 PL/SQLパッケージおよび相当するJavaでの操作

PL/SQLパッケージ	相当するJavaでの操作
DBMS_ALERT	JDBC でパッケージをコールします。
DBMS_DDL	JDBC を使用します。
DBMS_JOB	Java ストアド・サブプログラムを持つジョブをスケジューリングします。
DBMS_LOCK	JDBC でコールします。

PL/SQLパッケージ	相当するJavaでの操作
DBMS_MAIL	JavaMail を使用します。
DBMS_OUTPUT	サブクラス <code>oracle.aurora.rdbms.OracleDBMSOutputStream</code> または Java ストアド・サブプログラム <code>DBMS_JAVA.SET_STREAMS</code> を使用します。
DBMS_PIPE	JDBC でコールします。
DBMS_SESSION	JDBC を使用して、ALTER SESSION 文を実行します。
DBMS_SNAPSHOT	JDBC でコールします。
DBMS_SQL	JDBC を使用します。
DBMS_TRANSACTION	JDBC を使用して、ALTER SESSION 文を実行します。
DBMS_UTILITY	JDBC でコールします。
UTL_FILE	JAVAUSERPRIV 権限を付与して Java I/O エントリ・ポイントを使用します。

トピック:

- [PL/SQLとJavaの類似性](#)
- [Javaに対するPL/SQLの利点](#)
- [PL/SQLに対するJavaの利点](#)

20.5.1 PL/SQLとJavaの類似性

PL/SQLおよびJavaにはどちらもパッケージおよびライブラリがあります。

PL/SQLおよびJavaにはどちらも次のオブジェクト指向機能があります。

- どちらにも継承があります。
- PL/SQLには、**型進化**(型のメソッドおよび属性を、その型を使用するサブタイプおよび表データを保持しながら変更する機能)があります。
- Javaには、分散システムを開発するためのポリモフィズムおよびコンポーネント・モデルがあります。

20.5.2 Javaに対するPL/SQLの利点

SQLの拡張機能であるPL/SQLでは、すべてのSQLデータ型、データ・カプセル化、情報隠蔽、オーバーロードおよび例外処理がサポートされます。したがって、次のようになります。

- SQLデータ型は、JavaよりもPL/SQLで簡単に使用できます。

- SQL操作はJavaよりもPL/SQLで速く処理されます(特に大量のデータが扱われる場合、多くのデータベース・アクセスが行われる場合、またはバルク操作が使用される場合)。

Oracle9iでは、一部の高度なPL/SQL機能がJavaで使用できない場合があります(たとえば、自律型トランザクションやリモート・データベース用のdblink機能などです)。

コード開発は通常、JavaよりPL/SQLのほうが高速ですが、これは実際には、使用する開発ツールまたは開発環境に大きく依存します。

データ・ロジックがSQLに集中している場合は、PL/SQLが優先されます。つまり、アプリケーションのデータ処理またはデータデータ検証の要件は高くなります。

また、開発に利用できる、Oracle提供パッケージおよびサード・パーティ・ライブラリによる大規模なユーザー・ベースもあります。

20.5.3 PL/SQLに対するJavaの利点

Javaはオープン分散アプリケーション用に使用され、多くのJavaベース開発ツールが存在しています。Javaには、PL/SQLでは利用できない固有のメカニズムがあります。たとえば、Javaにはセキュリティ・メカニズム、自動ガベージ・コレクタ、型安全性メカニズム、バイト・コード検証機能、およびJava 2セキュリティが組み込まれています。また、Javaでは、組込みの配列境界チェック、組込みのネットワーク・アクセス・クラス、および有用な多数の即時使用可能なクラスを含むAPIなど、高速の開発機能が組み込まれています。Javaには、データベースで再利用できる、多数のクラス・ライブラリ、ツール、およびサード・パーティ・クラス・ライブラリを備えています。JavaにはPL/SQLよりも豊富な型体系があります。JavaはCORBA(クライアントに様々なコンピュータ言語を持つことができます)およびEnterprise Java Beansを使用できます。PL/SQLパッケージはCORBAまたはEnterprise Java Beansクライアントから起動できます。Javaによって、XMLツール、Internet File SystemまたはJavaMailを実行できます。

アプリケーションがERPシステム、RMIサーバー、Java/J2EE、およびWebサービスと対話する必要がある場合、これらのいずれもPL/SQLでは実行できないため、Javaが推奨されます。Javaはまた、中間層でアプリケーション部分を開発する必要がある場合に推奨されます。これは、ビジネス・ロジックが複雑であるか、直接SQLアクセスをほとんど必要とせずに集中的に計算し、中間層ドリブン・プレゼンテーション・ロジックを実装しているため、アプリケーションで透過的なJava永続性を必要とするか、コンテナ管理のインフラストラクチャ・サービスを必要とするためです。このため、アプリケーションをデータベース層と中間層にパーティション化する場合がある場合、アプリケーション部分を必要に応じて中間層に移行し、Java/J2EEを使用します。

20.6 プリコンパイラの概要

クライアント/サーバー・プログラムは、通常、プリコンパイラを使用して作成されます。プリコンパイラは、C、C++、COBOLなどの言語で作成された高レベル・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。クライアント・アプリケーションは、SQL文をホストするためホスト・プログラムと呼ばれ、その作成に使用される言語はホスト言語と呼ばれます。

プリコンパイラはホスト・プログラムを入力として受け取り、埋込みSQL文を標準データベース・ランタイム・ライブラリ・コールに変換し、ソース・プログラムを生成します。ユーザーは、通常の方法でソース・プログラムをコンパイルおよびリンクして、実行できます。

トピック:

- [Pro*C/C++プリコンパイラの概要](#)
- [Pro*COBOLプリコンパイラの概要](#)

関連項目:

20.6.1 Pro*C/C++プリコンパイラの概要

Pro*C/C++プリコンパイラの場合、ホスト言語はCまたはC++です。Pro*C/C++プリコンパイラの機能の一部は次のとおりです。

- スレッド・パッケージをサポートしているプラットフォームの場合は、マルチスレッド・プログラムを作成できます。同時接続は、シングル・スレッド・アプリケーションでもマルチスレッド・アプリケーションでもサポートされます。
- PL/SQLブロックを埋め込むことによって、パフォーマンスを向上できます。PL/SQLブロックは、ユーザーが独自に作成したか、またはOracle Databaseパッケージで提供されている、JavaやPL/SQLのサブプログラムを起動できます。
- プリコンパイラ・オプションを使用すると、プリコンパイル時のみでなく、実行時にもSQL文またはPL/SQL文の構文およびセマンティクスを確認できます。
- PL/SQLおよびJavaのストアド・サブプログラムを起動できます。COBOLまたはCで作成したモジュールは、Pro*C/C++から起動できます。共有ライブラリ内の外部Cサブプログラムは、プログラムから起動できます。
- 様々な環境で実行できるように、条件付きでコード・セクションをプリコンパイルできます。
- パフォーマンスを向上させるには、コード内で配列、構造体または構造体配列をホスト変数およびインジケータ変数として使用します。
- エラーおよび警告を処理してデータの整合性を保証することができます。エラーをどのように処理するかは、プログラマが制御します。
- プログラムで内部データ型からC言語データ型(またはその逆)に変換できます。
- 低レベルのCインタフェースおよびC++インタフェースであるOracle Call Interface(OCI)およびOracle C++ Call Interface(OCCI)は、プリコンパイラ・ソース内で使用できます。
- Pro*C/C++は、動的SQL(変数の値および文の構文をユーザーが入力できる手法)をサポートします。
- Pro*C/C++で、特殊なSQL文を使用してユーザー定義のオブジェクト型を含む表を操作できます。Object Type Translator(OTT)で、データベース内のADTおよび名前付きコレクション型を、ソースに含める構造体およびヘッダーにマップします。
- 結合配列、ネスト表およびVARRAYという3種類のコレクション型が一連のSQL文でサポートされています。これによって、データを高度に制御できます。
- ラージ・オブジェクトは、別の一連のSQL文によってアクセスされます。
- 動的SQLのための新しいANSI SQL規格が新しいアプリケーション用にサポートされ、様々なホスト変数を使用してSQL文を実行できます。動的SQLのための従来の手法も、既存のアプリケーションで使用できます。
- グローバリゼーション・サポートによって、マルチバイト・キャラクタおよびUCS2 Unicodeデータを使用できます。
- スクロール可能カーソルを使用して、結果セット内を移動できます。たとえば、結果セットの最終行のフェッチ、または結果セット内の絶対位置または相対位置へのジャンプが可能です。
- 接続プールは、いくつかの名前付き接続で共有されているデータベースへの物理接続のグループです。接続プールのオプションを有効化すると、Pro*C/C++アプリケーションのパフォーマンスを最適化できます。接続プールのオプションは、デフォルトでは無効になっています。

関連項目:

Pro*C/C++プリコンパイラの完全な情報は、『[Pro*C/C++プログラマーズ・ガイド](#)』を参照してください。

[例20-1](#)に、スキーマhr内のemployees表を問い合わせるCソース・プログラムのコードの一部を示します。

例20-1 Pro*C/C++アプリケーション

```
...
#define  UNAME_LEN    10
...
int    emp_number;
/* Define a host structure for the output values of a SELECT statement. */
/* No declare section needed if precompiler option MODE=ORACLE          */
struct {
    VARCHAR    last_name[UNAME_LEN];
    float      salary;
    float      commission_pct;
} emprec;
/* Define an indicator structure to correspond to the host output structure. */
struct {
    short emp_name_ind;
    short sal_ind;
    short comm_ind;
} emprec_ind;
...
/* Select columns last_name, salary, and commission_pct given the user's input
/* for employee_id. */
EXEC SQL SELECT last_name, salary, commission_pct
              INTO :emprec INDICATOR :emprec_ind
              FROM employees
              WHERE employee_id = :emp_number;
...
```

埋込みSELECT文と対話型(SQL*Plus)のSELECT文との違いはごくわずかです。すべての埋込みSQL文はEXEC SQLで始まります。コロン(:)がすべてのホスト(C)変数の前に付きます。データおよび(データ値がNULLの場合または文字列が切り捨てられたときに設定される)インジケータの戻り値は、(前述のコード部分にあるような)構造体、配列または構造体配列の中に格納できます。結果セットの値が複数の場合でも、前述の例(社員番号が一意のため結果が1つのみ)と同様の方法で簡単に処理されます。埋込みSQLでは、列および表の実際の名前を使用します。

プリコンパイラ・オプションのデフォルト値を使用することも、リソースの使用方法、エラーの通知方法、出力形式およびカーソル(特定の接続またはSQL文に対応するカーソル)の管理方法を制御するための値を入力することもできます。カーソルは、結果セットの値が複数の場合に使用します。

オプションは、構成ファイル内に入力するか、コマンドラインに入力するか、またはEXEC ORACLEで始まる特殊な文でソース・コード内に直接入力します。エラーが見つからなかった場合は、次に、Cプログラムの場合と同じように出力ソース・ファイルをコンパイルおよびリンクして実行します。

クライアントからのサーバー・データベース・アクセスを様々なプラットフォーム上に配置できるように作成するには、プリコンパイラを使用します。Pro*C/C++を使用すると、独自のユーザー・インタフェースを自由に設計し、既存のアプリケーションに自由にデータベース・アクセスを追加できます。

埋込みSQL文を作成する前に、SQL*Plusで対話型SQLをテストし、その後、少し変更を加えることで、埋込みSQLアプリケーションのテストを開始できます。

20.6.2 Pro*COBOLプリコンパイラの概要

Pro*COBOLプリコンパイラの場合、ホスト言語はCOBOLです。Pro*COBOLプリコンパイラの機能の一部は次のとおりです。

- PL/SQLまたはJavaのストアド・サブプログラムを起動できます。PL/SQLブロックを埋め込むことによって、パフォーマンスを向上できます。PL/SQLブロックでは、独自に作成したか、またはOracle Databaseパッケージで提供されているサブプログラムを起動できます。
- プリコンパイラ・オプションを使用すると、カーソル、エラー、構文検査、ファイル形式などの処理方法を定義できます。
- プリコンパイラ・オプションを使用すると、プリコンパイル時のみでなく、実行時にもSQL文またはPL/SQL文の構文およびセマンティクスを確認できます。
- 様々な環境で実行できるように、条件付きでコード・セクションをプリコンパイルできます。
- パフォーマンスを向上させるには、コード内で表、グループ項目またはグループ項目の表をホスト変数またはインジケータ変数として使用します。
- エラーおよび警告の処理方法をプログラミングし、データの整合性を保証できます。
- Pro*COBOLは、動的SQL(変数の値および文の構文をユーザーが入力できる手法)をサポートします。

関連項目:

Pro*COBOLプリコンパイラの詳細は、[『Pro*COBOLプログラマーズ・ガイド』](#)を参照してください。

[例20-2](#)に、スキーマhr内のemployees表を問い合わせるCOBOLソース・プログラムのコードの一部を示します。

例20-2 Pro*COBOLアプリケーション

```
...
WORKING-STORAGE SECTION.
*
* DEFINE HOST INPUT AND OUTPUT HOST AND INDICATOR VARIABLES.
* NO DECLARE SECTION NEEDED IF MODE=ORACLE.
*
01 EMP-REC-VARS.
   05 EMP-NAME      PIC X(10) VARYING.
   05 EMP-NUMBER    PIC S9(4) COMP VALUE ZERO.
   05 SALARY        PIC S9(5)V99 COMP-3 VALUE ZERO.
   05 COMMISSION    PIC S9(5)V99 COMP-3 VALUE ZERO.
   05 COMM-IND      PIC S9(4) COMP VALUE ZERO.
...
PROCEDURE DIVISION.
...
   EXEC SQL
       SELECT last_name, salary, commission_pct
       INTO :EMP-NAME, :SALARY, :COMMISSION:COMM-IND
       FROM employees
       WHERE employee_id = :EMP-NUMBER
   END-EXEC.
...
```

埋込みSELECT文と対話型(SQL*Plus)のSELECT文との違いはごくわずかです。すべての埋込みSQL文はEXEC SQLで始まります。コロンの(:)がすべてのホスト(COBOL)変数の前に付きます。SQL文はEND-EXECによって終了します。データおよび(データ値がNULLの場合または文字列が切り捨てられたときに設定される)インジケータの戻り値は、(前述のコード部分にあるような)グループ項目、表、またはグループ項目の表に格納できます。結果セットの値が複数の場合でも、前述の例(社員番号が一意のため結果が1つのみ)と同様の方法で簡単に処理されます。埋込みSQLでは、列および表の実際の名前を使用します。

プリコンパイラ・オプションのデフォルト値を使用することも、リソースの使用方法、エラーの通知方法、出力形式およびカーソル(特定の接続またはSQL文に対応するカーソル)の管理方法を制御するための値を入力することもできます。

オプションは、構成ファイル内に入力するか、コマンドラインに入力するか、またはEXEC ORACLEで始まる特殊な文でソース・コード内に直接入力します。エラーが見つからなかった場合は、COBOLプログラムの場合と同じように出力ソース・ファイルをコンパイルおよびリンクして実行できます。

クライアントからのサーバー・データベース・アクセスを様々なプラットフォーム上に配置できるように作成するには、プリコンパイラを使用します。Pro*COBOLを使用すると、独自のユーザー・インタフェースを自由に設計し、既存のCOBOLアプリケーションに自由にデータベース・アクセスを追加できます。

使用できる埋込みSQL文はANSI規格に準拠しているため、Oracle Net経由で多数のデータベースのデータにアクセスできます。

埋込みSQL文を作成する前に、SQL*Plusで対話型SQLをテストし、その後、少し変更を加えることで、埋込みSQLアプリケーションのテストを開始できます。

20.7 OCIおよびOCCIの概要

Oracle Call Interface(OCI)およびOracle C++ Call Interface(OCCI)は、APIです。これらによって、第三世代言語固有のサブプログラム起動を使用して、Oracle Databaseにアクセスしたり、SQL文実行のすべての過程を制御するアプリケーションを作成できます。これらのAPIは次のものを提供します。

- システム・メモリーおよびネットワーク接続の効率的な使用によるパフォーマンスと拡張性の向上
- 2層クライアント/サーバーまたは複数層環境での動的セッションおよびトランザクション管理に適した一貫性のあるインタフェース
- N層認証
- Oracle Databaseオブジェクトを使用したアプリケーション開発の包括的サポート
- 外部データベースへの接続
- 増加するユーザー数および要求数に対応したサービスを、ハードウェアを追加しなくても提供できるアプリケーション開発

OCIを使用すると、データベース内のデータおよびスキーマをCのようなホスト・プログラミング言語を使用して操作できます。

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。これらのAPIは、標準的なデータベース・アクセス関数および検索関数のライブラリを、実行時にアプリケーション内にリンクされる動的ランタイム・ライブラリの形式で提供します。SQLまたはPL/SQLを3GLプログラムに埋め込む必要はありません。

関連項目:

OCIおよびOCCIコールの詳細は、次を参照してください。

- [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](#)
- [Oracle Databaseアドバンスド・キューイング・ユーザーズ・ガイド](#)
- [Oracle Databaseグローバルゼーション・サポート・ガイド](#)
- [Oracle Databaseデータ・カートリッジ開発者ガイド](#)

トピック:

- [OCIおよびOCCIのメリット](#)

- [OCIおよびOCCI関数](#)
- [OCIおよびOCCIアプリケーションの手続き型要素および非手続き型要素](#)
- [OCIアプリケーションまたはOCCIアプリケーションの作成](#)

20.7.1 OCIおよびOCCIのメリット

OCIおよびOCCIは、Oracle Databaseに対する他のアクセス方法に比べて、次のような優れたメリットがあります。

- アプリケーション設計のすべての面に対する細かな制御。
- プログラムに対する高度な実行制御。
- 使い慣れた3GLプログラム手法およびアプリケーション開発ツール(ブラウザやデバッガなど)の使用。
- 動的SQL、メソッド4のサポート。
- 様々なプラットフォームにおいて、すべてのOracle Databaseのプログラム・インタフェースが使用可能。
- コールバックを使用した動的バインドおよび定義。
- サーバー・メタデータ層を表す記述機能。
- 登録されたクライアント・アプリケーションに対する非同期のイベント通知。
- 配列のデータ操作言語(DML)の機能拡張。
- コミット・リクエストを文の実行と関連付けてラウンド・トリップを減らす機能。
- 透過的プリフェッチ・バッファを使用して問合せを最適化し、ラウンド・トリップを減らす機能。
- OCIおよびOCCIハンドルに対する相互排他(mutex)ロックの実装が不要になるスレッド・セーフティ。
- 非ブロック化モードでのサーバー接続コールが実行中か、または完了できない場合、制御がOCIコードに戻る機能。

20.7.2 OCIおよびOCCI関数

OCIとOCCIの両方とも、次の種類の関数があります。

関数の種類	目的
リレーショナル	データベース・アクセスを管理し、SQL 文を処理します。
ナビゲーション	データベースから取得したオブジェクトを操作します。
データベースのマッピングおよび操作	Oracle Database データ型のデータ属性を操作します。
外部サブプログラム	PL/SQL から C コールバックを作成します(OCI のみ)

20.7.3 OCIおよびOCCIアプリケーションの手続き型要素および非手続き型要素

OCIおよびOCCIを使用すると、SQLが提供する強力な非手続き型データ・アクセス機能と、CやC++など数多くのプログラミング言語で提供される手続き型機能を組み合わせたアプリケーションを開発できます。手続き型言語および非手続き型言語には

次のような特性があります。

- 非手続き型言語プログラムでは、操作対象のデータ群は指定されますが、実行される操作およびその操作の実行方法は指定されません。SQLは非手続き型であるため、データベース・トランザクションの実行用として簡単に覚えやすく使用しやすい言語です。また、最新のリレーショナル・データベース・システムおよびオブジェクト・リレーショナル・データベース・システムにおけるデータのアクセスおよび操作に使用される標準言語でもあります。
- 手続き型言語プログラムでは、文の実行の多くが、その前後の文や、ループまたは条件ブランチなどの制御構造に依存します。これらはSQLでは使用できません。このような言語は、手続き型であるためにSQLより複雑になりますが、非常に柔軟で強力でもあります。

OCIプログラムまたはOCCIプログラムで非手続き型言語の要素と手続き型言語の要素を組み合わせることができるため、構造化プログラミング環境の中でOracle Databaseに簡単にアクセスできます。

OCIおよびOCCIでは、Oracle Databaseで使用可能なSQLのデータ定義、データ操作、問合せおよびトランザクション制御のすべての機能をサポートします。たとえば、OCIプログラムまたはOCCIプログラムはOracle Databaseに対する問合せを実行できます。この問合せで、次のように、入力(バインド)変数を使用してデータベースにデータを提供するようにプログラムに要求できます。

```
SELECT name FROM employees WHERE empno = :empnumber
```

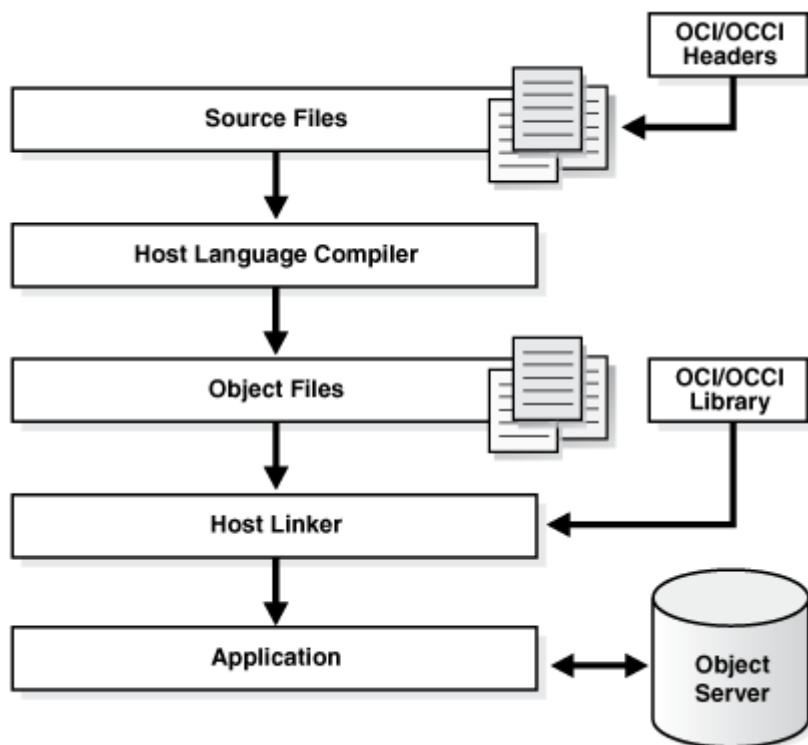
このSQL文では、:empnumberがアプリケーションによって提供される値のプレースホルダです。

また、SQLの手続き型拡張言語である、OracleのPL/SQLを使用することもできます。PL/SQLにより、SQLのみで作成されたアプリケーションよりも、さらに強力な柔軟性のあるアプリケーションを開発できます。OCIおよびOCCIは、Oracle Databaseにあるオブジェクトにアクセスし、操作する機能も提供しています。

20.7.4 OCIアプリケーションまたはOCCIアプリケーションの作成

[図20-1](#)に示すとおり、OCIまたはOCCIプログラムは非データベース・アプリケーションと同じ方法でコンパイルおよびリンクします。処理前のステップやプリコンパイル・ステップを別途行う必要はありません。

図20-1 OCIまたはOCCIの開発過程



ノート:



OCI プログラムおよび OCID プログラムを正しくリンクするには、プラットフォームによっては OCI ライブラリおよび OCID ライブラリ以外のライブラリを含める必要があります。必要な追加ライブラリの詳細は、プラットフォーム固有の Oracle マニュアルを参照してください。

20.8 プリコンパイラおよびOCIの比較

通常、プリコンパイラ・アプリケーションには同等のOCIアプリケーションより少ないコードが含まれているため、生産性が向上します。

データベースを詳細に制御する必要がある場合は、OCIアプリケーション(純OCIアプリケーションまたはOCIコールが埋め込まれたプリコンパイラ・アプリケーション)の方が適しています。

- OCIでは、セッションの多重化および移行をより詳細に制御できます。
- OCIでは、リストを含む任意の構造体を使用できるコールバックを使用して、動的バインドおよび定義が提供されます。
- OCIにはメタデータを扱う多くのコールがあります。
- OCIでは、クライアント・アプリケーションに対して非同期のイベントを通知できます。クライアントに対し、他のクライアントに伝播するための通知を生成する手段が提供されます。
- OCIでは、任意のエラー・メッセージを戻す前に、DML文は配列を使用してできるだけ多くの反復を完了します。
- 特殊な目的のためのOCIコールには、アドバンスド・キューイング、グローバルバージョン・サポート、データ・カートリッジおよび日時データ型のサポートが含まれます。
- OCIコールは、Pro*C/C++アプリケーションに埋め込むことができます。

20.9 Oracle Data Provider for .NET(ODP.NET)の概要

Oracle Data Provider for .NET(ODP.NET)は、Oracle Database用のデータ・プロバイダの実装です。

ODP.NETはOracle Database固有のAPIを使用して、.NETアプリケーションからデータベース機能およびデータへの高速かつ高信頼アクセスを提供します。また、Microsoft .NET Framework Class Libraryで入手可能なクラスおよびインタフェースを使用し、継承します。

Oracle Provider for OLE DBを使用するプログラマに対して、ADO(ActiveX Data Objects)では簡単なプログラミング・モデルを表すオートメーション・レイヤーを提供します。ADO.NETからも類似したプログラミング・モデルがパフォーマンスの向上のために提供されますが、オートメーション・レイヤーはありません。さらに重要な点として、ADO.NETモデルを使用することで、ODP.NETなどのネイティブ・プロバイダはOracle Database固有の特定の機能およびデータ型を公開できるようになります。

関連項目:

[Oracle Data Provider for .NETの開発者ガイドfor Microsoft Windows](#)

次に、Oracle Databaseに接続し、接続を切断する前にバージョン番号を表示する単純なC#アプリケーションを示します。

```
using System;
using Oracle.DataAccess.Client;
class Example
{
    OracleConnection con;
    void Connect()
    {
        con = new OracleConnection();
        con.ConnectionString = "User Id=hr;Password=password;Data Source=oracle";
        con.Open();
        Console.WriteLine("Connected to Oracle" + con.ServerVersion);
    }
    void Close()
    {
        con.Close();
        con.Dispose();
    }

    static void Main()
    {
        Example example = new Example();
        example.Connect();
        example.Close();
    }
}
```

ノート:



その他のサンプルは、ディレクトリ ORACLE_BASE¥ORACLE_HOME¥ODP.NET¥Samples で提供されます。

20.10 OraOLEDBの概要

Oracle Provider for OLE DB(OraOLEDB)は、高いパフォーマンスを提供し、OLE DBコンシューマがOracleデータへ効率的にアクセスできるようにするOLE DBデータ・プロバイダです。通常、この開発者ガイドでは、OLE DBまたはADOを介してOraOLEDBを使用していることを想定しています。

関連項目:

[『Microsoft Windows用Oracle Provider for OLE DB開発者ガイド for Microsoft Windows』](#)

21 複数のプログラミング言語を使用したアプリケーションの開発

この章では、他のプログラミング言語で記述された外部プロシージャをコールするデータベース・アプリケーションの開発方法について説明します。

トピック:

- [複数言語プログラムの概要](#)
- [外部プロシージャの概要](#)
- [外部プロシージャ用のコール仕様の概要](#)
- [外部プロシージャのロード](#)
- [外部プロシージャの公開](#)
- [Javaクラス・メソッドの発行](#)
- [外部Cプロシージャの発行](#)
- [コール仕様の位置](#)
- [コール仕様による外部Cプロシージャへのパラメータの受渡し](#)
- [CALL文による外部プロシージャの実行](#)
- [複数言語のプログラム・エラーおよび例外処理](#)
- [外部Cプロシージャでのサービス・ルーチンの使用](#)
- [外部Cプロシージャを使用したコールバックの実行](#)

21.1 複数言語プログラムの概要

Oracle Databaseは、次に示す複数の異なる言語での操作が可能です。

- [『Oracle Database PL/SQL言語リファレンス』](#)で説明されるPL/SQL
- [『Oracle Call Interfaceプログラマーズ・ガイド』](#)で説明される、Oracle Call Interface(OCI)を使用したC
- [『Oracle C++ Call Interfaceプログラマーズ・ガイド』](#)で説明される、Oracle C++ Call Interface (OCCI)を使用したC++
- [『Pro*C/C++プログラマーズ・ガイド』](#)で説明される、Pro*C/C++プリコンパイラを使用したCまたはC++
- [『Pro*COBOLプログラマーズ・ガイド』](#)で説明される、Pro*COBOLプリコンパイラを使用したCOBOL
- 『Oracle Provider for OLE DB開発者ガイドfor Microsoft Windows』で説明される、Oracle Provider for OLE DBを使用したVisual Basic。
- [『Oracle Data Provider for .NET開発者ガイドfor Microsoft Windows』](#)で説明される、Oracle Data Provider for .NETを使用した.NET
- JDBCおよびSQLクライアント側アプリケーション・プログラミング・インタフェース(API)を使用したJava。 [『Oracle](#)

[Database JDBC開発者ガイド](#)および[Oracle Database SQLJ開発者ガイド](#)を参照してください。

- [『Oracle Database Java開発者ガイド』](#)で説明される、データベース内のJava。これには、[『Oracle Database Java開発者ガイド』](#)で説明される、Javaストアド・プロシージャ(SQLに発行されデータベースに格納されるJavaメソッド)の使用も含まれます。

また、Oracle JVM Web Call-Outユーティリティでは、SQLオブジェクトやPL/SQLパッケージなどのデータベース・エンティティを表すJavaクラスの生成、Javaクライアント・プログラムでのSQL、PL/SQL、およびサーバー側JavaからWebサービスへの発行、およびデータベース内からの外部Webサービスの起動などが可能です。[『Oracle Database Java開発者ガイド』](#)を参照してください。

これらの実装言語の中から何をどのように選択できるでしょうか。各言語には、それぞれ異なるメリットがあります。使用しやすさ、特定の専門知識を持つプログラマーがいるかどうか、移植が必要かどうか、さらに既存コードの有無などが重要な決定要因になります。

ただし、アプリケーションでOracle Databaseをどのように使用するかによって、次のように選択範囲が狭くなる可能性があります。

- PL/SQLは、SQLトランザクション処理に特化した強力な開発ツールです。
- 演算集中型のタスクは、Cなどの下位レベル言語で最も効率的に実行されます。
- 移植性とセキュリティの両方を求める場合は、Javaを選択できます。
- Microsoftプログラミング言語の知識がある場合は、.NETを選択することを推奨します。

パフォーマンスに関して最も重要なことは、サーバーのアドレス空間内で実行されるのはPL/SQLおよびJavaメソッドのみであるということです。C/C++および.NETメソッドは外部プロシージャとしてディスパッチされ、サーバー・システム上で実行されますが、これはデータベース・サーバーのアドレス空間外です。Pro*COBOLおよびPro*C/C++はプリコンパイラであり、Visual BasicはCで実装されているOracle Provider for OLE DBおよびOCIを介してOracle Databaseにアクセスします。

以上のすべての要因を考慮すると、アプリケーションを複数の言語で実装する必要がある場合があることがわかります。たとえば、サーバーのアドレス空間内でJavaが実行されるため、既存のJavaアプリケーションをデータベース内にインポートし、PL/SQLおよびSQLからJava関数をコールしてこのテクノロジーを利用できます。

PL/SQL外部プロシージャにより、Cプロシージャ・コールをPL/SQLの本体として作成できます。このようなCプロシージャは、PL/SQLからは直接、またSQLからはPL/SQLプロシージャ・コールを介してコールできます。データベースでは、コール仕様というインターフェースが提供されています。コール仕様を使用すると、他の言語から外部プロシージャをコールできます。このサービスは、SQL、PL/SQL、CおよびJava間の相互通信用に設計されていますが、このような言語をコールできる基本言語ならどの言語からでもアクセスできます。たとえば、JavaまたはC以外の言語で作成したプロシージャでも、Cからコール可能なプロシージャであれば、SQLまたはPL/SQLからでも使用できます。したがって、候補となるC++プロシージャがある場合は、そのプロシージャでC++のextern "C"文を使用してそのプロシージャをCからコールできるようにします。

したがって、複数の異なる言語の強みおよび機能をプログラム環境に関係なく利用できます。固有の制約を持つ1つの言語に制限する必要はありません。外部プロシージャの使用によって、特定の目的に特定の言語をデプロイできるため、再利用性およびモジュール性が向上します。

21.2 外部プロシージャの概要

外部プロシージャは、動的リンク・ライブラリ(DLL)に格納されているプロシージャです。このプロシージャを基本言語に登録し、それをコールして特定の処理を実行できます。

たとえば、PL/SQLを使用する場合、PL/SQLが実行時にライブラリを動的にロードし、次に、プロシージャをPL/SQLプロシージャとみなしてコールします。このようなプロシージャは、現行のトランザクションに完全に組み込まれ、データベースをコールバックしてSQL操作を実行できます。

このようなプロシージャは必要なときにのみロードされるため、メモリーが節約されます。コール仕様が実装本体から切り離されているため、コール側プログラムに影響することなく、プロシージャを拡張できます。

外部プロシージャを使用すると、次のようなことができます。

- クライアント・アプリケーションの実行とデータベース・インスタンスからのプロセスを分離し、クライアント側で生じた問題がデータベースに影響を与えないようにすることができます。
- 演算集中型プログラムをクライアントから実行速度の速いサーバーへ移動できます(ネットワーク通信でのラウンド・トリップを回避するため)。
- データベース・サーバーと外部システムおよびデータ・ソースとのインタフェースが実現します。
- データベース・サーバー自体の機能を拡張できます。

ノート:



外部ライブラリ(DLL ファイル)は静的なリンクである必要があります。他の外部ライブラリ(DLL ファイル)からの外部記号を参照しないようにしてください。Oracle Database はこういった記号を解決するため、外部プロシージャの原因による失敗がありません。

関連項目:

外部プロシージャ保護の詳細は、『Oracle Databaseセキュリティ・ガイド』を参照してください

21.3 外部プロシージャ用のコール仕様の概要

外部プロシージャの発行は、コール仕様を介して行いますが、このコール仕様には、AS LANGUAGE句を介したAS EXTERNAL機能のスーパーセットが含まれます。AS LANGUAGEコール仕様を使用すると、外部Cプロシージャを発行できます。(Javaクラス・メソッドは外部プロシージャではありませんが、コール仕様を使用できます。)

ノート:



レガシー・アプリケーションをサポートするために、コール仕様では、AS EXTERNAL 句を使用して発行することもできます。ただし、アプリケーションを開発する場合は、AS LANGUAGE 句を使用することをお勧めします。

一般に、コール仕様を使用することによって、次のようなことができます。

- 適切なCまたはJavaのターゲット・プロシージャのディスパッチ
- データ型変換
- パラメータ・モードのマッピング
- 自動的なメモリー割当ておよびクリーンアップ

- SQLからコールされるパッケージ・ファンクションに必要な応じて指定される純粋度制約。
- データベース・トリガーからのJavaメソッドまたはCプロシージャのコール
- 位置の柔軟性: パフォーマンスを最適化し実装の詳細を隠すために、AS LANGUAGEコール仕様を、パッケージ仕様または型仕様、あるいはパッケージ(または型)本体に入れることができます。

既存プログラムを外部プロシージャとして使用するには、そのプログラムをロードして発行した後、コールします。

21.4 外部プロシージャのロード

外部CプロシージャまたはJavaメソッドをPL/SQLで使えるようにするには、プロシージャまたはメソッドをロードする必要があります。

ノート:



外部 C プロシージャは、DLL をサポートするプラットフォームまたは Solaris の .so ライブラリなどの動的にロード可能な共有ライブラリをサポートするプラットフォームのみで使用できます。

アプリケーションが外部Cプロシージャをコールすると、Oracle DatabaseまたはOracle Listenerが外部プロシージャ・エージェントextprocを起動します。アプリケーションは、Oracle DatabaseまたはOracle Listenerが確立したネットワーク接続を使用し、次の情報をextprocに渡します。

- DLLまたは共有ライブラリの名前
- 外部プロシージャの名前
- 外部プロシージャのすべてのパラメータ

その後、extprocはDLLまたは共有ライブラリをロードし、外部プロシージャを実行し、外部プロシージャによってアプリケーションに戻された値を渡します。アプリケーションとextprocは、同じコンピュータ上に存在する必要があります。

extprocは、使用されているコール規格に準拠するライブラリ内でプロシージャをコールできます。

ノート:

外部プロシージャのデフォルト構成では、Oracle Database および extproc と動作するネットワーク・リスナーは不要になりました。Oracle Database によって extproc が直接起動されるため、Oracle Listener が誤って extproc を起動する危険性が解消されました。セキュリティを最大限にするため、このデフォルト構成をお勧めします。



次のいずれかを使用する場合、Oracle Listener が extproc を起動するようにデフォルトの構成を変更する必要があります。

- マルチスレッドの extproc エージェント
- Windows 上の共有モードの Oracle Database
- 外部プロシージャを別の extproc エージェントにリダイレクトする、LIBRARY 仕様の AGENT 句または

関連項目:

コール規格の詳細は、[「CALLING STANDARD」](#)を参照してください。

デフォルト構成を変更すると、追加のネットワーク構成ステップが必要になります。

Cで作成された外部プロシージャ、またはCアプリケーションからコール可能な外部プロシージャを使用できるようにデータベースを設定するには、開発者またはデータベース管理者が次のステップを実行します。

1. [Cプロシージャの定義](#)
2. [環境の設定](#)
3. [DLLの識別](#)
4. [外部プロシージャを公開する](#)

21.4.1 Cプロシージャの定義

次のプロトタイプのいずれかを使用し、Cプロシージャを定義します。

- Kernighan & Ritchieスタイル・プロトタイプの定義は次のとおりです。

```
void C_findRoot(x)
float x;
...
```

- 全角ではない数値データ型(float、short、charなど)以外のISO/ANSIプロトタイプの定義は次のとおりです。

```
void C_findRoot(double x)
...
```

- デフォルトの引数格上げによってサイズが変わらないその他のデータ型。

デフォルトの引数格上げによってサイズが変わる定義の例は次のとおりです。

```
void C_findRoot(float x)
...
```

21.4.2 環境の設定

外部プロシージャでデフォルト構成を使用する場合、Oracle Databaseはextprocを直接起動します。listener.oraおよびtnsnames.oraの構成を変更する必要はありません。次の構文を使用し、外部プロシージャが使用する環境変数をextproc.oraファイルに定義します(ファイルは、UNIXオペレーティング・システムでは\$ORACLE_HOME/hs/adminに、WindowsではORACLE_HOME\hs\adminにあります)。

```
SET name=value (environment_variable_name value)
```

extprocがロードするDLLを制限するEXTPROC_DLLS環境変数を次のいずれかの値に設定します。

- NULLの場合、次のとおり設定します。

```
SET EXTPROC_DLLS=
```

デフォルトのこの設定では、extprocは、ディレクトリ\$ORACLE_HOME/binまたは\$ORACLE_HOME/libにあるDLLのみをロードします。

- ONLY:の後にコロンで区切られた(Windowsシステムではセミコロンで区切られた)DLLをリストして指定する場合、次のとおり設定します。

```
SET EXTPROC_DLLS=ONLY:DLL1:DLL2
```

この設定では、extprocはDLL1およびDLL2という名のDLLのみロードします。この設定で最大のセキュリティが得られます。

- コロンで区切られた(Windowsシステムではセミコロンで区切られた)DLLのリストの場合、次のとおり設定します。

```
SET EXTPROC_DLLS=DLL1:DLL2
```

この設定では、extprocは、DLL1およびDLL2という名のDLL、およびディレクトリ\$ORACLE_HOME/binまたは\$ORACLE_HOME/libに存在するDLLをロードします。

- ANYの場合、次のとおり設定します。

```
SET EXTPROC_DLLS=ANY
```

この設定では、extprocはすべてのDLLをロードします。

extprocプロセスの生成時に資格証明の使用を適用する、ENFORCE_CREDENTIAL環境変数を設定します。

ENFORCE_CREDENTIAL値はTRUEまたはFALSE (デフォルト)のいずれかです。ENFORCE_CREDENTIAL、およびextprocプロセスで予期される動作で、考えられる認証と偽装の使用例に基づく議論については、『Oracle Databaseセキュリティ・ガイド』の外部プロシージャ保護に関する情報を参照してください。

外部プロシージャのデフォルト構成を変更し、extprocエージェントをOracle Listenerから起動するには、Cで作成された外部プロシージャまたはCアプリケーションからコール可能な外部プロシージャを使用できるよう、データベース構成を次のように設定します。

ノート:



extprocで資格証明を使用する場合、Oracle Listenerはextprocエージェントの生成に使用できません。

1. エージェント用の構成パラメータ(デフォルト名extproc)を構成ファイルtnsnames.oraおよびlistener.ora内で設定します。この作業により、データベース起動時に外部プロシージャ・エージェントextprocとの接続が確立されます。
2. 外部プロシージャ専用のリスナー・プロセスを開始します。

リスナーは、extprocに必要な環境変数(ORACLE_HOME、ORACLE_SIDおよびLD_LIBRARY_PATHなど)をいくつか設定します。また、リスナーがlistener.oraエントリのENV\$セクションで固有の環境変数を定義すると、その環境変数はエージェント・プロセスに渡されます。それ以外は、エージェントに対してクリーンな環境を提供します。エージェント用に設定される環境変数は、クライアントおよびサーバー用に設定される環境変数から独立しています。したがって、エージェント・プロセス内で実行される外部プロシージャは、クライアントまたはサーバーのプロセス用に設定される環境変数を読み込むことはできません。

ノート:



環境変数自体は標準 C プロシージャの `setenv` および `getenv` を使用してそれぞれ設定し、読み込むことができます。この方法で設定された環境変数は、エージェント・プロセス専用です。そのプロセス内で実行されるすべての関数で読み込めますが、同一ホスト上で実行されている他のプロセスで読み込むことはできません。

3. 外部プロシージャ用のエージェントを専用モード(デフォルト)で実行するか、マルチスレッド・モードで実行するかを決定します。

専用モードでは、各セッションに対して専用エージェントが1つ起動されます。マルチスレッド・スレッドモードでは、単一のマルチスレッド `extproc` エージェントが起動されます。マルチスレッド `extproc` エージェントは、ユーザーごとに異なるスレッドを使用してコールを処理します。外部プロシージャをコールするユーザーが多いような構成では、マルチスレッドの `extproc` エージェントを使用してシステム・リソースを節約することをお勧めします。

エージェントを専用モードで実行する場合、エージェント・プロセスの構成はこれ以上必要ありません。

エージェントをマルチスレッド・モードで実行する場合、データベース管理者はエージェントが(マルチスレッド `extproc` エージェントとして)マルチスレッド・モードで起動されるようにデータベース・システムを構成する必要があります。この構成は、エージェント制御ユーティリティ `agtctl` を使用して行います。たとえば、次のコマンドを使用して `extproc` を起動します。

```
agtctl startup extproc agent_sid
```

この例の `agent_sid` は、この `extproc` エージェントが処理するシステム識別子です。このシステム識別子のエントリは、通常、`tnsnames.ora` ファイルにエントリとして追加されています。

ノート:



- マルチスレッド `extproc` エージェントを使用する場合、破損コール・スタックなどのエラーを避けるために、コールするライブラリはスレッド・セーフである必要があります。
- エージェント・プロセスを起動する、データベース・サーバー、エージェント・プロセスおよびリスナー・プロセスは、すべて同じホスト上に存在する必要があります。
- デフォルトでは、エージェント・プロセスはメイン・アプリケーションと同じデータベース・インスタンスで実行されます。信頼性が非常に重要な場合は、外部プロシージャ用のエージェント・プロセスを別のデータベース・インスタンス(同じホスト上)で実行すると、エージェントで問題が生じてもプライマリ・データベース・サーバーに影響を与えません。このようにするには、データベース・リンクを使用して別のデータベース・インスタンスを指定します。

『Oracle Call Interface プログラマーズ・ガイド』の図 F-1 は、マルチスレッド `extproc` エージェントのアーキテクチャを示しています。

関連項目:

`extproc` 管理のための `agtctl` の使用の詳細は、[『Oracle Call Interface プログラマーズ・ガイド』](#) を参照してください。

21.4.3 DLLの識別

ここでのDLLとは、外部プロシージャを格納する動的ロードが可能な任意のオペレーティング・システム・ファイルです。

セキュリティ上の理由から、DLLへのアクセスはDBAが制御します。DBAは、CREATE LIBRARY文を使用して、DLLを表す別名ライブラリというスキーマ・オブジェクトを作成します。次に、許可されているユーザーの場合は、DBAが別名ライブラリに対するEXECUTE権限を付与します。このかわりに、DBAは、ユーザーにCREATE ANY LIBRARY権限を付与することもあり、この場合、ユーザーは、次の構文を使用して自分の別名ライブラリを作成できます。

```
CREATE LIBRARY [schema_name.] library_name
  {IS | AS} 'file_path'
  [AGENT 'agent_link'];
```

ノート:

ANY 権限は強力なので、軽率に付与しないでください。詳細は、次を参照してください。



- ANY を含む、システム権限の管理の詳細は、[『Oracle Database セキュリティ・ガイド』](#)を参照してください。
- ユーザー・アカウントと権限のセキュリティのガイドラインは、[『Oracle Database セキュリティ・ガイド』](#)を参照してください。

DLL名だけではなく、ディレクトリ・オブジェクトを使用してDLLのパスを指定することをお勧めします。次の例では、DLLであるutils.soを表す別名ライブラリc_utilsを作成します。

```
CREATE LIBRARY C_utils AS 'utils.so' IN DLL_DIRECTORY;
```

DLL_DIRECTORYは、'/DLLs'を参照するディレクトリ・オブジェクトです。

または、次の例のようにDLLのフル・パスを指定することもできます。

```
CREATE LIBRARY C_utils AS '/DLLs/utils.so';
```

DLLを柔軟に指定するために、表記規則\${VAR_NAME}を使用してパスのルート部分を環境変数として指定し、その変数をlistener.oraエントリのENV\$セクションに設定できます。

次の例では、agent_linkという名前で指定されるエージェントを使用して、ライブラリC_utilsで外部プロシージャを実行します。

```
create database link agent_link using 'agent_tns_alias';
create or replace library C_utils is
  '${EP_LIB_HOME}/utils.so' agent 'agent_link';
```

環境変数EP_LIB_HOMEは、エージェントによって/usr/bin/dllなどのそのインスタンス用の適切なパスに展開されます。変数EP_LIB_HOMEは、エージェントがアクセスできるように、listener.oraファイル内に設定する必要があります。

セキュリティ上の理由から、デフォルト状態のextprocは、ディレクトリ\$ORACLE_HOME/binまたは\$ORACLE_HOME/libに存在するDLLのみをロードするようになっています。また、extprocに接続できるのは、同じシステムで実行されているOracle Databaseのクライアント・プロセスであるローカル・セッションのみです。

他のディレクトリからDLLをロードするには、環境変数EXTPROC_DLLSを設定する必要があります。この環境変数の値には、完全パスで修飾したDLL名をコロンで区切った(Windowsシステムではセミコロンで区切った)リストを指定します。たとえば:

```
EXTPROC_DLLS=/private1/home/johndoe/dll/myDll.so:/private1/home/johndoe/dll/newDll.so
```

extprocの環境変数はlistener.oraファイルのENVSパラメータから設定できますが、\$ORACLE_HOME/hs/adminディレクトリにあるextprocの初期化ファイルextproc.oraの環境変数も設定できます。extproc.oraおよびlistener.oraのENVSパラメータの両方が使用される場合、extproc.oraで定義された環境変数が優先されます。EXTPROC機能の詳細は、Oracle Netのマニュアルを参照してください。

ノート:



Windows システム上の extproc.ora では、パスの各円記号に対してドライブ文字および二重円記号(¥¥)を使用して、パスを指定します。(各二重円記号内の最初の円記号は、エスケープ文字として機能します。)

21.4.4 外部プロシージャを公開する

外部Cプロシージャを作成してから、これをDLLに追加します。プロシージャがDLL内にある場合、[「外部プロシージャの公開」](#)で説明されるコール仕様のメカニズムを使用して、プロシージャを公開します。

21.5 外部プロシージャの公開

Oracle Databaseで使える外部プロシージャは、コール仕様で公開される外部プロシージャのみです。Javaクラス・メソッドまたは外部Cプロシージャの名前、パラメータ型および戻り型が、対応するSQLの要素にマップされます。これはPL/SQLストアード・プロシージャと同様に作成しますが、本体には宣言やBEGIN ENDブロックではなく、AS LANGUAGE句を記述します。

AS LANGUAGE句では、次の指定を行います。

- プロシージャの作成言語
- Javaメソッド
 - Javaメソッドのシグネチャ
- Cプロシージャ
 - DLLに対応するCプロシージャ用の別名ライブラリ
 - DLL内のCプロシージャの名前
 - パラメータの受渡し方法を指定するための様々なオプション
 - 異なるシステム上でプロシージャを実行するための、外部プロシージャ・エージェントextprocの名前を保持するパラメータ(ある場合)

プロシージャ、ファンクション、パッケージ仕様部、パッケージ本体、型仕様または型本体用の通常のCREATE OR REPLACE構文を使用して、宣言を開始します。

コール仕様は、名前宣言およびパラメータ宣言の後に指定します。構文は次のとおりです。

```
{IS | AS} LANGUAGE {C | JAVA}
```

これに続いて、次のいずれかを指定します。

```
NAME java_string_literal_name
```

java_string_literal_nameは、Javaメソッドのシグネチャ

または

```
{ LIBRARY library_name [ NAME c_string_literal_name ] |  
  [ NAME c_string_literal_name ] LIBRARY library_name }  
[ AGENT IN ( argument [, argument]... ) ]  
[ WITH CONTEXT ]  
[ PARAMETERS (external_parameter[, external_parameter]...) ];
```

library_nameは別名ライブラリの名前で、c_string_literal_nameは外部Cプロシージャの名前です。
external_parameterは次を意味します。

```
{ CONTEXT  
| SELF [{TDO | property}]  
| {parameter_name | RETURN} [property] [BY REFERENCE] [external_datatype]}
```

propertyは次を意味します。

```
{INDICATOR [{STRUCT | TDO}] | LENGTH | DURATION | MAXLEN | CHARSETID | CHARSETFORM}
```

ノート:



Java と異なり、C では SQL 型は認識されません。このため、構文がより複雑になります。

トピック:

- [Javaクラス・メソッド用のAS LANGUAGE句](#)
- [外部Cプロシージャ用のAS LANGUAGE句](#)

21.5.1 Javaクラス・メソッド用のAS LANGUAGE句

AS LANGUAGE句は、PL/SQLとJavaクラス・メソッドの間のインタフェースです。

21.5.2 外部Cプロシージャ用のAS LANGUAGE句

次の副次句は、PL/SQLに対して、外部Cプロシージャを検索する場所、そのプロシージャのコール方法、プロシージャに何を渡すかを指示します。

- [LIBRARY](#)
- [NAME](#)
- [LANGUAGE](#)
- [CALLING STANDARD](#)
- [WITH CONTEXT](#)
- [PARAMETERS](#)
- [\[AGENT IN\]](#)

これらの副次句のうちLIBRARYのみが必須です。

21.5.2.1 LIBRARY

ローカルの別名ライブラリを指定します。(リモート・ライブラリの指定にデータベース・リンクは使用できません)。ライブラリ名は

PL/SQL識別子です。このため、名前を二重引用符で囲むと、名前の大/小文字が区別されます。(デフォルトでは、名前は大文字で格納されます)。別名ライブラリにはEXECUTE権限が必要です。

21.5.2.2 NAME

コール対象の外部Cプロシージャを指定します。プロシージャ名を二重引用符で囲むと、名前の大/小文字が区別されます。(デフォルトでは、名前は大文字で格納されます)。この副次句を省略すると、プロシージャ名はPL/SQLプロシージャ名を大文字で表したものがデフォルト設定されます。

ノート:



LANGUAGE および CALLING STANDARD は、旧版の AS EXTERNAL 句のみに適用されます。

21.5.2.3 LANGUAGE

外部プロシージャが作成されている第三世代言語を指定します。この副次句を省略すると、言語名はCにデフォルト設定されます。

21.5.2.4 CALLING STANDARD

外部プロシージャをコンパイルしたコール規格を指定します。サポートされているコール規格はCです。この副次句を省略すると、コール規格はCにデフォルト設定されます。

21.5.2.5 WITH CONTEXT

コンテキスト・ポインタが外部プロシージャに渡されることを指定します。コンテキスト・データ構造体は外部プロシージャに対して不透明ですが、外部プロシージャによってコールされるサービス・プロシージャでは使用できます。

21.5.2.6 PARAMETERS

外部プロシージャに渡されるパラメータの位置およびデータ型を指定します。現在の長さや最大長などのパラメータ・プロパティや、パラメータの受渡し方法(値によるか参照によるか)も指定できます。

21.5.2.7 AGENT IN

このプロシージャを実行するエージェント・プロセスの名前を保持するパラメータを指定します。これは、外部プロシージャ・エージェントextprocが、複数のエージェント・プロセスを使用して実行される場合を想定して行います。これによって、1つの外部プロシージャのエージェント・プロセスにエラーが発生した場合の信頼性が保証されます。エージェント・プロセスの名前(データベース・リンクの名前に対応)を渡すことができ、またtnsnames.oraおよびlistener.oraが両方のインスタンス間で正しく設定されている場合、外部プロシージャが、もう一方のインスタンスでコールされます。両方のインスタンスは、同じホスト上に存在する必要があります。

この操作は、CREATE LIBRARY文のAGENT句の場合と同様です。AGENT INを使用して実行時の値を指定すると、柔軟性が向上します。

このようにエージェント名を指定すると、別名ライブラリで宣言されるエージェント名はいずれもオーバーライドされます。エージェント名が指定されていない場合は、extprocエージェントがコール側プログラムと同じインスタンス上でデフォルト設定されます。

21.6 Javaクラス・メソッドの発行

Javaクラスおよびそのメソッドは、RDBMSライブラリ・ユニットに格納されます。この中には、LOADJAVAユーティリティまたはSQL文CREATEJAVAを使用して、Javaソース、バイナリおよびリソースをロードできます。ライブラリ・ユニットは、たとえばCで作成されたDLLに類似していますが、DLLには複数のプロシージャを入れられるのに対して、ライブラリ・ユニットはJavaクラスと1対1で対応します。

NAME句文字列は、Javaメソッドを一意に識別します。PL/SQLのファンクションまたはプロシージャおよびJavaには対応するパラメータが必要です。Javaメソッドがパラメータをとらない場合は、空のパラメータ・リストを作成する必要があります。

JavaクラスをRDBMSにロードするとき、クラスはSQLに対して自動的に発行されません。選択されたパブリック静的メソッドしかSQLに明示的に発行できないためです。ただし、適切な権限がある場合には、データベースに存在する他のJavaクラスからすべてのメソッドを起動することができます。

次に、引数の階乗を戻すJavaメソッド(J_calcFactorial)を発行する場合を考えてみます。

```
package myRoutines.math;
public class Factorial {
    public static int J_calcFactorial (int n) {
        if (n == 1) return 1;
        else return n * J_calcFactorial(n - 1);
    }
}
```

次のコール仕様では、SQL*Plusを使用して、JavaメソッドJ_calcFactorialをPL/SQLスタンド・ファンクションplsToJavaFac_funcとして発行します。

```
CREATE OR REPLACE FUNCTION Plstojavafac_func (N NUMBER) RETURN NUMBER AS
LANGUAGE JAVA
NAME 'myRoutines.math.Factorial.J_calcFactorial(int) return int';
```

21.7 外部Cプロシージャの発行

次の例では、C関数Cdivisor_funcを外部関数として発行するplsCallsCdivisor_funcというPL/SQLスタンドアロン・ファンクションを作成します。

```
CREATE OR REPLACE FUNCTION Plscallscdivisor_func (
/* Find greatest common divisor of x and y: */
x      PLS_INTEGER,
y      PLS_INTEGER)
RETURN PLS_INTEGER
AS LANGUAGE C
LIBRARY C_utils
NAME "Cdivisor_func"; /* Quotation marks preserve case. */
```

21.8 コール仕様の位置

Javaクラス・メソッドおよび外部Cプロシージャはともに、次のいずれかの位置にコール仕様を指定できます。

- スタンドアロンのPL/SQLプロシージャ
- PL/SQLパッケージ仕様部
- PL/SQLパッケージ本体
- ADT仕様

- ADT本体

トピック:

- [例: PL/SQLパッケージ内へのコール仕様の配置](#)
- [例: PL/SQLパッケージ本体内部へのコール仕様の配置](#)
- [例: ADT仕様内へのコール仕様の配置](#)
- [例: ADT本体内部へのコール仕様の配置](#)
- [例: AUTHIDを指定したJava](#)
- [例: オプションのAUTHIDを指定したC](#)
- [例: パッケージ内でのコール仕様の混合使用](#)

ノート:



次のいくつかの例では、コール仕様の完全指定に AUTHID 句および SQL_NAME_RESOLVE 句が必要な場合があります。

関連項目:

- PL/SQLからの外部プロシージャのコールの詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。
- SQLのCALL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。

21.8.1 例: PL/SQLパッケージ内へのコール仕様の配置

```
CREATE OR REPLACE PACKAGE Demo_pack
AUTHID DEFINER
AS
    PROCEDURE plsToC_demoExternal_proc (x PLS_INTEGER, y VARCHAR2, z DATE)
    AS LANGUAGE C
        NAME "C_demoExternal"
        LIBRARY SomeLib
        WITH CONTEXT
        PARAMETERS(CONTEXT, x INT, y STRING, z OCIDATE);
END;
```

21.8.2 例: PL/SQLパッケージ本体内部へのコール仕様の配置

```
CREATE OR REPLACE PACKAGE Demo_pack
AUTHID CURRENT_USER
AS
    PROCEDURE plsToC_demoExternal_proc(x PLS_INTEGER, y VARCHAR2, z DATE);
END;

CREATE OR REPLACE PACKAGE BODY Demo_pack
SQL_NAME_RESOLVE CURRENT_USER
AS
    PROCEDURE plsToC_demoExternal_proc (x PLS_INTEGER, y VARCHAR2, z DATE)
    AS LANGUAGE JAVA
        NAME 'pkg1.class4.methodProc1(int,java.lang.String,java.sql.Date)';
```

END;

21.8.3 例: ADT仕様内へのコール仕様の配置

ノート:



この項の例が機能するには、次のデータ構造を設定する必要があります(これには、権限 CREATE ANY LIBRARY が必要です)。

```
CREATE OR REPLACE LIBRARY SOME LIB AS '/tmp/lib.so';  
  
CREATE OR REPLACE TYPE Demo_typ
```

```
AUTHID DEFINER AS OBJECT (Attribute1 VARCHAR2(2000), SomeLib varchar2(20), MEMBER  
PROCEDURE plsToC_demoExternal_proc (x PLS_INTEGER, y VARCHAR2, z DATE) AS LANGUAGE C  
NAME "C_demoExternal" LIBRARY SomeLib WITH CONTEXT -- PARAMETERS(CONTEXT, x INT, y  
STRING, z OCIDATE) PARAMETERS(CONTEXT, x INT, y STRING, z OCIDATE, SELF) );
```

21.8.4 例: ADT本体内へのコール仕様の配置

```
CREATE OR REPLACE TYPE Demo_typ  
AUTHID CURRENT_USER  
AS OBJECT  
    (attribute1 NUMBER,  
    MEMBER PROCEDURE plsToJ_demoExternal_proc (x PLS_INTEGER, y VARCHAR2, z DATE)  
    );  
CREATE OR REPLACE TYPE BODY Demo_typ  
AS  
    MEMBER PROCEDURE plsToJ_demoExternal_proc (x PLS_INTEGER, y VARCHAR2, z DATE)  
    AS LANGUAGE JAVA  
        NAME 'pkg1.class4.J_demoExternal(int,java.lang.String,java.sql.Date)';  
END;
```

21.8.5 例: AUTHIDを指定したJava

スタンドアロンPL/SQLプロシージャ内にJavaクラス・メソッドを発行する例を示します。

```
CREATE OR REPLACE PROCEDURE plsToJ_demoExternal_proc (x PLS_INTEGER, y VARCHAR2, z  
DATE)  
    AUTHID CURRENT_USER  
AS LANGUAGE JAVA  
    NAME 'pkg1.class4.methodProc1(int,java.lang.String,java.sql.Date)';
```

21.8.6 例: オプションのAUTHIDを指定したC

スタンドアロンPL/SQLプログラム内にCプロシージャをAS EXTERNALとして発行する例を次に示します。AUTHID句はオプションです。これによって、Oracle Databaseリリース8.0の外部プロシージャとの互換性が保たれます。

```
CREATE OR REPLACE PROCEDURE plsToC_demoExternal_proc (x PLS_INTEGER, y VARCHAR2, z  
DATE)  
AS  
    EXTERNAL  
    NAME "C_demoExternal"  
    LIBRARY SomeLib  
    WITH CONTEXT  
    PARAMETERS(CONTEXT, x INT, y STRING, z OCIDATE);
```

21.8.7 例: パッケージ内でのコール仕様の混合使用

```
CREATE OR REPLACE PACKAGE Demo_pack
AUTHID DEFINER
AS
    PROCEDURE plsToC_InBodyOld_proc (x PLS_INTEGER, y VARCHAR2, z DATE);
    PROCEDURE plsToC_demoExternal_proc (x PLS_INTEGER, y VARCHAR2, z DATE);
    PROCEDURE plsToC_InBody_proc (x PLS_INTEGER, y VARCHAR2, z DATE);
    PROCEDURE plsToJ_InBody_proc (x PLS_INTEGER, y VARCHAR2, z DATE);
    PROCEDURE plsToJ_InSpec_proc (x PLS_INTEGER, y VARCHAR2, z DATE)
    IS LANGUAGE JAVA
        NAME 'pkg1.class4.J_InSpec_meth(int,java.lang.String,java.sql.Date)';
    PROCEDURE C_InSpec_proc (x PLS_INTEGER, y VARCHAR2, z DATE)
    AS LANGUAGE C
        NAME "C_demoExternal"
        LIBRARY SomeLib
        WITH CONTEXT
        PARAMETERS(CONTEXT, x INT, y STRING, z OCIDATE);
END;
CREATE OR REPLACE PACKAGE BODY Demo_pack
AS
    PROCEDURE plsToC_InBodyOld_proc (x PLS_INTEGER, y VARCHAR2, z DATE)
    AS EXTERNAL
        NAME "C_InBodyOld"
        LIBRARY SomeLib
        WITH CONTEXT
        PARAMETERS(CONTEXT, x INT, y STRING, z OCIDATE);
    PROCEDURE plsToC_demoExternal_proc (x PLS_INTEGER, y VARCHAR2, z DATE)
    AS LANGUAGE C
        NAME "C_demoExternal"
        LIBRARY SomeLib
        WITH CONTEXT
        PARAMETERS(CONTEXT, x INT, y STRING, z OCIDATE);

    PROCEDURE plsToC_InBody_proc (x PLS_INTEGER, y VARCHAR2, z DATE)
    AS LANGUAGE C
        NAME "C_InBody"
        LIBRARY SomeLib
        WITH CONTEXT
        PARAMETERS(CONTEXT, x INT, y STRING, z OCIDATE);
    PROCEDURE plsToJ_InBody_proc (x PLS_INTEGER, y VARCHAR2, z DATE)
    IS LANGUAGE JAVA
        NAME 'pkg1.class4.J_InBody_meth(int,java.lang.String,java.sql.Date)';
END;
```

21.9 コール仕様による外部Cプロシージャへのパラメータの受渡し

コール仕様によって、PL/SQLデータ型とCデータ型とのマッピングが可能になります。

外部Cプロシージャへのパラメータの受渡しは、次のような理由で複雑になります。

- 使用可能なPL/SQLのセットが、Cデータ型のセットと1対1で対応していない。
- Cとは異なり、PL/SQLにはNULLかどうかというRDBMS概念が含まれる。このため、PL/SQLパラメータはNULLにできるが、CパラメータはNULLにできない。
- 外部プロシージャで、CHAR、LONG RAW、RAWおよびVARCHAR2パラメータの現在の長さまたは最大長が必要。
- 外部プロシージャで、CHAR、VARCHAR2およびCLOBパラメータに関する文字セット情報が必要。
- PL/SQLで、外部プロシージャによって戻された値の現在の長さ、最大長またはNULL状態が必要。

ノート:



C 外部プロシージャに渡すことができるパラメータの最大数は 128 です。ただし、FLOAT 型または DOUBLE 型のパラメータを値渡しする場合は、最大数は 127 以下になります。どのくらい減るかは、パラメータの個数およびオペレーティング・システムによって異なります。目安としては、値渡しされる FLOAT 型または DOUBLE 型のパラメータ 1 つを 2 つ分として数えます。

トピック:

- [データ型の指定](#)
- [外部データ型のマッピング](#)
- [BY VALUEまたはBY REFERENCEによるパラメータの受渡し](#)
- [仮パラメータの宣言](#)
- [デフォルトのデータ型マッピングのオーバーライド](#)
- [プロパティの指定](#)

関連項目:

データ型マッピングの詳細は、[「データ型の指定」](#)を参照してください。

21.9.1 データ型の指定

パラメータは外部プロシージャに直接渡さないでください。かわりに、パラメータのPL/SQLデータ型を指定し、外部プロシージャを発行したPL/SQLプロシージャに渡します。PL/SQLデータ型は、デフォルトの外部データ型にマップされます([表21-1](#)を参照)。

ノート:



PL/SQL データ型の BINARY_INTEGER と PLS_INTEGER は同一です。わかりやすくするため、このマニュアルでは BINARY_INTEGER と PLS_INTEGER の両方を PLS_INTEGER とします。

表21-1 パラメータ・データ型のマッピング

PL/SQLデータ型	サポートされている外部データ型	デフォルトの外部データ型
BINARY_INTEGER BOOLEAN PLS_INTEGER	[UNSIGNED] CHAR [UNSIGNED] SHORT [UNSIGNED] INT [UNSIGNED] LONG SB1, SB2, SB4 UB1, UB2, UB4 SIZE_T	INT
NATURAL ^{Foot 1} NATURALN ^{Foot 1} POSITIVE ^{Foot 1} POSITIVEN ^{Foot 1} SIGNTYPE ^{Foot 1}	[UNSIGNED] CHAR [UNSIGNED] SHORT [UNSIGNED] INT [UNSIGNED] LONG SB1, SB2, SB4 UB1, UB2, UB4 SIZE_T	UNSIGNED INT
FLOAT	FLOAT	FLOAT

PL/SQLデータ型	サポートされている外部データ型	デフォルトの外部データ型
REAL		
DOUBLE PRECISION	DOUBLE	DOUBLE
CHAR CHARACTER LONG NCHAR NVARCHAR2 ROWID VARCHAR VARCHAR2	STRING OCISTRING	STRING
LONG RAW RAW	RAW OCIRAW	RAW
BFILE BLOB CLOB NCLOB	OCILOBLOCATOR	OCILOBLOCATOR
NUMBER DEC Foot 1 DECIMAL Foot 1 INT Foot 1 INTEGER Foot 1 NUMERIC Foot 1 SMALLINT Foot 1	OCINUMBER	OCINUMBER
DATE	OCIDATE	OCIDATE
TIMESTAMP TIMESTAMP WITH TIME ZONE TIMESTAMP WITH LOCAL TIME ZONE	OCIDateTime	OCIDateTime
INTERVAL DAY TO SECOND INTERVAL YEAR TO MONTH	OCIInterval	OCIInterval
composite object types: ADTs	dvoid	dvoid
composite object types: collections (associative arrays, varrays, nested tables)	OCICOLL	OCICOLL

脚注1

このPL/SQLデータ型は、コール仕様でAS EXTERNALを使用する場合にのみ準拠します。

21.9.2 外部データ型のマッピング

外部データ型はそれぞれCデータ型にマップされ、データ型変換が暗黙的に実行されます。外部データ型はそれぞれCデータ型にマップされ、データ型変換が暗黙的に実行されます。Cプロトタイプ・パラメータの宣言時のエラーを回避するには、[表21-2](#)を参照してください。この表には、特定の外部データ型およびPL/SQLパラメータ・モードに指定するCデータ型が示されています。たとえば、OUTパラメータの外部データ型がSTRING型の場合は、Cプロトタイプにはデータ型char *を指定します。

表21-2 外部データ型のマッピング

PL/SL型に対応する外部データ型	INモードまたはRETURNモードの場合、Cプロトタイプで指定	参照によるINモードまたは参照によるRETURNモードの場合、Cプロトタイプで指定	IN OUTモードまたはOUTモードの場合、Cプロトタイプで指定
CHAR	char	char *	char *
UNSIGNED CHAR	unsigned char	unsigned char *	unsigned char *
SHORT	short	short *	short *
UNSIGNED SHORT	unsigned short	unsigned short *	unsigned short *
INT	int	int *	int *
UNSIGNED INT	unsigned int	unsigned int *	unsigned int *
LONG	long	long *	long *
UNSIGNED LONG	unsigned long	unsigned long *	unsigned long *
CHAR	char	char *	char *
UNSIGNED CHAR	unsigned char	unsigned char *	unsigned char *
SHORT	short	short *	short *
UNSIGNED SHORT	unsigned short	unsigned short *	unsigned short *
INT	int	int *	int *
UNSIGNED INT	unsigned int	unsigned int *	unsigned int *
LONG	long	long *	long *
UNSIGNED LONG	unsigned long	unsigned long *	unsigned long *
SIZE_T	size_t	size_t *	size_t *
SB1	sb1	sb1 *	sb1 *
UB1	ub1	ub1 *	ub1 *

PL/SL型に対応する外部データ型	INモードまたはRETURNモードの場合、Cプロトタイプで指定	参照によるINモードまたは参照によるRETURNモードの場合、Cプロトタイプで指定	IN OUTモードまたはOUTモードの場合、Cプロトタイプで指定
SB2	sb2	sb2 *	sb2 *
UB2	ub2	ub2 *	ub2 *
SB4	sb4	sb4 *	sb4 *
UB4	ub4	ub4 *	ub4 *
FLOAT	float	float *	float *
DOUBLE	double	double *	double *
STRING	char *	char *	char *
RAW	unsigned char *	unsigned char *	unsigned char *
OCILOBLOCATOR	OCIlobLocator *	OCIlobLocator **	OCIlobLocator **
OCINUMBER	OCINumber *	OCINumber *	OCINumber *
OCISTRING	OCIString *	OCIString *	OCIString *
OCIRAW	OCIRaw *	OCIRaw *	OCIRaw *
OCIDATE	OCIDate *	OCIDate *	OCIDate *
OCICOLL	OCIColl * or OCIArray * or OCITable *	OCIColl ** or OCIArray ** or OCITable **	OCIColl ** or OCIArray ** or OCITable **
OCITYPE	OCIType *	OCIType *	OCIType *
TD0	OCIType *	OCIType *	OCIType *
ADT (final types)	dvoid*	dvoid*	dvoid*
ADT (nonfinal types)	dvoid*	dvoid*	dvoid**

コンポジット・データ型は、自己記述型ではありません。記述は、**型記述子オブジェクト**(TDO)に格納されています。オブジェクトおよびオブジェクトのインジケータ構造体に事前定義のOCIデータ型はありませんが、Oracle Databaseの**Object Type Translator**(OTT)によって生成されるデータ型を使用する必要があります。INDICATORおよびコンポジット・オブジェクトのオプションのTDO引数には、通常、Cデータ型のOCIType*があります。

REFおよびコレクション引数のOCICOLLはオプションで、完全性を保つためにのみ存在しています。REFまたはコレクション型を別のデータ型にマップすること、および別のデータ型をREFまたはコレクション型にマップすることはできません。

21.9.3 BY VALUEまたはBY REFERENCEによるパラメータの受渡し

BY VALUEを指定すると、スカラーINおよびRETURN引数が値渡しされます(デフォルト)。また、BY REFERENCEを指定すると、参照によって渡すことができます。

デフォルトまたはBY REFERENCEを指定した場合は、スカラーIN OUTおよびOUT引数が参照によって渡されます。IN OUTおよびOUT引数でのBY VALUEの指定はC用にはサポートされていません。BY REFERENCE/VALUE句の使用は、デフォルトで値渡しされる外部データ型に制限されます。これは、次の外部データ型のINおよびRETURN引数に適用されます。

```
[UNSIGNED] CHAR
[UNSIGNED] SHORT
[UNSIGNED] INT
[UNSIGNED] LONG
SIZE_T
SB1
SB2
SB4
UB1
UB2
UB4
FLOAT
DOUBLE
```

このリストに記載されていない外部データ型のINおよびRETURN引数、IN OUTおよびOUT引数は、すべて参照によって渡されます。

21.9.4 仮パラメータの宣言

一般に、外部プロシージャを発行するPL/SQLプロシージャは、次の例に示されているように、仮パラメータのリストを宣言します。

ノート:



この項の例には、次のようなデータ構造を設定しないと機能しないものもあります。

```
CREATE LIBRARY MathLib AS '/tmp/math.so';

CREATE OR REPLACE FUNCTION Interp_func (
```

```
/* Find the value of y at x degrees using Lagrange interpolation: */ x IN FLOAT, y IN FLOAT)
RETURN FLOAT AS LANGUAGE C NAME "Interp_func" LIBRARY MathLib;
```

それぞれの仮パラメータ宣言では、名前、パラメータ・モードおよび(デフォルトの外部データ型にマップされる)PL/SQLデータ型が指定されます。これが、外部プロシージャが必要とするすべての情報である可能性があります。これがすべてではない場合は、PARAMETERS句を使用して追加情報を指定できます。

- デフォルト以外の外部データ型

- パラメータの現在の長さまたは最大長
- パラメータのNULL/NOT NULLインジケータ
- 文字セットのIDおよび形式
- リスト内のパラメータの位置
- INパラメータを渡す方法(値による方法または参照による方法)

PARAMETERS句を使用する場合、次のことを認識しておく必要があります。

- 各仮パラメータには、PARAMETERS句に対応するパラメータが必要です。
- WITH CONTEXT句を含める場合は、パラメータ・リスト内でのコンテキスト・ポインタの位置を示すパラメータCONTEXTを指定する必要があります。
- 外部プロシージャがファンクションの場合は、パラメータRETURNを指定できますが、末尾に指定する必要があります。RETURNを指定しないときは、デフォルトの外部型が使用されます。

21.9.5 デフォルトのデータ型マッピングのオーバーライド

PARAMETERS句を使用して、デフォルトのデータ型マッピングをオーバーライドすることができる場合があります。たとえば、外部データ型INTから外部データ型CHARにPL/SQLデータ型のBOOLEANをマッピングしなおすことができます。

21.9.6 プロパティの指定

PARAMETERS句は、PL/SQLの仮パラメータおよびファンクション結果に関する追加情報を外部プロシージャに渡すために使用することもできます。これは、次のプロパティを1つ以上指定して行います。

```
INDICATOR [{STRUCT | TDO}]
LENGTH
DURATION
MAXLEN
CHARSETID
CHARSETFORM
SELF
```

[表21-3](#)に、使用可能外部データ型とデフォルト外部データ型、PL/SQLデータ型および特定のプロパティに使用できるPL/SQLパラメータ・モードを示します。(CからPL/SQLにデータを戻す場合に指定する)MAXLENは、INパラメータには適用できません。

表21-3 プロパティおよびデータ型

プロパティ	使用可能外部データ型(C)	デフォルトの外部データ型(C)	使用可能PL/SQLデータ型	使用可能PL/SQLモード	デフォルトのPL/SQL受渡し方法
INDICATOR	SHORT	SHORT	all scalars	IN IN OUT OUT RETURN	BY VALUE BY REFERENCE BY REFERENCE BY REFERENCE
LENGTH	[UNSIGNED] SHORT [UNSIGNED] INT [UNSIGNED]	INT INT	CHAR LONG RAW RAW	IN IN OUT OUT RETURN	BY VALUE BY REFERENCE BY REFERENCE

プロパティ	使用可能外部データ 型(C)	デフォルトの外部 データ型(C)	使用可能 PL/SQLデータ 型	使用可能 PL/SQLモー ド	デフォルトの PL/SQL受渡し方 法
	LONG		VARCHAR2		BY REFERENCE
MAXLEN	[UNSIGNED] SHORT [UNSIGNED] LONG	INT INT	CHAR LONG RAW RAW VARCHAR2	IN OUT OUT RETURN	BY REFERENCE BY REFERENCE BY REFERENCE
CHARSETID CHARSETFORM	UNSIGNED SHORT UNSIGNED INT UNSIGNED LONG	UNSIGNED INT	CHAR CLOB VARCHAR2	IN IN OUT RETURN	BY VALUE BY REFERENCE BY REFERENCE BY REFERENCE

次の例では、PARAMETERS句はPL/SQLの仮パラメータおよびファンクション結果のプロパティを指定します。

```
CREATE OR REPLACE FUNCTION plsToCparse_func (
  x    IN PLS_INTEGER,
  y    IN OUT CHAR)
RETURN CHAR AS LANGUAGE C
  LIBRARY c_utils
  NAME "C_parse"
  PARAMETERS (
    x,          -- stores value of x
    x INDICATOR, -- stores null status of x
    y,          -- stores value of y
    y LENGTH,   -- stores current length of y
    y MAXLEN,   -- stores maximum length of y
    RETURN INDICATOR,
    RETURN);
```

このPARAMETERS句を使用すると、Cプロトタイプは次のようになります。

```
char *C_parse( int x, short x_ind, char *y, int *y_len, int *y_maxlen,
  short *retind );
```

Cプロトタイプ内の追加パラメータは、INDICATOR(x用)、LENGTH(y用)およびMAXLEN(y用)の他に、PARAMETERS句のファンクション結果のINDICATORにも対応します。パラメータRETURNは結果値を格納し、C関数識別子に対応します。

トピック:

- [INDICATOR](#)
- [LENGTHおよびMAXLEN](#)
- [CHARSETIDおよびCHARSETFORM](#)
- [パラメータの再配置](#)
- [SELF](#)
- [BY REFERENCE](#)
- [WITH CONTEXT](#)
- [言語間のパラメータ・モードのマッピング](#)

21.9.6.1 INDICATOR

INDICATORは、別のパラメータがNULLかどうかを示す値を持つパラメータです。PL/SQLでは、RDBMSにおけるNULLかどうかという概念が言語に組み込まれているため、インジケータは必要ありません。ただし、外部プロシージャは、パラメータまたはファンクションの結果がNULLであるかどうかを判断する必要がある場合があります。また、外部プロシージャは、戻り値がNULLでありそれに応じた処理が必要であることをサーバーに指示する必要がある場合もあります。

このような場合は、プロパティINDICATORを使用して、インジケータを仮パラメータに関連付けることができます。PL/SQLプロシージャがファンクションの場合、[「プロパティの指定」](#)に示すとおり、ファンクション結果にインジケータに関連付けられます。

インジケータの値を調べるには、定数OCI_IND_NULLおよびOCI_IND_NOTNULLを使用できます。インジケータがOCI_IND_NULLと等しい場合は、関連付けられているパラメータまたはファンクション結果はNULLです。インジケータがOCI_IND_NOTNULLと等しい場合は、関連付けられているパラメータまたはファンクション結果はNULLではありません。

INパラメータは読み込み専用ですが、この場合、INDICATORは(BY REFERENCEを指定しないかぎり)値渡しされ、(BY REFERENCEを指定しても)読み込み専用です。OUT、IN OUTおよびRETURNパラメータの場合、INDICATORはデフォルトで参照によって渡されます。

INDICATORにはSTRUCTオプションまたはTDOオプションを指定することもできます。INDICATORをオブジェクトのプロパティとして指定することはサポートされていないため、また、オブジェクトの引数にはINDICATORスカラーのかわりに完全なインジケータ構造体があるため、STRUCTオプションを使用して指定する必要があります。コンポジット・オブジェクトおよびコレクションには、型記述子オブジェクト(TDO)を使用する必要があります。

21.9.6.2 LENGTHおよびMAXLEN

PL/SQLでは、RAWパラメータまたは文字列パラメータの長さを示す標準的な方法はありません。ただし、このようなパラメータの長さの受渡しを外部プロシージャとの間で行う必要がある場合があります。プロパティLENGTHおよびMAXLENを使用すると、仮パラメータの現在の長さおよび最大長を格納するパラメータを指定できます。

ノート:



型が RAW または LONG RAW のパラメータの場合は、プロパティ LENGTH を使用してください。また、そのパラメータが IN OUT および NULL か、または OUT および NULL の場合は、対応する C パラメータを長さ 0(ゼロ)に設定してください。

INパラメータの場合は、LENGTHは(BY REFERENCEを指定しないかぎり)値渡しされ、読み込み専用です。OUT、IN OUTおよびRETURNパラメータの場合は、LENGTHはデフォルトで参照によって渡されます。

MAXLENはINパラメータには適用されません。OUT、IN OUTおよびRETURNパラメータの場合、MAXLENは参照によって渡され、読み込み専用です。

21.9.6.3 CHARSETIDおよびCHARSETFORM

Oracle Databaseではグローバル化・サポートを提供していますが、これを使用すると、シングルバイトおよびマルチバイトの文字データを処理し、文字セット間で変換を行うことができます。また、アプリケーションを異なる言語環境で実行することもできます。

デフォルトでは、サーバーおよびエージェントが同じ\$ORACLE_HOME値を使用する場合、エージェントはサーバー(ALTER SESSION文で指定した設定を含む)と同一のグローバル化・サポート設定を使用します。

エージェントが別の\$ORACLE_HOME(2つの異なる別名およびシンボリック・リンクによって同じ場所が指定されている場合でも)

で実行されている場合、文字セット以外はサーバーと同じグローバリゼーション・サポート設定が使用されます。エージェント用のデフォルト文字セットはエージェントの環境設定NLS_LANGおよびNLS_NCHARによって定義されます。

プロパティCHARSETIDおよびCHARSETFORMでは、渡された文字データの生成元である、デフォルト以外の文字セットが識別されます。CHAR、CLOBおよびVARCHAR2型パラメータにより、CHARSETIDおよびCHARSETFORMを使用して文字セットのIDおよび形式を外部プロシージャに渡すことができます。

INパラメータの場合、CHARSETIDおよびCHARSETFORMは(BY REFERENCEを指定しないかぎり)値渡しされ、(BY REFERENCEを指定した場合でも)読み込み専用です。OUT、IN OUTおよびRETURNパラメータの場合、CHARSETIDおよびCHARSETFORMは参照によって渡され、読み込み専用です。

これらのプロパティのOCI属性名は、OCI_ATTR_CHARSET_IDおよびOCI_ATTR_CHARSET_FORMです。

関連項目:

- OCI_ATTR_CHARSET_IDとOCI_ATTR_CHARSET_FORMの詳細は、[『Oracle Call Interfaceプログラマーズ・ガイド』](#)を参照してください
- OCIで各国語データを使用する場合の詳細は、[『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。

21.9.6.4 パラメータの再配置

外部プロシージャの仮パラメータは、それぞれPARAMETERS句の中に対応するパラメータが必要です。PL/SQLではパラメータを位置ではなく名前によって対応付けるため、対応するパラメータの位置は異なる可能性があります。ただし、PARAMETERS句および外部プロシージャのためのCプロトタイプでは、同数のパラメータが同じ順序で必要です。

21.9.6.5 SELF

SELFは、オブジェクト型のメンバーであるプロシージャに常に存在する引数で、オブジェクトのインスタンスそのものです。ほとんどの場合、この引数は暗黙的で、PL/SQLプロシージャの引数リストには含まれていません。ただし、SELFは、PARAMETERS句の引数として明示的に指定する必要があります。

たとえば、個人の名前および生年月日で構成されるPersonオブジェクトを作成し、さらに、このオブジェクト型の表を作成します。後でこの表の各Personの年令を判断する必要があるとします。

1. SQL*Plusでは、Personオブジェクト型は次のように作成できます。

```
CREATE OR REPLACE TYPE Person1_typ AS OBJECT (  
    Name_      VARCHAR2(30),  
    B_date     DATE,  
    MEMBER FUNCTION calcAge_func RETURN NUMBER  
);  
/
```

2. このメンバー・ファンクションの本文を次のように宣言します。

```
CREATE OR REPLACE TYPE BODY Person1_typ AS  
    MEMBER FUNCTION calcAge_func RETURN NUMBER  
    AS LANGUAGE C  
    NAME "age"  
    LIBRARY agelib  
    WITH CONTEXT  
    PARAMETERS (  
        CONTEXT,  
        SELF,
```



```

        SELF INDICATOR STRUCT,
        SELF TDO,
        RETURN INDICATOR
    );
END;
/

```

(通常、メンバー・ファンクションはPL/SQLで実装しますが、この例では外部プロシージャとなっています。)

calcAge_funcメンバー関数は引数をとらず、数値を戻します。メンバー関数は、常に、対応付けられているオブジェクト型のインスタンスに対してコールされます。オブジェクト・インスタンス自体は、常にメンバー関数の暗黙的な引数になります。暗黙的な引数を参照するには、SELFキーワードを使用します。これは、PARAMETERS句の中でSELFへの参照をサポートすることによって、外部プロシージャ構文内に組み込まれています。

3. 一致する表を作成し、移入します。

```

CREATE TABLE Person_tab OF Person1_tpy;
INSERT INTO Person_tab
VALUES ('BOB', TO_DATE('14-MAY-85'));
INSERT INTO Person_tab
VALUES ('JOHN', TO_DATE('22-DEC-71'));

```

4. この表から対象の情報を取得します。

```

SELECT p.name, p.b_date, p.calcAge_func() FROM Person_tab p;
NAME                                B_DATE      P.CALCAGE_
-----
BOB                                14-MAY-85      0
JOHN                               22-DEC-71      0

```

外部メンバー関数と、Object-Type-Translator(OTT)によって生成される構造体の定義を実装するCコードを次に示します。

```

#include <oci.h>
struct PERSON
{
    OCIStrng    *NAME;
    OCIDate      B_DATE;
};
typedef struct PERSON PERSON;

struct PERSON_ind
{
    OCIInd      _atomic;
    OCIInd      NAME;
    OCIInd      B_DATE;
};
typedef struct PERSON_ind PERSON_ind;

OCINumber *age (ctx, person_obj, person_obj_ind, tdo, ret_ind)
OCIExtProcContext *ctx;
PERSON          *person_obj;
PERSON_ind      *person_obj_ind;
OCITYPE         *tdo;
OCIInd          *ret_ind;
{
    sword      err;
    text       errbuf[512];
    OCIEnv     *envh;
    OCISvcCtx  *svch;
    OCIError   *errh;
    OCINumber  *age;
    int        inum = 0;

```

```

sword      status;

/* get OCI Environment */
err = OCIExtProcGetEnv( ctx, &envh, &svch, &errh );
/* initialize return age to 0 */
age = (OCINumber *)OCIExtProcAllocCallMemory(ctx, sizeof(OCINumber));
status = OCINumberFromInt(errh, &inum, sizeof(inum), OCI_NUMBER_SIGNED,
                           age);
if (status != OCI_SUCCESS)
{
    OCIExtProcRaiseExcp(ctx, (int)1476);
    return (age);
}
/* return NULL if the person object is null or the birthdate is null */
if ( person_obj_ind->_atomic == OCI_IND_NULL ||
    person_obj_ind->B_DATE == OCI_IND_NULL )
{
    *ret_ind = OCI_IND_NULL;
    return (age);
}
/* The actual implementation to calculate the age is left to the reader,
   but an easy way of doing this is a callback of the form:
       select trunc(months_between(sysdate, person_obj->b_date) / 12)
       from DUAL;
*/
*ret_ind = OCI_IND_NOTNULL;
return (age);
}

```

21.9.6.6 BY REFERENCE

Cでは、INスカラー・パラメータは値渡しされる(パラメータの値が渡される)か、参照によって渡す(値へのポインタが渡される)ことができます。スカラーを指すポインタが外部プロシージャで必要な場合は、BY REFERENCE句を指定し、参照によってパラメータを渡します。

```

CREATE OR REPLACE PROCEDURE findRoot_proc (
    x IN DOUBLE PRECISION)
AS LANGUAGE C
    LIBRARY c_utils
    NAME "C_findRoot"
    PARAMETERS (
        x BY REFERENCE);

```

この場合、Cプロトタイプは次のように指定します。

```
void C_findRoot(double *x);
```

デフォルト(PARAMETERS句がない場合に使用)は次のように指定します。

```
void C_findRoot(double x);
```

21.9.6.7 WITH CONTEXT

WITH CONTEXT句を含めることによって、外部プロシージャで、パラメータ、例外、メモリー割当ておよびユーザー環境に関する情報にアクセスできるようになります。WITH CONTEXT句は、コンテキスト・ポインタが外部プロシージャに渡されることを指定します。たとえば、次のPL/SQLファンクションを作成するとします。

```

CREATE OR REPLACE FUNCTION getNum_func (
    x IN REAL)
RETURN PLS_INTEGER AS LANGUAGE C
    LIBRARY c_utils
    NAME "C_getNum"

```

```
WITH CONTEXT
PARAMETERS (
    CONTEXT,
    x BY REFERENCE,
    RETURN INDICATOR);
```

Cプロトタイプは次のとおりです。

```
int C_getNum(
    OCIExtProcContext *with_context,
    float *x,
    short *retind);
```

コンテキスト・データ構造体は外部プロシージャに対して不透明ですが、外部プロシージャによってコールされるサービス・プロシージャでは使用できます。

PARAMETERS句も含める場合は、パラメータ・リスト内でのコンテキスト・ポインタの位置を示すパラメータCONTEXTを指定する必要があります。PARAMETERS句を省略すると、コンテキスト・ポインタは外部プロシージャに渡される最初のパラメータです。

21.9.6.8 言語間のパラメータ・モードのマッピング

PL/SQLは、IN、IN OUTおよびOUTパラメータ・モード、および値を戻すプロシージャのRETURN句をサポートします。

21.10 CALL文による外部プロシージャの実行

これで、Javaクラス・メソッドまたは外部Cプロシージャが発行されました。次は、これをコールします。

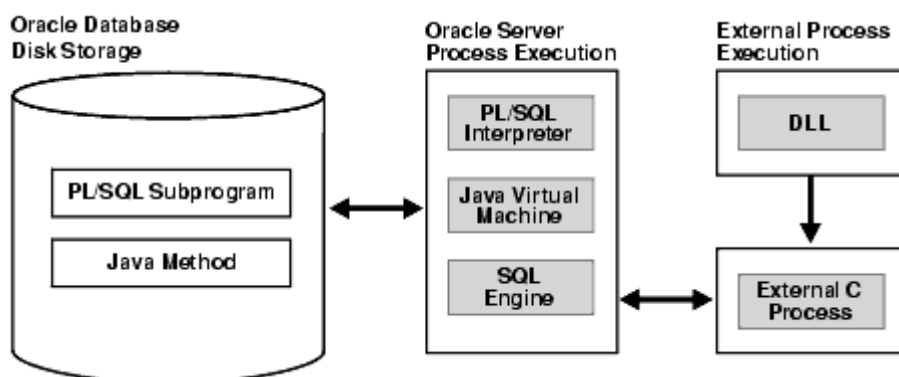
外部プロシージャは直接コールしないでください。かわりに、CALL文を使用して、外部プロシージャを発行したPL/SQLプロシージャをコールしてください。

このようなコールは、通常のPL/SQLプロシージャに対するコールと同じようにコーディングして、次のような場所で利用できます。

- 無名ブロック
- スタンドアロンおよびパッケージ・プロシージャ
- オブジェクト型のメソッド
- データベース・トリガー
- SQL文(パッケージ関数のみへのコール)。

サーバー側またはクライアント側(たとえば、Oracle Formsのようなツール内)で実行されるPL/SQLブロックまたはプロシージャは、外部プロシージャをコールできます。サーバー側では、外部プロシージャは別のプロセスのアドレス空間内で実行されるため、データベースが保護されます。[図21-1](#)に、Oracle Databaseと外部プロシージャ間の処理を示します。

図21-1 Oracle Databaseと外部プロシージャ



トピック:

- [外部プロシージャの事前条件](#)
- [CALL文の構文](#)
- [Javaクラス・メソッドのコール](#)
- [外部Cプロシージャのコール](#)

関連項目:

[CALL文の構文](#)

21.10.1 外部プロシージャの事前条件

外部プロシージャをコールする前に、実行環境に存在する権限、許可およびシノニムを考慮する必要があります。

トピック:

- [外部プロシージャの権限](#)
- [許可の管理](#)
- [外部プロシージャのシノニムの作成](#)

21.10.1.1 外部プロシージャの権限

外部プロシージャがCALL仕様経由でコールされるとき、プロシージャは実行者権限ではなく定義者権限によって実行されます。

実行者権限で実行されるプログラムは、特定のスキーマには限定されません。コール側のサイトで実行され、コールした側の可視性と許可でデータベース項目(表やビューなど)にアクセスします。一方、定義者権限で実行されるプログラムでは、プログラムが定義されているスキーマに限定されます。このプログラムは定義を行う側で、サイトの定義者のスキーマ内の定義サイトで実行され、定義者の可視性および許可でデータベース項目にアクセスします。

21.10.1.2 許可の管理

外部プロシージャをコールするには、ユーザーはコール仕様に対するEXECUTE権限が必要です。この権限を付与するには、GRANT文を使用します。たとえば、この文により、ユーザーjohndoeはコール仕様がplsToJ_demoExternal_procである外部プロシージャをコールできます。

```
GRANT EXECUTE ON plsToJ_demoExternal_proc TO johndoe;
```

プロシージャをコールする必要があるユーザーにのみ、コール仕様に対するEXECUTE権限を付与します。

関連項目:

GRANT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。

21.10.1.3 外部プロシージャのシノニムの作成

開発者またはDBAは、便宜上、CREATE PUBLIC SYNONYM文を使用して外部プロシージャのシノニムを作成できます。次の例では、すべてのユーザーがアクセス可能なパブリック・シノニムをDBAが作成します。PUBLICを指定しない場合、シノニムはプライベートになり、そのスキーマ内以外ではアクセスできません。

```
CREATE PUBLIC SYNONYM Rfac FOR johndoe.RecursiveFactorial;
```

21.10.2 CALL文の構文

SQLのCALL文から外部プロシージャをコールします。CALL文はSQL*Plusから対話形式で実行できます。構文は次のとおりです。

```
CALL [schema.][{object_type_name | package_name}]procedure_name[@dblink_name]
[(parameter_list)] [INTO :host_variable][INDICATOR][:indicator_variable];
```

これは、SELECT myproc(...) FROM DUALという形式のSQL文を使用してプロシージャmyprocを実行することと基本的には同じです。ただし、この場合は、SELECTの実行に関連するオーバーヘッドが発生しません。

たとえば、この章で発行したplsToC_demoExternal_procを動的SQLを使用してコールする無名PL/SQLブロックを次に示します。PL/SQLは、外部CプロシージャC_demoExternal_procに3つのパラメータを渡します。

```
DECLARE
  xx NUMBER(4);
  yy VARCHAR2(10);
  zz DATE;
BEGIN
  EXECUTE IMMEDIATE
    'CALL plsToC_demoExternal_proc(:xxx, :yyy, :zzz)' USING xx,yy,zz;
END;
```

CALL文のセマンティクスは、同等のBEGIN ENDブロックと同じです。

ノート:



CALL は、単独で PL/SQL の BEGIN END ブロックに入れることのできない唯一の SQL 文です。これは、BEGIN END ブロック内での EXECUTE IMMEDIATE 文の一部に使用できます。

21.10.3 Javaクラス・メソッドのコール

[「Javaクラス・メソッドのパブリッシュ」](#)で発行するJ_calcFactorialクラス・メソッドをコールするには:

1. 次のように、SQL*Plusのホスト変数を2つ宣言して初期化します。

```
VARIABLE x NUMBER
VARIABLE y NUMBER
EXECUTE :x := 5;
```

2. J_calcFactorialをコールします。

```
CALL J_calcFactorial(:x) INTO :y;
PRINT y
```

結果:

```
Y
-----
120
```

21.10.4 外部Cプロシージャのコール

外部Cプロシージャをコールするには、PL/SQLが適切なDLLパスを検出する必要があります。PL/SQLエンジンは、プロシージャ宣言のAS LANGUAGE句からのライブラリ別名に基づいて、パスをデータ・ディクショナリから取得します。

次に、PL/SQLはリスナー・プロセスに警告を出し、リスナー・プロセスがセッション固有のエージェントを起動します。デフォルトでは、このエージェントをextprocといいます。listener.oraファイルで別の名前指定できます。リスナーは接続をエージェントに渡し、PL/SQLはDLLの名前、外部プロシージャの名前およびパラメータをエージェントに渡します。

次に、エージェントはDLLをロードし、外部プロシージャを実行します。また、エージェントは(例外を呼び出すなどの)サービス・コールおよびOracle Databaseへのコールバックも処理します。最後に、エージェントは、外部プロシージャによって戻された値をPL/SQLに渡します。

ノート:



DLL のキャッシュはいくらか発生しますが、DLL はキャッシュ内に残らない可能性があります。したがって、グローバル変数は DLL には格納しないでください。

外部プロシージャが完了した後、エージェントはOracle Databaseセッションの終わりまでアクティブなまま残ります。ログオフするとエージェントが停止されます。この結果、何度コールしても、エージェントを起動するのは1回で済みますが、演算によるメリットがコールのコストを上回るときにのみ外部プロシージャをコールするようにします。

ここでは、[「外部Cプロシージャの発行」](#)で発行したPL/SQLファンクションplsCallsCdivisor_funcを、無名ブロックからコールします。PL/SQLは整数パラメータを2つ外部関数Cdivisor_funcに渡し、この関数が最大公約数を戻します。

```
DECLARE
  g    PLS_INTEGER;
  a    PLS_INTEGER;
  b    PLS_INTEGER;
CALL plsCallsCdivisor_func(a, b);
IF g IN (2,4,8) THEN ...
```

21.11 複数言語のプログラム・エラーおよび例外処理

AS EXTERNALコール仕様がTYPE仕様部またはPACKAGE仕様部で検出された場合、PL/SQLコンパイラはコンパイル時例外を呼び出します。

Cプログラムでは、OCIExtproc関数を使用して例外を呼び出すことができます。

21.12 外部Cプロシージャでのサービス・ルーチンの使用

サービス・ルーチンが外部プロシージャからコールされると、サービス・ルーチンは例外を呼び出し、メモリーを割り当て、サーバーへのコールバック用のOCIハンドルをコールします。サービス・ルーチンを使用するには、WITH CONTEXT句を指定する必要があります。WITH CONTEXT句を使用すると、コンテキスト構造体を外部プロシージャに渡すことができます。コンテキスト構造体は、ヘッダー・ファイルociextp.hの中に次のように宣言されています。

```
typedef struct OCIExtProcContext OCIExtProcContext;
```

ノート:



ociextp.h は、Linux および UNIX では\$ORACLE_HOME/plsql/public にあります。

サービス・プロシージャ:

- [OCIExtProcAllocCallMemory](#)
- [OCIExtProcRaiseExcp](#)
- [OCIExtProcRaiseExcpWithMsg](#)

21.12.1 OCIExtProcAllocCallMemory

OCIExtProcAllocCallMemory サービス・ルーチンは、外部プロシージャ・コールの間、nバイトのメモリを割り当てます。この関数によって割り当てられるメモリは、PL/SQLに制御が戻った直後に自動的に解放されます。

ノート:



外部プロシージャでは、このサービス・ルーチンによって割り当てられたメモリが自動的に処理されるため、C 関数の free をコールしないでください。

この関数のCプロトタイプは、次のとおりです。

```
dvoid *OCIExtProcAllocCallMemory(  
    OCIExtProcContext *with_context,  
    size_t amount);
```

パラメータwith_contextおよびamountは、それぞれ、コンテキスト・ポインタと割り当てられるバイト数です。この関数は、割り当てられたメモリを指す型が未定のポインタを返します。戻り値が0(ゼロ)の場合は、失敗を示します。

SQL*Plusで、外部関数plsToC_concat_funcを次のように発行するとします。

```
CREATE OR REPLACE FUNCTION plsToC_concat_func (  
    str1 IN VARCHAR2,  
    str2 IN VARCHAR2)  
RETURN VARCHAR2 AS LANGUAGE C  
NAME "concat"  
LIBRARY stringlib  
WITH CONTEXT  
PARAMETERS (  
    CONTEXT,  
    str1    STRING,  
    str1    INDICATOR short,  
    str2    STRING,  
    str2    INDICATOR short,  
    RETURN INDICATOR short,  
    RETURN LENGTH short,  
    RETURN STRING);
```

C_concatは、コールされたときに2つの文字列を連結し、結果を返します。

```
select plsToC_concat_func('hello ', 'world') from DUAL;  
PLSTOC_CONCAT_FUNC('HELLO','WORLD')  
-----  
hello world
```


いずれの文字列もNULLの場合は、結果もNULLになります。次の例で示されるように、C_concatはOCIExtProcAllocCallMemoryを使用して結果文字列のメモリーを割り当てます。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>
#include <ociextp.h>
char *concat(ctx, str1_i, str2, str2_i, ret_i, ret_l)
OCIExtProcContext *ctx;
char *str1;
short str1_i;
char *str2;
short str2_i;
short *ret_i;
short *ret_l;
{
    char *tmp;
    short len;
    /* Check for null inputs. */
    if ((str1_i == OCI_IND_NULL) || (str2_i == OCI_IND_NULL))
    {
        *ret_i = (short)OCI_IND_NULL;
        /* PL/SQL has no notion of a NULL ptr, so return a zero-byte string. */
        tmp = OCIExtProcAllocCallMemory(ctx, 1);
        tmp[0] = '\0';
        return(tmp);
    }
    /* Allocate memory for result string, including NULL terminator. */
    len = strlen(str1) + strlen(str2);
    tmp = OCIExtProcAllocCallMemory(ctx, len + 1);

    strcpy(tmp, str1);
    strcat(tmp, str2);

    /* Set NULL indicator and length. */
    *ret_i = (short)OCI_IND_NOTNULL;
    *ret_l = len;
    /* Return pointer, which PL/SQL frees later. */
    return(tmp);
}
#ifdef LATER
static void checkerr (/*_ OCIError *errhp, sword status _*/);
void checkerr(errhp, status)
OCIError *errhp;
sword status;
{
    text errbuf[512];
    sb4 errcode = 0;
    switch (status)
    {
    case OCI_SUCCESS:
        break;
    case OCI_SUCCESS_WITH_INFO:
        (void) printf("Error - OCI_SUCCESS_WITH_INFO\n");
        break;
    case OCI_NEED_DATA:
        (void) printf("Error - OCI_NEED_DATA\n");
        break;
    case OCI_NO_DATA:
        (void) printf("Error - OCI_NODATA\n");
        break;
    case OCI_ERROR:
        (void) OCIErrorGet((dvoid *)errhp, (ub4) 1, (text *) NULL, &errcode,
                           errbuf, (ub4) sizeof(errbuf), OCI_HTYPE_ERROR);
        (void) printf("Error - %.*s\n", 512, errbuf);
    }
}
```

```

        break;
    case OCI_INVALID_HANDLE:
        (void) printf("Error - OCI_INVALID_HANDLE\n");
        break;
    case OCI_STILL_EXECUTING:
        (void) printf("Error - OCI_STILL_EXECUTE\n");
        break;
    case OCI_CONTINUE:
        (void) printf("Error - OCI_CONTINUE\n");
        break;
    default:
        break;
}
}
char *concat(ctx, str1, str1_i, str2, str2_i, ret_i, ret_l)
OCIExtProcContext *ctx;
char *str1;
short str1_i;
char *str2;
short str2_i;
short *ret_i;
short *ret_l;
{
    char *tmp;
    short len;
    /* Check for null inputs. */
    if ((str1_i == OCI_IND_NULL) || (str2_i == OCI_IND_NULL))
    {
        *ret_i = (short)OCI_IND_NULL;
        /* PL/SQL has no notion of a NULL ptr, so return a zero-byte string. */
        tmp = OCIExtProcAllocCallMemory(ctx, 1);
        tmp[0] = '\0';
        return(tmp);
    }
    /* Allocate memory for result string, including NULL terminator. */
    len = strlen(str1) + strlen(str2);
    tmp = OCIExtProcAllocCallMemory(ctx, len + 1);
    strcpy(tmp, str1);
    strcat(tmp, str2);
    /* Set NULL indicator and length. */
    *ret_i = (short)OCI_IND_NOTNULL;
    *ret_l = len;
    /* Return pointer, which PL/SQL frees later. */
    return(tmp);
}
/*=====*/
int main(char *argv, int argc)
{
    OCIExtProcContext *ctx;
    char *str1;
    short str1_i;
    char *str2;
    short str2_i;
    short *ret_i;
    short *ret_l;
    /* OCI Handles */
    OCIEnv *envhp;
    OCIServer *srvhp;
    OCISvcCtx *svchp;
    OCIError *errhp;
    OCISession *authp;
    OCISstmt *stmthp;
    OCILobLocator *clob, *blob;
    OCILobLocator *Lob_loc;
    /* Initialize and Logon */
    (void) OCIInitialize((ub4) OCI_DEFAULT, (dvoid *)0,
                        (dvoid * (*)(dvoid *, size_t)) 0,

```

```

        (dvoid * (*)(dvoid *, dvoid *, size_t))0,
        (void (*)(dvoid *, dvoid *)) 0 );
(void) OCIEnvInit( (OCIEnv **) &envhp,
                  OCI_DEFAULT, (size_t) 0,
                  (dvoid **) 0 );
(void) OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &errhp, OCI_HTYPE_ERROR,
                      (size_t) 0, (dvoid **) 0);
/* Server contexts */
(void) OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &srvhp, OCI_HTYPE_SERVER,
                      (size_t) 0, (dvoid **) 0);
/* Service context */
(void) OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &svchp, OCI_HTYPE_SVCCTX,
                      (size_t) 0, (dvoid **) 0);
/* Attach to Oracle Database */
(void) OCIServerAttach( srvhp, errhp, (text *)"", strlen(""), 0);
/* Set attribute server context in the service context */
(void) OCIAttrSet ((dvoid *) svchp, OCI_HTYPE_SVCCTX,
                  (dvoid *)srvhp, (ub4) 0,
                  OCI_ATTR_SERVER, (OCIError *) errhp);
(void) OCIHandleAlloc((dvoid *) envhp,
                    (dvoid **)&authp, (ub4) OCI_HTYPE_SESSION,
                    (size_t) 0, (dvoid **) 0);

(void) OCIAttrSet((dvoid *) authp, (ub4) OCI_HTYPE_SESSION,
                (dvoid *) "samp", (ub4)4,
                (ub4) OCI_ATTR_USERNAME, errhp);

(void) OCIAttrSet((dvoid *) authp, (ub4) OCI_HTYPE_SESSION,
                (dvoid *) "password", (ub4) 4,
                (ub4) OCI_ATTR_PASSWORD, errhp);
/* Begin a User Session */
checkerr(errhp, OCISessionBegin ( svchp, errhp, authp, OCI_CRED_RDBMS,
                                (ub4) OCI_DEFAULT));
(void) OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX,
                (dvoid *) authp, (ub4) 0,
                (ub4) OCI_ATTR_SESSION, errhp);
/* -----User Logged In-----*/
printf ("user logged in %n");
/* allocate a statement handle */
checkerr(errhp, OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &stmthp,
                                OCI_HTYPE_STMT, (size_t) 0, (dvoid **) 0));
checkerr(errhp, OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &lob_loc,
                                (ub4) OCI_DTYPE_LOB,
                                (size_t) 0, (dvoid **) 0));
/* ----- subprogram called here-----*/
printf ("calling concat...%n");
concat(ctx, str1, str1_i, str2, str2_i, ret_i, ret_l);
return 0;
}
#endif

```

21.12.2 OCIExtProcRaiseExcp

OCIExtProcRaiseExcpサービス・ルーチンは、事前定義の例外を呼び出します。この例外には、1から32,767の有効な Oracle Databaseエラー番号が含まれている必要があります。この例外には、1から32,767の有効な Oracle Databaseエラー番号が含まれている必要があります。クリーン・アップ処理後、外部プロシーダをすぐに戻す必要があります(OUTまたはIN OUTパラメータには値は何も割り当てられません)。この関数のCプロトタイプは、次のとおりです。

```

int OCIExtProcRaiseExcp(
    OCIExtProcContext *with_context,
    size_t errnum);

```

パラメータwith_contextおよびerror_numberは、それぞれ、コンテキスト・ポインタおよびOracle Databaseエラー番

号です。戻り値のOCIEXTPROC_SUCCESSおよびOCIEXTPROC_ERRORは、正常終了および失敗を示します。

SQL*Plusで、外部プロシージャplsTo_divide_procを次のように発行するとします。

```
CREATE OR REPLACE PROCEDURE plsTo_divide_proc (  
    dividend IN PLS_INTEGER,  
    divisor   IN PLS_INTEGER,  
    result    OUT FLOAT)  
AS LANGUAGE C  
    NAME "C_divide"  
    LIBRARY MathLib  
    WITH CONTEXT  
    PARAMETERS (  
        CONTEXT,  
        dividend INT,  
        divisor   INT,  
        result    FLOAT);
```

C_divideは、コールされたときに2つの数値の比率を計算します。次の例で示すように、除数が0(ゼロ)の場合、C_divideはOCIExtProcRaiseExcpを使用して事前定義の例外ZERO_DIVIDEを次のように呼び出します。

```
void C_divide (ctx, dividend, divisor, result)  
OCIExtProcContext *ctx;  
int    dividend;  
int    divisor;  
float  *result;  
{  
    /* Check for zero divisor. */  
    if (divisor == (int)0)  
    {  
        /* Raise exception ZERO_DIVIDE, which is Oracle Database error 1476. */  
        if (OCIExtProcRaiseExcp(ctx, (int)1476) == OCIEXTPROC_SUCCESS)  
        {  
            return;  
        }  
        else  
        {  
            /* Incorrect parameters were passed. */  
            assert(0);  
        }  
    }  
    *result = (float)dividend / (float)divisor;  
}
```

21.12.3 OCIExtProcRaiseExcpWithMsg

OCIExtProcRaiseExcpWithMsgサービス・ルーチンはユーザー定義例外を呼び出し、ユーザー定義エラー・メッセージを戻します。この関数のCプロトタイプは、次のとおりです。

```
int OCIExtProcRaiseExcpWithMsg(  
    OCIExtProcContext *with_context,  
    size_t    error_number,  
    text      *error_message,  
    size_t    len);
```

パラメータwith_context、error_number、error_messageは、それぞれ、コンテキスト・ポインタ、Oracle Databaseエラー番号、エラー・メッセージ・テキストです。パラメータlenは、エラー・メッセージの長さを格納します。メッセージがNULLで終了する文字列の場合、lenは0(ゼロ)です。戻り値のOCIEXTPROC_SUCCESSおよびOCIEXTPROC_ERRORは、正常終了および失敗を示します。

この前の例では、外部プロシージャplsTo_divide_procを発行しました。次の例では、別の実装方法を使用します。ここ

では、除数が0(ゼロ)の場合、C_divideはOCIExtProcRaiseExcpWithMsgを使用してユーザー定義例外を呼び出します。

```
void C_divide (ctx, dividend, divisor, result)
OCIExtProcContext *ctx;
int    dividend;
int    divisor;
float  *result;
/* Check for zero divisor. */
if (divisor == (int)0)
{
    /* Raise a user-defined exception, which is Oracle Database error 20100,
       and return a null-terminated error message. */
    if (OCIExtProcRaiseExcpWithMsg(ctx, (int)20100,
        "divisor is zero", 0) == OCIEXTPROC_SUCCESS)
    {
        return;
    }
    else
    {
        /* Incorrect parameters were passed. */
        assert(0);
    }
}
*result = dividend / divisor;
}
```

21.13 外部Cプロシージャを使用したコールバックの実行

コールバックを有効にするには、OCIExtProcGetEnv関数を使用します。

トピック:

- [OCIExtProcGetEnv](#)
- [OCIコールバック用のオブジェクト・サポート](#)
- [コールバックに関する制限事項](#)
- [外部Cプロシージャのデバッグ](#)
- [例: 外部Cプロシージャのコール](#)
- [外部Cプロシージャのグローバル変数](#)
- [外部Cプロシージャの静的変数](#)
- [外部Cプロシージャに関する制限事項](#)

21.13.1 OCIExtProcGetEnv

OCIExtProcGetEnvサービス・ルーチンを使用すると、外部プロシージャ・コール中にデータベースへのOCIコールバックが可能になります。この関数によって取得された環境ハンドルは、既存の接続を再利用してデータベースに戻ります。データベースとの接続を新しく確立する必要がある場合は、このハンドルを使用できません。自分でハンドルを作成する必要があります。

この関数のCプロトタイプは、次のとおりです。

```
sword OCIExtProcGetEnv ( OCIExtProcContext *with_context,
    OCIEnv envh,
    OCISvcCtx svch,
    OCIError errh )
```

パラメータwith_contextはコンテキスト・ポインタで、パラメータenvh、svch、errhは、それぞれ、OCI環境、サービス、エラー・ハンドルです。戻り値のOCIEXTPROC_SUCCESSおよびOCIEXTPROC_ERRORは、正常終了および失敗を示します。

外部CプロシージャおよびJavaクラス・メソッドでは、どちらもデータベースをコールバックしてSQL操作を実行できます。実際の例は、[「例：外部Cプロシージャのコール」](#)を参照してください。

ノート:



コールバックは、必ずしも同一セッションで発生するとはかぎりません。OCI logon を使用して、別セッション内でSQL 文を実行することができます。

Oracle Database上で実行される外部Cプロシージャでは、サービス・ルーチンをコールしてOCI環境およびサービス・ハンドルを取得できます。OCIを使用すると、SQL文およびPL/SQLサブプログラムの実行、データのフェッチ、およびLOBの操作のためにコールバックを使用することができます。コールバックおよび外部プロシージャは同一のユーザー・セッションおよびトランザクション・コンテキストで動作するため、ユーザー権限も同一です。

SQL*Plusで、次のスクリプトを実行するとします。

```
CREATE TABLE Emp_tab (empno NUMBER(10))
CREATE PROCEDURE plsToC_insertIntoEmpTab_proc (
    empno PLS_INTEGER)
AS LANGUAGE C
    NAME "C_insertEmpTab"
    LIBRARY insert_lib
    WITH CONTEXT
    PARAMETERS (
        CONTEXT,
        empno LONG);
```

後で、次のように外部プロシージャplsToC_insertIntoEmpTab_procからサービス・ルーチンOCIExtProcGetEnvをコールできます。

```
#include <stdio.h>
#include <stdlib.h>
#include <oratypes.h>
#include <oci.h> /* includes ociextp.h */
...
void C_insertIntoEmpTab (ctx, empno)
OCIExtProcContext *ctx;
long empno;
{
    OCIEnv      *envhp;
    OCISvcCtx   *svchp;
    OCIError    *errhp;
    int         err;
    ...
    err = OCIExtProcGetEnv(ctx, &envhp, &svchp, &errhp);
    ...
}
```

コールバックを使用しない場合は、oci.hを含める必要はなく、ociextp.hを含めます。

21.13.2 OCIコールバック用のオブジェクト・サポート

外部プロシージャからオブジェクト関連コールバックを実行するために、extprocエージェント内のOCI環境がオブジェクト・モードで完全に初期化されています。この環境へのハンドルは、OCIExtProcGetEnvプロシージャを使用して取得します。

オブジェクト・ランタイム環境を使用すると、OCIによって提供される静的および動的なオブジェクト・サポートが使用できます。静的サポートを使用するには、OTTを使用して該当するオブジェクト型用のC構造体を生成し、次に、従来のCコードを使用してオブジェクトの属性にアクセスします。

外部プロシージャの作成時に型がわからないオブジェクトについては、かわりの動的なオブジェクト・アクセス方法によって、OCIDescribeAnyがコールされてその型の属性およびメソッド情報が取得されます。その後、OCIObjectGetAttrおよびOCIObjectSetAttrをコールして、属性値を取得して設定します。

現在の外部プロシージャ・モデルはステートレスのため、コールバックを実行する外部プロシージャごとにOCIExtProcGetEnvをコールするか、または、OCIExtProcサービス・ルーチンをコールします。外部プロシージャがコールされるたびに、コール後にコールバック・メカニズムがクリーン・アップされ、OCIハンドルが解放されます。

21.13.3 コールバックに関する制限事項

コールバックでは、次のSQL文およびOCIサブプログラムはサポートされていません。

- COMMITなどのトランザクション制御文
- CREATEなどのデータ定義文
- 次のオブジェクト指向OCIサブプログラム

```
OCIObjectNew
OCIObjectPin
OCIObjectUnpin
OCIObjectPinCountReset
OCIObjectLock
OCIObjectMarkUpdate
OCIObjectUnmark
OCIObjectUnmarkByRef
OCIObjectAlwaysLatest
OCIObjectNotAlwaysLatest
OCIObjectMarkDeleteByRef
OCIObjectMarkDelete
OCIObjectFlush
OCIObjectFlushRefresh
OCIObjectGetTypeRef
OCIObjectGetObjectRef
OCIObjectExists
OCIObjectIsLocked
OCIObjectIsDirtied
OCIObjectIsLoaded
OCIObjectRefresh
OCIObjectPinTable
OCIObjectArrayPin
OCICacheFlush,
OCICacheFlushRefresh,
OCICacheRefresh
OCICacheUnpin
OCICacheFree
OCICacheUnmark
OCICacheGetObjects
OCICacheRegister
```

-
- OCIGetPieceInfoなどのポーリング・モードOCIサブプログラム
- 次のOCIサブプログラム

```
OCIEnvInit
OCIInitialize
```



```
OCIPasswordChange
OCIServerAttach
OCIServerDetach
OCISessionBegin
OCISessionEnd
OCISvcCtxToLda
OCITransCommit
OCITransDetach
OCITransRollback
OCITransStart
```

また、OCIサブプログラムOCIHandleAllocでは、次のハンドル・タイプはサポートされていません。

```
OCI_HTYPE_SERVER
OCI_HTYPE_SESSION
OCI_HTYPE_SVCCTX
OCI_HTYPE_TRANS
```

21.13.4 外部Cプロシージャのデバッグ

外部プロシージャが失敗する場合は、通常、そのプロトタイプに不具合があります。プロトタイプが、PL/SQLによって内部的に生成されたプロトタイプと一致しないということです。これは、互換性のないCデータ型を指定した場合に発生する可能性があります。たとえば、型がREALのOUTパラメータを渡すためにfloat *を指定したとします。float、double *または他のCデータ型を指定すると、結果が不一致になります。

このような場合は、次のエラーを受け取ることがあります。

```
lost RPC connection to external routine agent
```

このエラーは、外部プロシージャによってコア・ダンプが発生したため、extprocが異常終了したことを表します。Cプロトタイプ・パラメータの宣言時にエラーを回避するには、前述の表を参照してください。

PL/SQLでは、外部プロシージャのデバッグを支援する目的でユーティリティ・パッケージDEBUG_EXTPROCが提供されています。このパッケージをインストールするには、PL/SQLデモ・ディレクトリにあるスクリプトdbgextp.sqlを実行します。(ディレクトリの場所については、ご使用のOracle Databaseのインストール・ガイドまたはユーザズ・ガイドを参照してください。)

このパッケージを使用するには、dbgextp.sqlの指示に従ってください。Oracle Databaseアカウントには、パッケージに対するEXECUTE権限およびCREATE LIBRARY権限が必要です。

ノート:



DEBUG_EXTPROC は、実行中のプロセスにアタッチできるデバッガ付きのプラットフォームでのみ動作します。

21.13.5 例: 外部Cプロシージャのコール

PL/SQLデモ・ディレクトリには、外部プロシージャのコール方法を示すスクリプトextproc.sqlもあります。付属ファイルextproc.cには、外部プロシージャ用のCソース・コードが含まれています。

デモを実行するには、extproc.sqlにある指示に従ってください。SCOTTアカウントを使用する必要があり、このアカウントにはCREATE LIBRARY権限が必要です。

21.13.6 外部Cプロシージャのグローバル変数

グローバル変数は関数の外で宣言され、その値はプログラムのすべての関数によって共有されます。したがって、外部プロシ

ジャの場合、DLL内のすべての関数がグローバル変数の値を共有します。グローバル変数は関数の存続期間を超えて存続するデータの格納にも使用されます。しかし、グローバル変数の使用は、次の2つの理由のため、お薦めしません。

- スレッド処理

エージェント・プロセスは非スレッド構成になっているため、一度にアクティブになる関数は1つのみです。マルチスレッドの extproc エージェントの場合は、一度に複数の関数をアクティブにできます。その場合は、それらが同時にグローバル変数にアクセスしようとして、正常な結果が得られなくなる可能性があります。

- DLLのキャッシング

グローバル変数にデータを格納することで、関数 func1 が関数 func2 にデータを渡すとして、func1 の完了後 DLL キャッシュはアンロードされ、この結果、グローバル変数のすべての値が失われる可能性があります。次に、func2 が実行されるときに、DLL は再ロードされ、グローバル変数はすべて 0 (ゼロ) に初期化されます。

21.13.7 外部Cプロシージャの静的変数

静的変数には外部変数および内部変数の2種類があります。外部静的変数は、特殊なグローバル変数であり、使用をお薦めしません。内部静的変数は、特定の関数に対するローカルな変数ですが、関数がアクティブになるたびに生成、消滅するのではなく、存在したまま残ります。このため、この変数は1つの関数内にプライベートで永続的な記憶域を提供します。このような変数は、同一関数を後でコールするときにデータを渡すために使用します。ただし、[「外部Cプロシージャのグローバル変数」](#)で説明するDLLキャッシュ機能によって、すべてのDLLはコール間でアンロードおよびリロードできるため、内部静的変数の値が失われます。



:

動的リンク・ライブラリの作成方法は、RDBMS サブディレクトリ/public にあるテンプレート makefile を参照してください。

外部プロシージャをコールする場合:

- INパラメータには書き込みを行わないでください。OUTパラメータの容量をオーバーフローさせないようにしてください。(PL/SQLでは、このようなエラー条件のランタイム・チェックは実行されません。)
- OUTパラメータまたは関数結果を読み込まないでください。
- IN OUTパラメータとOUTパラメータおよび関数結果には、必ず値を代入してください。代入しないと、外部プロシージャが正常に戻りません。
- WITH CONTEXTおよびPARAMETERS句を含める場合は、パラメータ・リスト内でのコンテキスト・ポインタの位置を示すパラメータCONTEXTを指定する必要があります。
- PARAMETERS句を含めたときに、外部プロシージャが関数の場合は、パラメータRETURNを最後の位置に指定する必要があります。
- 各仮パラメータには、PARAMETERS句に対応するパラメータが必要です。また、PARAMETERS句のパラメータのデータ型が、Cプロトタイプのデータ型と互換性があることも確認します。これは、暗黙の変換が実行されないためです。
- 型がRAWまたはLONG RAWのパラメータの場合は、プロパティLENGTHを使用してください。また、そのパラメータがIN OUTまたはOUTおよびNULLの場合は、対応するCパラメータの長さを0(ゼロ)に設定する必要があります。

21.13.8 外部Cプロシージャに関する制限事項

外部プロシージャには次の制限が適用されます。

- この機能は、DLLをサポートするプラットフォームのみで使用可能です。
- CプロシージャまたはCからコール可能なプロシージャのみがサポートされます。
- 分散トランザクションと組み合わせた外部プロシージャ・コールアウトはサポートされません。
- PL/SQLカーソル変数またはレコードを外部プロシージャに渡すことはできません。レコードについては、かわりにオブジェクト型のインスタンスを使用できます。
- LIBRARY副次句には、リモート・ライブラリを指定するデータベース・リンクは使用できません。
- 外部プロシージャに渡すことができるパラメータの最大数は128です。ただし、FLOAT型またはDOUBLE型のパラメータを値渡しする場合は、最大数は127以下になります。どのくらい減るかは、パラメータの個数およびオペレーティング・システムによって異なります。目安としては、値渡しされるFLOAT型またはDOUBLE型のパラメータ1つを2つ分として数えます。

22 Oracle XAを使用したアプリケーションの開発

ノート:

次の理由により、可能であれば XA は使用しないでください。



- XA ではパフォーマンスが低下する可能性があります。
- XA ではインダウト・トランザクションが発生する可能性があります。
- XA は、リカバリ可能な停止後に、アプリケーションを続行する機能を拡張するために、Oracle Database 12c リリース 1 (12.1.0.1)の機能の利点を活用できない場合もあります。

不可避と思われる(たとえば、Oracle と Oracle 以外のリソースを同一トランザクションで使用する必要がある)場合でも、XA の使用を回避できることがあります。

この章では、Oracle XAライブラリの使用方法について説明します。通常、このライブラリはトランザクション・モニターで作業するアプリケーションで使用します。XAの機能は、トランザクションが複数のデータベースと相互処理するアプリケーションで最も効果的です。

トピック:

- [X/Open DTP](#)
- [Oracle XAライブラリ・サブプログラム](#)
- [XAアプリケーションの開発およびインストール](#)
- [XAアプリケーションのトラブルシューティング](#)
- [Oracle XAの問題および制限事項](#)

関連項目:

- 基本的なアーキテクチャを含む、XAの概要についての分散TP: XA仕様。
<https://www2.opengroup.org/ogsys/jsp/publications/PublicationDetails.jsp?publicationid=11144>を参照してください。
- Oracle XAライブラリのバックグラウンドおよび参照情報は、『[Oracle Call Interfaceプログラマーズ・ガイド](#)』を参照してください。
- ライブラリ・リンク・ファイル名の詳細は、プラットフォーム固有のOracle Databaseのマニュアルを参照してください。
- プラットフォーム固有のOracle XAライブラリの変更点、不具合および制限事項については、READMEを参照してください。

22.1 X/Open DTP

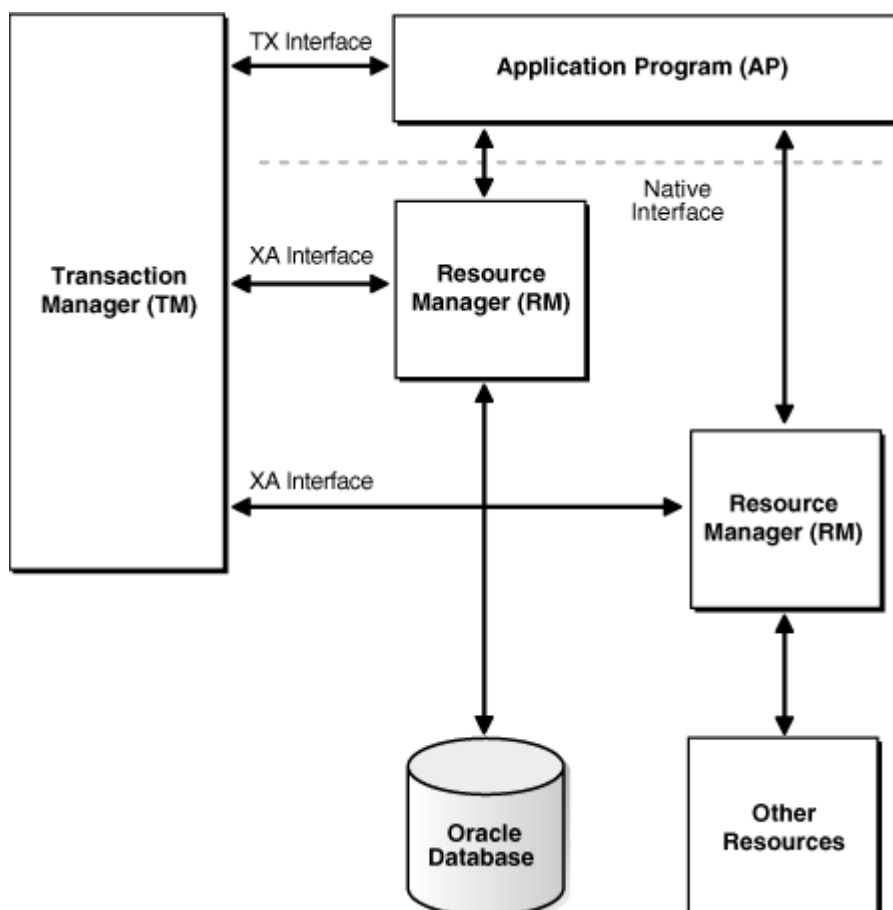
X/Open Distributed Transaction Processing(DTP)アーキテクチャは、複数のアプリケーション・プログラム(AP)が複数

の異なるリソース・マネージャ(RM)から提供されるリソースを共有できるようにするための、標準のアーキテクチャまたはインタフェースを定義しています。APとRM間の作業を調整し、グローバル・トランザクションを実現します。

Oracle XAライブラリは、X/Openソフトウェア・アーキテクチャのXAインタフェース仕様に準拠しています。Oracle XAライブラリは外部インタフェースであり、Oracleのクライアント側トランザクション・マネージャ(TM)以外のクライアント側TMがグローバル・トランザクションを調整可能にすることで分散トランザクションにOracle Database以外のRMを組み込めるようにします。たとえば、クライアント・アプリケーションでは、Oracle DatabaseトランザクションとNTFSファイル・システム内のトランザクションを1つのグローバル・トランザクションとして管理できます。

[図22-1](#)に、X/Open DTPモデルの例を示します。

図22-1 DTPモデルの例



トピック:

- [DTP用語](#)
- [必須のパブリック情報](#)

22.1.1 DTP用語

- [リソース・マネージャ\(RM\)](#)
- [分散トランザクション](#)
- [ブランチ](#)
- [トランザクション・マネージャ\(TM\)](#)
- [トランザクション処理モニター\(TPM\)](#)
- [2フェーズ・コミット・プロトコル](#)

- [アプリケーション・プログラム\(AP\)](#)
- [TXインタフェース](#)
- [密結合と疎結合](#)
- [動的登録と静的登録](#)

リソース・マネージャ(RM)

リソース・マネージャは、障害発生後に通常の状態に戻ることができる共有かつリカバリ可能なリソースを制御します。たとえば、リレーショナル・データベース、トランザクション・キューおよびトランザクション・ファイル・システムなどです。Oracle DatabaseはRMであり、障害発生後にREDOログおよびロールバック・セグメントを使用して一貫性のある状態に復帰します。

分散トランザクション

分散トランザクションは、複数の分散リソースに対する更新を伴い、分散RM間で「すべてまたはなし」のセマンティクスを必要とするクライアント・トランザクションであり、**グローバル・トランザクション**とも呼ばれます。

ブランチ

ブランチは、1つのRMに含まれる1つの作業単位です。複数のブランチが1つのグローバル・トランザクションを構成します。Oracle Databaseの場合、各ブランチはデータベース・サーバー内のローカル・トランザクションにマップします。

トランザクション・マネージャ(TM)

トランザクション・マネージャは、トランザクション境界を指定するためのAPIを提供し、コミットおよびリカバリを管理します。TMでは、分散RM間で「すべてまたはなし」のセマンティクスを提供する2フェーズ・コミット・エンジンが実装されます。

外部TMとは、Oracle Databaseの外部に常駐する中間層コンポーネントです。通常、データベースには固有の内部TMがあります。業界標準に準拠したTMを使用することで、Oracle Databaseは単一のトランザクション内で他の異機種間RMと共存できます。

トランザクション処理モニター(TPM)

TMは、通常、次のようなトランザクション処理モニター(TPM)が提供します。

- Oracle Tuxedo
- IBM Transarc Encina
- IBM CICS

TPMは、トランザクション要求を発行するクライアント・プロセスとその要求を処理するバックエンド・サーバー間の要求フローを調整します。TPMは、基本的にネットワーク上に分散されているアプリケーション・サーバーやRMなどのように、様々な種類のバックエンド・プロセスに対してサービスを要求するトランザクションを調整します。

TPMは、分散トランザクションを完了するために必要なコミットまたはロールバックを同期化します。TPMのTM部分が、分散コミットおよび分散ロールバックの発生するタイミングを制御します。このため、分散APでTPMを利用する場合、TPMのTM部分が2フェーズ・コミット・プロトコルを制御します。RMを使用して、TMはこのタスクを実行します。

TMは分散コミットまたはロールバックを制御するため、XAインタフェースを介してOracle Database(または他のRM)と直接通信する必要があります。TMは、トランザクション内のすべてのRMに関する情報に基づいて、「[Oracle XAライブラリ・サブプログラム](#)」で説明するOracle XAライブラリ・サブプログラムを使用してOracle Databaseにトランザクションの処理方法を指示します。

2フェーズ・コミット・プロトコル

Oracle XAライブラリ・インタフェースは、2フェーズ・コミット・プロトコルに従います。各イベントの順序は次のとおりです。

1. 準備フェーズでは、TMは各RMに対して、トランザクションの任意の部分をコミットできるように要求します。これが可能な場合は、RMは準備ができた状態を記録し、TMに肯定的に応答します。可能でない場合は、RMはすべての作業をロールバックし、TMに否定的に応答し、そのトランザクションに関する情報を消去します。プロトコルによって、アプリケーションまたはRMは、準備フェーズが完了する前にトランザクションを一方的にロールバックできます。
2. フェーズ2では、TMはコミット決定を記録し、トランザクションに参加しているすべてのRMに対してコミットまたはロールバックを発行します。TMは、すべてのRMがフェーズ1に対して肯定的に応答した場合にのみ、RMのコミットを発行できます。

アプリケーション・プログラム(AP)

アプリケーション・プログラムは、トランザクション境界を定義し、トランザクションを構成するアクションを指定します。たとえば、APはプリコンパイラでもOracle Call Interface(OCI)プログラムでもかまいません。APは、RMのネイティブ・インタフェース(たとえばSQL)を使用して、RMのリソースを操作します。

TXインタフェース

アプリケーション・プログラムは、TMを介し、**TX**と呼ばれるインタフェースを使用してすべてのトランザクション制御操作を開始および完了します。APがXAインタフェースを直接使用することはありません。APが中間層で分岐するブランチを認識することはありません。アプリケーション・スレッドがブランチの作業を明示的に結合、休止、一時停止および再開するかわりに、トランザクション処理モニターのTM部分がAPのためにグローバル・トランザクションのブランチを管理します。最終的に、APはTMをコールしてすべてをコミットするか、または何もコミットしません。

ノート:



TX インタフェースおよびその関連のサブプログラムのネーミング規則はベンダー固有です。たとえば、tx_open コールは、使用中のシステムでは tp_open と呼ばれている場合があります。また、トランザクション RPC への入力時など、このコールが暗黙的な場合があります。詳細は、トランザクション処理モニターに付属のマニュアルを参照してください。

密結合と疎結合

RMですべての分離セマンティクスについて単一エンティティとみなされるアプリケーション・スレッドは、密結合となります。密結合ブランチは、相互の変更を認識する必要があります。さらに、外部クライアントは、密結合セットの変更をすべて認識するか、まったく認識しない必要があります。アプリケーション・スレッドが密結合されていない場合は、疎結合となります。

動的登録と静的登録

Oracle Databaseは、動的登録と静的登録の両方をサポートします。動的登録の場合、RMは最初にアプリケーション・コールバックを実行します。静的登録の場合、関係のないRMがあっても、最初に、各RMに対してxa_startをコールする必要があります。

22.1.2 必須のパブリック情報

Oracle Databaseは、リソース・マネージャとして、[表22-1](#)に示す情報を発行する必要があります。

表22-1 Oracle Databaseが発行する必要のあるXA機能

XA機能	Oracle Databaseの詳細

XA機能	Oracle Databaseの詳細
xa_switch_t 構造体	Oracle Database の xa_switch_t 構造体名は、静的登録の場合は xaosw、動的登録の場合は xaoswd です。これらの構造体には、RM のエントリ・ポイントおよびその他の情報が含まれています。
xa_switch_t リソース・マネージャ	xa_switch_t 構造体内の Oracle Database のリソース・マネージャ名は Oracle_XA です。
クローズ文字列	xa_close で使用されるクローズ文字列は無視され、NULL として扱われます。
オープン文字列	xa_open で使用されるオープン文字列の書式の詳細は、 「xa_open 文字列の定義」 を参照してください。
ライブラリ	Oracle XA を使用するアプリケーションをリンクするために必要なライブラリには、プラットフォーム固有の名前が付けられています。TPM 固有のライブラリをリンクする必要があることを除けば、通常のプリコンパイラまたは OCI プログラムをリンクする手順と同じです。 プリコンパイラ(Pro*C/C++、Pro*COBOL など)を使用していない場合は、 \$ORACLE_HOME/rdbms/lib/xaonsl.o または \$ORACLE_HOME/rdbms/lib32/xaonsl.o (64 ビット・プラットフォーム上での 32 ビット・アプリケーションの場合)をリンクしてください。
要件	なし。XA をサポートする機能は Standard Edition および Enterprise Edition に含まれています。

22.2 Oracle XAライブラリ・サブプログラム

Oracle XAライブラリ・サブプログラムを使用することで、TMはトランザクションの処理方法をOracle Databaseに指示できます。一般に、TMはxa_openを使用してリソースをオープンする必要があります。通常、リソースはAPでtx_openをコールした結果としてオープンされます。一部のTMでは、アプリケーションが開始したときに、暗黙的にxa_openをコールします。

同様に、アプリケーションがリソースの使用を完了したときに発生するクローズがあります(xa_closeを使用)。このクローズが発生するのは、APがtx_closeをコールしたとき、またはアプリケーションが終了したときです。

TMはRMに対して、次のように他の複数のタスクを実行するように指示します。

- トランザクションの開始、およびトランザクションに対するIDの対応付け
- トランザクションのロールバック
- トランザクションの準備およびコミット

トピック:

- [Oracle XAライブラリ・サブプログラム](#)
- [Oracle XAインタフェース拡張機能](#)

22.2.1 Oracle XAライブラリ・サブプログラム

[表22-2](#)に、XAライブラリ・サブプログラムを示します。

表22-2 XAライブラリ・サブプログラム

XAサブプログラム	説明:
xa_open	RM に接続します。
xa_close	RM との接続を切断します。
xa_start	トランザクションを開始し、指定のトランザクション ID(XID)に対応付けるか、またはプロセスと既存トランザクションに対応付けます。
xa_end	指定された XID からプロセスを切断します。
xa_rollback	指定された XID に対応付けられているトランザクションをロールバックします。
xa_prepare	指定された XID に対応付けられているトランザクションを準備します。これは、2 フェーズ・コミット・プロトコルの第 1 のフェーズです。
xa_commit	指定された XID に対応付けられているトランザクションをコミットします。これは、2 フェーズ・コミット・プロトコルの第 2 のフェーズです。
xa_recover	準備されたヒューリスティックにコミットまたはロールバックされたトランザクションのリストを取得します。
xa_forget	指定された XID に対応付けられているヒューリスティックに完了したトランザクションの情報を消去します。

一般に、APでは、xa_open文字列で実行されるロールを理解する以外に、[表22-2](#)に示したサブプログラムについて考慮する必要はありません。

22.2.2 Oracle XAインタフェース拡張機能

[表22-3](#)に示すように、Oracle DatabaseのXAインタフェースには、いくつかの関数が追加されています。

表22-3 Oracle XAインタフェース拡張機能

関数	説明:
OCISvcCtx *xaoSvcCtx(text *dbname)	指定された XA 接続の OCI サービス・ハンドルを戻します。 dbname パラメータは、xa_open 文字列で渡された DB パラメータと同じである必要があります。OCI アプリケーションでは、接続ハンドルを取得するために、sqlld2 コールのかわりにこのルーティングを使用できます。このため、OCI アプリケーションは

関数	説明:
	sql lib ライブラリにリンクする必要はありません。サービス・ハンドルは、OCISvcCtxToLda(OCI バージョン 8)を使用して OCI バージョン 7 のログイン・データ領域(LDA)に変換できます。クライアント・アプリケーションは、OCI コールを完了した後で、OCILdaToSvcCtx を使用して Oracle7 の LDA をサービス・ハンドルに変換する必要があります。
OCIEnv *xaoEnv(text *dbname)	指定された XA 接続の OCI 環境ハンドルを戻します。 dbname パラメータは、xa_open 文字列で渡された DB パラメータと同じである必要があります。
int xaosterr(OCISvcCtx *SvcCtx, sb4 error)	Oracle Database エラー・コードを XA エラー・コードに変換します(動的登録にのみ適用可能)。最初のパラメータは、データベースで作業を実行するために使用するサービス・ハンドルです。2 番目のパラメータは、Oracle Database から戻されるエラー・コードです。この関数は、OCI 文から戻されたエラーが xa_start の違反によって発生したものかどうかを判断するために使用します。この関数は、エラーが XA モジュールによって生成されたものでない場合は XA_OK を戻し、エラーが XA モジュールによって生成されたものである場合は有効な XA エラーを戻します。

22.3 XAアプリケーションの開発およびインストール

この項では、Oracle XAライブラリの開発およびインストールについて説明します。

- [DBAまたはシステム管理者の責任](#)
- [アプリケーション開発者の責任](#)
- [xa_open文字列の定義](#)
- [XAアプリケーションの開発およびインストール](#)
- [Oracle XAを使用したトランザクション制御の管理](#)
- [プリコンパイラまたはOCIアプリケーションのTPMアプリケーションへの移行](#)
- [Oracle XAライブラリ・スレッド・セーフティの管理](#)
- [DBMS_XAパッケージの使用](#)

22.3.1 DBAまたはシステム管理者の責任

DBAまたはシステム管理者の責任は次のとおりです。

1. アプリケーション開発者の支援によってオープン文字列を定義します。

2. 静的データ・ディクショナリ・ビューDBA_PENDING_TRANSACTIONSが存在することを確認し、xa_open文字列で指定したすべてのOracleユーザーに対して、このビューのREADまたはSELECT権限を付与します。

自分が作成したペンディング(インダウト)トランザクションをコミットまたはロールバックするOracleユーザーにFORCE TRANSACTION権限を付与します。権限の付与には、コマンドCOMMIT FORCE local_tran_idまたはROLLBACK FORCE local_tran_idを使用します。

他のユーザーが作成したXAトランザクションをコミットまたはロールバックするOracleユーザーにFORCE ANY TRANSACTION権限を付与します。たとえば、ユーザーBが作成したトランザクションをユーザーAがコミットまたはロールバックする場合、ユーザーAにはFORCE ANY TRANSACTION権限が必要となります。

Oracle Databaseリリース7のクライアント・アプリケーションの場合は、Oracle XAライブラリで使用されるすべてのOracle Databaseアカウントに対して、動的パフォーマンス・ビューV\$XATRANS\$のSELECT権限を付与する必要があります。このビューは、XAライブラリのインストール時に作成されている必要があります。必要な場合は、Oracle DatabaseユーザーSYSとしてSQLスクリプトxaview.sqlを実行し、このビューを手動で作成できます。

3. オープン文字列情報を使用して、RMをTPM構成にインストールします。TPMベンダーの指示に従ってください。

DBAまたはシステム管理者は、Oracle Databaseに接続するプロセスをTPMシステムが開始することを認識する必要があります。TPMのマニュアルを参照して、このプロセスのためにどのような環境が存在するのか、ユーザーIDは何である必要があるかを判断してください。この環境で\$ORACLE_HOMEおよび\$ORACLE_SIDに正しい値が設定されていることを確認してください。

4. XAトレース・ファイルが書き込まれるディレクトリに対する書き込み許可を、そのユーザーIDに対して付与します。
5. Oracle XAアプリケーションをオンラインにするために、適切なデータベース・インスタンスを起動します。このタスクは、TPMサーバーを開始する前に実行してください。

関連項目:

- オープン文字列を定義する方法、およびデフォルトとは異なるOracleシステム識別子(SID)またはトレース・ディレクトリを指定する方法の詳細は、[「xa_open文字列の定義」](#)を参照してください。
- catxpend.sqlスクリプトの位置は、使用中のプラットフォーム固有のOracle Databaseのマニュアルを参照してください。

22.3.2 アプリケーション開発者の責任

アプリケーション開発者の責任は次のとおりです。

1. DBAまたはシステム管理者の支援によってオープン文字列を定義します。[「xa_open文字列の定義」](#)を参照してください。
2. アプリケーションを開発します。

プリコンパイラ用のトランザクション指向のSQL文に関する特別な制限事項を守ってください。

関連項目:

[XAアプリケーションの開発およびインストール](#)

3. TPMベンダーの指示に従ってアプリケーションをリンクします。

22.3.3 xa_open文字列の定義

オープン文字列は、トランザクション・モニターがデータベースをオープンするために使用します。オープン文字列内の最大文字数は256文字です。

トピック:

- [xa_open文字列の構文](#)
- [xa_open文字列の必須フィールド](#)
- [xa_open文字列のオプション・フィールド](#)

22.3.3.1 xa_open文字列の構文

[例22-1](#)に示す構文を使用してオープン文字列を定義できます。

次の文字列は、サンプル・パラメータ設定を示しています。

```
ORACLE_XA+DB=MANAGERS+SqlNet=SID1+ACC=P/username/password
+SesTM=10+LogDir=/usr/local/xalog
ORACLE_XA+DB=PAYROLL+SqlNet=SID2+ACC=P/username/password
+SesTM=10+LogDir=/usr/local/xalog
ORACLE_XA+SqlNet=SID3+ACC=P/username/password
+SesTM=10+LogDir=/usr/local/xalog
```

次の各項では、required_fieldsおよびoptional_fieldsプレースホルダに有効なパラメータについて説明します。

- [xa_open文字列の必須フィールド](#)
- [xa_open文字列のオプション・フィールド](#)

ノート:



- オープン文字列の作成時には、必須フィールドおよびオプションのフィールドをどのような順序でも入力できます。
- すべてのフィールド名は大/小文字が区別されません。その値の大/小文字が区別されるかどうかは、プラットフォームによって異なります。
- 実際の情報文字列の一部にプラス文字(+)を使用することはできません。

例22-1 xa_open文字列

```
ORACLE_XA{+required_fields...} [+optional_fields...]
```

22.3.3.2 xa_open文字列の必須フィールド

[例22-1](#)に示したrequired_fieldsプレースホルダは、[表22-4](#)に示す名前/値ペアのいずれかを指します。

表22-4 xa_open文字列の必須フィールド

構文要素	説明:
ACC=P//	ユーザーおよびパスワード情報が明示的には提供されないことと、オペレーティン

構文要素	説明:
	グ・システム認証フォームが使用されることを指定します。
Acc=P/user/password	有効な Oracle Database アカウントのユーザー名とパスワードを指定します。 「DBA またはシステム管理者の責任」 で説明したとおり、HR に DBA_PENDING_TRANSACTIONS 表の READ または SELECT 権限がある ことを確認してください。
SesTm=session_time_limit	システムによりトランザクションが終了されるまでの、あるサービスから次のサービス まで、あるいはサービスからトランザクションのコミットまたはトランザクションまでの最 大許容秒数を指定します。たとえば、SesTM=15 は、セッション・アイドル時間の 上限が 15 秒であることを示します。 たとえば、TPM でクライアントとサーバー間にリモート・サブプログラム・コールが使 用されている場合は、SesTM は、ある RPC の完了から次の RPC の初期化の 間、tx_commit または tx_rollback との間に適用されます。 値が 0(ゼロ)の場合は、制限がないことを示します。値 0(ゼロ)はできるだけ指 定しないでください。問題があった場合に、リソースを長時間拘束することがありま す。また、子プロセスに SesTM=0 がある場合、親プロセスが終了した後は、 SesTM 設定の効果がなくなります。

関連項目:

管理者の認証の詳細は、[『Oracle Database管理者ガイド』](#)を参照してください。

22.3.3.3 xa_open文字列のオプション・フィールド

[例22-1](#)に示したoptional_fieldsプレースホルダは、[表22-5](#)に示す名前/値ペアのいずれかを指します。

表22-5 xa_open文字列のオプション・フィールド

構文要素	説明:
NoLocal= true false	ローカル・トランザクションが許可されるかどうかを指定します。デフォルト値は false で す。アプリケーションでローカル・トランザクションを禁止する必要がある場合は、値を true に設定します。
DB=db_name	Oracle Database プリコンパイラがデータベースの識別に使用する名前を指定しま す。たとえば、DB=payroll は、データベース名が payroll であり、アプリケーショ ン・サーバー・プログラムがこの名前を AT 句で使用することを指定します。 Oracle Database プリコンパイラのデフォルトのデータベースのみを使用する(SQL 文 で AT 句を使用しない)AP では、オープン文字列の DB=db_name 句を省略する必

構文要素	説明:
	要があります。明示的に指定したデータベースを使用するアプリケーションは、 DB=db_name フィールドにそのデータベース名を指定する必要があります。Oracle Database バージョン 7 の OCI プログラムでは、OCI サービス・コンテキストと等価のロ グイン・データ領域(Lda_Def)の正しいコンテキストを取得するために、sqlld2 関数 をコールする必要があります。バージョン 8 の OCI プログラムでは、xaoSvcCtx 関数 をコールして、OCISvcCtx サービス・コンテキストを取得する必要があります。 db_name は SID ではありません。オープンするデータベースを検索するためには使用 されません。このオープン文字列でオープンされたデータベースと、SQL 文を実行するた めに AP で使用される名前とを対応付けます。SID は、TPM アプリケーション・サー バーの ORACLE_SID 環境変数か、オープン文字列の Oracle Net 句で指定した SID から設定されます。Oracle Net 句については、この項で後述します。一部の TPM ベンダーは、同じオープン文字列を使用するサーバー・グループの名前を指定する 方法を提供しています。グループ名と db_name の両方に対して同じ名前を選択す る方が便利な場合があります。
LogDir=log_dir	Oracle XA ライブラリのエラーとトレース情報が格納されるローカル・システム上のパス 名を示します。デフォルトは、ORACLE_HOME が設定されている場合は \$ORACLE_HOME/rdbms/log で、それ以外の場合は現行のディレクトリを指定し ます。たとえば、LogDir=/xa_trace は、/xa_trace ディレクトリにロギング情報 があることを示しています。指定したディレクトリが存在し、アプリケーション・サーバーでそ のディレクトリに確実に書き込めるようにしてください。
Objects= true false	アプリケーションをオブジェクト・モードで初期化するかどうかを指定します。デフォルト値は false です。アプリケーションが、OCIRawAssignBytes などのオブジェクト・モードを 必要とする特定の API コールを使用する必要がある場合、true を指定します。
MaxCur=maximum_#_of_open _cursors	データベースのオープン時に割り当てられるカーソルの数を指定します。これは、プリコン パイラ・オプション maxopencursors と同じ用途で機能します。たとえば、 MaxCur=5 は、プリコンパイラで 5 つのオープン・カーソルがキャッシュされることを示しま す。このパラメータは、ソース・コード内またはコンパイル時に指定したプリコンパイラ・オプ ション maxopencursors をオーバーライドします。
SqlNet=db_link	システムにログインするために使用する Oracle Net データベース・リンクを示します。こ の文字列は、tnsnames.ora のエントリである必要があります。たとえば、文字列 SqlNet=inst1_disp の場合は、tnsnames.ora に定義されていれば、インス タンス 1 の共有サーバーに接続できます。 サーバー環境変数を制御できない場合は、SqlNet パラメータを使用して ORACLE_SID を指定できます。また、サーバーが複数の Oracle Database インス タンスにアクセスする必要があるときにも、このパラメータを使用する必要があります。実

構文要素	説明:
	際にリモート・データベースにアクセスせずに Oracle Net 文字列を使用するには、パイプ・ドライバを使用します。たとえば、SqlNet=localsid1 と指定します。 localsid1 は、tnsnames.ora ファイルで定義されている別名です。
Loose_Coupling=true false	ロックが共有されるかどうかを示します。同じグローバル・トランザクション内の複数の Oracle Database トランザクション・ブランチには、密結合または疎結合のどちらの方法も使用できます。ブランチが疎結合の場合は、ロックを共有できません。疎結合ブランチの場合は、値を true に設定します。ブランチが密結合の場合は、ロックを共有できます。密結合ブランチの場合は、値を false に設定します。デフォルト値は false です。 ノート: Oracle RAC の実行時に、トランザクション・ブランチが別の Oracle RAC インスタンスにある場合は、Loose_Coupling=false であってもこれらは疎結合です。
SesWt=session_wait_limit	XA_RETRY が戻される前に Oracle Database が別のセッションに使用されているトランザクション・ブランチを待機する秒数を示します。デフォルト値は 60 秒です。
Threads=true false	アプリケーションがマルチスレッドかどうかを指定します。デフォルト値は false です。アプリケーションがマルチスレッドである場合、設定は true です。
FAN=true false	アプリケーションで高速アプリケーション通知(FAN)を使用するかどうかを指定します。デフォルト値は false です。アプリケーションで FAN を使用する場合、設定は true です。

関連項目:

FANの詳細は、[『Oracle Database管理者ガイド』](#)を参照してください。

22.3.4 プリコンパイラでのOracle XAの使用

カーソルは、Oracle XAアプリケーションで使用する場合はトランザクションの存続期間のみ有効です。明示的カーソルは、トランザクションが開始した後にオープンし、コミットまたはロールバックの前にクローズする必要があります。

プリコンパイラとのインタフェースでは、次の2つのオプションがあります。

- [デフォルトのデータベースでのプリコンパイラの使用](#)
- [指定されたデータベースでのプリコンパイラの使用](#)

この項の例では、プリコンパイラPro*C/C++を使用しています。

22.3.4.1 デフォルトのデータベースでのプリコンパイラの使用

デフォルトのデータベースでプリコンパイラにインタフェースするには、オープン文字列で使用するDB=db_nameフィールドが存

在しないことを確認してください。このフィールドが存在しない場合は、デフォルトの接続が指定されます。プロセスごとに1つのデフォルト接続のみが使用できます。

次に、デフォルトのPro*C/C++接続を識別するオープン文字列の例を示します。

```
ORACLE_XA+SqlNet=maildb+ACC=P/username/password
+SesTM=10+LogDir=/usr/local/logs
```

DB=db_nameが存在せず、空のデータベースID文字列であることを示しています。

SQL文の構文は次のようになります。

```
EXEC SQL UPDATE Emp_tab SET Sal = Sal*1.5;
```

22.3.4.2 指定されたデータベースでのプリコンパイラの使用

指定されたデータベースでプリコンパイラにインタフェースするには、オープン文字列にDB=db_nameフィールドを含めます。参照するすべてのデータベースは、対応するオープン文字列で指定した同一のdb_nameを参照する必要があります。

アプリケーションには、デフォルト・データベースの他に、名前を指定されたデータベースが1つ以上含まれることがあります。たとえば、あるデータベースで従業員の給与を更新し、別のデータベースでその従業員の部門番号(DEPTNO)を、第3のデータベースでその従業員の管理者を更新するとします。トランザクション・マネージャに[例22-2](#)のようなオープン文字列を構成します。

例22-2 オープン文字列構成のサンプル

```
ORACLE_XA+DB=MANAGERS+SqlNet=SID1+ACC=P/username/password
+SesTM=10+LogDir=/usr/local/xalog
ORACLE_XA+DB=PAYROLL+SqlNet=SID2+ACC=P/username/password
+SesTM=10+LogDir=/usr/local/xalog
ORACLE_XA+SqlNet=SID3+ACC=P/username/password
+SesTM=10+LogDir=/usr/local/xalog
```

[例22-2](#)の最後のオープン文字列にDB=db_nameフィールドはありません。

アプリケーション・サーバー・プログラムでは、次のような宣言を入力します。

```
EXEC SQL DECLARE PAYROLL DATABASE;
EXEC SQL DECLARE MANAGERS DATABASE;
```

ここでも、デフォルトの接続(DBフィールドを含まない3番目のオープン文字列に対応)には宣言は必要ありません。

更新を実行するときは、次に示すような文を入力します。

```
EXEC SQL AT PAYROLL UPDATE Emp_Tab SET Sal=4500 WHERE Empno=7788;
EXEC SQL AT MANAGERS UPDATE Emp_Tab SET Mgr=7566 WHERE Empno=7788;
EXEC SQL UPDATE Emp_Tab SET Deptno=30 WHERE Empno=7788;
```

最後の文は、デフォルトのデータベースを参照しているためAT句はありません。

Oracle Databaseプリコンパイラのリリース1.5.3以上では、次に示す例のように、AT句に文字ホスト変数を使用できます。

```
EXEC SQL BEGIN DECLARE SECTION;
  DB_NAME1 CHARACTER(10);
  DB_NAME2 CHARACTER(10);
EXEC SQL END DECLARE SECTION;

...
SET DB_NAME1 = 'PAYROLL'
SET DB_NAME2 = 'MANAGERS'
...
EXEC SQL AT :DB_NAME1 UPDATE...
EXEC SQL AT :DB_NAME2 UPDATE...
```

注意:



XA アプリケーションでは、xa_open を介して作成された以外の接続を作成しないでください。XA 以外の接続で実行されるすべての作業は、グローバル・トランザクションの範囲から外れることになり、個別にコミットする必要があります。

22.3.5 OCIでのOracle XAの使用

Oracle XAライブラリを使用するOracle Call Interfaceアプリケーションでは、リソース・マネージャにログインするためにOCISessionBeginをコールしないでください。ログインは、TPMを介して行うようにしてください。そのようなアプリケーションでは、関数xaoSvcCtxを実行して、リソース・マネージャへのアクセスに必要なサービス・コンテキスト構造体を取得できます。

環境ハンドルをOCI関数に渡す必要があるアプリケーションでは、そのハンドルを検索するためにxaoEnvもコールする必要があります。

アプリケーション・サーバーでは同時に複数のOracle Databaseのリソース・マネージャをオープンできるため、適切なサービス・コンテキストを取得するためには適切な引数を使用して関数xaoSvcCtxをコールする必要があります。

関連項目:

[Oracle Call Interfaceプログラマーズ・ガイド](#)

22.3.6 Oracle XAを使用したトランザクション制御の管理

XAライブラリを使用する場合、トランザクションは、トランザクションをコミットまたはロールバックするSQL文によっては制御されません。制御は、トランザクションを開始および終了するTMに受け入れられるAPIによって行われます。[表22-2](#)に示したXAライブラリ・サブプログラムではなく、[表22-6](#)に示したTXインタフェースなど、トランザクション・マネージャにより提供されるAPIをコールします。

TMは、通常、XAインタフェースを介してトランザクションを制御します。このインタフェースには、[表22-2](#)に示す関数が含まれています。

表22-6 TXインタフェース関数

TX関数	説明:
tx_open	リソース・マネージャにログインします。
tx_close	リソース・マネージャからログアウトします。
tx_begin	トランザクションを開始します。
tx_commit	トランザクションをコミットします。
tx_rollback	トランザクションをロールバックします。

ほとんどのTPMアプリケーションは、アプリケーション・クライアントがサービスを要求し、アプリケーション・サーバーがサービスを提供するというクライアント/サーバー・アーキテクチャを使用しています。「[プリコンパイラ・アプリケーションの例](#)」に示す例では、そのようなクライアント/サーバー・モデルを使用しています。サービスとは論理作業単位であり、Oracle DatabaseがRMである場合は、関連する作業単位を実行する一連のSQL文で構成されます。

たとえば、「貸方記入」というサービスが口座番号および貸方記入額を受け取ると、このサービスは、データベース内の特定の表にある情報を更新するSQL文を実行します。また、サービスはその他のサービスを要求することもあります。たとえば、「振替」サービスの場合は、「貸方記入」および「借方記入」にサービスを要求します。

通常、アプリケーション・クライアントは、トランザクション内で作業を実行するためにアプリケーション・サーバーにサービスを要求します。ただし、一部のTPMシステムでは、アプリケーション・クライアント自身がローカルにサービスを提供できます。「[プリコンパイラ・アプリケーションの例](#)」に示すとおり、トランザクション制御文は、クライアントまたはサーバーのどちらの側でも記述できます。

複数のプロセスを同一のトランザクションに参加させるために、TPMは参加プロセス間でトランザクション情報が伝達できる通信APIを提供しています。通信APIの例として、RPC、疑似RPC関数、送信/受信関数があります。

主要ベンダーが異なる通信関数をサポートしているため、次の例では、通信APIを汎用化するために通信疑似関数 `tpm_service` を使用しています。

X/Openの準備段階の仕様には、通信関数を提供する代替方法がいくつか取り入れられています。主要TPMベンダーでは、これらの代替方法を1つ以上はサポートしています。

22.3.7 プリコンパイラ・アプリケーションの例

次の例は、プリコンパイラ・アプリケーションを示しています。アプリケーション・サーバーがTPM固有の方法でRMシステムにログインしているとします。[例22-3](#)に、アプリケーション・サーバーによって開始されるトランザクションを示します。

例22-3 アプリケーション・サーバーによって開始されるトランザクション

```
/* ***** Client: ***** */
tpm_service("ServiceName");           /*Request Service*/
/* ***** Server: ***** */
ServiceName()
{
    <get service specific data>
    tx_begin();                        /* Begin transaction boundary */
    EXEC SQL UPDATE ...;
    /* This application server temporarily becomes */
    /* a client and requests another service. */
    tpm_service("AnotherService");
    tx_commit();                      /* Commit the transaction */
    <return service status back to the client>
}
```

[例22-4](#)に、アプリケーション・クライアントによって開始されるトランザクションを示します。

例22-4 アプリケーション・クライアントによって開始されるトランザクション

```
/* ***** Client: ***** */
tx_begin();                          /* Begin transaction boundary */
tpm_service("Service1");
tpm_service("Service2");
tx_commit();                          /* Commit the transaction */
/* ***** Server: ***** */
Service1()
{
    <get service specific data>
    EXEC SQL UPDATE ...;
    <return service status back to the client>
}
```

```

}
Service2()
{
  <get service specific data>
  EXEC SQL UPDATE ...;
  ...
  <return service status back to client>
}

```

22.3.8 プリコンパイラまたはOCIアプリケーションのTPMアプリケーションへの移行

既存のプリコンパイラまたはOCIアプリケーションを、Oracle XAライブラリを使用するTPMアプリケーションに移行するには、次の手順に従います。

1. アプリケーション・クライアントがアプリケーション・サーバーにサービスを要求できるように、アプリケーションを「サービス」のフレームワークに再編成します。TPMには、アプリケーションでtx_open関数およびtx_close関数を使用するように要求するTPMと、ログインおよびログオフを暗黙的に実行するTPMがあります。

オープン文字列にSqlNetパラメータを指定しないと、アプリケーションはデフォルトのOracle Netドライバを使用します。このため、ORACLE_HOMEおよびORACLE_SID環境変数を正しく定義して、アプリケーション・サーバーを立ち上げる必要があります。これは、TPMに固有の方法で行われます。これを行う方法の詳細は、TPMベンダーのマニュアルを参照してください。

2. アプリケーションで正規の接続文および切断文を置き換えます。たとえば、接続文EXEC SQL CONNECT(プリコンパイラの場合)またはOCISessionBegin、OCIServerAttachおよびOCIEnvCreate(OCIの場合)をtx_openで置き換えます。切断文EXEC SQL COMMIT/ROLLBACK WORK RELEASE(プリコンパイラの場合)またはOCISessionEnd/OCIServerDetach(OCIの場合)をtx_closeで置き換えます。
3. アプリケーションで、グローバル・トランザクションの正規のコミット文またはロールバック文を置き換え、トランザクションを明示的に開始します。

たとえば、tx_commit/tx_rollbackによってCOMMIT/ROLLBACK文のEXEC SQL COMMIT/ROLLBACK WORK(プリコンパイラの場合)またはOCITransCommit/OCITransRollback(OCIの場合)を置き換え、tx_beginをコールすることによってトランザクションを開始します。

ノート:



前述の例は、ローカル・トランザクションではなくグローバル・トランザクションにのみ該当します。ローカル・トランザクションは、Oracle API でコミットまたはロールバックする必要があります。

4. アプリケーションで、トランザクションを終了する前にフェッチ状態をリセットします。一般的には、release_cursor=noを使用します。release_cursor=yesは、文が1回しか実行されないことが確実なときにのみ使用します。

表22-7に、プリコンパイラまたはOCIアプリケーションをTPMアプリケーションに移行するときに、正規のOracle Database文を置き換えるTPM関数を示します。

表22-7 TPM用に置き換える文

正規のOracle Database文	TPM関数

正規のOracle Database文	TPM関数
CONNECTuser/password	tx_open(暗黙の可能性あり)
トランザクションの暗黙的な開始	tx_begin
SQL	SQL を実行するサービス
COMMIT	tx_commit
ROLLBACK	tx_rollback
切断	tx_close(暗黙の可能性あり)

22.3.9 Oracle XAライブラリ・スレッド・セーフティの管理

スレッドをサポートするトランザクション・モニターを使用する場合は、Oracle XAライブラリを使用してスレッド・セーフなアプリケーションを作成できます。ただし、注意が必要な問題がいくつかあります。

制御のスレッド(またはスレッド)とは、リソース・マネージャへの一連の接続のことです。スレッド化されていないシステムでは、各プロセスは制御のスレッドとみなされます。これは、各プロセスにはRMへの接続がそれぞれ独自にあり、それぞれが独立したRM表をメンテナンスしているためです。スレッド化されているシステムでは、各スレッドにはRMとの自律型接続があり、プライベートなRM表をメンテナンスします。このプライベートな表はスレッドのそれぞれに対して割り当てて必要があり、またスレッドが終了したときは(異常終了であっても)割り当てを解除する必要があります。

ノート:



Oracle Database では、データベースにアクセスする各スレッドに独自の接続が必要です。

トピック:

- [オープン文字列でのスレッドの指定](#)
- [Oracle XAのスレッドにおける制限事項](#)

22.3.9.1 オープン文字列でのスレッドの指定

xa_open文字列では、Threads=句が指定されます。TMでスレッドを使用できるようにするには、この句をtrueに指定する必要があります。デフォルトは、falseです。ほとんどの場合、スレッドはTMによって作成され、アプリケーションはスレッドがいつ作成されたのか認識しません。このため、サービス・コンテキストは、TMアプリケーション用に作成される各サービス内のスタック上に割り当ててをお勧めします。そのサービス内でOracle Database関連のコールを実行する前に、xaoSvcCtx関数をコールして初期化済のOCIサービス・コンテキストを取得する必要があります。このコンテキストをサービス内でOCIコールに使用できます。

22.3.9.2 Oracle XAのスレッドにおける制限事項

スレッドを使用するときは、次の制限事項が適用されます。

- アプリケーション・スレッドがそれぞれいつ開始されるかが明示的にトランザクション・モニターに通知されないかぎり、トランザクション・モニター上でアプリケーション・サーバー・プロセスの一部として実行されるPro*またはOCIコードをスレッド化することはできません。通常、これは、TMベンダーが提供する特別なCコンパイラを使用して実現されます。
- Pro*文EXEC SQL ALLOCATEおよびEXEC SQL USEはサポートされません。このため、スレッド化が使用可能であるときは、埋込みSQL文を非XA接続にまたがって使用することはできません。
- プロセスの1つのスレッドがXAを介してOracle Databaseに接続した場合、そのプロセスの他のすべてのスレッドもXAを介してOracle Databaseに接続する必要があります。1つのスレッドでEXEC SQL CONNECTを介して接続し、他のスレッドでxa_openを介して接続することはできません。

22.3.10 DBMS_XAパッケージの使用

PL/SQLアプリケーションは、DBMS_XAパッケージとともにOracle XAライブラリを使用できます。

[例22-5](#)では、1つ目のPL/SQLセッションがトランザクションを開始しますがコミットせず、2つ目のセッションがトランザクションを再開し、3つ目のセッションがコミットします。3つのセッションはすべてHRスキーマに接続されます。

例22-5 DBMS_XAパッケージの使用

```
REM Session 1 starts a transaction and does some work.
DECLARE
  rc PLS_INTEGER;
  oer PLS_INTEGER;
  xae EXCEPTION;
BEGIN
  rc := DBMS_XA.XA_START(DBMS_XA_XID(123), DBMS_XA.TMNOFLAGS);
  IF rc!=DBMS_XA.XA_OK THEN
    oer := DBMS_XA.XA_GETLASTOER();
    DBMS_OUTPUT.PUT_LINE('ORA-' || oer || ' occurred, XA_START failed');
    RAISE xae;
  ELSE DBMS_OUTPUT.PUT_LINE('XA_START(new xid=123)    OK');
  END IF;
  UPDATE employees SET salary=salary*1.1 WHERE employee_id = 100;
  rc := DBMS_XA.XA_END(DBMS_XA_XID(123), DBMS_XA.TMSUSPEND);
  IF rc!=DBMS_XA.XA_OK THEN
    oer := DBMS_XA.XA_GETLASTOER();
    DBMS_OUTPUT.PUT_LINE('ORA-' || oer || ' occurred, XA_END failed');
    RAISE xae;
  ELSE DBMS_OUTPUT.PUT_LINE('XA_END(suspend xid=123)  OK');
  END IF;
  EXCEPTION
    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE
        ('XA error('||rc||') occurred, rolling back the transaction ...');
      rc := DBMS_XA.XA_END(DBMS_XA_XID(123), DBMS_XA.TMSUCCESS);
      rc := DBMS_XA.XA_ROLLBACK(DBMS_XA_XID(123));
      IF rc != DBMS_XA.XA_OK THEN
        oer := DBMS_XA.XA_GETLASTOER();
        DBMS_OUTPUT.PUT_LINE('XA-'||rc||', ORA-' || oer ||
          ' XA_ROLLBACK does not return XA_OK');
        raise_application_error(-20001, 'ORA-'||oer||
          ' error in rolling back a failed transaction');
      END IF;
      raise_application_error(-20002, 'ORA-'||oer||
        ' error in transaction processing, transaction rolled back');
END;
/
SHOW ERRORS
DISCONNECT
REM Session 2 resumes the transaction and does some work.
DECLARE
```



```

rc PLS_INTEGER;
oer PLS_INTEGER;
s NUMBER;
xae EXCEPTION;
BEGIN
rc := DBMS_XA.XA_START(DBMS_XA_XID(123), DBMS_XA.TMRESUME);
IF rc!=DBMS_XA.XA_OK THEN
oer := DBMS_XA.XA_GETLASTOER();
DBMS_OUTPUT.PUT_LINE('ORA-' || oer || ' occurred, xa_start failed');
RAISE xae;
ELSE DBMS_OUTPUT.PUT_LINE('XA_START(resume xid=123) OK');
END IF;
SELECT salary INTO s FROM employees WHERE employee_id = 100;
DBMS_OUTPUT.PUT_LINE('employee_id = 100, salary = ' || s);
rc := DBMS_XA.XA_END(DBMS_XA_XID(123), DBMS_XA.TMSUCCESS);
IF rc!=DBMS_XA.XA_OK THEN
oer := DBMS_XA.XA_GETLASTOER();
DBMS_OUTPUT.PUT_LINE('ORA-' || oer || ' occurred, XA_END failed');
RAISE xae;
ELSE DBMS_OUTPUT.PUT_LINE('XA_END(detach xid=123) OK');
END IF;
EXCEPTION
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE
('XA error('||rc||') occurred, rolling back the transaction ...');
rc := DBMS_XA.XA_END(DBMS_XA_XID(123), DBMS_XA.TMSUCCESS);
rc := DBMS_XA.XA_ROLLBACK(DBMS_XA_XID(123));
IF rc != DBMS_XA.XA_OK THEN
oer := DBMS_XA.XA_GETLASTOER();
DBMS_OUTPUT.PUT_LINE('XA-'||rc||', ORA-' || oer ||
' XA_ROLLBACK does not return XA_OK');
raise_application_error(-20001, 'ORA-'||oer||
' error in rolling back a failed transaction');
END IF;
raise_application_error(-20002, 'ORA-'||oer||
' error in transaction processing, transaction rolled back');
END;
/
SHOW ERRORS
DISCONNECT
REM Session 3 commits the transaction.
DECLARE
rc PLS_INTEGER;
oer PLS_INTEGER;
xae EXCEPTION;
BEGIN
rc := DBMS_XA.XA_COMMIT(DBMS_XA_XID(123), TRUE);
IF rc!=DBMS_XA.XA_OK THEN
oer := DBMS_XA.XA_GETLASTOER();
DBMS_OUTPUT.PUT_LINE('ORA-' || oer || ' occurred, XA_COMMIT failed');
RAISE xae;
ELSE DBMS_OUTPUT.PUT_LINE('XA_COMMIT(commit xid=123) OK');
END IF;
EXCEPTION
WHEN xae THEN
DBMS_OUTPUT.PUT_LINE
('XA error('||rc||') occurred, rolling back the transaction ...');
rc := DBMS_XA.XA_ROLLBACK(DBMS_XA_XID(123));
IF rc != DBMS_XA.XA_OK THEN
oer := DBMS_XA.XA_GETLASTOER();
DBMS_OUTPUT.PUT_LINE('XA-'||rc||', ORA-' || oer ||
' XA_ROLLBACK does not return XA_OK');
raise_application_error(-20001, 'ORA-'||oer||
' error in rolling back a failed transaction');
END IF;
raise_application_error(-20002, 'ORA-'||oer||
' error in transaction processing, transaction rolled back');

```

```
END;  
/  
SHOW ERRORS  
DISCONNECT  
QUIT
```

関連項目:

DBMS_XAパッケージの詳細は、『[Oracle Database PL/SQLパッケージおよびタイプ・リファレンス](#)』を参照してください

22.4 XAアプリケーションのトラブルシューティング

トピック:

- [Oracle XAトレース・ファイルへのアクセス](#)
- [インダウトまたはペンディングのOracle XAトランザクションの管理](#)
- [SYSアカウント表を使用したOracle XAトランザクションの監視](#)

22.4.1 Oracle XAトレース・ファイルへのアクセス

Oracle XAライブラリでは、すべてのエラーおよびトレース情報がトレース・ファイルに記録されます。この情報は、XAエラー・コードを補足するときに役立ちます。たとえばxa_open障害が発生したとき、その原因として、オープン文字列が正しくなかったか、Oracle Databaseインスタンスの検索が正常に実行されなかったか、またはログイン認可が正常に実行されなかったかを示します。

トレース・ファイル名はxa_db_namdate.trcです。db_nameはオープン文字列のフィールドDB=db_nameで指定したデータベース名で、dateは情報がトレース・ファイルに記録された日付です。オープン文字列でDB=db_nameを指定しない場合、自動的にデフォルトでNULLになります。

たとえば、xa_NULL06022005.trcは、2005年6月2日に作成されたトレース・ファイルを示しています。リソース・マネージャがオープンされたときに、DBフィールドがオープン文字列に指定されていませんでした。ファイル名

xa_Finance12152004.trcは、2004年12月15日に作成されたトレース・ファイルを示しています。リソース・マネージャがオープンされたときに、DBフィールドはオープン文字列で「Finance」と指定されていました。

ノート:



オープン文字列に同一の DB フィールドおよび LogDir フィールドを持つ複数の Oracle XA ライブラリ RM は、同じ日に発生したすべてのトレース情報を同じトレース・ファイルに記録します。

トレース・ファイルに次の内容が含まれているとします。

```
1032.12345.2: ORA-01017: invalid username/password; logon denied  
1032.12345.2: xalgn: XAER_INVAL; logon denied
```

[表22-8](#)に、各要素の意味を示します。

表22-8 サンプル・トレース・ファイルの内容

文字列	説明:

文字列	説明:
1032	情報が記録された時刻。
12345	プロセス ID(PID)。
2	リソース・マネージャ ID
xaolgn	モジュール名
XAER_INVALID	XA 標準での指定に従って戻されたエラー
ORA-01017	戻された Oracle Database 情報

トピック:

- [xa_open文字列DbgFl](#)
- [トレース・ファイルの位置](#)

22.4.1.1 xa_open文字列DbgFl

通常、XAトレース・ファイルはエラーが検出された場合にのみオープンされます。xa_open文字列DbgFlは、XAライブラリに関する詳細を記録するトレース機能を提供します。デフォルト値は0(ゼロ)です。次の組合せのいずれかに設定できます。

- 0x1: XAインタフェース内の各サブプログラムの入口および出口をトレースできます。この値は、TPモニターがどのXAコールを作成し、どのトランザクション識別子を生成しているかを正確に調べるときに役立ちます。
- 0x2: 他の非公開XAライブラリ・プログラムへの入口および出口をトレースできます。通常、これはOracle Database開発者用の機能です。
- 0x4: OCIに対する専用コールのように、XAライブラリが作成する他の様々な関心のあるコールをトレースできます。通常、これはOracle Database開発者用の機能です。

ノート:



各フラグは ub4 の独立ビットであるため、複数のフラグから出力を取得するには、フラグの値の組合せを設定する必要があります。

22.4.1.2 トレース・ファイルの位置

XAアプリケーションでは、トレース・ファイルの位置が次のアルゴリズムに従って判別されます。

1. オープン文字列で指定されているLogDirディレクトリ。
2. オープン文字列にLogDirを指定しないと、Oracle XAアプリケーションは(Oracleホームがアクセス可能な場合は)次のディレクトリにトレース・ファイルを作成しようとします。

- %ORACLE_HOME%\rdbms\trace(Windowsの場合)

- \$ORACLE_HOME/rdbms/log(LinuxおよびUNIXの場合)

3. Oracle XAアプリケーションでは、Oracleホームの位置を判別できない場合、トレース・ファイルは現在の作業ディレクトリに作成されます。

22.4.2 インダウトまたはペンディングのOracle XAトランザクションの管理

インダウト・トランザクションまたはペンディング・トランザクションとは、準備はできているがデータベースに対してコミットされていないトランザクションです。一般的に、TPMシステムが提供するTMは、インダウト・トランザクションまたはペンディング・トランザクションの障害を解決し、リカバリします。次の状況では、DBAがインダウト・トランザクションをオーバーライドする必要がある場合があります。

- 他のトランザクションが必要とするデータをロックしている場合。
- 適切な時間の経過後も解決されない場合。

前述のような状況でインダウト・トランザクションをオーバーライドする方法の詳細、およびインダウト・トランザクションをコミットまたはロールバックするかどうかを決定する方法の詳細は、TPMのマニュアルを参照してください。

22.4.3 SYSアカウント表を使用したOracle XAトランザクションの監視

Oracle DatabaseのSYSアカウントには次のビューがあり、正規のOracle DatabaseアプリケーションおよびOracle XAアプリケーションによって生成されたトランザクションが含まれています。

- DBA_PENDING_TRANSACTIONS
- V\$GLOBAL_TRANSACTION
- DBA_2PC_PENDING
- DBA_2PC_NEIGHBORS

Oracle XAアプリケーションによって生成されたトランザクションの場合は、次に示す列情報がDBA_2PC_NEIGHBORS表に適用されます。

- DBID列は常にxa_orcl
- DBUSER_OWNER列は常にdb_namexa.oracle.com

db_nameは、オープン文字列で常にDB=db_nameと指定されることに注意してください。オープン文字列にこのフィールドを指定しないと、この列の値はOracle XAアプリケーションが生成するトランザクションについてはNULLxa.oracle.comになります。

たとえば、次のSQL文を使用すると、Oracle XAアプリケーションによって生成されたインダウト・トランザクションの詳細情報を取得できます。

```
SELECT *
FROM DBA_2PC_PENDING p, DBA_2PC_NEIGHBORS n
WHERE p.LOCAL_TRAN_ID = n.LOCAL_TRAN_ID
AND n.DBID = 'xa_orcl';
```

別の方法として、トランザクション処理モニターが使用するフォーマットIDがわかっている場合は、

DBA_PENDING_TRANSACTIONSまたはV\$GLOBAL_TRANSACTIONを使用できます。

DBA_PENDING_TRANSACTIONSでは準備済トランザクションの両方のリストが提供されますが、

V\$GLOBAL_TRANSACTIONでは、すべてのアクティブなグローバル・トランザクションのリストが提供されます。

22.5 Oracle XAの問題および制限事項

トピック:

- [Oracle XAアプリケーションでのデータベース・リンクの使用](#)
- [Oracle XAアプリケーションでのトランザクション・ブランチの管理](#)
- [Oracle Real Application Clusters\(Oracle RAC\)でのOracle XAの使用](#)
- [SQLに基づくOracle XAの制限事項](#)
- [その他の制限事項](#)

22.5.1 Oracle XAアプリケーションでのデータベース・リンクの使用

Oracle XAアプリケーションは、次の制限事項に違反しないかぎり、データベース・リンク経由で他のOracle Databaseインスタンスにアクセスできます。

- 共有サーバー構成を使用する必要があります。

トランザクション処理モニター(TPM)は、共有サーバーを使用してOracle Database Aへの接続をオープンします。データベース・リンクに必要なオペレーティング・システム・ネットワーク接続は、専用サーバー・プロセスではなくディスパッチャによってオープンされます。このため、様々なサービスやスレッドでトランザクションを操作できます。

この制限事項が満たされていない場合、XAトランザクション内でデータベース・リンクを使用すると、専用サーバー・プロセスと他方のOracle Database Bの間にオペレーティング・システム・ネットワーク接続が作成されます。このネットワーク接続は専用サーバー・プロセス間で移動できないため、データベースAのこの専用サーバー・プロセスから連結解除することはできません。データベース・リンクを介してデータベースBにアクセスするときに、ORA-24777エラーが戻されます。

- アクセスする他のデータベースは、別のOracle Databaseである必要があります。

この制限事項が満たされている場合、Oracle Databaseはデータベース・リンクを許可し、トランザクション・プロトコルを他のOracle Databaseインスタンスに伝播(準備、ロールバックおよびコミット)します。

共有サーバー構成を使用できない場合は、リモート・データベースにEXEC SQL AT構文を使用してPro*C/C++アプリケーション経由でアクセスします。

init.oraパラメータOPEN_LINKS_PER_INSTANCEは、移行可能なオープン・データベース・リンク接続の数を指定します。これらのdblink接続は、トランザクションのコミット後に接続をキャッシュできるように、XAトランザクションによって使用されます。接続を作成したユーザーがトランザクションも作成した場合、別のトランザクションがデータ・リンク接続を使用できます。このパラメータは、1つのセッションにおけるリモート・データベースへの同時オープン接続の最大数(データベース・リンクを含む)を指定するinit.oraパラメータOPEN_LINKSとは異なります。OPEN_LINKSパラメータは、XAアプリケーションには適用できません。

22.5.2 Oracle XAアプリケーションでのトランザクション・ブランチの管理

同じグローバル・トランザクション内の複数のOracle Databaseトランザクション・ブランチには、密結合または疎結合のどちらの方法も使用できます。トランザクション・ブランチが**密結合**の場合は、ロックを共有できます。そのため、あるトランザクション・ブランチでのCOMMIT前更新は、同じグローバル・トランザクションに属する他のブランチで参照可能です。疎結合トランザクション・ブランチの場合、ブランチはロックを共有せず、他のブランチでの更新を認識しません。

密結合ブランチの場合、Oracle Databaseは文を実行する前にDXロックを取得します。システムが文の実行前にロックを取得しないため、疎結合トランザクション・ブランチの同時実行性が増します。デメリットは、すべてのトランザクション・ブランチが2

フェーズ・コミットを実行する必要があることです。XAの1フェーズ最適化は使用できません。

[表22-9](#)に、密結合ブランチと疎結合ブランチのトレードオフを示します。

表22-9 密結合および疎結合トランザクション・ブランチ

属性	密結合ブランチ	疎結合ブランチ
2 フェーズ・コミット	読取り専用最適化 [全ブランチについて準備、最後のブランチについてコミット]	2 フェーズ [全ブランチについて準備およびコミット]
シリアライズ	データベース・コール	なし

22.5.3 Oracle Real Application Clusters(Oracle RAC)でのOracle XAの使用

Oracle Database 11gリリース1(11.1)では、XAトランザクションは複数のOracle RACインスタンスにまたがって、XAを使用するすべてのアプリケーションでOracle RAC環境を活用し、アプリケーションの可用性とスケーラビリティを拡張できるようになりました。

ノート:



分散トランザクションと組み合わせた外部プロシージャ・コールアウトはサポートされません。

トピック:

- [GLOBAL_TXN_PROCESSES初期化パラメータ](#)
- [Oracle RACでのトランザクション・ブランチの管理](#)
- [Oracle RACでのDTPサービス\(10.2\)を使用したインスタンス・リカバリの管理](#)
- [Oracle RACのXIDのグローバル一意性](#)
- [密結合と疎結合](#)

22.5.3.1 GLOBAL_TXN_PROCESSES初期化パラメータ

初期化パラメータGLOBAL_TXN_PROCESSESは、各Oracle RACインスタンスのGTXnバックグラウンド・プロセス数を指定します。デフォルト値は1です。

分散トランザクションが複数のOracle RACインスタンスにまたがる場合、このパラメータはクラスタ全体でデフォルト値のままとなります。これにより、Oracle RACインスタンス全体にわたって実行される作業単位は、リソースを共有し、単一のトランザクションとして機能します(つまり、作業単位は密結合となります)。また、クラスタ内の任意のノードへの2フェーズ・コミット要求の送信も可能となります。

関連項目:

22.5.3.2 Oracle RACでのトランザクション・ブランチの管理

ノート:

このトピックは、次のいずれかが当てはまる場合に適用されます。



- 初期化パラメータ GLOBAL_TXN_PROCESSES はすべての Oracle RAC インスタンスの初期化ファイルのデフォルト値でない。
- Oracle XA アプリケーションが以前に連結解除されたトランザクションのブランチを再開または結合する。

Oracle Databaseでは、様々なインスタンスがOracle RACの異なるトランザクション・ブランチで操作できます。たとえば、ノード1がブランチAで操作し、ノード2がブランチBで操作できます。Oracle Database 11gリリース1(11.1)より前は、トランザクション・ブランチが異なるインスタンスにある場合は疎結合であり、ロックを共有しませんでした。この場合、Oracle Databaseでは、様々なアプリケーション・スレッドの異なる作業単位が、リソースを共有しない個別エンティティとして処理されました。

また、複数のインスタンスが1つのトランザクション・ブランチで操作する場合もあります。たとえば、次のように1つのトランザクションがノード1およびノード2にあるとします。

ノード1

1. xa_start
2. SQL操作
3. xa_end (SUSPEND)

ノード2

1. xa_start (RESUME)
2. xa_prepare
3. xa_commit
4. xa_end

直前の順序では、ノード2は物理的に異なるノード(ノード1)にあるブランチを再開しないため、Oracle Databaseではエラーが戻されます。

Oracle Database 11gリリース1 (11.1)より前のリリースでは、Oracle RACで密結合を実現する方法として**分散トランザクション処理(DTP)サービス**、つまりカーディナリティ(1)によって、ロード・バランシングが有効かどうかに関係なくすべての密結合ブランチが確実に同じインスタンスに配置されるサービスを使用していました。中間層コンポーネントは、常に1つのOracle RAC インスタンスにマップする共通の論理データベース・サービス名を使用して、Oracle Databaseをアドレス指定しました。データベース・インスタンスの物理特性は、データベース・サービスの中間名リゾルバによって隠されました。DTPサービスを使用すると、密結合グローバル・トランザクションの参加者はすべて1つのインスタンス上でブランチを作成できました。

Oracle Database 11gリリース1 (11.1)では、密結合ブランチでのXAトランザクションのサポートにDTPサービスは不要になりました。デフォルトでは、異なるOracle RACインスタンスにある密結合ブランチは密結合のままとなります。つまり、Oracle RACインスタンス全体でロックとリソースを共有します。

たとえば、DTPサービスを使用すると、次の一連のアクションが同じインスタンス上で発生します。

1. xa_start
2. SQL操作
3. xa_end (SUSPEND)
4. xa_start (RESUME)
5. SQL操作
6. xa_prepare
7. xa_commitまたはxa_rollback

さらに、それぞれが同じDTPサービスでOracle RACをアドレス指定する場合は、複数の密結合ブランチが同じインスタンスにあります。

クラスタのインスタンスをすべて活用するには、分散トランザクションを管理する各ノードに1つ以上、複数のDTPサービスを作成します。グローバル分散トランザクションのブランチは、すべて同じインスタンスに存在します。そのため、Oracle RACクラスタのすべてのインスタンスおよびノードを利用して多数の分散XAトランザクションのロード・バランシングを実行し、アプリケーションのスループットを最大化できます。

関連項目:

Real Application Clusters構成で分散トランザクションを管理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。

22.5.3.3 Oracle RACでのDTPサービス(10.2)を使用したインスタンス・リカバリの管理

Oracle Database 10gリリース2(10.2)より前は、Oracle RACでの障害検出とフェイルオーバーおよびフェイルバックのトリガーはTMで実行されていました。インダウト・トランザクションに関する情報をDBA_2PC_PENDINGに確実に伝播させるために、TMではインダウト・トランザクションの解決に進む前にxa_recoverを起動する必要がありました。インスタンスに障害が発生した場合、XAクライアント・ライブラリはSYS.DBMS_XA.DIST_TXN_SYNCプロシージャを実行して障害を起こしたインスタンスのUNDOセグメントがリカバリされたことを確認するまで、他のインスタンスにフェイルオーバーできませんでした。Oracle Database 10gリリース2(10.2)では、処理中のトランザクションに関する十分な情報がTMにあれば、このようにxa_recoverを起動するという要件はありません。

ノート:



Oracle Database 9g リリース 2(9.2)以降のリリースでは、xa_recover はリモート・サイトで分散データ操作言語(DML)が完了するまで待機する必要があります。

Oracle RACでDTPサービスを使用すると、次のようなメリットがあります。

- インスタンス障害の検出が自動化されます。
- インスタンスのフェイルオーバーとフェイルバックが自動化されます。インスタンスに障害が発生すると、このインスタンスで管理されているDTPサービスが他のインスタンスにフェイルオーバーします。このフェイルオーバーにより、クライアントは強制的に再接続されますが、サービスの論理名は変わりません。フェイルオーバーは自動的で、管理者が介入する必要はありません。管理者は、サービス再配置文を使用してフェイルバックを実行できますが、フェイルバック関連のリカバリはすべてデータベース・サーバー内で自動的に処理されます。
- クライアントではなくOracle Databaseがインスタンスのリカバリを実行できます。データベースでは、中間層TMの関与

を必要とせずに、他のインスタンスによって準備されたトランザクションの状態を判別できます。

関連項目:

- インスタンス・リカバリ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 RACでのサービスおよび分散トランザクション処理の詳細は、[『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。

22.5.3.4 Oracle RACのXIDのグローバル一意性

Oracle Database 11gリリース1(11.1)より前のリリースでは、Oracle RACデータベースでは指定されたXIDがクラスタ内のXAトランザクションで一意であるかどうかは判断できません。

たとえば、Oracle RACインスタンス1にXID Fmt(x).Tx(1).Br(1)、Oracle RACインスタンス2に別のXID Fmt(x).Tx(1).Br(1)があるとします。XIDがOracle RACインスタンス間で一意ではありませんが、どちらもブランチを開始してSQLを実行できます。

Oracle Database 11gリリース1 (11.1)では、Oracle RACデータベースはOracle RACインスタンス間の重複するXIDを検出し、重複したXIDを持つブランチが開始しないようにしています。

関連項目:

Oracle RACでのサービスおよび分散トランザクション処理の詳細は、[『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。

22.5.3.5 密結合と疎結合

同じグローバル・トランザクション内の複数のOracle Databaseトランザクション・ブランチには、密結合または疎結合のどちらかを使用できます。通常、結合のタイプはxa_open文字列のLoose_Couplingフィールドで定義します(詳細は、[表22-5](#)を参照してください)。ただし、Oracle RACの実行時に、トランザクション・ブランチが別のOracle RACインスタンスにある場合は、Loose_Coupling=falseであってもこれらは疎結合です。

関連項目:

- Oracle RACでのサービスおよび分散トランザクション処理の詳細は、[『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。
- [Oracle XAアプリケーションでのトランザクション・ブランチの管理](#)

22.5.4 SQLに基づくOracle XAの制限事項

この項では、次のSQL操作に関する制限事項について説明します。

- [ロールバックおよびコミット](#)
- [DDL文](#)
- [セッション状態](#)

- [EXEC SQL](#)

22.5.4.1 ロールバックおよびコミット

グローバル・トランザクションの進行状況の調整および監視はトランザクション・マネージャが行うため、グローバル・トランザクションのロールバックまたはコミットを独立して行うOracle Database固有の文をアプリケーションに入れないでください。ただし、ローカル・トランザクションではロールバックおよびコミットを使用できます。

グローバル・トランザクションの途中で、プリコンパイラ・アプリケーションにEXEC SQL ROLLBACK WORKを使用しないでください。同様に、OCIアプリケーションでは、OCITransRollbackまたはバージョン7で同等のorolを実行しないでください。グローバル・トランザクションをロールバックするには、tx_rollbackをコールします。

同様に、グローバル・トランザクションの途中で、プリコンパイラ・アプリケーションにEXEC SQL COMMIT WORK文を使用しないでください。OCIアプリケーションでは、OCITransCommitまたはバージョン7で同等のocomを実行しないでください。たとえば、tx_commitまたはtx_rollbackを使用して、グローバル・トランザクションを終了してください。

22.5.4.2 DDL文

CREATE TABLEなどのデータ定義言語(DDL)文は、暗黙的なコミットを意味するためOracle XAアプリケーションでは実行できません。

22.5.4.3 セッション状態

Oracle Databaseでは、セッション状態が複数のTPMサービス間で有効であることを保証しません。たとえば、あるTPMサービスがセッション変数(グローバル・パッケージ変数など)を更新した場合、同じグローバル・トランザクションの一部として実行している別のTPMサービスにはこの変更が認識されないことがあります。セーブポイントは1つのTPMサービス内で使用してください。アプリケーションが、別のTPMサービスで作成されたセーブポイントを参照しないようにしてください。同様に、アプリケーションが、別のTPMサービスで実行されたカーソルからフェッチしないようにしてください。

22.5.4.4 EXEC SQL

接続および切断にEXEC SQL文を使用しないでください。つまり、EXEC SQL CONNECT、EXEC SQL COMMIT WORK RELEASEまたはEXEC SQL ROLLBACK WORK RELEASEを使用しないでください。

22.5.5 その他の制限事項

- Oracle XAとゲートウェイの両方を同じセッションで使用できません。
- Oracle Databaseでは、関連付けの移行はサポートされません(関連付けの移行とは、トランザクション・マネージャが中断中のブランチ関連付けを別のブランチで再開する手段です)。
- XAのオプションの機能である非同期XAコールはサポートされません。
- TRANSACTIONS初期化パラメータを、グローバル・トランザクションの同時実行予測数に設定します。初期化パラメータOPEN_LINKS_PER_INSTANCEは、移行可能なオープン・データベース・リンク接続の数を指定します。これらのデータベース・リンク接続は、トランザクションのコミット後に接続をキャッシュできるように、XAトランザクションによって使用されます。

関連項目:

[Oracle XAアプリケーションでのデータベース・リンクの使用](#)

- スレッドごとのxa_openコールの最大数は32です。
- TPモニター・ベースのXAアプリケーションを構築する場合、リンク・ラインで、TPモニター・ライブラリ(ax_regおよびax_unreg記号を定義する)をOracle Databaseクライアント共有ライブラリの前に指定する必要があります。プラットフォームで共有ライブラリがサポートされていない場合、またはリンカーがリンク・ライン内のライブラリの順序を区別しない場合は、Oracle Databaseの非共有クライアント・ライブラリを使用します。これらのリンク制限は、XAの動的登録(Oracle XAスイッチxaoswd)を使用する場合のみ適用できます。

23 パブリッシュ・サブスクライブ・モデルでのアプリケーションの開発

この章では、パブリッシュ・サブスクライブ・モデルでアプリケーションを開発する方法について説明します。

トピック:

- [パブリッシュ・サブスクライブ・モデルの概要](#)
- [パブリッシュ・サブスクライブのアーキテクチャ](#)
- [パブリッシュ・サブスクライブの概念](#)
- [パブリッシュ・サブスクライブ・メカニズムの例](#)

23.1 パブリッシュ・サブスクライブ・モデルの概要

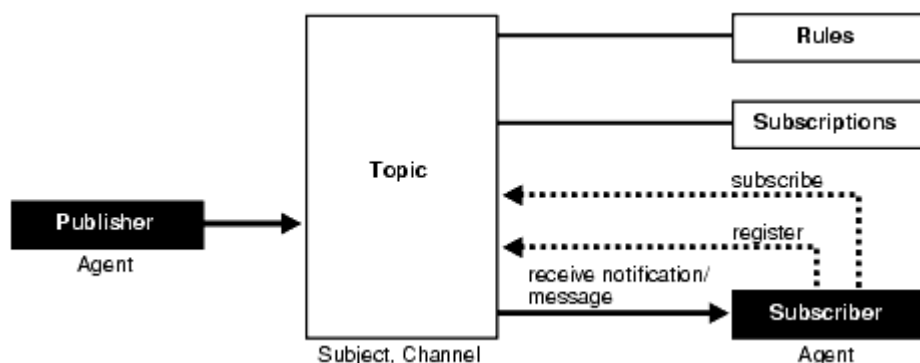
データベースは企業内の最も重要な情報資源であるため、この役割を補完するために、Oracleでは、企業の情報配信およびメッセージ交換用にパブリッシュ・サブスクライブ・ソリューションを作成しました。

ネットワーキング・テクノロジーおよびネットワーキング製品によって、多数のコンピュータ、アプリケーションおよびユーザーにまたがる高度な接続が可能です。これらの環境では、疎結合で自律的に動作する分散システムに非同期通信を提供することが重要であり、ネットワーク障害に強い操作性が要求されます。この要件は、メッセージ機能、メッセージ指向ミドルウェア(MOM)、メッセージ・キューイング、パブリッシュ・サブスクライブなどの特長を持つ様々なミドルウェア製品によって満たされています。

パブリッシュ・サブスクライブ・パラダイムを介して通信するアプリケーションには、受信者を明示的に指定したり意図された受信者を知らせなくても、メッセージを発行できる送信アプリケーション(パブリッシャ)が必要です。同様に、受信アプリケーション(サブスクライバ)は、サブスクライバが登録しているメッセージのみを受信する必要があります。

この送信者と受信者の切離しは、通常、パブリッシャとサブスクライバの間に存在して間接的なレベルとして機能するエンティティによって実現されます。このエンティティが、サブジェクトまたはチャネルを表すキューです。[図23-1](#)に、パブリッシュおよびサブスクライブの機能を示します。

図23-1 Oracleのパブリッシュ・サブスクライブの機能



サブスクライバは、あるキューにエンキューされたメッセージに関心を示すことによって、また、サブジェクト・ベースまたはコンテンツ・ベースのルールをフィルタとして使用することによって、キューをサブスクライブします。この結果、指定されたキューに一連のルールベースのサブスクリプションが対応付けられます。

実行時、パブリッシャは様々なキューにメッセージを転送します。次に、キュー(基礎となるインフラストラクチャの配信メカニズム)は、様々なサブスクリプションに一致したメッセージを該当するサブスクライバに配信します。

23.2 パブリッシュ・サブスクライブのアーキテクチャ

Oracle Databaseには、データベースに対応したパブリッシュ・サブスクライブのメッセージ機能をサポートする機能が含まれています。

- [データベース・イベント](#)
- [Oracle Advanced Queuing](#)
- [クライアント通知](#)

23.2.1 データベース・イベント

データベース・イベントは、データベース・イベントを公開するための宣言定義、検出およびこれらのイベントの実行時の発行をサポートします。この機能によって、イベント駆動方式でエンドユーザーに情報を能動的に発行し、従来のプル型の情報アクセスの方法を補足することができます。

関連項目:

[Oracle Database PL/SQL言語リファレンス](#)

23.2.2 Oracle Advanced Queuing

Oracle Advanced Queuing(AQ)では、キュー・ベースのパブリッシュ・サブスクライブ・パラダイムがサポートされます。データベース・キューは、メッセージの永続的な格納場所として機能し、キューを基にしたパブリッシュおよびサブスクライブが可能になります。ルール・エンジンおよびサブスクリプション・サービスが、設定された関心事項に基づいて、受信者にメッセージを動的に送付します。これによって、送信者と受信者の間のアドレス関係が切り離され、送信者と受信者間での明示的なメッセージ・アドレスングを補足することができます。

関連項目:

[Oracle Databaseアドバンスド・キューイング・ユーザーズ・ガイド](#)

23.2.3 クライアント通知

クライアント通知は、メッセージに関心を持つサブスクライバに対するメッセージの非同期配信をサポートします。これによって、データベース・クライアントは特定のキューに対する関心を登録し、そのようなキューで発行が発生した場合に通知を受け取ることができます。データベース・クライアントへのメッセージの非同期配信は、情報を取得するために使用される従来のポーリングとは対照的な技法です。

関連項目:

[Oracle Call Interfaceプログラマーズ・ガイド](#)

23.3 パブリッシュ・サブスクライブの概念

キュー

キューは、名前付きの関心対象サブジェクトをサポートするエンティティです。キューは永続または非永続(軽量)の特性を持ちます。

永続キューは、メッセージの永続的なコンテナとして機能します。メッセージは、遅延付き高信頼モードで配信されます。

非永続(軽量)キューの基礎となるインフラストラクチャでは、公開されたメッセージを接続クライアントに対して最高で1回送信します。

エージェント

パブリッシャおよびサブスクライバは、内部的にはエージェントとして表されます。

エージェントは、サブスクリプションを介してキューに関心を示す持続的で論理的なサブスクライブ・エンティティです。エージェントには、関連するサブスクリプション、アドレス、メッセージの配信モードなどのプロパティがあります。この意味では、エージェントはパブリッシャまたはサブスクライバの電子的なプロキシです。

クライアント

クライアントは、一時的な物理エンティティです。クライアントの属性には、クライアント・プログラムが実行される物理プロセス、ノード名、クライアント・アプリケーション・ロジックが含まれます。単一のエージェントにかわって複数のクライアントが動作する場合もあります。認証されている場合は、同一のクライアントが複数のエージェントにかわって動作できます。

キューでのルール

キューでのルールは、メッセージ形式属性またはメッセージ・ヘッダー属性に関する一連の事前定義済演算子を使用する条件式として指定されます。各キューには、そのキューが示すメッセージの構造を記述したメッセージ内容形式が対応付けられています。メッセージ形式は、構造化されていなくても(RAW)、正しく定義された構造体(ADT)であってもかまいません。これによって、サブジェクト・ベースおよびコンテンツ・ベースの両方のサブスクリプションが可能です。

サブスクライバ

サブスクライバ(エージェント)は、ルールを使用してキューにサブスクリプションを指定できます。サブスクライバは永続的であり、カタログに格納されます。

データベース・イベント発行フレームワーク

データベースは、重要な情報公開元を表します。イベント・フレームワークは、データベース・イベント発行の宣言定義ができるように提案されたものです。これらの事前定義済イベントが発生すると、このフレームワークがイベントを検出および公開します。これによって、パブリッシュ・サブスクライブ機能の一部として、エンドユーザーに対してイベント駆動方式によるアクティブな情報配信が可能になります。

登録

登録は、エージェントにかわって動作する所定のクライアントによって対応付けられた配信情報のプロセスです。エージェントとクライアントの区別に関連して、サブスクリプションと登録の間には重要な区別があります。

サブスクリプションは、エージェントによる特定のキューへの関心を示します。配信の場所および方法は指定しません。配信情報は、クライアントに対応付けられた物理的なプロパティで、論理エージェント(サブスクライバ)の一時的な発現です。エージェントにかわって動作する特定のクライアント・プロセスは、配信を行う 場所を示すホストとポート、および配信の方法を示すコールバックを対応付けることによって、配信情報を登録します。

メッセージの公開

パブリッシャは、適切なキューイング・インタフェースを使用して、キューにメッセージを公開します。インタフェースは、キューが実装されているモデルに依存することがあります。たとえば、エンキュー・コールは、メッセージの公開を表します。

ルール・エンジン

指定されたキューにメッセージが転送または公開されると、ルール・エンジンはそのキューに関して定義されたすべてのルールから、公開されたメッセージと一致する一連の候補となるルールを取得します。

サブスクリプション・サービス

指定されたキュー上の候補ルールのリストに応じて、候補ルールに一致する一連のサブスクライバを評価できます。次に、このサブスクリプション・リストに対応する一連のエージェントが決定され、通知されます。

ポスト

キューは、登録されたすべてのクライアントに対して、公開された該当メッセージを通知します。この概念は**ポスト**と呼ばれます。関心があるすべてのクライアントに通知が必要な場合、キューは、登録されたすべてのクライアントにメッセージをポストします。

メッセージの受信

サブスクライバは、次のメカニズムのいずれかを介して、メッセージを受信できます。

- サブスクライバにかわって動作するクライアント・プロセスが、登録メカニズムを使用してコールバックを指定します。その後、メッセージがサブスクライバのサブスクリプションと一致する場合、ポスト・メカニズムによってコールバックが非同期に起動されます。メッセージ・コンテンツは、コールバック関数に渡される場合があります(非永続キューの場合のみ)。
- サブスクライバにかわって動作するクライアント・プロセスが、登録メカニズムを使用してコールバックを指定します。その後、ポスト・メカニズムによってコールバック関数が非同期に起動されますが、完全なメッセージ・コンテンツは渡されません。コールバック関数はクライアントへの通知として機能し、これに続いてプル型の方法でメッセージ・コンテンツを取得します(永続的キューの場合のみ)。
- サブスクライバにかわって動作するクライアント・プロセスが、周期的またはその他の方法で適宜キューから単純にメッセージを取得します。メッセージの遅延がある場合、エンド・クライアントへの非同期配信は行われません。

23.4 パブリッシュ・サブスクライブ・メカニズムの例

この例では、データベース・イベント、クライアント通知、AQがどのように連携してパブリッシュ・サブスクライブを実装するかを示します。

- ユーザー・スキーマの下で、パブリッシュ・サブスクライブ・メカニズムをサポートするために必要なすべてのオブジェクトを持つpubsubを作成します。このコードでは、エージェントsnoopは、ログイン・イベントで公開されるメッセージをサブスクライブします。ユーザーpubsubがAQ機能を使用するには、AQ_ADMINISTRATOR_ROLE権限と、DBMS_AQおよびDBMS_AQADMのEXECUTE権限が必要です。

```
Rem -----
REM create queue table for persistent multiple consumers:
Rem -----
Rem Create or replace a queue table
BEGIN
DBMS_AQADM.CREATE_QUEUE_TABLE(
  Queue_table      => 'Pubsub.Raw_msg_table',
  Multiple_consumers => TRUE,
  Queue_payload_type => 'RAW',
  Compatible       => '8.1');
END;
/
Rem -----
Rem Create a persistent queue for publishing messages:
Rem -----
Rem Create a queue for logon events
BEGIN
```

```

    DBMS_AQADM.CREATE_QUEUE(
        Queue_name      => 'Pubsub.Logon',
        Queue_table      => 'Pubsub.Raw_msg_table',
        Comment          => 'Q for error triggers');
END;
/
Rem -----
Rem  Start the queue:
Rem -----
BEGIN
    DBMS_AQADM.START_QUEUE('pubsub.logon');
END;
/
Rem -----
Rem  define new_enqueue for convenience:
Rem -----
CREATE OR REPLACE PROCEDURE New_enqueue(
        Queue_name      IN VARCHAR2,
        Payload          IN RAW ,
        Correlation      IN VARCHAR2 := NULL,
        Exception_queue  IN VARCHAR2 := NULL)
AS
    Enq_ct      DBMS_AQ.Enqueue_options_t;
    Msg_prop    DBMS_AQ.Message_properties_t;
    Enq_msgid   RAW(16);
    Userdata    RAW(1000);

BEGIN
    Msg_prop.Exception_queue := Exception_queue;
    Msg_prop.Correlation := Correlation;
    Userdata := Payload;
    DBMS_AQ.ENQUEUE(Queue_name, Enq_ct, Msg_prop, Userdata, Enq_msgid);
END;
/
Rem -----
Rem  add subscriber with rule based on current user name,
Rem  using correlation_id
Rem -----

DECLARE
Subscriber Sys.Aq$_agent;
BEGIN
    Subscriber := sys.aq$_agent('SNOOP', NULL, NULL);
    DBMS_AQADM.ADD_SUBSCRIBER(
        Queue_name      => 'Pubsub.logon',
        Subscriber       => subscriber,
        Rule             => 'CORRID = ''HR'' ');
END;
/
Rem -----
Rem  create a trigger on logon on database:
Rem -----
Rem  create trigger on after logon:
CREATE OR REPLACE TRIGGER pubsub.Systrig2
    AFTER LOGON
    ON DATABASE
    BEGIN
        New_enqueue('Pubsub.Logon', HEXTORAW('9999'), Dbms_standard.login_user);
    END;
/

```

- サブスクリプションが作成された後、次のステップとして、クライアントがコールバック関数を使用した通知を登録します。これにはOracle Call Interface(OCI)を使用します。次のコードは、登録に必要なステップを実行します。セッション・ハンドルの割当ておよび初期化を行う最初のステップは、例をわかりやすくするために、ここでは省略します。

```

ub4 namespace = OCI_SUBSCR_NAMESPACE_AQ;
/* callback function for notification of logon of user 'HR' on database: */
ub4 notifySnoop(ctx, subscrhp, pay, payl, desc, mode)
dvoid *ctx;
OCISubscription *subscrhp;
dvoid *pay;
ub4 payl;
dvoid *desc;
ub4 mode;
{
    printf("Notification : User HR Logged on\n");
}
int main()
{
    OCISession *authp = (OCISession *) 0;
    OCISubscription *subscrhpSnoop = (OCISubscription *)0;
    /******
       Initialize OCI Process/Environment
       Initialize Server Contexts
       Connect to Server
       Set Service Context
    *****/
    /* Registration Code Begins */
    /* Each call to initSubscriptionHn allocates
       and Initialises a Registration Handle */

    initSubscriptionHn(    &subscrhpSnoop,    /* subscription handle */
        "ADMIN:PUBSUB.SNOOP", /* subscription name */
        /* <agent_name>:<queue_name> */
        (dvoid*)notifySnoop); /* callback function */
    /******
       The Client Process does not need a live Session for Callbacks
       End Session and Detach from Server
    *****/
    OCISessionEnd ( svchp, errhp, authp, (ub4) OCI_DEFAULT);
    /* detach from server */
    OCIServerDetach( srvhp, errhp, OCI_DEFAULT);
    while (1) /* wait for callback */
        sleep(1);
}
void initSubscriptionHn (subscrhp,
subscriptionName,
func)
OCISubscription **subscrhp;
char* subscriptionName;
dvoid * func;
{
    /* allocate subscription handle: */
    (void) OCIHandleAlloc((dvoid *) envhp, (dvoid **)subscrhp,
        (ub4) OCI_HTYPE_SUBSCRIPTION,
        (size_t) 0, (dvoid **) 0);
    /* set subscription name in handle: */
    (void) OCIAttrSet((dvoid *) *subscrhp, (ub4) OCI_HTYPE_SUBSCRIPTION,
        (dvoid *) subscriptionName,
        (ub4) strlen((char *)subscriptionName),
        (ub4) OCI_ATTR_SUBSCR_NAME, errhp);
    /* set callback function in handle: */
    (void) OCIAttrSet((dvoid *) *subscrhp, (ub4) OCI_HTYPE_SUBSCRIPTION,
        (dvoid *) func, (ub4) 0,
        (ub4) OCI_ATTR_SUBSCR_CALLBACK, errhp);
    (void) OCIAttrSet((dvoid *) *subscrhp, (ub4) OCI_HTYPE_SUBSCRIPTION,
        (dvoid *) 0, (ub4) 0,
        (ub4) OCI_ATTR_SUBSCR_CTX, errhp);
    /* set namespace in handle: */
    (void) OCIAttrSet((dvoid *) *subscrhp, (ub4) OCI_HTYPE_SUBSCRIPTION,
        (dvoid *) &namespace, (ub4) 0,
        (ub4) OCI_ATTR_SUBSCR_NAMESPACE, errhp);
}

```

```
    checkerr(errhp, OCISubscriptionRegister(svchp, subscrhp, 1, errhp,  
        OCI_DEFAULT));  
}
```

ユーザーHRがデータベースにログインすると、クライアントに通知され、コールバック関数notifySnoopが起動されます。

24 Oracle ODBC Driverの使用方法

この章の構成は、次のとおりです。

トピック:

- [Oracle ODBC Driverについて](#)
- [すべてのユーザー](#)
- [上級ユーザー](#)
- [プログラマ](#)

24.1 Oracle ODBC Driverについて

ODBCとは

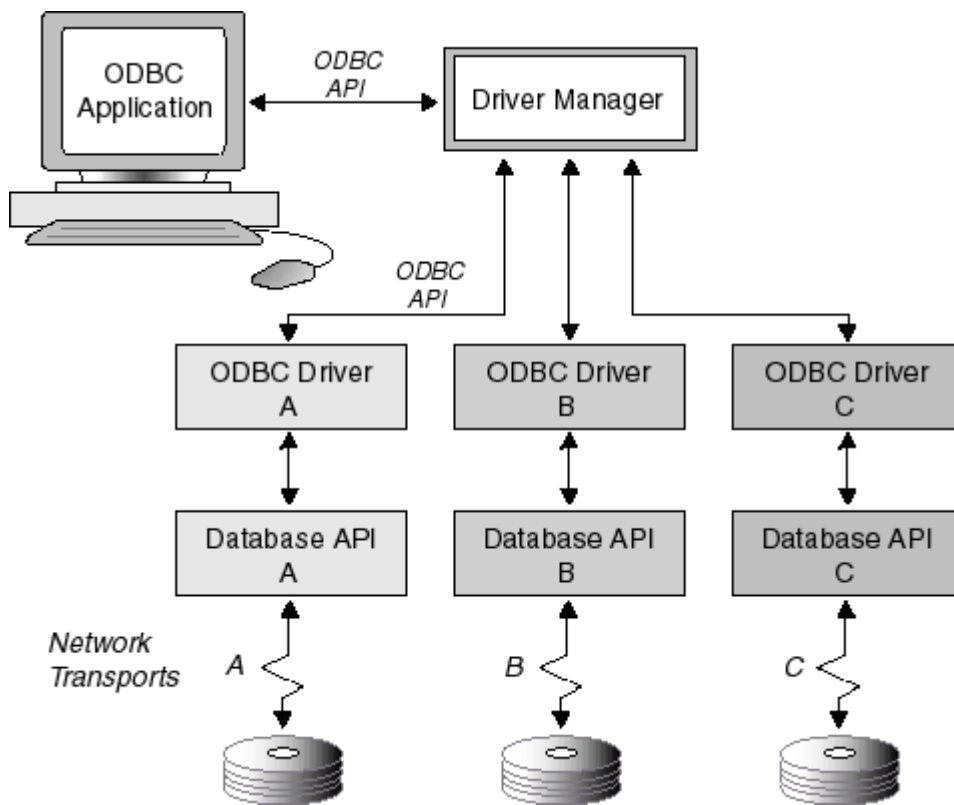
Open Database Connectivity (ODBC)は、1つのアプリケーションが多くの異なるデータソースにアクセスできるようにする標準インタフェースを提供します。アプリケーションのソース・コードをデータ・ソースごとにコンパイルしなおす必要はありません。データベース・ドライバにより、アプリケーションが特定のデータ・ソースにリンクされます。データベース・ドライバは、アプリケーションが特定のデータソースにアクセスするためにオンデマンドで呼び出すことのできるダイナミック・リンク・ライブラリです。したがって、アプリケーションはデータベース・ドライバが存在するすべてのデータ・ソースにアクセスできます。

ODBCインタフェースには次のものが定義されています。

- ODBC関数コールのライブラリ。アプリケーションではこのライブラリを利用してデータソースに接続し、構造化問合せ言語(SQL)文を実行して結果を取り出すことができます。
- SQL-99仕様に基づくSQL構文
- エラー・コードの標準セット
- データソースへの接続とログインの標準的な方法
- データ型の標準的な表現

次の図に、ODBCモデルのコンポーネントを示します。モデルは、ODBCアプリケーションがODBCアプリケーション・プログラム・インタフェース(API)を介してドライバ・マネージャをコールするところから始まります。ドライバ・マネージャは、Microsoftドライバ・マネージャまたはunixODBCドライバ・マネージャです。ドライバ・マネージャは、ここでもODBC APIを使用してODBCドライバをコールします。ODBCドライバは、データベースAPIを使用し、ネットワーク通信リンク経由でデータベースにアクセスします。次の図に、異なる3つのデータベースにアクセスするODBCアプリケーションを示します。

図24-1 ODBCモデルのコンポーネント



関連トピック

[Oracle ODBC Driverとは](#)

24.2 すべてのユーザー

トピック:

- [Oracle ODBC Driver](#)
- [構成タスク](#)
- [oraodbc.iniファイルの変更](#)
- [データソースへの接続](#)
- [トラブルシューティング](#)

24.2.1 Oracle ODBC Driver

トピック:

- [Oracle ODBC Driverとは](#)
- [新機能と変更点](#)
- [サポートされていない機能](#)
- [インストールによって作成されるファイル](#)
- [ドライバの準拠レベル](#)
- [既知の制限事項](#)

24.2.1.1 Oracle ODBC Driverとは

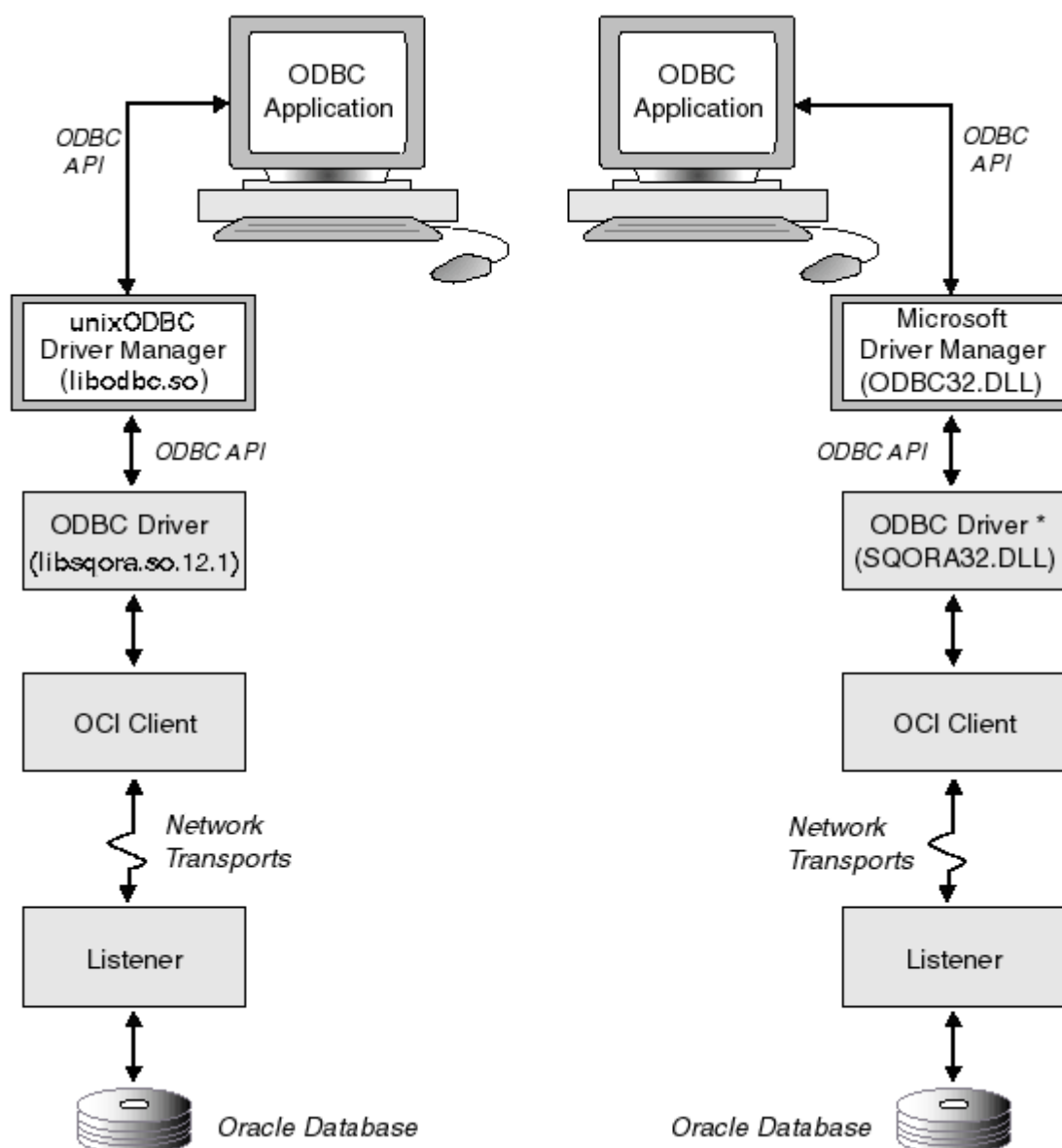
Oracle ODBC Driverを使用すると、Microsoft WindowsおよびLinux、Solaris、IBM Advanced Interactive eExecutive (AIX)などのUNIXプラットフォーム上のアプリケーションで、Oracle Net Servicesソフトウェアを使用し、ODBCインタフェースを介して、Oracleデータベースに対する読み取りおよび書き込みアクセスができます。

Oracle ODBC Driverは、Oracle Call Interface(OCI)のクライアント/サーバー・ソフトウェアを介して、データソースへの要求の送信およびデータソースからの応答を受信します。OCIクライアントとOracleサーバー間の通信には、Oracle Net Services通信プロトコルが使用されます。

Oracle ODBC Driverは、ODBC SQL構文をデータソースにアクセスできる構文に変換します。データソースから結果が戻されると、Oracle ODBC Driverは戻された結果をODBC SQL構文に変換します。

次の図に、前の段落で説明しているOracle ODBC Driverアーキテクチャを示します。

図24-2 Oracle ODBC Driverアーキテクチャ



* Oracle ODBCリソース・データ定義言語(DLL)ファイル(sqresxx.dll、xxは言語の略称)には、言語に関連する全情報が含まれています。使用されるデフォルトのリソース・ファイルはsqresus.dllです。

OCIクライアント/サーバー・ソフトウェアの詳細は、OCIのマニュアルを参照してください。

関連トピック

[データソースの構成](#)

[データソースへの接続](#)

[ドライバの準拠レベル](#)

[新機能と変更点](#)

[インストールによって作成されるファイル](#)

24.2.1.2 新機能と変更点

トピック:

- [Oracle ODBC Driverリリース19c、バージョン19.1.0.0.0の新機能](#)
- [Oracle ODBC Driverリリース18c、バージョン18.1.0.0.0の新機能](#)
- [Oracle ODBC Driverリリース12.2.0.1.0の新機能](#)
- [Oracle ODBC Driverリリース12.1.0.2.0の新機能](#)
- [Oracle ODBC Driverリリース12.1.0.1.0の新機能](#)
- [Oracle ODBC Driverリリース11.2.0.1.0の新機能](#)
- [Oracle ODBC Driverリリース11.1.0.1.0の新機能](#)
- [Oracle ODBC Driverリリース10.1.0.2.0の新機能](#)
- [Oracle ODBC Driverリリース10.1.0.2.0の変更点](#)

Oracle ODBC Driverリリース19c、バージョン19.1.0.0.0の新機能

Microsoft Windows Server 2008、Windows Server 2008 R2、Windows Server 2012、Windows Server 2012 R2、Windows 7、Windows 8、Windows 8.1、Windows 10、Linux X86-64 (32ビット、64ビット)、Sun Solaris SPARC64 (32ビット、64ビット)、IBM AIX 5L (32ビット、64ビット)、Sun Solaris X64 (32ビット、64ビット)、HPUX IA64 (32ビット、64ビット)、ZLinux (32ビット、64ビット)の各オペレーティング・システム用のOracle ODBC Driverリリース19c、バージョン19.1.0.0.0ソフトウェアの新機能はありません。

Oracle ODBC Driverリリース18c、バージョン18.1.0.0.0の新機能

Microsoft Windows Server 2008、Windows Server 2008 R2、Windows Server 2012、Windows Server 2012 R2、Windows 7、Windows 8、Windows 8.1、Windows 10、Linux X86-64 (32/64 bit)、Sun Solaris SPARC64 (32,64 bit)、IBM AIX 5L (32、64ビット)、Sun Solaris X64 (32、64ビット)、HPUX IA64 (32、64ビット)、ZLinux (32、64ビット)の各オペレーティング・システム用のOracle ODBC Driverリリース18.1.0.0.0ソフトウェアの機能は、次のとおりです。

- unixODBC ODBCドライバ・マネージャは、unixODBC-2.3.2からunixODBC-2.3.4にアップグレードされています。

Oracle ODBC Driverリリース12.2.0.1.0の新機能

Microsoft Windows Server 2008、Windows Server 2008 R2、Windows Server 2012、Windows Server 2012 R2、Windows 7、Windows 8、Windows 8.1、Windows 10、Linux X86-64 (32/64 bit)、Sun Solaris SPARC64 (32,64 bit)、IBM AIX 5L (32、64ビット)、Sun Solaris X64 (32、64ビット)、HPUX IA64 (32、64ビット)、ZLinux (32、64ビット)の各オペレーティング・システム用のOracle ODBC Driverリリース12.2.0.1.0ソフトウェアの機能は、次のとおりです。

トウェアの機能は、次のとおりです。

- 長い識別子のサポートが追加されました。

Oracle ODBC Driverは、128バイトの長さのオブジェクトをサポートするようになりました。以前のリリースでは、オブジェクトの長さは30バイトに制限されていました。

- タイムゾーン付きのタイム・スタンプとローカル・タイムゾーン付きのタイム・スタンプのサポートが追加されました。

この機能は、ODBC TIMESTAMPデータ型が使用されている既存のODBCアプリケーションに対する変更を必要としません。既存のアプリケーションでODBC TIMESTAMP日付型を使用し、データベース列がTIMESTAMPの場合、現在の動作が維持されます。

データベース列TIMESTAMP WITH TIMEZONEまたはTIMESTAMP WITH LOCAL TIMEZONEでは、ODBC TIMESTAMP_STRUCT内の時間コンポーネントは、ユーザーのセッションタイム・ゾーンにあります。この動作は、ユーザーのアプリケーションに対して透過的なため、ODBCアプリケーションを変更する必要がありません。

Oracle ODBC Driverリリース12.1.0.2.0の新機能

Microsoft Windows Server 2008、Windows Server 2008 R2、Windows Server 2012、Windows 7、Windows 8、Windows 10、Linux X86-64 (32/64ビット)、Sun Solaris SPARC64 (32、64ビット)、IBM AIX 5L (32、64ビット)、Sun Solaris X64 (32、64ビット)、HPUX IA64 (32、64ビット)、ZLinux (32、64ビット)の各オペレーティング・システム用のOracle ODBC Driverリリース12.1.0.2.0ソフトウェアの機能は、次のとおりです。

Microsoft Windows 10プラットフォームが追加されました。

Oracle ODBC Driverリリース12.1.0.1.0の新機能

Microsoft Windows Server 2008、Windows Server 2008 R2、Windows Server 2012、Windows 7、Windows 8、Linux X86-64 (32/64ビット)、Sun Solaris SPARC64 (32、64ビット)、IBM AIX 5L (32、64ビット)、Sun Solaris X64 (32、64ビット)、HPUX IA64 (32、64ビット)、ZLinux (32、64ビット)の各オペレーティング・システム用のOracle ODBC Driverリリース12.1.0.1.0ソフトウェアの機能は、次のとおりです。

- Oracle ODBC Driverは、VARCHAR2、NVARCHAR2およびRAWデータで32KBデータ列をサポートするようになりました。
- odbc.iniファイルまたは接続レベル属性の新しいパラメータ:
 - SQL_TRANSLATE_ERRORS = {T|F} [デフォルトはF (false)]

SQL翻訳フレームワーク機能を使用している移行済のサード・パーティODBCアプリケーションでは、サーバーから返されるエラーがネイティブ・データベース形式を取ると予想されます。ユーザーは、SQL翻訳フレームワーク・モードで稼働しているOracle DatabaseのSQL翻訳プロファイルにエラーの翻訳を登録できます。エラーの翻訳が登録されたら、ODBCアプリケーションのユーザーはオプションSQLTranslateErrors = Tを有効化し、登録されているとおりのネイティブ・エラーを受け取ることができます。

詳細は、[表24-4](#)を参照してください。

- Oracle ODBC Driverが、RefCursorを使用せずに暗黙的結果を返すことのできるストアド・プロシージャの実行をサポートするようになりました。このサポートにより、Oracleに移行したサード・パーティのODBCアプリケーションで、旧ベンダーが提供していたこの同じ機能を使用しやすくなります。
- Oracle Databaseの自動増分機能をサポートするためのSQLColAttribute()フィールド識別子の拡張サポート。Oracle ODBC Driver固有のヘッダー・ファイルsqora.hをアプリケーションに含めることで、この機能を使用できます。

- SQL_COLUMN_AUTO_INCREMENT

Oracle Database 12cリリース1 (12.1.0.1)以降では、Oracleが自動増分列をサポートするため、Oracle ODBC Driverは既存のSQLColAttribute()識別子SQL_COLUMN_AUTO_INCREMENTを通じて同じサポートを拡張しました。このプロパティは読み取り専用であり、列が自動増分の場合にSQL_TRUEを返し、それ以外の場合にSQL_FALSEを返します。

- SQL_ORCLATTR_COLUMN_PROP

Oracle Database 12cリリース1 (12.1.0.1)以降では、Oracle ODBC Driverは、列の属性を返す新しいドライバ固有フィールド識別子SQL_ORCLATTR_COLUMN_PROPをサポートします。この識別子は、次に示すようにすべての列プロパティを持つSQLULEN値を返します。

```
+-----+
| 32 | ... | 10 | 9 | 8 | ..... | 3 | 2 | 1 |
+-----+
|      |      |      |
|      |      | -> Column is auto-increment?
|      |      | -> Auto value is always generated?
|      |      | -> If generated by default when null?
```

- Oracle Database 12cリリース1 (12.1.0.1)でサポートされるODBC API

- `SQLMoreResults()`

暗黙的結果のODBCサポートを実装します。

Oracle ODBC Driverリリース11.2.0.1.0の新機能

Microsoft Windows XP、Microsoft Windows 2003 Server、Microsoft Windows Vista、Linux X86-32 (RHEL AS 4、5)、Linux X86-64 (RHEL AS 4、5) (32/64ビット)、Sun Solaris SPARC64 (9、10) (32/64ビット)、IBM AIX 5L 5.2 (32/64ビット)、Linux IA64 (64ビット)、Linux on Power (32/64ビット)、Sun Solaris X64 (64ビット)、Hewlett Packard Itanium (32/64ビット)オペレーティング・システム用のOracle ODBC Driverリリース 11.2.0.1.0ソフトウェアの機能は次のとおりです。

- LONGおよびLONG RAWデータのプリフェッチ

Oracle ODBC Driverは、ODBCアプリケーションのパフォーマンスを向上させるために、LONGまたはLONG RAWデータをプリフェッチするように拡張されています。これを行うには、WindowsではLONGデータの最大サイズ(MaxLargeData)をレジストリに設定し(データソース名(DSN)にレジストリ・キーMaxLargeDataを追加する必要があります)、UNIXプラットフォームではodbc.iniファイルにこれを手動で設定する必要があります。この機能強化により、ユーザーが設定したMaxLargeDataサイズに応じてOracle ODBC Driverのパフォーマンスが最大10倍向上します。MaxLargeDataのデフォルト値は0です。設定できるMaxLargeDataの最大値は64KB (65536バイト)です。

MaxLargeDataの値が65536より大きい値に設定されていても、フェッチされるデータは65536バイトのみです。データベースに65536バイトより大きいLONGまたはLONG RAWデータがある場合は、単一行のフェッチが行われ、完全なLONGデータがフェッチされるように、MaxLargeDataを0 (デフォルト値)に設定します。ポーリング以外のモードでMaxLargeDataサイズより少ないバッファ・サイズを渡すと、データベースのLONGデータ・サイズがバッファ・サイズより大きい場合には、データの切捨てエラーが発生します。

- メタデータをフェッチするためのOCIDescribeAny()の使用オプション

アプリケーションが、REF CURSORSを返す小さなパッケージ・プロシージャを大量にコールする場合、ドライバにOCIDescribeAny()の使用を強制することでパフォーマンスを向上させることができます。このオプションを有効にす

るには、UNIXプラットフォームで、またWindowsではレジストリを使用して、odbc.iniのUseOCIDescribeAnyの値をT (True)に設定します。デフォルト値はF (False)です。

Oracle ODBC Driverリリース11.1.0.1.0の新機能

Windows XP、Linux、SolarisおよびIBM AIXオペレーティング・システム用のOracle ODBC Driverリリース11.1.0.1.0ソフトウェアの機能は次のとおりです。

- RULEヒントを無効化(DRH接続文字列)

新しい接続オプション「RULEヒントを無効化」が追加されました。これにより、ユーザーは、カタログAPIでRULEヒントを使用するかどうかを選択するオプションを指定できます。カタログAPIのODBCドライバのパフォーマンスを向上させるための変更が行われました。オプションのデフォルト値は、RULEヒントがカタログAPIで使用されないことを意味するTRUEです。

- NUMBERをFLOATとしてバインド(BNF接続文字列)

新しい接続オプション「NUMBERをFLOATとしてバインド」が追加されました。列に浮動小数点データが含まれる場合にFLOATとしてのNUMBER列の列バインディングを導入することにより、バインド変数をFLOATとして使用する問合せ実行が高速になります。

- 文キャッシング

各セッションの文のキャッシュを提供および管理するOCI文キャッシュ機能のサポートが追加されました。OCI文のキャッシュ・オプションのサポートを実装することで、ユーザーが同じ接続で同じ文を複数回解析する場合にOracle ODBC Driverのパフォーマンスが向上します。文キャッシュ・フラグのデフォルト値はFALSEです。

Oracle ODBC Driverリリース10.1.0.2.0の新機能

Windows 98、Windows 2000、Windows XPおよびWindows NT X86オペレーティング・システム用のOracle ODBC Driverリリース10.1.0.2.0ソフトウェアの機能は次のとおりです。

- TIMESTAMPをDATEとしてバインド(BTD接続文字列)

新規の「TIMESTAMPをDATEとしてバインド」接続オプションが追加されました。これによりODBCドライバのSQL_TIMESTAMPデータ型を、OracleのTIMESTAMPデータ型ではなくDATEデータ型でバインドできます(これはデフォルトです)。

- MONTHNAME (exp)ファンクション

日付式で表される月の名前を返すMONTHNAME (exp)ファンクションのサポートが追加されました。たとえば、「April」などです。

- DAYNAME (exp)ファンクション

日付式で表される日の名前を返すDAYNAME (exp)ファンクションのサポートが追加されました。たとえば、「Tuesday」などです。

- インスタント・クライアントの構成

インスタント・クライアント・モードの構成がサポートされるようになりました。

Oracle ODBC Driverリリース10.1.0.2.0の変更点

Oracle ODBC Driverリリース10.1.0.2.0で変更された機能、また非推奨機能は次のとおりです。

- Microsoft Transaction Serverの無効化

Microsoft Transaction Server (MTS)を無効にするデフォルト設定が、FALSEからTRUEに変更されました。MTSのサポートは、デフォルトで無効になっています。

- 浮動小数点データ型

Oracleのデータ型であるBINARY_FLOATおよびBINARY_DOUBLEは、ODBCデータ型のSQL_REALおよびSQL_DOUBLEにそれぞれマップされるように変更されました。

- SQLGetDataの拡張(GDE接続文字列)

このリリースでは、SQLGetDataの拡張接続は非推奨です。このオプションの機能は常に使用できます。

- LONG列の強制取り出し(FRL接続文字列)

このリリースでは、「LONG列強制取り出し」接続オプションは非推奨です。このオプションの機能は常に使用できます。

- 「翻訳オプション」構成タブ

このリリースでは、「Oracle ODBCドライバ構成」ダイアログ・ボックスにあった「翻訳オプション」タブが非推奨になりました。

- リリース・ノート

リリース・ノートのファイル名が、ODBCReInotes.wriからODBCReInotesUS.htmに変更されました。

関連項目:

- Timestamp例は、[「ODBCのOracle TIMESTAMP WITH TIME ZONEおよびTIMESTAMP WITH LOCAL TIME ZONE列型のサポートについて」](#)を参照してください
- 32KBの列の作成は、[『Oracle Database PL/SQL言語リファレンス』](#)および[『Oracle Database SQL言語リファレンス』](#)を参照してください。
- SQL翻訳フレームワークの詳細は、[SQL翻訳フレームワークのアーキテクチャと概要](#)、[翻訳フレームワークのインストールと構成](#)および[移行例](#)を参照してください
- Oracle Databaseによる暗黙的結果のサポートの詳細は、[Oracle Database移行ガイド](#)を参照してください
- 自動増分の詳細は、[『Oracle Call Interfaceプログラマーズ・ガイド』](#)を参照してください
- SQLMoreResults()関数の詳細は、[表24-6](#)および[「SQLMoreResults関数」](#)を参照してください。

24.2.1.3 サポートされていない機能

現在のOracle ODBC Driverでサポートされていない機能。

Oracle ODBC Driverは、ODBC 3.0の次にあげる機能はサポートしていません。

- 間隔データ型
- 共有接続
- 共有環境

Oracle ODBC Driverでは、次に示すSQL文字列関数はサポートしていません。

- BIT_LENGTH
- CHAR_LENGTH

- CHARACTER_LENGTH
- DIFFERENCE
- OCTET_LENGTH
- POSITION

Oracle ODBC Driverでは、次に示すSQL数値関数はサポートされていません。

- ACOS
- ASIN
- ATAN
- ATAN2
- COT
- DEGREES
- RADIANS
- RAND
- ROUND

Oracle ODBC Driverでは、次に示すSQL関数(時間、日付、間隔)はサポートされていません。

- CURRENT_DATE
- CURRENT_TIME
- CURRENT_TIMESTAMP
- EXTRACT
- TIMESTAMPDIFF

24.2.1.4 インストールによって作成されるファイル

次の表に、Oracle ODBC Driverキットによってインストールされるファイルを示します。

表24-1 Oracle ODBC Driverキットでインストールされるファイル

説明:	Windowsインストールのファイル名	UNIXインストールのファイル名
Oracle ODBC データベース・アクセ ス DLL	sqora32.dll	libsqora.so.nn.n (nn.n はバージョン番号を表 します。たとえば、libsqora.so.20.1)
Oracle ODBC Driver 設定 DLL	sqoras32.dll	なし
Oracle ODBC リ ソース DLL	sqresus.dll	なし
日本語用の Oracle ODBC リ ソース DLL	sqresja.dll	なし

説明:	Windowsインストールのファイル名	UNIXインストールのファイル名
Oracle ODBC Driver メッセージ・ ファイル	oraodbcus.msb	oraodbcus.msb
日本語用の Oracle ODBC Driver メッセージ・ ファイル	oraodbcja.msb	oraodbcja.msb
Oracle ODBC Driver リリース・ ノート	Oracle Database ODBC Driver リリース・ノート	Oracle Database ODBC Driver リリース・ノート
Oracle ODBC Driver Instant Client リリース・ノ ート	ODBC_IC_Readme_Win.html	ODBC_IC_Readme_Unix.html
Oracle ODBC Driver ヘルプ・ファ イル	sqora.htm	sqora.htm
日本語用の Oracle ODBC Driver ヘルプ・ファ イル	sqora.htm	sqora.htm
Oracle ODBC Driver Instant Client インストー ル・スクリプト	odbc_install.exe	odbc_update_ini.sh
Oracle ODBC Driver Instant Client アンインス トール・スクリプト	odbc_uninstall.exe	なし

Microsoftドライバ・マネージャおよびAdministratorファイル

Microsoft ODBC 3.52コンポーネントとともにインストールされるファイルのリストは、Microsoft ODBC 3.52 Software Development Kit and Programmer's Referenceを参照してください。

Microsoft ODBCコンポーネントは、Microsoft Data Access Component (MDAC)キットに含まれています。
WindowsのOracle ODBC Driverは、MDACバージョン2.8を使用してテストされています。

unixODBCドライバ・マネージャおよびAdministratorファイル

unixODBCドライバ・マネージャとともにインストールされるファイルのリストは、unixODBCのREADMEファイルおよびINSTALLファイルを参照してください。

関連項目:

- MDACキットをダウンロードする[MDAC Kit](#)
- unixODBCドライバをダウンロードする[Unix ODBCドライバ・マネージャ](#)

24.2.1.5 ドライバの準拠レベル

ODBCは、次の2つの分野でドライバの準拠レベルを定義しています。

- ODBCアプリケーション・プログラミング・インタフェース(API)
- ODBC SQL-99の構文

Oracle ODBC Driverは、APIのすべてのコア機能と、レベル1およびレベル2機能セットの一部をサポートしています。

Oracle ODBC DriverはSQL-92のエントリ・レベル仕様のスーパーセットであるSQL-99のコア仕様と広く互換性があります。SQL-99でサポートされる機能のリストを取得するには、アプリケーションにおいて適切な情報型を指定してSQLGetInfoをコールする必要があります。

関連項目:

コアAPI機能サポートの詳細は、[「API準拠」](#)を参照してください

24.2.1.6 既知の制限事項

次の事項は、Oracle ODBC Driverではサポートされていません。

- ODBC ASYNCインタフェース
- アプリケーションでの[Ctrl]+[C]を使用した取消し

24.2.2 構成タスク

トピック:

- [Oracle Net Servicesの構成](#)
- [データソースの構成](#)
- [「Oracle ODBCドライバ構成」ダイアログ・ボックス](#)

24.2.2.1 Oracle Net Servicesの構成

[データソースを構成する](#)前に、ネットワーク・データベース・サービスを構成して各Transparent Network Substrate (TNS) サービス名のエントリを構成する必要があります。これを行うには、Oracle Net Configuration Assistant (NETCA)を使用します。

NETCAを使用してtnsnames.oraファイルに各TNSサービス名のエントリを作成できます。Oracle Net Servicesのインストール時に、NETCAもインストールされます。

24.2.2.2 データソースの構成

ノート:



次の構成ステップは Windows ユーザー用です。UNIX ユーザーは、odbc_update_ini.sh ファイルを使用して DSN を作成する必要があります。

Oracle ODBC Driverをインストールして[Oracle Net Servicesを構成](#)した後、Oracle ODBC Driverを使用する前に、データソースを構成する必要があります。

アプリケーションとデータソース間で通信を行うには、構成情報が設定されている必要があります。Oracle ODBC Driverは、この構成情報からアクセス対象の情報を取得します。

データソースは、アクセス対象のデータと、それに対応付けられたオペレーティング・システム、データベース管理システムおよびデータベース管理システムにアクセスするためのネットワーク・プラットフォームで構成されます。Oracle ODBC Driverが発行する要求のデータソースはOracleデータベースであり、このデータソースはOracle Net Servicesによるトランスポートをサポートしています。

Oracleデータソースを構成または追加するには:

Oracle ODBC Driverをインストールしたら、ODBCデータソース管理者を使用して、各OracleデータベースにOracleデータソースを構成または追加します。Oracle ODBC Driverは、データソースを追加する際にユーザーが入力する情報を使用してデータにアクセスします。次のステップを実行します。

1. 「スタート」メニューから「プログラム」→「管理ツール」→「データ ソース (ODBC)」を選択します。インストールされているドライバのリストが表示されます。
2. 新規データ ソースの作成ウィンドウで「追加」をクリックし、インストール済ドライバのリストで「Oracle ODBC Driver」を選択します。
3. 「終了」をクリックします。[「Oracle ODBCドライバ構成」ダイアログ・ボックス](#)が表示されます。DSNとTNSサービス名は必ず入力する必要があります。このダイアログ・ボックスでその他の必要な情報を入力することも、フィールドを空白のままにして、アプリケーションを実行するときに情報を指定することもできます。
4. データを入力したら、「OK」をクリックするか[Enter]を押します。

データソースはいつでも変更または削除できます。次の項では、データソースの追加、変更または削除方法を説明します。

Oracleデータソースを変更するには:

1. 「スタート」メニューから「プログラム」→「管理ツール」→「データ ソース (ODBC)」を選択します。
2. 「ODBC データ ソース アドミニストレータ」ダイアログ・ボックスで、データソースのリストからデータソースを選択し、「構成」をクリックします。「Oracle ODBCドライバ構成」ダイアログ・ボックスが表示されます。
3. [「Oracle ODBCドライバ構成」ダイアログ・ボックス](#)で、必要に応じてオプション値を変更し、「OK」をクリックします。

Oracleデータソースを削除するには:

1. 「スタート」メニューから「プログラム」→「管理ツール」→「データ ソース (ODBC)」を選択します。

2. 「ODBC データ ソース アドミニストレータ」ダイアログ・ボックスで、データソースのリストから削除するデータソースを選択します。
3. 「削除」、「はい」の順にクリックして、削除を確定します。

24.2.2.3 「Oracle ODBCドライバ構成」ダイアログ・ボックス

ノート:



「Oracle ODBC ドライバ構成」ダイアログ・ボックスを使用できるのは、Microsoft Windows ユーザーのみです。

次のスクリーンショットは「Oracle ODBCドライバ構成」ダイアログ・ボックスの例を示したものです。

図24-3 「Oracle ODBCドライバ構成」ダイアログ・ボックス

次のリストは、前図に示した「Oracle ODBCドライバ構成」ダイアログ・ボックスにあるメイン設定オプションおよびフィールドの説明です。ダイアログ・ボックスの下半分にあるタブは次以降のトピックで説明します。

- データソース名(DSN) - ODBCでデータソースを識別する名前。たとえば、「odbc-pc」です。DSNは入力する必要があります。
- 説明 - データソースに含まれているデータについての説明やコメント。たとえば、「全従業員の入社日、給与履歴、現行評価」などします。「説明」フィールドはオプションです。
- TNSサービス名 - ODBCドライバがデータを取得するOracleデータベースの位置。これは、[Oracle Net Servicesを構成](#)したときに、Oracle Net Configuration Assistant (NETCA)を使用して入力したものと同名です。詳細は、NETCAのマニュアルと[「Oracle ODBC Driverの最初の使用について」](#)を参照してください。TNSサービス名は、使用可能なTNS名のプルダウン・リストから選択できます。たとえば、「ODBC-PC」です。TNSサービス名を入

力する必要があります。

- ユーザーID - データにアクセスするために使用するサーバー上のアカウントのユーザー名です。たとえば、「scott」です。「ユーザーID」フィールドはオプションです。

DSNとTNSサービス名は必ず入力する必要があります。このダイアログ・ボックスでその他の情報を入力することも、フィールドを空白のままにして、アプリケーションを実行するときに情報を指定することもできます。

前述のメイン設定オプションの他に、「接続テスト」ボタンがあります。このボタンは、DSN定義で指定されたデータベースに接続することによってODBC環境が適切に構成されているかを確認します。「接続テスト」ボタンを押し、ユーザー名およびパスワードを入力してください。

「Oracle ODBCドライバ構成」ダイアログ・ボックスの下半分にあるオプション・タブの説明は、次のリンクのいずれかをクリックします。

[アプリケーション・オプション](#)

[Oracleオプション](#)

[Workaroundsオプション](#)

[SQL Server移行オプション](#)

アプリケーション・オプション

次のスクリーンショットは、「Oracle ODBCドライバ構成」ダイアログ・ボックスにある「アプリケーション」タブの例を示したものです。

図24-4 「Oracle ODBCドライバ構成」ダイアログ・ボックスの「アプリケーション」オプション・タブ

The screenshot shows the 'Oracle ODBC Driver Configuration' dialog box with the 'Application' tab selected. The top section contains input fields for 'Data Source Name', 'Description', 'TNS Service Name' (a dropdown menu), and 'User ID'. To the right of these fields are buttons for 'OK', 'Cancel', 'Help', and 'Test Connection'. The bottom section contains a tabbed interface with 'Application', 'Oracle', 'Workarounds', and 'SQLServer Migration' tabs. The 'Application' tab is active, showing several options: 'Enable Result Sets' (checked), 'Enable Query Timeout' (checked), 'Read-Only Connection' (unchecked), 'Enable Closing Cursors' (unchecked), and 'Enable Thread Safety' (checked). Below these are two dropdown menus: 'Batch Autocommit Mode' set to 'Commit only if all statements succeed' and 'Numeric Settings' set to 'Use Oracle NLS settings'.

次のリストは、前図で示した「アプリケーション」タブにあるフィールドの説明です。

- 結果セットを有効化 - Oracle結果セットの処理を有効にします。結果セットがアプリケーションに必要な場合、結果セットのサポートを無効にできます。パッケージからの結果セットを含んでいないプロシージャ・コールは、少々パフォー

マンスが低下します。デフォルトでは、結果セットは有効です。

- 問合せタイムアウトを有効化 - SQL問合せで、タイムアウトを有効にします。デフォルトでは、Oracle ODBC DriverはSQLSetStmtAttr関数用にSQL_ATTR_QUERY_TIMEOUT属性をサポートします。このボックスが選択されていない場合、Oracle ODBC Driverは、「不可」メッセージを返します。デフォルトでは、問合せタイムアウトは有効です。
- 読取専用接続 - 読取り専用アクセスを指定するには、このボックスを選択します。デフォルトは、書込み可能な接続です。
- カーソル・クローズを有効化 - カーソル・クローズを有効にします。デフォルトでは、カーソル・クローズは無効(フィールドが空)になっています。これは、不必要なパフォーマンス低下の原因となるためこの動作が望ましくない場合に、カーソルのクローズをコールしてもOCIカーソルのクローズが強制されないことを意味します。カーソルのクローズをコールしたときにOCIカーソルを強制的にクローズする場合は、カーソル・クローズを有効にします。

ノート:



カーソルがクローズされるたびに、パフォーマンスに影響があります。

- スレッド・セーフティを有効化 - データソースについてスレッド・セーフティを無効にすることもできます。スレッド・セーフティが不要な場合は、このオプションを指定しなければスレッド・セーフティのオーバーヘッドをなくすることができます。デフォルトではスレッド・セーフティは有効です。
- バッチ自動コミット・モード - デフォルトでは、すべての文が成功した場合、コミットが実行されます。
- 数値の設定 - 文字列としてバインドされた数値データの受渡しで、どの数値設定を使用して小数点文字や3桁セパレータを決定するかを指定します。このオプションでは、Oracle NLSの設定(デフォルトの設定)、Microsoftのデフォルトの地域設定(Oracle OLE DBドライバの動作とあわせて相互運用性を向上)、US数値設定(英語環境以外でMS AccessまたはDAO (Database Access Object)を使用する際に必要)の選択ができます。

関連項目:

メイン構成設定オプションは、[「Oracle ODBCドライバ構成」ダイアログ・ボックス](#)を参照してください

Oracleオプション

次のスクリーンショットは、「Oracle ODBCドライバ構成」ダイアログ・ボックスにあるOracleオプション・タブの例を示したものです。

図24-5 「Oracle ODBCドライバ構成」ダイアログ・ボックスの「Oracle」オプション・タブ

Oracle ODBC Driver Configuration

Data Source Name

Description

TNS Service Name

User ID

OK

Cancel

Help

Test Connection

Application
Oracle
Workarounds
SQLServer Migration

Fetch Buffer Size

Enable LOBs ☒

Enable Statement Caching ☐

Cache Buffer Size

Max Token Size

Translate ORA errors ☐

Failover Support

Enable Failover ☒

Retry

Delay

Aggregate SQL Type

Lob Prefetch Size

次のリストは、前図で示した「Oracle」タブにあるフィールドの説明です。

- **フェッチ・バッファ・サイズ** - 単一の問合せでアプリケーション・プログラムがリクエストする行数に関係なく、ODBCドライバがOracle Databaseから一度にプリフェッチするデータの行数を決定するために使用されるメモリー量。ただし、プリフェッチされる行数は、単一の問合せで指定された列の幅と数に依存します。通常一度に20行未満をフェッチするアプリケーションでは、低速なネットワーク接続上で動作する場合や非常に負荷の高いサーバーにアクセスする場合には特に、レスポンス時間が向上します。フェッチ・バッファ・サイズを大きく設定しすぎると、レスポンス時間が悪化したり、メモリーを大量に消費することがあります。

ノート:



LONG および LOB データ型がある場合、ODBC ドライバのプリフェッチ行数は、フェッチ・バッファ・サイズに依存しません。LONG および LOB データ型では、パフォーマンスの改善が見込めず、過度のメモリー消費を引き起こす可能性があります。LONG および LOB データ型が存在する場合にのみ、ODBC Driver はフェッチ・バッファ・サイズを無視し、既定の行数をプリフェッチします。

- **LOBを有効化** - Oracle LOBの書込みを可能にします。アプリケーションにOracle LOBの書込みが不要であれば、LOBのサポートは無効にできます。LOBを有効にすると、INSERT文とUPDATE文の処理時にわずかながらパフォーマンスが低下します。LOBの書込みはデフォルトで有効ですが、LOBデータ型をサポートしないOracleデータベースで

は無効になります。

- ステートメントのキャッシングを有効化 - 文キャッシュ機能を有効にします。これにより、ユーザーが問合せおよび関連パラメータの同じテキストを複数回解析する必要がある場合に、問合せの解析パフォーマンスが向上します。デフォルトでは無効になります。
- キャッシュバッファサイズ - 文のキャッシュには、サービス・コンテキスト(OCI_ATTR_STMTCACHE_SIZE)の属性を使用して変更可能な最大サイズ(文の数)があります。デフォルトのキャッシュ・バッファ・サイズは20で、文キャッシングのオプションが有効化されている場合にのみ使用されます。キャッシュ・バッファ・サイズを0に設定すると、文のキャッシング機能は無効になります。
- 最大トークン・サイズ - トークン・サイズを4KB (4096バイト)から開始する最も近い1KB (1024バイト)の倍数に設定します。デフォルト・サイズは8KB (8192バイト)です。設定できる最大値は128KB (131068バイト)です。
- ORAエラーを変換 - SQL翻訳フレームワーク機能を使用している移行済のサード・パーティODBCアプリケーションでは、サーバーから返されるエラーがネイティブ・データベース形式を取ると予想されます。このオプションを有効にすると、SQL翻訳プロファイルに登録されているエラー翻訳に基づいたネイティブ・エラーを受け取ることができます。
- 「Oracle」オプション・タブの「フェイルオーバー」領域には次のフィールドがあります:
 - フェイルオーバーを有効化 - Oracle Fail SafeとOracle Parallel Serverのフェイルオーバーの再試行を有効にします。このオプションは、Oracle Fail SafeとOracle Parallel Serverのフェイルオーバー機能に対する拡張です。このオプションで追加フェイルオーバー再試行が構成可能になります。デフォルトは有効です。
 - 再試行 - 接続のフェイルオーバーを試みる回数。デフォルトは10回です。
 - 遅延 - フェイルオーバーを試みる間隔の秒数。デフォルトは10秒です。
- 集計SQLタイプ - 集計関数で返される数値型(SQL_FLOAT、SQL_DOUBLEまたはSQL_DECIMAL)を指定します。
- LOBプリフェッチ・サイズ - 一度にデータベースからプリフェッチするLOBデータの量(バイト単位)を設定します。デフォルト・サイズは8192です。

ノート:



Oracle Fail Safe は非推奨で、今後のリリースではサポート対象外になり、使用できなくなる可能性があります。Oracle RAC One Node などの他の単一ノード・フェイルオーバー・オプションを評価することをお勧めします

ノート:



Oracle Fail Safe と Oracle Parallel Server の設定や使用方法については、それぞれの製品のドキュメントを参照してください。

関連項目:

メイン構成設定オプションは、[「Oracle ODBCドライバ構成」ダイアログ・ボックス](#)を参照してください

Workaroundsオプション

次のスクリーンショットは、「Oracle ODBCドライバ構成」ダイアログ・ボックスにあるWorkaroundsオプション・タブの例を示したものです。

図24-6 「Oracle ODBCドライバ構成」ダイアログ・ボックスの「Workarounds」オプション・タブ

The screenshot shows the 'Oracle ODBC Driver Configuration' dialog box with the 'Workarounds' tab selected. The dialog has four input fields at the top: 'Data Source Name', 'Description', 'TNS Service Name' (with a dropdown arrow), and 'User ID'. To the right of these fields are four buttons: 'OK', 'Cancel', 'Help', and 'Test Connection'. Below the input fields is a tabbed interface with four tabs: 'Application', 'Oracle', 'Workarounds' (which is active), and 'SQLServer Migration'. The 'Workarounds' tab contains a list of options with checkboxes and a text input field:

Option	Value
Bind TIMESTAMP as DATE	☐
Force SQL_WCHAR Support	☐
Disable Microsoft Transaction Server	☒
Set Metadata Id Default to SQL_TRUE	☐
Pre-fetch size for LONG column data	0
Disable SQLDescribeParam	☐
Bind NUMBER As FLOAT	☐
Disable RULE Hint	☒
Use OCIDescribeAny	☐

次のリストは、前図で示した「Workarounds」タブにあるフィールドの説明です。

- **TIMESTAMPをDATEとしてバインド** - このボックスを選択すると、Oracle ODBC Driverにより、SQL_TIMESTAMPパラメータがOracleのTIMESTAMP型(デフォルト)ではなくOracle DATE型で強制的にバインドされます。
- **SQL_WCHAR強制サポート** - このボックスを選択すると、SQLDescribeCol、SQLColumnsおよびSQLProcedureColumnsにより、SQL_CHAR列にはSQL_WCHARのデータ型、SQL_VARCHAR列にはSQL_WVARCHARのデータ型、およびSQL_LONGVARCHAR列にはSQL_WLONGVARCHARのデータ型がそれぞれ無条件で戻されるようになります。この機能により、これらのODBCコールの結果に依存するアプリケーション(たとえば、ADOアプリケーション)でのUnicodeのサポートが可能になります。デフォルトでは、このサポートは無効になっています。
- **MTS無効** - このボックスをクリアすると、Microsoft Transaction Server (MTS)サポートが有効になります。MTSのサポートは、デフォルトで無効になっています。
- **METADATA IDデフォルトにSQL_TRUEをセット** - このボックスを選択すると、SQL_ATTR_METADATA_ID接続と接続時の文属性のデフォルト値がSQL_TRUEに変更されます。通常ではSQL_ATTR_METADATA_IDはSQL_FALSEがデフォルトです。接続後に明示的に属性値を変更するアプリケーションのODBCコールが、このオプションにより影響を受けることなく、関数は問題なく終了します。デフォルトでは、このオプションは無効です。
- **LONG列データのサイズをプリフェッチ** - LONGまたはLONG RAWデータをプリフェッチしてODBCアプリケーションのパフォーマンスを向上させるには、この値を設定します。この機能強化により、ユーザーが設定したプリフェッチ・サイズに応じてOracle ODBC Driverのパフォーマンスが最大10倍向上します。デフォルトの値は0。設定できる最大値は64KB (65536バイト)です。

プリフェッチ・サイズの値が65536より大きい値に設定されていても、フェッチされるデータは65536バイトのみです。データベースに65536バイトより大きいLONGまたはLONG RAWデータがある場合は、単一行のフェッチが行われ、完全なLONGデータがフェッチされるように、プリフェッチ・サイズを0 (デフォルト値)に設定します。ポーリング以外のモードでプリフェッチ・サイズより少ないバッファ・サイズを渡すと、データベースのLONGデータ・サイズがバッファ・サイズより大きい場合には、データの切捨てエラーが発生します。

- SQLDescribeParamを無効化 - SQLDescribeParam関数が有効な場合、すべてのパラメータに対してSQL_VARCHARデータ型が戻されます。SQL_WCHAR強制サポート関数も有効な場合は、すべてのパラメータに対してSQL_WVARCHARデータ型が戻されます。デフォルトでは、この関数は有効です。
- NUMBERをFLOATとしてバインド - このボックスを選択すると、Oracle ODBC Driverで、FLOATデータを含むNUMBER列がBinary Float (デフォルト)ではなくFLOATとしてバインドされます。
- RULEヒントを無効化 - このボックスをクリアすると、カタログ問合せで指定されているRULE Hintが有効になります。デフォルトでは、RULE Hintのオプションは無効です。
- OCIDecribeAnyを使用 - このボックスを選択すると、アプリケーションがREF CURSORSを戻す小さなパッケージ・プロシージャを頻繁にコールする場合にドライバにOCIDecribeAny()の使用を強制することで、パフォーマンスが向上します。

関連項目:

- DATEおよびTIMESTAMPの詳細は、[「データ型の実装」](#)を参照してください
- SQL_ATTR_METADATA_ID属性の詳細は、[「ODBC API関数の実装」](#)を参照してください
- メイン構成設定オプションは、[「Oracle ODBCドライバ構成」ダイアログ・ボックス](#)を参照してください

SQL Server移行オプション

次のスクリーンショットは、「Oracle ODBCドライバ構成」ダイアログ・ボックスにあるSQL Server移行オプション・タブの例を示したものです。

図24-7 「Oracle ODBCドライバ構成」ダイアログ・ボックスの「SQLServer Migration」オプション・タブ

前の図の「SQLServer Migration」オプション・タブのフィールドは次のとおりです。

- **Enable EXEC Syntax:** SQL Server EXEC構文のサポートを有効にします。Oracle Databaseサーバーによって処理される前に、EXEC文で指定されるサブプログラム・コールは、対応するOracleサブプログラム・コールに変換されます。デフォルトではこのオプションは無効になっています。
- **スキーマ:** 変換されたOracleサブプログラムは、ユーザーのデフォルト・スキーマに定義されます。ただし、同じSQL Serverデータベースからのすべてのサブプログラムを、データベース名をスキーマ名としてOracleスキーマに移行する場合、このフィールドは「データベース」に設定します。同じSQL Serverユーザーに所有されていたすべてのサブプログラムを同じOracleスキーマに定義する場合、このフィールドは「オーナー」に設定します。デフォルトでは、このフィールドは空欄になっています。

関連項目:

メイン構成設定オプションは、[「\[Oracle ODBCドライバ構成\]ダイアログ・ボックス」](#)を参照してください

24.2.3 oraodbc.iniファイルの変更

トピック:

- [ロック・タイムアウトの短縮](#)

24.2.3.1 ロック・タイムアウトの短縮

Oracleサーバーは、トランザクション間のロック競合が解消されるまで無期限に待機します。Oracle ODBC Driverの oraodbc.iniファイルのLockTimeOutエントリを設定することで、Oracleサーバーがロックの解消を待機する時間を制限できます。LockTimeOutパラメータに入力する値は、Oracleサーバーがリクエストしたロックを取得できない場合にタイムアウトになるまでの秒数です。次の例では、Oracleサーバーは60秒後にタイムアウトになります。

[Oracle ODBC Driver Common]

24.2.4 データソースへの接続

トピック:

[Oracleデータソースへの接続](#)

24.2.4.1 Oracleデータソースへの接続

Oracle ODBC Driverでは、データソースに接続するために、使用するコンピュータにOCIクライアント・ソフトウェアがインストールされていて、対応するリスナーがOracleサーバーで実行されている必要があります。Oracle Net Services for Windowsは、ダイナミック・リンク・ライブラリ(DLL)に基づくアプリケーションです。Oracle Net Servicesの詳細は、Oracle Net Servicesのマニュアルを参照してください。

接続処理の一部として、アプリケーションがユーザーに情報を要求することがあります。アプリケーションからOracleデータソースについての情報を要求された場合は、次の手順を実行します。

1. 「TNSサービス名」ボックスで、TNSサービス名の名前を入力します。
2. 「ユーザー名」ボックスに、Oracleデータベースにアクセスするために使用する名前を入力します。
3. 「パスワード」ボックスに、Oracleデータベースにアクセスするために使用するパスワードを入力します。
4. 「OK」をクリックします。

アプリケーションは、データソース内のデータにアクセスするにはデータソースに接続する必要があります。データソースに接続するタイミングは、アプリケーションによって異なります。たとえば、アプリケーションがユーザーの要求時にのみデータソースに接続する場合もあれば、起動時に自動的に接続する場合もあります。アプリケーションがいつデータソースに接続するかは、そのアプリケーションのマニュアルを参照してください。

追加情報にアクセスするには、次のリンクのいずれかをクリックします。

- すべてのユーザー:
 - [データソースの構成](#)
- プログラマ:
 - [SQLDriverConnectの実装](#)
 - [データソース構成オプション](#)

24.2.5 トラブルシューティング

トピック:

- [Oracle ODBC Driverの最初の使用について](#)
- [パスワードの期限切れ](#)

24.2.5.1 Oracle ODBC Driverの最初の使用について

Oracle ODBC Driverを初めて使用する際に役立つ情報について説明します。

Oracle ODBC Driver開発者のホーム[「ODBC Developer Center」](#)を参照してください。ここでは、Oracle ODBC Driverの機能やOracle Instant Client ODBCのインストール・ガイド、Oracle Instant Client ODBCのダウンロード・サイト、Oracle ODBCのディスカッション・フォーラム、Oracle ODBC Driverの開発ガイドおよび関連技術に関する情報

などのリソースを入手できます。

24.2.5.2 パスワードの期限切れ

この項には、期限切れパスワードに関する情報が含まれます。

期限切れパスワードの動作

データベースの接続時にパスワードが期限切れの場合、パスワードを変更するよう要求されます。正常にパスワードが変更されると、データベースに接続されます。ただし、SQL_DRIVER_NOPROMPTパラメータ値を使用してSQLDriverConnectコールでデータベースに接続すると、Oracle ODBC Driverはパスワードを変更するよう要求しません。かわりにエラーとなり、パスワードが期限切れであることを示すエラー番号およびメッセージを表示します。

24.3 上級ユーザー

トピック:

- [Oracle ODBC DriverのTNSサービス名の作成](#)
- [SQL文](#)
- [データ型](#)
- [データ型の実装](#)
- [データ型に関する制限事項](#)
- [エラー・メッセージ](#)

24.3.1 Oracle ODBC DriverのTNSサービス名の作成

Oracle Net Servicesを使用してOracle ODBC DriverのTNSサービス名を作成するには、Oracle Net Servicesと同時にインストールされるOracle Net Configuration Assistant (NETCA)を使用します。NETCAによって、Oracle ODBC DriverのTNSサービス名のエントリがtnsnames.oraファイルに作成されます。

24.3.2 SQL文

Oracle ODBC DriverはSQL-92のエントリ・レベル仕様のスーパーセットであるSQL-99のコア仕様と広く互換性があります。Oracleの文法に加えて、ODBC仕様の付録Cで概要を説明しているベンダー固有のエスケープ・シーケンスもサポートしています。ODBCの設計に従って、Oracle ODBC Driverは固有のSQL構文をOracle Databaseに渡します。

関連項目:

- 上級ユーザーは、[「データ型」](#)を参照してください
- [ODBC SQL構文の実装](#)

プログラム

24.3.3 データ型

Oracleデータベースのデータ型はOracle ODBC DriverによってODBCのSQLデータ型にマップされます。

ノート:



Microsoft ODBC 3.52 Software Development Kit and Programmers Reference の Appendix D にあるすべての変換は、適切な情報型を指定した SQLGetInfo のコールでリストされる ODBC SQL データ型についてサポートされています。

関連項目:

- 上級ユーザー:
 - [データ型の実装](#)
 - [データ型に関する制限事項](#)
 - [SQL文](#)
- プログラマ:
 - [データ型の実装\(プログラミング\)](#)

24.3.4 データ型の実装

トピック:

- [DATEおよびTIMESTAMP](#)
- [浮動小数点のデータ型](#)

DATEおよびTIMESTAMP

OracleのDATEおよびTIMESTAMPデータ型のセマンティクスは、同名のODBCデータ型と必ずしも正確に対応していません。OracleのDATEデータ型には日付情報および時間情報の両方が、SQL_DATEデータ型には日付情報のみが含まれます。OracleのTIMESTAMPデータ型にも日付と時間の情報が格納されていますが、その小数秒の精度は他方に比較して高くなります。ODBCドライバでは、OracleのDATEおよびTIMESTAMP列の両方のデータ型がSQL_TIMESTAMPとしてレポートされ、情報の紛失が回避されています。また、ODBCドライバにより、SQL_TIMESTAMPパラメータがOracleのTIMESTAMP値としてバインドされます。

浮動小数点データ型

リリース10.1以降のOracleサーバーに接続すると、ODBCドライバにより、Oracleの浮動小数点のデータ型であるBINARY_FLOATおよびBINARY_DOUBLEが、ODBCデータ型であるSQL_REALおよびSQL_DOUBLEにそれぞれマップされます。以前のリリースでは、SQL_REALおよびSQL_DOUBLEがOracleの汎用数値データ型にマップされていました。

関連項目:

[DATEおよびTIMESTAMPデータ型](#)

24.3.5 データ型に関する制限事項

Oracle ODBC DriverおよびOracleデータベースでは、データ型について制限があります。次の表に制限事項を示します。

表24-2 Oracle ODBC DriverおよびOracle Databaseデータ型に関する制限事項

制限されるデータ型	説明:
リテラル	Oracle Database では SQL 文内のリテラルが 32K バイトに制限されます。
SQL_LONGVARCHAR および SQL_WLONGVARCHAR	SQL_LONGVARCHAR データ(LONG 型)の Oracle での制限は 2,147,483,647 バイトです。SQL_LONGVARCHAR データ(CLOB 型)の Oracle での制限は 4GB です。これは、クライアント・ワークステーションのメモリによる制限です。
SQL_LONGVARCHAR および SQL_LONGVARBINARY	Oracle データベースでは 1 つの表では LONG データ列を 1 つしか使用できません。LONG データ型とは、SQL_LONGVARCHAR(LONG)および SQL_LONGVARBINARY(LONG RAW)です。かわりに CLOB および BLOB を使用することをお勧めします。1 つの表で使用できる CLOB および BLOB の列数に制限はありません。

24.3.6 エラー・メッセージ

エラーが発生すると、Oracle ODBC Driverは固有のエラー番号、SQLSTATE (ODBCエラー・コード)およびエラー・メッセージを戻します。ドライバが検出したエラーとOracleサーバーが戻したエラーの両方から、この情報を導出します。

固有のエラー

データソースで発生したエラーについては、Oracle ODBC DriverはOracleサーバーから戻された固有のエラーを戻します。Oracle ODBC Driverまたはドライバ・マネージャがエラーを検出した場合は、Oracle ODBC Driverは0 (ゼロ)という固有のエラーを戻します。

SQLSTATE

データソースで発生したエラーについては、Oracle ODBC Driverは戻された固有のエラーを適切なSQLSTATEにマップします。Oracle ODBC Driverがエラーを検出した場合は、Oracle ODBC Driverが適切なSQLSTATEを生成します。ドライバ・マネージャがエラーを検出した場合は、ドライバ・マネージャが適切なSQLSTATEを生成します。

エラー・メッセージ

データソースで発生したエラーについては、Oracle ODBC DriverはOracleサーバーから戻されたメッセージに基づくエラー・メッセージを戻します。Oracle ODBC Driverまたはドライバ・マネージャで発生したエラーについては、Oracle ODBC DriverはSQLSTATEに関連するテキストに基づくエラー・メッセージを戻します。

エラー・メッセージの書式は次のとおりです。

```
[vendor] [ODBC-component] [data-source] error-message
```

大カッコ([])内の接頭辞によってエラーのソースが識別されます。次の表に、Oracle ODBC Driverから返される接頭辞の値を示します。エラーがデータソースで発生したときは、[vendor]および[ODBC-component]という接頭辞によって、ベンダーと、データソースからエラーを受け取るODBCコンポーネントの名前が識別されます。

表24-3 Oracle ODBC Driverが戻す接頭辞のエラー・メッセージ値

エラー・ソース	接頭辞	値

エラー・ソース	接頭辞	値
ドライバ・マネージャ	[vendor][ODBC-component][data-source]	[Microsoft/unixODBC][ODBC Driver Manager]N/A
Oracle ODBC Driver	[vendor][ODBC-component][data-source]	[ORACLE][ODBC Driver]N/A
Oracle サーバー	[vendor][ODBC-component][data-source]	[ORACLE][ODBC Driver]N/A

たとえば、次に示すフォーマットのようにエラー・メッセージに[Ora]という接頭辞が含まれない場合、そのエラーはOracle ODBC Driverのエラーであり、自明であるために接頭辞が省略されています。

```
[Oracle][ODBC]Error message text here
```

次に示すフォーマットのようにエラー・メッセージに[Ora]という接頭辞が含まれる場合、Oracle ODBC Driverのエラーではありません。

ノート:



エラー・メッセージに接頭辞[Ora]が含まれていても、実際のエラーはいくつかのソースの 1 つで発生している可能性があります。

```
[Oracle][ODBC][Ora]Error message text here
```

エラー・メッセージのテキストが次に示す接頭辞で始まる場合は、そのエラーについての詳しい情報がOracleサーバーのマニュアルに記載されています。

ORA-

Oracle Net Servicesエラーおよびトレース・ロギングは、OCIソフトウェアがインストールされたWindowsシステムのORACLE_HOME\NETWORKディレクトリまたはUNIXシステムのORACLE_HOME/NETWORKディレクトリの下、具体的にはそれぞれlogディレクトリとtraceディレクトリにあります。データベース・ロギングは、Oracleサーバー・ソフトウェアがインストールされたWindowsシステムのORACLE_HOME\RDBMSディレクトリまたはUNIXシステムのORACLE_HOME/rdbmsディレクトリにあります。

サーバーのエラー・メッセージの詳細は、Oracleサーバーのマニュアルを参照してください。

24.4 プログラマ

トピック:

- [接続文字列のフォーマット](#)
- [SQLDriverConnectの実装](#)
- [プログラムでのロック・タイムアウトの削減](#)
- [odbc32.lib \(Windows\)またはlibodbc.so \(UNIX\)とのリンク](#)

- [ROWIDの情報](#)
- [WHERE句でのROWID](#)
- [結果セットの有効化](#)
- [EXEC構文の有効化](#)
- [Oracle RAC環境における接続障害のイベント通知の有効化](#)
- [ODBCを通じた暗黙的結果機能の使用](#)
- [ODBCでのOracle TIMESTAMP WITH TIME ZONEおよびTIMESTAMP WITH LOCAL TIME ZONE列タイプのサポートについて](#)
- [Oracleクライアント\(OCI、SQL*Plus、Oracle ODBC Driverなど\)でORA_SDTZを設定した場合の影響について](#)
- [サポートされている機能](#)
- [Unicodeのサポート](#)
- [パフォーマンスとチューニング](#)

24.4.1 接続文字列のフォーマット

次の表に、SQLDriverConnect関数コールの接続文字列引数に組み込むことができるキーワードを示します。指定されていないキーワードは、データソースのAdministratorのエントリから読み込まれます。接続文字列に指定される値はAdministratorのエントリに格納されている値を上書きします。SQLDriverConnect関数の詳細は、Microsoft ODBC 3.52 Software Development Kit and Programmers Referenceを参照してください。

表24-4 SQLDriverConnect関数コールの接続文字列引数に組み込むことができるキーワード

キーワード	意味	コメント
DSN	ODBC データソース名	ユーザー指定の名前。
DBQ	TNS サービス名	ユーザー指定の名前。
UID	ユーザーID またはユーザー名	ユーザー指定の名前。
PWD	パスワード	ユーザー指定パスワード。空のパスワードに対しては、PWD=;と指定。
DBA	データベース属性	W=書込みアクセス。 R=読取り専用アクセス。
APA	アプリケーション属性	T=スレッド・セーフティ有効。 F=スレッド・セーフティ無効。

キーワード	意味	コメント
RST	結果セット	T=結果セット有効。 F=結果セット無効。
QTO	問合せタイムアウト・オプション	T=クエリー・タイムアウト有効。 F=クエリー・タイムアウト無効。
CSR	カーソルのクローズ	T=カーソル・クローズ有効。 F=カーソル・クローズ無効。
BNF	NUMBER を FLOAT としてバインド	T=NUMBER を FLOAT としてバインド。 F=NUMBER を NUMBER としてバインド。
DRH	RULE ヒントを無効化	T=RULE ヒントを無効化。 F=RULE ヒントを有効化。
BAM	バッチ自動コミット・モード	IfAllSuccessful=すべての文が成功した場合のみコミット (古い動作)。 UpToFirstFailure=最初に失敗した文までコミット(V7 ODBC の動作)。 AllSuccessful=すべての成功した文をコミット(Oracle Database に接続している場合のみ、その他のデータベー スでは、V7 と同じ動作)。
FBS	フェッチ・バッファ・サイズ	ユーザー指定の数値(バイト単位で 0 以上を指定)。 デフォルトは 60,000 バイトです。
FEN	フェイルオーバー	T=フェイルオーバー有効。 F=フェイルオーバー無効。
FRC	フェイルオーバー再試行数	ユーザー指定の数値。 デフォルト値は 10 です。

キーワード	意味	コメント
FDL	ファイルオーバー遅延	ユーザー指定の数値。 デフォルト値は 10 です。
LOB	LOB 書込み	T=LOB 有効。 F=LOB 無効。
MTS	Microsoft Transaction Server サポート	T=無効。 F=有効。
FWC	SQL_WCHAR 強制サポート	T=強制 SQL_WCHAR 有効。 F=強制 SQL_WCHAR 無効。
EXC	EXEC 構文	T=EXEC 構文有効。 F=EXEC 構文無効。
XSM	スキーマ・フィールド	Default=デフォルト。 Database=データベース名。 Owner=所有者名。
MDI	メタデータ ID のデフォルトの設定	T=SQL_ATTR_METADATA_ID のデフォルトを SQL_TRUE に設定。 F=SQL_ATTR_METADATA_ID のデフォルトを SQL_FALSE に設定。
DPM	SQLDescribeParam の無効化	T=SQLDescribeParam 無効。 F=SQLDescribeParam 有効。
BTD	TIMESTAMP を DATE としてバインド	T=SQL_TIMESTAMP を Oracle DATE でバインド。 F=SQL_TIMESTAMP を Oracle TIMESTAMP でバインド。

キーワード	意味	コメント
NUM	数値の設定	NLS=Oracle NLS 数値設定を使用(小数点および 3 桁区切りの決定)。 MS=Microsoft 地域設定を使用。 US=US 設定を使用。
ODA	OCIDescribeAny()の使用	T=アプリケーションが REF CURSORS を戻す小さなパッケージ・プロシージャを頻繁にコールする場合にパフォーマンスを向上させるために OCIDescribeAny()コールを使用。 F=OCIDescribeAny()を使用しない。デフォルトでは、OCIDescribeAny()値の使用は FALSE です。
STE	SQL で ORA エラーを変換	T=ORA エラーを変換。
	Oracle ODBC Driver が Oracle エラー・コードを変換するかどうかを指定	F=ORA エラーを変換しない。デフォルトでは、SQLTranslateErrors は FALSE です。
TSZ	トークン・サイズ	ユーザー指定の数値。 トークン・サイズを 4KB (4096 バイト)から開始する最も近い 1KB (1024 バイト)の倍数に設定します。デフォルト・サイズは 8KB (8192 バイト)です。設定できる最大値は 128KB (131068 バイト)です。

次のキーワードが接続文字列に指定された場合、Oracle ODBC DriverはAdministratorで定義されている値を読み取りません。

```
DRIVER={Oracle ODBC Driver}
```

有効な接続文字列の例:

```
1) DSN=Personnel;UID=Kotzwinkle;PWD=;2) DRIVER={Oracle ODBC Driver};UID=Kotzwinkle;PWD=whatever;DBQ=instl_alias;DBA=W;
```

関連項目:

- すべてのユーザーは、[「Oracleデータソースへの接続」](#)を参照してください
- プログラマは、[「SQLDriverConnectの実装」](#)を参照してください

24.4.2 SQLDriverConnectの実装

次の表に、SQLDriverConnect接続文字列に必要なキーワードを示します。

表24-5 SQLDriverConnect接続文字列に必要なキーワード

キーワード	説明:
DSN	データソース名。
DBQ	TNS サービス名。 「Oracle ODBC Driver の TNS サービス名の作成」 を参照してください。詳細は、Oracle Net Services のマニュアルを参照してください。
UID	ユーザーのログイン ID またはユーザー名。
PWD	ユーザー指定のパスワード。

24.4.3 プログラムでのロック・タイムアウトの短縮

Oracleサーバーは、トランザクション間のロック競合が解消されるまで無期限に待機します。データソースに接続する前に ODBC SQLSetConnectAttr関数をコールすることによって、ロック解消をOracleサーバーが待機する時間の長さを制限できます。ODBC SQLSetStmtAttr関数のSQL_ATTR_QUERY_TIMEOUT属性に0 (ゼロ)以外の値を指定します。

ODBC SQLSetConnectAttr関数を使用してロック・タイムアウト値を指定すると、その値によってoraodbc.iniファイルに指定されている値が上書きされます。

関連項目:

oraodbc.iniファイルに値を指定する方法の詳細は、[「ロック・タイムアウトを短縮する」](#)を参照してください

24.4.4 odbcc32.lib (Windows)またはlibodbc.so (UNIX)とのリンク

Windowsプラットフォームの場合、プログラムをリンクするときは、インポート・ライブラリodbc32.libとリンクする必要があります。UNIXプラットフォームの場合は、ODBCアプリケーションをlibodbc.soにリンクする必要があります。

24.4.5 ROWIDの情報

ODBCのSQLSpecialColumns関数は、表内の列に関する情報を返します。Oracle ODBC Driverでは、Oracle表に対応付けられているOracleのROWIDについての情報が戻されます。

24.4.6 WHERE句でのROWID

ROWIDはSQL文のWHERE句で使用できます。ただし、ROWID値はパラメータ・マーカー内で使用する必要があります。

24.4.7 結果セットの有効化

Oracleの参照カーソル(結果セット)によって、アプリケーションはストアド・ファンクションやストアド・プロシージャを使用してデータ

を取り出せるようになります。次にODBCを介して結果セットを使用可能にする方法を示します。

- ストアド・プロシージャのコールにはODBC構文を使用する必要があります。システム固有のPL/SQLは、ODBCを介してサポートされていません。次にパッケージ内のおよびパッケージなしのプロシージャまたはファンクションのコール方法を示します。ここでのパッケージ名をRSETとします。

```
Procedure call:
{CALL Example1(?)}
{CALL RSET.Example1(?)}
Function Call:
{? = CALL Example1(?)}
{? = CALL RSET.Example1(?)}
```

- PL/SQLの参照カーソル・パラメータは、プロシージャのコール時には省略されます。たとえば、Example2プロシージャが4つのパラメータを持つと仮定します。パラメータ1と3は参照カーソル・パラメータで、パラメータ2と4は文字列です。コールは、次のように指定されます。

```
{CALL RSET.Example2("Literal 1", "Literal 2")}
```

次のサンプル・アプリケーションでは、Oracle ODBC Driverを使用して結果セットを返す方法を示します。

```
/*
 * Sample Application using Oracle reference cursors via ODBC
 *
 * Assumptions:
 *
 * 1) Oracle Sample database is present with data loaded for the EMP table.
 * 2) Two fields are referenced from the EMP table ename and mgr.
 * 3) A data source has been setup to access the sample database.
 *
 * Program Description:
 *
 * Abstract:
 *
 * This program demonstrates how to return result sets using
 * Oracle stored procedures
 *
 * Details:
 *
 * This program:
 * Creates an ODBC connection to the database.
 * Creates a Packaged Procedure containing two result sets.
 * Executes the procedure and retrieves the data from both result sets.
 * Displays the data to the user.
 * Deletes the package then logs the user out of the database.
 *
 *
 * The following is the actual PL/SQL this code generates to
 * create the stored procedures.
 *
 * DROP PACKAGE ODBCRefCur;
 *
 * CREATE PACKAGE ODBCRefCur AS
 * TYPE ename_cur IS REF CURSOR;
 * TYPE mgr_cur IS REF CURSOR;
 * PROCEDURE EmpCurs(Ename IN OUT ename_cur, Mgr IN OUT mgr_cur, pjob IN VARCHAR2);
 * END;
 *
 * CREATE or REPLACE PACKAGE BODY ODBCRefCur AS
 * PROCEDURE EmpCurs(Ename IN OUT ename_cur, Mgr IN OUT mgr_cur, pjob IN VARCHAR2)
 * AS
 * BEGIN
 * IF NOT Ename%ISOPEN
 * THEN
```



```

* OPEN Ename for SELECT ename from emp;
* END IF;
* IF NOT Mgr%ISOPEN
* THEN
* OPEN Mgr for SELECT mgr from emp where job = pjob;
* END IF;
* END;
* END;
*
*/
/* Include Files */
#ifdef WIN32
#include <windows.h>
#endif
#include <stdio.h>
#include <sql.h>
#include <sqlext.h>

/* Defines */
#define JOB_LEN 9
#define DATA_LEN 100
#define SQL_STMT_LEN 500

/* Procedures */
void DisplayError(SWORD HandleType, SQLHANDLE hHandle, char *Module);

/* Main Program */
int main()
{
    SQLHENV hEnv;
    SQLHDBC hDbc;
    SQLHSTMT hStmt;
    SQLRETURN rc;
    char *DefUserName = "scott";
    char *DefPassWord = "tiger";
    SQLCHAR ServerName[DATA_LEN];
    SQLCHAR *pServerName=ServerName;
    SQLCHAR UserName[DATA_LEN];
    SQLCHAR *pUserName=UserName;
    SQLCHAR Password[DATA_LEN];
    SQLCHAR *pPassword=Password;
    char Data[DATA_LEN];
    SQLINTEGER DataLen;
    char error[DATA_LEN];
    char *charptr;
    SQLCHAR SqlStmt[SQL_STMT_LEN];
    SQLCHAR *pSqlStmt=SqlStmt;
    char *pSalesMan = "SALESMAN";
    SQLINTEGER sqlnts=SQL_NTS;
    /* Allocate the Environment Handle */
    rc = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hEnv );
    if (rc != SQL_SUCCESS)
    {
        printf( "Cannot Allocate Environment Handle/n");
        printf( "/nHit Return to Exit/n");
        charptr = gets ((char *)error);
        exit(1);
    }
    /* Set the ODBC Version */
    rc = SQLSetEnvAttr(hEnv, SQL_ATTR_ODBC_VERSION, (void *)SQL_OV_ODBC3, 0);
    if (rc != SQL_SUCCESS)
    {
        printf("Cannot Set ODBC Version/n");
        printf("/nHit Return to Exit/n");
        charptr = gets((char *)error);
        exit(1);
    }
}

```

```

/* Allocate the Connection handle */
rc = SQLAllocHandle(SQL_HANDLE_DBC, hEnv, &hDbc);
if (rc != SQL_SUCCESS)
{
    printf("Cannot Allocate Connection Handle/n");
    printf("/nHit Return to Exit/n");
    charptr = gets((char*) error);
    exit(1);
}
/* Get User Information */
lstrcpy((char*) pUserName, DefUserName);
lstrcpy((char*) pPassword, DefPassWord);
/* Data Source name */
printf( "/nEnter the ODBC Data Source Name/n" );
charptr = gets((char*) ServerName);
/* User Name */
printf("/nEnter User Name Default [%s]/n", pUserName);
charptr = gets((char*) UserName);
if (*charptr == '/0')
{
    lstrcpy((char*) pUserName, (char*) DefUserName);
}
/* Password */
printf( "/nEnter Password Default [%s]/n", pPassWord);
charptr = gets((char*) PassWord);
if (*charptr == '/0')
{
    lstrcpy((char*) pPassWord, (char*) DefPassWord);
}
/* Connection to the database */
rc = SQLConnect(hDbc, pServerName, (SQLSMALLINT) lstrlen((char *)pServerName),
pUserName,
(SQLSMALLINT) lstrlen((char*)pUserName), pPassWord,
(SQLSMALLINT) lstrlen((char *)pPassWord));
if (rc != SQL_SUCCESS)
{
    DisplayError(SQL_HANDLE_DBC, hDbc, "SQLConnect");
}

/* Allocate a Statement */
rc = SQLAllocHandle(SQL_HANDLE_STMT, hDbc, &hStmt);
if (rc != SQL_SUCCESS)
{
    printf( "Cannot Allocate Statement Handle/n");
    printf( "/nHit Return to Exit/n");
    charptr = gets((char *)error);
    exit(1);
}
/* Drop the Package */
lstrcpy((char *) pSqlStmt, "DROP PACKAGE ODBCRefCur");
rc = SQLExecDirect(hStmt, pSqlStmt, lstrlen((char *)pSqlStmt));
/* Create the Package Header */
lstrcpy((char *) pSqlStmt, "CREATE PACKAGE ODBCRefCur AS/n" );
lstrcat((char *) pSqlStmt, " TYPE ename_cur IS REF CURSOR;/n" );
lstrcat((char *) pSqlStmt, " TYPE mgr_cur IS REF CURSOR;/n" );
lstrcat((char *) pSqlStmt, " PROCEDURE EmpCurs (Ename IN OUT ename_cur," );
lstrcat((char *) pSqlStmt, " Mgr IN OUT mgr_cur,pjob IN VARCHAR2);/n/n");
lstrcat((char *) pSqlStmt, "END;/n" );
rc = SQLExecDirect(hStmt, pSqlStmt, lstrlen((char *)pSqlStmt));
if (rc != SQL_SUCCESS)
{
    DisplayError(SQL_HANDLE_STMT, hStmt, "SQLExecDirect");
}
/* Create the Package Body */
lstrcpy((char *) pSqlStmt, "CREATE PACKAGE BODY ODBCRefCur AS/n" );
lstrcat((char *) pSqlStmt, " PROCEDURE EmpCurs (Ename IN OUT ename_cur," );
lstrcat((char *) pSqlStmt, " Mgr IN OUT mgr_cur, pjob IN VARCHAR2)/n" );

```

```

lstrcat((char *) pSqlStmt, " AS/n" );
lstrcat((char *) pSqlStmt, " BEGIN/n" );
lstrcat((char *) pSqlStmt, " IF NOT Ename%ISOPEN/n" );
lstrcat((char *) pSqlStmt, " THEN/n" );
lstrcat((char *) pSqlStmt, " OPEN Ename for SELECT ename from emp;/n" );
lstrcat((char *) pSqlStmt, " END IF;/n/n" );
lstrcat((char *) pSqlStmt, " IF NOT Mgr%ISOPEN/n THEN/n" );
lstrcat((char *) pSqlStmt, " OPEN Mgr for SELECT mgr from emp where job =
pjob;/n");
lstrcat((char *) pSqlStmt, " END IF;/n" );
lstrcat((char *) pSqlStmt, " END;/n" );
lstrcat((char *) pSqlStmt, "END;/n" );
rc = SQLExecDirect(hStmt, pSqlStmt, strlen((char *)pSqlStmt));
if(rc != SQL_SUCCESS)
    DisplayError(SQL_HANDLE_STMT, hStmt, "SQLExecDirect");
/* Bind the Parameter */
rc = SQLBindParameter(hStmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, JOB_LEN, 0,
pSalesMan, 0, &sqlnts);
/* Call the Store Procedure which executes the Result Sets */
lstrcpy( (char *) pSqlStmt, "{CALL ODBCRefCur.EmpCurs(?)}");
rc = SQLExecDirect(hStmt, pSqlStmt, strlen((char *)pSqlStmt));
if(rc != SQL_SUCCESS)
    DisplayError(SQL_HANDLE_STMT, hStmt, "SQLExecDirect");
/* Bind the Data */
rc = SQLBindCol(hStmt, 1, SQL_C_CHAR, Data, sizeof(Data), &DataLen);
if(rc != SQL_SUCCESS)
    DisplayError(SQL_HANDLE_STMT, hStmt, "SQLBindCol");

/* Get the data for Result Set 1 */
printf("/nEmployee Names/n/n");
while(rc == SQL_SUCCESS)
{
    rc = SQLFetch(hStmt);
    if(rc == SQL_SUCCESS)
        printf("%s/n", Data);
    else
        if(rc != SQL_NO_DATA)
            DisplayError(SQL_HANDLE_STMT, hStmt, "SQLFetch");
}
printf( "/nFirst Result Set - Hit Return to Continue/n");
charptr = gets ((char *)error);
/* Get the Next Result Set */
rc = SQLMoreResults( hStmt );
if(rc != SQL_SUCCESS)
    DisplayError(SQL_HANDLE_STMT, hStmt, "SQLMoreResults");
/* Get the data for Result Set 2 */
printf("/nManagers/n/n");
while (rc == SQL_SUCCESS)
{
    rc = SQLFetch(hStmt);
    if(rc == SQL_SUCCESS)
        printf("%s/n", Data);
    else
        if (rc != SQL_NO_DATA)
            DisplayError(SQL_HANDLE_STMT, hStmt, "SQLFetch");
}
printf("/nSecond Result Set - Hit Return to Continue/n");
charptr = gets((char *)error);
/* Should Be No More Results Sets */
rc = SQLMoreResults( hStmt );
if (rc != SQL_NO_DATA)
    DisplayError(SQL_HANDLE_STMT, hStmt, "SQLMoreResults");
/* Drop the Package */
lstrcpy((char *)pSqlStmt, "DROP PACKAGE ODBCRefCur");
rc = SQLExecDirect(hStmt, pSqlStmt, strlen((char *)pSqlStmt));
/* Free handles close connections to the database */
SQLFreeHandle( SQL_HANDLE_STMT, hStmt );

```

```

SQLDisconnect( hDbc );
SQLFreeHandle( SQL_HANDLE_DBC, hDbc );
SQLFreeHandle( SQL_HANDLE_ENV, hEnv );
printf( "\nAll Done - Hit Return to Exit\n");
charptr = gets ((char *)error);
return(0);
}

/* Display Error Messages */
void DisplayError( SWORD HandleType, SQLHANDLE hHandle, char *Module )
{
    SQLCHAR MessageText[255];
    SQLCHAR SQLState[80];
    SQLRETURN rc=SQL_SUCCESS;
    LONG NativeError;
    SWORD RetLen;
    SQLCHAR error[25];
    char *charptr;
    rc = SQLGetDiagRec(HandleType, hHandle, 1, SQLState, &NativeError, MessageText, 255,
&RetLen);
    printf( "Failure Calling %s\n", Module );
    if (rc == SQL_SUCCESS || rc == SQL_SUCCESS_WITH_INFO)
    {
        printf( "/t/t/t State: %s\n", SQLState);
        printf( "/t/t/t Native Error: %d\n", NativeError );
        printf( "/t/t/t Error Message: %s\n", MessageText );
    }
    printf( "\nHit Return to Exit\n");
    charptr = gets ((char *)error);
    exit(1);
}

```

24.4.8 EXEC構文の有効化

このオプションを有効にすると、変更なしに変換可能な場合、Oracle ODBC Driverによって、SQL Server EXEC文の構文が対応するOracleプロシージャ・コールに変換されます。

SQL Serverプロシージャの完全な名前は、次に示す最大4つの識別子で構成されます。

- サーバー名
- データベース名
- 所有者名
- プロシージャ名

この名前の書式は次のとおりです。

```
[[[server.][database.][owner_name].]procedure_name
```

SQL ServerデータベースからOracleへの移行時、それぞれのSQL Serverプロシージャ(またはファンクション)定義は、対応するOracle構文に変換され、Oracleスキーマに定義されます。移行したプロシージャは、多くの場合、次のいずれかの方法で再編成(およびスキーマで作成)されます。

- すべてのプロシージャが1つのスキーマに移行されます(デフォルト・オプション)。
- 1つのSQL Serverデータベースに定義されているすべてのプロシージャが、そのデータベース名の付いたスキーマに移行されます。
- 特定のユーザーが所有するすべてのプロシージャが、そのユーザー名の付いたスキーマに移行されます。

移行したプロシージャを編成するこれら3通りの方法をサポートするためには、いずれかのスキーマ名オプションを指定してプロシージャ名を変換できます。変換したOracleプロシージャ・コールのオブジェクト名では、大/小文字が区別されません。

24.4.9 Oracle RAC環境における接続障害のイベント通知の有効化

Oracle Real Application Clusters (Oracle RAC)データベース環境で接続障害が発生したときに、SQLSetConnectAttr関数のSQL_ORCLATTR_FAILOVER_CALLBACKおよびSQL_ORCLATTR_FAILOVER_HANDLE属性が設定されている場合、イベント通知が有効になります。どちらの属性も、SQLSetConnectAttr関数を使用して設定されます。新規属性のシンボルは、ファイルsqora.hで定義されています。

SQL_ORCLATTR_FAILOVER_CALLBACK属性は、障害イベントが発生したときにコールするルーチンのアドレスを指定します。

SQL_ORCLATTR_FAILOVER_HANDLE属性は、コールバック・ルーチンのパラメータとして渡されるコンテキスト・ハンドルを指定します。この属性は、ODBCアプリケーションで障害イベントが発生している接続を判別するために必要です。

コールバック・ルーチンの関数プロトタイプは次のとおりです。

```
void failover_callback(void *handle, SQLINTEGER fo_code)
```

'handle'パラメータは、SQL_ORCLATTR_FAILOVER_HANDLE属性により設定された値です。この属性が設定されていない場合、NULLが戻されます。

fo_codeパラメータにより、発生している障害イベントが識別されます。障害イベントは、OCIプログラミング・インタフェースで定義されているイベントに直接マップされます。発生する可能性があるイベントのリストを次に示します。

- ODBC_FO_BEGIN
- ODBC_FO_ERROR
- ODBC_FO_ABORT
- ODBC_FO_REAUTH
- ODBC_FO_END

この機能の使用方法を示したサンプル・プログラムは次のとおりです。

```
/*
NAME
ODBCCallbackTest

DESCRIPTION
Simple program to demonstrate the connection failover callback feature.

PUBLIC FUNCTION(S)
main

PRIVATE FUNCTION(S)

NOTES

Command Line: ODBCCallbackTest filename [odbc-driver]
*/

#include <windows.h>
#include <tchar.h>
#include <malloc.h>
#include <stdio.h>
#include <string.h>
#include <sql.h>
```

```

#include <sqlext.h>
#include "sqora.h"

/*
** Function Prototypes
*/
void display_errors(SQLSMALLINT HandleType, SQLHANDLE Handle);
void failover_callback(void *Handle, SQLINTEGER fo_code);

/*
** Macros
*/
#define ODBC_STS_CHECK(sts) \
    if (sts != SQL_SUCCESS) \
    { \
        display_errors(SQL_HANDLE_ENV, hEnv); \
        display_errors(SQL_HANDLE_DBC, hDbc); \
        display_errors(SQL_HANDLE_STMT, hStmt); \
        return FALSE; \
    }

/*
** ODBC Handles
*/
SQLHENV *hEnv = NULL; // ODBC Environment Handle
SQLHANDLE *hDbc = NULL; // ODBC Connection Handle
SQLHANDLE *hStmt = NULL; // ODBC Statement Handle

/*
** Connection Information
*/
TCHAR *dsn = _T("odbctest");
TCHAR *uid = _T("scott");
TCHAR *pwd = _T("tiger");
TCHAR *szSelect = _T("select * from emp");

/*
** MAIN Routine
*/
main(int argc, char **argv)
{
    SQLRETURN rc;

    /*
    ** Allocate handles
    */
    rc = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, (SQLHANDLE *)&hEnv);
    ODBC_STS_CHECK(rc);

    rc = SQLSetEnvAttr(hEnv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER)SQL_OV_ODBC3, 0);
    ODBC_STS_CHECK(rc);

    rc = SQLAllocHandle(SQL_HANDLE_DBC, hEnv, (SQLHANDLE *)&hDbc);
    ODBC_STS_CHECK(rc);

    /*
    ** Connect to the database
    */
    rc = SQLConnect(hDbc, dsn, (SQLSMALLINT)_tcslen(dsn),
        uid, (SQLSMALLINT)_tcslen(uid),
        pwd, (SQLSMALLINT)_tcslen(pwd));
    ODBC_STS_CHECK(rc);

    /*
    ** Set the connection failover attributes
    */

```

```

    rc = SQLSetConnectAttr(hDbc, SQL_ORCLATTR_FAILOVER_CALLBACK, &failover_callback,
0);
    ODBC_STS_CHECK(rc);

    rc = SQLSetConnectAttr(hDbc, SQL_ORCLATTR_FAILOVER_HANDLE, hDbc, 0);
    ODBC_STS_CHECK(rc);

    /*
    ** Allocate the statement handle
    */
    rc = SQLAllocHandle(SQL_HANDLE_STMT, hDbc, (SQLHANDLE *)&hStmt);
    ODBC_STS_CHECK(rc);

    /*
    ** Wait for connection failovers
    */
    while (TRUE)
    {
        Sleep(5000);
        rc = SQLExecDirect(hStmt, szSelect, _tcslen(szSelect));
        ODBC_STS_CHECK(rc);

        rc = SQLFreeStmt(hStmt, SQL_CLOSE);
        ODBC_STS_CHECK(rc);
    }

    /*
    ** Free up the handles and close the connection
    */
    rc = SQLFreeHandle(SQL_HANDLE_STMT, hStmt);
    ODBC_STS_CHECK(rc);

    rc = SQLDisconnect(hDbc);
    ODBC_STS_CHECK(rc);

    rc = SQLFreeHandle(SQL_HANDLE_DBC, hDbc);
    ODBC_STS_CHECK(rc);

    rc = SQLFreeHandle(SQL_HANDLE_ENV, hEnv);
    ODBC_STS_CHECK(rc);

    return TRUE;
}

/*
** Failover Callback Routine
*/
void failover_callback(void *Handle, SQLINTEGER fo_code)
{
    switch(fo_code)
    {
        {
        case ODBC_FO_BEGIN:
            printf("ODBC_FO_BEGIN received\n");
            break;

        case ODBC_FO_ERROR:
            printf("ODBC_FO_ERROR received\n");
            break;

        case ODBC_FO_ABORT:
            printf("ODBC_FO_ABORT received\n");
            break;

        case ODBC_FO_REAUTH:
            printf("ODBC_FO_REAUTH received\n");
            break;
        }
    }
}

```



```

case ODBC_FO_END:
    printf("ODBC_FO_END received\n");
    break;

default:
    printf("Invalid or unknown ODBC failover code received\n");
    break;
}
return;
}

/*
** Retrieve the errors associated with the handle passed
** and display them.
*/
void display_errors(SQLSMALLINT HandleType, SQLHANDLE Handle)
{
    SQLTCHAR MessageText[256];
    SQLTCHAR SqlState[5+1];
    SQLSMALLINT i=1;
    SQLINTEGER NativeError;
    SQLSMALLINT TextLength;
    SQLRETURN sts = SQL_SUCCESS;

    if (Handle == NULL) return;

    /* Make sure all SQLState text is null terminated */
    SqlState[5] = '\0';

    /*
    ** Fetch and display all diagnostic records that exist for this handle
    */
    while (sts == SQL_SUCCESS)
    {
        NativeError = 0;
        TextLength = 0;
        sts = SQLGetDiagRec(HandleType, Handle, i, SqlState, &NativeError, (SQLTCHAR
*)&MessageText, sizeof(MessageText), &TextLength);
        if (sts == SQL_SUCCESS)
        {
            printf("[%s]%s\n", SqlState, MessageText);
            if (NativeError != 0)
                printf("Native Error Code: %d\n", NativeError);
            i++;
        }
    }
    return;
}

```

24.4.10 ODBCを通じた暗黙的結果機能の使用

このオプションは、サード・パーティODBCアプリケーションをOracleデータベースに移行し、前のベンダーでサポートされている暗黙的結果機能を使用する場合に使用します。Oracle ODBC Driverでは、ストアード・プロシージャまたは匿名PL/SQLブロックでの暗黙的結果がサポートされます。現在のリリースでは、暗黙的結果はSELECT文でのみ戻されます。

次のコード例は、暗黙的結果に匿名SQLスクリプトを使用したODBCテスト・ケースの例を示しています。

```

const char *query1="declare ¥
                    c1 sys_refcursor; ¥
                    c2 sys_refcursor; ¥
                    begin ¥
                    open c1 for select empno,ename from emp where rownum<=3; ¥
                    dbms_sql.return_result(c1); ¥

```

```

        open c2 for select empno,ename from emp where rownum<=3; ¥
        dbms_sql.return_result(c2); end; ";

int main( )
{
    ...
    //Allocate all required handles and establish a connection to the database.

    //Prepare and execute the above anonymous PL/SQL block
    SQLPrepare (hstmt, (SQLCHAR *) query1, SQL_NTS);
    SQLExecute(hstmt);

    //Bind the columns for the results from the first SELECT statement in an anonymous
    block.
    SQLBindCol (hstmt, 1, SQL_C_ULONG, &eno, 0, &jind);
    SQLBindCol (hstmt, 2, SQL_C_CHAR, empname, sizeof (empname),&enind);

    //Fetch implicit results through the SQLFetch( ) call.
    while((retCode = SQLFetch(hstmt)) != SQL_NO_DATA)
    {
    //Do whatever you want to do with the data.
    }

    retCode = SQLMoreResults(hstmt);

    if(retCode == SQL_SUCCESS)
    {
        printf("SQLMoreResults returned with SQL_SUCCESS¥n");
    }

    //Bind the columns for the results from the second SELECT statement in an anonymous
    block.
    SQLBindCol (hstmt, 1, SQL_C_ULONG, &eno, 0, &jind);
    SQLBindCol (hstmt, 2, SQL_C_CHAR, empname, sizeof (empname),&enind);

    //Fetch implicit results through the SQLFetch( ) call.
    while((retCode = SQLFetch(hstmt)) != SQL_NO_DATA)
    {
    //Do whatever you want to do with data.
    }
}
}

```

24.4.11 ODBCでのOracle TIMESTAMP WITH TIME ZONEおよび TIMESTAMP WITH LOCAL TIME ZONE列タイプのサポートについて

タイムゾーンはシステム変数ORA_SDTZに定められています。システム変数を 'OS_TZ'、'DB_TZ' または有効なタイム・ゾーン値に設定できます。ORA_SDTZを 'OS_TZ' に設定すると、オペレーティング・システムのタイムゾーンが使用されます。

'DB_TZ' に設定した場合は、データベースに設定されたデフォルトのタイムゾーンが使用されます。

ORA_SDTZが設定されていない場合は、デフォルトでオペレーティング・システムのタイムゾーンが使用されます。

ノート:



Microsoft Windows 環境のレジストリ、システム環境変数またはコマンド・プロンプト・ウィンドウで ORA_SDTZ 変数を設定する場合、タイム・ゾーン値を引用符で囲まないでください。

関連項目:

日時データ型およびタイム・ゾーン・サポートの詳細は、[Oracle Databaseグローバル化バージョン・サポート・ガイド](#)を参照

ODBCデータ型TIMESTAMP_STRUCTの変数を使用したこれらのタイムゾーン列からのデータのフェッチ

次の例では、ODBCデータ型TIMESTAMP_STRUCTの変数を使用して、TIMESTAMP WITH TIME ZONE列およびTIMESTAMP WITH LOCAL TIME ZONE列からデータをフェッチする方法を示します。

例24-1 ODBCデータ型TIMESTAMP_STRUCTの変数を使用してTIMESTAMP WITH TIME ZONE列およびTIMESTAMP WITH LOCAL TIME ZONE列からデータをフェッチする方法

```
int main()
{
...
...
/* TSTAB table's DDL statement:
 * -----
 * CREATE TABLE TSTAB (COL_TSTZ TIMESTAMP WITH TIME ZONE,
 * COL_TSLTZ TIMESTAMP WITH LOCAL TIME ZONE);
 *
 * Insert statement:
 * -----
 * Sample #1:
 * -----
 * INSERT INTO TSTAB VALUES (TIMESTAMP '2010-03-13 03:47:30.123456
America/Los_Angeles'
 * TIMESTAMP '2010-04-14 04:47:30.123456 America/Los_Angeles');
 *
 * Sample #2:
 * -----
 * INSERT INTO TSTAB VALUES ('22-NOV-1963 12:30:00.000000 PM',
 * '24-NOV-1974 02:30:00.000000 PM');
 *
 * Refer Oracle Database documentations to know more details about TIMESTAMP
 * WITH TIME ZONE and TIMESTAMP WITH LOCAL TIME ZONE columns.
 */
SQLCHAR sqlSelQuery[] = "SELECT COL_TSTZ, COL_TSLTZ FROM TSTAB";
TIMESTAMP_STRUCT timestampcol1;
TIMESTAMP_STRUCT timestampcol2;
...
...
/* Allocate the ODBC statement handle. */
SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
/* Execute the statement sqlSelQuery. */
SQLExecDirect(hstmt, sqlSelQuery, SQL_NTS);

/* Bind the variable to read the value from the TIMESTAMP WITH TIME ZONE column. */
SQLBindCol(hstmt, 1, SQL_C_TIMESTAMP, &timestampcol1, sizeof(timestampcol1), NULL);

/* Bind the variable to read the value from the TIMESTAMP WITH LOCAL TIME ZONE
column. */
SQLBindCol(hstmt, 2, SQL_C_TIMESTAMP, &timestampcol2, sizeof(timestampcol2), NULL);
...
...
/* Fetch data from the TSTAB table. */
retcode = SQLFetch(hstmt);
/* Values of column COL_TSTZ and COL_TSLTZ are available in variables
 * timestampcol1 and timestampcol2 respectively. Refer to Microsoft ODBC
 * documentation for more information about data type TIMESTAMP_STRUCT. */
...
...
/* Close the statement. */
SQLFreeStmt(hstmt, SQL_CLOSE);
/* Free the statement handle. */
```

```
SQLFreeHandle(SQL_HANDLE_STMT, hstmt); ... } }
```

例24-2 TIMESTAMP WITH TIME ZONE列およびTIMESTAMP WITH LOCAL TIME ZONE列にデータを挿入する方法

```
int main()
{
...
...
SQLCHAR sqlInsQuery[] = "INSERT INTO TSTAB VALUES (?, ?)";
TIMESTAMP_STRUCT timestampcol1;
TIMESTAMP_STRUCT timestampcol2;
...
...
/* Input the value for column COL_TSTZ in table TSTAB. */
timestampcol1.year = 2000;
timestampcol1.month = 1;
timestampcol1.day = 1;
timestampcol1.hour = 0;
timestampcol1.minute = 0;
timestampcol1.second = 1;
timestampcol1.fraction = 1000;

/* Input the value for column COL_TSLTZ in table TSTAB. */
timestampcol1.year = 2012;
timestampcol1.month = 2;
timestampcol1.day = 5;
timestampcol1.hour = 10;
timestampcol1.minute = 30;
timestampcol1.second = 10;
timestampcol1.fraction = 1000;
...
...
/* Allocate the ODBC statement handle. */
SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
...
...
/* Bind the input value for column COL_TSTZ. */
SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_TIMESTAMP, SQL_TIMESTAMP,
0, 0, &timestampcol1, sizeof(timestampcol1), NULL);

/* Bind the input value for column COL_TSLTZ. */
SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_TIMESTAMP, SQL_TIMESTAMP,
0, 0, &timestampcol2, sizeof(timestampcol2), NULL);
...
...
/* Execute the statement sqlInsQuery. */
SQLExecDirect(hstmt, sqlInsQuery, SQL_NTS);

/* Close the statement. */
SQLFreeStmt(hstmt, SQL_CLOSE);
/* Free the statement handle. */
SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
...
...
}
```

24.4.12 Oracleクライアント(OCI、SQL*Plus、Oracle ODBC Driverなど)でORA_SDTZを設定した場合の影響について

Oracleクライアントでシステム変数ORA_SDTZを設定した場合の影響について説明します。

タイムゾーンはシステム変数ORA_SDTZに定められています。

次の項では、Oracleクライアント(OCI、SQL*Plus、Oracle ODBC Driverなど)でシステム変数ORA_SDTZを設定した場合や設定なかった場合の影響について説明します。これらの項の例は、インド(GMT+5:30)のタイムゾーンで実行されています。

関連項目:

セッション・タイム・ゾーンの詳細は、『[Oracle Databaseグローバル化ゼーション・サポート・ガイド](#)』を参照してください

環境設定

環境を設定するには、TSLTZ (TIMESTAMP WITH LOCAL TIME ZONE)列のある次の表を作成し、01/01/2016 00:00 GMTの値を次に示すようにTSLTZ列に挿入します。

例24-3 環境の設定方法

次の例では、例を示す後続の項の環境を設定しています。

```
SQL> create table timezone_demo(col1 TIMESTAMP WITH LOCAL TIME ZONE);
Table created.
SQL> INSERT INTO TIMEZONE_DEMO VALUES(TIMESTAMP '2016-01-01 00:00:00.000000
ETC/GREENWICH');
1 row created.
```

環境内でORA_SDTZが設定されない場合

環境内でORA_SDTZが設定されない場合は、オペレーティング・システム(OS)のタイムゾーン設定がOracleクライアントのデフォルトのタイムゾーンとして使用されます。たとえば:

例24-4 ORA_SDTZが設定されない場合

```
C:\Users\example.ORADEV>set ORA_SDTZ=
C:\Users\example.ORADEV>sqlplus scott/password@//host01.example.com:1521/ORCL12C1
SQL*Plus: Release 12.1.0.2.0 Production on Fri Apr 22 12:03:52 2016
Copyright (c) 1982, 2014, Oracle. All rights reserved.
Last Successful login time: Fri Apr 22 2016 11:47:12 +05:30
Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options
SQL> select sessiontimezone from dual;
SESSIONTIMEZONE
-----
+05:30
SQL> select * from timezone_demo;
COL1
-----
01-JAN-16 05.30.00.000000 AM
```

環境内でORA_SDTZをオペレーティング・システム(OS)のタイムゾーンに設定する場合

ORA_SDTZがオペレーティング・システム(OS)・タイム・ゾーンに設定されている場合、Oracleクライアントのユーザー・セッションはOSタイム・ゾーン設定に設定されます。環境で設定を解除するか、ORA_SDTZをOS_TZに設定できます。たとえば:

例24-5 ORA_SDTZがオペレーティング・システム(OS)のタイムゾーンに設定される場合

```
C:\Users\example.ORADEV>set ORA_SDTZ=OS_TZ
C:\Users\example.ORADEV>sqlplus scott/password@//host01.example.com:1521/ORCL12C1
SQL*Plus: Release 12.1.0.2.0 Production on Fri Apr 22 11:42:36 2016
Copyright (c) 1982, 2014, Oracle. All rights reserved.
Last Successful login time: Fri Apr 22 2016 11:42:09 +05:30
Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
```

```

With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options
SQL> select sessiontimezone from dual;
SESSIONTIMEZONE
-----
+05:30
SQL> select * from timezone_demo;
COL1
-----
01-JAN-16 05.30.00.000000 AM

```

環境内でORA_SDTZを特定のタイムゾーンに設定する場合

Oracleクライアントは、特定のタイム・ゾーン(たとえば、ヘルシンキ・タイムゾーン)に調整されたタイム・スタンプ値を取得するように設定できます。これを行うには、次を設定します

ORA_SDTZ

対象: 対応するタイムゾーンのOracleタイムゾーンのリージョン名(ヘルシンキのタイムゾーンのOracleタイムゾーンのリージョン名はEurope/Helsinki)。たとえば:

例24-6 ORA_SDTZが特定のタイムゾーンに設定される場合

```

C:\Users\example.ORADEV>set ORA_SDTZ=Europe/Helsinki
C:\Users\example.ORADEV>sqlplus scott/password@//host01.example.com:1521/ORCL12C1
SQL*Plus: Release 12.1.0.2.0 Production on Fri Apr 22 11:47:10 2016
Copyright (c) 1982, 2014, Oracle. All rights reserved.
Last Successful login time: Fri Apr 22 2016 09:16:18 EUROPE/HELSINKI EEST
Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options
SQL> select sessiontimezone from dual;
SESSIONTIMEZONE
-----
Europe/Helsinki
SQL> select * from timezone_demo;
COL1
-----
01-JAN-16 02.00.00.000000 AM

```

24.4.13 サポートされている機能

トピック:

- [APIへの準拠](#)
- [ODBC API関数の実装](#)
- [ODBC SQL構文の実装](#)
- [データ型の実装\(プログラミング\)](#)

24.4.13.1 APIへの準拠

Oracle ODBC Driverのリリース9.2.0.0.0以降では、すべてのコア、レベル2およびレベル1関数をサポートしています。

また、Oracle ODBC Driverのリリース9.2.0.0.0以降では、変換DLLをサポートしています。

次のトピックでは、Oracle ODBC Driverで実装されているODBC API関数について説明します。

関連項目:

- 上級ユーザーは、[「エラー・メッセージ」](#)を参照してください
- プログラマは、[「ODBC API関数の実装」](#)を参照してください

24.4.13.2 ODBC API関数の実装

次の表は、Oracle ODBC Driverが関数を実装する方法を示したものです。

表24-6 Oracle ODBC Driverで特定の関数を実装する方法

関数	説明:
SQLConnect	SQLConnect には、DBQ、ユーザーID およびパスワードのみが必要です。
SQLDriverConnect	SQLDriverConnect は、DSN、DBQ、UID および PWD キーワードを使用します。
SQLMoreResults	暗黙的結果の ODBC サポートを実装します。これは、Oracle Database 12c リリース 1 (12.1.0.1)に実装されている新しい API です。詳細は、 SQLMoreResults 関数 を参照してください。
SQLSpecialColumns	SQLSpecialColumns は、SQL_BEST_ROWID 属性を指定してコールされると、ROWID 列を戻します。
SQLProcedures および SQLProcedureColumns	次の項を参照してください。
すべてのカタログ関数	SQL_ATTR_METADATA_ID 文属性が SQL_TRUE である場合、文字列引数は識別子引数として扱われ、大/小文字の区別はされません。この場合、アンダースコア(_)およびパーセント記号(%)は検索用パターン文字ではなく、実際の文字として処理されます。この属性が SQL_FALSE である場合、通常の引数またはパターン値の引数となり文字どおりに扱われ、大/小文字は区別されます。

24.4.13.3 ODBC SQL構文の実装

比較述語で比較の2番目の式としてパラメータ・マーカーが使用され、そのパラメータの値がSQLBindParameterを使用してSQL_NULL_DATAに設定されている場合、比較は失敗します。これは、ODBC SQLのNULL述語構文と一致しています。

24.4.13.4 データ型の実装(プログラミング)

プログラマにとって特に注意が必要なのは、CHAR、VARCHARおよびVARCHAR2データ型の実装です。

SQL_VARCHARの値がfSqlTypeの場合、SQLGetTypeInfoはOracle Databaseデータ型VARCHAR2を返します。また、SQLGetTypeInfoは、SQL_CHARのfSqlType値に対しては、Oracleデータベース・データ型であるCHARを返します。

24.4.14 Unicodeのサポート

トピック:

- [ODBC環境内でのUnicodeのサポート](#)

- [ODBC APIでのUnicodeのサポート](#)
- [ドライバ・マネージャのUnicode機能](#)
- [SQLGetDataのパフォーマンス](#)
- [Unicodeの例](#)

24.4.14.1 ODBC環境内のUnicodeサポート

MicrosoftまたはunixODBC ODBCドライバ・マネージャ(ドライバ・マネージャ)によって、Unicodeをサポートしているかどうかに関係なく、すべてのODBCドライバがUnicode対応しているように動作します。これによって、ODBCアプリケーションは、基礎となるODBCドライバのUnicode機能に関係なく記述できます。

Driver ManagerがANSI ODBCドライバに対するUnicodeサポートをエミュレートできる範囲は、Unicodeデータとローカル・コード・ページ間で可能な変換内容によって制限されます。Driver ManagerがデータをUnicodeからローカル・コード・ページに変換する際、データが失われる場合があります。基礎となるODBCドライバがUnicodeをサポートしていないかぎり、Unicodeの完全なサポートは不可能です。Oracle ODBC DriverはUnicodeをフル・サポートしています。

24.4.14.2 ODBC API内でのUnicodeサポート

ODBC APIは、"W"と"A"という決まった接尾辞によって、UnicodeおよびANSIエントリ・ポイントの両方をサポートします。ODBCアプリケーション開発者は、接尾辞を指定してエントリ・ポイントを明示的にコールする必要はありません。UNICODEおよび_UNICODEプリプロセッサ定義を使用してコンパイルされたODBCアプリケーションによって、適切なコールが生成されます。たとえば、SQLPrepareコールはSQLPrepareWとしてコンパイルされます。

Cデータ型のSQL_C_WCHARがODBCインタフェースに追加されたため、アプリケーションでは、入力パラメータをUnicodeとしてエンコードするように指定するか、またはUnicodeとして返された列データを要求できます。マクロのSQL_C_TCHARは、UnicodeおよびANSIの両方で作成する必要があるアプリケーションで有効です。SQL_C_TCHARマクロは、Unicodeアプリケーションの場合はSQL_C_WCHARとして、ANSIアプリケーションの場合はSQL_C_CHARとしてコンパイルされます。

SQLデータ型のSQL_WCHAR、SQL_WVARCHARおよびSQL_WLONGVARCHARがODBCインタフェースに追加され、表内で定義された列がUnicodeで表現されるようになりました。これらの値は、SQLDescribeCol、SQLColAttribute、SQLColumnsおよびSQLProcedureColumnsへのコールから返すことも可能です。

Unicodeエンコードは、SQL列型NCHAR、NVARCHAR2およびNLOBについてサポートされています。また、SQL列型CHARおよびVARCHAR2についても、キャラクタ・セマンティクスが列定義で指定されている場合は、Unicodeエンコードがサポートされます。

Oracle ODBC DriverはこれらのSQL列型をサポートしており、これらをODBC SQLデータ型にマップします。

次の表に、サポートされているSQLデータ型および対応するODBC SQLデータ型を示します。

表24-7 サポートされるSQLデータ型および対応するODBC SQLデータ型

SQLデータ型	ODBC SQLデータ型
CHAR	SQL_CHAR または SQL_WCHAR 脚注 1
VARCHAR2	SQL_VARCHAR または SQL_WVARCHAR 脚注 2

SQLデータ型	ODBC SQLデータ型
NCHAR	SQL_WCHAR
NVARCHAR2	SQL_WVARCHAR
NCLOB	SQL_WLONGVARCHAR

脚注1

文字セマンティクスが列定義で指定されている場合およびデータベースの文字セットがUnicodeの場合、CHARはSQL_WCHARにマップされます。

脚注2

文字セマンティクスが列定義で指定されている場合およびデータベースの文字セットがUnicodeの場合、VARCHAR2はSQL_WVARCHARにマップされます。

24.4.14.3 ドライバ・マネージャのUnicode機能

ドライバ・マネージャは、ODBCドライバがUnicodeをサポートしていないことを検出すると、次を実行します。

- ANSI ODBCドライバをコールする前にUnicode用関数コールをANSI用関数コールに変換します。文字列引数はUnicodeからローカル・コード・ページに変換されます。たとえば、SQLPrepareWコールはSQLPrepareコールに変換されます。SQL文パラメータの文字列はUnicodeからローカル・コード・ページに変換されます。
- 文字データのリターン・パラメータを、ローカル・コード・ページからUnicodeに変換します。たとえば、SQLColAttributeを使用して列名を戻す場合です。
- SQL_C_WCHARでバインドされた列に対し、データをローカル・コード・ページからUnicodeコード・ページに変換します。
- SQL_C_WCHARでバインドされた入力パラメータに対し、データをUnicodeからローカル・コード・ページに変換します。

24.4.14.4 SQLGetDataのパフォーマンス

SQLGetData関数を使用すると、ODBCアプリケーションはデータ型を指定して、データのフェッチ後に列を取得できます。OCIでは、Oracle ODBC Driverがフェッチ前にデータ型を指定する必要があります。OCIの場合、Oracle ODBC Driverは、データベース内で定義されている列のデータ型の情報を使用し、OCI経由で列データをフェッチする最適な方法を決定します。

文字データを含む列がSQLBindColによってバインドされていない場合、Oracle ODBC Driverは、その列をUnicodeとしてフェッチするか、またはローカル・コード・ページとしてフェッチするかを判断する必要があります。ドライバは、デフォルトとして列をUnicodeで受け取ることが可能ですが、結果として不要な変換を2回行うことになります。たとえば、データベースでデータがANSIでエンコードされている場合、Oracle ODBC Driverにデータを送る際にANSIからUnicodeへ変換されるとします。ODBCアプリケーションがSQL_C_CHARとしてデータを要求すると、元のエンコーディングに戻すために、さらに変換が行われることになります。

データのフェッチでは、Oracleクライアントのデフォルトのエンコーディングが使用されます。ODBCアプリケーションでは、WCHARデータ型として列またはパラメータをバインドすることにより、このデフォルトを上書きして、Unicodeとしてデータをフェッチすることができます。

24.4.14.5 Unicodeの例

Oracle ODBC Driver自体がTCHARマクロを使用して実装されているため、ODBCアプリケーション・プログラムでドライバを

活用するためにTCHARを使用することをお勧めします。

次のプログラムではTCHARの使用方法を示します。TCHARはUNICODEおよび_UNICODEを指定してコンパイルする場合は、WCHARデータ型になります。

- [例1 データベースへの接続](#)
- [例2 単純なデータ取出し](#)
- [例3 SQLGetDataを使用したデータ取出し\(フェッチ後のバインド\)](#)
- [例4 単純なデータ更新](#)
- [例5 LONGデータ\(CLOB\)の更新と取出し](#)

例1 データベースへの接続

SQLConnectに対してUnicodeリテラルを使用する以外、他との違いはありません。

```
SQLHENV envHnd;
SQLHDBC conHnd;
SQLHSTMT stmtHnd;
RETCODE rc;
rc = SQL_SUCCESS;
// ENV is allocated
rc = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &envHnd);
// Connection Handle is allocated
rc = SQLAllocHandle(SQL_HANDLE_DBC, envHnd, &conHnd);
rc = SQLConnect(conHnd, _T("stpc19"), SQL_NTS, _T("scott"), SQL_NTS, _T("tiger"),
SQL_NTS);
.
.
.
if (conHnd)
{
    SQLDisconnect(conHnd);
    SQLFreeHandle(SQL_HANDLE_DBC, conHnd);
}
if (envHnd)
    SQLFreeHandle(SQL_HANDLE_ENV, envHnd);
```

例2 単純なデータ取出し

次の例では、従業員名と職種をEMP表から取り出します。すべてのODBC関数にTCHAR準拠のデータを指定する必要があることを除いて、ANSIの場合と違いはありません。Unicodeアプリケーションの場合は、SQLBindColをコールするときにバッファの長さをBYTE長で指定する必要があります(たとえば、sizeof(ename))。

```
/*
** Execute SQL, bind columns, and Fetch.
** Procedure:
**
** SQLExecDirect
** SQLBindCol
** SQLFetch
**
*/
static SQLTCHAR *sqlStmt = _T("SELECT ename, job FROM emp");
SQLTCHAR ename[50];
SQLTCHAR job[50];
SQLINTEGER enamelen, joblen;

_tprintf(_T("Retrieve ENAME and JOB using SQLBindCol 1.../n[%s]/n"), sqlStmt);
/* Step 1: Prepare and Execute */
```

```

rc = SQLExecDirect(stmtHnd, sqlStmt, SQL_NTS); /* select */
checkSQLErr(envHnd, conHnd, stmtHnd, rc);

/* Step 2: Bind Columns */
rc = SQLBindCol(stmtHnd, 1, SQL_C_TCHAR, ename, sizeof(ename), &enamelen);
checkSQLErr(envHnd, conHnd, stmtHnd, rc);

rc = SQLBindCol(stmtHnd, 2, SQL_C_TCHAR, job, sizeof(job), &joblen);
checkSQLErr(envHnd, conHnd, stmtHnd, rc);

do
{
    /* Step 3: Fetch Data */
    rc = SQLFetch(stmtHnd);
    if (rc == SQL_NO_DATA)
        break;
    checkSQLErr(envHnd, conHnd, stmtHnd, rc);
    _tprintf(_T("ENAME = %s, JOB = %s/n"), ename, job);
} while (1);
_tprintf(_T("Finished Retrieval/n/n"));

```

例3 SQLGetDataを使用したデータ取出し(フェッチ後のバインド)

この例では、SQLGetDataの使用方法を示します。ODBCプログラミングに精通していない場合は、OCIプログラムの場合と異なり、SQLGetDataを使用することで、データをバインドする前にフェッチすることが可能です。Unicode固有の事項に関しては、ANSIアプリケーションとの違いはありません。

```

/*
** Execute SQL, bind columns, and Fetch.
** Procedure:
**
** SQLExecDirect
** SQLFetch
** SQLGetData
*/
static SQLTCHAR *sqlStmt = _T("SELECT ename,job FROM emp"); // same as Case 1.
SQLTCHAR ename[50];
SQLTCHAR job[50];

_tprintf(_T("Retrieve ENAME and JOB using SQLGetData.../n[%s]/n"), sqlStmt);
if (rc != SQL_SUCCESS)
{
    _tprintf(_T("Failed to allocate STMT/n"));
    goto exit2;
}

/* Step 1: Prepare and Execute */
rc = SQLExecDirect(stmtHnd, sqlStmt, SQL_NTS); // select
checkSQLErr(envHnd, conHnd, stmtHnd, rc);
do
{
    /* Step 2: Fetch */
    rc = SQLFetch(stmtHnd);
    if (rc == SQL_NO_DATA)
        break;
    checkSQLErr(envHnd, conHnd, stmtHnd, rc);
    /* Step 3: GetData */
    rc = SQLGetData(stmtHnd, 1, SQL_C_TCHAR, (SQLPOINTER)ename, sizeof(ename), NULL);
    checkSQLErr(envHnd, conHnd, stmtHnd, rc);

    rc = SQLGetData(stmtHnd, 2, SQL_C_TCHAR, (SQLPOINTER)job, sizeof(job), NULL);
    checkSQLErr(envHnd, conHnd, stmtHnd, rc);
    _tprintf(_T("ENAME = %s, JOB = %s/n"), ename, job);
} while (1);
_tprintf(_T("Finished Retrieval/n/n"));

```

例4 単純なデータ更新

この例では、データの更新方法を示します。同様に、SQLBindParameterに対するデータ長は、Unicodeアプリケーションの場合でもBYTE長で指定します。

```
/*
 *
 ** Execute SQL, bind columns, and Fetch.
 ** Procedure:
 **
 ** SQLPrepare
 ** SQLBindParameter
 ** SQLExecute
 */
static SQLTCHAR *sqlStmt = _T("INSERT INTO emp(empno,ename,job) VALUES(?,?,?)");
static SQLTCHAR *empno = _T("9876"); // Emp No
static SQLTCHAR *ename = _T("ORACLE"); // Name
static SQLTCHAR *job = _T("PRESIDENT"); // Job

_tprintf(_T("Insert User ORACLE using SQLBindParameter.../n[%s]/n"), sqlStmt);

/* Step 1: Prepare */
rc = SQLPrepare(stmtHnd, sqlStmt, SQL_NTS);
checkSQLErr(envHnd, conHnd, stmtHnd, rc);

/* Step 2: Bind Parameter */
rc = SQLBindParameter(stmtHnd, 1, SQL_PARAM_INPUT, SQL_C_TCHAR, SQL_DECIMAL, 4, 0,
(SQLPOINTER)empno, 0, NULL);
checkSQLErr(envHnd, conHnd, stmtHnd, rc);
rc = SQLBindParameter(stmtHnd, 2, SQL_PARAM_INPUT, SQL_C_TCHAR, SQL_CHAR,
lstrlen(ename)*sizeof(TCHAR), 0, (SQLPOINTER)ename, lstrlen(ename)*sizeof(TCHAR),
NULL);
checkSQLErr(envHnd, conHnd, stmtHnd, rc);

rc = SQLBindParameter(stmtHnd, 3, SQL_PARAM_INPUT, SQL_C_TCHAR, SQL_CHAR,
lstrlen(job)*sizeof(TCHAR), 0, (SQLPOINTER)job, lstrlen(job)*sizeof(TCHAR), NULL);
checkSQLErr(envHnd, conHnd, stmtHnd, rc);

/* Step 3: Execute */
rc = SQLExecute(stmtHnd);
checkSQLErr(envHnd, conHnd, stmtHnd, rc);
```

例5 LONGデータ(CLOB)の更新と取出し

この例は、OracleのCLOBのような大きなデータの更新と取出しをする、最も複雑な場合の例です。データ長はBYTE長である必要があるため、BYTE長を導出するにはlstrlen(TCHAR data)*sizeof(TCHAR)が必要です。

```
/*
 ** Execute SQL, bind columns, and Fetch.
 ** Procedure:
 **
 ** SQLPrepare
 ** SQLBindParameter
 ** SQLExecute
 ** SQLParamData
 ** SQLPutData
 **
 ** SQLExecDirect
 ** SQLFetch
 ** SQLGetData
 */
static SQLTCHAR *sqlStmt1 = _T("INSERT INTO clobtbl(clob1) VALUES(?)");
static SQLTCHAR *sqlStmt2 = _T("SELECT clob1 FROM clobtbl");
SQLTCHAR clobdata[1001];
SQLTCHAR resultdata[1001];
```

```

SQLINTEGER ind = SQL_DATA_AT_EXEC;
SQLTCHAR *bufp;
SQLTCHAR ch;
int clobdatalen, chunksize, dtsize, retchklen, i, len;

_tprintf(_T("Insert CLOB1 using SQLPutData.../n[%s]/n"), sqlStmt1);

/* Set CLOB Data */
for (i=0, ch=_T('A'); i< sizeof(clobdata)/sizeof(SQLTCHAR); ++i, ++ch)
{
    if (ch > _T('Z'))
        ch = _T('A');
    clobdata[i] = ch;
}
clobdata[sizeof(clobdata)/sizeof(SQLTCHAR)-1] = _T('/0');
clobdatalen = lstrlen(clobdata);
chunksize = clobdatalen / 7;

/* Step 1: Prepare */
rc = SQLPrepare(stmtHnd, sqlStmt1, SQL_NTS);
checkSQLErr(envHnd, conHnd, stmtHnd, rc);

/* Step 2: Bind Parameter with SQL_DATA_AT_EXEC */
rc = SQLBindParameter(stmtHnd, 1, SQL_PARAM_INPUT, SQL_C_TCHAR, SQL_LONGVARCHAR,
clobdatalen*sizeof(TCHAR), 0, (SQLPOINTER)clobdata, clobdatalen*sizeof(TCHAR), &ind);
checkSQLErr(envHnd, conHnd, stmtHnd, rc);

/* Step 3: Execute */
rc = SQLExecute(stmtHnd);
checkSQLErr(envHnd, conHnd, stmtHnd, rc);
sdhamoth: Continuation:

/* Step 4: ParamData (initiation) */
rc = SQLParamData(stmtHnd, (SQLPOINTER*)&bufp);
checkSQLErr(envHnd, conHnd, stmtHnd, rc);

for (dtsize=0, bufp = clobdata; dtsize < clobdatalen; dtsize += chunksize, bufp +=
chunksize)
{
    if (dtsize+chunksize<clobdatalen)
        len = chunksize;
    else
        len = clobdatalen-dtsize;

    /* Step 5: PutData */
    rc = SQLPutData(stmtHnd, (SQLPOINTER)bufp, len*sizeof(TCHAR));
    checkSQLErr(envHnd, conHnd, stmtHnd, rc);
}

/* Step 6: ParamData (termination) */
rc = SQLParamData(stmtHnd, (SQLPOINTER*)&bufp);
checkSQLErr(envHnd, conHnd, stmtHnd, rc);

rc = SQLFreeStmt(stmtHnd, SQL_CLOSE);
_tprintf(_T("Finished Update/n/n"));
rc = SQLAllocStmt(conHnd, &stmtHnd);
if (rc != SQL_SUCCESS)
{
    _tprintf(_T("Failed to allocate STMT/n"));
    goto exit2;
}

/* Clear Result Data */
memset(resultdata, 0, sizeof(resultdata));
chunksize = clobdatalen / 15; /* 15 times to put */

/* Step 1: Prepare */

```

```

rc = SQLExecDirect(stmtHnd, sqlStmt2, SQL_NTS); /* select */
checkSQLErr(envHnd, conHnd, stmtHnd, rc);

/* Step 2: Fetch */
rc = SQLFetch(stmtHnd);
checkSQLErr(envHnd, conHnd, stmtHnd, rc);

for(dtsize=0, bufp = resultdata; dtsize < sizeof(resultdata)/sizeof(TCHAR) && rc !=
SQL_NO_DATA; dtsize += chunksize-1, bufp += chunksize-1)
{
    if (dtsize+chunksize<sizeof(resultdata)/sizeof(TCHAR))
        len = chunksize;
    else
        len = sizeof(resultdata)/sizeof(TCHAR)-dtsize;

    /* Step 3: GetData */
    rc = SQLGetData(stmtHnd, 1, SQL_C_TCHAR, (SQLPOINTER)bufp, len*sizeof(TCHAR),
&retchklen);
}
if (!_tcscmp(resultdata, clobdata))
{
    _tprintf(_T("Succeeded!!/n/n"));
}
else
{
    _tprintf(_T("Failed!!/n/n"));
}
}

```

24.4.15 パフォーマンスとチューニング

トピック:

- [一般的なODBCプログラミングのヒント](#)
- [データソース構成オプション](#)
- [DATEおよびTIMESTAMPデータ型](#)

24.4.15.1 一般的なODBCプログラミングのヒント

この項では、ODBCアプリケーションのパフォーマンスを向上させるためのいくつかの一般的なプログラミングのヒントを記述しています。

- アプリケーションがデータソースからの接続および切断を頻繁に繰り返す場合、接続のプーリングを有効にします。プールされた接続の再利用は、接続の再確立に比べて非常に効率的です。
- 文を準備する回数を最小限にします。可能な場合は、バインド・パラメータを使用し、別のパラメータ値に対して文を再利用できるようにします。SQLExecuteごとに文を処理するのに比べ、一度文を処理して数回実行する方がはるかに効果的です。
- アプリケーションで取得しないとわかっている列をSELECT文に含めないでください(特にLONG列)。データベース・サーバー・プロトコルの性質上、アプリケーションが列をバインドする場合、またはSQLGetDataをバインドする場合のいずれの場合も、LONG列がSELECT文に含まれている場合は、ODBCドライバはLONG列全体の内容をフェッチする必要があります。
- データソースを更新しないトランザクションを実行中の場合は、ODBC SQLSetConnectAttr関数のSQL_ATTR_ACCESS_MODE属性をSQL_MODE_READ_ONLYに設定します。
- ODBCのエスケープ句を使用していない場合は、ODBC SQLSetConnectAttr関数またはODBC

SQLSetStmtAttr関数のSQL_ATTR_NOSCAN属性をTRUEに設定します。

- 行数の多い表からデータを取得する場合は、ODBCのSQLFetch関数のかわりにODBCのSQLFetchScroll関数を使用します。
- 同じSQL文が複数回使用される場合には、OCI文キャッシュを有効にします(StatementCache=T)。
- NUMBER列をFLOATとしてバインドすると問合せの実行が高速になります(BindAsFLOAT=T)。
- LONGまたはLONG RAWのフェッチ中に、MaxLargeData=<value>に設定するとパフォーマンスが最適化されます。
- Ref Cursorを戻す小さなパッケージ・プロシージャに対して大量のコールが行われるアプリケーションでUseOCIDescribeAny=Tと設定すると、パフォーマンスが向上します。

24.4.15.2 データソース構成オプション

このトピックでは、次のODBCデータソース構成オプションのパフォーマンスに関連する事項を説明します。

トピック:

- [結果セットの有効化](#)
- [LOBの有効化](#)
- [TIMESTAMPをDATEとしてバインド](#)
- [カーソル・クローズの有効化](#)
- [スレッド・セーフティの有効化](#)
- [フェッチ・バッファ・サイズ](#)

結果セットの有効化

このオプションによって、プロシージャ・コールから結果セット(RefCursorなど)を返すサポートが有効になります。デフォルトでは、結果セットを返すサポートが有効です。

ODBCドライバは、RefCursorパラメータが含まれるかどうかを判断するために、データベース・サーバーへ問い合わせプロシージャのパラメータ・セットおよびそれらのデータ型を判断する必要があります。この問合せによって、最初にプロシージャが準備完了になり実行されると、追加のネットワーク・ラウンド・トリップが発生します。

LOBの有効化

このオプションによって、LOBを挿入および更新するサポートが有効になります。デフォルトは有効です。

ODBC Driverでは、LOBパラメータがあるかどうかを判断するために、データベース・サーバーを問い合わせ、INSERT文またはUPDATE文の各パラメータのデータ型を判別する必要があります。INSERTまたはUPDATE文が最初に処理および実行される際、この問合せが追加のネットワーク・ラウンド・トリップを発生させます。

TIMESTAMPをDATEとしてバインド

SQL_TIMESTAMPパラメータを適切なOracleデータ型でバインドします。このオプションがTRUEである場合、SQL_TIMESTAMPはOracleのDATEデータ型でバインドされます。このオプションがFALSEである場合、SQL_TIMESTAMPはOracleのTIMESTAMPデータ型でバインドされます(デフォルト)。

カーソル・クローズの有効化

ODBC関数SQLFreeStmtのSQL_CLOSEオプションは、関連するカーソルを文とともにクローズし、保留中のすべての結果を廃棄します。アプリケーションは、SQLPrepareなしで文を再実行しカーソルを再開させることができます。このオプションを使用

する典型的な例は、しばらくの間アイドル状態になり、後で同じSQL文を再利用するアプリケーションの場合です。この場合、アプリケーションがアイドル状態になっている間、関連するサーバー・リソースを解放することがあります。

Oracle ODBC Driverが階層化されているOracle Call Interface (OCI)は、カーソル・クローズ機能をサポートしていません。デフォルトでは、SQL_CLOSEオプションはOracle ODBC Driverには影響がありません。カーソルおよび関連のリソースは、データベース・サーバー上にオープンされた状態で存在します。

このオプションを有効にすると、データベース・サーバー上の関連するカーソルがクローズします。ただし、その結果、SQL文の解析コンテキストが失われます。ODBCアプリケーションは、SQLPrepareコールなしで再び文を実行することができます。ただし、実際にはODBCドライバは文を解析してからすべて実行することになります。このオプションを有効にした場合、一度文を解析し、それを繰り返して実行するアプリケーションのパフォーマンスには重大な影響があります。

サーバーの関連リソースを解放することが確実に必要である場合のみ、このオプションを有効にします。

スレッド・セーフティの有効化

アプリケーションがシングル・スレッドの場合、このオプションを無効にできます。デフォルトでは、ODBCドライバは、すべての内部構造(環境、接続、文)へのアクセスがスレッド・セーフティであることを保証しています。シングル・スレッドのアプリケーションは、このオプションを無効にすることにより、スレッド・セーフティのオーバーヘッドの一部を回避できます。このオプションを無効にすることにより、通常は多少のパフォーマンスの改善がみられます。

フェッチ・バッファ・サイズ

[「Oracle ODBCドライバ構成」ダイアログ・ボックスの「Oracle」オプション・タブ](#)の「フェッチ・バッファ・サイズ」をバイト単位で設定します。この値は、ODBC DriverがOracle Databaseからクライアントのキャッシュに1回にプリフェッチするデータの行数を決定し、アプリケーション・プログラムが1回の問合せで要求する行数とは関係ありません。これによって、パフォーマンスが向上します。

通常一度に20行未満をフェッチするアプリケーションでは、低速なネットワーク接続上で動作する場合や非常に負荷の高いサーバーにアクセスする場合には特に、レスポンス時間が向上します。これを大きく設定しすぎると、レスポンス時間が悪化したり、メモリーを大量に消費することがあります。デフォルトは64,000バイトです。アプリケーションに最適な値を選択します。

ノート:



LONG および LOB データ型がある場合、ODBC ドライバのプリフェッチ行数は、フェッチ・バッファ・サイズに依存しません。LONG および LOB データ型では、パフォーマンスの改善が見込めず、過度のメモリー消費を引き起こす可能性があります。LONG および LOB データ型が存在する場合、ODBC Driver はフェッチ・バッファ・サイズを無視し、設定された行数をプリフェッチします。

24.4.15.3 DATEデータ型およびTIMESTAMPデータ型

索引を含むDATE列をWHERE句で使用すると、データベースのパフォーマンスに影響する場合があります。たとえば:

```
SELECT * FROM EMP WHERE HIREDATE = ?
```

この例ではHIREDATE列の索引を使用して、瞬時に問合せを実行できます。ただし、HIREDATEは実際にはDATE値であり、ODBCドライバによりTIMESTAMPのパラメータ値が提供されるため、Oracleサーバーの問合せ最適化には変換機能を適用する必要があります。誤った結果(パラメータ値に0以外の小数秒がある場合に発生する可能性があります)を防ぐために、最適化は、HIREDATE列に変換を適用するため、次のような文になります。

```
SELECT * FROM EMP WHERE TO_TIMESTAMP(HIREDATE) = ?
```

残念ながら、これによりHIREDATE列の索引の使用は無効になり、かわりにサーバーによる表のスキャンが順次実行されます。表に多くの列が含まれる場合、これには時間がかかります。この状況を回避するため、ODBCドライバには「TIMESTAMPをDATEとしてバインド」接続オプションが用意されています。このオプションが有効になっている場合、SQL_TIMESTAMPパラメータはODBCドライバによりOracleのTIMESTAMPデータ型ではなくDATEデータ型でバインドされます。これにより、問合せオプティマイザでDATE列の索引が使用できるようになります。

ノート:



このオプションは、DATE 列を TIMESTAMP 列としてバインドする Microsoft Access や類似のプログラムで使用するを目的としています。実際の TIMESTAMP 列がある場合、またはデータが失われる可能性がある場合は、このオプションを使用しないでください。Microsoft Access では、主キーとして選択されたあらゆる列を使用して、このような問合せが実行されています。

関連項目:

[データ型の実装](#)

25 識別コード・パッケージの使用

識別コード・パッケージは、電子製品コード(EPC)などOracle Databaseの各種製品コードや識別コード間の格納、取出し、エンコード、デコード、および翻訳を行うツールや手法を提供する、Oracle Databaseの機能です。また、データ型、メタデータ表およびビュー、およびEPC標準RFIDタグまたは新型RFIDタグをユーザー表に格納するためのPL/SQLパッケージも提供します。

識別コード・パッケージによって、Oracle Databaseは、EPCコード体系の認識、効率的な記憶域およびコンポーネント・レベルのEPCデータの取得、EPCglobal Tag Data Translation 1.0(TDT)標準に準拠することが可能になります。TDT標準は、各種EPC RFIDタグ表現間のデコード、エンコード、および翻訳の方法について定義したものです。

このパッケージはまた、EPC標準には含まれていない既存のコード体系をアプリケーションで使用するための拡張可能なフレームワークを提供します。このフレームワークにより、Oracle Databaseは古いシステムにも、今後EPC標準に含まれる可能性のある、進化する識別コードにも対応可能となります。

また、識別コード・パッケージでは、最初にエンコード・カテゴリを登録してエンコード・タイプを登録し、次にエンコード・タイプに関連付けられた新しいコンポーネントを登録して、開発者は独自の識別コードを作成できます。

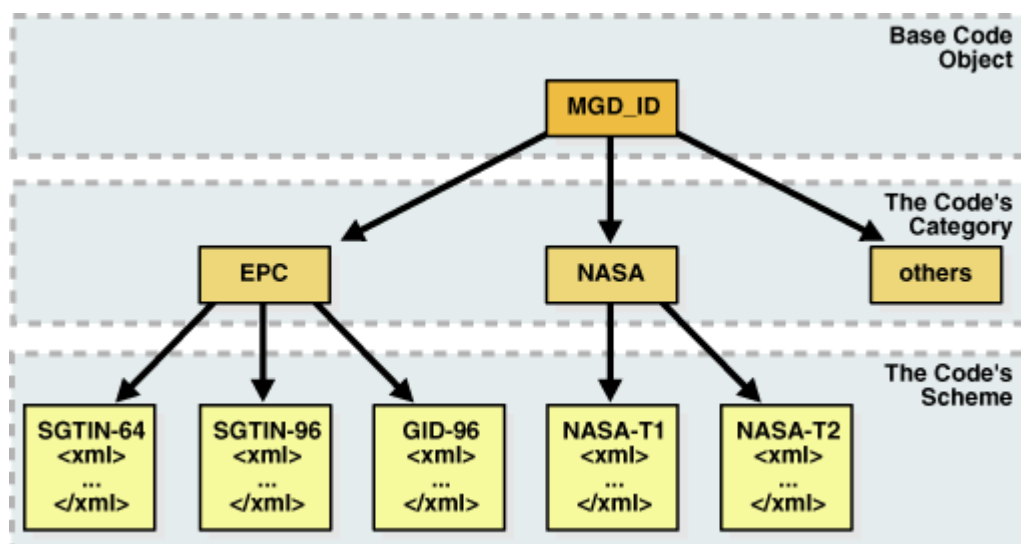
トピック:

- [識別の概念](#)
- [識別コード・パッケージの概要](#)
- [識別コード・パッケージの使用](#)
- [識別コード・パッケージ・タイプ](#)
- [DBMS_MGD_ID_UTLパッケージ](#)
- [識別コード・メタデータ表およびビュー](#)
- [電子製品コード\(EPC\)の概念](#)
- [Oracle Database Tag Data Translationスキーマ](#)

25.1 識別の概念

データベース・オブジェクトMGD_IDが定義され、独自に作成した既存の識別コードとEPC標準の識別コードの使用が可能になりました。MGD_IDオブジェクトは、特定のカテゴリが属するベース・コード・オブジェクトとして、または、EPCカテゴリ、NASAカテゴリなど数多くのカテゴリのRFIDタグの型としての役割を果たします。各カテゴリには、タグ表現構造およびそのコンポーネントを定義する一連のタグ・スキームまたは文書が存在します。EPCカテゴリにおいて、様々なエンコーディングのタイプ(EPC標準バージョン1.1で定義)を表すコード体系(SGTIN-64、SGTIN-96、GID-96など)の定義に必要なメタデータは、デフォルトではデータベースにロードされます。ユーザーは、[図25-1](#)で示すように独自のカテゴリとスキームのエンコーディングの定義が可能であり、データベースにもロードできます。

図25-1 RFIDコード・カテゴリおよびそのスキーム



MGD_IDオブジェクトには、2つの属性、category_id、および名前/値ペアのコンポーネントで構成されるリストが含まれます。MGD_IDオブジェクトが格納されると、オブジェクトの作成時にタグ表現をこれらのコンポーネントの名前/値ペアに解析する必要があります。

EPC標準バージョン1.1は、既存で既知のコード体系に依存しない汎用識別子(GID)タイプを1つ、EAN.UCC仕様に基づくDomain Identifierのタイプを5つ定義します。さらに、米国防総省(USDOD)の識別タイプも定義します。EAN.UCCに基づく識別タイプは、serialized global trade identification number(SGTIN)、serial shipping container code(SSCC)、serialized global location number(SGLN)、global returnable asset identifier(GRAI)、およびglobal individual asset identifier(GIAI)の5つです。

ビットレベルのエンコーディングを1つ持つGID以外の識別タイプはすべて、長さが64ビットか96ビットかによって、それぞれ2つのエンコーディングがあります。合計すると、EPCタグには13種類の異なる標準エンコーディングが存在することになります。また、タグURIおよび純粋な識別表現などバイナリ以外の表現でのエンコーディングも可能です。

各EPCエンコーディングには、独自の構造と編成があります。[表25-1](#)を参照してください。EPCエンコーディング構造フィールド名は、識別コード・パッケージAPIのparameter_listパラメータの名前/値ペアに関連しています。たとえば、SGTIN-64では、構造フィールド名はFilter Value、Company Prefix Index、Item Reference、およびSerial Numberです。

表25-1 EPCエンコーディングの一般的な構造

エンコーディング名	ヘッダー長(ビット)	フィールド名(parameter_listの名前/値ペア)およびフィールド長(ビット)
GID-96	8	General Manager Number (8)、Object Class (24)、Serial Number (36)
SGTIN-64	2	Filter Value (3)、Company Prefix Index (14)、Item Reference (20)、Serial Number (25)
SGTIN-96	8	Filter Value (3)、Partition (3)、Company Prefix (20-40)、Item Reference (24-4)、Serial Number (38)
SSCC-64	8	Filter Value (3)、Company Prefix Index (14)、Serial Reference (39)

エンコーディング名	ヘッダー長(ビット)	フィールド名(parameter_listの名前/値ペア)およびフィールド長(ビット)
SSCC-96	8	Filter Value (3)、Partition (3)、Company Prefix (20-40)、Serial Reference (38-18)、Unallocated (24)
SGLN-64	8	Filter Value (3)、Company Prefix Index (14)、Location Reference (20)、Serial Number (19)
SGLN-96	8	Filter Value (3)、Partition (3)、Company Prefix (20-40)、Location Reference (21-1)、Serial Number (41)
GRAI-64	8	Filter Value (3)、Company Prefix Index (14)、Asset Type (20)、Serial Number (19)
GRAI-96	8	Filter Value (3)、Partition (3)、Company Prefix (20-40)、Asset Type (24-4)、Serial Number (38)
GIAI-64	8	Filter Value (3)、Company Prefix Index (14)、Individual Asset Reference (39)
GIAI-96	8	Filter Value (3)、Partition (3)、Company Prefix (20-40)、Individual Asset Reference (62-42)
USDOD-64	8	Filter Value (2)、Government Managed Identifier (30)、Serial Number (24)
USDOD-96	8	Filter Value (4)、Government Managed Identifier (48)、Serial Number (36)

EPCglobalは11個のタグ・スキーム(GID-96、SGTIN-64、SGTIN-96など)を定義しています。これらのスキームにはそれぞれ多くの表現がありますが、現在最も頻繁に使用されているのはBINARY、TAG_URI、およびPURE_IDENTITYです。たとえば、SGTIN-64の情報は次のように表現できます。

```
BINARY: 10011000000000000001000001110110001000010000011111110011000110010
PURE_IDENTITY: urn:epc:id:sgtin:0037000.030241.1041970
TAG_URI: urn:epc:tag:sgtin-64:3.0037000.030241.1041970
LEGACY: gtin=00037000302414;serial=1041970
ONS_HOSTNAME: 030241.0037000.sgtin.id.example.com
```

これらの表現にはタグに関するすべての情報が含まれているものがありますが(BINARYおよびTAG_URI)、他の表現には一部の情報のみが含まれています(PURE_IDENTITY)。そのため、タグをTAG_URIからPURE_IDENTITY表現へ翻訳することはできますが、フィルタ値などの詳細情報がないかぎり、逆への翻訳は実行できません。

EPCglobalは、様々なEPC RFIDタグ表現間のデコード、エンコードおよび翻訳を定義したTag Data Translation

1.0(TDT)標準をリリースしました。デコーディングは、ある表現をフィールド/値ペアに解析することを意味し、エンコーディングはこれらのフィールドから表現を再構成することを意味します。翻訳とは、ある表現をデコーディングし、すぐに別の表現にエンコーディングすることを意味します。TDTはそれぞれがスキームを表す一連のXMLファイルを使用してこの情報を定義します。たとえば、SGTIN-64スキームは、バイナリと純粋な識別など様々なSGTIN-64表現間でのデコード、エンコードおよび翻訳を定義します。EPCglobal TDTスキームの詳細は、EPCglobal Tag Data Translationの仕様を参照してください。

TDT仕様の主な機能は、同じXMLスキームを使用してすべてのEPCスキームを定義できることです。この方法により、RFIDアプリケーションがパーサー、エンコーダ、およびトランスレータの記述に使用するEPCメタデータを定義する標準的な方法が作成されます。TDT仕様に従ってアプリケーションを記述した場合、そのアプリケーションから一連のEPCタグ・スキームの更新と、メタデータに従ったアクションの変更が可能である必要があります。

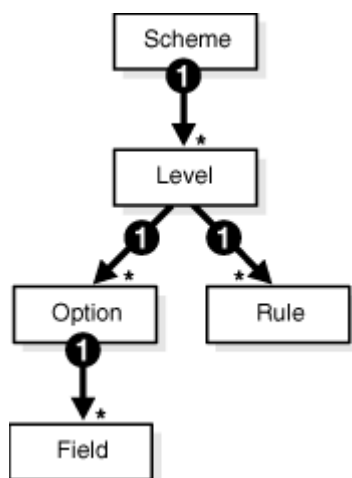
Oracle Databaseメタデータ構造はTDT標準と類似していますが、同一なものではありません。EPCglobal TDT仕様に合わせるには、Oracle RFIDパッケージですべてのTDT互換スキームを取り込み、定義された汎用のOracle Databaseメタデータにシームレスに翻訳できる必要があります。詳細は、[表25-4](#)のEPC_TO_ORACLE関数を参照してください。

フィールドからのタグ表現の再構築、つまり、タグ・データの事前定義済の表現へのエンコーディングは、MGD_ID.format関数を使用して簡単に実行できます。同様に、タグ表現のMGD_IDオブジェクトへのデコーディングおよびこれらのオブジェクトのタグ表現へのエンコーディングも、MGDID.translate関数を使用して簡単に実行できます。詳細は、[表25-3](#)のFORMATメンバー関数およびTRANSLATE静的関数を参照してください。

EPCglobal TDT標準は強力な拡張性も高いため、Oracle RFID標準メタデータはTDT仕様と密接に関連しています。開発者は、このOracle Database TDT XMLスキームを参照し、独自のタグ構造を定義できます。

[図25-2](#)に、Oracle Database Tag Data Translationマークアップ言語スキーム図を示します。

図25-2 Oracle Database Tag Data Translationマークアップ言語スキーム図



タグ・データ翻訳XMLのトップレベル要素は「スキーム」です。それぞれのスキームは、様々なタグ・エンコーディング表現またはレベルを定義します。SGTIN-64およびGID-96は、タグ・コード体系の例です。また、BINARYまたはPURE_IDENTITYは、これらのスキーム内のレベルの例です。各レベルには、様々な表現のフィールドへの解析方法を定義した一連のオプション、および追加の作業が必要なフィールドからの値の導出方法を定義したルールがあります。ルールには、外部表参照または解析された他のフィールドとの連結などがあります。詳細は、EPCglobalのタグ翻訳仕様を参照してください。

関連項目:

- EPCの概念の概要は、[「電子製品コード\(EPC\)の概念」](#)を参照してください
- 実際のOracle Database TDT XMLスキームは、[「Oracle Database Tag Data Translationスキーム」](#)を参

25.2 識別コード・パッケージの概要

識別コード・パッケージは、現在のRFIDタグとサポートされるエンコーディング・タイプのEPCビット・エンコーディングの標準ファミリ、および現在のEPC標準には含まれていない進化するタグ・エンコーディングの両方をサポートする、拡張可能なフレームワークを提供します。

識別コード・パッケージは次のADTを定義します。

- MGD_ID -- 次の項目を定義します(詳細は、[表25-2](#)の MGD_ID ADTを参照してください)。
 - category_idおよびcomponentsの2つの属性。
 - RFIDタグを表現する識別コード・タイプ・オブジェクトを構成するための4つのMGD_IDコンストラクタ関数。
 - これらのADTを操作する一連のメンバー・サブプログラム。

[「識別コード・パッケージの使用」](#)は、これらのADTとメンバー関数の使用方法について説明しています。

[「識別コード・パッケージ・タイプ」](#)と[「DBMS_MGD_ID_UTLパッケージ」](#)では、これらのADTと一連のユーティリティ・サブプログラムの参照情報を簡単に説明しています。

- MGD_ID_COMPONENT — コンポーネント名を識別するcomp_nameと、コンポーネント値を識別するcomp_valueの2つの属性を定義します。
- MGD_ID_COMPONENT_VARRAY — MGD_IDCOMPONENT型の要素を最大128個格納できる配列型を定義します。この配列型は、コンポーネントのリストがある識別コード・タイプ・オブジェクトを作成するための2つのコンストラクタ関数で使います。

識別コード・パッケージは、事前定義済のEPC_ENCODING_CATEGORY encoding_category属性定義とサポートされるエンコーディング・タイプのビット・エンコーディング構造を提供することで、EPC仕様バージョン1.1をサポートします。この情報は、提供されるエンコーディング・メタデータ・ビューMGD_USR_ID_CATEGORYとMGD_USR_ID_SCHEME、読み取り専用ビューMGD_ID_CATEGORYとMGD_ID_SCHEME、およびその基礎となる表MGD_ID_CATEGORY_TAB、MGD_ID_SCHEME_TABとMGD_ID_XML_VALIDATORでメタ情報として格納されます。詳細は、次の項およびファイルを参照してください。

- [「電子製品コード\(EPC\)の概念」](#)は、EPC仕様バージョン1.1の製品コードおよびコード体系ファミリについて説明しています。
- [「識別コード・メタデータ表およびビュー」](#)は、識別コード・メタ表とビューの構造、および様々なRFIDタグを解釈するために識別コード・パッケージがメタデータをどのように使用するかについて説明しています。
- mgdmeta.sqlファイルは、EPC_ENCODING_CATEGORYカテゴリおよびそれぞれの特定のコード体系のメタ表データを示します。

何千ものRFIDタグをユーザー表のMGD_ID列型の列に格納した後に、この列に索引を作成することで、問合せのパフォーマンスが向上します。詳細は、次の項を参照してください。

- [「MGD_ID列型のメンバー関数を使用したファンクション索引の作成」](#)は、MGD_ID ADTのメンバー関数を使用したファンクション索引またはビットマップ・ファンクション索引の作成について説明しています。

識別コード・パッケージは様々なユーティリティ・サブプログラムから構成されるユーティリティ・パッケージを提供します。詳細は、次の項を参照してください。

- [「識別コード・パッケージ・タイプ」](#)および[「DBMS_MGD_ID_UTLパッケージ」](#)は、各メンバー・サブプログラムについて説明しています。プロキシ情報の設定および削除には、プロキシ・ユーティリティを使用します。メタデータ・ユーティリティは、カテゴリIDの取得、カテゴリのタグ・スキームのリフレッシュ、カテゴリのタグ・スキームの削除、およびタグ・スキームの検証に使用できます。変換ユーティリティは、標準のEPCglobal Tag Data Translation(TDT)のOracle Database TDTファイルへの翻訳に使用します。

識別コード・パッケージは拡張可能であり、新規または進化するRFIDタグ固有の識別コード・タイプを独自に作成できます。エンコーディング・タイプに応じて、識別コード・タイプ、category_id属性値、およびコンポーネント構造が定義可能です。詳細は、次の項を参照してください。

- [「識別コードの新規カテゴリの作成」](#)では、独自の識別コードの作成方法について説明しています。その手順では、まずエンコーディング・カテゴリを登録し、その後にエンコーディング・カテゴリに関連するスキームを登録します。
- [「識別コード・メタデータ表およびビュー」](#)は、識別コード・メタ表とビューの構造、およびメタデータ表とビューにメタ情報を格納して情報を登録する方法について説明しています。

関連項目:

参照情報の詳細は、[『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。

25.3 識別コード・パッケージの使用

トピック:

- [MGD_ID ADTを使用したOracle DatabaseへのRFIDタグの格納](#)
- [MGD_ID列型のメンバー関数を使用したファンクション索引の構築](#)
- [MGD_ID ADT関数の使用](#)
- [識別コードのカテゴリの定義と既存のカテゴリへのコード体系の追加](#)

25.3.1 MGD_ID ADTを使用したOracle DatabaseへのRFIDタグの格納

トピック:

- [MGD_ID列型を使用した表の作成と、列へのEPCタグ・エンコーディングの格納](#)
- [RFIDタグを表現するMGD_IDオブジェクトの構成](#)
- [データベース表へのMGD_IDオブジェクトの挿入](#)
- [MGD_ID列型の問合せ](#)

25.3.1.1 MGD_ID列型を使用した表の作成と、列へのEPCタグ・エンコーディングの格納

RFIDタグを表現する列型としてMGD_IDを使用し、表を作成できます。たとえば:

例1.MGD_ID列型の使用

```
CREATE TABLE Warehouse_info (
    Code          MGD_ID,
    Arrival_time  TIMESTAMP,
    Location      VARCHAR2(256);
    ...);
```

SQL*Plusコマンド:

```
describe warehouse_info;
```

結果:

Name	Null?	Type
CODE	NOT NULL	MGDSYS.MGD_ID
ARRIVAL_TIME		TIMESTAMP(6)
LOCATION		VARCHAR2(256)

25.3.1.2 RFIDタグを表現するMGD_IDオブジェクトの構成

MGD_IDオブジェクトを構成するには、複数の方法があります。

- [MGD_IDオブジェクト\(SGTIN-64\)の構成、およびカテゴリIDとコンポーネントのリストの受渡し](#)
- [MGD_IDオブジェクト\(SGTIN-64\)の構成、およびカテゴリID、タグ識別子と追加の必須パラメータのリストの受渡し](#)
- [MGD_IDオブジェクト\(SGTIN-64\)の構成、およびカテゴリ名、カテゴリ・バージョン\(NULLの場合、最新のバージョンを使用\)とコンポーネントのリストの受渡し](#)
- [MGD_IDオブジェクト\(SGTIN-64\)の構成、およびカテゴリ名、カテゴリ・バージョン、タグ識別子と追加の必須パラメータのリストの受渡し](#)

25.3.1.2.1 MGD_IDオブジェクト(SGTIN-64)の構成、およびカテゴリIDとコンポーネントのリストの受渡し

RFIDタグがEPC標準に準拠している場合、そのカテゴリIDとコンポーネントのリストを使用してMGD_IDオブジェクトを作成できます。たとえば:

```
call DBMS_MGD_ID_UTL.set_proxy('example.com', '80');
call DBMS_MGD_ID_UTL.refresh_category('1');
select MGD_ID ('1',
               MGD_ID_COMPONENT_VARRAY(
                 MGD_ID_COMPONENT('companyprefix', '0037000'),
                 MGD_ID_COMPONENT('itemref', '030241'),
                 MGD_ID_COMPONENT('serial', '1041970'),
                 MGD_ID_COMPONENT('schemes', 'SGTIN-64')
               )
       ) from DUAL;
call DBMS_MGD_ID_UTL.remove_proxy();
@constructor11.sql
.
.
.
MGD_ID ('1', MGD_ID_COMPONENT_VARRAY
         (MGD_ID_COMPONENT('companyprefix', '0037000'),
          MGD_ID_COMPONENT('itemref', '030241'),
          MGD_ID_COMPONENT('serial', '1041970'),
          MGD_ID_COMPONENT('schemes', 'SGTIN-64')))
.
.
.
```

25.3.1.2.2 MGD_IDオブジェクト(SGTIN-64)の構成、およびカテゴリID、タグ識別子と追加の必須パラメータのリストの受渡し

MGD_IDオブジェクトの作成に追加の必須パラメータのリストが存在する場合、このコンストラクタを使用します。たとえば:

```
call DBMS_MGD_ID_UTL.set_proxy('example.com', '80');
call DBMS_MGD_ID_UTL.refresh_category('1');
```

```

select MGD_ID('1',
            'urn:epc:id:sgtin:00370000.030241.1041970',
            'filter=3;scheme=SGTIN-64') from DUAL;
call DBMS_MGD_ID_UTL.remove_proxy();
@constructor22.sql
.
.
.
MGD_ID('1', MGD_ID_COMPONENT_VARRAY(MGD_ID_COMPONENT('filter', '3'),
    MGD_ID_COMPONENT('schemes', 'SGTIN-64'),
    MGD_ID_COMPONENT('companyprefixlength', '7'),
    MGD_ID_COMPONENT('companyprefix', '00370000'),
    MGD_ID_COMPONENT('scheme', 'SGTIN-64'),
    MGD_ID_COMPONENT('serial', '1041970'),
    MGD_ID_COMPONENT('itemref', '030241'))))
.
.
.

```

25.3.1.2.3 MGD_IDオブジェクト(SGTIN-64)の構成、およびカテゴリ名、カテゴリ・バージョン(NULLの場合、最新のバージョンを使用)とコンポーネントのリストの受渡し

カテゴリIDとコンポーネントのリストとともにカテゴリ・バージョンを指定する必要がある場合、このコンストラクタを使用します。たとえば:

```

call DBMS_MGD_ID_UTL.set_proxy('example.com', '80');
call DBMS_MGD_ID_UTL.refresh_category
    (DBMS_MGD_ID_UTL.get_category_id('EPC', NULL));
select MGD_ID('EPC', NULL,
            MGD_ID_COMPONENT_VARRAY(
                MGD_ID_COMPONENT('companyprefix', '00370000'),
                MGD_ID_COMPONENT('itemref', '030241'),
                MGD_ID_COMPONENT('serial', '1041970'),
                MGD_ID_COMPONENT('schemes', 'SGTIN-64')
            )
        ) from DUAL;
call DBMS_MGD_ID_UTL.remove_proxy();
@constructor33.sql
.
.
.
MGD_ID('1', MGD_ID_COMPONENT_VARRAY
    (MGD_ID_COMPONENT('companyprefix', '00370000'),
      MGD_ID_COMPONENT('itemref', '030241'),
      MGD_ID_COMPONENT('serial', '1041970'),
      MGD_ID_COMPONENT('schemes', 'SGTIN-64')
    )
  )
.
.
.

```

25.3.1.2.4 MGD_IDオブジェクト(SGTIN-64)の構成、およびカテゴリ名、カテゴリ・バージョン、タグ識別子と追加の必須パラメータのリストの受渡し

カテゴリ・バージョンと追加のパラメータのリストが必要な場合、このコンストラクタを使用します。

```

call DBMS_MGD_ID_UTL.set_proxy('example.com', '80');
call DBMS_MGD_ID_UTL.refresh_category
    (DBMS_MGD_ID_UTL.get_category_id('EPC', NULL));
select MGD_ID('EPC', NULL,
            'urn:epc:id:sgtin:00370000.030241.1041970',
            'filter=3;scheme=SGTIN-64') from DUAL;
call DBMS_MGD_ID_UTL.remove_proxy();

```

```
@constructor44.sql
.
.
.
MGD_ID('1', MGD_ID_COMPONENT_VARRAY
  (MGD_ID_COMPONENT('filter', '3'),
   MGD_ID_COMPONENT('schemes', 'SGTIN-64'),
   MGD_ID_COMPONENT('companyprefixlength', '7'),
   MGD_ID_COMPONENT('companyprefix', '0037000'),
   MGD_ID_COMPONENT('scheme', 'SGTIN-64'),
   MGD_ID_COMPONENT('serial', '1041970'),
   MGD_ID_COMPONENT('itemref', '030241')
  )
)
.
.
.
```

25.3.1.3 データベース表へのMGD_IDオブジェクトの挿入

次の例は、表に各MGD_IDオブジェクトと追加の列値を挿入し、WAREHOUSE_INFO表を移入する方法を示しています。

```
call DBMS_MGD_ID_UTL.set_proxy('example.com', '80');
call DBMS_MGD_ID_UTL.refresh_category
  (DBMS_MGD_ID_UTL.get_category_id('EPC', NULL));
INSERT INTO WAREHOUSE_INFO (code, arrival_time, location)
  values (MGDSYS.MGD_ID ('EPC',
                        NULL,
                        'urn:epc:id:sgtin:0037000.030241.1041970',
                        null
                        ),
          SYSDATE,
          'SHELF_123');
INSERT INTO WAREHOUSE_INFO (code, arrival_time, location)
  values (MGDSYS.MGD_ID ('EPC',
                        NULL,
                        'urn:epc:id:sgtin:0037000.053021.1012353',
                        null
                        ),
          SYSDATE,
          'SHELF_456');
INSERT INTO WAREHOUSE_INFO (code, arrival_time, location)
  values (MGDSYS.MGD_ID ('EPC',
                        NULL,
                        'urn:epc:id:sgtin:0037000.020140.10174832',
                        null
                        ),
          SYSDATE,
          'SHELF_1034');
COMMIT;
call DBMS_MGD_ID_UTL.remove_proxy();
```

25.3.1.4 MGD_ID列型の問合せ

MGD_ID列型を問い合わせるには、3つの方法があります。

- MGD_ID列型を問い合わせます。アイテム参照が030241であるすべてのアイテムを検索します。

```
SELECT location, wi.code.get_component('itemref') as itemref,
       wi.code.get_component('serial') as serial
FROM warehouse_info wi WHERE wi.code.get_component('itemref') = '030241';
```

LOCATION	ITEMREF	SERIAL
SHELF_123	030241	1041970

- MGD_ID ADTのメンバー関数を使用して問い合わせます。表内のすべてのRFIDタグの純粋な識別表現を選択します。

```
SELECT wi.code.format(null, 'PURE_IDENTITY')
       as PURE_IDENTITY FROM warehouse_info wi;
PURE_IDENTITY
-----
urn:epc:id:sgtin:0037000.030241.1041970
urn:epc:id:gid:0037000.053021.1012353
urn:epc:id:sgtin:0037000.020140.10174832
```

詳細は、[「MGD_IDオブジェクトとget_component関数の使用」](#)を参照してください。メンバー関数のリストは[表25-3](#)を参照してください。

25.3.2 MGD_ID列型のメンバー関数を使用したファンクション索引の構築

get_componentメンバー関数またはそのバリエーションの簡易関数を使用するファンクション索引を作成し、RFIDタグの特定のコンポーネントの問合せのパフォーマンスを改善できます。たとえば：

```
CREATE INDEX warehouseinfo_idx2
on warehouse_info(code.get_component('itemref'));
```

また、get_componentメンバー関数またはそのバリエーションの簡易関数を使用するビットマップ・ファンクション索引を作成し、RFIDタグの特定のコンポーネントの問合せのパフォーマンスも改善できます。たとえば：

```
CREATE BITMAP INDEX warehouseinfo_idx3
on warehouse_info(code.get_component('serial'));
```

25.3.3 MGD_ID ADT関数の使用

MGD_ID ADTには、次のADTで動作するメンバー・サブプログラムが含まれます。詳細は、[表25-2](#)のMGD_ID_COMPONENT、MGD_ID_COMPONENT_VARRAYおよびMGD_ID ADTの参照情報を参照してください。MGD_ID ADTの定義およびそのメンバー・サブプログラムについては、mgdtyp.sqlファイルを参照してください。

トピック：

- [MGD_IDオブジェクトとget_component関数の使用](#)
- [標準的な表現からのタグ・データの解析](#)
- [フィールドからのタグ表現の再構成](#)
- [タグ表現間の翻訳](#)

25.3.3.1 MGD_IDオブジェクトとget_component関数の使用

get_component関数の定義は、次のとおりです。

```
MEMBER FUNCTION get_component(component_name IN VARCHAR2)
RETURN VARCHAR2 DETERMINISTIC,
```

識別コードの各コンポーネントには名前が付けられています。この名前は、コード・タイプを登録する際に定義されます。

get_component関数は、コンポーネントの名前であるcomponent_nameをパラメータとして取り、識別コードを解析するためにメタデータ表内に登録されたメタデータを使用し、component_nameの名前とコンポーネントを戻します。

get_component関数はSQL問合せで使用できます。たとえば、itemrefという名前のコンポーネントのコーディング済アイテムの現在の場所の検索、つまり、03024のアイテムを参照するすべてのアイテムを検索するとします。コード・タグがitemref

を1つのコンポーネントとしてエンコーディングしているため、次のSQL問合せを使用できます。

```
SELECT location,
       w.code.get_component('itemref') as itemref,
       w.code.get_component('serial')  as serial
FROM   warehouse_info w
       WHERE w.code.get_component('itemref') = '030241';
```

LOCATION	ITEMREF	SERIAL
-----	-----	-----
SHELF_123	030241	1041970

他のメンバー関数のリストは、[表25-3](#)を参照してください。

関連項目:

識別コード・タイプの作成方法の詳細は、[「識別コードのカテゴリの定義と既存のカテゴリへのコード体系の追加」](#)を参照してください。

25.3.3.2 標準的な表現からのタグ・データの解析

RFIDリーダーは、タグに格納されているビット列を読み取ります。タグ・データと、リーダーID、タイムスタンプなどのその他の情報は、まずエッジ・サーバーで処理および正規化され、事前にフィルタリングされます。その後、多くのアプリケーション・シナリオでは、情報は永続的に格納され後で取得される必要があります。[表25-1](#)で示したように、Oracle DatabaseではEPC標準で定義されたコード表現体系が事前に登録されているため、様々なEPCタグのコード構造表現を理解します。これにより、Oracle DatabaseはすべてのEPCコード体系を理解し、様々なタグ表現をフィールドに解析できます。ユーザーは、他のエンコーディング・テクノロジーを使用した識別コードの独自のコーディング構造を登録することもできます。このように、システムは拡張可能です。

[「識別概念」](#)で示したとおり、それぞれのEPCglobalタグ・スキーム(GID-96、SGTIN-64、SGTIN-96など)には様々な表現があります。最も頻繁に使用されるのは、BINARY、TAG_URIおよびPURE_IDENTITYです。

これらの表現にはタグに関するすべての情報が含まれているものがありますが(BINARYおよびTAG_URI)、他の表現には一部の情報のみが含まれています(PURE_IDENTITY)。そのため、タグをTAG_URIからPURE_IDENTITY表現へ翻訳することはできますが、フィルタ値などの詳細情報がないかぎり、逆への翻訳(PURE_IDENTITYからTAG_URIへ)は実行できません。

MGD_IDコンストラクタの1つは、カテゴリ名(EPCなど)、カテゴリ・バージョン、タグ識別子(EPCの場合、以前記述した表現に含まれる識別子である必要があります)、およびタグ表現の解析に必要な追加のパラメータ・リストの4つのフィールドを使用します。たとえば、次のコードはBINARY表現からMGD_IDオブジェクトを作成します。

```
SELECT MGD_ID
      ('EPC',
       null,
       '1001100000000000000010000011101100010000100000111111110011000110010',
       null
      )
  AS NEW_RFID_CODE FROM DUAL;
NEW_RFID_CODE(CATEGORY_ID, COMPONENTS(NAME, VALUE))
```

MGD_ID
('1', MGD_ID_COMPONENT_VARRAY(MGD_ID_COMPONENT('filter', '3'), MGD_ID_COMPONENT('schemes', 'SGTIN-64'), MGD_ID_COMPONENT('companyprefixlength', '7'), MGD_ID_COMPONENT('companyprefix', '0037000'), MGD_ID_COMPONENT('companyprefixindex', '1'), MGD_ID_COMPONENT('serial', '1041970'), MGD_ID_COMPONENT('itemref', '030241')))

たとえば、次のようにフィルタ値を追加し、タグのTAG_URI表現からコールが実行された場合、同一のオブジェクトが作成されます。

```
SELECT MGD_ID ( 'EPC',
               null,
               'urn:epc:tag:sgtin-64:3.00370000.030241.1041970',
               null
             )
  as NEW_RFID_CODE FROM DUAL;
NEW_RFID_CODE(CATEGORY_ID, COMPONENTS(NAME, VALUE))
-----
MGD_ID ( '1',
        MGD_ID_COMPONENT_VARRAY (
          ( MGD_ID_COMPONENT('filter', '3'),
            MGD_ID_COMPONENT('schemes', 'SGTIN-64'),
            MGD_ID_COMPONENT('companyprefixlength', '7'),
            MGD_ID_COMPONENT('companyprefix', '0037000'),
            MGD_ID_COMPONENT('serial', '1041970'),
            MGD_ID_COMPONENT('itemref', '030241')
          )
        )
      )
```

25.3.3.3 フィールドからのタグ表現の再構成

識別コード・パッケージのその他の便利な機能として、タグ・データを事前定義済の表現にエンコーディングできます。たとえば、倉庫からある在庫を小売業者に送る前に、どの在庫を小売業者に送るかを記述した請求書を送る必要があります。請求書は、倉庫から送る純粋な識別URIのリストとなります。WAREHOUSE_INFO表のすべての在庫を送る場合、次の例では必要なURIを構成します。

```
SELECT wi.code.format (null, 'PURE_IDENTITY')
  as PURE_IDENTITY FROM warehouse_info wi;
PURE_IDENTITY
-----
urn:epc:id:sgtin:00370000.030241.1041970
urn:epc:id:gid:00370000.053021.1012353
urn:epc:id:sgtin:00370000.020140.10174832
```

25.3.3.4 タグ表現間の翻訳

識別コード・パッケージは、タグ表現をMGD_IDオブジェクトにデコードし、これらのオブジェクトをタグ表現にエンコードできます。MGD_ID.translate関数を使用して、この2つのステップを1つにまとめることができます。静的な翻訳により、RFIDタグを1つの表現から別の表現に変換できます。たとえば：

```
SELECT MGD_ID.translate ( 'EPC',
                         null,
                         'urn:epc:id:sgtin:00370000.030241.1041970',
                         'filter=3;scheme=SGTIN-64',
                         'BINARY'
                       )
  as BINARY FROM DUAL;
BINARY
-----
100110000000000000001000001110110001000010000011111110011000110010
```

この例では、バイナリ表現には純粋な識別表現より多くの情報が含まれます。特に、バイナリ表現にはフィルタ値も含まれます。この場合、SGTIN-64とSGTIN-96の区別にスキーム値も指定する必要があります。そのため、翻訳コールを正常に実行するためには、ファンクション・コールにおいて欠落したフィルタのパラメータ情報を提供し、スキーム名を指定する必要があります。

25.3.4 識別コードのカテゴリの定義と既存のカテゴリへのコード体系の追加

トピック:

- [識別コードのカテゴリの作成](#)
- [新規作成したカテゴリへの2つのメタデータ・スキームの追加](#)

25.3.4.1 識別コードのカテゴリの作成

EPCglobal TDT標準は強力で拡張性も高いため、Oracle Database RFID標準メタデータはTDT仕様と密接に関連しています。そのため、識別コード・パッケージは拡張可能であり、汎用的なメタデータを使用して独自のカテゴリとタグ構造を作成できます。識別コードのカテゴリを作成するには、DBMS_MGD_ID_UTIL.create_category関数を使用します。

たとえば、CONTRACTOR_TAGとEMPLOYEE_TAGの2つの型のタグがあるMGD_SAMPLE_CATEGORYというカテゴリを作成します。このカテゴリと2つのメタデータ・スキームを使用する会社では、正社員と契約社員に異なるアクセス権限を付与する必要がある場合があります。そのため、セキュリティ・ソフトウェアはRFIDリーダーにおいて2つの権限の違いを迅速に識別できる必要があります。次のスクリプトはMGD_SAMPLE_CATEGORYというカテゴリをカテゴリ・バージョン1.0で作成します。組織名はOracle、URIは<http://www.oracle.com/mgd/sample>となります。例は、[「新規作成したカテゴリへの2つのメタデータ・スキームの追加」](#)を参照してください。

25.3.4.2 新規作成したカテゴリへの2つのメタデータ・スキームの追加

続いて、次のようなCONTRACTOR_TAGメタデータ・スキームを作成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<TagDataTranslation version="0.04" date="2005-04-18T16:05:00Z"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema"
  xmlns="oracle.mgd.idcode">
  <scheme name="CONTRACTOR_TAG" optionKey="1" xmlns="">
    <level type="URI" prefixMatch="mycompany.contractor.">
      <option optionKey="1" pattern="mycompany.contractor.([0-9]*).([0-9]*)"
        grammar="'mycompany.contractor.'" contractorID '.' divisionID">
        <field seq="1" characterSet="[0-9]*" name="contractorID"/>
        <field seq="2" characterSet="[0-9]*" name="divisionID"/>
      </option>
    </level>
    <level type="BINARY" prefixMatch="11">
      <option optionKey="1" pattern="11([01]{7})([01]{6})"
        grammar="'11' contractorID divisionID ">
        <field seq="1" characterSet="[01]*" name="contractorID"/>
        <field seq="2" characterSet="[01]*" name="divisionID"/>
      </option>
    </level>
  </scheme>
</TagDataTranslation>
```

CONTRACTOR_TAGスキームには、2つのエンコーディング・レベル、つまり2つのタグ表現方法が含まれます。最初のレベルはURI、2つ目のレベルはBINARYです。URI表現は、接頭辞「mycompany.contractor.」で始まり、ピリオドで区切られた2つの数値フィールドが続きます。2つのフィールドの名前はcontractorIDとdivisionIDです。オプション・タグのパターン・フィールドはタグURI表現の解析構造を定義します。grammarフィールドはURI表現の再構成方法を定義します。BINARY表現も同じ方法で理解できます。この表現は接頭辞「01」で始まり、同じ2つのフィールドcontractorIDとdivisionIDが続きます。ここでは、それぞれバイナリ形式となります。このXMLメタデータ構造を指定すると、contractorタグはURI表現およびBINARY表現からデコードされ、結果のフィールドをこれらの表現のいずれかに再度エンコードできます。

EMPLOYEE_TAGスキームも同様に定義され、次のようになります。

```
<?xml version="1.0" encoding="UTF-8"?>
<TagDataTranslation version="0.04" date="2005-04-18T16:05:00Z"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema"
    xmlns="oracle.mgd.idcode">
  <scheme name="EMPLOYEE_TAG" optionKey="1" xmlns="">
    <level type="URI" prefixMatch="mycompany.employee.">
      <option optionKey="1" pattern="mycompany.employee.([0-9]*).([0-9]*)"
        grammar="'mycompany.employee.'" employeeID '.' divisionID">
        <field seq="1" characterSet="[0-9]*" name="employeeID"/>
        <field seq="2" characterSet="[0-9]*" name="divisionID"/>
      </option>
    </level>
    <level type="BINARY" prefixMatch="01">
      <option optionKey="1" pattern="01([01]{7})([01]{6})"
        grammar="'01'" employeeID divisionID ">
        <field seq="1" characterSet="[01]*" name="employeeID"/>
        <field seq="2" characterSet="[01]*" name="divisionID"/>
      </option>
    </level>
  </scheme>
</TagDataTranslation>;
```

前に作成したカテゴリIDにこれらのスキームを追加するには、DBMS_MGD_ID_UTIL.add_scheme関数を使用します。

次のスクリプトは、MGD_SAMPLE_CATEGORYカテゴリの作成、contractorスキームとemployeeスキームのMGD_SAMPLE_CATEGORYカテゴリへの追加、MGD_SAMPLE_CATEGORYスキームの検証、contractorスキームとemployeeスキームのタグ翻訳のテストを実行します。その後、contractorスキームを削除し、contractorスキームのタグ翻訳をテストします。削除したcontractorスキームについては、予期した例外が戻されることを確認します。employeeスキームの翻訳をテストし、予期した値が返されることを確認してからMGD_SAMPLE_CATEGORYカテゴリを削除します。

```
--contents of add_scheme2.sql
SET LINESIZE 160
CALL DBMS_MGD_ID_UTL.set_proxy('example.com', '80');
-----
---CREATE CATEGORY, ADD_SCHEME, REMOVE_SCHEME, REMOVE_CATEGORY-----
-----
DECLARE
  amt          NUMBER;
  buf          VARCHAR2(32767);
  pos          NUMBER;
  tdt_xml      CLOB;
  validate_tdtxml VARCHAR2(1042);
  category_id  VARCHAR2(256);
BEGIN
  -- remove the testing category if it exists
  DBMS_MGD_ID_UTL.remove_category('MGD_SAMPLE_CATEGORY', '1.0');
  -- create the testing category 'MGD_SAMPLE_CATEGORY', version 1.0
  category_id := DBMS_MGD_ID_UTL.CREATE_CATEGORY('MGD_SAMPLE_CATEGORY', '1.0',
'Oracle',
'http://www.oracle.com/mgd/sample');
  -- add contractor scheme to the category
  DBMS_LOB.CREATETEMPORARY(tdt_xml, true);
  DBMS_LOB.OPEN(tdt_xml, DBMS_LOB.LOB_READWRITE);

  buf := '<?xml version="1.0" encoding="UTF-8"?>
<TagDataTranslation version="0.04" date="2005-04-18T16:05:00Z"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema"
    xmlns="oracle.mgd.idcode">
  <scheme name="CONTRACTOR_TAG" optionKey="1" xmlns="">
    <level type="URI" prefixMatch="mycompany.contractor.">
      <option optionKey="1" pattern="mycompany.contractor.([0-9]*).([0-9]*)"
        grammar="'mycompany.contractor.'" contractorID '.' divisionID">
        <field seq="1" characterSet="[0-9]*" name="contractorID"/>
        <field seq="2" characterSet="[0-9]*" name="divisionID"/>
      </option>
    </level>
  </scheme>
</TagDataTranslation>';
```

```

    </option>
  </level>
  <level type="BINARY" prefixMatch="11">
    <option optionKey="1" pattern="11([01]{7})([01]{6})"
      grammar="'11' contractorID divisionID ">
      <field seq="1" characterSet="[01]*" name="contractorID"/>
      <field seq="2" characterSet="[01]*" name="divisionID"/>
    </option>
  </level>
</scheme>
</TagDataTranslation>';
amt := length(buf);
pos := 1;
DBMS_LOB.WRITE(tdt_xml, amt, pos, buf);
DBMS_LOB.CLOSE(tdt_xml);
DBMS_MGD_ID_UTL.ADD_SCHEME(category_id, tdt_xml);
-- add employee scheme to the category
DBMS_LOB.CREATETEMPORARY(tdt_xml, true);
DBMS_LOB.OPEN(tdt_xml, DBMS_LOB.LOB_READWRITE);

buf := '<?xml version="1.0" encoding="UTF-8"?>
<TagDataTranslation version="0.04" date="2005-04-18T16:05:00Z"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema"
  xmlns="oracle.mgd.idcode">
<scheme name="EMPLOYEE_TAG" optionKey="1" xmlns="">
  <level type="URI" prefixMatch="mycompany.employee.">
    <option optionKey="1" pattern="mycompany.employee.([0-9]*).([0-9]*)"
      grammar="'mycompany.employee.'" employeeID '.' divisionID">
      <field seq="1" characterSet="[0-9]*" name="employeeID"/>
      <field seq="2" characterSet="[0-9]*" name="divisionID"/>
    </option>
  </level>
  <level type="BINARY" prefixMatch="01">
    <option optionKey="1" pattern="01([01]{7})([01]{6})"
      grammar="'01' employeeID divisionID ">
      <field seq="1" characterSet="[01]*" name="employeeID"/>
      <field seq="2" characterSet="[01]*" name="divisionID"/>
    </option>
  </level>
</scheme>
</TagDataTranslation>';
amt := length(buf);
pos := 1;
DBMS_LOB.WRITE(tdt_xml, amt, pos, buf);
DBMS_LOB.CLOSE(tdt_xml);
DBMS_MGD_ID_UTL.ADD_SCHEME(category_id, tdt_xml);
-- validate the scheme
dbms_output.put_line('Validate the MGD_SAMPLE_CATEGORY Scheme');
validate_tdtxml := DBMS_MGD_ID_UTL.validate_scheme(tdt_xml);
dbms_output.put_line(validate_tdtxml);
dbms_output.put_line('Length of scheme xml is: '||DBMS_LOB.GETLENGTH(tdt_xml));
-- test tag translation of contractor scheme
dbms_output.put_line(
  mgd_id.translate('MGD_SAMPLE_CATEGORY', NULL,
    'mycompany.contractor.123.45',
    NULL, 'BINARY'));
dbms_output.put_line(
  mgd_id.translate('MGD_SAMPLE_CATEGORY', NULL,
    '111111011101101',
    NULL, 'URI'));
-- test tag translation of employee scheme
dbms_output.put_line(
  mgd_id.translate('MGD_SAMPLE_CATEGORY', NULL,
    'mycompany.employee.123.45',
    NULL, 'BINARY'));
dbms_output.put_line(
  mgd_id.translate('MGD_SAMPLE_CATEGORY', NULL,

```

```

        '011111011101101',
        NULL, 'URI'));
DBMS_MGD_ID_UTL.REMOVE_SCHEME(category_id, 'CONTRACTOR_TAG');
-- Test tag translation of contractor scheme. Doesn't work any more.
BEGIN
    dbms_output.put_line(
        mgd_id.translate('MGD_SAMPLE_CATEGORY', NULL,
            'mycompany.contractor.123.45',
            NULL, 'BINARY'));
    dbms_output.put_line(
        mgd_id.translate('MGD_SAMPLE_CATEGORY', NULL,
            '111111011101101',
            NULL, 'URI'));
EXCEPTION
    WHEN others THEN
        dbms_output.put_line('Contractor tag translation failed: '||SQLERRM);
END;
-- Test tag translation of employee scheme. Still works.
BEGIN
    dbms_output.put_line(
        mgd_id.translate('MGD_SAMPLE_CATEGORY', NULL,
            'mycompany.employee.123.45',
            NULL, 'BINARY'));
    dbms_output.put_line(
        mgd_id.translate('MGD_SAMPLE_CATEGORY', NULL,
            '011111011101101',
            NULL, 'URI'));
EXCEPTION
    WHEN others THEN
        dbms_output.put_line('Employee tag translation failed: '||SQLERRM);
END;
-- remove the testing category, which also removes all the associated schemes
DBMS_MGD_ID_UTL.remove_category('MGD_SAMPLE_CATEGORY', '1.0');
END;
/
SHOW ERRORS;
call DBMS_MGD_ID_UTL.remove_proxy();
@add_scheme3.sql
.
.
.
Validate the MGD_SAMPLE_CATEGORY Scheme
EMPLOYEE_TAG;URI,BINARY;divisionID,employeeID
Length of scheme xml is: 933
111111011101101
mycompany.contractor.123.45
011111011101101
mycompany.employee.123.45
Contractor tag translation failed: ORA-55203: Tag data translation level not found
ORA-06512: at "MGDSYS.DBMS_MGD_ID_UTL", line 54
ORA-06512: at "MGDSYS.MGD_ID", line 242
ORA-29532: Java call terminated by uncaught Java
exception: oracle.mgd.idcode.exceptions.TDTLevelNotFound: Matching level not
found for any configured scheme
011111011101101
mycompany.employee.123.45
.
.
.

```

25.4 識別コード・パッケージ・タイプ

[表25-2](#)は識別コード・パッケージADTを示します。

表25-2 識別コード・パッケージADT

ADT名	説明:
MGD_ID_COMPONENT ADT	コンポーネントを定義する名前/値ペア属性を指定するデータ型です。
MGD_ID_COMPONENT_VARRAY ADT	データ型を指定します。最大 128 個のコンポーネントのリストを名前/値属性ペアで指定するデータ型で、識別コード・タイプ・オブジェクトを作成するための 2 つのコンストラクタ関数で使われます。
MGD_ID ADT	この識別コードとコンポーネントのリストのコード・カテゴリのカテゴリ識別子を指定する識別コード・タイプを表します。

[表25-3](#)はMGD_ID ADTのサブプログラムを示します。

MGD_ID ADTで定義されたサブプログラムに渡されるすべての値と名前は、特に記載のない限り、大/小文字を区別しません。大/小文字を保持するには、値を二重引用符で囲みます。

表25-3 MGD_ID ADTサブプログラム

サブプログラム	説明:
MGD_ID コンストラクタ関数	識別コード・タイプ・オブジェクト MGD_ID を作成し、それ自体を返します。
FORMAT メンバー関数	MGD_ID コンポーネントを指定した場合の識別コードの表現を返します。
GET_COMPONENT メンバー関数	MGD_ID コンポーネントの値を返します。
TO_STRING メンバー関数	category_id パラメータ値とコンポーネントの名前/値属性ペアを連結します。
TRANSLATE 静的関数	識別コードの 1 つの MGD_ID 表現を別の MGD_ID 表現に翻訳します。

25.5 DBMS_MGD_ID_UTLパッケージ

[表25-4](#)は DBMS_MGD_ID_UTLパッケージのユーティリティ・プログラムを示します。

MGD_ID ADTで定義されたサブプログラムに渡されるすべての値と名前は、特に記載のない限り、大/小文字を区別しません。大/小文字を保持するには、値を二重引用符で囲みます。

表25-4 DBMS_MGD_ID_UTLパッケージ・ユーティリティ・サブプログラム

サブプログラム	説明:

サブプログラム	説明:
ADD_SCHEME プロシージャ	既存のカテゴリにタグ・データ翻訳スキームを追加します。
CREATE_CATEGORY 関数	カテゴリまたはカテゴリのバージョンを作成します。
EPC_TO_ORACLE 関数	EPCglobal Tag Data Translation(TDT)XML を Oracle Database Tag Data Translation XML に変換します。
GET_CATEGORY_ID 関数	カテゴリ名とカテゴリ・バージョンを指定した場合、カテゴリ ID を戻します。
GET_COMPONENTS 関数	指定したスキームのすべての関連するコンポーネント名をセミコロン(;)で区切って戻します。
GET_ENCODINGS 関数	指定したスキームのエンコーディング(フォーマット)のリストをセミコロン(;)で区切って戻します。
GET_JAVA_LOGGING_LEVEL 関数	現在の Java のトレース・ロギング・レベルを表す整数を戻します。
GET_PLSQL_LOGGING_LEVEL 関数	現在の PL/SQL のトレース・ロギング・レベルを表す整数を戻します。
GET_SCHEME_NAMES 関数	指定したカテゴリのスキーム名のリストをセミコロン(;)で区切って戻します。
GET_TDT_XML 関数	指定したスキームの Oracle Database Tag Data Translation XML を戻します。
GET_VALIDATOR 関数	Oracle Database Tag Data Translation スキームを戻します。
REFRESH_CATEGORY 関数	指定したカテゴリの Java スタックに関するメタデータ情報をリフレッシュします。
REMOVE_CATEORY 関数	関連するすべての TDT XML を含むカテゴリを削除します。
REMOVE_PROXY プロシージャ	プロキシ・サーバーのホストとポートの設定を解除します。
REMOVE_SCHEME プロシージャ	カテゴリのタグ・スキームを削除します。
SET_JAVA_LOGGING_LEVEL プロシージャ	Java のロギング・レベルを設定します。

[SET_PLSQL_LOGGING_LEVEL](#) プロシージャ PL/SQL のトレース・ロギング・レベルを設定します。

[SET_PROXY](#) プロシージャ

インターネット・アクセス用に、プロキシ・サーバーのホストとポートを設定します。

[VALIDATE_SCHEME](#) 関数

入力タグ・データ翻訳 XML を Oracle Database Tag Data Translation スキーマで検証します。

25.6 識別コード・メタデータ表およびビュー

この項では、識別コード・メタデータ表とビュー、および様々なRFIDタグを解釈するために識別コード・パッケージによってメタデータがどのように使用されるかについて説明します。これらのメタ表ビュー、およびトリガーの作成は、識別コード・パッケージのインストール時に自動的に実行されます。

エンコーディング・メタデータ・ビューはエンコーディング・カテゴリとスキームの格納に使用します。アプリケーション開発者は、自身の識別コードのメタ情報をこれらのビューに挿入できます。MGD_ID ADTは、エンコーディングのメタデータがメタ表に格納されているかぎり、そのエンコーディングを解釈するように設計されています。アプリケーション開発者がEPC仕様バージョン1.1で定義されたエンコーディングのみを使用する場合、EPC仕様バージョン1.1で指定された製品コードは事前定義済のため、メタ表について考慮する必要はありません。

エンコーディング・メタデータ・ビューには次の2種類があります。

- `user_mgd_id_category`では、セッション・ユーザーが定義したエンコーディング・カテゴリ情報が格納されます。
- `user_mgd_id_scheme`では、セッション・ユーザーが定義したエンコーディング・タイプ情報が格納されます。

システムで事前定義済のエンコーディング・メタデータとユーザー定義のメタデータを確認するために、次の読取り専用ビューを問い合わせることができます。

- `mgd_id_category`では、システムまたはセッション・ユーザーが定義したエンコーディング・カテゴリ情報を問い合わせることができます。
- `mgd_id_scheme`では、システムまたはセッション・ユーザーが定義したエンコーディング・タイプ情報を問い合わせることができます。

前述のビューの基礎となるメタデータ表は、次のとおりです。

- `mgd_id_xml_validator`
- `mgd_id_category_tab`
- `mgd_id_scheme_tab`

識別コード・パッケージのシステム・ユーザー以外は、この表を操作できません。ユーザーは、メタデータ表を直接使用できません。ユーザーは、読取り専用ビューとDBMS_MGD_ID_UTLパッケージに記述されたメタデータ関数を使用する必要があります。

関連項目:

DBMS_MGD_ID_UTLパッケージの詳細は、[『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』](#)を参照してください。

ださい。

メタデータ・ビュー定義

[表25-5](#)、[表25-6](#)、[表25-7](#) および[表25-8](#) は、mgdview.sqlファイルで定義したとおり、それぞれ MGD_ID_CATEGORY、USER_ID_CATEGORY、MGD_ID_SCHEMEおよびUSER_MGD_ID_SCHEMEのメタデータ・ビュー定義を示します。

表25-5 MGD_ID_CATEGORYメタデータ・ビューの定義と説明

列名	データ型	説明:
CATEGORY_ID	NUMBER(4)	カテゴリ識別子
CATEGORY_NAME	VARCHAR2(256)	カテゴリ名
AGENCY	VARCHAR2(256)	カテゴリを定義した組織
VERSION	VARCHAR2(256)	カテゴリ・バージョン
URI	VARCHAR2(256)	カテゴリを示す URI

表25-6 USER_MGD_ID_CATEGORYメタデータ・ビューの定義と説明

列名	データ型	説明:
CATEGORY_ID	NUMBER(4)	カテゴリ識別子
CATEGORY_NAME	VARCHAR2(256)	カテゴリ名
AGENCY	VARCHAR2(256)	カテゴリを定義した組織
VERSION	VARCHAR2(256)	カテゴリ・バージョン
URI	VARCHAR2(256)	カテゴリを示す URI

表25-7 MGD_ID_SCHEMEメタデータ・ビューの定義と説明

列名	データ型	説明:
CATEGORY_ID	NUMBER(4)	カテゴリ識別子
TYPE_NAME	VARCHAR2(256)	SGTIN-96、GID-96 などのコード体系名
TDT_XML	CLOB	このコード体系のタグ・データ翻訳 XML

列名	データ型	説明:
ENCODINGS	VARCHAR2(256)	カンマで区切られたエンコーディング。たとえば、SGTIN-96 の場合は LEGACY, TAG_ENCODING, PURE_IDENTITY, BINARY。
COMPONENTS	VARCHAR2(1024)	各レベルから抽出し組み合わせた関連コンポーネント名。それぞれカンマで区切られます。たとえば、GID-96 の場合は objectclass, generalmanager, serial。

表25-8 USER_MGD_ID_SCHEMEメタデータ・ビューの定義と説明

列名	データ型	説明:
CATEGORY_ID	NUMBER(4)	カテゴリ識別子
TYPE_NAME	VARCHAR2(256)	SGTIN-96、GID-96 などのコード体系名
TDT_XML	CLOB	このコード体系のタグ・データ翻訳 XML
ENCODINGS	VARCHAR2(256)	カンマで区切られたエンコーディング。たとえば、SGTIN-96 の場合は LEGACY, TAG_ENCODING, PURE_IDENTITY, BINARY。
COMPONENTS	VARCHAR2(1024)	各レベルから抽出し組み合わせた関連コンポーネント名。それぞれカンマで区切られます。たとえば、GID-96 の場合は objectclass, generalmanager, serial。

25.7 電子製品コード(EPC)の概念

トピック:

- [RFIDテクノロジーとEPCバージョン1.1コード体系](#)
- [製品コードの概念と現在の使用](#)

25.7.1 RFIDテクノロジーとEPCバージョン1.1コード体系

Radio Frequency Identification(RFID)テクノロジーは、直列処理が不要となり、重要なサプライ・チェーン・トランザクションの自動化が可能であるため、供給業者、卸売業者、製造業者、小売業者の間で進化を続けています。電子製品コード(EPC)とは、RFIDタグやその他の手段によりオブジェクトを識別する世界共通の識別スキームであり、新たな基準として幅広い支持を集めています。在庫管理の改善と、サプライ・チェーンの可視性を高めることで、企業における倉庫と配送のコストが削減できます。

標準化されたEPC識別子は、様々な業界のニーズに応えるよう設計されたメタコード体系です。その結果、EPCはすべてのEPC準拠タグで一意的となるような一連のコード体系および手段を表すようになりました。EPCバージョン1.1には、次の固有の

コード体系が含まれます。

- 一般ID(GID)
- EAN.UCC Global Trade Item Number(GTIN)のシリアル化バージョン
- EAN.UCC Serial Shipping Container Code(SSCC)
- EAN.UCC Global Location Number(GLN)
- EAN.UCC Global Returnable Asset Identifier(GRAI)
- EAN.UCC Global Individual Asset Identifier(GIAI)

RFIDアプリケーションは、大量のEPCデータをデータベースに格納する必要があります。EPCデータを効率的に使用するには、データベース側でEPCデータの異なるコード体系を認識する必要もあります。

EPCは新たな基準です。そのため、様々な業界で使用されている番号付けのスキームをすべて網羅しておらず、EPCバージョン1.0からEPCバージョン1.1への変更からも明らかであるように、EPC自体が現在も進化しています。

識別コード・パッケージにより、Oracle DatabaseでEPCコード体系を認識できるようになります。これにより、Oracle DatabaseにおいてEPCデータの効率的な記憶域およびコンポーネント・レベルの取得が可能になるのみではなく、EPCデータのエンコーディングとデコーディング、およびビット・エンコーディングとURIエンコーディング間の変換をサポートする機能を持つようになります。

識別コード・パッケージは、開発者がEPC標準に含まれていない自分のコード体系を定義する拡張可能なフレームワークを提供します。この拡張機能により、Oracle Databaseは進化する将来のEPC標準にも対応可能となります。

この章では、EPCを含む様々な製品コードのOracle Databaseでの格納、取得、エンコーディングおよびデコーディングの要件について説明し、識別コード・パッケージ・ソリューションがデータ型、メタデータ表およびPL/SQLパッケージを提供することで、これらの要件をどのように満たしているかについて説明します。

25.7.2 製品コードの概念と現在の使用

この項では、次の製品コードについて説明します。

- [電子製品コード\(EPC\)](#)
- [Global Trade Identification Number \(GTIN\)およびSerializable Global Trade Identification Number \(SGTIN\)](#)
- [Serial Shipping Container Code\(SSCC\)](#)
- [Global Location Number \(GLN\)およびSerializable Global Location Number \(SGLN\)](#)
- [Global Returnable Asset Identifier\(GRAI\)](#)
- [Global Individual Asset Identifier\(GIAI\)](#)
- [RFID EPCネットワーク](#)

25.7.2.1 電子製品コード(EPC)

電子製品コード(tm)(EPC(tm))は、Radio Frequency Identification(RFID)タグおよびその他の手段を使用して物理オブジェクトを世界的に識別するための識別スキームです。標準のEPCデータは、個別のオブジェクトを一意に識別するEPC(またはEPC識別子)およびEPCタグを効果的かつ効率的に読み取る必要がある場合はオプションのフィルタ値で構成されています。この標準のデータに加え、EPCタグの特定のクラスでは、ユーザー定義データも使用できます。

EPC識別子は、既存のコード体系(可能な場合)の対処と、必要に応じたスキームの定義により、様々な業界のニーズをサポートするために設計されたメタコード体系です。これらのコード体系は、特定の業界または業界グループのドメインでオブジェクト識別子を提供していることを示すため、ドメインIDと呼ばれます。これにより、EPCは一連のコード体系(または「ネームスペース」)を表し、すべてのEPC準拠タグで一意的な手段となります。

EPCGlobal EPC Data Standards Version 1.1では、EPCの抽象的な内容と、RFIDタグ、インターネットURIおよびその他の表現の形式における具体的な認識を定義します。EPCバージョン1.1では、特定のコード体系には、汎用識別子(GID)、EAN.UCC Global Trade Item Number(GTIN®)のシリアル化バージョン、EAN.UCC Serial Shipping Container Code(SSCC®)、EAN.UCC Global Location Number(GLN®)、EAN.UCC Global Returnable Asset Identifier(GRAI®)、EAN.UCC Global Individual Asset Identifier(GIAI®)が含まれます。

25.7.2.1.1 EPCの純粋な識別

EPCの純粋な識別とは、特定の物理エンティティまたは論理エンティティに関連付けられた識別であり、RFタグ、バーコードおよびデータベース・フィールドなどの特定のエンコーディング媒体に依存しません。そのため、純粋な識別は、エンティティを識別するために使用される抽象的な名前または数値です。純粋な識別は、特定のエンティティを一意的に識別するために必要な情報のみで構成されます。

25.7.2.1.2 EPCエンコーディング

EPCエンコーディングは、純粋なエンコーディングと、フィルタ値などの詳細情報を、特定の構文(通常は特定のサイズの値フィールドで構成される)に変換するものです。指定した純粋な識別には、バーコード・エンコーディング、様々なタグ・エンコーディング、および様々なURIエンコーディングなどの多数のエンコーディングが含まれている可能性があります。エンコーディングには、識別以外の追加データ(一部のエンコーディングで使用されるフィルタ値)などが含まれている場合もあり、保持する追加データはコード体系により指定します。

たとえば、EAN.UCCシステムで定義されたSerial Shipping Container Code(SSCC)形式は純粋な識別の例です。EPC-SSCC 96ビット形式にエンコーディングされたSSCCはエンコーディングの一例です。

25.7.2.1.3 EPCタグのビットレベルのエンコーディング

タグのEPCエンコーディングはビット文字列であり、階層化された可変長のヘッダーとそれに続く一連の数値フィールドから構成されます。数値フィールドは、ヘッダー値によってその全体の長さ、構造、機能が完全に決定されます。

25.7.2.1.4 EPC識別URI

EPC識別URIは、統一資源識別子(URI)としての純粋な識別の表現です。

25.7.2.1.5 EPCタグのURIエンコーディング

EPCタグのURIエンコーディングは、urn:epc:tag:sgtin-64:3.0652642.800031.400など、特定のEPCタグのビットレベル・エンコーディングを表現したものです。

25.7.2.1.6 EPCエンコーディング・プロシージャ

EPCエンコーディング・プロシージャは、様々な情報からEPCタグのビットレベル・エンコーディングを生成します。

25.7.2.1.7 EPCデコーディング・プロシージャ

EPCデコーディング・プロシージャは、EPCタグのビットレベル・エンコーディングをEAN.UCCコードに変換します。

25.7.2.2 Global Trade Identification Number(GTIN)およびSerializable Global Trade Identification Number(SGTIN)

Global Trade Identification Number(GTIN)は、EAN.UCCシステム内で、世界規模で取引品目を一意に識別する場合に使用します。Serialized Global Trade Identification Number(SGTIN)はEPC標準バージョン1.1での識別型です。General EAN.UCC Specifications(GenSpec5.0)で定義されているEAN.UCC GTINコードに基づいています。GTINは特定の製品やSKUなど、特定のオブジェクトのクラスを識別します。GTINと一意のシリアル番号を組み合わせたものはシリアル化GTIN(SGTIN)と呼ばれます。

25.7.2.3 Serial Shipping Container Code(SSCC)

Serial Shipping Container Code(SSCC)はGeneral EAN.UCC Specifications(GenSpec5.0)で定義されています。EAN.UCCシステム内の物流単位の一意的識別にはSSCCを使用します。SSCCは個別のオブジェクトへの割当てを想定しています。

25.7.2.4 Global Location Number(GLN)およびSerializable Global Location Number(SGLN)

Global Location Number(GLN)はGeneral EAN.UCC Specifications(GenSpec5.0)で定義されています。GLNは、ドックのドアや倉庫のスロットなどの個別の一意的物理的位置、または倉庫全体などの物理的位置の集合を表現できます。また、GLNは業務(発注など)を行う組織のような論理的エンティティも表現できます。GLNと一意のシリアル番号を組み合わせたものはシリアル化GLN(SGLN)と呼ばれます。ただし、EAN.UCCコミュニティがGLN拡張の適切な方法を決定するまでは、シリアル番号フィールドは予約済となり使用できません。

25.7.2.5 Global Returnable Asset Identifier(GRAI)

特定の値の輸送装置または再利用可能なパッケージを、リターンブル資産といいます。Global Returnable Asset Identifier(GRAI)は、リターンブル資産を一意的に識別するためにGeneral EAN.UCC Specifications(GenSpec5.0)で定義されています。

25.7.2.6 Global Individual Asset Identifier(GIAI)

Global Individual Asset Identifier(GIAI)はGeneral EAN.UCC Specifications(GenSpec5.0)で定義されています。GTINとは異なり、GIAIは個別のオブジェクトへの割当てを想定しています。Global Individual Asset Identifier(GIAI)は、会社の固定在庫の一部であるエンティティを一意的に識別します。GIAIは、組織のすべての固定資産を識別します。

25.7.2.7 RFID EPCネットワーク

RFID EPCネットワークは、資産を識別、追跡および検索します。物理オブジェクトは、一意のRFIDが有効となっているEPCで識別します。

25.8 Oracle Database Tag Data Translationスキーマ

Oracle Database Tag Data TranslationスキーマはEPCglobal TDTスキーマに類似していますが、同一なものではありません。Oracle Database TDTの内容は次のとおりです。

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="oracle.mgd.idcode"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tdt="oracle.mgd.idcode" elementFormDefault="qualified"
  attributeFormDefault="unqualified" version="1.0">
```

```

<xsd:simpleType name="InputFormatList">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="BINARY"/>
    <xsd:enumeration value="STRING"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="LevelTypeList">
  <xsd:restriction base="xsd:string">
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="SchemeNameList">
  <xsd:restriction base="xsd:string">
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="ModeList">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="EXTRACT"/>
    <xsd:enumeration value="FORMAT"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="CompactionMethodList">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="32-bit"/>
    <xsd:enumeration value="16-bit"/>
    <xsd:enumeration value="8-bit"/>
    <xsd:enumeration value="7-bit"/>
    <xsd:enumeration value="6-bit"/>
    <xsd:enumeration value="5-bit"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="PadDirectionList">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="LEFT"/>
    <xsd:enumeration value="RIGHT"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="Field">
  <xsd:attribute name="seq" type="xsd:integer" use="required"/>
  <xsd:attribute name="name" type="xsd:string" use="required"/>
  <xsd:attribute name="bitLength" type="xsd:integer"/>
  <xsd:attribute name="characterSet" type="xsd:string" use="required"/>
  <xsd:attribute name="compaction" type="tdt:CompactionMethodList"/>
  <xsd:attribute name="compression" type="xsd:string"/>
  <xsd:attribute name="padChar" type="xsd:string"/>
  <xsd:attribute name="padDir" type="tdt:PadDirectionList"/>
  <xsd:attribute name="decimalMinimum" type="xsd:long"/>
  <xsd:attribute name="decimalMaximum" type="xsd:long"/>
  <xsd:attribute name="length" type="xsd:integer"/>
</xsd:complexType>
<xsd:complexType name="Option">
  <xsd:sequence>
    <xsd:element name="field" type="tdt:Field" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="optionKey" type="xsd:string" use="required"/>
  <xsd:attribute name="pattern" type="xsd:string"/>
  <xsd:attribute name="grammar" type="xsd:string" use="required"/>
</xsd:complexType>
<xsd:complexType name="Rule">
  <xsd:attribute name="type" type="tdt:ModeList" use="required"/>
  <xsd:attribute name="inputFormat" type="tdt:InputFormatList" use="required"/>
  <xsd:attribute name="seq" type="xsd:integer" use="required"/>
  <xsd:attribute name="newFieldName" type="xsd:string" use="required"/>
  <xsd:attribute name="characterSet" type="xsd:string" use="required"/>
  <xsd:attribute name="padChar" type="xsd:string"/>
  <xsd:attribute name="padDir" type="tdt:PadDirectionList"/>
  <xsd:attribute name="decimalMinimum" type="xsd:long"/>
  <xsd:attribute name="decimalMaximum" type="xsd:long"/>

```



```

<xsd:attribute name="length" type="xsd:string"/>
<xsd:attribute name="function" type="xsd:string" use="required"/>
<xsd:attribute name="tableURI" type="xsd:string"/>
<xsd:attribute name="tableParams" type="xsd:string"/>
<xsd:attribute name="tableXPath" type="xsd:string"/>
<xsd:attribute name="tableSQL" type="xsd:string"/>
</xsd:complexType>
<xsd:complexType name="Level">
  <xsd:sequence>
    <xsd:element name="option" type="tdt:Option" minOccurs="1"
      maxOccurs="unbounded"/>
    <xsd:element name="rule" type="tdt:Rule" minOccurs="0"
      maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="type" type="tdt:LevelTypeList" use="required"/>
  <xsd:attribute name="prefixMatch" type="xsd:string"/>
  <xsd:attribute name="requiredParsingParameters" type="xsd:string"/>
  <xsd:attribute name="requiredFormattingParameters" type="xsd:string"/>
</xsd:complexType>
<xsd:complexType name="Scheme">
  <xsd:sequence>
    <xsd:element name="level" type="tdt:Level" minOccurs="4" maxOccurs="5"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="tdt:SchemeNameList" use="required"/>
  <xsd:attribute name="optionKey" type="xsd:string" use="required"/>
</xsd:complexType>
<xsd:complexType name="TagDataTranslation">
  <xsd:sequence>
    <xsd:element name="scheme" type="tdt:Scheme" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="version" type="xsd:string" use="required"/>
  <xsd:attribute name="date" type="xsd:dateTime" use="required"/>
</xsd:complexType>
<xsd:element name="TagDataTranslation" type="tdt:TagDataTranslation"/>
</xsd:schema>

```

26 スキーマ・オブジェクトの依存性の理解

オブジェクトAの定義でオブジェクトBを参照している場合、AはBに依存しています。この章では、スキーマ・オブジェクトの相互間の依存性と、Oracle Databaseがそのような依存性を自動的に追跡し管理する方法について説明します。こうした自動依存性管理により、AによってBの古いバージョンが使用されることがなく、Bの変更後にAを明示的に再コンパイルする必要もほとんどなくなります。

トピック:

- [スキーマ・オブジェクトの依存性の概要](#)
- [オブジェクトの依存性の問合せ](#)
- [オブジェクトのステータス](#)
- [依存オブジェクトの無効化](#)
- [無効化を回避するためのガイドライン](#)
- [オブジェクトの再有効化](#)
- [スキーマの範囲内での名前解決](#)
- [ローカル依存性の管理](#)
- [リモート依存性の管理](#)
- [リモート・プロシージャ・コール\(RPC\)の依存性管理](#)
- [共有SQLの依存性管理](#)

26.1 スキーマ・オブジェクトの依存性の概要

スキーマ・オブジェクトのタイプによっては、定義の中で他のオブジェクトを参照できるものがあります。たとえば、ビューは表や他のビューを参照する問合せによって定義され、サブプログラムの本体の中には、他のオブジェクトを参照するSQL文を含めることができます。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。

26.1.1 例: 依存オブジェクト・タイプと参照オブジェクト・タイプの表示

[例26-1](#)は、データベース内の依存オブジェクト・タイプと参照オブジェクト・タイプの表示方法(DBAとしてログインしている場合)を示します。

例26-1 依存オブジェクト・タイプと参照オブジェクト・タイプの表示

依存オブジェクト・タイプの表示:

```
SELECT DISTINCT TYPE
FROM DBA_DEPENDENCIES
ORDER BY TYPE;
```

結果:

```
TYPE
-----
DIMENSION
EVALUATION CONTEXT
```

```

FUNCTION
INDEX
INDEXTYPE
JAVA CLASS
JAVA DATA
MATERIALIZED VIEW
OPERATOR
PACKAGE
PACKAGE BODY
PROCEDURE
RULE
RULE SET
SYNONYM
TABLE
TRIGGER
TYPE
TYPE BODY
UNDEFINED
VIEW
XML SCHEMA

22 rows selected.

```

参照オブジェクト・タイプの表示:

```

SELECT DISTINCT REFERENCED_TYPE
FROM DBA_DEPENDENCIES
ORDER BY REFERENCED_TYPE;

```

結果:

```

REFERENCED_TYPE
-----
EVALUATION CONXT
FUNCTION
INDEX
INDEXTYPE
JAVA CLASS
LIBRARY
OPERATOR
PACKAGE
PROCEDURE
SEQUENCE
SYNONYM
TABLE
TYPE
VIEW
XML SCHEMA
14 rows selected.

```

参照オブジェクトの定義を変更した場合、その変更の内容によって、依存オブジェクトがエラーなしに機能し続けることができない場合があります。たとえば、表を削除すると、その表に基づくビューは使用できなくなります。

26.1.2 例：一部の依存オブジェクトを無効にするスキーマ・オブジェクトの変更

一部の依存性のみを無効にするスキーマ・オブジェクトの変更例として、次の例に示す2つのビューについて考えてみます。これらのビューは、HR.EMPLOYEES表に基づいています。

[例26-2](#)は、EMPLOYEES表から、表内のすべての列を選択するSIXFIGURESビューと、EMAIL列を含まないCOMMISSIONEDビューの2つを作成します。この例が示すように、EMAIL列を変更すると、SIXFIGURESは無効化されますが、COMMISSIONEDはされません。

例26-2 一部の依存オブジェクトを無効にするスキーマ・オブジェクトの変更

```
CREATE OR REPLACE VIEW sixfigures AS
SELECT * FROM employees
WHERE salary >= 1000000;
CREATE OR REPLACE VIEW commissioned AS
SELECT first_name, last_name, commission_pct
FROM employees
WHERE commission_pct > 0.00;
```

SQL*Plus書式設定コマンド:

```
COLUMN object_name FORMAT A16
```

問合せ:

```
SELECT object_name, status
FROM user_objects
WHERE object_type = 'VIEW'
ORDER BY object_name;
```

結果:

OBJECT_NAME	STATUS
COMMISSIONED	VALID
EMP_DETAILS_VIEW	VALID
SIXFIGURES	INVALID

3 rows selected.

EMPLOYEES表のEMAIL列を長くする:

```
ALTER TABLE employees MODIFY email VARCHAR2(100);
```

問合せ:

```
SELECT object_name, status
FROM user_objects
WHERE object_type = 'VIEW'
ORDER BY object_name;
```

結果:

OBJECT_NAME	STATUS
COMMISSIONED	VALID
EMP_DETAILS_VIEW	INVALID
SIXFIGURES	VALID

26.1.3 例: 複数のオブジェクトに依存するビュー

ビューは、その問合せ内で参照されているすべてのオブジェクトに依存します。[例26-3](#)のビューは、表employeesおよびdepartmentsによって変わります。

例26-3 複数のオブジェクトに依存するビュー

```
CREATE OR REPLACE VIEW v AS
SELECT last_name, first_name, department_name
FROM employees e, departments d
WHERE e.department_id = d.department_id
ORDER BY last_name;
```

次の点に注意してください。

- CREATE文を実行すると、すべての依存性が自動的に更新されます。
- 動的SQL文では、依存性は作成されません。たとえば次の文を実行しても、tab1に対する依存性は作成されません。

```
EXECUTE IMMEDIATE 'SELECT * FROM tab1'
```

26.2 オブジェクトの依存性の問合せ

静的データ・ディクショナリ・ビューUSER_DEPENDENCIES、ALL_DEPENDENCIESおよびDBA_DEPENDENCIESには、データベース・オブジェクト間の依存性が表示されます。

utldtree.sql SQLスクリプトを実行すると、オブジェクト依存性ツリーに関する情報が表示されるビューDEPTREEと、DEPTREEをソートした印刷用のビューであるIDEPTREEが作成されます。

関連項目:

[DEPTREE](#)、[IDEPTREE](#)および[utldtree.sql](#)スクリプトの詳細は、『Oracle Databaseリファレンス』を参照してください。

26.3 オブジェクトのステータス

各データベース・オブジェクトには、[表26-1](#)に示すステータス値の1つが割り当てられています。

表26-1 データベース・オブジェクトのステータス

ステータス	意味
有効	オブジェクトが、データ・ディクショナリ内の現在の定義を使用して正常にコンパイルされた。
コンパイル・エラー	オブジェクトの最後のコンパイル実行時にエラーが発生した。
無効	参照先のオブジェクトが変更されたため、オブジェクトに無効とマークされている。(無効になるのは依存オブジェクトのみ。)
未認可	参照オブジェクトに対するアクセス権限が取り消された。(権限取消しの対象となるのは依存オブジェクトのみ。)

ノート:



静的データ・ディクショナリ・ビューUSER_OBJECTS、ALL_OBJECTS および DBA_OBJECTS では、「コンパイル・エラー」、「無効」、「権限取消し」はすべて同じものとみなされ、INVALID と表示されます。

26.4 依存オブジェクトの無効化

オブジェクトAがオブジェクトBに、オブジェクトBがオブジェクトCにそれぞれ依存している場合、A、Bはそれぞれ、B、Cに対する直

接依存オブジェクトであり、AはCに対する間接依存オブジェクトです。

直接依存オブジェクトは、それ自体に影響を与える参照オブジェクトへの変更(参照オブジェクトのシグネチャの変更)によってのみ無効になります。

間接依存オブジェクトは、それ自体に影響を与えない参照オブジェクトへの変更によって無効になることがあります。つまり、Cの変更によってBが無効になると、それによってA(およびBの他のすべての直接依存オブジェクトと間接依存オブジェクト)が無効になります。これを連鎖的な無効化と呼びます。

緩やかな無効化では、参照オブジェクトを変更するデータ定義言語(DDL)文によって、そのすべての依存オブジェクトが無効になります。

高度な無効化では、参照オブジェクトを変更するDDL文によって、次の文のいずれかに該当する依存オブジェクトのみが無効になります。

- 依存オブジェクトが、DDL文によって変更された参照オブジェクトの属性に依存している場合。
- 依存オブジェクトのコンパイル済メタデータが、変更された参照オブジェクトに対して正しいものでなくなった場合。

たとえば、ビューvが表tの列c1およびc2を選択している場合、tの列c3のみを変更するDDL文では、vは無効になりません。

DDL文CREATE OR REPLACE objectは、次の条件下では効果がありません。

- objectがPL/SQLオブジェクトで、新しいPL/SQLソース・テキストが既存のPL/SQLソース・テキストと同一であり、objectとともに保存されているPL/SQLコンパイル・パラメータ設定がセッション環境のパラメータ設定と同一である。
- objectはシノニムで、この文はターゲット・オブジェクトを変更しません。

[表26-2](#)の左側の列の操作では高度な無効化が行われますが、右側の列のケースは例外です。右側の列のケース、および[表26-2](#)に示されていないすべての操作では、緩やかな無効化が行われます。

表26-2 高度な無効化が発生する操作

演算子	例外
ALTER TABLE table ADD column	● 依存オブジェクト(ビューを除く)で、table に SELECT * が使用された場合。 ● 依存オブジェクトで、table%rowtype が使用された場合。 ● 依存オブジェクトで、列リストを指定することなく table に対して INSERT が実行された場合。 ● 依存オブジェクトが、結合が含まれる問合せ内の table を参照している場合。 ● 依存オブジェクトが、PL/SQL 変数を参照する問合せ内の table を参照している場合。
ALTER TABLE table {MODIFY RENAME DROP SET UNUSED} column	● 依存オブジェクトが column を直接参照している場合。

演算子	例外
ALTER TABLE table DROP CONSTRAINT not_null_constraint	● 依存オブジェクトで、table に SELECT * が使用された場合。 ● 依存オブジェクトで、table%ROWTYPE が使用された場合。 ● 依存オブジェクトで、列リストを指定することなく table に対して INSERT が実行された場合。 ● 依存オブジェクトが、行全体に依存するトリガーである(定義で列を指定していない)場合。 ● 依存オブジェクトが、削除された列の右側の列に依存するトリガーである場合。
CREATE OR REPLACE VIEW view オンライン表再定義(DBMS_REDEFINITION)	新しい定義と以前の定義の列リストが異なり、次の 1 つ以上に該当する場合。 ● 依存オブジェクトが、ビューまたは表の新しい定義の中で修正または削除された列を参照している場合。 ● 依存オブジェクトで、view%rowtype または table%rowtype が使用されている場合。 ● 依存オブジェクトで、列リストを指定することなくビューまたは表に対して INSERT が実行された場合。 ● ビューの新しい定義に新しい列が追加され、かつ結合が含まれている問合せ内のビューまたは表を依存オブジェクトが参照している場合。 ● ビューの新しい定義に新しい列が追加され、PL/SQL を参照する問合せ内のビューまたは表を依存オブジェクトが参照している場合。 ● 依存オブジェクトが、RELIES ON 句でビューまたは表を参照している場合。
CREATE OR REPLACE SYNONYM synonym	● 新旧の synonym のターゲットが異なり、かつ一方が表ではない場合。

演算子	例外
	● synonym の新旧のターゲットがどちらも表で、かつその 2 つの表の列リストまたは付与権限が異なる場合。 ● synonym の新旧のターゲットがどちらも表で、かつ依存オブジェクトがビューであり、そのビューが参照している列が以前のターゲットの一意索引に関与し、新しいターゲットの一意索引には関与していない場合。
DROP INDEX	● 索引がファンクション索引であり、依存オブジェクトが、行全体に依存するトリガー、またはファンクション索引が作成された後で table に追加された列に依存するトリガーである場合。 ● 索引が一意索引であり、依存オブジェクトがビューであって、かつそのビューが一意索引に関与している列を参照している場合。
CREATE OR REPLACE {PROCEDURE FUNCTION}	コール・シグネチャが変更された場合。コール・シグネチャとは、パラメータ・リスト(パラメータの順序、名前および型)、戻り型、ACCESSIBLE BY 句(ホワイト・リスト)、純粋度^{脚注 1}、決定性、並列度、パイプライン処理および(プロシージャまたはファンクションが C または Java で実装されている場合は)実装の各プロパティを指します。
CREATE OR REPLACE PACKAGE	● ACCESSIBLE BY 句(ホワイト・リスト)が変更された場合。 ● 依存オブジェクトが、削除または名前変更されたパッケージ・アイテムを参照している場合。 ● 依存オブジェクトが参照しているパッケージ・プロシージャまたはパッケージ・ファンクションのコール・シグネチャまたはエントリ・ポイント番号^{脚注 2}が変更された場合。 参照されているプロシージャまたはファンクションにオーバーロードの候補が複数ある場合、それらの候補のコール・シグネチャまたはエントリ・ポイント番号が変更されるか、候補が追加または削除されると、

依存オブジェクトは無効。

- 依存オブジェクトが参照しているパッケージ・カーソルのコール・シグネチャ、行タイプまたはエントリ・ポイント番号が変更された場合。
- 依存オブジェクトが参照しているパッケージ・タイプまたはパッケージ・サブタイプの定義が変更された場合。
- 依存オブジェクトが参照しているパッケージの変数または定数の名前、データ型、初期値、オフセット番号のいずれかが変更された場合。
- パッケージ純粋度¹が変更された場合。

脚注1

純粋度とは、SQL問合せ内部でPL/SQLファンクションを起動する際に不測の影響(予期せぬデータ変更など)が生じるのを回避するための一連のルールです。パッケージ純粋度とは、パッケージ初期化ブロック内のコードの純粋度を指します。

脚注2

プロシージャやファンクションのエントリ・ポイント番号は、PL/SQLパッケージ・コードにおける位置によって決まります。PL/SQLパッケージの最後に追加されたプロシージャやファンクションには、新しいエントリ・ポイント番号が付与されます。

ノート:

[表 26-2](#) の操作により無効化される依存オブジェクトが、静的データ・ディクショナリ・ビュー*_OBJECTS および*_OBJECTS_AE に反映されるのは、(コンパイル時または実行時に)そのオブジェクトの参照が試行された後、または次のいずれかのサブプログラムが起動された後です。



- DBMS_UTILITY.COMPILE_SCHEMA
- 任意の UTL_RECOMP サブプログラム

トピック:

- [セッションの状態と参照パッケージ](#)
- [セキュリティ認可](#)

関連項目:

- PL/SQLのコンパイル・パラメータの設定の詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください

い。

- DBMS_UTILITY.COMPILE_SCHEMA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。
- UTL_RECOMPサブプログラムの詳細は、[『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。

26.4.1 セッションの状態と参照パッケージ

パッケージ構成を参照するセッションごとに、各パッケージの独自のインスタンスが存在します。このインスタンスには、パブリック変数とプライベート変数、カーソルおよび定数の永続的な状態が含まれます。その後、セッションのインスタンス化されたパッケージのいずれかが無効になった後、再度有効化されると、そのセッションのパッケージのインスタンス化は、状態も含めてすべて失われる可能性があります。

関連項目:

- パッケージのインスタンス化の詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。
- パッケージ状態の詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。

26.4.2 セキュリティ認可

データ操作言語(DML)オブジェクト権限またはシステム権限をユーザーまたはPUBLICに対して付与または取り消すと、Oracle Databaseによって、その所有者のすべての依存オブジェクトが無効化されます。これは、依存オブジェクトの所有者が、すべての参照オブジェクトに関する必要な権限を引き続き持つことを確認するためです。

26.5 無効化を回避するためのガイドライン

依存オブジェクトの無効化を回避するには、次のガイドラインに従ってください。

- [パッケージの最後へのアイテムの追加](#)
- [ビューを介した各表の参照](#)

26.5.1 パッケージの最後へのアイテムの追加

アイテムをパッケージに追加するときは、パッケージの最後に追加します。このようにすると、既存の最上位パッケージ・アイテムのエントリ・ポイント番号が維持され、それらのパッケージ・アイテムが無効になりません。

たとえば、次のパッケージについて考えます。

```
CREATE OR REPLACE PACKAGE pkg1 AUTHID DEFINER IS
  FUNCTION get_var RETURN VARCHAR2;
END;
/
```

次のようにアイテムをpkg1の最後に追加すると、get_varファンクションを参照する依存オブジェクトが無効になりません。

```
CREATE OR REPLACE PACKAGE pkg1 AUTHID DEFINER IS
  FUNCTION get_var RETURN VARCHAR2;
  PROCEDURE set_var (v VARCHAR2);
END;
/
```

次のようにアイテムをget_var関数とset_varプロシージャの間に挿入すると、set_var関数を参照する依存オブジェクトが無効になります。

```
CREATE OR REPLACE PACKAGE pkg1 AUTHID DEFINER IS
  FUNCTION get_var RETURN VARCHAR2;
  PROCEDURE assert_var (v VARCHAR2);
  PROCEDURE set_var (v VARCHAR2);
END;
/
```

26.5.2 ビューを介した各表の参照

ビューを使用して表を間接的に参照すると、次のことができるようになります。

- 依存ビューまたは依存PL/SQLオブジェクトを無効にせずに、表に列を追加できます。
- 依存オブジェクトを無効にせずに、ビューで参照されていない列を変更または削除できます。

CREATE OR REPLACE VIEW文を使用しても、新しいROWTYPEが古いROWTYPEと一致する場合は、既存のビューまたはその依存オブジェクトが無効になることはありません。

26.6 オブジェクトの再有効化

参照されたときに無効になっているオブジェクトは、使用できるように有効化する必要があります。有効化は、オブジェクトが参照されるときに自動的に行われます。ユーザーが明示的な操作を行う必要はありません。

オブジェクトが無効な場合、ステータスはコンパイル・エラー、権限取消し、または無効のいずれかです。これらの用語の定義は、[表26-1](#)を参照してください。

トピック:

- [コンパイルされたがエラーがあるオブジェクトの再有効化](#)
- [未許可オブジェクトの再有効化](#)
- [無効SQLオブジェクトの再有効化](#)
- [無効PL/SQLオブジェクトの再有効化](#)

26.6.1 コンパイルされたがエラーがあるオブジェクトの再有効化

コンパイラでは、コンパイルされたがエラーのあるオブジェクトの自動的な再有効化は行われません。コンパイラはオブジェクトを再コンパイルします。オブジェクトがエラーなしで再コンパイルされると、再有効化されます。それ以外は無効のままです。

26.6.2 未許可オブジェクトの再有効化

コンパイラは、そのすべての参照オブジェクトに対するアクセス権限が未許可オブジェクトにあるかどうかを調べます。ある場合、未許可オブジェクトはコンパイラによって再コンパイルなしで再有効化されます。ない場合は、コンパイラによって対応するエラー・メッセージが発行されます。

26.6.3 無効SQLオブジェクトの再有効化

SQLコンパイラによって無効なオブジェクトが再コンパイルされます。オブジェクトがエラーなしで再コンパイルされると、再有効化されます。それ以外は無効のままです。

26.6.4 無効PL/SQLオブジェクトの再有効化

無効のPL/SQLプログラム・ユニット(プロシージャ、ファンクションまたはパッケージ)の場合、PL/SQLコンパイラが、参照オブジェクトの変更が無効オブジェクトに影響を与えるものだったかどうかを調べます。影響を与えた場合、コンパイラによって無効なオブジェクトが再コンパイルされます。オブジェクトがエラーなしで再コンパイルされると、再有効化されます。それ以外は無効のままです。影響がなかった場合、無効オブジェクトはコンパイラによって再コンパイルなしで再有効化されます。

26.7 スキーマの範囲内での名前解決

SQL文で参照されるオブジェクト名は1つ以上の部分で構成されます。各部分はピリオドで区切られます。たとえば、`hr.employees.department_id`は3つの部分から構成されています。

Oracle Databaseでは、次のプロシージャを使用して、オブジェクト名の問題を試行します。

ノート:



プロシージャを正常に実行するには、オブジェクト名のすべての部分が現行エディションで表示されている必要があります。

1. オブジェクト名の先頭の要素を修飾します。

オブジェクト名に要素が1つしかない場合は、その要素が最初の要素となります。要素が2つ以上ある場合は、最も左にあるピリオドの左側の要素が先頭の要素です。たとえば、`hr.employees.department_id`では、`hr`が先頭の要素です。

先頭の要素を修飾する手順は次のとおりです。

- a. オブジェクト名が、SELECT文のFROM句に含まれる表名であり、2つ以上の構成要素を持つ場合は、ステップdに進みます。それ以外の場合は、ステップbに進みます。

現在のスキーマ内で、最初の要素と一致する名前を持つオブジェクトが検索されます。

見つかった場合は、ステップ2に進みます。それ以外の場合は、ステップcに進みます。

- b. 最初の要素と一致するパブリック・シノニムが検索されます。

見つかった場合は、ステップ2に進みます。それ以外の場合は、ステップdに進みます。

- c. 最初の要素と名前が一致するスキーマが検索されます。

該当するスキーマが見つかった場合、オブジェクト名に後続の要素があれば、ステップeに進みます。それ以外の場合は、エラーが戻され、オブジェクト名を修飾できません。

- d. ステップdで見つかったスキーマ内で、オブジェクト名の2番目の要素と一致する名前を持つ表またはSQLファンクションが検索されます。

見つかった場合は、ステップ2に進みます。それ以外の場合は、エラーが戻され、オブジェクト名を修飾できません。

ノート:



このステップで検出された SQL ファンクションは、ステップ d で検出されたスキーマによって再定義されています。

2. スキーマ・オブジェクトは修飾されています。オブジェクト名の残りの要素は、このスキーマ・オブジェクトの有効な部分と一致する必要があります。

たとえば、オブジェクト名が `hr.employees.department_id` である場合、`hr` がスキーマとして修飾されます。`employees` が表として修飾されている場合、`department_id` はその表の列に対応する必要があります。また、`employees` がパッケージとして修飾されている場合、`department_id` はそのパッケージのパブリック定数、変数、プロシージャまたはファンクションに対応する必要があります。

Oracle Databaseによる参照の解決では、オブジェクトは、他のオブジェクトが存在しないという事実に依存する場合があります。このような状況が発生するのは、依存オブジェクトで使用されている参照の解釈が、別のオブジェクトの存在により多義になる場合です。

関連項目:

- SQLとPL/SQLでの名前解決の違いの詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。
- 分散データベース・システムでの名前解決の詳細は、[『Oracle Database管理者ガイド』](#)を参照してください。
- [エディション・オブジェクトと非エディション・オブジェクトの名前解決](#)

26.8 ローカル依存性の管理

ローカルの依存性管理は、Oracle Databaseが単一データベースのオブジェクト間の依存性を管理するときに行われます。たとえば、プロシージャ内の文が同じデータベースの表を参照することができます。

26.9 リモート依存性の管理

リモートの依存性管理は、Oracle Databaseがネットワーク上の分散環境で依存性を管理するときに行われます。たとえば、Oracle Formsトリガーは、データベースのスキーマ・オブジェクトに依存することができます。分散データベースでは、ローカル・ビューはリモート表を参照できます。

Oracle Databaseでは、分散データベースの依存性も管理されます。たとえば、Oracle Formsアプリケーションは、表を参照するトリガーを含むことがあります。データベース・システムは、そのオブジェクト間の依存性を把握する必要があります。Oracle Databaseがリモート依存性を管理する際は、関係するオブジェクトに応じて様々なメカニズムを使用します。

トピック:

- [ローカル・データベース・プロシージャとリモート・データベース・プロシージャの間の依存性](#)
- [その他のリモート・オブジェクト間の依存性](#)
- [アプリケーションの依存性](#)

26.9.1 ローカル・データベース・プロシージャとリモート・データベース・プロシージャの間の依存性

分散データベース・システムでのストアド・プロシージャ(関数、パッケージおよびトリガーを含む)間の依存性は、タイムスタンプ・

チェックまたはシグネチャ・チェックを使用して管理されます。

動的初期化パラメータREMOTE_DEPENDENCIES_MODEによって、タイムスタンプとシグネチャのどちらでリモート依存性を管理するかが決まります。

関連項目:

- [Oracle Database PL/SQL言語リファレンス](#)
- [タイムスタンプ依存性モード](#)
- [RPCシグネチャ依存性モード](#)

26.9.2 その他のリモート・オブジェクト間の依存性

Oracle Databaseでは、ローカル・プロシージャとリモート・プロシージャ間の依存性を除き、リモート・スキーマ・オブジェクト間の依存性は管理されません。

たとえば、リモート表を参照する問合せによってローカル・ビューが作成および定義されるとします。また、ローカル・プロシージャに、同じリモート表を参照するSQL文が含まれるとします。その後、表の定義が変更されます。

したがって、表の変更後にビューとプロシージャが使用され、そのビューとプロシージャがエラーとなった場合でも、ローカル・ビューとプロシージャは無効になりません。この場合、ビューとプロシージャを、エラーが戻されないように手動で変更する必要があります。このような場合は、依存オブジェクトを不必要に再コンパイルするより、依存性を管理しないことをお勧めします。

26.9.3 アプリケーションの依存性

データベース・アプリケーションのコードは、接続先データベースのオブジェクトを参照できます。たとえば、Oracle Call Interface(OCI)およびプリコンパイラ・アプリケーションは無名PL/SQLブロックを発行できます。Oracle Formsアプリケーションのトリガーは、スキーマ・オブジェクトを参照できます。

これらのアプリケーションは、参照しているスキーマ・オブジェクトに依存しています。依存性管理の方法は開発環境によって異なります。Oracle Databaseでは、アプリケーションの依存性が自動的に追跡されることはありません。

関連項目:

データベース・アプリケーション内のリモート依存性管理の詳細は、アプリケーション開発ツールおよびオペレーティング・システムのマニュアルを参照してください。

26.10 リモート・プロシージャ・コール(RPC)の依存性管理

リモート・プロシージャ・コール(RPC)の依存性管理は、分散データベース・システムでローカル・ストアド・プロシージャがリモート・プロシージャをコールするときに行われます。動的初期化パラメータREMOTE_DEPENDENCIES_MODEは、依存性モードを制御します。選択肢は、タイムスタンプ依存性モード、またはRPCシグネチャ依存性モードです。

トピック:

- [タイムスタンプ依存性モード](#)
- [RPCシグネチャ依存性モード](#)
- [依存性モードの制御](#)

26.10.1 タイムスタンプ依存性モード

プロシージャのコンパイル時に、タイムスタンプがデータ・ディクショナリに記録されます。タイムスタンプは、プロシージャが作成、変更または置換された時刻を示します。

コンパイル済のプロシージャには、このプロシージャがコールする各リモート・プロシージャの情報(リモート・プロシージャのスキーマ、パッケージ名、プロシージャ名、タイムスタンプなど)が含まれます。

タイムスタンプ依存性モードでは、ローカル・リモート・プロシージャがリモート・プロシージャをコールすると、Oracle Databaseは、リモート・プロシージャに対するローカル・プロシージャのタイムスタンプと、リモート・プロシージャの現在のタイムスタンプを比較します。2つのタイムスタンプが一致した場合、ローカル・プロシージャとリモート・プロシージャの両方が実行されます。どちらも再コンパイルされません。

2つのタイムスタンプが一致しない場合は、ローカル・プロシージャが無効になり、コール元の環境にエラーが返されます。新しいタイムスタンプのリモート・プロシージャに依存する他のすべてのローカル・プロシージャも無効になります。

タイムスタンプは、ローカル・プロシージャ本体の文によってリモート・プロシージャが実行されるときに比較されます。したがって、ローカル・プロシージャ内の無効なプロシージャ・コールよりも前にある文は、正常に実行されることがあります。無効なコールの後の分は実行されません。ローカル・プロシージャは再コンパイルが必要です。

無効なコールの前にDML文がある場合、この文と無効なコールが同じPL/SQLブロックにある場合のみ、この文がロールバックされます。たとえば、次のコードではUPDATE文がロールバックされます。

```
BEGIN
```

```
    UPDATE table SET ...  
    invalid_proc;  
    COMMIT;
```

```
END;
```

しかし、次のコードではUPDATE文はロールバックされません。

```
UPDATE table SET ...  
EXECUTE invalid_proc;  
COMMIT;
```

タイムスタンプ依存性モードには次のデメリットがあります。

- ネットワーク上での依存オブジェクトに対して、パフォーマンスに関係なく、不要な再コンパイルが行われることがよくあります。
- クライアント側アプリケーションでPL/SQLが使用されていると、このモードでは、クライアント側でアプリケーションの実行が妨げられる状況になる場合があります。

このようなアプリケーションの例はOracle Formsです。インストール中、クライアント・サイトでOracle Formsが使用するクライアント側PL/SQLプロシージャを再コンパイルする必要があります。また、クライアント側プロシージャがサーバー・プロシージャに依存しているときに、サーバー・プロシージャが変更または自動再コンパイルされた場合は、クライアント側PL/SQLプロシージャを再コンパイルする必要があります。ただし、クライアントではPL/SQLコンパイラは使用できません。したがって、クライアント・アプリケーション開発者は、新しいバージョンのアプリケーションをすべての顧客に再配布する必要があります。

26.10.2 RPCシグネチャ依存性モード

Oracle Databaseは、リモート依存性を処理するために、RPCシグネチャを提供します。ローカル環境では常に再コンパイルが可能であるため、RPCシグネチャはローカル依存性に影響を与えません。

RPCシグネチャは、各コンパイル済ストアド・プログラム・ユニットと関連付けられています。これは、次の特性を基準にユニットを識別しています。

- 名前
- パラメータの数
- 各パラメータのデータ型クラス
- 各パラメータのモード
- ファンクションの戻り値のデータ型クラス

RPCシグネチャは、先行する特性の少なくとも1つが変更された場合のみ変更されます。

ノート:



RPCシグネチャには、DETERMINISTIC、PARALLEL_ENABLE、または純粋度の情報は含まれません。リモート・システムのあるファンクションについてこれらの設定が変更された場合、これらに基づく最適化は自動的に再検討されません。したがって、SQL文でリモート・ファンクションをコールしたり、ファンクションに基づいたインデックスで使用したりすると、問合せの結果が正しくならないことがあります。

コンパイル済のプログラム・ユニットには、このプログラムがコールする各リモート・プロシージャのRPCシグネチャ(および、リモート・プロシージャのスキーマ、パッケージ名、プロシージャ名、タイムスタンプ)が含まれます。

RPCシグネチャ依存性モードでは、ローカル・プログラム・ユニットがリモート・プログラム・ユニットでサブプログラムをコールすると、データベースはタイムスタンプの不一致を無視し、リモート・サブプログラムに対するローカル・ユニットのRPCシグネチャと、リモート・サブプログラムの現在のRPCシグネチャを比較します。RPCシグネチャが一致する場合、コールは正常に行われます。一致しない場合、データベースからローカル・ユニットにエラーが返され、このユニットが無効化されます。

たとえば、プロシージャget_emp_nameが、Bostonにあるサーバー(BOSTON_SERVER)に格納されているとします。

```
CREATE OR REPLACE PROCEDURE get_emp_name (  
    emp_number    IN    NUMBER,  
    hiredate      OUT VARCHAR2,  
    emp_name      OUT VARCHAR2) AUTHID DEFINER AS  
BEGIN  
    SELECT last_name, TO_CHAR(hire_date, 'DD-MON-YY')  
    INTO emp_name, hiredate  
    FROM employees  
    WHERE employee_id = emp_number;  
END;  
/
```

get_emp_nameがBOSTON_SERVERでコンパイルされたときに、Oracle Databaseにより、RPCシグネチャとタイムスタンプの両方が記録されます。

たとえば、print_nameをコールするPL/SQLプロシージャget_emp_nameがCaliforniaにあるサーバーにあるとしましょう。

```
CREATE OR REPLACE PROCEDURE print_ename (emp_number IN NUMBER) AUTHID DEFINER AS  
    hiredate VARCHAR2(12);
```

```

    ename      VARCHAR2(10);
BEGIN
    get_emp_name@BOSTON_SERVER(emp_number, hiredate, ename);
    dbms_output.put_line(ename);
    dbms_output.put_line(hiredate);
END;
/

```

print_nameがCaliforniaサーバーでコンパイルされると、このデータベースはBostonサーバーに接続し、get_emp_nameのRPCシグネチャをCaliforniaサーバーに送信してから、get_emp_nameのRPCシグネチャをコンパイル済のprint_nameに記録します。

実行時、print_nameがget_emp_nameをコールすると、データベースは、コンパイル済のprint_nameに記録されたget_emp_nameのRPCシグネチャをBostonサーバーに送信します。記録されたRPCシグネチャが、Bostonサーバーのget_emp_nameの現在のRPCシグネチャと一致した場合、コールは正常に行われます。それ以外の場合、データベースはprint_nameにエラーを返し、その後、これが無効化されます。

トピック:

- [パラメータの名前とデフォルト値の変更](#)
- [パラメータ・モードINの指定の変更](#)
- [サブプログラム本体の変更](#)
- [パラメータのデータ型クラスの変更](#)
- [パッケージ・タイプの変更](#)

26.10.2.1 パラメータの名前とデフォルト値の変更

サブプログラム・パラメータの名前またはデフォルト値を変更しても、サブプログラムのRPCシグネチャは変更されません。たとえば、次の2つの例でプロシージャP1のRPCシグネチャは同じです。

```

PROCEDURE P1 (Param1 IN NUMBER := 100);
PROCEDURE P1 (Param2 IN NUMBER := 200);

```

ただし、コール側で新しいデフォルト値を取得できるようにする必要があるアプリケーションの場合、コールされたプロシージャを再コンパイルする必要があります。

26.10.2.2 パラメータ・モードINの指定の変更

サブプログラム・パラメータ・モードINはデフォルトなので、暗黙的または明示的に指定できます。指定を暗黙的から明示的、または明示的から暗黙的に変更しても、サブプログラムのRPCシグネチャは変更されません。たとえば、次の2つの例でプロシージャP1のRPCシグネチャは同じです。

```

PROCEDURE P1 (Param1 NUMBER);      -- implicit specification
PROCEDURE P1 (Param1 IN NUMBER);   -- explicit specification

```

26.10.2.3 サブプログラム本体の変更

サブプログラムの本体を変更しても、サブプログラムのRPCシグネチャは変更されません。

[例26-4](#)では、プロシージャget_hire_dateの本体のみ変更されます。したがって、get_hire_dateのRPCシグネチャは変更されません。

例26-4 プロシージャget_hire_dateの本体の変更

```

CREATE OR REPLACE PROCEDURE get_hire_date (
  emp_number IN NUMBER,
  hiredate OUT VARCHAR2,
  emp_name OUT VARCHAR2) AUTHID DEFINER AS
BEGIN
  SELECT last_name, TO_CHAR(hire_date, 'DD-MON-YY')
  INTO emp_name, hiredate
  FROM employees
  WHERE employee_id = emp_number;
END;
/
CREATE OR REPLACE PROCEDURE get_hire_date (
  emp_number IN NUMBER,
  hiredate OUT VARCHAR2,
  emp_name OUT VARCHAR2) AUTHID DEFINER AS
BEGIN
  -- Change date format model
  SELECT last_name, TO_CHAR(hire_date, 'DD/MON/YYYY')
  INTO emp_name, hiredate
  FROM employees
  WHERE employee_id = emp_number;
END;
/

```

26.10.2.4 パラメータのデータ型クラスの変更

パラメータのデータ型を同じクラスの別のデータ型に変更しても、RPCシグネチャは変更されませんが、別のクラスのデータ型に変更すると、シグネチャも変更されます。

[表26-3](#)に、データ型クラスと、クラスを構成するデータ型を示します。[NCHAR](#)など、表26-3に含まれないデータ型はデータ型クラスには属していません。これらの型を変更すると、必ずRPCシグネチャも変更されます。

表26-3 データ型クラス

データ型クラス	クラスのデータ型
文字	CHAR CHARACTER
VARCHAR	VARCHAR VARCHAR2 STRING LONG ROWID
RAW	RAW LONG RAW
整数	BINARY_INTEGER PLS_INTEGER SIMPLE_INTEGER BOOLEAN NATURAL NATURALN POSITIVE POSITIVEN
数値	NUMBER INT INTEGER SMALLINT DEC DECIMAL REAL FLOAT NUMERIC DOUBLE PRECISION
日時	DATE TIMESTAMP TIMESTAMP WITH TIME_ZONE TIMESTAMP WITH LOCAL TIME ZONE INTERVAL YEAR TO MONTH INTERVAL DAY TO SECOND

[例26-5](#)はパラメータhiredateのデータ型をVARCHAR2からDATEに変更します。VARCHAR2とDATEは同じデータ型クラス

ではないので、プロシージャget_hire_dateのRPCシグネチャが変更されます。

例26-5 get_hire_dateパラメータのデータ型クラスの変更

```
CREATE OR REPLACE PROCEDURE get_hire_date (  
    emp_number    IN    NUMBER,  
    hiredate      OUT DATE,  
    emp_name      OUT VARCHAR2) AS  
BEGIN  
    SELECT last_name, TO_CHAR(hire_date, 'DD/MON/YYYY')  
    INTO emp_name, hiredate  
    FROM employees  
    WHERE employee_id = emp_number;  
END;  
/
```

26.10.2.5 パッケージ・タイプの変更

パッケージ・タイプの名前、または内部コンポーネントの名前を変更しても、パッケージのRPCシグネチャは変更されません。

[例26-6](#)では、パッケージemp_packageの内部にレコード型emp_data_typeが定義されます。次に、レコード・フィールドの名前が変更されます。レコード型は変更されません。最後に、レコード型の名前が変更されます。特性は変更されません。パッケージのRPCシグネチャは変更されません。

例26-6 パッケージ・レコード型のフィールド名の変更

```
CREATE OR REPLACE PACKAGE emp_package AUTHID DEFINER AS  
    TYPE emp_data_type IS RECORD (  
        emp_number    NUMBER,  
        hiredate      VARCHAR2(12),  
        emp_name      VARCHAR2(10)  
    );  
    PROCEDURE get_emp_data (  
        emp_data      IN OUT emp_data_type  
    );  
END;  
/  
  
CREATE OR REPLACE PACKAGE emp_package AUTHID DEFINER AS  
    TYPE emp_data_type IS RECORD (  
        emp_num      NUMBER,  
        hire_dat     VARCHAR2(12),  
        empname      VARCHAR2(10)  
    );  
    PROCEDURE get_emp_data (  
        emp_data      IN OUT emp_data_type  
    );  
END;  
/  
  
CREATE OR REPLACE PACKAGE emp_package AUTHID DEFINER AS  
    TYPE emp_data_record_type IS RECORD (  
        emp_num      NUMBER,  
        hire_dat     VARCHAR2(12),  
        empname      VARCHAR2(10)  
    );  
    PROCEDURE get_emp_data (  
        emp_data      IN OUT emp_data_record_type  
    );  
END;  
/
```

26.10.3 依存性モードの制御

動的初期化パラメータREMOTE_DEPENDENCIES_MODEは、依存性モードを制御します。初期化パラメータ・ファイルに次の

ように指定されているとします。この場合は、依存性を解決するためにタイムスタンプのみが使用されます(明示的かつ動的にオーバーライドされない場合)。

```
REMOTE_DEPENDENCIES_MODE = TIMESTAMP
```

初期化パラメータ・ファイルでパラメータが次のように指定されている場合は、依存性を解決するためにRPCシグネチャのみが使用されます(明示的かつ動的にオーバーライドされない場合)。

```
REMOTE_DEPENDENCIES_MODE = SIGNATURE
```

DDL文を使用して動的にモードを変更することができます。たとえば、次の例では現行セッションの依存性モードが変更されます。

```
ALTER SESSION SET REMOTE_DEPENDENCIES_MODE = {SIGNATURE | TIMESTAMP}
```

次の例は、起動後にシステム全体の依存性モードを変更します。

```
ALTER SYSTEM SET REMOTE_DEPENDENCIES_MODE = {SIGNATURE | TIMESTAMP}
```

init.oraパラメータ・ファイルに、あるいはALTER SESSIONまたはALTER SYSTEM文を使用して、REMOTE_DEPENDENCIES_MODEパラメータが指定されない場合、TIMESTAMPがデフォルト値になります。したがって、REMOTE_DEPENDENCIES_MODEパラメータを明示的に使用するか適切なDDL文を使用しないかぎり、サーバーはタイムスタンプ依存性モードを使用して実行します。

REMOTE_DEPENDENCIES_MODE=SIGNATUREを使用すると、次のようになります。

- リモート・プロシージャのパラメータの初期値を変更しても、そのリモート・プロシージャをコールするローカル・プロシージャは無効になりません。リモート・プロシージャへのコールでパラメータが指定されない場合は、初期値が使用されます。この場合、無効化と再コンパイルが自動的にには行われないため、古い初期値が使用されます。新しい初期値を確認するには、手動でコール側プロシージャを再コンパイルします。
- オーバーロードされたプロシージャ(既存プロシージャと同名のプロシージャ)をパッケージへ追加しても、リモート・プロシージャをコールするローカル・プロシージャは無効にされません。このオーバーロードの結果として、ローカル・プロシージャからの既存のコールがタイムスタンプ・モードで再バインドされた場合、ローカル・プロシージャは無効にされないため、RPCシグネチャ・モードでは、この再バインドは行われません。再バインドを成功させるには、ローカル・プロシージャを手動で再コンパイルする必要があります。
- 新しい型と以前の型が同じになるように既存のパッケージ・プロシージャのパラメータの型が変更された場合でも、ローカルのコール側プロシージャは自動的に無効化または再コンパイルされません。新しい型のセマンティクスを取得するには、コール側プロシージャを手動で再コンパイルする必要があります。

トピック:

- [依存性の解決](#)
- [依存性管理の提案](#)

26.10.3.1 依存性の解決

REMOTE_DEPENDENCIES_MODE = TIMESTAMP (デフォルト値)のとき、プログラム・ユニット間の依存性は、実行時にタイムスタンプを比較することで処理されます。コールされたリモート・プロシージャのタイムスタンプが、コールしたプロシージャのタイムスタンプと一致しない場合、コール側の(依存)ユニットは無効になり、再コンパイルする必要が生じます。この場合、ローカルPL/SQLコンパイラがない場合は、コール側のアプリケーションを処理できません。

タイムスタンプ依存性モードでは、RPCシグネチャは比較されません。ローカルPL/SQLコンパイラがある場合は、コール側プロ

シーが実行されると自動的に再コンパイルされます。

REMOTE_DEPENDENCIES_MODE = SIGNATUREの場合、コール側のユニットの記録済タイムスタンプは、まず最初にコールされたリモート・ユニット内の現在のタイムスタンプと比較されます。タイムスタンプが一致すると、コールが処理されます。タイムスタンプが一致しない場合は、コールされたリモート・サブプログラムのRPCシグネチャは、コール側サブプログラムに記録されたままの状態、コールされたサブプログラムの現在のRPCシグネチャと比較されます。([「パラメータのデータ型クラスの変更」](#)で説明する基準を使用して)シグネチャが一致しない場合、エラーがコール元セッションに返されます。

26.10.3.2 依存性管理の提案

REMOTE_DEPENDENCIES_MODEパラメータを設定する際には、次のガイドラインに従います。

- サーバー側PL/SQLユーザーは、パラメータをTIMESTAMPに設定して(つまりデフォルトのままにして)タイムスタンプ依存性モードを使用できます。
- サーバー側PL/SQLユーザーが、分散システムを使用しており、不要な再コンパイルを回避したい場合には、RPCシグネチャ依存性モードを使用できます。
- クライアント側PL/SQLユーザーは、パラメータをSIGNATUREに設定する必要があります。次の操作を実行できます。
 - クライアント・サイトにアプリケーションをインストールするときに、プロシージャを再コンパイルする必要がありません。
 - サーバーをアップグレードしてもタイムスタンプの不一致が発生しません。
- サーバー側でRPCシグネチャ・モードを使用するときは、プロシージャをパッケージ仕様部のプロシージャ(またはファンクション)宣言の最後に追加します。宣言リストの中間にプロシージャを追加すると、依存プロシージャの不必要な無効化と再コンパイルを引き起こすことがあります。

26.11 共有SQLの依存性管理

スキーマ・オブジェクト相互間の依存性管理に加えて、Oracle Databaseでは共有プール内の各共有SQL領域の依存性も管理します。表またはビュー、シノニム、順序を生成、変更または削除したり、プロシージャまたはパッケージ仕様部を再コンパイルすると、それらに依存する共有SQL領域もすべて無効になります。共有SQL領域が無効になった後で、その領域に対応するカーソルを実行すると、Oracle DatabaseによってSQL文が再解析されて、共有SQL領域が再生成されます。

27 エディションベースの再定義の使用

エディションベースの再定義(EBR)を使用すると、アプリケーションの使用中にそのデータベース・コンポーネントをアップグレードでき、停止時間を最小化あるいは排除することができます。

トピック:

- [エディションベースの再定義の概要](#)
- [エディション](#)
- [エディショニング・ビュー](#)
- [crosseditionトリガー](#)
- [EBR機能に関する情報の表示](#)
- [アプリケーションをアップグレードするためのEBRの使用](#)

27.1 エディションベースの再定義の概要

アプリケーションを使用中にアップグレードするには、アプリケーションのデータベース・コンポーネントを構成するデータベース・オブジェクトをコピーし、コピーしたオブジェクトを個別に再定義する必要があります。アプリケーションのユーザーは、この変更による影響を受けず、変更されていないアプリケーションを引き続き実行できます。変更が正しいことを確認できたら、アップグレードしたアプリケーションをすべてのユーザーが使用できるようにします。

EBRでは、1つ以上のコンポーネント機能を使用します。使用する機能と停止時間は次の要素によって異なります。

- 再定義するデータベース・オブジェクトの種類
- データベース・オブジェクトの再定義中に、ユーザーがそのデータベース・オブジェクトをどの程度使用できるか
- 以前のアプリケーションを使用しているユーザーがいるときに、アップグレードしたアプリケーションを他のユーザーが使用できるようにするかどうか

データベース・オブジェクトをコピーし、コピーしたデータベース・オブジェクトを分離した状態で再定義するには、常にエディション機能を使用します。この章で説明するオンラインでのアプリケーション・アップグレード手順がエディションベースの再定義(EBR)と呼ばれるのはこのためです。

再定義するすべてのオブジェクトがエディション化(定義は[「エディション・オブジェクトと非エディション・オブジェクト」](#)を参照)されている場合、使用する機能はエディションのみです。

表はエディション・オブジェクトではありません。1つ以上の表の構造を変更する場合は、エディショニング・ビュー機能も使用します。

表の構造を変更している間、他のユーザーがその表のデータを変更する必要がある場合は、forward crosseditionトリガーも使用します。アップグレード前のアプリケーションとアップグレード後のアプリケーションが同時に通常使用(ホット・ロールオーバー)されている場合、reverse crosseditionトリガーも使用します。crosseditionトリガーはアプリケーションの永続的な要素ではありません。全ユーザーがアップグレード後のアプリケーションを使用するようになったら削除します。

あるアプリケーションが他のエディションでも実行される場合に、1つのエディションでそのアプリケーションに対して実行可能なEBR操作は、ライブ操作です。

27.2 エディション

エディションは非スキーマ・オブジェクトであり、所有者が存在しません。エディションは1つのネームスペースに作成されます。複数のエディションがデータベースに共存することができます。

データベースには少なくとも1つのエディションが必要です。新規作成またはアップグレードされたOracle Databaseはすべて、ora\$baseという名前の1つのエディションから開始されます。

ノート:



マルチテナント・コンテナ・データベース(CDB)では、エディション、エディショニング・ビューまたはクロスエディション・トリガーの範囲は機能が作成されたプラガブル・データベース(PDB)です。非 CDB の場合、これらの各機能の範囲はデータベース全体です。

トピック:

- [エディション・オブジェクトと非エディション・オブジェクト](#)
- [エディションの作成](#)
- [エディション・オブジェクトおよび変更時のコピー](#)
- [一部のユーザーに対してエディションを使用可能にする](#)
- [すべてのユーザーに対してエディションを使用可能にする](#)
- [現行エディションとセッション・エディション](#)
- [エディションのリタイア](#)
- [エディションの削除](#)

関連項目:

CDBおよびPDBの詳細は、[『Oracle Database管理者ガイド』](#)を参照してください。

27.2.1 エディション・オブジェクトと非エディション・オブジェクト

ノート:



ユーザーとスキーマという用語は同義です。スキーマ・オブジェクトの所有者は、これを所有するユーザー/スキーマです。

エディション・オブジェクトには、所有者でエディション対応のスキーマ・オブジェクト・タイプとEDITIONABLEプロパティの両方があります。エディションにはエディション・オブジェクトの独自のコピーがあり、そのエディションに対してはこのコピーのみが表示可能です。

非エディション・オブジェクトには、所有者で非エディション対応のスキーマ・オブジェクト・タイプまたはNONEDITIONABLEプロパティのいずれかがあります。エディションは、非エディション・オブジェクトの独自のコピーを持つことはできません。非エディション・オブジェクトは、すべてのエディションに対して表示可能です。

オブジェクトは、その所有者におけるそのタイプに対するエディションの有効化によってエディション・オブジェクトになる場合、潜在的にエディション対応になります。

エディション・オブジェクトは、スキーマとエディションの両方に属し、そのOBJECT_NAME、OWNER、およびEDITION_NAMEにより一意に識別されます。非エディション・オブジェクトは、スキーマのみに属し、そのOBJECT_NAMEおよびOWNERにより(EDITION_NAMEはNULL)一意に識別されます。(厳密に言えば、オブジェクトを一意に識別するにはオブジェクトのNAMESPACEも必要ですが、オブジェクトを参照する文では暗黙的または明示的にNAMESPACEが指定されるので、これは無視しても構いません)。

オブジェクトのOBJECT_NAME、OWNER、およびEDITION_NAMEは、静的データ・ディクショナリ・ビュー*_OBJECTSおよび*_OBJECTS_AEに表示できます。

オブジェクトのEDITION_NAMEを知らなくても、このオブジェクトを参照できます(知っていても、指定できません)。参照のコンテキストで、エディションを暗黙的に指定します。コンテキストがデータ定義言語(DDL)文である場合、エディションは、このコマンドを発行したセッションの現行エディションです。コンテキストがソース・コードである場合、エディションは、オブジェクトが実オブジェクトであるものです。

トピック:

- [エディション・オブジェクトと非エディション・オブジェクトの名前解決](#)
- [エディション・オブジェクトに依存可能な非エディション・オブジェクト](#)
- [エディション対応、およびエディション非対応スキーマ・オブジェクト・タイプ](#)
- [ユーザーに対するエディションの有効化](#)
- [EDITIONABLEおよびNONEDITIONABLEプロパティ](#)
- [エディション・オブジェクトの規則](#)

関連項目:

- [ユーザーに対するエディションの有効化](#)
- [現行エディションとセッション・エディション](#)
- [エディション・オブジェクトおよび変更時のコピー](#)

27.2.1.1 エディション・オブジェクトと非エディション・オブジェクトの名前解決

オブジェクト名の解決を試行するために、Oracle Databaseは、[「スキーマの範囲内での名前解決」](#)で説明されている手順を使用します。プロシージャを正常に実行するには、オブジェクト名のすべての部分が現行エディションで表示されている必要があります。

エディション・オブジェクトの名前解決時には、現行エディションのエディション・オブジェクトと非エディション・オブジェクトの両方が表示可能です。

非エディション・オブジェクトの名前解決時には、非エディション・オブジェクトのみが表示可能です。したがって、エディション・オブジェクトを参照する非エディション・オブジェクトを作成しようとする([「エディション・オブジェクトに依存可能な非エディション・オブジェクト」](#)で説明されている場合を除く)、作成は失敗し、エラーが表示されます。

参照対象のエディション・オブジェクトを変更する場合、そのすべての依存(直接および間接)が無効になります。無効なオブジェクトが参照されると、データベースはそのオブジェクトを検証しようとします。

関連項目:

- [現行エディションとセッション・エディション](#)
- 無効化、再有効化、名前解決を含む、スキーマ・オブジェクト間の依存性の詳細は、[「スキーマ・オブジェクトの依存性の理解」](#)を参照してください

27.2.1.2 エディション・オブジェクトに依存可能な非エディション・オブジェクト

通常、非エディション・オブジェクトはエディション・オブジェクトに依存できません。これは、名前解決時にはエディション・オブジェクトは表示できないためです。ただし、非エディション・オブジェクトは、名前解決時にエディション・オブジェクトを検索するためにエディション(評価エディション)を指定すると、エディション・オブジェクトに依存できます。評価エディションを指定するには、非エディション・オブジェクトは次のいずれかである必要があります。

- マテリアライズド・ビュー
- 仮想列

トピック:

- [マテリアライズド・ビュー](#)
- [仮想列](#)

27.2.1.2.1 マテリアライズド・ビュー

マテリアライズド・ビューは、評価エディションを指定することによってエディション・オブジェクトに依存できるようにする非エディション・オブジェクトです。エディション・オブジェクトに依存するマテリアライズド・ビューは、`query_rewrite_clause`で指定する特定のエディション範囲内の問合せリライトに対してのみ適格な場合があります。

マテリアライズド・ビューを作成するための簡単な構文は、次のとおりです。

```
CREATE MATERIALIZED VIEW [ schema.] materialized_view other_clauses  
[ evaluation_edition_clause ] [ query_rewrite_clause ] AS subquery
```

この場合、`evaluation_edition_clause`は、次のとおりです。

```
EVALUATE USING { CURRENT EDITION | EDITION edition | NULL EDITION }
```

また、`query_rewrite_clause`は、次のとおりです。

```
{ DISABLE | ENABLE } QUERY REWRITE  
[ unusable_before_clause ] [ unusable_beginning_clause ]
```

この場合、`unusable_before_clause`は、次のとおりです。

```
UNUSABLE BEFORE { CURRENT EDITION | EDITION edition }
```

また、`unusable_beginning_clause`は、次のとおりです。

```
UNUSABLE BEGINNING WITH { CURRENT EDITION | EDITION edition | NULL EDITION }
```

`CURRENT EDITION`は、DDL文が実行されるエディションを指します。`NULL EDITION`を指定することは、これが含まれる句を省略することと同じです。`evaluation_edition_clause`を省略すると、名前解決時にエディション・オブジェクトは表示できなくなります。

評価エディションまたは使用不可エディションを無効化、有効化または変更するには、`ALTER MATERIALIZED VIEW`文を使

用します。

マテリアライズド・ビューの評価エディションおよび使用不可エディションを表示するには、静的データ・ディクショナリ・ビュー *_MVIEWSを使用します。

評価エディションを削除すると、マテリアライズド・ビューが無効になります。マテリアライズド・ビューが使用可能なエディションを削除しても、マテリアライズド・ビューは無効になりません。

関連項目:

CREATE MATERIALIZED VIEW文の詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。

[Oracle Database SQL言語リファレンス](#)

[Oracle Databaseリファレンス](#)

27.2.1.2.2 仮想列

仮想列(生成列とも呼ばれます)は、ディスク領域を使用しません。データベースは、式を評価することによってリクエストごとに仮想列で値を生成します。式によってPL/SQLファンクションを起動できます(PL/SQLファンクションはエディション・オブジェクトでもかまいません)。仮想列は、評価エディションを指定することにより、エディション化PL/SQLファンクションを起動する式に依存できるようにします。

仮想列を作成するための構文は、次のとおりです。

```
column [ datatype ] [ GENERATED ALWAYS ] AS ( column_expression )  
[ VIRTUAL ] [ evaluation_edition_clause ]  
[ unusable_before_clause ] [ unusable_beginning_clause ]  
[ inline_constraint ]...
```

evaluation_edition_clauseの詳細は、[「マテリアライズド・ビュー」](#)を参照してください。

仮想列が起動するファンクションの依存性は保持されません。したがって、評価エディションが削除されるか、仮想列が非エディション・ファンクションに依存しているときに、ファンクションがエディション化されると、次のいずれかの例外が発生する可能性があります。

- 仮想列の問合せ試行
- 仮想列が含まれる行の更新試行
- 仮想列へのアクセスを試行するトリガー

仮想列の評価エディションを表示するには、静的データ・ディクショナリ・ビュー *_TAB_COLSを使用します。

関連項目:

- 仮想列定義の完全な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。
- ALL_TAB_COLSの詳細は、[『Oracle Databaseリファレンス』](#)を参照してください。

27.2.1.3 エディション対応、およびエディション非対応スキーマ・オブジェクト・タイプ

スキーマ内のスキーマ・オブジェクト・タイプをエディション対応にするには、データベース内でエディション対応にする必要があります。データベース内でエディション対応であるスキーマ・オブジェクト・タイプは、COMPATIBLE初期化パラメータの値によって決定され、動的パフォーマンス・ビューV\$EDITIONABLE_TYPESによって表示されます。

COMPATIBLEの値が12以上である場合、データベース内で次のスキーマ・オブジェクト・タイプがエディション対応になります。

- SYNONYM
- VIEW
- SQL翻訳プロファイル
- すべてのPL/SQLオブジェクト型:
 - FUNCTION
 - LIBRARY
 - PACKAGEおよびPACKAGE BODY
 - PROCEDURE
 - TRIGGER
 - TYPEおよびTYPE BODY

他のすべてのスキーマ・オブジェクト・タイプはデータベースおよびすべてのスキーマ内で非エディション対応であり、そのタイプのオブジェクトは常にエディション化されません。非エディション対応スキーマ・オブジェクト・タイプの例はTABLEです。表は常に非エディション化オブジェクトです。

スキーマ・オブジェクト・タイプがデータベース内でエディション対応である場合、スキーマ内でエディション対応にできます。

関連項目:

- COMPATIBLE初期化パラメータの管理の詳細は、[『Oracle Database管理者ガイド』](#)を参照してください。
- V\$EDITIONABLE_TYPESの詳細は、[『Oracle Databaseリファレンス』](#)を参照してください。
- [ユーザーに対するエディションの有効化](#)

27.2.1.4 ユーザーに対するエディションの有効化

ノート:

- エディションの有効化は、ライブ操作ではありません。
- データベースがリリース 11.2 からリリース 12.1 にアップグレードされると、アップグレード前のデータベースでエディションに対して有効であったユーザーは、アップグレード後のデータベースでもエディションに対して有効になり、デフォルトのスキーマ・オブジェクト・タイプはこれらのスキーマでエディション対応になります。デフォルトのスキーマ・オブジェクト・タイプは、静的データ・ディクショナリ・ビューDBA_EDITIONED_TYPES によって表示されます。アップグレード前のデータベースでエディションに対して無効だったユーザーは、アップグレード後のデータベースでもエディションに対して無効であり、スキーマ・オブジェクト・タイプはこれらのスキーマではエディション対応になりません。
- エディションが有効になっているユーザー確認するには、静的データ・ディクショナリ・ビューDBA_USERS または USER_USERS の EDITIONS_ENABLED 列を参照してください。

ユーザーに対してエディションを有効化するには、CREATE USER、またはALTER USER文のENABLE EDITIONS句を使用します。

ALTER USER文を使用すると、スキーマ内でエディション対応になるスキーマ・オブジェクト・タイプを指定できます。

```
ALTER USER user ENABLE EDITIONS [ FOR type [, type ]... ]
```

FORリストから省略するタイプはすべて、データベース内でエディション対応であるとしても、スキーマ内で非エディション対応になります。(タイプがデータベース内で非エディション対応である場合、このタイプはすべてのスキーマ内で常に非エディション対応になります。)

ALTER USER文からFORリストを省略するか、CREATE USER文を使用してユーザーに対してエディションを有効化すると、スキーマ内でエディション対応になるタイプは、静的データ・ディクショナリ・ビューDBA_EDITIONED_TYPES (『Oracle Databaseリファレンス』を参照)によってこのスキーマに対して表示されるタイプです。

エディションの有効化は遡及的で、これを元に戻すことはできません。ユーザーに対してエディションを有効化すると、このユーザーは永続的にエディション対応になります。スキーマ内のスキーマ・オブジェクト・タイプに対してエディションを有効化すると、このタイプはそのスキーマ内で永続的にエディション対応になります。エディション対応のユーザーが所有するか所有予定のオブジェクトはすべて、そのタイプがスキーマ内でエディション対応であるとともにEDITIONABLEプロパティを持つ場合、エディション・オブジェクトになります。EDITIONABLEプロパティの詳細は、[「EDITIONABLEおよびNONEDITIONABLEプロパティ」](#)を参照してください。

トピック:

- [エディション化されていない依存を持つ潜在的なエディション・オブジェクト](#)
- [エディションを有効化できないユーザー](#)

関連項目:

[CREATE USER](#)および[ALTER USER](#)の各文の完全な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

DBA_EDITIONED_TYPESの詳細は、[『Oracle Databaseリファレンス』](#)を参照してください。

DBA_USERSの詳細は、[『Oracle Databaseリファレンス』](#)を参照してください。

USER_USERSの詳細は、[『Oracle Databaseリファレンス』](#)を参照してください。

27.2.1.4.1 エディション化されていない依存を持つ潜在的なエディション・オブジェクト

潜在的なエディション・オブジェクトにエディション化されていない依存がある場合、潜在的なエディション・オブジェクトの所有者に対してエディションを有効化できるのは、次のいずれかが該当する場合のみです。

- 潜在的なエディション・オブジェクトの所有者に対してエディションを有効化すると、エディション化されていない依存がエディション化される場合。
- 次のように、FORCEを指定します。

```
ALTER USER user ENABLE EDITIONS [ FOR type [, type ]... ] FORCE;
```

前述の文では、指定されたユーザーについてエディションは有効化され、エディション・オブジェクトに対する非エディション依存オブジェクトは無効化されます。



ノート:

前述の文が、抽象データ型(ADT)を含む非エディション依存オブジェクトを無効化し、ユーザーが、無効化されたオブジェクトが依存するエディション・オブジェクトを含むエディションを削除した場合、無効化されたオブジェクトを再コンパイルすることはできません。したがって、オブジェクトは無効のままになります。

FORCEは、次の状況で便利です。ユーザーAとBに対してエディションを有効化する場合。ユーザーAが潜在的エディション・オブジェクトa1とa2の所有者である場合。ユーザーBが潜在的エディション・オブジェクトb1とb2の所有者である場合。オブジェクトa1がオブジェクトb1に依存している場合。オブジェクトb2がオブジェクトa2に依存している場合。ユーザーAとBに対してエディションを有効化させるには、次のようにします。

1. FORCEを使用して、ユーザーAについてエディションを有効化します。

```
ALTER USER A ENABLE EDITIONS FORCE;
```

これで、a1とa2はエディション・オブジェクトになり、非エディション・オブジェクトb2 (a2に依存するもの)は無効になりました。

2. ユーザーBのエディションを有効化します。

```
ALTER USER B ENABLE EDITIONS;
```

これで、b1とb2はエディション・オブジェクトになります。ただし、b2は無効のままです。

3. COMPILEで、適切なALTER文を使用して、b2を再コンパイルします。PL/SQLオブジェクトについては、REUSE SETTINGSも指定してください。

たとえば、b2がプロシージャの場合、次の文を使用します。

```
ALTER PROCEDURE b2 COMPILE REUSE SETTINGS
```

FORCEは、次の状況では必要ありません。潜在的エディション・オブジェクトc1を所有しているユーザーCに対してエディションを有効化する場合。オブジェクトc1が、ユーザーDにより所有されていて、潜在的エディション依存オブジェクトd1を持っている場合。ユーザーDが、所有者がCである依存オブジェクトを持つ、潜在的エディション・オブジェクトを所有していない場合。最初にd1をエディション・オブジェクトにして、Dに対してエディションを有効化した場合、非エディション・オブジェクトはエディション・オブジェクトに依存できないという規則に違反せずに、Cに対してエディションを有効化することができます。

関連項目:

- PL/SQLオブジェクトのALTER文の詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。
- SQLオブジェクトのALTER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。
- 依存オブジェクトの無効化の詳細は、[「依存オブジェクトの無効化」](#)を参照してください。

27.2.1.4.2 エディションを有効化できないユーザー

次のユーザーに対してはエディションを有効化できません。

- Oracleが保持しているユーザー
Oracleが保持しているユーザーでは、*_USERSビューの列ORACLE_MAINTAINEDの値はYです。
- CDBの共通ユーザー
- 1つ以上の進化したADTを所有するユーザー。

有効化しようとする、エラーORA-38820が発生します。ADTが表への依存性を持っていない場合、ALTER TYPE RESET文を使用して、バージョンを1にリセットすることにより、それ以上、表が進化するとみなされないようにすることができます(ADTのバージョンを1にリセットすると、これに依存するものは無効化されます。)

関連項目:

- CDBでの共通ユーザーの詳細は、[『Oracle Database管理者ガイド』](#)を参照してください。
- ALTER TYPE RESET文の構文は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。

27.2.1.5 EDITIONABLEおよびNONEDITIONABLEプロパティ

ノート:



データベースをリリース 11.2 からリリース 12.1 にアップグレードすると、ユーザーが作成したスキーマ内のオブジェクトが EDITIONABLE プロパティを取得し、パブリック・シノニムが NONEDITIONABLE プロパティを取得します。

データベース内でエディション対応であるスキーマ・オブジェクト・タイプに対してCREATEおよびALTER文を使用すると、作成または変更するオブジェクトをEDITIONABLEまたはNONEDITIONABLEとして指定できます。

DBMS_SQL_TRANSLATOR.CREATE_PROFILEプロシージャを使用すると、作成するSQL翻訳プロファイルをEDITIONABLEまたはNONEDITIONABLEとして指定できます。

どのオブジェクトがEDITIONABLEかを確認するには、静的データ・ディクショナリ・ビュー *_OBJECTSまたは *_OBJECTS_AEのEDITIONABLE列を参照してください。

トピック:

- [新規EDITIONABLEおよびNONEDITIONABLEオブジェクトの作成](#)
- [EDITIONABLEおよびNONEDITIONABLEオブジェクトの置換および変更](#)

関連項目:

- PL/SQLスキーマ・オブジェクトのCREATEおよびALTER文の構文とセマンティクスの詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください
- SQLスキーマ・オブジェクトのCREATEおよびALTER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
- DBMS_SQL_TRANSLATOR.CREATE_PROFILEプロシージャの詳細は、[『Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス』](#)を参照してください
- *_OBJECTSの詳細は、[『Oracle Databaseリファレンス』](#)を参照してください。
- *_OBJECTS_AEの詳細は、[『Oracle Databaseリファレンス』](#)を参照してください。

27.2.1.5.1 新規EDITIONABLEおよびNONEDITIONABLEオブジェクトの作成

データベース内でタイプがエディション対応である新規スキーマ・オブジェクトを作成する場合、EDITIONABLEまたはNONEDITIONABLEプロパティを指定できます。このプロパティを省略する場合、オブジェクトは次のいずれかでないかぎり、デ

フォルトではEDITIONABLEになります。

- PUBLIC SYNONYM (デフォルトではNONEDITIONABLEです)
- PACKAGE BODY (パッケージ仕様のプロパティを継承します)
- TYPE BODY (タイプ仕様のプロパティを継承します)

PACKAGE BODYまたはTYPE BODYでは、プロパティを指定する場合、対応するパッケージまたはタイプ仕様のプロパティと一致する必要があります。

それ自体のスキーマ内でエディション対応であるタイプのEDITIONABLEオブジェクトを作成する場合、新規オブジェクトは、オブジェクトの作成時に現行エディションであるエディションでのみ表示可能なエディション・オブジェクトです。エディション・オブジェクトの作成は、新規オブジェクトを表示できないエディションに関するライブ操作です。

NONEDITIONABLEプロパティを持つオブジェクト、またはそれ自体のスキーマ内で非エディション対応であるタイプのオブジェクトを作成する場合、新規オブジェクトは、すべてのエディションで表示可能な非エディション・オブジェクトです。

現行エディションでは、objという名前のスキーマ・オブジェクトがスキーマ内にあるが、別のエディションでは、objという名前のエディション・オブジェクトがスキーマ内にあるとします。現行エディションのスキーマでは、objという名前のオブジェクトを作成できますが、これはエディション・オブジェクトである(つまり、OBJECT_NAME、OWNERおよびEDITION_NAMEにより一意に識別される)必要があります。新規オブジェクトのタイプ(同じ名前を持つ既存のエディション・オブジェクトのタイプとは異なる場合があります)はスキーマ内でエディション対応である必要があり、新規オブジェクトはEDITIONABLEプロパティを持つ必要があります。

関連項目:

- 現行エディションの詳細は、[「現行エディションとセッション・エディション」](#)を参照してください。
- [例: エディション・オブジェクトの削除](#)
- [例: 削除済継承オブジェクトの名前を使用したオブジェクトの作成](#)

27.2.1.5.2 EDITIONABLEおよびNONEDITIONABLEオブジェクトの置換および変更

既存のオブジェクトを(CREATE OR REPLACEまたはALTER文を使用して)置換または変更する場合、次のようになります。

- スキーマがエディションに対して有効化されていない場合、オブジェクトのプロパティをEDITIONABLEからNONEDITIONABLE (またはこの逆)に変更できます。
- 置換または変更対象のオブジェクトのタイプについてスキーマがエディションに対して有効化されている場合、オブジェクトのプロパティをEDITIONABLEからNONEDITIONABLE (またはこの逆)に変更できません。

エディション・オブジェクトの変更は、変更されたオブジェクトを表示できないエディションに関するライブ操作です。

27.2.1.6 エディション・オブジェクトの規則

- 非エディション・オブジェクトは、通常は、エディション・オブジェクトに依存できません。
- 抽象データ型(ADT)をエディション化し、同時に進化させることはできません。
- エディション・オブジェクトを、FOREIGN KEY制約の始点、または終点にすることはできません。

この規則は、エディション・ビューに対してのみ影響します。エディション化されたビューとは、通常のビュー、またはエディショニング・ビューのどちらかです。

関連項目:

- [エディション・オブジェクトと非エディション・オブジェクトの名前解決](#)
- 型進化の詳細は、『[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。

27.2.2 エディションの作成

ノート:



CDB のルートではエディションを作成しないことをお勧めします。

エディションを作成するには、SQL文CREATE EDITIONを使用します。

エディションは既存のエディションの子として作成する必要があります。CREATE EDITION文によって作成される最初のエディションの親はora\$baseです。次の文によって、ora\$baseの子としてエディションe2が作成されます。

```
CREATE EDITION e2
```

(「[例: エディション・オブジェクトおよび変更時のコピー](#)」およびその他は前述の文を使用します。)

エディションの子は最大で1つです。

エディションの子孫は、エディションの子、子の子、その子などを指します。エディションの祖先は、エディションの親、親の親、その親などを指します。ルート・エディションには親はありません。また、リーフ・エディションには子はありません。

関連項目:

- 使用するための権限も含む、CREATE EDITION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。
- CDBの詳細は、『[Oracle Database管理者ガイド](#)』を参照してください。

27.2.3 エディション・オブジェクトおよび変更時のコピー

エディションを作成すると、親エディションのエディション・オブジェクトがすべてこのエディションにコピーされます。1つのエディション内のエディション・オブジェクトを変更しても、他のエディション内のこのエディション・オブジェクトのコピーには影響しません。

前述のパラグラフでは、どのような処理が行われるかを概念的に説明しています。実際には、パフォーマンスを最適化するために、Oracle Databaseがエディション・オブジェクトを祖先エディションから子孫エディションにコピーするのは、子孫エディションがオブジェクトを変更する場合のみです。この方法は変更時のコピーと呼ばれます。

子孫エディションに(実際にではなく)概念的にコピーされたエディション・オブジェクトは、継承オブジェクトと呼ばれます。子孫エディションのユーザーがDDL文で継承オブジェクトを参照すると、Oracle Databaseはオブジェクトを子孫エディションに実際にコピーします。このコピー操作は実現化と呼ばれ、子孫エディションで実オブジェクトが作成されます。

ノート:



前述のパラグラフの実現化規則には例外が1つあります。CREATE OR REPLACE object 文によって継承オブジェクトが同一オブジェクト(つまり、ソース・コードおよび設定が同じオブジェクト)に置換された場合、Oracle

Database は子孫エディションで実オブジェクトを作成しません。

27.2.3.1 例: エディション・オブジェクトおよび変更時のコピー

[例27-1](#)では、エディションora\$baseでプロシージャhelloを作成してから、ora\$baseの子としてエディションe2を作成します。e2がhelloを起動するときは、ora\$baseで継承プロシージャを起動します。このとき、e2はhelloを変更して、実オブジェクト化します。ここで、e2がhelloを起動するときは、独自の実プロシージャを起動します。エディションora\$base内のプロシージャhelloは変更されずに残ります。

例27-1 エディション・オブジェクトおよび変更時のコピー

1. 次のプロシージャがora\$baseでエディション・オブジェクトであると仮定します。

```
CREATE OR REPLACE PROCEDURE hello IS
BEGIN
    DBMS_OUTPUT.PUT_LINE('Hello, edition 1. ');
END hello;
/
```

2. ora\$baseで、プロシージャを起動します。

```
BEGIN hello(); END;
/
```

結果:

```
Hello, edition 1.
```

```
PL/SQL procedure successfully completed.
```

3. 子エディションを作成します。

```
CREATE EDITION e2;
```

概念的に、プロシージャが子エディションにコピーされ、子エディションではこのコピーのみが表示可能です。このコピーは継承オブジェクトであり、実オブジェクトではありません。

4. 子エディションを使用します。

```
ALTER SESSION SET EDITION = e2;
```

5. プロシージャを起動します。

```
BEGIN hello(); END;
/
```

概念的に、子エディションはプロシージャ(これは親エディションora\$base内のプロシージャと同等です)の独自のコピーを起動します。ただし、子エディションは実際には親エディション内のプロシージャを起動します。結果:

```
Hello, edition 1.
```

```
PL/SQL procedure successfully completed.
```

6. プロシージャを変更します。

```
CREATE OR REPLACE PROCEDURE hello IS
BEGIN
    DBMS_OUTPUT.PUT_LINE('Hello, edition 2. ');
END hello;
/
```

Oracle Databaseは子エディション内のプロシーダを実現化し、変更は、親エディション内のプロシーダではなく、子エディション内の実オブジェクトにのみ影響します。

7. プロシーダを起動します。

```
BEGIN hello(); END;  
/
```

子エディションは独自の実プロシーダを起動します。

```
Hello, edition 2.  
PL/SQL procedure successfully completed.
```

8. 親エディションに戻ります。

```
ALTER SESSION SET EDITION = ora$base;
```

9. プロシーダを起動し、これが変更されていないことを確認します。

```
BEGIN hello(); END;  
/
```

結果:

```
Hello, edition 1.  
PL/SQL procedure successfully completed.
```

関連項目:

ALTER SESSION SET EDITIONの詳細は、[「セッション・エディションの変更」](#)を参照してください。

27.2.3.2 例: エディション・オブジェクトの削除

[例27-2](#)は、エディションora\$baseでプロシーダgoodbyeを作成してから、ora\$baseの子としてエディションe2を作成します。e2でgoodbyeが削除されると、e2はgoodbyeを起動できなくなりますが、ora\$baseでは起動できます

e2でプロシーダgoodbyeが削除されたため、次のようになります。

- 子孫では、プロシーダgoodbyeが継承されません。
- goodbyeというオブジェクトはe2で表示可能でないため、e2では、goodbyeという名前を付けてオブジェクトを作成できますが、これはエディション・オブジェクトである必要があります。goodbyeという名前の新規エディション・オブジェクトをe2が作成すると、e2の子孫がこのオブジェクトを継承します。

例27-2 エディション・オブジェクトの削除

1. 次のプロシーダがora\$baseでエディション・オブジェクトであると仮定します。

```
CREATE OR REPLACE PROCEDURE goodbye IS  
BEGIN  
    DBMS_OUTPUT.PUT_LINE( 'Good-bye!' );  
END goodbye;  
/
```

2. プロシーダを起動します。

```
BEGIN goodbye; END;  
/
```

結果:

```
Good-bye!  
PL/SQL procedure successfully completed.
```

3. ora\$baseの子としてエディションe2を作成します。

```
CREATE EDITION e2;
```

e2では、プロシージャは継承オブジェクトです。

4. エディションe2を使用します。

```
ALTER SESSION SET EDITION = e2;
```

ALTER SESSION SET EDITIONはトップレベルのSQL文である必要があります。

5. e2で、プロシージャを起動します。

```
BEGIN goodbye; END;  
/
```

e2がora\$baseでプロシージャを起動します。

```
Good-bye!  
PL/SQL procedure successfully completed.
```

6. e2で、プロシージャを削除します。

```
DROP PROCEDURE goodbye;
```

7. e2で、削除したプロシージャの起動を試行します。

```
BEGIN goodbye; END;  
/
```

結果:

```
BEGIN goodbye; END;  
*  
ERROR at line 1:  
ORA-06550: line 1, column 7:  
PLS-00201: identifier 'GOODBYE' must be declared  
ORA-06550: line 1, column 7:  
PL/SQL: Statement ignored
```

8. ora\$baseに戻ります。

```
ALTER SESSION SET EDITION = ora$base;
```

9. ora\$baseで、プロシージャを起動します。

```
BEGIN goodbye; END;  
/
```

結果:

```
Good-bye!  
PL/SQL procedure successfully completed.
```


関連項目:

- 使用するための権限を含め、DROP PROCEDURE文の詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。
- [新規EDITIONABLEおよびNONEDITIONABLEオブジェクトの作成](#)
- ALTER SESSION SET EDITIONの詳細は、[「セッション・エディションの変更」](#)を参照してください。

27.2.3.3 例: 削除済継承オブジェクトの名前を使用したオブジェクトの作成

[例27-3](#)では、e2にgoodbyeという名前のファンクションが作成され、その後、e2の子としてe3という名前のエディションが作成されます。e3でプロシージャgoodbye(e2で削除)を起動しようとすると、エラーが発生しますが、e3でファンクションgoodbye(e2で作成)は正常に起動されます。

例27-3 削除済継承オブジェクトの名前を使用したオブジェクトの作成

1. e2に戻ります。

```
ALTER SESSION SET EDITION = e2;
```

2. e2で、goodbyeという名前のファンクションを作成します。

```
CREATE OR REPLACE FUNCTION goodbye  
  RETURN BOOLEAN  
IS  
BEGIN  
  RETURN(TRUE);  
END goodbye;  
/
```

このファンクションはエディション・オブジェクトである必要があります。デフォルトでは、これはEDITIONABLEプロパティを持ちます。タイプFUNCTIONがスキーマ内でエディション対応ではない場合、ALTER USER文を使用してエディション対応にする必要があります。

3. エディションe3を作成します。

```
CREATE EDITION e3 AS CHILD OF e2;
```

エディションe3がファンクションgoodbyeを継承します。

4. エディションe3を使用します。

```
ALTER SESSION SET EDITION = e3;
```

5. e3で、プロシージャgoodbyeの起動を試行します。

```
BEGIN  
  goodbye;  
END;  
/
```

結果:

```
ERROR at line 2:  
ORA-06550: line 2, column 3:  
PLS-00306: wrong number or types of arguments in call to 'GOODBYE'  
ORA-06550: line 2, column 3:  
PL/SQL: Statement ignored
```

6. e3で、ファンクションgoodbyeを起動します。

```
BEGIN
  IF goodbye THEN
    DBMS_OUTPUT.PUT_LINE( 'Good-bye!' );
  END IF;
END;
/
```

結果:

```
Good-bye!
PL/SQL procedure successfully completed.
```

関連項目:

- ALTER SESSION SET EDITIONの詳細は、[「セッション・エディションの変更」](#)を参照してください。
- ユーザーに対してエディションを有効化する手順は、[「ユーザーに対するエディションの有効化」](#)を参照してください。

27.2.4 一部のユーザーに対してエディションを使用可能にする

エディションの作成者には、エディションに対するUSE権限WITH GRANT OPTIONが自動的に付与されます。他のユーザーにエディションのUSE権限を付与するには、SQL文GRANT USE ON EDITIONを使用します。

関連項目:

GRANT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。

27.2.5 すべてのユーザーに対してエディションを使用可能にする

すべてのユーザーがエディションを使用できるようにするには、次のいずれかの手順を実行します。

- エディションのUSE権限をPUBLICに付与します。

```
GRANT USE ON EDITION edition_name TO PUBLIC
```

- このエディションを、データベースのデフォルト・エディションにします。

```
ALTER DATABASE DEFAULT EDITION = edition_name
```

これには、edition_nameのUSE権限がPUBLICに効率的に付与されるため、すべてのユーザーがエディションを使用できるという副作用があります。

関連項目:

- ALTER DATABASE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。
- GRANT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。

27.2.6 現行エディションとセッション・エディション

各データベース・セッションで一度に使用されるエディションは1つのみです。データベース・セッションがある時点で使用するエディ

ションは、現行エディションと呼ばれます。データベース・セッションが開始すると、現行エディションがセッション・エディションになります。これは、データベース・セッションが開始するときのエディションです。セッション・エディションを変更すると、現行エディションも同様に変更されます。ただし、現行エディションとセッション・エディションが異なる状況もあります。

トピック:

- [最初のセッション・エディション](#)
- [セッション・エディションの変更](#)
- [現行エディション名とセッション・エディション名の表示](#)
- [現行エディションがセッション・エディションと異なる状況](#)

27.2.6.1 最初のセッション・エディション

データベースに接続するときに、最初のセッション・エディションを指定できます。最初のセッション・エディションはデータベースのデフォルト・エディションか、USE権限を持っている任意のエディションにすることができます。使用できるエディションの名前を表示するには、次の問合せを使用します。

```
SELECT EDITION_NAME FROM ALL_EDITIONS;
```

接続時に最初のセッション・エディションをどのように指定するかは、データベースの接続方法によって異なります。インタフェースのドキュメントを参照してください。

関連項目:

- デフォルト・エディションの設定の詳細は、[『Oracle Database管理者ガイド』](#)を参照してください。
- SQL*Plusを使用するデータベースへの接続の詳細は、[『SQL*Plusユーザズ・ガイドおよびリファレンス』](#)を参照してください。
- Oracle Call Interface(OCI)を使用するデータベースへの接続の詳細は、[『Oracle Call Interfaceプログラマーズ・ガイド』](#)を参照してください。
- JDBCを使用するデータベースへの接続の詳細は、[『Oracle Database JDBC開発者ガイド』](#)を参照してください。

Oracle Database 11gリリース2 (11.2.0.2)では、接続時にセッション・エディションを指定しない場合、次のようになります。

- データベース・サービスを使用してデータベースに接続し、そのサービスで最初のセッション・エディションが指定された場合、そのサービスの最初のセッション・エディションが最初のセッション・エディションになります。
- そうでない場合、データベースのデフォルト・エディションが最初のセッション・エディションになります。

リリース11.2.0.2では、データベース・サービスの作成または変更時に、その最初のセッション・エディションを指定できます。

データベース・サービスを作成または変更するには、`srvctl add service`または`srvctl modify service`コマンドを使用することをお勧めします。サービスのデフォルトの最初のセッション・エディションを指定するには、`-edition`オプションを使用します。

また、`DBMS_SERVICE.CREATE_SERVICE`または`DBMS_SERVICE.MODIFY_SERVICE`プロシージャを使用してデータベース・サービスを作成または変更して、`EDITION`属性を使用してデフォルトの最初のセッション・エディションを指定できます。

ノート:



Oracle Database 11g リリース 2(11.2.0.1)では、DBMS_SERVICE.CREATE_SERVICE および DBMS_SERVICE.MODIFY_SERVICE プロシージャは、Oracle Clusterware および Oracle Restart により管理されるデータベースで非推奨です。

関連項目:

- srvctl add serviceコマンドの-editionオプションの詳細は、『Oracle Database管理者ガイド』を参照してください
- srvctl modify serviceコマンドの-editionオプションの詳細は、『Oracle Database管理者ガイド』を参照してください
- DBMS_SERVICE.CREATE_SERVICEプロシージャの EDITION属性の詳細は、[『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』](#)を参照してください。
- DBMS_SERVICE.MODIFY_SERVICEプロシージャの EDITION属性の詳細は、[『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。

27.2.6.2 セッション・エディションの変更

データベースに接続した後で、SQL文ALTER SESSION SET EDITIONを使用してセッション・エディションを変更することができます。セッション・エディションは、データベースのデフォルト・エディションか、USE権限のある任意のエディションに変更できます。セッション・エディションを変更すると、現行エディションも同じエディションに変更されます。

[「例: エディション・オブジェクトおよび変更時のコピー」](#)および[「例: エディション・オブジェクトの削除」](#)からの次に示す文は、セッション・エディション(および現在のエディション)を最初にe2に変更し、後でora\$baseに変更します。

```
ALTER SESSION SET EDITION = e2
...
ALTER SESSION SET EDITION = ora$base
```

ノート:



ALTER SESSION SET EDITION はトップレベルの SQL 文である必要があります。たとえば、ログオン・トリガーでエディションの変更を遅らせるには、DBMS_SESSION.SET_EDITION_DEFERRED プロシージャを使用します。

関連項目:

- ALTER SESSION SET EDITION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。
- DBMS_SESSION.SET_EDITION_DEFERREDプロシージャの詳細は、[『Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス』](#)を参照してください。

27.2.6.3 現行エディション名とセッション・エディション名の表示

次の文によって、現行エディションの名前が返されます。

```
SELECT SYS_CONTEXT('USERENV', 'CURRENT_EDITION_NAME') FROM DUAL;
```

次の文によって、セッション・エディションの名前が返されます。

```
SELECT SYS_CONTEXT('USERENV', 'SESSION_EDITION_NAME') FROM DUAL;
```

関連項目:

SYS_CONTEXT関クションの詳細は、『[Oracle Database SQL言語リファレンス](#)』を参照してください。

27.2.6.4 現行エディションがセッション・エディションと異なる状況

次のような場合に、現行エディションがセッション・エディションと異なることがあります。

- crosseditionトリガーが起動される場合。
- 例27-4のように、DBMS_SQL.[PARSE](#)プロシージャをコールして、文を実行するエディションを指定することで、文を実行する場合。

文が実行している間、指定されたエディションが現行エディションになりますが、セッション・エディションは変更されません。

[例27-4](#)では、セッション・エディションと現行エディションの名前を返す関クションが作成されます。その後、子エディションが作成され、子エディションによってその関クションが2回起動されます。1回目はセッション・エディションと現行エディションは同じです。2回目は異なります。別のエディションがパラメータとしてDBMS_SQL.PARSEプロシージャに渡されるためです。

例27-4 セッション・エディションと異なる現行エディション

1. セッション・エディションと現行エディションの名前を返す関クションを作成します。

```
CREATE OR REPLACE FUNCTION session_and_current_editions
RETURN VARCHAR2
IS
BEGIN
RETURN
  'Session: ' || SYS_CONTEXT('USERENV', 'SESSION_EDITION_NAME') ||
  ' / ' ||
  'Current: ' || SYS_CONTEXT('USERENV', 'CURRENT_EDITION_NAME');
END session_and_current_editions;
/
```

2. 子エディションを作成します。

```
CREATE EDITION e2 AS CHILD OF ora$base;
```

3. 子エディションを使用します。

```
ALTER SESSION SET EDITION = e2;
```

4. 関数を呼び出します。

```
BEGIN
  DBMS_OUTPUT.PUT_LINE (session_and_current_editions());
END;
/
```

結果:

```
Session: E2 / Current: E2
```

```
PL/SQL procedure successfully completed.
```

5. 関数をもう一度呼び出します。

```
DECLARE
  c      NUMBER := DBMS_SQL.OPEN_CURSOR();
  v      VARCHAR2(200);
  dummy  NUMBER;
  stmt   CONSTANT VARCHAR2(32767)
        := 'SELECT session_and_current_editions() FROM DUAL';
BEGIN
  DBMS_SQL.PARSE (c => c,
                  statement => stmt,
                  language_flag => DBMS_SQL.NATIVE,
                  edition => 'ora$base');

  DBMS_SQL.DEFINE_COLUMN (c, 1, v, 200);
  dummy := DBMS_SQL.EXECUTE_AND_FETCH (c, true);
  DBMS_SQL.COLUMN_VALUE (c, 1, v);
  DBMS_SQL.CLOSE_CURSOR(c);
  DBMS_OUTPUT.PUT_LINE (v);
END;
/
```

結果:

Session: E2 / Current: ORA\$BASE

PL/SQL procedure successfully completed.

関連項目:

- [crosseditionトリガーとエディションの相互作用](#)
- DBMS_SQL.PARSEプロシージャの詳細は、『[Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス](#)』を参照してください。

27.2.7 エディションのリタイア

新しいエディション(アップグレードしたアプリケーション)をすべてのユーザーが使用できるようにしたら、SYS以外のユーザーが古いエディションを使用できないようにするために、古いエディション(元のアプリケーション)をリタイアさせます。

ノート:



古いエディションがデータベースのデフォルト・エディションである場合は、この古いエディションをリタイアさせる前に、別のエディションをデフォルトにします。

```
ALTER DATABASE DEFAULT EDITION = edition_name
```

エディションをリタイアさせるには、このエディションに対するUSE権限を、すべての権限受領者から取り消します。権限受領者のリストを表示するには、次の問合せを使用します。ここで、eには削除するエディションの名前が入ります。

```
SELECT GRANTEE, PRIVILEGE
FROM DBA_TAB_PRIVS
WHERE TABLE_NAME = :e AND TYPE = 'EDITION'
/
```

エディションをリタイアさせる場合、それに応じて非エディション・オブジェクトの評価エディションおよび未使用エディションを更新しま

す。

関連項目:

- 評価エディションおよび未使用エディションの変更の詳細は、[「エディション・オブジェクトに依存可能な非エディション・オブジェクト」](#)を参照してください。
- REVOKE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。
- ALTER DATABASE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。

27.2.8 エディションの削除

ノート:



エディションに crossedition トリガーが含まれている場合は、エディションを削除する前に、[「crossedition トリガーの削除」](#)を参照してください。

エディションを削除するには、DROP EDITION文を使用します。エディションに実オブジェクトがある場合は、実オブジェクトを削除するCASCADE句を指定する必要があります。

DROP EDITION edition CASCADE文が正常終了前に(たとえば、停電などの理由で)中断された場合、静的データ・ディクショナリ・ビュー*_EDITIONSには、editionに対するUSABLEの値がNOであることが表示されます。このように使用できないeditionに対して実行できる唯一の操作は、DROP EDITION CASCADEです。

エディションは次の場合に削除します。

- アプリケーション・アップグレードをロールバックする必要がある場合。
- (オプション)エディションをリタイアさせた場合。

エディションを削除できるのは、次のすべての条件に該当する場合のみです。

- エディションがルート・エディション、またはリーフ・エディションである場合。
- エディションは使用されていません。(つまり、セッションの現行エディションまたはセッション・エディションではない場合。)
- このエディションが、データベースのデフォルト・エディションではない場合。

エディションがルートであり、COMPATIBLEパラメータが12.2.0以上に設定されている場合、エディションは使用不可としてマークされます。対象オブジェクトは、使用可能な子孫エディションで継承されなくなったエディション・オブジェクトです。使用不可のエディション内の各対象オブジェクトは、自動的にスケジュールされたメンテナンス・プロセスによって削除されます。使用不可のルート・エディションに残っているオブジェクトがなくなると、エディション自体が自動的に削除されます。このクリーン・アップ・プロセスが、検出された各使用禁止ルート・エディションごとに繰り返されます。ユーザーは、DBMS_EDITIONS_UTILITIES.CLEAN_UNUSABLE_EDITIONSプロシージャ([Oracle Database PL/SQLパッケージおよびタイプ・リファレンス](#)を参照)を手動で実行することで、このプロセスをオンデマンドで実行できます。

COMPATIBLEが12.1.0以下に設定されている場合、ルート・エディションは、その子孫が継承するオブジェクトを持っている必要はありません。ルート・エディションから継承された各オブジェクトは、エディションを削除する前に、実オブジェクト化するか、明示的に削除する必要があります。

エディションがリーフの場合はリーフ内のすべてのエディション・オブジェクトが削除され、その後、文の実行が完了する前に、エディ

ション自体も削除されます。

ノート:

エディションを削除した後、次の両方の条件に該当する場合、エディション・オブジェクトに依存する非エディション・オブジェクトを再コンパイルすることはできません。



- 非エディション・オブジェクトに ADT が含まれている。
- 非エディション・オブジェクトが依存するエディション・オブジェクトの所有者が、FORCE を使用してエディションに対して有効化されたときに、非エディション・オブジェクトが無効化されていた。

子エディションで継承オブジェクトを明示的に実オブジェクト化するには、次のようにします。

1. 子エディションをセッション・エディションにします。

手順は、[「セッション・エディションの変更」](#)を参照してください。

2. COMPILEで、適切なALTER文を使用して、このオブジェクトを再コンパイルします。PL/SQLオブジェクトについては、REUSE SETTINGSも指定してください。

たとえば、次の文ではプロシージャp1が実オブジェクト化されます。

```
ALTER PROCEDURE p1 COMPILE REUSE SETTINGS
```

エディションを削除する場合、それに応じて非エディション・オブジェクトの評価エディションおよび未使用エディションを更新します。

関連項目:

- ALTER LIBRARY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。
- ALTER VIEW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。
- ALTER FUNCTION文の詳細は、[Oracle Database PL/SQLリファレンス](#)を参照
- ALTER PACKAGE文の詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。
- ALTER PROCEDURE文の詳細は、[Oracle Database PL/SQL言語リファレンス](#)を参照
- ALTER TRIGGER文の詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。
- ALTER TYPE文の詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。
- DROP EDITION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。
- PL/SQLオブジェクトのALTER文の詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。
- SQLオブジェクトのALTER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。
- 評価エディションおよび未使用エディションの変更の詳細は、[エディション・オブジェクトに依存可能な非エディション・オブジェクト](#)を参照してください。

27.3 エディショニング・ビュー

エディショニング・ビュー以外のビューで定義できるトリガーのタイプは、INSTEAD OFトリガーのみです。エディショニング・ビューでは、表で定義できるトリガー・タイプすべてを定義できます(ただし、crosseditionトリガー(一時トリガー)とINSTEAD OF トリガーは例外です)。このため、またエディショニング・ビューはエディション化が可能であることによって、エディショニング・ビューを使用することで、実表がエディション化されたかのように扱うことができます。ただし、アップグレードされたアプリケーションにより、新しい索引または制約が必要とされる場合、エディショニング・ビューに索引または制約を追加することはできません。これらは実表に追加する必要があります。

ノート:



実表または実表上の索引を変更する場合は、「[非ブロック化およびブロック化の DDL 文](#)」を参照してください。

エディショニング・ビューは1つの実表の列のサブセットを選択し、オプションとしてそれらの列に別名を提供します。別名を提供するとき、エディショニング・ビューは物理列名(実表で使用される)を論理列名(アプリケーションで使用される)にマップします。エディショニング・ビューは、表のAPIに似ています。

直接ではなく、エディショニング・ビュー経由で表にアクセスすることに対するパフォーマンス・ペナルティはありません。つまり、SQL SELECT、INSERT、UPDATE、DELETE、またはMERGE文で1つ以上のエディショニング・ビューが1回以上使用されているときに、必要に応じて、エディショニング・ビューの名前を実表の名前で置き換え、列名を調整した場合でも、パフォーマンスは変わりません。

静的データ・ディクショナリ・ビュー*_[_EDITIONING_VIEWS](#)は、セッション・エディションで表示可能なデータベース内のすべてのエディショニング・ビューを示します。*_[_EDITIONING_VIEWS_AE](#)は、すべてのエディションにおけるデータベース内のすべてのエディショニング・ビューのすべての実オブジェクトを示します。

トピック:

- [エディショニング・ビューの作成](#)
- [パーティション拡張エディショニング・ビュー名](#)
- [エディショニング・ビューの書き込み可能性の変更](#)
- [エディショニング・ビューの置換](#)
- [削除または名前変更された実表](#)
- [実表への索引と制約の追加](#)
- [SQLオプティマイザ索引ヒント](#)

関連項目:

静的データ・ディクショナリ・ビュー*_[_EDITIONING_VIEWS](#)および*_[_EDITIONING_VIEWS_AE](#)の詳細は、『Oracle Databaseリファレンス』を参照してください。

27.3.1 エディショニング・ビューの作成

エディショニング・ビューを作成する前に、所有者がエディション対応であり、スキーマ・オブジェクト・タイプVIEWが所有者でエディション対応である必要があります。

エディショニング・ビューを作成するには、SQL文CREATE VIEWをキーワードEDITIONINGとともに使用します。エディショニング・ビューを読み取り専用にするには、WITH READ ONLYを指定します。読み取り/書込みにするにはWITH READ ONLYを省略します。NONEDITIONABLEは指定しないでください。指定すると、エラーが発生します。

エディショニング・ビューが読み取り専用の場合、未変更アプリケーションのユーザーが実表のデータを見ることはできますが、変更することはできません。この実表は準可用性を備えています。準可用性を利用できるのは、ユーザーが読むだけで変更しない、オンライン・ディクショナリなどのアプリケーションです。実表にcrosseditionトリガーを定義しない場合は、エディショニング・ビューを読み取り専用にします。

エディショニング・ビューが読み取り/書込みの場合、未変更アプリケーションのユーザーが、実表のデータを見ることも変更することもできます。この実表は最大可用性を備えています。最大可用性が必要となるのは、ユーザーが注文を発行するオンライン・ストアなどのアプリケーションです。実表にcrosseditionトリガーを定義する場合は、エディショニング・ビューを読み取り/書込みにします。

エディショニング・ビューは、実表からの列のサブセットの選択と列の別名提供の他にも機能を実行する必要があるため、エディショニング・ビューを作成するCREATE VIEW文には制限があります。制限に違反すると、FORCEを指定していてもビューの作成が失敗します。

関連項目:

- 制限を含む、CREATE VIEW文を使用したエディショニング・ビューの作成の詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。
- [ユーザーに対するエディションの有効化](#)

27.3.2 パーティション拡張エディショニング・ビュー名

パーティション表に定義されているエディショニング・ビューには、パーティション拡張名を付けることができます。この名前は、実表のパーティションおよびサブパーティションを示すパーティション名とサブパーティション名を含みます。

パーティション拡張表名に対応するデータ操作言語(DML)は、パーティション拡張エディショニング・ビュー名にも対応します。次の文が対応しています。

- DELETE
- INSERT
- SELECT
- UPDATE

関連項目:

パーティション表の参照の詳細は、[『Oracle Database SQL言語リファレンス』](#)を参照してください。

27.3.3 エディショニング・ビューの書込み可能性の変更

既存のエディショニング・ビューを読み取り専用から読み取り/書込みに変更するには、SQL文ALTER VIEW READ WRITEを使用します。既存のエディショニング・ビューを読み取り/書込みから読み取り専用に変更するには、SQL文ALTER VIEW READ ONLYを使用します。

関連項目:

ALTER VIEW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。

27.3.4 エディショニング・ビューの置換

エディショニング・ビューを置換するには、SQL文CREATE VIEWをOR REPLACE句およびキーワードEDITIONINGとともに使用します。

エディショニング・ビューを置換できるのは別のエディショニング・ビューのみです。置換対象のエディショニング・ビューに定義されているトリガーはすべて保持されます。

27.3.5 削除または名前変更された実表

エディショニング・ビューの定義に基づいている実表を削除したり名前を変更したりしても、エディショニング・ビューは削除されませんが、エディショニング・ビューとその依存オブジェクトが無効になります。ただし、エディショニング・ビューに定義されているトリガーはすべて保持されます。

27.3.6 実表への索引と制約の追加

アップグレードされたアプリケーションにより、新しい索引または制約が必要とされる場合は、これらを実表に追加する必要があります。これらをエディショニング・ビューに追加することはできません。

新しい索引が古いエディション(元のアプリケーション)に悪い影響を与える場合は、これらを非表示にします。新しい索引の使用が必要なcrosseditionトリガーで、これらをINDEXヒントに指定します。

すべてのユーザーがアップグレードされたアプリケーションのみを使用している場合は、次のようにします。

- 新しい索引を使用しているのがcrosseditionトリガーのみである場合は、これらの索引を削除します。
- アップグレードされたアプリケーションが新しい索引で便利である場合は、これらを表示します。

関連項目:

- [索引を管理するためのガイドライン](#)
- 索引ヒントの詳細は、『[Oracle Database SQL言語リファレンス](#)』を参照してください。
- [SQLオプティマイザ索引ヒント](#)

27.3.7 SQLオプティマイザ索引ヒント

SQLオプティマイザ索引ヒントは、索引を構成する列の論理名を使用して指定されます。論理列名を使用してエディショニング・ビューに指定されたSQLオプティマイザ索引ヒントは、実表の対応する物理列の索引にマップする必要があります。

関連項目:

ヒントの使用方法の詳細は、『[Oracle Database SQL言語リファレンス](#)』を参照してください。

27.4 crosseditionトリガー

crosseditionトリガーと非crosseditionトリガーの最も大きな違いは、エディションとの相互作用の方法です。crosseditionトリガーは、それが実であるエディションにのみ表示されます。子孫エディションでは表示されません。forward crosseditionトリガーは、データを古いエディションが使用していた列から、新しいエディションが使用している列に移動します。reverse crosseditionトリガーは逆方向へデータを移動します。

その他の重要な違いは次のとおりです。

- crosseditionトリガーは、他の表に定義されたトリガーに対して並べ替えることができますが、非crosseditionトリガーは同じ表に定義されたトリガーに対してのみ並べ替えることができます。
- crosseditionトリガーは一時的に使用します。構成を変更した表をすべてのユーザーが使用できるようにした後で削除します。

トピック:

- [forward crosseditionトリガー](#)
- [reverse crosseditionトリガー](#)
- [crosseditionトリガーとエディションの相互作用](#)
- [crosseditionトリガーの作成](#)
- [アップグレード前の表現をアップグレード後の表現に変換](#)
- [crosseditionトリガーの削除](#)

関連項目:

トリガーの詳細は、[Oracle Database PL/SQL言語リファレンス](#)を参照

27.4.1 forward crosseditionトリガー

アップグレード後のエディションでDML変更を表に行うと、新しい列または新しい表のみに書き込まれ、アップグレード前(祖先)エディションのユーザーが読取りまたは書き込みを行う可能性がある列に対しては行われません。ただし、祖先エディションのユーザーが表データを変更する場合、表示しているエディショニング・ビューがそれらの変更内容を正確に反映する必要があります。これは、forward crosseditionトリガーを使用することで実現します。

forward crosseditionトリガーはトランスフォームを定義します。これは、古い行を1つ以上の新しい行に変換するための規則です。古い行は、アップグレード前の表現に含まれるデータ行です。新しい行は、アップグレード後の表現に含まれるデータ行です。トリガーの名前はトリガー自体、およびこのトリガーにより定義された変換を表します。

27.4.2 reverse crosseditionトリガー

アップグレード前のエディションとアップグレード後のエディションが同時に通常使用(ホット・ロールオーバー)されている場合、reverse crosseditionトリガーを使用して、アップグレード後エディションのユーザーが表データを変更するときに、変更内容がアップグレード前エディションに正確に反映されるようにします。

27.4.3 crosseditionトリガーとエディションの相互作用

crosseditionトリガーと非crosseditionトリガーの最も大きな違いは、エディションとの相互作用の方法です。

ここでは、現行エディションは、トリガーを起動するDML文が実行されるエディションを指します。現行エディションがセッション・エディションと異なることがあります。

トピック:

- [表示可能なトリガー](#)
- [起動できるトリガーの種類](#)
- [起動順序](#)
- [crosseditionトリガーの実行](#)

関連項目:

[現行エディションがセッション・エディションと異なる状況](#)

27.4.3.1 表示可能なトリガー

エディションは、非crosseditionトリガーを他のエディション・オブジェクトと同じように継承します([「エディション・オブジェクトおよび変更時のコピー」](#)を参照)。

エディションはcrosseditionトリガーは継承しません。crosseditionトリガーは、別のエディションで実行されるDML文に対応して起動されることもあります。名前が表示されるのは作成されたエディションのみです。したがって、エディションでは、祖先エディションで作成されたcrosseditionトリガーの名前を再利用できます。crosseditionトリガーの名前を再利用しても、古いトリガーの起動条件は変更されません。

静的データ・ディクショナリ・ビューに示されるcrosseditionトリガーは、現行エディションの実オブジェクトです。

27.4.3.2 起動できるトリガーの種類

起動できるトリガーの種類は、トリガーするDML文のカテゴリによって異なります。

カテゴリ:

- [forward crosseditionトリガー-SQL](#)
- [reverse crosseditionトリガー-SQL](#)
- [アプリケーションSQL](#)

ノート:

SQL INSERT 文に APPEND ヒントを指定しても crossedition トリガーの起動を防ぐことはできません。



関連項目:

APPEND ヒントの詳細は、[『Oracle Database SQL 言語リファレンス』](#)を参照してください。

27.4.3.2.1 forward crosseditionトリガー-SQL

forward crosseditionトリガー-SQLは、次のいずれかの方法で実行されるSQLです。

- forward crosseditionトリガー本体から直接起動

forward crosseditionトリガーのローカル・サブプログラムが起動された場合のみ、このカテゴリには、起動されたサブプログラムのSQLが含まれます。

- apply_crossedition_triggerパラメータに対してNULL以外の値を指定してDBMS_SQL.PARSEプロシージャを起動

apply_crossedition_triggerパラメータに対して唯一有効なNULL以外の値は、forward crosseditionトリガーの非修飾名です。

forward crosseditionトリガーが別のコンパイル・ユニットに含まれるサブプログラムを起動する場合、このトリガーが apply_crossedition_triggerパラメータにNULL以外の値が指定されたDBMS_SQL.PARSEプロシージャにより起動された場合のみ、このサブプログラムのSQLがforward crosseditionトリガーSQLになります。

forward crosseditionトリガーSQLは、次の条件をすべて満たすトリガーのみ起動できます。

- forward crosseditionトリガーである。
- 現行エディションまたは現行エディションの子孫で作成された。
- 実行中のforward crosseditionトリガーに明示的に後続する。

関連項目:

DBMS_SQL.PARSEプロシージャの詳細は、[『Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス』](#)を参照してください。

27.4.3.2.2 reverse crosseditionトリガーSQL

reverse crosseditionトリガーSQLは、reverse crosseditionトリガー本体から直接実行されるSQLです。reverse crosseditionトリガーのローカル・サブプログラムが起動された場合のみ、このカテゴリには、起動されたサブプログラムのSQLが含まれます。

reverse crosseditionトリガーSQLは、次の条件をすべて満たすトリガーのみ起動できます。

- reverse crosseditionトリガーである。
- 現行エディションまたは現行エディションの先祖で作成された。
- 実行中のreverse crosseditionトリガーよりも前に明示的に実行される。

27.4.3.2.3 アプリケーションSQL

アプリケーションSQLは、crosseditionトリガーSQLを除くすべてのSQLで、次のDML文が含まれます。

- DBMS_SQLパッケージを使用してコーディングされた動的SQL DML文。
- Javaストアド・プロシージャおよび外部プロシージャによって実行されるDML文(これらのプロシージャがCALLトリガーで起動された場合も含む)

アプリケーションSQLは、次の規則に従って、noncrosseditionトリガーとcrosseditionトリガーの両方を起動します。

トリガーの種類	トリガーの起動条件

トリガーの種類	トリガーの起動条件
noncrossedition	トリガーが現行エディションで表示可能かつ有効である。
forward crossedition	トリガーが現行エディションの子孫で作成された。
reverse crossedition	トリガーが現行エディションまたは現行エディションの祖先で作成された。

関連項目:

DBMS_SQLパッケージの詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。

27.4.3.3 起動順序

特定のDML文に対応してトリガーを起動するためのトリガーの条件は次のとおりです。

- 適切な種類である
- 選択の条件を満たしている(たとえば、DML文の種類やWHEN句)
- 有効化されている

これらの提供を満たすトリガーの場合、起動順序は、FOLLOWSおよびPRECEDES句、トリガーの種類、およびエディションに依存します。

トピック:

- [FOLLOWS句およびPRECEDES句](#)
- [トリガーの種類とエディション](#)

関連項目:

- トリガーの起動順序に関する概要は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。
- [起動できるトリガーの種類](#)

27.4.3.3.1 FOLLOWS句およびPRECEDES句

トリガーAとBが同じタイミング・ポイントで起動されるとき、次のいずれかに該当するとAはBよりも先に起動します。

- Aが明示的にBに先行する場合。
- Bが明示的にAに続く場合。

この規則は次のような条件とは無関係です。

- トリガーが有効か無効か。
- UPDATE OF句に指定された列が変更されるかどうか。
- WHEN句が満たされるかどうか。
- 2つのトリガーが同じ種類のDML文(INSERT、UPDATEまたはDELETE)に関連付けられているかどうか。

- トリガーのタイミング・ポイントが重なっているかどうか。

明示的な先行または後続の関係がないトリガーの起動順序は予測できません。

27.4.3.3.2 トリガーの種類とエディション

トリガーを起動するDML文に関連付けられた各タイミング・ポイントでは、条件に合うトリガーが次の順序で起動されます。[1](#)から[3](#)ではFOLLOWSの関係が適用され、[4](#)と[5](#)ではPRECEDESの関係が適用されます。

1. crosseditionトリガー以外のトリガー
2. 現行エディションで作成されたforward crosseditionトリガー
3. 現行エディションの子孫で作成されたforward crosseditionトリガー(子孫の作成順(子、孫、その子など))
4. 現行エディションで作成されたreverse crosseditionトリガー
5. 現行エディションの祖先で作成されたreverse crosseditionトリガー(祖先の作成と逆順(親、その親、さらにその親など))

27.4.3.4 crosseditionトリガーの実行

crosseditionトリガーは、自らが作成されたエディションを使用して実行します。crosseditionトリガーがコールするすべてのコード(パッケージ参照、PL/SQLサブプログラム・コールおよびSQL文を含む)も、crosseditionトリガーが作成されたエディションで実行します。

PL/SQLパッケージが複数のエディションで実パッケージである場合、単一セッション内であっても、パッケージの変数やその他の状態は各エディションのみで公開されます。各crosseditionトリガーおよびそれらのトリガーがコールするコードは、crosseditionトリガーが作成されたエディションを使用して実行するため、同一セッションがパッケージの複数のバージョンを同じ名前でインスタンス化することがあります。

27.4.4 crosseditionトリガーの作成

crosseditionトリガーを作成する前に、所有者がエディション対応であり、スキーマ・オブジェクト・タイプTRIGGERが所有者でエディション対応である必要があります。(手順は、[「ユーザーに対するエディションの有効化」](#)を参照してください。)

次の規則に注意しながらSQL文CREATE TRIGGERを使用して、crosseditionトリガーを作成します。

- crosseditionトリガーは、ビューではなく表に定義する必要があります。
- crosseditionトリガーには、EDITIONABLEプロパティが必要です。
- crosseditionトリガーはDMLトリガー(単純トリガーまたは複合トリガー)であることが必要です。
crosseditionトリガー本体でのDML文としては、静的SQL文またはネイティブ動的SQL文のいずれかを使用できます。
- crosseditionトリガーは、REVERSEを指定しないかぎりforwardになります。(FORWARDの指定はオプションです。)
- FOLLOWS句を使用できるのは、forward crosseditionトリガーまたは非crosseditionトリガーを作成する場合のみです。(FOLLOWS句は、作成中のトリガーが、指定したトリガーの起動後に起動することを指定します。)
- PRECEDES句を使用できるのは、reverse crosseditionトリガーを作成する場合のみです。(PRECEDES句は、作成中のトリガーが、指定したトリガーの起動前に起動することを指定します。)
- FOLLOWS句またはPRECEDES句に指定するトリガーは存在している必要がありますが、有効になっていなくても、正常にコンパイルされていなくてもかまいません。

- 非crosseditionトリガーと同じく、crosseditionトリガーは、DISABLEを指定しないかぎり作成された時点で有効になっています。(ENABLEの指定はオプションです。)

ヒント:



crossedition トリガーは無効な状態で作成して、正常にコンパイルされたことを確認した後で有効にします。有効な状態で作成してからコンパイルが失敗すると、既存のアプリケーションのユーザーの操作が影響を受けます。

- crosseditionトリガー本体の操作は多重呼出し不変(操作を複数回実行するのは冗長であり、結果は変わらない)である必要があります。

関連項目:

- crosseditionトリガーを作成するCREATE TRIGGER文の使用方法的詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。
- 静的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。
- ネイティブ動的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。

27.4.4.1 forward crosseditionトリガー本体のコーディング

forward crosseditionトリガーの本体の操作は、予測が不可能であるため、多重呼出し不変である必要があります。

- 古い行に対して本体が最初に実行されるコンテキスト。

可能性は次のとおりです。

- 祖先エディションのユーザーが、トリガー(予期せぬ変更)を起動するDML文を実行するとき
- トリガーが定義する変換を適用するとき(古いフォーマットから新しいフォーマットに行を一括アップグレードします)
- 本体が古い行それぞれについて何回実行されるか。

トピック:

- [データ変換における衝突の処理](#)
- [他の表への変更の処理](#)

関連項目:

変換の適用の詳細は、[「アップグレード前の表現をアップグレード後の表現に変換」](#)を参照してください。

27.4.4.1.1 データ変換における衝突の処理

forward crosseditionトリガーが(表の新しい列ではなく)新しい表を移入する場合、その本体は必ずデータ変換の衝突に対応します。

たとえば、新しい表のある列にUNIQUE制約が含まれるとします。予期せぬ変更によりforward crosseditionトリガーが起動され、これにより新しい表に1行挿入されます。その後、別の予期せぬ変更により、forward crosseditionトリガーが起動され

るか、またはユーザーがトリガーにより定義された変換を適用します。このトリガーは新しい表への行の挿入を試行しますが、これはUNIQUE制約に違反しています。

衝突処理方法がトリガーの実行理由に応じて変わる場合、その理由はファンクション `APPLYING_CROSSEDITION_TRIGGER` により判断できます。このファンクションがトリガーの本体から直接呼び出された場合、変換を一括で適用しているためにトリガーが実行されている場合は `BOOLEAN` 値の `TRUE`、予期せぬ変更によりトリガーが実行されている場合は `FALSE` を返します。(`APPLYING_CROSSEDITION_TRIGGER` は、パッケージ `DBMS_STANDARD` で定義されています。これにはパラメータはありません。) これにはパラメータはありません。)

衝突を無視し、既存の行と衝突しない行を挿入するには、`INSERT` 文に `IGNORE_ROW_ON_DUPKEY_INDEX` ヒントを入れます。

このような衝突を無視したくないが、これらに対応するために、衝突がどこで発生しているかを知る必要があるという場合は、`INSERT` または `UPDATE` 文に `CHANGE_DUPKEY_ERROR_INDEX` ヒントを入れ、索引または列のセットを指定します。その後、索引、または列のセットに一意キーの違反が発生した場合、`ORA-00001` のかわりに `ORA-38911` が報告されます。`ORA-38911` に対する例外ハンドラを書くことができます。

ノート:



ヒントの構文もありますが、`IGNORE_ROW_ON_DUPKEY_INDEX` および `CHANGE_DUPKEY_ERROR_INDEX` は必須です。これらはオプティマイザにより必ず使用されます。

[例27-5](#)は `APPLYING_CROSSEDITION_TRIGGER` ファンクション、および `IGNORE_ROW_ON_DUPKEY_INDEX` ヒントと `CHANGE_DUPKEY_ERROR_INDEX` ヒントを使用して、データ変換の衝突に対応する `crossedition` トリガーを作成します。このトリガーは `table1` の古い行を `table2` の新しい行に変換します。この表を次のように作成されます。

```
CREATE TABLE table1 (key NUMBER, value VARCHAR2(20));
CREATE TABLE table2 (key NUMBER, value VARCHAR2(20), last_updated TIMESTAMP);
CREATE UNIQUE INDEX i2 on table2(key);
```

関連項目:

- `IGNORE_ROW_ON_DUPKEY_INDEX` の詳細は、[『Oracle Database SQL言語リファレンス』](#) を参照してください。
- `CHANGE_DUPKEY_ERROR_INDEX` の詳細は、[『Oracle Database SQL言語リファレンス』](#) を参照してください。
- ヒントの概要は、[『Oracle Database SQL言語リファレンス』](#) を参照してください。

例27-5 データ変換の衝突に対応する `crossedition` トリガー

```
CREATE OR REPLACE TRIGGER trigger1
BEFORE INSERT OR UPDATE ON table1
FOR EACH ROW
CROSSEDITION
DECLARE
    row_already_present EXCEPTION;
    PRAGMA EXCEPTION_INIT(row_already_present, -38911);
BEGIN
    IF APPLYING_CROSSEDITION_TRIGGER THEN
        /* The trigger is running because of applying the transform.
           If the old edition of the app has already caused this trigger
           to insert a row, we do not modify the row as part of applying
           the transform. Therefore, insert the new row into table2 only if
           it is not already there. */
```

```

INSERT /*+ IGNORE_ROW_ON_DUPKEY_INDEX(table2(key)) */
INTO table2
VALUES(:new.key, :new.value, to_date('1900-01-01', 'YYYY-MM-DD'));
ELSE
/* The trigger is running because of a serendipitous change.
If no previous run of the trigger has already inserted
the corresponding row into table2, insert the new row;
otherwise, update the previously inserted row. */
BEGIN
    INSERT /*+ CHANGE_DUPKEY_ERROR_INDEX(table2(key)) */
    INTO table2
    VALUES(:new.key, :new.value, SYSTIMESTAMP);
EXCEPTION WHEN row_already_present THEN
    UPDATE table2
    SET value = :new.value, last_updated = SYSTIMESTAMP
    WHERE key = :new.key;
END;
END IF;
END;
/

```

27.4.4.1.2 他の表への変更の処理

トリガーが定義された表以外の表を変更する明示的なSQL文がforward crosseditionトリガーの本体に含まれ、それらの表の行にトリガーが定義された表の行との1対1の対応がない場合、次の状況を正しく処理するためのロック・メカニズムを本体コードに実装する必要があります。

- 祖先エディションの複数のユーザーが、トリガーの定義されている表に対するDML文を同時に発行する状況。
- 祖先エディションのユーザーのうち、少なくとも1人が、トリガーの定義されている表に対するDML文を発行する状況。

27.4.5 アップグレード前の表現をアップグレード後の表現に変換

アップグレードしているアプリケーションを構成するデータベース・オブジェクトを(新しいエディションで)再定義した後、アプリケーション・データを(古いエディションで)アップグレード前の表現から(新しいエディションで)アップグレード後の表現に変換する必要があります。この変換の規則はトランスフォームと呼ばれ、これらはforward crosseditionトリガーにより定義されます。

古い行の一部は、予期せぬ変更、つまり、forward crosseditionトリガーが起動したアップグレード前のアプリケーションのユーザーが行った変更により、新しい行にトランスフォームされている可能性があります。しかし、予期せぬ変更によりトランスフォームされなかった行はすべてアップグレード前の表現のままです。古い行がすべて新しい行に確実にトランスフォームされるようにするには、アプリケーション・データに格納されている表で定義したトランスフォームを適用する必要があります。

トランスフォームを適用する方法には次の3通りがあります。

- トランスフォームを定義しているトリガーを、表のすべての行に対して一度に1行ずつ起動します。
- トリガーを起動するかわりに、トリガーで行われる内容をより速く処理するSQL文を実行し、そのトリガーに続くトリガーがあれば起動します。

表全体を置換した場合、または新しい表を作成した場合は、最初の方法より2番目の方法をお勧めします。

- プロシージャDBMS_EDITIONS_UTILITIES.SET_NULL_COLUMN_VALUES_TO_EXPRを起動し、メタデータ操作を使用して、トランスフォームを新規列に適用します。

この3番目の方法の場合、インストール時間は最も短くなりますが、トランスフォームを表す式に関する制限があり、メタデータが実データによって置換されるまで新規列の問合せが遅くなります。

次の場所において、メタデータは実データによって置換されます。

- 更新される個別列要素内。
- オンライン表再定義を使用して表が圧縮されるすべての列要素内。

。

トランスフォームを適用する最初の2つの方法の場合、DBMS_SQL.PARSEプロシージャ、またはDBMS_PARALLEL_EXECUTEパッケージのサブプログラムを起動します。大量のデータがある場合は2番目の方法をお勧めします。このサブプログラムを使用すると、大容量の表のデータを並列で増分的に更新することができます。これは次の2つのステップで行われます。

1. 表の行をグループ化して小さなチャンクに分けます。
2. 必要なUPDATE文を並列でチャンクに適用し、1チャンクの処理が終了するたびにコミットします。

この利点は次のとおりです。

- 表全体のロックではなく、相対的に短い時間で一度に1セットの行のみをロックする。
- 操作全体が完了する前に問題が発生してもそれまでに処理した内容が失われない。

DBMS_SQL.PARSEプロシージャとDBMS_PARALLEL_EXECUTEサブプログラムの両方で、`apply_crossedition_trigger`、`fire_apply_trigger`および`sql_stmt`の実際のパラメータ値は次に示すように同じです。

- `apply_crossedition_trigger`には、適用されるトランスフォームを定義するforward crosseditionトリガーの名前を指定します。
- 表のすべての行に対して一度に1行ずつトリガーを起動するには、次のようにします。
 - `fire_apply_trigger`の値にTRUEを指定します。
 - `sql_stmt`には、起動するforward crosseditionトリガーの選択以外に重要な影響を及ぼさないSQL文を指定します。たとえば、一部の列を各行の既存の値に設定するUPDATE文を指定します。
- トリガーで行われる内容を処理するSQL文を実行し、そのトリガーに続くトリガーがあれば起動します。
 - `fire_apply_trigger`の値にFALSEを指定します。
 - `sql_stmt`には、forward crosseditionトリガーの処理内容をより速く行うSQL文を指定します。たとえば、1つ以上のPL/SQLサブプログラムをコールするPL/SQL無名ブロックを指定します。

関連項目:

- DBMS_SQL.PARSEプロシージャの詳細は、[『Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス』](#)を参照してください。
- DBMS_PARALLEL_EXECUTE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。
- forward crosseditionトリガーの詳細は、[「forward crosseditionトリガー」](#)を参照してください。
- DBMS_EDITIONS_UTILITIES.SET_NULL_COLUMN_VALUES_TO_EXPRプロシージャの詳細は、[『Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス』](#)を参照してください。

27.4.5.1 更新内容の消失の防止

トランスフォームの適用中に更新内容が消失するのを防ぐには、次の手順を使用します。

1. crosseditionトリガーを有効化します。
2. 影響を受ける表への保留中の変更がコミットまたはロールバックされるまで待機します。

DBMS_UTILITY.WAIT_ON_PENDING_DMLプロシーダを使用します。

3. トランスフォームを適用します。

ノート:

このシナリオでは、forward crossedition トリガーが変更するのはそれが定義された表のみですが、リスクを説明するには十分です。crossedition トリガーがまだ有効になっていないときに、セッション 1 が UPDATE 文を表に対して発行し、次にセッション 2 が crossedition トリガーを有効にし、すぐにトランスフォームを適用するとします。

セッション 1 とセッション 2 の両方が同じ行(行 n)を変更しようすると競合状態が発生する可能性があります。どちらのセッションが先に行 n に到達するかは運次第です。最初に到達したセッションがその変更をコミットしてロックを解放するまで、2 番目に行 n に到達したセッションは待機する必要があるとしても、両方の更新は成功します。

問題は、セッション 2 が競合に勝った場合に起こります。その SQL 文はトリガーが有効になった後にコンパイルされたため、その文を実装するプログラムはトリガー・アクションも実装します。そのため、目的のアップグレード後の列の値は列 n に設定されます。ここでセッション 1 が行 n に到達しました。この SQL 文はトリガーが有効になる前にコンパイルされたため、この文を実装するプログラムはトリガー・アクションを実装しません。そのため、セッション 2 がアップグレード後の列に設定した値は変更されず、セッション 1 が行 n を更新する前のソース列の値を反映します。つまり、セッション 1 の更新の目的の副作用が消失します。

関連項目:

DBMS_UTILITY.WAIT_ON_PENDING_DMLプロシーダの詳細は、『[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。

27.4.6 crosseditionトリガーの削除

crosseditionトリガーを削除するには、DROP TRIGGER文を使用します。また、DROP EDITION文でCASCADE句を使用して、これらが実であるエディションを削除することにより、crosseditionトリガーを削除できます。

crosseditionトリガーは次の場合に削除します。

- アプリケーション・アップグレードをロールバックしている(アップグレード後のエディションを削除している)場合。
アップグレード後のエディションを削除する前に、新しい列の制約をすべて無効化、または削除する必要があります。
- アプリケーションのアップグレードが完了し、すべてのユーザーがアップグレード後のエディションを使用できるようにしている場合。

すべてのセッションでアップグレード後のエディションが使用されている場合は、forward crosseditionトリガーを削除できます。ただし、reverse crosseditionトリガーを削除する前に、古い列の制約をすべて無効化、または削除する

必要があります。

制約を無効化または削除するには、ALTER TABLE文にDISABLE CONSTRAINTまたはDROP CONSTRAINT句を付けて使用します。

関連項目:

- DROP TRIGGER文の詳細は、『[Oracle Database PL/SQL言語リファレンス](#)』を参照してください。
- エディションの削除の詳細は、『[エディションの削除](#)』を参照してください。
- ALTER TABLE文の詳細は、『[Oracle Database SQL言語リファレンス](#)』を参照してください。

27.5 EBR機能に関する情報の表示

トピック:

- [エディションに関する情報の表示](#)
- [エディションニング・ビューに関する情報の表示](#)
- [crosseditionトリガーに関する情報の表示](#)

27.5.1 エディションに関する情報の表示

[表27-1](#)は、エディションに関する情報を表示する静的データ・ディクショナリ・ビューについて簡単に説明しています。

表27-1 エディション情報を含む*_ディクショナリ・ビュー

ビュー	説明:
*_EDITIONS	データベースのすべてのエディションを示します。
*_EDITION_COMMENTS	データベースのすべてのエディションに関連付けられたコメントを示します。
*_EDITIONED_TYPES	デフォルトで各スキーマ内でエディション化されるスキーマ・オブジェクト・タイプをリストします。
*_OBJECTS	現行エディションで表示可能なデータベースのすべてのオブジェクトを示します。このビューには、オブジェクトごとに、エディション対応であるかどうかが表示されます。
*_OBJECTS_AE	すべてのエディションにおけるデータベース内のすべてのオブジェクトを示します。このビューには、オブジェクトごとに、エディション対応であるかどうかが表示されます。
*_ERRORS	現行エディションのデータベース内のすべてのエラーを示します。
*_ERRORS_AE	すべてのエディションのデータベース内のすべてのエラーを示します。
*_USERS	データベースのすべてのユーザーを示します。エディションが有効になっているユーザーを表示

ビュー	説明:
	するときに役立ちます。
*_SERVICES	データベースのすべてのサービスを示します。EDITIONS 列には、デフォルトの初期現行エディションが示されます。
*_MVIEWS	すべてのマテリアライズド・ビューを示します。マテリアライズド・ビューがエディション・オブジェクトを参照している場合、このビューには、評価エディション、およびマテリアライズド・ビューが問合せリライトに対して適格であるエディションの範囲が示されます。
*_TAB_COLS	すべての表、ビューおよびクスタのすべての列を示します。このビューには、仮想列ごとに評価エディションと使用可能な範囲が示されます。

ノート:

*_OBJECTS および*_OBJECTS_AE に含まれる依存オブジェクトが、[表 26-2](#) の操作によって無効化されるのは、次のいずれかの後のみです。



- オブジェクトの参照(コンパイル中または実行中)
- DBMS_UTILITY.COMPILE_SCHEMA の呼出し
- UTL_RECOMP サブプログラムの起動

関連項目:

- 特定のビューの詳細は、『[Oracle Databaseリファレンス](#)』を参照してください。
- DBMS_UTILITY.COMPILE_SCHEMA プロシージャの詳細は、『[Oracle Database PL/SQLパッケージおよびタイプ・リファレンス](#)』を参照してください。
- UTL_RECOMP サブプログラムの詳細は、『[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。

27.5.2 エディションニング・ビューに関する情報の表示

[表27-2](#)は、エディションニング・ビューに関する情報を表示する静的データ・ディクショナリ・ビューについて簡単に説明しています。

表27-2 エディションニング・ビュー情報を含む*_ディクショナリ・ビュー

ビュー	説明:
*_VIEWS	エディションニング・ビューを含む、現行エディションで表示可能なデータベースのすべてのビューを示します。

ビュー	説明:
*_<u>EDITIONING_VIEWS</u>	現行エディションで表示可能なデータベースのすべてのエディショニング・ビューを示します。エディショニング・ビューと実表の関係を表示するときに役立ちます。 *_<u>OBJECTS_AE</u> と結合すると、さらに情報を得ることができます。
*_<u>EDITIONING_VIEWS_AE</u>	すべてのエディションにおけるデータベース内のすべてのエディショニング・ビューのすべての実オブジェクトを示します。
*_<u>EDITIONING_VIEW_COLS</u>	現行エディションで表示可能なデータベースのすべてのエディショニング・ビューの列を示します。エディショニング・ビューの列とマッピング先の表の列との関係を表示するときに役立ちます。 *_<u>OBJECTS_AE</u> または*_TAB_COL(あるいは両方)と結合すると、さらに情報を得ることができます。
*_<u>EDITIONING_VIEW_COLS_AE</u>	すべてのエディションのデータベースにあるすべてのエディショニング・ビューの列を示します。

*_EDITIONING_VIEWSの各行は、*_VIEWSの1つの行と厳密に一致します。EDITIONING_VIEW = 'Y'の*_VIEWSの各行は、*_EDITIONING_VIEWSの1つの行と厳密に一致します。したがって、次の例のWHERE句は冗長です。

```
SELECT ...
FROM DBA_EDITIONING_VIEWS INNER JOIN DBA_VIEWS
USING (OWNER, VIEW_NAME)
WHERE EDITIONING_VIEW = 'Y'
AND ...
```

*_EDITIONING_VIEWSの行と一致する*_VIEWSの行は、定義によりEDITIONING_VIEW = 'Y'を満たします。反対に、EDITIONING_VIEW = 'N'の*_VIEWSの行に対応する行は*_EDITIONING_VIEWSにはありません。

関連項目:

特定のビューの詳細は、[『Oracle Databaseリファレンス』](#)を参照してください。

27.5.3 crosseditionトリガーに関する情報の表示

トリガーの情報を表示する静的データ・ディクショナリ・ビューは、[『Oracle Databaseリファレンス』](#)を参照してください。静的データ・ディクショナリ・ビューに示されるcrosseditionトリガーは、現行エディションの実オブジェクトです。

実行される一連のcrosseditionトリガーが異なる場合、子カーソルを共有することはできません。動的パフォーマンス・ビューV\$SQL_SHARED_CURSORおよびGV\$SQL_SHARED_CURSORには、これに該当するかどうかを示すCROSSEDITION_TRIGGER_MISMATCH列があります。

関連項目:

[V\\$SQL_SHARED_CURSORの詳細は、](#)[『Oracle Databaseリファレンス』](#)を参照してください。

27.6 アプリケーションをアップグレードするためのEBRの使用

EBRを使用して、アプリケーションをオンラインでアップグレードするには、まず、アプリケーションを準備する必要があります。

1. アプリケーション・ユーザーおよび適切なスキーマ・オブジェクト・タイプをスキーマ内でエディション対応にします。

エディショニング・ビューを作成(次のステップ)するスキーマでは、タイプVIEWはエディション対応である必要があります。

手順は、[「ユーザーに対するエディションの有効化」](#)を参照してください。

2. エディショニング・ビューを使用するようにアプリケーションを準備します。

手順は、[「アプリケーションでのエディショニング・ビューの使用準備」](#)を参照してください。

エディショニング・ビューを表示し、EBRを使用して、アプリケーションをオンラインで必要な回数、アップグレードすることができます。アップグレードのたびに、次のようにしてください。

- 再定義するオブジェクトのタイプがすべてエディション対応である場合(表はエディション対応ではありません)は、[「エディションのみを使用するEBRの手順」](#)の手順を使用します。
- 1つ以上の表の構造を変更するとき、その間に他のユーザーがそれらの表のデータを変更できなくてもよい場合は、[「エディショニング・ビューを使用するEBRの手順」](#)の手順を使用します。
- 1つ以上の表の構造を変更するとき、その間に他のユーザーがそれらの表のデータを変更できる必要がある場合は、[「crosseditionトリガーを使用するEBRの手順」](#)の手順を使用します。

トピック:

- [アプリケーションでのエディショニング・ビューの使用準備](#)
- [エディションのみを使用するEBRの手順](#)
- [エディショニング・ビューを使用するEBRの手順](#)
- [crosseditionトリガーを使用するEBRの手順](#)
- [アプリケーション・アップグレードのロールバック](#)
- [使用されていない表の列が占める領域の再利用](#)
- [例: アプリケーションをアップグレードするためのEBRの使用](#)

27.6.1 アプリケーションでのエディショニング・ビューの使用準備

1つ以上の表を使用するアプリケーションでは、エディショニング・ビューで各表がカバーされる必要があります。エディショニング・ビューは、次のすべての記述が正しい場合にのみ、表をカバーします。

- アプリケーション内の通常オブジェクトはすべて、エディショニング・ビューを通じてのみ、表を参照できます。(通常オブジェクトは、エディショニング・ビューまたはcrosseditionトリガー以外のすべてのオブジェクトです。エディショニング・ビューとcrosseditionトリガーは表を参照する必要があります。)
- アプリケーション・ユーザーは、表ではなく、エディショニング・ビューに対してのみオブジェクト権限を付与されます。
- 仮想プライベート・データベース(VPD)ポリシーは、表ではなく、エディショニング・ビューにのみアタッチされます。(日常的な監査、およびファイングレイン監査(FGA)ポリシーは、表にのみアタッチされます。)

エディショニング・ビューが実オブジェクト化された場合、VPDポリシーのコピーは実オブジェクト化されたエディショニング・ビューにアタッチされます。(ポリシーは名前と、アタッチ先オブジェクトにより一意に識別されます。)ポリシー・ファンクションも実オブジェクト化

されている場合、このポリシーのコピーは実オブジェクト化されたポリシー・ファンクションを使用します。それ以外の場合は元のポリシー・ファンクションを使用します。

VPDポリシーを示す静的なデータ・ディクショナリ・ビュー*_POLICIESは、エディションに応じて、異なる結果を表示することができます。

関連項目:

- VPDポリシーに関する情報を示す静的データ・ディクショナリ・ビューを含め、VPDに関する詳細は、[『Oracle Databaseセキュリティ・ガイド』](#)を参照してください。
- *_POLICIESの詳細は、[『Oracle Databaseリファレンス』](#)を参照してください。

既存のアプリケーションがエディショニング・ビューを使用していない場合は、使用する各表に対して次の手順を実行して、エディショニング・ビューを使用できるようにアプリケーションを準備します。

1. 表に新しい名前を付けます(現在の表名をエディショニング・ビューに付けるためです)。

新しい名前は、元の名前に関連したものとし、変更履歴を示すようにしてください。たとえば、元の表名がDataの場合、新しい表名をData_1とすることができます。

2. (オプション)表の各列に新しい名前を付けます。

このときも、新しい名前は、元の名前に関連したものとし、変更履歴を示すようにしてください。たとえば、NameおよびNumberをName_1およびNumber_1のように変更できます。

名前を変更した列に依存するトリガーはここで無効になります。詳細は、[表26-2](#)でALTER TABLE table RENAME columnの項目を参照してください。

3. エディショニング・ビューを作成し、表の元の名前を付けます。

手順は、[「エディショニング・ビューの作成」](#)を参照してください。

エディショニング・ビューには表の元の名前が付けられるため、その表名を参照しているオブジェクトがエディショニング・ビューを参照するようになります。

4. トリガーが表に定義されている場合は、トリガーを削除してから、作成時に使用したコードを再実行します。

こうすることで、表に定義されていたトリガーが、エディショニング・ビューに対して定義されます。

5. VPDポリシーが表にアタッチされている場合は、このポリシーとポリシー・ファンクションを削除してから、作成時に使用したコードを再実行します。

これにより、表にアタッチされていたVPDポリシーが、エディショニング・ビューにアタッチされます。

6. すべてのアプリケーション・ユーザーから、表に対するすべてのオブジェクト権限を取り消します。

表に対し、どのアプリケーション・ユーザーがどのオブジェクト権限を持っているかを確認するには、次の問合せを使用します。

```
SELECT GRANTEE, PRIVILEGE
FROM DBA_TAB_PRIVS
WHERE TABLE_NAME='table_name';
```

7. ステップ6で取り消された権限すべてについて、エディショニング・ビューで同じ権限を付与します。

8. 表を参照するプライベート・シノニムを所有するユーザーごとに、タイプSYNONYMがスキーマ内でエディション対応である

- よう指定し、エディションを有効化します(詳細は、[「ユーザーに対するエディションの有効化」](#)を参照)。
9. 表を参照するプライベート・シノニムの所有者に、これらのシノニムの再作成が必要であることを通知します。

27.6.2 エディションのみを使用するEBRの手順

次の手順は、再定義するすべてのオブジェクトがエディション化(定義は[「エディション・オブジェクトと非エディション・オブジェクト」](#)を参照)されている場合のみ使用します。表はエディション・オブジェクトではありません。

1. 新しいエディションを作成します。

手順は、[「エディションの作成」](#)を参照してください。

2. 新しいエディションをセッション・エディションにします。

手順は、[「セッション・エディションの変更」](#)を参照してください。

3. アプリケーションのエディション・オブジェクトに必要な変更を行います。

4. すべてのオブジェクトが有効であることを確認します。

静的データ・ディクショナリ*_OBJECTS_AEを問い合わせます。これは、すべてのエディションにおけるデータベース内のすべての実オブジェクトを示します。無効オブジェクトが残っている場合は、任意のUTL_RECOMPサブプログラム([『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)を使用して再コンパイルします。

5. 変更内容が意図したように機能していることを確認します。

機能している場合は、ステップ6に進みます。

そうでない場合は、さらに変更を続ける(ステップ3に戻る)か、またはアプリケーション・アップグレードをロールバックします(ステップについては、[「アプリケーション・アップグレードのロールバック」](#)を参照)。

6. 新しいエディション(アップグレードされたアプリケーション)をすべてのユーザーに対して使用可能にします。

手順は、[「すべてのユーザーに対してエディションを使用可能にする」](#)を参照してください。

7. 古いエディション(元のアプリケーション)をリタイアさせ、SYS以外のすべてのユーザーがアップグレードされたアプリケーションのみを使用するようにします。

手順は、[「エディションのリタイア」](#)を参照してください。

[例27-6](#)は、ここで説明した手順を使用して、非常に単純なPL/SQLプロシージャを変更する方法を示します。

例27-6 非常に単純なプロシージャのEBR

1. この例で使用するPL/SQLプロシージャを作成します。

```
CREATE OR REPLACE PROCEDURE hello IS
BEGIN
    DBMS_OUTPUT.PUT_LINE('Hello, edition 1. ');
END hello;
/
```

2. PL/SQLプロシージャを起動します。

```
BEGIN hello(); END;
/
```

結果:

```
Hello, edition 1.
```

```
PL/SQL procedure successfully completed.
```

3. プロシージャのEBRを実行します。

- a. 新しいエディションを作成します。

```
CREATE EDITION e2 AS CHILD OF ora$base;
```

結果:

```
Edition created.
```

- b. 新しいエディションをセッション・エディションにします。

```
ALTER SESSION SET EDITION = e2;
```

結果:

```
Session altered.
```

- c. プロシージャを変更します。

```
CREATE OR REPLACE PROCEDURE hello IS
BEGIN
    DBMS_OUTPUT.PUT_LINE('Hello, edition 2. ');
END hello;
/
```

結果:

```
Procedure created.
```

- d. 変更内容が意図したように機能していることを確認します。

```
BEGIN hello(); END;
/
```

結果:

```
Hello, edition 2.
PL/SQL procedure successfully completed.
```

- e. すべてのユーザーが新しいエディションを使用できるようにします(システム権限が必要です)。

```
ALTER DATABASE DEFAULT EDITION = e2;
```

- f. 古いエディションをリタイアさせます(システム権限が必要です)。

権限受領者のリストを表示します。

```
SELECT GRANTEE, PRIVILEGE
FROM DBA_TAB_PRIVS
WHERE TABLE_NAME = UPPER('ora$base')
/
```

結果:

GRANTEE	PRIVILEGE
-----	-----
PUBLIC	USE

1 row selected.

すべての権限受領者について、古いエディションの使用を取り消します。

```
REVOKE USE ON EDITION ora$base FROM PUBLIC;
```

27.6.3 エディショニング・ビューを使用するEBRの手順

1つ以上の表の構造を変更するとき、その間に他のユーザーがそれらの表のデータを変更できる必要がない場合のみ、次の手順を使用します。

1. 新しいエディションを作成します。

手順は、[「エディションの作成」](#)を参照してください。

2. 新しいエディションをセッション・エディションにします。

手順は、[「セッション・エディションの変更」](#)を参照してください。

3. 新しいエディションで、エディショニング・ビューが読取り専用の場合は、読取り/書込みに変更します。

手順は、[「エディショニング・ビューの書込み可能性の変更」](#)を参照してください。

4. 新しいエディションを除くすべてのエディションで、エディショニング・ビューを読取り専用にします。

5. アプリケーションのオブジェクトに必要な変更を行います。

6. すべてのオブジェクトが有効であることを確認します。

静的データ・ディクショナリ*_OBJECTS_AEを問い合わせます。これは、すべてのエディションにおけるデータベース内のすべての実オブジェクトを示します。無効オブジェクトが残っている場合は、任意のUTL_RECOMPサブプログラム([『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)を使用して再コンパイルします。

7. 変更内容が意図したように機能していることを確認します。

機能している場合は、ステップ8に進みます。

そうでない場合は、さらに変更を続ける(ステップ5に戻る)か、またはアプリケーション・アップグレードをロールバックします(ステップについては、[「アプリケーション・アップグレードのロールバック」](#)を参照)。

8. アップグレードされたアプリケーションをすべてのユーザーに対して使用可能にします。

手順は、[「すべてのユーザーに対してエディションを使用可能にする」](#)を参照してください。

9. 古いエディション(元のアプリケーション)をリタイアさせ、SYS以外のすべてのユーザーがアップグレードされたアプリケーションのみを使用するようにします。

手順は、[「エディションのリタイア」](#)を参照してください。

27.6.4 crosseditionトリガーを使用するEBRの手順

1つ以上の表の構造を変更するとき、その間に他のユーザーがそれらの表のデータを変更する必要がある場合のみ、次の手順を使用します。

1. 新しいエディションを作成します。

手順は、[「エディションの作成」](#)を参照してください。

2. 新しいエディションをセッション・エディションにします。

手順は、[「セッション・エディションの変更」](#)を参照してください。

3. アプリケーションのオブジェクトに永続的な変更を行います。

たとえば、表に新しい列を追加し、新しい永続サブプログラムを作成します。

ここで、変更したオブジェクトに依存するオブジェクトが無効になることがあります。詳細については、[表26-2](#)を参照してください。

4. すべてのオブジェクトが有効であることを確認します。

静的データ・ディクショナリ*_OBJECTS_AEを問い合わせます。これは、すべてのエディションにおけるデータベース内のすべての実オブジェクトを示します。無効オブジェクトが残っている場合は、任意のUTL_RECOMPサブプログラム([『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)を使用して再コンパイルします。

5. 一時オブジェクト、すなわちcrosseditionトリガー(無効状態)とトリガーで必要なサブプログラムを作成します。

手順は、[「crosseditionトリガーの作成」](#)を参照してください。

reverse crosseditionトリガーが必要になるのは、ステップ[10](#)(オプション)を実行する場合のみです。

6. crosseditionトリガーが正常にコンパイルされたら、crosseditionトリガーを有効にします。

ALTER TRIGGER文にENABLEオプションを付けて使用します。この文の詳細は、[『Oracle Database PL/SQL言語リファレンス』](#)を参照してください。

7. 保留中の変更がコミットまたはロールバックされるまで待機します。

プロシージャDBMS_UTILITY.WAIT_ON_PENDING_DMLを使用します。このプロシージャの詳細は、[『Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス』](#)を参照してください。

8. トランスフォームを適用します。

手順については、[「アップグレード前の表現をアップグレード後の表現に変換」](#)を参照してください。



ノート:

このステップが既存の行を処理した後で、祖先エディションのユーザーが同じ行のデータを更新、挿入または削除するかどうかを予測することはできません。

9. 変更内容が意図したように機能していることを確認します。

機能している場合は、ステップ[10](#)に進みます。

そうでない場合は、さらに変更を続ける(ステップ [3](#)に戻る)か、またはアプリケーション・アップグレードをロールバックします(ステップについては、[「アプリケーション・アップグレードのロールバック」](#)を参照)。

10. (オプション)セッション・エディションのUSE権限を、アップグレードしたアプリケーションの初期のユーザーに付与します。

手順は、[「一部のユーザーに対してエディションを使用可能にする」](#)を参照してください。

11. アップグレードされたアプリケーションをすべてのユーザーに対して使用可能にします。

手順は、[「すべてのユーザーに対してエディションを使用可能にする」](#)を参照してください。

12. 制約を無効化、または削除してから、crosseditionトリガーを削除します。

手順は、[「crosseditionトリガーの削除」](#)を参照してください。

13. 古いエディション(元のアプリケーション)をリタイアさせ、SYS以外のすべてのユーザーがアップグレードされたアプリケーションのみを使用するようにします。

手順は、[「エディションのリタイア」](#)を参照してください。

27.6.5 アプリケーション・アップグレードのロールバック

アプリケーション・アップグレードをロールバックするには、次のようにします。

1. セッション・エディションを、アップグレード用に作成した新しいエディション以外のものに変更します。

手順は、[「セッション・エディションの変更」](#)を参照してください。

2. アップグレード用に作成した新しいエディションを削除します。

手順は、[「エディションの削除」](#)を参照してください。

3. アップグレード中に、新しい表の列を作成した場合、これらの列が占めていた領域を再利用します(手順については、[「使用されていない表の列が占める領域の再利用」](#)を参照)。

27.6.6 使用されていない表の列が占める領域の再利用

新しい表の列を作成したアップグレードをロールバックする場合

未使用の列が占めている領域を再利用するには、次のようにします。

1. 未使用の列の値をNULLに設定します。

この操作を行っている間に、他のユーザーをロックアウトしてしまわないようにするには、DBMS_PARALLEL_EXECUTE プロシージャを使用します([『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』](#)を参照)。

2. 未使用の列をUNUSEDに設定します。

ALTER TABLE文([『Oracle Database SQL言語リファレンス』](#)を参照)をSET UNUSED句([『Oracle Database SQL言語リファレンス』](#)を参照)とともに使用します。

3. 表を縮小します。

ALTER TABLE文([『Oracle Database SQL言語リファレンス』](#)を参照)をSHRINK SPACE句([『Oracle Database SQL言語リファレンス』](#)を参照)とともに使用します。

27.6.7 例: アプリケーションをアップグレードするためのEBRの使用

この例は、エディション、エディショニング・ビュー、forward crosseditionトリガーおよびreverse crosseditionトリガーを使用します。

トピック:

- [既存のアプリケーション](#)
- [アプリケーションでのエディショニング・ビューの使用準備](#)
- [アプリケーションをアップグレードするためのEBRの使用](#)

ノート:



アプリケーションのアップグレードに EBR を使用する前に、このアプリケーションで使用されるスキーマすべてについてエディションを有効化する必要があります。手順は、[「ユーザーに対するエディションの有効化」](#)を参照してください。

27.6.7.1 既存のアプリケーション

既存のアプリケーション(アップグレード対象のアプリケーション)は、トリガーが定義された1つの表で構成されています。

既存のアプリケーションにはトリガーがあり、チェックすることができます。次の例では、既存のアプリケーションを示します。

- [例：既存のアプリケーションの作成](#)
- [例：既存の表のデータの表示](#)

27.6.7.1.1 例：既存アプリケーションが作成された方法

このアプリケーションは[例27-7](#)に従って作成されたものです。

例27-7 既存のアプリケーションの作成

1. 表を作成します。

```
CREATE TABLE Contacts(  
  ID          NUMBER(6,0) CONSTRAINT Contacts_PK PRIMARY KEY,  
  Name        VARCHAR2(47),  
  Phone_Number VARCHAR2(20)  
);
```

2. 表にデータを移入します(例は示されていません)。
3. 表にトリガーを作成する準備をします。

```
ALTER TABLE Contacts ENABLE VALIDATE CONSTRAINT Contacts_PK;  
  
DECLARE Max_ID INTEGER;  
BEGIN  
  SELECT MAX(ID) INTO Max_ID FROM Contacts;  
  EXECUTE IMMEDIATE '  
    CREATE SEQUENCE Contacts_Seq  
      START WITH ' || To_Char(Max_ID + 1);  
END;  
/
```

4. トリガーを作成します。

```
CREATE TRIGGER Contacts_BI  
  BEFORE INSERT ON Contacts FOR EACH ROW  
BEGIN  
  :NEW.ID := Contacts_Seq.NEXTVAL;  
END;  
/
```

27.6.7.1.2 例：既存表のデータ表示

[例27-8](#)は、データ移入後の表Contactsを示しています。

例27-8 既存表のデータ表示

問合せ:

```
SELECT * FROM Contacts
```

```
ORDER BY Name;
```

結果:

ID	NAME	PHONE_NUMBER
174	Abel, Ellen	011.44.1644.429267
166	Ande, Sundar	011.44.1346.629268
130	Atkinson, Mozhe	650.124.6234
105	Austin, David	590.423.4569
204	Baer, Hermann	515.123.8888
116	Baida, Shelli	515.127.4563
167	Banda, Amit	011.44.1346.729268
172	Bates, Elizabeth	011.44.1343.529268
192	Bell, Sarah	650.501.1876
151	Bernstein, David	011.44.1344.345268
129	Bissot, Laura	650.124.5234
169	Bloom, Harrison	011.44.1343.829268
185	Bull, Alexis	650.509.2876
187	Cabrio, Anthony	650.509.4876
148	Cambrault, Gerald	011.44.1344.619268
154	Cambrault, Nanette	011.44.1344.987668
110	Chen, John	515.124.4269
...		
120	Weiss, Matthew	650.123.1234
200	Whalen, Jennifer	515.123.4444
149	Zlotkey, Eleni	011.44.1344.429018

107 rows selected.

Contactsを再定義する必要があるとします。Name列をFirst_Name列とLast_Name列で置換し、Country_Code列を追加します。また、この構造変更を行う際に、他のユーザーがContactsのデータを変更できることが必要です。

EBRのすべての機能が必要になります。つまり、エディション(常に必要)、エディショニング・ビュー(表を再定義するため)、および crosseditionトリガー(表の再定義中に他のユーザーが表のデータを変更する必要があるため)です。

27.6.7.2 アプリケーションでのエディショニング・ビューの使用準備

[例27-9](#)は、新しいエディションでエディショニング・ビューを作成する方法を示します。Contacts表を再定義している間、エディショニング・ビューを使用して他のユーザーがその表にアクセスします。

例27-9 既存の表のエディショニング・ビューの作成

1. 表に新しい名前を付けます(現在の表名をエディショニング・ビューに付けるためです)。

```
ALTER TABLE Contacts RENAME TO Contacts_Table;
```

2. (オプション)表の列に新しい名前を付けます。

```
ALTER TABLE Contacts_Table
  RENAME COLUMN Name TO Name_1;
ALTER TABLE Contacts_Table
  RENAME COLUMN Phone_Number TO Phone_Number_1;
```

3. エディショニング・ビューを作成します。

```
CREATE OR REPLACE EDITIONING VIEW Contacts AS
  SELECT
    ID
      ID,
    Name_1
      Name,
    Phone_Number_1
      Phone_Number
  FROM Contacts_Table;
```

4. トリガーContacts_BIを表からエディショニング・ビューに移動します。

```
DROP TRIGGER Contacts_BI;

CREATE TRIGGER Contacts_BI
  BEFORE INSERT ON Contacts FOR EACH ROW
BEGIN
  :NEW.ID := Contacts_Seq.NEXTVAL;
END;
/
```

27.6.7.3 サンプル・アプリケーションをアップグレードするためのEBRの使用

トリガーを使用して、既存のアプリケーションをアップグレードできます。

次の例は、トリガーを使用してサンプル・アプリケーションを更新する方法を示します。

- [例: サンプル・アプリケーションをアップグレードするエディションの作成](#)
- [例: 表の変更とエディショニング・ビューの置換](#)
- [例: crosseditionトリガーの作成と有効化](#)
- [例: トランスフォームの適用](#)
- [例: 変更された表のデータ表示](#)

27.6.7.3.1 例: サンプル・アプリケーションをアップグレードするエディションの作成

[例27-10](#)では、既存のアプリケーション([「既存のアプリケーション」](#)内)をアップグレードするエディションを作成し、新しいエディションをセッション・エディションにして、新しいエディションが実際にセッション・エディションであることをチェックする方法を示します。

例27-10 サンプル・アプリケーションをアップグレードするエディションの作成

1. 新しいエディションを作成します。

```
CREATE EDITION Post_Upgrade AS CHILD OF Ora$Base;
```

2. 新しいエディションをセッション・エディションにします。

```
ALTER SESSION SET EDITION = Post_Upgrade;
```

3. セッション・エディションをチェックします。

```
SELECT
SYS_CONTEXT('Userenv', 'Current_Edition_Name') "Current_Edition"
FROM DUAL;
```

結果:

```
Current_Edition
```

```
-----
POST_UPGRADE
```

```
1 row selected.
```

[「例: サンプル・アプリケーションをアップグレードするエディションの作成」](#)は、Post_Upgradeエディションで、物理表に新しい列を追加し、列の追加により無効化されたトリガーを再コンパイルする方法を示します。エディショニング・ビューContactsが表の列を適切な論理名で選択するように、このエディショニング・ビューを置換する方法を示します。

ノート:



実表を変更するため、「[非ブロック化およびブロック化の DDL 文](#)」を参照してください。

27.6.7.3.2 例: 表の変更とエディショニング・ビューの置換

[例27-11](#)は、Post_Upgradeエディションで、forward crosseditionトリガーにより使用されるプロシージャを2つ作成し、無効な状態のforward crosseditionトリガーとreverse crosseditionトリガーの両方を作成してから、これらを有効化する方法を示します。

例27-11 表の変更とエディショニング・ビューの置換

1. 物理表に新しい列を追加します。

```
ALTER TABLE Contacts_Table ADD (  
    First_Name_2      varchar2(20),  
    Last_Name_2       varchar2(25),  
    Country_Code_2    varchar2(20),  
    Phone_Number_2    varchar2(20)  
);
```

(これは非ブロック化DDLです。)

2. 無効化されたトリガーを再コンパイルします。

```
ALTER TRIGGER Contacts_BI COMPILE REUSE SETTINGS;
```

3. エディショニング・ビューを置換して、このエディショニング・ビューが置換列を適切な論理名で選択するようにします。

```
CREATE OR REPLACE EDITIONING VIEW Contacts AS  
SELECT  
    ID                ID,  
    First_Name_2      First_Name,  
    Last_Name_2       Last_Name,  
    Country_Code_2    Country_Code,  
    Phone_Number_2    Phone_Number  
FROM Contacts_Table;
```

27.6.7.3.3 例: crosseditionトリガーの作成と有効化

[例27-12](#)は、Post_Upgradeエディションで、トランスフォームを適用する方法を示します。

例27-12 crosseditionトリガーの作成と有効化

1. forward crosseditionトリガーにより使用される1つ目のプロシージャを作成します。

```
CREATE OR REPLACE PROCEDURE Set_First_And_Last_Name (  
    Name          IN  VARCHAR2,  
    First_Name    OUT VARCHAR2,  
    Last_Name     OUT VARCHAR2)  
IS  
    Comma_Pos NUMBER := INSTR(Name, ',');  
BEGIN  
    IF Comma_Pos IS NULL OR Comma_Pos < 2 THEN  
        RAISE Program_Error;  
    END IF;  
  
    Last_Name := SUBSTR(Name, 1, Comma_Pos-1);  
    Last_Name := RTRIM(Ltrim(Last_Name));  
  
    First_Name := SUBSTR(Name, Comma_Pos+1);  
    First_Name := RTRIM(LTRIM(First_Name));
```



```
END Set_First_And_Last_Name;
/
```

2. forward crosseditionトリガーにより使用される2つ目のプロシーダを作成します。

```
CREATE OR REPLACE PROCEDURE Set_Country_Code_And_Phone_No (
    Phone_Number      IN  VARCHAR2,
    Country_Code      OUT VARCHAR2,
    Phone_Number_V2   OUT VARCHAR2)
IS
    Char_To_Number_Error EXCEPTION;
    PRAGMA EXCEPTION_INIT(Char_To_Number_Error, -06502);
    Bad_Phone_Number EXCEPTION;
    Nmbr VARCHAR2(30) := REPLACE(Phone_Number, '.', '-');

    FUNCTION Is_US_Number(Nmbr IN VARCHAR2)
        RETURN BOOLEAN
    IS
        Len NUMBER := LENGTH(Nmbr);
        Dash_Pos NUMBER := INSTR(Nmbr, '-');
        n PLS_INTEGER;
    BEGIN
        IF Len IS NULL OR Len <> 12 THEN
            RETURN FALSE;
        END IF;
        IF Dash_Pos IS NULL OR Dash_Pos <> 4 THEN
            RETURN FALSE;
        END IF;
        BEGIN
            n := TO_NUMBER(SUBSTR(Nmbr, 1, 3));
        EXCEPTION WHEN Char_To_Number_Error THEN
            RETURN FALSE;
        END;

        Dash_Pos := INSTR(Nmbr, '-', 5);

        IF Dash_Pos IS NULL OR Dash_Pos <> 8 THEN
            RETURN FALSE;
        END IF;

        BEGIN
            n := TO_NUMBER(SUBSTR(Nmbr, 5, 3));
        EXCEPTION WHEN Char_To_Number_Error THEN
            RETURN FALSE;
        END;

        BEGIN
            n := TO_NUMBER(SUBSTR(Nmbr, 9));
        EXCEPTION WHEN Char_To_Number_Error THEN
            RETURN FALSE;
        END;

        RETURN TRUE;
    END Is_US_Number;

BEGIN
    IF Nmbr LIKE '011-%' THEN
        DECLARE
            Dash_Pos NUMBER := INSTR(Nmbr, '-', 5);
        BEGIN
            Country_Code := '+' || TO_NUMBER(SUBSTR(Nmbr, 5, Dash_Pos-5));
            Phone_Number_V2 := SUBSTR(Nmbr, Dash_Pos+1);
        EXCEPTION WHEN Char_To_Number_Error THEN
            raise Bad_Phone_Number;
        END;
    ELSIF Is_US_Number(Nmbr) THEN
```

```

        Country_Code := '+1';
        Phone_Number_V2 := Nmbr;
    ELSE
        RAISE Bad_Phone_Number;
    END IF;
EXCEPTION WHEN Bad_Phone_Number THEN
    Country_Code := '+0';
    Phone_Number_V2 := '000-000-0000';
END Set_Country_Code_And_Phone_No;
/

```

3. 無効な状態のforward crosseditionトリガーを作成します。

```

CREATE OR REPLACE TRIGGER Contacts_Fwd_Xed
BEFORE INSERT OR UPDATE ON Contacts_Table
FOR EACH ROW
FORWARD CROSSSESSION
DISABLE
BEGIN
    Set_First_And_Last_Name(
        :NEW.Name_1,
        :NEW.First_Name_2,
        :NEW.Last_Name_2
    );
    Set_Country_Code_And_Phone_No(
        :NEW.Phone_Number_1,
        :NEW.Country_Code_2,
        :NEW.Phone_Number_2
    );
END Contacts_Fwd_Xed;
/

```

4. forward crosseditionトリガーを有効化します。

```

ALTER TRIGGER Contacts_Fwd_Xed ENABLE;

```

5. 無効な状態のreverse crosseditionトリガーを作成します。

```

CREATE OR REPLACE TRIGGER Contacts_Rvrs_Xed
BEFORE INSERT OR UPDATE ON Contacts_Table
FOR EACH ROW
REVERSE CROSSSESSION
DISABLE
BEGIN
    :NEW.Name_1 := :NEW.Last_Name_2 || ', ' || :NEW.First_Name_2;
    :NEW.Phone_Number_1 :=
    CASE :NEW.Country_Code_2
        WHEN '+1' THEN
            REPLACE(:NEW.Phone_Number_2, '-', '.')
        ELSE
            '011.' || LTRIM(:NEW.Country_Code_2, '+') || '. ' ||
            REPLACE(:NEW.Phone_Number_2, '-', '.')
    END;
END Contacts_Rvrs_Xed;
/

```

6. reverse crosseditionトリガーを有効化します。

```

ALTER TRIGGER Contacts_Rvrs_Xed ENABLE;

```

7. 保留中の変更がコミットまたはロールバックされるまで待機します。

```

DECLARE
    scn          NUMBER := NULL;
    timeout CONSTANT INTEGER := NULL;

```

```

BEGIN
  IF NOT DBMS_UTILITY.WAIT_ON_PENDING_DML(Tables => 'Contacts_Table',
                                           timeout => timeout,
                                           scn      => scn)
  THEN
    RAISE_APPLICATION_ERROR(-20000,
      'Wait_On_Pending_DML() timed out. CETs were enabled before SCN: '||SCN);
  END IF;
END;
/

```

関連項目:

DBMS_UTILITY.WAIT_ON_PENDING_DMLプロシーダの詳細は、『[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。

27.6.7.3.4 例: トランスフォームの適用

[Post_Upgrade](#)エディションで、例27-13の例は、トランスフォームを適用する方法を示します。

例27-13 トランスフォームの適用

```

DECLARE
  c NUMBER := DBMS_SQL.OPEN_CURSOR();
  x NUMBER;
BEGIN
  DBMS_SQL.PARSE(
    c
    Language_Flag      => DBMS_SQL.NATIVE,
    Statement          => 'UPDATE Contacts_Table SET ID = ID',
    Apply_Crossedition_Trigger => 'Contacts_Fwd_Xed'
  );
  x := DBMS_SQL.EXECUTE(c);
  DBMS_SQL.CLOSE_CURSOR(c);
  COMMIT;
END;
/

```

27.6.7.3.5 例: 変更された表のデータ表示

[例27-14](#)は、Post_Upgradeエディションで、変更が意図したとおりに機能しているかどうかをチェックする方法を示します。

「[例: 変更された表のデータの表示](#)」を「[例: 既存の表のデータの表示](#)」と比較します。

例27-14 変更された表のデータ表示

1. 列が読みやすくなるように書式設定します。

```

COLUMN ID FORMAT 999
COLUMN Last_Name FORMAT A15
COLUMN First_Name FORMAT A15
COLUMN Country_Code FORMAT A12
COLUMN Phone_Number FORMAT A12

```

2. 問合せ:

```

SELECT * FROM Contacts
ORDER BY Last_Name;

```

結果:

ID	FIRST_NAME	LAST_NAME	COUNTRY_CODE	PHONE_NUMBER

174	Ellen	Abel	+44	1644-429267
166	Sundar	Ande	+44	1346-629268
130	Mozhe	Atkinson	+1	650-124-6234
105	David	Austin	+1	590-423-4569
204	Hermann	Baer	+1	515-123-8888
116	Shelli	Baida	+1	515-127-4563
167	Amit	Banda	+44	1346-729268
172	Elizabeth	Bates	+44	1343-529268
192	Sarah	Bell	+1	650-501-1876
151	David	Bernstein	+44	1344-345268
129	Laura	Bissot	+1	650-124-5234
169	Harrison	Bloom	+44	1343-829268
185	Alexis	Bull	+1	650-509-2876
187	Anthony	Cabrio	+1	650-509-4876
154	Nanette	Cambrault	+44	1344-987668
148	Gerald	Cambrault	+44	1344-619268
110	John	Chen	+1	515-124-4269
...				
120	Matthew	Weiss	+1	650-123-1234
200	Jennifer	Whalen	+1	515-123-4444
149	Eleni	Zlotkey	+44	1344-429018

107 rows selected.

変更が意図どおりに機能した場合は、[「crosseditionトリガーを使用するEBRの手順」](#)のステップ10から13を実行することができます。

28 トランザクション・ガードの使用

トランザクション・ガードは、計画済および計画外の停止の場合に、実行を最大1回にするためにアプリケーションで使用される汎用ツールを提供します。アプリケーションでは論理トランザクションIDを使用して、停止に続くデータベース・セッション内でオープンになっている最終トランザクションのコミット結果が判断されます。トランザクション・ガードを使用しないと、アプリケーションが停止の後にリプレイ操作を実行しようとして、トランザクションが重複してコミットされ論理破損が発生する可能性があります。トランザクション・ガードは、自動で透過的にトランザクションをリプレイするために、アプリケーション・コンティニューイティによって使用されます。

トランザクション・ガードには次のような利点があります。

- 返された結果を維持します - コミット済か未コミットかにかかわらず、その結果を信頼できるようになります
- すべてのトランザクションで既知のコミット結果を保証
- 再送信が必要なアプリケーションに対して、最大1回のトランザクションを実行するために使用できます
- 自動で透過的にトランザクションをリプレイするために、アプリケーション・コンティニューイティによって使用されます

この章は、トランザクション・ガードを使用するテクノロジーまたは製品環境に関連する主な概念および技術をよく理解していることを前提としています。

トピック:

- [トランザクション・ガードが解決する問題](#)
- [トランザクション・ガードが提供するソリューション](#)
- [トランザクション・ガードの概念および範囲](#)
- [トランザクション・ガードのデータベース構成](#)
- [トランザクション・ガードを使用するアプリケーションの開発](#)
- [トランザクション・ガードおよびアプリケーション・コンティニューイティとの関係](#)

関連項目:

- トランザクション・ガードとともにOracle Java Database Connectivity (JDBC)を使用する方法の詳細は、『Oracle Database JDBC開発者ガイド』を参照してください
- OCIとトランザクション・ガードの使用方法の詳細は、『Oracle Call Interfaceプログラマーズ・ガイド』を参照してください

28.1 トランザクション・ガードが解決する問題

トランザクション・ガードがないアプリケーションの場合、停止後にアプリケーションをリカバリする際の根本的問題が、クライアントに返送されたコミット・メッセージに継続性がないことです。クライアントとサーバー間に間隔がある場合、クライアントには通信の失敗というエラー・メッセージが示されます。このエラーでは次のことがアプリケーションに知らされません。すなわち、送信によってコミット操作が実行されたかどうか、もしくは、プロシージャ・コールが予定されたすべてのコミットおよびセッション状態の変更を完了および実行したか、コールが途中で失敗したか、またはクライアントから切断したまま現在も実行中というさらに悪い状態になっているかということです。

トランザクション・ガードがない場合、サーバーに対する通信障害の後、保証されたスケーラブルな方法で最後のコミット操作の結果を確認することが不可能または非常に困難になります。データベースに対する送信がコミットされたかどうかをアプリケーションが判断する必要がある場合は、アプリケーション内で可能性のあるすべてのコミット・ポイントについて結果を問い合わせるカスタム例外コードを追加する必要があります。システム内のあらゆる場所で障害が発生する可能性があることを考慮すると、これはほとんど非現実的です。問合せは送信ごとに固有である必要があるためです。アプリケーションが構築されて本番稼働した後では、これは完全に実現不可能です。さらに、この問合せが実行された直後にトランザクションがコミットされる可能性があるため、問合せによって正しい回答を得ることはできません。実際に、通信障害の発生後、サーバーがクライアントが切断されたことに気付かずに、送信をまだ実行中である可能性があります。データベース内のPL/SQLまたはJavaでは、プロシージャ型送信の場合、この送信が完了まで実行されたか途中で中断されたかに関するレコードもありません。このようなプロシージャがコミットされた可能性があると同時に、このプロシージャに対する後続の処理が実行されていない可能性もあります。

最後の送信がコミットされたか、近いうちにコミットされるか、または完了まで実行されていないかを識別できない場合、アプリケーションがリプレイを試みることで、すでに永続化されている変更の再発行が試みられる可能性があるため、トランザクションの送信が重複し、他の形式の論理的破損が発生する可能性があります。

トランザクション・ガードがない場合、トランザクションが開始されており、コミットが発行されている場合、クライアントに返送されたコミット・メッセージに永続性はありません。クライアントは、トランザクションがコミットされたかどうかを認識しないまま残されます。トランザクションは、非トランザクション状態が不正な場合、またはすでにコミットされている場合は、有効に再送信できません。コミットの保証または完了情報がない場合、再送信を行うと、トランザクションが複数回、不正な状態のセッションで適用される可能性があります。

28.2 トランザクション・ガードが提供するソリューション

Oracle Database 12cリリース1 (12.1.0.1)で有効なトランザクション・ガードによって、自動的かつ透過的に基準化された方法で冪等性を達成するために、アプリケーションで使用される新たな統合ツールが提供されます。主要機能は次のとおりです。

- データベース(Oracle Database 12cリリース1 (12.1.0.1)以降)でサポートされているすべてのトランザクション・タイプに対して、コミット時に論理トランザクション識別子(LTXID)を保存することにより、COMMIT結果の永続性を実現します。これには、PL/SQL内から、リモート・トランザクション(1フェーズXAトランザクション)内から、および他の方法では汎用的手段を使用して識別できないコールアウトから、自動コミットを使用して実行したトランザクションの冪等性が含まれます。
- LTXIDを使用することにより、LTXIDによって識別される進行中のトランザクションのコピーが複数存在する場合に、論理トランザクション識別子によって保護されているデータベース・トランザクションを重複できないようにすることなどを目的として、最大1回の実行セマンティクスをサポートします。
- 停止状況とは関係なく、LTXIDによって保護されている同じトランザクションの別の送信をコミットできないようにするために、進行中の処理のコミットをブロックします。
- LTXIDでコミットされた処理が最上位コール(クライアントからサーバー)の一部としてコミットされたか、サーバーのプロシージャ(PL/SQLなど)に埋め込まれているかを識別します。埋め込まれたコミット状態とは、コミットは完了しているが、コミットが実行されたプロシージャ全体がまだ完了まで実行されていないことを示します。コミット以降の処理は、そのプロシージャ自体がデータベース・エンジンに戻るまでは完了したことを保証することはできません。
- コミット解決の対象であるデータベースが元の送信より進んでいるか、同期が取れているか、または遅れているかどうかを識別し、クライアントからのトランザクションの送信順序にギャップがある場合は拒否します。サーバーまたはクライアントがLTXIDの順序と同期が取れていないときに結果を取得しようとすると、エラーとみなされます。
- LTXIDが変更されると、JDBC Thinクライアント・ドライバでコールバックが起動されます。これは、WebLogic

Serverやサード・パーティなどの上位レイヤーのアプリケーションが、使用可能な現在のLTXIDを必要に応じて保持するために使用できます。

- グローバルに分散したデータベースやMultitenantインフラストラクチャに統合されるデータベース全体にわたってネームスペースの一意性を確保します。これには、Oracle Real Application Clusters (Oracle RAC)とRAC One、Data GuardおよびMultitenantデータベースが含まれます。
- Multitenantインフラストラクチャに統合されるグローバル・データベースおよびデータベース全体にわたってサービス名の一意性を確保します。これにより、接続がトランザクション情報に正しく転送されます。

28.3 トランザクション・ガードの概念および範囲

この項では、トランザクション・ガードに関する主な概念、およびトランザクション・ガードが網羅する内容と網羅しない内容について説明します。

トピック:

- [論理トランザクション識別子\(LTXID\)](#)
- [最大1回の実行](#)
- [トランザクション・ガードのカバレッジ](#)
- [トランザクション・ガードの除外](#)

関連項目:

- トランザクション・ガードの処理の詳細は、『Oracle Database概要』を参照してください
- トランザクション・ガードとともにOracle Java Database Connectivity (JDBC)を使用する方法の詳細は、『Oracle Database JDBC開発者ガイド』を参照してください

28.3.1 論理トランザクション識別子(LTXID)

アプリケーションでは論理トランザクションID (LTXID)という概念を使用して、停止に続くデータベース・セッション内でオープンになっている最終トランザクションの結果が判断されます。論理トランザクションIDは、OCIセッション・ハンドル内、およびJDBC ThinドライバとODP.NETドライバの接続オブジェクト内に格納されます。論理トランザクションIDは、最大1回セマンティクスの基礎となります。

トランザクション・ガードプロトコルにより、次が保証されます。

- 各論理トランザクションの実行が一意になります。
- サポートされているコミット時に重複を検出することにより、すべてのコミット・ポイントにおいてプロトコルが回避されないようにする必要があります。
- トランザクションがコミットされると、再試行の保存期間中は論理トランザクションIDが永続化されます(デフォルト = 24時間、最大 = 30日)。
- 結果を取得するとき、LTXIDをブロックすることにより、そのLTXIDの以前の進行中のバージョンをコミットできなくし、未コミット・ステータスを強制します。LTXIDが同じである以前のバージョンがすでにコミットまたは強制されている場合、LTXIDをブロックすると、同じ結果が得られます。

論理セッション番号はセッションの確立時に自動的に割り当てられます。これは、アプリケーションが読み取ることができない不透

明な構造です。スケーラビリティを確保するために、各LTXIDにはコミット番号と呼ばれる実行番号があります。これは、データベースに対するラウンド・トリップごとにデータベース・トランザクションがコミットされると、増分されます。この実行コミット番号はゼロベースです。

28.3.2 最大1回の実行

トランザクション・ガードは、論理トランザクション識別子(LTXID)を使用してトランザクションの重複を回避します。トランザクションの最大1回の実行を保証するこの機能は、トランザクションの冪等性と呼ばれます。LTXIDはコミット時に継続され、ロールバックの後に再利用されます。通常の実行中、LTXIDは、各データベース・トランザクションについてクライアントおよびサーバーの両方で自動的にセッションで保持されます。コミット時には、LTXIDはトランザクションのコミットの一部として永続化されます。

最大1回プロトコルでは、リプレイについて合意した保存期間中はデータベースがLTXIDを保持することが必要になります。デフォルトの保存期間は24時間ですが、これより短いまたは長い期間が必要な場合もあり、1週間以上になることも考えられます。保存期間が長くなるほど、最大1回チェックにより、LTXIDを使用した古いトランザクションのリプレイがブロックされる期間が長くなります。この設定はサービスごとに使用できます。Data GuardやActive Data Guardを使用する場合など、複数のデータベースが関連する場合、REDOの使用を介して関連するデータベースごとにLTXIDがレプリケートされます。

Oracle JDBC Thin (OCI、OCCIおよびODP.NETクライアント用として類似するAPIがあります)用として提供されているgetLTXID APIにより、アプリケーションはデッド・セッションで使用されていた論理トランザクション識別子を取得できるようになります。これは、この最後のトランザクションのステータスを確認するために必要になります。

DBMS_APP_CONT.GET_LTXID_OUTCOME PL/SQLサブプログラムを使用すると、アプリケーションで、指定された論理トランザクション識別子のアクションの結果を検索できます。DBMS_APP_CONT.GET_LTXID_OUTCOMEのコールには、結果が判明するようにLTXIDのコミットをブロックするサーバーが関わる場合があります。これは、このLTXIDを使用するトランザクションが進行中であるかコミットしようとしている場合の必要条件になります。トランザクション・ガードを使用するアプリケーションは、リカバリ可能なエラーの後にLTXIDを取得してから、リプレイを試行する前にDBMS_APP_CONT.GET_LTXID_OUTCOMEをコールします。

関連項目:

DBMS_APP_CONT.GET_LTXID_OUTCOME PL/SQLサブプログラムの詳細は、[『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。

28.3.3 トランザクション・ガードのカバレッジ

システム内の各データベースでトランザクション・ガードを使用できます。これには、単一インスタンス・データベース、Real Application Clusters、Data GuardおよびActive Data Guard間の再起動やフェイルオーバーが含まれます。

トランザクション・ガードは、次に示すOracle Database 12cの構成でサポートされます。

- 単一インスタンスのOracle RDBMS
- Real Application Clusters
- Data Guard
- Active Data Guard
- Multitenant: PDB/CDBの切断/接続を含む(12.2の場合はPDB/CDB全体に再配置しますが、"with clone" オプションを除く)

- 前述のデータベース構成用のグローバル・データ・サービス

トランザクション・ガードはOracle Database 12cに対して、次に示すのトランザクション・タイプをサポートしています。

- ローカル・トランザクション
- データ定義言語(DDL)トランザクション
- データ制御言語(DCL)トランザクション
- 分散トランザクション
- リモート・トランザクション
- パラレル・トランザクション
- 成功時のコミット(自動コミット)
- 埋込みコミットがサポートされているクライアント・ドライバを使用したPL/SQL
- Oracle Database 12c リリース2 (12.2.0.1)以降、One Phase Optimizationsを使用するXAトランザクション(XAコミット・フラグのTMONEPHASEおよび読取り最適化を含む)
- サービス句を指定したALTER SESSION SETコンテナ(このサービスはトランザクション・ガードを使用)

トランザクション・ガードは、次に示すクライアント・ドライバをサポートしています。

- 12c JDBCタイプ4ドライバ
- 12c OCIおよびOCCIクライアント・ドライバ
- 12c Oracle Data Provider for .NET (ODP.NET)、管理対象外ドライバ
- 12c ODP.NET、ODAC 12c リリース4以降の管理対象ドライバ

28.3.4 トランザクション・ガードとXAトランザクション

Oracle Database 12.2 リリース以降、トランザクション・ガードでは1フェーズ・トランザクションの結果を判別するために、XAトランザクションをサポートしています。トランザクション・ガードは、commitの操作中にTMONEPHASEを使用するローカル・トランザクションとXAトランザクションをサポートします。アプリケーションがTMTWOPHASEを使用するXAトランザクションを発行すると、トランザクション・ガードはそのトランザクションのために自らを無効にします。その後、自動的に次のトランザクションに向けて、再度有効にします。これにより、トランザクション・ガードは次に示すXAトランザクションをサポートできます。

1. autocommitを使用するローカル・トランザクション
2. 明示的なcommitを使用するローカル・トランザクション
3. TMONEPHASEフラグを使用してコミットするXAトランザクション

TPモニターとアプリケーションでは、トランザクション・ガードを使用して、commit操作の結果を取得できます。トランザクション・ガードは外部管理のTMTWOPHASEコミット操作のために自らを無効にし、その後、次のトランザクションのために自動的に再度有効にします。トランザクション・ガードAPIがTMTWOPHASEトランザクションで使用されると、トランザクションガードが無効になっているため、警告メッセージが返されます。TPモニターは、TMTWOPHASEトランザクションのコミット結果を所有します。この機能により、TPモニターはTMONEPHASE操作の明確な結果を返すことができます。

28.3.5 トランザクション・ガードの除外

トランザクション・ガードは、再帰的トランザクションおよび自律型トランザクションを意図的に除外することにより、これらを再実行

できるようにします。

Oracle Database 12cリリース2では、トランザクション・ガードで次の機能も除外されます。

- 2フェーズXAトランザクションは外部で管理されます。XAトランザクションを使用する場合、トランザクション・ガードは1フェーズXAトランザクションのコミット結果を維持しますが、外部管理された2フェーズトランザクションの場合は結果をTPモニターが所有するため、暗黙的に自らを無効にします。
- 転送トランザクション用の読取り/書込みデータベース・リンクのあるActive Data Guard
- 論理データベース間で失敗する場合に結果を判別するGolden Gateおよびロジカル・スタンバイ。Golden Gateおよびロジカル・スタンバイ・エンドポイントはトランザクション・ガードを使用できます
- トランザクション・ガードが有効な場合、完全なデータベースのインポートは実行できません。完全なデータベースのインポートには、トランザクション・ガードを使用しない管理サービスを使用します。ユーザーとオブジェクトのインポートは除外されません。
- TAFとアプリケーション・コンティニューイティは、トランザクション・ガードを内部的に処理します。次の場所では、トランザクション・ガードをアプリケーションでコード化しないでください。
 - 失敗したTAFからの復帰
 - TAFコールバック(TAF、またはOCIおよびODP.NETのアプリケーション・コンティニューイティ向け)
 - Javaのアプリケーション・コンティニューイティ向けのJDBC初期化コールバック

トランザクション・ガードは、レプリケーション・テクノロジーによって維持されるデータベース全体のフェイルオーバーを除外します。

- Golden Gateへのレプリケーション
- ロジカル・スタンバイへのレプリケーション
- PDBクローン句(12c リリース2以降のPDBのオンラインでの再配置を除く)
- すべてのサード・パーティのレプリケーション・ソリューション

Golden Gate、Logical Standbyまたはサード・パーティ・レプリケーションなどのレプリケーション・テクノロジーを使用してデータベースのレプリカを使用している場合、この構成ではプライマリ・データベースとセカンダリ・データベース間でトランザクション・ガードを使用できない可能性があります。

トランザクション・ガードは、レプリケーションに関与する各データベースで使用できます。この場合、各データベースでは、異なるデータベース固有の識別子を使用する必要があります。V\$DATABASEを使用して、各データベースのDBIDを取得します。

28.4 トランザクション・ガードのデータベース構成

この項には、トランザクション・ガードを使用するためのデータベースの構成に関する情報が含まれます。

トピック:

- [構成チェックリスト](#)
- [トランザクション履歴表](#)
- [サービス・パラメータ](#)

28.4.1 構成チェックリスト

アプリケーションとともにトランザクション・ガードを使用するには、次を実行する必要があります。

- Oracle Database 12cリリース1 (12.1.0.1)以降を使用します。
- すべてのデータベース処理に対してアプリケーション・サービスを使用します。Oracle RACを使用している場合は `srvctl` コマンドを使用して、またはOracle RACを使用していない場合は `DBMS_SERVICE.CREATE_SERVICE` PL/SQLサブプログラムを使用してサービスを作成します。

デフォルト・データベース・サービスは使用しないでください。これらのサービスは管理用であり、操作できません。つまり、`db_name`または`db_unique_name`に設定されているサービス名は使用しないでください。

- `DBMS_APP_CONT`パッケージに対する権限を、`GET_LTXID_OUTCOME`をコールするデータベース・ユーザーに付与します。

```
GRANT EXECUTE ON DBMS_APP_CONT TO <user-name>;
```

- トランザクション・ガードをDDL文で使用する場合は、`DDL_LOCK_TIMEOUT`を増やします。

アプリケーションとともにトランザクション・ガードを使用するには、次を実行することをお勧めします。

- 最適なパフォーマンスを実現するためにトランザクション履歴表を特定および定義します。
- Oracle RACまたはOracle Data Guardを使用している場合、エラー時に通信してクライアントをすぐに中断できるように、FANが構成されていることを確認します。
- 次のパラメータを設定します: `AQ_HA_NOTIFICATIONS = TRUE` (OCI FANを使用している場合)。

関連項目:

- `DDL_LOCK_TIMEOUT`の詳細は、[『Oracle Databaseリファレンス』](#)を参照してください。
- [トランザクション履歴表](#)

28.4.2 トランザクション履歴表

トランザクション履歴表は、論理トランザクション識別子(LTXID)からデータベース・トランザクションへのマッピングを保持します。この表にアクセスできるのは、DBA権限を持つデータベース・ユーザーのみです。これはOracle Databaseによって自動的に保持されるため、トランザクション履歴表に対してDDLまたはDML文を直接発行しないでください。

デフォルトでは、トランザクション履歴表(`LTXID_TRANS`)は、データベースの作成およびアップグレード時に`SYSAUX`表領域に作成されます。最後のパーティションの記憶域を使用して、インスタンスの追加時に新規パーティションが追加されます。ただし、この表領域の場所がパフォーマンスに最適ではない場合、DBAはパーティションを別の表領域に移動できます。たとえば、次の文を使用すると、トランザクション履歴表が変更され、`FastPace`という名前の表領域に移動されます。

```
ALTER TABLE LTXID_TRANS move partition LTXID_TRANS_4
tablespace FastPace
storage ( initial 10G next 10G
minextents 1 maxextents 121 );
```

関連項目:

- `ALTER TABLE`文の詳細は、『Oracle Database SQL言語リファレンス』を参照してください

28.4.3 サービス・パラメータ

コミットの結果および保存用のサービスを構成します。

たとえば:

```
COMMIT_OUTCOME = TRUE  
RETENTION_TIMEOUT = <retention-value>
```

COMMIT_OUTCOMEは、コミットが実行された後にトランザクションのコミット結果にアクセスできるかどうかを確認します。この機能は、コミットの結果を永続化します。この機能は、停止前に実行された最後のトランザクションのステータスを強制するためにアプリケーションによって使用されます。この機能は、Oracleリプレイ・ドライバおよびWebLogic Serverによって内部的に使用され、他のアプリケーションが結果を確認するために使用することもできます。COMMIT_OUTCOMEで使用可能な値はFALSE (デフォルト)およびTRUEであり、トランザクション・ガードを有効に使用するにはこの値をTRUEに設定する必要があります。

COMMIT_OUTCOMEについて考慮する必要がある事項は次のとおりです。

- DBMS_APP_CONT.GET_LTXID_OUTCOMEプロシージャを使用するには、COMMIT_OUTCOMEをTRUEに設定する必要があります。
- COMMIT_OUTCOMEは、Active Data Guardおよび読取り専用データベースに対しては効果がありません。トランザクション・ガードと、DMLを転送するデータベース・リンクと組み合わせた読取り/書込みActive Data Guardの併用はサポートされていません。
- COMMIT_OUTCOMEは、ユーザー定義データベース・サービスで許可されます。データベース・サービスでの使用は除外されます。このサービスはData Guardでは切り替わらず、プライマリ・データベースでの計画済停止に対して開始、停止または無効化できないためです。

RETENTION_TIMEOUTは、コミット結果が保存される合計時間を設定するためにCOMMIT_OUTCOMEとともに使用されます。保存タイムアウト値は秒単位で指定します。デフォルトは86400 (24時間)、最大は2592000 (30日)です。srvctlコマンドまたはDBMS_SERVICE PL/SQLパッケージを使用して、保存タイムアウト値を指定できます。

関連項目:

- [srvctl add service](#)および[srvctl modify service](#)コマンドの詳細は、『Oracle Database管理者ガイド』を参照してください
- DBMS_SERVICE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。
- DBMS_APP_CONT.GET_LTXID_OUTCOMEプロシージャの詳細は、『[Oracle Database PL/SQLパッケージおよびタイプ・リファレンス](#)』を参照してください。

28.4.3.1 例: サーバー・プール用のサービスの追加および変更

Oracle RACまたはOracle RAC Oneを使用している場合、srvctlコマンドを使用してサービスを作成および変更します。

[例28-1](#)には、srvctlの使用方法を示します。Global Data Services (GDSCTL)も使用できます。

例28-1 サーバー・プール用のサービスの追加および変更

```
srvctl add service -database orcl -service GOLD -poolname ora.Srvpool -  
commit_outcome TRUE -retention 604800  
srvctl modify service -database orcl -service GOLD -commit_outcome TRUE -retention  
604800
```

28.4.3.2 例: 管理者管理サービスの追加

Oracle RACまたはOracle RAC Oneを使用している場合、`srvctl`コマンドを使用してサービスを作成および変更します。

[例28-2](#)には、`srvctl`の使用方法を示します。Global Data Services (GDSCTL)も使用できます

例28-2 管理者管理サービスの追加

```
srvctl add service -database codedb -service GOLD -preferred serv1 -available serv2 -
commit_outcome TRUE -retention 604800
```

28.4.3.3 例: サービスの変更(PL/SQL)

単一インスタンス・データベースを使用している場合、`DBMS_SERVICE.MODIFY_SERVICE` PL/SQLプロシージャを使用してサービスを変更し、FANを使用します。

[例28-3](#)では、サービスを変更します(ただし、`<service-name>`は実際のサービス名に変更します)。

例28-3 サービスの変更(PL/SQL)

```
DECLARE
  params dbms_service.svc_parameter_array;
BEGIN
  params('COMMIT_OUTCOME') := 'true';
  params('RETENTION_TIMEOUT') := 604800;
  params('aq_ha_notifications') := 'true';
  dbms_service.modify_service('<service-name>', params);
END;
```

28.5 トランザクション・ガードを使用するアプリケーションの開発

トランザクション・ガードを使用するには、[「構成チェックリスト」](#)の要件と推奨事項を確認します。また、リカバリ可能なエラーが発生した場合には次のエラー処理のステップに従います。



ノート:

TAFを使用する場合は、[「トランザクション・ガードと透過的アプリケーション・フェイルオーバー」](#)に進んでください。

1. エラーがリカバリ可能なエラーであり、データベース・セッションが使用不可になっていることを調べます。
2. クライアント・ドライバ提供のAPI (JDBCには`getLTXID`、OCIには`LTXID`とともに`OCI_ATTR_GET`、およびODP.NETには`LogicalTransactionId`)を使用して、前に失敗したセッションから`LTXID`を取得します。
3. 新規セッションをそのセッションの独自`LTXID`とともに取得します。
4. APIから取得した`LTXID`を使用して`DBMS_APP_CONT.GET_LTXID_OUTCOME` PL/SQLプロシージャを起動します。戻り状態により、ドライバは最後のトランザクションが`COMMITTED` (TRUE/FALSE) および`USER_CALL_COMPLETED` (TRUE/FALSE) であったことを確認します。クライアントとデータベースが同期していないと、このPL/SQLファンクションによってエラーが返されます(たとえば、同じデータベースでない場合またはデータベースがリストアされた場合)。
5. アプリケーションが結果を戻して、ユーザーが判別できます。アプリケーションは自動的にリプレイできます。リプレイ自体によって停止する場合は、リプレイ・セッションの`LTXID`が`DBMS_APP_CONT.GET_LTXID_OUTCOME` プロシージャに使用されます。

28.5.1 典型的なトランザクション・ガードの使用例

次の疑似コードは、トランザクション・ガードの標準使用例です。

```
Receive a FAN down event (or recoverable error)

FAN aborts the dead session

If recoverable error (new OCI_ATTRIBUTE for OCI, isRecoverable for JDBC)
    Get last LTXID from dead session using getLTXID or from your callback
    Obtain a new session
    Call GET_LTXID_OUTCOME with last LTXID to obtain COMMITTED and
USER_CALL_COMPLETED status

If COMMITTED and USER_CALL_COMPLETED
    Then return result

ELSEIF COMMITTED and NOT USER_CALL_COMPLETED
    Then return result with a warning (that details such as out binds or row count
were not returned)

ELSEIF NOT COMMITTED
    Cleanup and resubmit request, or return uncommitted result to the client
```

28.5.2 LTXIDの使用の詳細

リプレイおよび戻り結果を得るには、アプリケーションまたはサード・パーティ・コンテナが、セッションごとにサーバーでコミットされる次のLTXIDにアクセスできる必要があります。LTXIDは、リカバリ可能な停止後に失敗したセッションからAPI (JDBCにはgetLTXID、およびOCIにはLTXIDとともにOCI_ATTR_GET)を使用して取得できます。

JDBC Thinドライバには、データベースから取得したコミット番号変更ごとに実行されるコールバックが用意されています。サード・パーティ・コンテナは、このコールバックを使用して、フェイルオーバーが必要な際の使用に備えて現在のLTXIDを保存できます。各セッションでは現在のLTXIDが使用中であるため、コールバックは以前のLTXIDをオーバーライドできます。

フェイルオーバーが完了せずにカスケード(つまり、1つの障害からのリカバリ時に別の障害が発生)する場合、アプリケーションは、現在のセッションで有効なLTXIDを取得してGET_LTXID_OUTCOMEに渡す必要があります。

[表28-1](#)は、LTXID関連のアクションが必要な状態または状況、そのアプリケーション・アクション、および次に使用するLTXIDをそれぞれ示しています。

表28-1 LTXIDの状態または状況、アプリケーション・アクション、および次に使用するLTXID

状態および状況	アプリケーションのアクション	次に使用するLTXID (コンテナのLTXID変更に対するコールバック - JDBC Thinのみ)
アプリケーションが、リカバリ可能なエラーを受信し、GET_LTXID_OUTCOME をコールしてトランザクション・ステータスを確認します。	アプリケーションは、新規接続を確立し (独自の LTXID-B 0 を使用)、最後に失敗したセッションの LTXID (LTXID-A)を使用して GET_LTXID_OUTCOME をコールします。	新規 LTXID-B 0 登録時にも JDBC コールバックを使用して設定します
アプリケーションが、最後のセッション・トランザクション・ス	コミット済ステータスをクライアントに戻す	(該当なし)

状態および状況	アプリケーションのアクション	次に使用するLTXID (コンテナのLTXID変更に対するコールバック - JDBC Thinのみ)
テータスが COMMITTED および USER_CALL_COMPLETE Dであることを確認します。	と、アプリケーションを続行できます。	
アプリケーションが、最後の セッション・トランザクション・ス テータスが COMMITTED で あり、 USER_CALL_COMPLETE Dでないことを確認します。	コミット済ステータスをクライアントに戻し、 (該当なし) 終了します。コール内の処理が完了して いないため(たとえば、OUT バインドまた は行カウントが戻されなかった)、一部の アプリケーションは続行できません。アプリ ケーションが続行可能かどうかは、アプリ ケーションに依存します。	
アプリケーションで、最後の セッション・トランザクション・ス テータスが NOT COMMITTED であることを確 認します。	アプリケーションが結果をユーザーに返す か、必要であればクリーンアップし、有効 な新セッションの LTXID (LTXID-B 0) を使用して再発行します。 新規リクエストによってコミットが実行され ると、サーバーは LTXID-B 2 以上のコ ミット・メッセージを戻します。	新規 LTXID-B 2 ..N 登録時にも JDBC コールバックを使用して設定しま す
アプリケーションがリプレイを決 定した場合、リカバリ可能な エラーを受信します。	アプリケーションは、新規接続を確立し (独自の LTXID-C 0 を使用)、最後の セッションの LTXID (LTXID-B N)を使 用して GET_LTXID_OUTCOME をコー ルします。	新規セッションの LTXID-C 0。 登録時にも JDBC コールバックを使用して設定しま す
アプリケーションがリプレイを決 定した場合、もう 1 つのリカ バリ可能なエラーを受信しま す。	アプリケーションは、新規接続を確立し (独自の LTXID-D 0 を使用)、最後の セッションの LTXID (LTXID-C N)を使 用して GET_LTXID_OUTCOME をコー ルします。	新規セッションの LTXID-D 0。 登録時にも JDBC コールバックを使用して設定しま す

28.5.3 トランザクション・ガードと透過的アプリケーション・フェイルオーバー

トランザクション・ガードで透過的アプリケーション・フェイルオーバー(TAF)が有効になっている場合、TAFは開発者のエラーを処理します。TAFを使用しているときはトランザクション・ガードをコード化しないでください。Oracle Database 12c リリース1 (12.1.0.1)以降には、トランザクション・ガードのコードが組み込まれているためです。TAFとトランザクション・ガードの両方を使用する場合、開発者は次のTAFエラーを使用して、ロールバック、安全な再発行またはコミットされていない状態への復帰を実

行できます。

- ORA-25402
- ORA-25408
- ORA-25405

開発者はTAFが有効なときに、GET_LTXID_OUTCOMEプロシーダを直接使用しないでください。これは、TAFがすでにトランザクション・ガードの処理を行っているためです。

ノート:

TAF はセッションの失敗時には呼び出されません(これには、オペレーティング・システム・レベルの"kill -9"または ALTER SYSTEM KILL セッションが含まれます)。TAF は、次に示す状態で呼び出されます。



- INSTANCE 失敗
- FAN NODE DOWN イベント
- SHUTDOWN トランザクション
- POST_TRANSACTION 切断

28.5.4 ODP.NETでのトランザクション・ガードの使用

ODP.NETでトランザクション・ガードを使用する場合、次のルールが適用されます。

- 両方のODP.NETプロバイダでXAに昇格した後は、LTXIDを使用できません。
- Oracle Database 12c リリース2 (12.2.0.1)以降では、ODP.NETはそれ自体の可用性と処理能力に基づいて、アプリケーションのトランザクション・ガードを処理します。ODP.NETを使用する場合、ODP.NETがアプリケーションのかわりにコミット結果を取得できない場合にのみ、LTXIDがアプリケーションに公開されます。たとえば、Data Guard へのフェイルオーバー時などです。
- 開発者は、TAFコールバックまたはJDBC初期化コールバックで、トランザクション・ガードをコード化しないでください。トランザクション・ガードはユーザーのために処理されます。

28.5.5 接続プールLTXIDの使用

接続プールは、LTXIDを管理するために別のユースケースを作成します。これは、接続およびセッションが事前に確立されて共有されるためです。接続プールおよび中間層の最も簡単なモデルでは、LTXIDはセッション・ハンドル(クライアント側のセッション)ごとに存在します。これは、接続プールからチェックアウトされる際にアプリケーション・リクエストに関連付けられ、プールにチェックインして戻される際にアプリケーション・リクエストから切断されます。チェックアウトとチェックインの間、セッションのLTXIDはこのアプリケーション・リクエストによって排他的に保持されます。チェックイン後、LTXIDはアイドル状態のプール・セッションに属します。これは、接続をチェックアウトする次のアプリケーション・リクエストに関連付けられます。

トランザクション・ガードをこの方法で使用する場合:

- 現在のHTTPリクエストに関する重複の検出およびフェイルオーバーをサポートできます
- タイムアウトしたリクエストや、アプリケーションによる任意の再送信を取り消すことができます(実際のCancel操作。

[Ctrl]+[C]ではありません)。

28.5.6 XA 1フェーズ最適化のコミット結果の向上

Oracle Database 12c リリース2 (12.2.0.1)以降では、トランザクション・ガードとトランザクション処理モニター(TPM)を併用して、1フェーズ最適化(TMONEPLHASEフラグ)の使用時に、コミット操作の結果を判断します。トランザクション・ガードはGET_LTXID_OUTCOMEパッケージを使用して、リソースマネージャへの接続が失われたかどうかや、あいまいなエラーが返されたかどうかをTPMが判断できるようにします。

表28-2 トランザクション・マネージャの状態/状況およびアクション

状態および状況	トランザクション・マネージャのアクション
Commit が発行されず、トランザクションがロールバックされている場合。	トランザクション・マネージャは rollback を返します。
Commit が発行され、あいまいな結果が返された場合。	トランザクション・マネージャはエラーがリカバリ可能な場合に、トランザクション・ガード(XA)を使用して結果を判断できます。
トランザクションが COMMITTED の場合。	トランザクション・マネージャは COMMITTED を返します。
トランザクションが UNCOMMITTED の場合。	トランザクション・マネージャは新しい接続を流用して、COMMIT を再発行します。元の LTXID は、GET_LTXID_OUTCOME をコールすることでブロックされます。

28.5.7 トランザクション・ガード開発の追加要件

トランザクション・ガードは、開発者がリカバリ可能なエラー後に使用し、既知の結果を提供するためのツールです。これは、最後のセッションがデッド状態であることを示すエラーが戻されたときに使用する必要があります。

次の場合、トランザクション・ガードAPIは使用しないでください。

- 現在のセッションでGET_LTXID_OUTCOMEを使用しないでください。エラーが戻されます。
- リカバリ可能なエラーを受信しなかったセッション(つまり、ライブ・セッション)でGET_LTXID_OUTCOMEを使用しないでください。このセッションのコミットはブロックされます。
- 別のユーザーから、または別のデータベースに対して、GET_LTXID_OUTCOMEを使用しないでください。エラーが戻されます。
- 即時使用するのではなく、後で使用する場合はLTXIDを取得および保存しないでください。GET_LTXID_OUTCOMEの結果が有効なのは、最後のオープン・トランザクションまたは完了したトランザクションに対してのみです。同じセッションで以前のトランザクションとともに使用すると、エラーが戻されます。
- アプリケーションがTAFを使用している場合は、トランザクション・ガードをコード化しないでください。かわりに新しいTAFエラーコードを使用して、結果を返します。



ノート:

このルールはアプリケーション・コンティニューイティには適用できません。

関連項目:

TAFの詳細は、[「トランザクション・ガードと透過的アプリケーション・フェイルオーバー」](#)を参照してください。

28.6 トランザクション・ガードおよびアプリケーション・コンティニューイティとの関係

トランザクション・ガードには、データベース・トランザクションごとに一意の識別子(LTXID)が用意されています。この識別子は、トランザクションのコミット結果を問い合わせるために使用したり、トランザクションが1回のみ適用されるようにするために使用できます。トランザクション・ガードはアプリケーション・コンティニューイティによって使用され、これによって自動的に有効化されますが、独立して有効化することもできます。トランザクション・ガードは、アプリケーション・コンティニューイティがリプレイするトランザクションが複数回適用されるのを阻止します。アプリケーションがアプリケーション・レベルのリプレイを実装した場合、冪等性を実現するためにアプリケーションをトランザクション・ガードと統合する必要があります。

コード化を必要としないソリューションの場合は、アプリケーション・コンティニューイティを使用するようにアプリケーションを構成します。独自のリプレイを開発する場合、アプリケーション開発者はトランザクション・ガードを使用してコード化を行います。アプリケーションは、トランザクション・ガードとアプリケーション・コンティニューイティの両方に向けてコード化できます。まず、アプリケーション・コンティニューイティに効果が現れます。アプリケーション・コンティニューイティがリプレイできない場合にのみ、カスタムのトランザクション・ガードのコードに効果が現れます。両方を使用する必要はありませんが、アプリケーションがトランザクション・ガードとアプリケーション・コンティニューイティの両方を使用する場合は互換性があります。アプリケーション・コンティニューイティに加えて、トランザクション・ガードAPIもアプリケーションに追加すると、トランザクション・ガードはリプレイが無効になった場合や失敗した場合にコミット結果を返すことができます。

関連項目:

- Oracle RACでのトランザクション・ガードおよびアプリケーション・コンティニューイティの詳細は、[『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。
- JDBCを使用するデータベースへの接続の詳細は、[『Oracle Database JDBC開発者ガイド』](#)を参照してください。
- Oracle Call Interface(OCI)を使用するデータベースへの接続の詳細は、[『Oracle Call Interfaceプログラマーズ・ガイド』](#)を参照してください。
- ODP.NETドライバの詳細は、[『Oracle Data Provider for .NET開発者ガイドfor Microsoft Windows』](#)を参照してください。