Oracle® Database
2 日で開発者ガイド
19c

F16102-01(原本部品番号:E94801-01)

2019年1月

タイトルおよび著作権情報

Oracle Database 2日で開発者ガイド, 19c

F16102-01

原本著者: Chuck Murray

原本協力者: Eric Belden、Bjorn Engsig、Nancy Greenberg、Pat Huey、Christopher Jones、Sharon Kennedy、Thomas Kyte、Simon Law、Bryn Llewellen、Sheila Moore

Corporationおよびその関連会社は、第三者のコンテンツ、製品、サービスへのアクセスまたは使用によって損失、費用、あるいは損害が発生しても一切の責任を負いかねます。

目次

表一覧

はじめに

ここでは、『*Oracle Database 2*日で開発者ガイド』の概要を示します。

このマニュアルでは、Oracle Databaseでのアプリケーション開発の基本概念について説明します。Structured Query Language (SQL)、およびオラクル社がSQLデータベース言語をサーバーベースの手続き型言語として機能拡張したProcedural Language/Structured Query Language (PL/SQL)を介して、トピックの基本機能を使用する方法について説明します。

- 対象読者
  このマニュアルは、Oracle Databaseのアプリケーション開発について学習しようとするすべてのユーザーを対象とし、Oracle Databaseに初めて携わる開発者にアプリケーション開発の概要を示すことを主な目的としています。
- ドキュメントのアクセシビリティ
- 関連ドキュメント
  『*Oracle Database 2*日で開発者ガイド』で説明する概念およびタスクについて理解を深めた後、次のような他のOracle Database開発ガイドを参照することをお薦めします。
- 表記規則
  『*Oracle Database 2*日で開発者ガイド』では次の表記規則を使用します。

対象読者

このマニュアルは、Oracle Databaseのアプリケーション開発について学習しようとするすべてのユーザーを対象とし、Oracle Databaseに初めて携わる開発者にアプリケーション開発の概要を示すことを主な目的としています。

このマニュアルは、リレーショナル・データベースの概念およびOracle Databaseでのアプリケーション開発に使用するオペレーティング・システム環境について理解していることを前提としています。

親トピック: はじめに

ドキュメントのアクセシビリティについて

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

Oracle Supportへのアクセス

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

親トピック: はじめに

関連ドキュメント

『*Oracle Database 2*日で開発者ガイド』で説明する概念およびタスクについて理解を深めた後、次のような他のOracle Database開発ガイドを参照することをお薦めします。

- 『*Oracle Application Express*アプリケーション・ビルダー・ユーザーズ・ガイド』
- 『*Oracle Database 2*日で*Java*開発者ガイド』

詳細は、次を参照してください。

- 『*Oracle Database*概要』
- 『*Oracle Database*開発ガイド』
- 『*Oracle Database SQL*言語リファレンス』
- 『*Oracle Database PL/SQL*言語リファレンス』

親トピック: はじめに

表記規則

『*Oracle Database 2*日で開発者ガイド』では次の表記規則を使用します。

規則	意味
太字	太字は、操作に関連する Graphical User Interface 要素、または本文中で定義されている用語および用語集に記載されている用語を示します。
イタリック体	イタリックは、ドキュメントのタイトル、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変数を示します。
固定幅フォント	固定幅フォントは、段落内のコマンド、URL、サンプル内のコード、画面に表示されるテキスト、または入力するテキストを示します。

親トピック: はじめに

1 2日でOracle Database開発の概要

Oracle Database開発者は、Oracleテクノロジ・スタックを使用するアプリケーションのデータベース・コンポーネントの作成またはメンテナンスを担当します。Oracle Database開発者は、アプリケーションを開発するか、または既存のアプリケーションを転用し、Oracle Database環境で実行します。

- このマニュアルについて
  このマニュアルはアプリケーション開発者を対象とするOracle Databaseのドキュメント・セットの一部です。
- Oracle Databaseについて
  Oracle Databaseは、関連情報を**スキーマ**と呼ばれる論理構造にグループ化します。論理構造には、**スキーマ・オブジェクト**が含まれます。
- サンプル・スキーマHRについて
  HRスキーマはサンプル・スキーマの1つで、Oracle Databaseとともにインストールできます。このスキーマには従業員に関する情報、つまり、部門、事業所、職歴に関する情報および他の関連情報が含まれています。すべてのスキーマと同様、HRにも表、ビュー、索引、プロシージャ、ファンクション、その他の属性があります。このマニュアルの例とチュートリアルでは、スキーマを使用します。

関連項目:

Oracle Database開発者の職務の詳細は、『*Oracle Database*概要』を参照してください。

1.1 このマニュアルについて

このガイドはアプリケーション開発者を対象とするOracle Databaseのドキュメント・セットの一部です。

このドキュメントでは、次を実行します。

- Oracle Databaseでの開発の基本概念について説明します。
- チュートリアルと例を使用して、SQLおよびPL/SQLの基本機能の使用方法を示します。
- 説明事項の詳細についての参照情報を提供します。
- 単純なOracle Databaseアプリケーションの開発およびデプロイ方法を示します。

「2日でOracle Database開発の概要」(この章)では、このガイドが対象とする読者に対し、ガイドの構成を説明します。また重要なOracle Databaseの概念について紹介し、チュートリアルと例で使用するサンプル・スキーマについて説明します。

Oracle Databaseへの接続および検索では、Oracle Databaseへの接続方法、スキーマ・オブジェクトおよびOracle Databaseの表のプロパティとデータの表示方法および問合せを使用してOracle Databaseの表からデータを取得する方法について説明します。

DML文とトランザクションでは、データ操作言語(DML)文とトランザクションについて紹介します。DML文は、Oracle Databaseの表データを追加、変更および削除します。トランザクションは、1つ以上のSQL文が並んだものであり、Oracle Databaseは、それを1つの単位として扱うので、すべての文が実行されるか何も実行されないことになります。

スキーマ・オブジェクトの作成と管理では、データ定義言語(DDL)について紹介します。DDLはスキーマ・オブジェクトを作成、変更、削除します。

ストアド・サブプログラムとパッケージの開発では、多くの異なるデータベース・アプリケーションでビルディング・ブロックとして使用でき

るストアド・サブプログラムとパッケージについて紹介します。

トリガーの使用では、指定されたイベントに対応して自動的に実行するストアドPL/SQLユニットであるトリガーについて紹介します。

グローバル環境での作業では、グローバリゼーション・サポート(National Language Support (NLS)パラメータとSQLとPL/SQLのUnicode関連機能)について紹介します。

有効なアプリケーションの作成では、スケーラブルなアプリケーションの作成方法および推奨されるプログラミングおよびセキュリティ上の措置の利用方法について説明します。

「単純なOracle Databaseアプリケーションの開発」では、単純なOracle Databaseアプリケーションの開発方法を示します。

「Oracle Databaseアプリケーションのデプロイ」では、例として「単純なOracle Databaseアプリケーションの開発」で開発したアプリケーションを使用して、Oracle Databaseアプリケーションのデプロイ方法(他のユーザーが実行できる1つ以上の環境へのインストール方法)について説明します。

親トピック: 2日でOracle Database開発の概要

1.2 Oracle Databaseについて

Oracle Databaseは、関連情報を**スキーマ**と呼ばれる論理構造にグループ化します。論理構造には、**スキーマ・オブジェクト**が含まれます。

ユーザー名とパスワードを使用してデータベースに接続する際に、ユーザーはスキーマを指定して、自分がその所有者であることを示します。Oracle Database では、ユーザー名とユーザーが接続するスキーマ名は同じです。

- スキーマ・オブジェクトについて
  Oracle Databaseのすべてのオブジェクトは、1つのスキーマのみに属し、そのスキーマを使用した一意の名前が付けられています。
- Oracle Databaseへのアクセスについて
  Oracle Databaseには、SQL*PlusやSQL Developerなどのクライアント・プログラムを介してのみアクセスできます。

親トピック: 2日でOracle Database開発の概要

1.2.1 スキーマ・オブジェクトについて

Oracle Databaseのすべてのオブジェクトは、1つのスキーマのみに属し、そのスキーマを使用した一意の名前が付けられています。

スキーマは、次のようなオブジェクトを持つことができます。

- **表**

  表は、Oracle Databaseのデータ記憶領域の基本単位です。表は、ユーザーがアクセス可能なすべてのデータを保持します。各表には、それぞれのデータの**レコード**を表す**行**が含まれています。行は、レコードの**フィールド**を表す**列**で構成されています。

- **索引**

  索引はオプションのオブジェクトであり、これを使用すると、表からデータを取得するパフォーマンスが改善されます。索引は、表の1つ以上の列に作成され、データベース内で自動的にメンテナンスされます。

- **ビュー**

  複数の表の情報を組み合せて一元的に表示するビューを作成できます。ビューは、表と他のビューの両方の情報に依

存する可能性があります。

- **順序**

  表のすべてのレコードが一意である必要がある場合、順序を使用することによって、各レコードのIDを表す数値列に対する一意の整数番号のシリアル・リストを生成できます。

- **シノニム**

  シノニムは、スキーマ・オブジェクトの別名です。これは、オブジェクトの所有者をわからなくしたり、SQL文を単純化するなど、セキュリティや利便性の向上に使用されます。

- **ストアド・サブプログラム**

  ストアド・サブプログラム(または、**スキーマ・レベル・サブプログラム**)は、データベースに格納されたプロシージャやファンクションです。これらは、データベースにアクセスするクライアント・アプリケーションから起動されます。

  トリガーは、特定の表またはビューで指定したイベントが発生した際にデータベースによって自動的に実行されるストアド・サブプログラムです。トリガーは、特定のデータへのアクセスを制限し、ロギングを実行できます。

- **パッケージ**

  パッケージは、ユニットとして継続的に使用するためにデータベースに格納された明示カーソルおよび変数とともに、それらを使用する関連サブプログラムをグループ化したものです。ストアド・サブプログラムと同じように、パッケージ・サブプログラムは、データベースにアクセスするクライアント・アプリケーションから起動されます。

通常、アプリケーションが使用するオブジェクトは、どれも同じスキーマに属しています。

関連項目:

- スキーマ・オブジェクトの詳細は、『*Oracle Database*概要』を参照してください。
- 表の作成および管理
- 索引の管理
- ビューの作成および管理
- 順序の作成および管理
- シノニムの作成および管理
- ストアド・サブプログラムとパッケージの開発
- トリガーの使用

親トピック: Oracle Databaseについて

1.2.2 Oracle Databaseへのアクセスについて

Oracle Databaseには、SQL*PlusやSQL Developerなどのクライアント・プログラムを介してのみアクセスできます。

Oracle Database に対するクライアント・プログラム・インタフェースは、構造化問合せ言語(SQL)です。Oracleには、Procedural Language/SQL(PL/SQL)というSQLに対する拡張機能が用意されています。

- SQL*Plusについて
  SQL*Plus (*sequel plus*と発音する)は、対話型のバッチ問合せツールです。Oracle Databaseをインストールす

ると、一緒にインストールされます。このツールには、データベースへの接続時にクライアントとして機能する、コマンドライン・ユーザー・インタフェースがあります。

- SQL Developerについて
  SQL Developer (*sequel developer*と発音する)は、Oracle Databaseのグラフィカル・ユーザー・インタフェースで、Oracle Databaseをインストールするとデフォルトでインストールされ、Oracle Technology Networkから無料でダウンロードできます。
- 構造化問合せ言語(SQL)について
  構造化問合せ言語(SQL) (*sequel*と発音する)は、セット・ベースの高度なコンピュータ言語であり、Oracle Databaseのデータにアクセスする際に、すべてのプログラムおよびユーザーによって使用されます。
- Procedural Language/SQL (PL/SQL)について
  Procedural Language/SQL (PL/SQL) (*P L sequel*と発音する)は、Oracle Databaseが独自にSQLに行った拡張です。条件付き制御やループのような手続き型の要素を追加することで、宣言型プログラム制御と命令型プログラム制御のギャップが埋められています。
- 他のクライアント・プログラム、言語および開発ツールについて
  他のいくつかのクライアント・プログラム、言語およびツールを利用できます。

親トピック: Oracle Databaseについて

1.2.2.1 SQL*Plusについて

SQL*Plusは、対話型のバッチ問合せツールです。Oracle Databaseをインストールすると、一緒にインストールされます。このツールには、データベースへの接続時にクライアントとして機能する、コマンドライン・ユーザー・インタフェースがあります。

SQL*Plusは独自のコマンドと環境を持っています。SQL*Plusの環境では、SQL*Plusコマンド、SQL文、PL/SQL文およびオペレーティング・システムのコマンドを入力して実行し、次のような作業を行えます。

- 問合せ結果の整形、計算実行、格納および印刷
- 表およびオブジェクト定義の検証
- バッチ・スクリプトの開発および実行
- データベース管理の実行

SQL*Plusを使用して、対話的なレポート生成、バッチ処理としてのレポート生成、およびテキスト・ファイル、スクリーンまたはインターネットでの閲覧用のHTMLファイルへの結果の出力が可能です。HTML出力機能を使用するとレポートを動的に生成できます。

SQL DeveloperでSQL*Plusを使用できます。詳細は、『*Oracle SQL Developer*ユーザーズ・ガイド』を参照してください。

関連項目:

- 「SQL*PlusからOracle Databaseへの接続」
- SQL*Plus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。

親トピック: Oracle Databaseへのアクセスについて

1.2.2.2 SQL Developerについて

SQL Developer (*sequel developer*と発音)は、Oracle Databaseのグラフィカル・ユーザー・インタフェースで、Oracle Databaseをインストールするとデフォルトでインストールされ、Oracle Technology Networkから無料でダウンロードできま

す。

SQL DeveloperはSQLおよびPL/SQLの最新の統合開発環境(IDE)として機能し、データベース・オブジェクトを管理するためのグラフィカル・インタフェースを提供します。レポートの作成、データ・モデルの設計、サードパーティ・データベースのOracleへの移行(REST対応の表およびビュー)、およびOracle REST Data Servicesのデプロイと管理を行うこともできます。SQLワークシートから、SQL文、PL/SQL文およびSQL*Plusコマンドとスクリプトを入力して実行できます。

注意:

SQL Developer では複数の方法でタスクを実行できますが、このマニュアルではすべての方法について説明しません。

関連項目:

- 「SQL DeveloperからOracle Databaseへの接続」
- 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。

親トピック: Oracle Databaseへのアクセスについて

1.2.2.3 構造化問合せ言語(SQL)について

構造化問合せ言語(SQL)(*sequel*と発音する)は、セット・ベースの高度なコンピュータ言語であり、Oracle Databaseのデータにアクセスする際に、すべてのプログラムおよびユーザーによって使用されます。

SQLは宣言型、または非手続き型の言語です。つまり、SQLは方法ではなく、何をするかについて記述します。結果を取得する方法ではなく、必要な結果セットを指定します(たとえば現在の従業員の名前)。

関連項目:

- SQLの概要は、『*Oracle Database概要*』を参照してください。
- 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。

親トピック: Oracle Databaseへのアクセスについて

1.2.2.4 Procedural Language/SQL(PL/SQL)について

Procedural Language/SQL (PL/SQL)は、Oracle Databaseが独自にSQLに行った拡張です。条件付き制御やループのような手続き型の要素を追加することで、宣言型プログラム制御と命令型プログラム制御のギャップが埋められています。

PL/SQLでは、定数と変数、プロシージャとファンクション、型とその型の変数、およびトリガーを宣言できます。例外も処理できます(ランタイム・エラー)。また、Oracleのデータベース・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で再利用することを目的としてデータベースに格納できるプロシージャ、ファンクション、パッケージ、タイプおよびトリガーなどのPL/SQLユニットを作成できます。

PL/SQLソース・プログラムの基本単位は、関連する宣言と文をグループ化するブロックです。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。

関連項目:

- PL/SQLの概要は、『*Oracle Database概要*』を参照してください。
- 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。

親トピック: Oracle Databaseへのアクセスについて

1.2.2.5 他のクライアント・プログラム、言語および開発ツールについて

他のいくつかのクライアント・プログラム、言語および開発ツールを利用できます。

> **注意:**
>
> 前述のリストの一部の製品は Oracle Database に付属していないため、別個にダウンロードする必要があります。

- Oracle Application Express
  Oracle Application Expressは、プログラミングの経験が浅い開発者でも、短期間で確実かつスケーラブルなWebアプリケーションを作成できるアプリケーション開発およびデプロイ・ツールです。埋め込まれているApplication Builderによって、表やストアド・プロシージャなどのスキーマ・オブジェクトを使用する完全なアプリケーションまたはHTMLインタフェースが、タブ、ボタン、ハイパーテキスト・リンクを介してリンクされたページの集まりに組み立てられます。
- Oracle Java Database Connectivity (JDBC)
  Oracle Java Database Connectivity(JDBC)は、JavaでSQL文をOracle Databaseなどのオブジェクト・リレーショナル・データベースに送ることができるようにするAPIです。Oracle DatabaseのJDBCは、JDBC 3.0およびJDBC RowSet(JSR-114)規格、XAおよび非XA接続に対応した先進の接続キャッシュ、SQLおよびPL/SQLデータ型のJavaへの公開、SQLデータへの迅速なアクセスを完全にサポートします。
- Hypertext Preprocessor (PHP)
  Hypertext Preprocessor (PHP)は、Webアプリケーションを迅速に開発するための、強力なインタプリタ型のサーバー側スクリプト言語です。PHPはオープン・ソース言語であり、BSD型ライセンスで配布されています。PHPはデータベースのアクセス要求を直接HTMLページに組み込めるよう設計されています。
- Oracle Call Interface (OCI)
  Oracle Call Interface (OCI)は、C言語のアプリケーションからOracle Databaseに直接アクセスするための、独自のC言語APIです。
- Oracle C++ Call Interface (OCCI)
  Oracle C++ Call Interface (OCCI)は、C++言語のアプリケーションからOracle Databaseに直接アクセスするための、独自のC++言語APIです。OCIと同様、OCCIはリレーショナルおよびオブジェクト指向型のプログラミング・パラダイムをサポートしています。
- Open Database Connectivity (ODBC)
  Open Database Connectivity (ODBC)は、データベースにアクセスするためのAPIのセットで、データベースに接続してデータベース上でSQL文を実行します。ODBCドライバを使用するアプリケーションは、スプレッドシートやカンマ区切りファイルなど、不均一なデータ・ソースにアクセスできます。

- Pro*C/C++プリコンパイラ
  Pro*C/C++プリコンパイラを使用すると、CまたはC++ソース・ファイルにSQL文を埋め込むことができます。プリコンパイラでは、ソース・プログラムを入力として受け入れ、埋込みSQL文を標準のOracleランタイム・ライブラリ・コールに変換し、コンパイル、リンクおよび実行ができるように変更したソース・プログラムを生成します。
- Pro*COBOLプリコンパイラ
  Pro*COBOLプリコンパイラを使用すると、COBOLソース・ファイルにSQL文を埋め込むことができます。プリコンパイラでは、ソース・プログラムを入力として受け入れ、埋込みSQL文を標準のOracleランタイム・ライブラリ・コールに変換し、コンパイル、リンクおよび実行ができるように変更したソース・プログラムを生成します。
- Microsoft .NET Framework
  Microsoft .NET Frameworkは、アプリケーションとXML Webサービスを作成、デプロイおよび実行するための多言語環境です。
- Oracle Provider for OLE DB (OraOLEDB)
  Oracle Provider for OLE DB(OraOLEDB)は、オープンかつ標準的なデータ・アクセス方法であり、様々な型のデータに対するアクセスおよび操作において、一連のComponent Object Model(COM)インタフェースを使用します。このインタフェースは様々なデータベース開発元から提供されています。

関連項目:

- Oracle Database開発者用のツールの詳細は、『*Oracle Database*概要』を参照してください。
- プログラミング環境の選択の詳細は、『*Oracle Database*開発ガイド』を参照してください。

親トピック: Oracle Databaseへのアクセスについて

1.2.2.5.1 Oracle Application Express

Oracle Application Expressは、プログラミングの経験が浅い開発者でも、短期間で確実かつスケーラブルなWebアプリケーションを開発および配置できるツールです。埋め込まれているApplication Builderによって、表やストアド・プロシージャなどのスキーマ・オブジェクトを使用する完全なアプリケーションまたはHTMLインタフェースが、タブ、ボタン、ハイパーテキスト・リンクを介してリンクされたページの集まりに組み立てられます。

関連項目:

Oracle Application Express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。

親トピック: 他のクライアント・プログラム、言語および開発ツールについて

1.2.2.5.2 Oracle Java Database Connectivity(JDBC)

Oracle Java Database Connectivity (JDBC)は、JavaでSQL文をOracle Databaseのようなオブジェクト・リレーショナル・データベースに送ることができるようにするAPIです。Oracle DatabaseのJDBCは、JDBC 3.0およびJDBC RowSet(JSR-114)規格、XAおよび非XA接続に対応した先進の接続キャッシュ、SQLおよびPL/SQLデータ型のJavaへの公開、SQLデータへの迅速なアクセスを完全にサポートします。

関連項目:

JDBCの詳細は、次を参照してください。

- 『*Oracle Database*概要』
- 『*Oracle Database*開発ガイド』
- 『*Oracle Database 2*日で*Java*開発者ガイド』

親トピック: 他のクライアント・プログラム、言語および開発ツールについて

1.2.2.5.3 Hypertext Preprocessor (PHP)

Hypertext Preprocessor (PHP)は、Webアプリケーションを迅速に開発するための、強力なインタプリタ型のサーバー側スクリプト言語です。PHPはオープン・ソース言語であり、BSD型ライセンスで配布されています。PHPはデータベースのアクセス要求を直接HTMLページに組み込めるよう設計されています。

親トピック: 他のクライアント・プログラム、言語および開発ツールについて

1.2.2.5.4 Oracle Call Interface(OCI)

Oracle Call Interface (OCI)はC言語のアプリケーションからOracle Databaseに直接アクセスするための、独自のC言語のAPIです。

OCIソフトウェア開発キットは、Oracle Instant Clientの一部としてインストールでき、標準のOracleクライアントをインストールしたりORACLE_HOMEを持つことなくアプリケーションを実行できます。アプリケーションは変更せずに使用でき、ディスク領域の使用を大幅に抑えることができます。

関連項目:

- OCIの詳細は、『*Oracle Database*開発ガイド』を参照してください。
- OCIの完全な情報は、『*Oracle Call Interface*プログラマーズ・ガイド』を参照してください。

親トピック: 他のクライアント・プログラム、言語および開発ツールについて

1.2.2.5.5 Oracle C++ Call Interface(OCCI)

Oracle C++ Call Interface (OCCI)は、C++アプリケーションからOracle Databaseに直接アクセスするための、独自のC++言語のAPIです。OCIと同様、OCCIはリレーショナルおよびオブジェクト指向型のプログラミング・パラダイムをサポートしています。

OCCIソフトウェア開発キットは、Oracle Instant Clientの一部としてインストールでき、標準のOracleクライアントをインストールしたりORACLE_HOMEを持つことなくアプリケーションを実行できます。アプリケーションは変更せずに使用でき、ディスク領域の使用を大幅に抑えることができます。

関連項目:

- OCCI の詳細は、『*Oracle Database*開発ガイド』を参照してください。
- OCCIの完全な情報は、『*Oracle C++ Call Interface*プログラマーズ・ガイド』を参照してください。

親トピック: 他のクライアント・プログラム、言語および開発ツールについて

1.2.2.5.6 Open Database Connectivity

Open Database Connectivity (ODBC)は、データベースにアクセスするためのAPIのセットで、データベースに接続してデータベース上でSQL文を実行します。ODBCドライバを使用するアプリケーションは、スプレッドシートやカンマ区切りファイルなど、不均一なデータ・ソースにアクセスできます。

Oracle ODBC DriverはODBC 3.51仕様に準拠します。すべてのコアAPIやLevel 1およびLevel 2のファンクションのサブセットをサポートしています。Microsoft社はWindowsプラットフォーム用のドライバ・マネージャ・コンポーネントを提供しています。

OCI、OCCIおよびJDBCと同様に、ODBCはOracle Instant Client Installationの一部です。

関連項目:

- 『*Oracle Database*概要』
- WindowsでOracle ODBCドライバを使用する方法の詳細は、『*Oracle Services for Microsoft Transaction Server*開発者ガイド *for Microsoft Windows*』を参照してください。
- Oracle ODBCドライバをLinuxで使用する方法については、『*Oracle Database*管理者リファレンス*for Linux and UNIX-Based Operating Systems*』を参照してください。

親トピック: 他のクライアント・プログラム、言語および開発ツールについて

1.2.2.5.7 Pro*C/C++プリコンパイラ

Pro*C/C++プリコンパイラにより、C言語またはC++言語のソース・ファイルにSQL文を組み込むことができます。プリコンパイラでは、ソース・プログラムを入力として受け入れ、埋込みSQL文を標準のOracleランタイム・ライブラリ・コールに変換し、コンパイル、リンクおよび実行ができるように変更したソース・プログラムを生成します。

関連項目:

- Oracleプリコンパイラの詳細は、『*Oracle Database*概要』を参照してください。
- Pro*C/C++プリコンパイラの詳細は、『*Oracle Database*開発ガイド』を参照してください。
- 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。

親トピック: 他のクライアント・プログラム、言語および開発ツールについて

1.2.2.5.8 Pro*COBOLプリコンパイラ

Pro*COBOLプリコンパイラにより、COBOL言語のソース・ファイルにSQL文を組み込むことができます。プリコンパイラでは、ソース・プログラムを入力として受け入れ、埋込みSQL文を標準のOracleランタイム・ライブラリ・コールに変換し、コンパイル、リンクおよび実行ができるように変更したソース・プログラムを生成します。

関連項目:

- Oracleプリコンパイラの詳細は、『*Oracle Database*概要』を参照してください。
- Pro*COBOLプリコンパイラの詳細は、『*Oracle Database*開発ガイド』を参照してください。

- 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。

親トピック: 他のクライアント・プログラム、言語および開発ツールについて

1.2.2.5.9 Microsoft .NET Framework

Microsoft .NET Frameworkは、アプリケーションとXML Webサービスを作成、デプロイおよび実行するための多言語環境です。

Microsoft .NET Frameworkのメイン・コンポーネントは次のとおりです。

- 共通言語ランタイム(CLR)

  共通言語ランタイム(CLR)とは中間言語の開発環境と実行環境で、実行されているアプリケーションの管理を補助します。

- Frameworkクラス・ライブラリ(FCL)

  Frameworkクラス・ライブラリ(FCL)は、パッケージ済機能の一貫したオブジェクト指向のライブラリを提供します。

Oracle Data Provider for .NET(ODP.NET)

Oracle Data Provider for .NET (ODP.NET)は、.NETクライアント・アプリケーションからOracle Databaseへの迅速で有効なADO.NETデータ・アクセスを提供します。ODP.NETにより、開発者はセキュア・ファイル、XML DB、アドバンスト・キューイングを含むOracle Databaseの高度なOracle Database機能を利用できます。

Visual Studio対応Oracle Developer Tools(ODT)

Visual Studio対応Oracle Developer Tools(ODT)は、Visual Studio環境と統合されるアプリケーション・ツールのセットです。これらのツールは、Oracleの機能にアクセスするためのグラフィカル・ユーザー・インタフェースを提供し、広範なアプリケーション開発作業を可能にし、開発上の生産性および使い易さを向上させます。Oracle Developer Toolsは、Visual Basic、C#および他の.NET言語を使用した.NETストアド・プロシージャのプログラミングと実装をサポートしています。

.NETストアド・プロシージャ

Oracle Database Extensions for .NETは、WindowsでのOracle Databaseのデータベース・オプションです。このオプションによって、Microsoft Windows用のOracle Databaseを使用する.NETストアド・プロシージャまたはファンクションを、Visual Basic .NETまたはVisual C#を使用して作成および実行できるようになります。

.NETアセンブリに.NETプロシージャおよびファンクションを作成した後は、Oracle Developer Tools for Visual Studioのコンポーネントであるオラクル Deployment Wizard for .NETを使用してそれらをOracle Databaseにデプロイできます。

Oracle Providers for ASP.NET

Oracle Providers for ASP.NETは、ASP.NET開発者に対して、Oracle Database内のWebアプリケーションに共通の状態を簡単に格納できる方法を提供します。これらのプロバイダは、既存のMicrosoft ASP.NETプロバイダにモデル化され、類似するスキーマおよびプログラミング・インタフェースを共有して.NET開発者に使い慣れたインタフェースを提供します。Oracleでは、メンバーシップ、プロファイル、ロールなどのプロバイダがサポートされます。

関連項目:

- *Oracle Data Provider for .NET*開発者ガイド*for Microsoft Windows*
- 『*Oracle Database Extensions for .NET*開発者ガイド*for Microsoft Windows*』
- 『*Oracle Database*開発ガイド』

親トピック: 他のクライアント・プログラム、言語および開発ツールについて

1.2.2.5.10 Oracle Provider for OLE DB(OraOLEDB)

Oracle Provider for OLE DB (OraOLEDB)は、Component Object Model (COM)インタフェースのセットを使用して異なるタイプのデータにアクセスしたり操作を行うオープンな標準データ・アクセス方式です。このインタフェースは様々なデータベース開発元から提供されています。

関連項目:

OraOLEDBの詳細は、『*Oracle Provider for OLE DB*開発者ガイド *for Microsoft Windows*』を参照してください。

親トピック: 他のクライアント・プログラム、言語および開発ツールについて

1.3 サンプル・スキーマHRについて

HRサンプル・スキーマは、Oracle Databaseとともにインストールできます。このスキーマには従業員に関する情報、つまり、部門、事業所、職歴に関する情報および他の関連情報が含まれています。すべてのスキーマと同様、HRにも表、ビュー、索引、プロシージャ、ファンクション、その他の属性があります。このマニュアルの例とチュートリアルでは、スキーマを使用します。

関連項目:

- HR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。
- ユーザーHRとしてOracle Databaseに接続する方法の詳細は、「ユーザーHRとしてのOracle Databaseへの接続」を参照してください。

親トピック: 2日でOracle Database開発の概要

2 Oracle Databaseへの接続および検索

Oracle Databaseへは、SQL*PlusやSQL Developerなどのクライアント・プログラムからのみ接続できます。データベースに接続すると、スキーマ・オブジェクトを表示したり、Oracle Databaseの表のプロパティとデータを表示したり、問合せを使用してOracle Databaseの表からデータを取得できます。

クライアント・プログラムを使用してOracle Databaseに接続した後、そのクライアント・プログラムのコマンドを入力して実行します。詳細は、クライアント・プログラムのマニュアルを参照してください。

- SQL*PlusからOracle Databaseへの接続
  SQL*Plusは、Oracle Databaseにアクセス可能なクライアント・プログラムです。このトピックでは、SQL*Plusを起動し、Oracle Databaseに接続する方法を示します。
- SQL DeveloperからOracle Databaseへの接続
  SQL Developerは、Oracle Databaseにアクセス可能なクライアント・プログラムです。
- ユーザーHRとしてのOracle Databaseへの接続
  このドキュメントのチュートリアルと例を実行するには、ユーザーHRとしてOracle Databaseに接続する必要があります。
- SQL*PlusによるOracle Databaseの参照
  ユーザーHRとしてSQL*PlusからOracle Databaseに接続する場合、HRスキーマ・オブジェクトとEMPLOYEES表のプロパティを表示できます。
- SQL DeveloperによるOracle Databaseの参照
  ユーザーHRとしてSQL DeveloperからOracle Databaseに接続する場合、HRスキーマ・オブジェクトとEMPLOYEES表のプロパティを表示できます。
- 表データの選択

2.1 SQL*PlusからOracle Databaseへの接続

SQL*Plusは、Oracle Databaseにアクセスできるクライアント・プログラムです。このトピックでは、SQL*Plusを起動し、Oracle Databaseに接続する方法を示します。

注意:

次の手順の 3 と 4 では、ユーザー名とパスワードが必要です。

SQL*PlusからOracle Databaseへ接続するには、次の手順を実行します。

1. Windowsシステムで行う場合は、Windowsのコマンド・プロンプトを表示します。
2. コマンド・プロンプトで、sqlplusと入力してから[Enter]キーを押します。
3. ユーザー名のプロンプトで、ユーザー名を入力してから[Enter]キーを押します。
4. パスワードのプロンプトで、パスワードを入力してから[Enter]キーを押します。

注意:

セキュリティのため、パスワードは画面上に表示されません。

Oracle Databaseインスタンスに接続されます。

SQL*Plus環境に入っています。 SQL>プロンプトで、SQL*Plusコマンド、SQL文、PL/SQL文およびオペレーティング・システムのコマンドを入力して実行できます。

SQL*Plusを終了するには、exitと入力して[Enter]キーを押します。

注意:

SQL*Plus を終了すると SQL*Plus セッションは終了されますが、Oracle Database インスタンスは停止されません。

例2-1では、SQL*Plusを起動し、Oracle Databaseに接続し、SQLのSELECT文を実行して、SQL*Plusを終了します。ユーザーの入力は**太字**になっています。

例2-1 SQL*PlusからOracle Databaseへの接続

```
> sqlplus
SQL*Plus: Release 12.1.0.1.0 Production on Thu Dec 27 07:43:41 2012

Copyright (c) 1982, 2012, Oracle.  All rights reserved.

Enter user-name: your_user_name
Enter password: your_password

Connected to:
Oracle Database 12c Enterprise Edition Release - 12.1.0.1.0 64bit Production

SQL> select count(*) from employees;

  COUNT(*)
----------
       107

SQL> exit

Disconnected from Oracle Database 12c Enterprise Edition Release - 12.1.0.1.0 64bit Production
>
```

関連項目:

- 「ユーザーHRとしてのSQL*PlusからOracle Databaseへの接続」
- SQL*Plusの概要は、「SQL*Plusについて」を参照してください。
- SQL*Plusの起動とOracle Databaseへの接続の詳細は、『*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。

2.2 SQL DeveloperからOracle Databaseへの接続

SQL Developerは、Oracle Databaseにアクセス可能なクライアント・プログラムです。

現在利用可能なリリースのSQL Developerを次のサイトからダウンロードして使用できます。

http://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/

この項では、SQL Developerがシステムにインストールされていることを前提として、SQL Developerを起動し、Oracle Databaseに接続する方法を示します。SQL Developerがインストールされていないシステムの場合、インストール手順の詳細は、『Oracle SQL Developerユーザーズ・ガイド』を参照してください。

注意:

この後の手順は、次のようになっています。

- JDK を含まない SQL Developer キットを使用しており、それを初めてシステムで起動するときには、Java Development Kit へのパスを指定する必要があります。
- パスワードの名前とパスワードの入力を求められた場合は、入力する必要があります。

SQL DeveloperからOracle Databaseに接続するには、次の手順を実行します。

1. SQL Developerを起動します。

   手順は、『*Oracle SQL Developer*ユーザーズ・ガイド』を参照してください。

   システムでSQL Developerを最初に起動する場合、Java Development Kit (JDK)インストールへのパスを入力するよう求められます(例: C:¥Program Files¥Java¥jdk1.8.0_65)。プロンプトに続けてパスを入力するか、または参照を行い、[Enter]キーを押します。

2. 「接続」フレームで、「接続の作成」アイコンをクリックします。

3. 「データベース接続の作成/選択」ウィンドウで、次の手順を実行します。

   a. 「接続名」、「ユーザー名」および「パスワード」の各フィールドに適切な値を入力します。

      セキュリティのため、入力したパスワードの文字はアスタリスクで表示されます。

      「パスワード」フィールドの近くに「パスワードの保存」チェック・ボックスがあります。デフォルトでは、選択解除されています。デフォルトを受け入れることをお薦めします。

   b. 「Oracle」ペインが表示されない場合は、「Oracle」タブをクリックします。

   c. 「Oracle」ペインでデフォルト値を受け入れます。

      (デフォルト値は、接続タイプ: 基本、ロール: デフォルト、ホスト名: localhost、ポート: 1521、SIDオプション: 選択、SIDフィールド: xeです。)

   d. 「テスト」ボタンをクリックします。

      接続がテストされます。接続が成功すると、「ステータス」インジケータが空白から「成功」に変わります。

e. テストが成功したら、「接続」をクリックします。

「データベース接続の作成/選択」ウィンドウが閉じます。「接続」フレームに、手順3で「接続名」フィールドに入力した名前の接続が表示されます。

SQL Developer環境に入っています。

SQL Developerを終了するには、「ファイル」メニューから「終了」を選択します。

> **注意:**
>
> SQL Developer を終了すると SQL Developer セッションは終了されますが、Oracle Database インスタンスは停止されません。次回 SQL Developer を起動するときには、これまでの手順で作成した接続が残っています。SQL Developer では、手順 3 で入力したパスワードを要求されます(「パスワードの保存」チェック・ボックスを選択していない場合のみ)。

関連項目:

- 「ユーザーHRとしてのSQL DeveloperからOracle Databaseへの接続」
- SQL Developerの概要は、「SQL Developerについて」を参照してください。
- SQL Developerを使用したOracle Databaseへの接続の作成の詳細は、『*Oracle SQL Developer*ユーザーズ・ガイド』を参照してください。

親トピック: Oracle Databaseへの接続および検索

2.3 ユーザーHRとしてのOracle Databaseへの接続

このドキュメントのチュートリアルと例を実行するには、ユーザーHRとしてOracle Databaseに接続する必要があります。

ユーザーHRは、このマニュアルの例とチュートリアルで使用されるHRサンプル・スキーマを所有します。

- HRアカウントのロック解除
  ユーザーHRとしてOracle Databaseに接続するには、その前にHRアカウントのロックを解除し、パスワードをリセットする必要があります。
- ユーザーHRとしてのSQL*PlusからOracle Databaseへの接続
  SQL*Plusを使用してHRユーザーとしてOracle Databaseに接続できます。
- ユーザーHRとしてのSQL DeveloperからOracle Databaseへの接続
  SQL Developerを使用してHRユーザーとしてOracle Databaseに接続できます。

親トピック: Oracle Databaseへの接続および検索

2.3.1 HRアカウントのロック解除

ユーザーHRとしてOracle Databaseに接続するには、その前にHRアカウントのロックを解除し、パスワードをリセットする必要があります。

デフォルトでは、HRスキーマがインストールされた際にHRアカウントがロックされ、パスワードが期限切れになります。

> **注意:**
>
> 次の手順では、ALTER USER システム権限を持つユーザーの名前とパスワードが必要です。

hrアカウントをロック解除し、パスワードをリセットするには、次の手順を実行します。

1. SQL*Plusを使用して、ALTER USERのシステム権限があるユーザーとしてOracle Databaseに接続します。
2. SQL>プロンプトで、HRアカウントをロック解除し、パスワードをリセットします。

> **注意:**
>
> 安全なパスワードを選択してください。セキュアなパスワードの詳細は、『*Oracle Database セキュリティ・ガイド*』を参照してください。

```
ALTER USER HR ACCOUNT UNLOCK IDENTIFIED BY password;
```

システムは次のように応答します。

```
User altered.
```

HRアカウントがロック解除され、パスワードが**password**になりました。

これでユーザーHRとして、パスワード**password**を使用して、Oracle Databaseに接続できます。

関連項目:

- SQL DeveloperでのSQL*Plusの利用の詳細は、『*Oracle SQL Developerユーザーズ・ガイド*』を参照してください。

親トピック: ユーザーHRとしてのOracle Databaseへの接続

2.3.2 ユーザーHRとしてのSQL*PlusからOracle Databaseへの接続

SQL*Plusを使用してHRユーザーとしてOracle Databaseに接続できます。

> **注意:**
>
> HR アカウントがロックされている場合は、「HR アカウントのロック解除」を参照してから、この項に戻ってください。

ユーザーHRとして、SQL*PlusからOracle Databaseへ接続するには、次の手順を実行します。

注意:

この作業では、HR アカウントのパスワードが必要です。

1. Oracle Databaseに接続している場合は、現在の接続をクローズします。
2. 「SQL*PlusからOracle Databaseへの接続」の指示に従って、手順3でユーザー名HRを入力し、手順4でHRアカウントのパスワードを入力します。

   これで、ユーザーHRとしてOracle Databaseに接続しました。

関連項目:

SQL*Plusを使用したHRの接続の作成例は、『*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。

親トピック: ユーザーHRとしてのOracle Databaseへの接続

2.3.3 ユーザーHRとしてのSQL DeveloperからOracle Databaseへの接続

SQL Developerを使用してHRユーザーとしてOracle Databaseに接続できます。

注意:

HR アカウントがロックされている場合は、「HR アカウントのロック解除」を参照してから、この項に戻ってください。

ユーザーHRとしてSQL DeveloperからOracle Databaseに接続するには、次の手順を実行します。

注意:

この作業では、HR アカウントのパスワードが必要です。

「SQL DeveloperからOracle Databaseへの接続」の指示に従って、手順3で次の値を入力します。

- 「接続名」には、hr_connと入力します。

  (異なる名前を入力できますが、このマニュアルのチュートリアルでは、接続にhr_connという名前を付けたと想定します。)
- 「ユーザー名」には、HRと入力します。
- 「パスワード」には、HRアカウントのパスワードを入力します。

これで、ユーザーHRとしてOracle Databaseに接続しました。

親トピック: ユーザーHRとしてのOracle Databaseへの接続

2.4 SQL*PlusによるOracle Databaseの参照

ユーザーHRとしてSQL*PlusからOracle Databaseに接続する場合、HRスキーマ・オブジェクトとEMPLOYEES表のプロパティを表示できます。

> **注意:**
>
> HR として SQL*Plus から Oracle Database に接続していない場合、「ユーザーHR として SQL*Plus から Oracle Database への接続」を参照してから、この項に戻ってください。

- SQL*PlusによるHRスキーマ・オブジェクトの表示
  SQL*Plusを使用して、静的データ・ディクショナリ・ビューUSER_OBJECTSに問い合せ、HRスキーマに属するオブジェクトを表示できます。
- SQL*PlusによるEMPLOYEES表のプロパティとデータの表示
  SQL*Plusコマンド、SQL SELECT文および静的データ・ディクショナリ・ビューを使用してHR.EMPLOYEES表のプロパティとデータを表示できます。

親トピック: Oracle Databaseへの接続および検索

2.4.1 SQL*PlusによるHRスキーマ・オブジェクトの表示

SQL*Plusを使用して、静的データ・ディクショナリ・ビューUSER_OBJECTSに問い合せ、HRスキーマに属するオブジェクトを表示できます。

例2-2では、HRスキーマに属するオブジェクトの名前およびデータ型を表示する方法を示しています。

例2-2 SQL*PlusによるHRスキーマ・オブジェクトの表示

```
COLUMN OBJECT_NAME FORMAT A25
COLUMN OBJECT_TYPE FORMAT A25

SELECT OBJECT_NAME, OBJECT_TYPE FROM USER_OBJECTS
ORDER BY OBJECT_TYPE, OBJECT_NAME;
```

結果は次のようになります。

```
OBJECT_NAME               OBJECT_TYPE
------------------------- -------------------------
COUNTRY_C_ID_PK           INDEX
DEPT_ID_PK                INDEX
DEPT_LOCATION_IX          INDEX
EMP_DEPARTMENT_IX         INDEX
EMP_EMAIL_UK              INDEX
EMP_EMP_ID_PK             INDEX
EMP_JOB_IX                INDEX
EMP_MANAGER_IX            INDEX
EMP_NAME_IX               INDEX
JHIST_DEPARTMENT_IX       INDEX
JHIST_EMPLOYEE_IX         INDEX
JHIST_EMP_ID_ST_DATE_PK   INDEX
JHIST_JOB_IX              INDEX
```

```
JOB_ID_PK                 INDEX
LOC_CITY_IX               INDEX
LOC_COUNTRY_IX            INDEX
LOC_ID_PK                 INDEX
LOC_STATE_PROVINCE_IX     INDEX
REG_ID_PK                 INDEX
ADD_JOB_HISTORY           PROCEDURE
SECURE_DML                PROCEDURE
DEPARTMENTS_SEQ           SEQUENCE
EMPLOYEES_SEQ             SEQUENCE
LOCATIONS_SEQ             SEQUENCE
COUNTRIES                 TABLE
DEPARTMENTS               TABLE
EMPLOYEES                 TABLE
JOBS                      TABLE
JOB_HISTORY               TABLE
LOCATIONS                 TABLE
REGIONS                   TABLE
SECURE_EMPLOYEES          TRIGGER
UPDATE_JOB_HISTORY        TRIGGER
EMP_DETAILS_VIEW          VIEW

34 rows selected.
```

関連項目:

- USER_OBJECTSの詳細は、『Oracle Databaseリファレンス』を参照してください。
- 問合せを使用した表データの表示の詳細は、「表データの選択」を参照してください。
- スキーマHRの一般情報は、"「サンプル・スキーマHRについて」を参照してください。

親トピック: SQL*PlusによるOracle Databaseの参照

2.4.2 SQL*PlusによるEMPLOYEES表のプロパティとデータの表示

SQL*Plusコマンド、SQL SELECT文および静的データ・ディクショナリ・ビューを使用してHR.EMPLOYEES表のプロパティとデータを表示できます。

SQL*PlusコマンドDESCRIBEを使用してHRスキーマのEMPLOYEES表の列のプロパティを表示したり、SQL文SELECTを使用してデータを表示したりできます。表の別のプロパティを表示するには、静的データ・ディクショナリ・ビュー(USER_CONSTRAINTS、USER_INDEXES、USER_TRIGGERSなど)を使用します。

例2-3では、HRスキーマのEMPLOYEES表のプロパティを表示する方法を示しています。

例2-3 SQL*PlusによるEMPLOYEES表のプロパティの表示

```
DESCRIBE EMPLOYEES
```

結果:

```
Name                                      Null?    Type
----------------------------------------- -------- ------------

EMPLOYEE_ID                               NOT NULL NUMBER(6)
FIRST_NAME                                         VARCHAR2(20)
LAST_NAME                                 NOT NULL VARCHAR2(25)
```

```
EMAIL                                     NOT NULL VARCHAR2(25)
PHONE_NUMBER                                       VARCHAR2(20)
HIRE_DATE                                 NOT NULL DATE
JOB_ID                                    NOT NULL VARCHAR2(10)
SALARY                                             NUMBER(8,2)
COMMISSION_PCT                                     NUMBER(2,2)
MANAGER_ID                                         NUMBER(6)
DEPARTMENT_ID                                      NUMBER(4)
```

例2-4では、HRスキーマのEMPLOYEES表のデータの一部を表示する方法を示しています。

例2-4 SQL*PlusによるEMPLOYEES表のデータの表示

```
COLUMN FIRST_NAME FORMAT A20
COLUMN LAST_NAME FORMAT A25
COLUMN PHONE_NUMBER FORMAT A20

SELECT LAST_NAME, FIRST_NAME, PHONE_NUMBER FROM EMPLOYEES
ORDER BY LAST_NAME;
```

結果は次のようになります。

```
LAST_NAME                 FIRST_NAME           PHONE_NUMBER
------------------------- -------------------- --------------------
Abel                      Ellen                011.44.1644.429267
Ande                      Sundar               011.44.1346.629268
Atkinson                  Mozhe                650.124.6234
Austin                    David                590.423.4569
Baer                      Hermann              515.123.8888
Baida                     Shelli               515.127.4563
Banda                     Amit                 011.44.1346.729268
Bates                     Elizabeth            011.44.1343.529268
...
Urman                     Jose Manuel          515.124.4469
Vargas                    Peter                650.121.2004
Vishney                   Clara                011.44.1346.129268
Vollman                   Shanta               650.123.4234
Walsh                     Alana                650.507.9811
Weiss                     Matthew              650.123.1234
Whalen                    Jennifer             515.123.4444
Zlotkey                   Eleni                011.44.1344.429018

107 rows selected.
```

関連項目:

- DESCRIBE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。
- 問合せを使用した表データの表示の詳細は、「表データの選択」を参照してください。
- 静的データ・ディクショナリ・ビューの詳細は、『*Oracle Database*リファレンス』を参照してください。

親トピック: SQL*PlusによるOracle Databaseの参照

2.5 SQL DeveloperによるOracle Databaseの参照

ユーザーHRとしてSQL DeveloperからOracle Databaseに接続する場合、HRスキーマ・オブジェクトとEMPLOYEES表のプロパティを表示できます。

- チュートリアル: SQL DeveloperによるHRスキーマ・オブジェクトの表示
  このチュートリアルではSQL Developerを使用してHRスキーマに属するオブジェクトの表示方法(つまり、HRスキーマを**参照**する方法)を説明します。
- チュートリアル: SQL DeveloperによるEMPLOYEES表のプロパティとデータの表示
  このチュートリアルでは、SQL Developerを使用してHRスキーマのEMPLOYEES表のプロパティとデータを確認する方法を説明します。

親トピック: Oracle Databaseへの接続および検索

2.5.1 チュートリアル: SQL DeveloperによるHRスキーマ・オブジェクトの表示

このチュートリアルではSQL Developerを使用してHRスキーマに属するオブジェクトを表示する方法(つまり、**HRスキーマを**参照する方法)を説明します。

> 注意:
>
> ユーザーHR として SQL Developer から Oracle Database に接続していない場合、「ユーザーHR としての SQL Developer から Oracle Database への接続」を参照してから、このチュートリアルに戻ってください。

HRスキーマを参照するには、次の手順を実行します。

1. 「接続」フレームで、hr_connアイコンの左にあるプラス記号(+)をクリックします。

   データベースに接続していない場合は、「接続情報」ウィンドウが開きます。データベースに接続している場合は、hr_connの情報が展開されます(手順2の「「OK」をクリックします」の後の情報を参照)。

2. 「接続情報」ウィンドウが開かれた場合は、次のようにします。

   a. 「ユーザー名」フィールドに、hrと入力します。

   b. 「パスワード」フィールドに、ユーザーHRのパスワードを入力します。

   c. 「OK」をクリックします。

   hr_connの情報を展開します。プラス記号がマイナス記号(-)に変わり、hr_connアイコンの下に「表」、「ビュー」、「索引」などの項目があるスキーマ・オブジェクト・タイプのリストが表示されます(マイナス記号をクリックすると、hr_connの情報が閉じます。マイナス記号はプラス記号に変わり、リストは非表示になります)。

関連項目:

- SQL Developerユーザー・インタフェースの詳細は、『*Oracle SQL Developer*ユーザーズ・ガイド』を参照してください
- スキーマHRの一般情報は、「サンプル・スキーマHRについて」を参照してください。

親トピック: SQL DeveloperによるOracle Databaseの参照

2.5.2 チュートリアル: SQL DeveloperによるEMPLOYEES表のプロパティとデータの表示

このチュートリアルでは、SQL Developerを使用してHRスキーマのEMPLOYEES表のプロパティとデータを確認する方法を説明します。

> 注意:
>
> HR スキーマを参照していない場合、「チュートリアル: SQL Developer による HR スキーマ・オブジェクトの表示」の手順を実行してから、このチュートリアルに戻ってください。

EMPLOYEES表のプロパティとデータを表示するには、次の手順を実行します。

1. 「接続情報」フレームで、「表」を展開します。

   「表」,の下に、HRスキーマにある表のリストが表示されます。

2. 表EMPLOYEESを選択します。

   Oracle SQL Developerウィンドウの右側フレームの「列」ペインに、この表にあるすべての列のリストが表示されます。各列の右側に、名前やデータ型などのプロパティがあります。(列のすべてのプロパティを参照するには、水平スクロール・バーを右に動かします。)

3. 右側のフレームで、「データ」タブをクリックします。

   「データ」ペインが表示され、この表にあるすべてのレコードの番号付きリストが表示されます。(さらにレコードを参照するには、垂直スクロール・バーを下へ移動します。さらにレコードの列を参照するには、水平スクロール・バーを右へ移動します。)

4. 右側のフレームで、「制約」タブをクリックします。

   「制約」ペインが表示され、この表に対するすべての制約のリストが表示されます。各制約の右側には、名前、型、検索条件などのプロパティがあります。(制約のすべてのプロパティを参照するには、水平スクロール・バーを右へ動かします。)

5. 適切なタブをクリックして、他のプロパティを参照します。

   EMPLOYEES表を作成するためのSQL文を確認するには、「SQL」タブをクリックします。SQL文は、EMPLOYEESという名前のペインに表示されます。このペインを閉じるには、名前EMPLOYEESの右のxをクリックします。

関連項目:

SQL Developerユーザー・インタフェースの詳細は、『*Oracle SQL Developerユーザーズ・ガイド*』を参照してください

親トピック: SQL DeveloperによるOracle Databaseの参照

2.6 表データの選択

> 注意:
>
> この項のチュートリアルおよび例を実行するには、ユーザーHR として SQL Developer から Oracle Database に接続する必要があります。手順は、「ユーザーHR としての SQL Developer から Oracle Database への接続」を参照してください。

- 問合せについて
  問合せ、つまりSQLのSELECT文は、1つ以上の表またはビューのデータを選択します。
- SQL Developerにおける問合せの実行
  この項では、ワークシートを使用してSQL Developerで問合せを実行する方法を説明します。
- チュートリアル: 表のすべての列の選択
  このチュートリアルは、EMPLOYEES表のすべての列を選択する方法を示します。
- チュートリアル: 表の特定の列の選択
  このチュートリアルは、EMPLOYEES表のFIRST_NAME列、LAST_NAME列およびDEPARTMENT_ID列のみを選択する方法を示します
- 新規のヘッダーの下での選択された列の表示
  問合せ結果を表示する際のデフォルトの列ヘッダーは、列の名前です。新規のヘッダーの下に列が表示されるようにするには、列名の直後に新規のヘッダー(**別名**)を指定します。別名により、問合せの期間に対する列名が変更されますが、データベース内の対応する名前は変更されません。
- 指定された条件を満たすデータの選択
  指定した条件に一致するデータのみを選択するには、SELECT文にWHERE句を含めます。
- 選択したデータのソート
  問合せの結果を表示する場合に、ORDER BY句で順序を指定しないと、レコードは任意の順序になります。
- 複数の表からのデータの選択
  複数の表からデータを選択するには、**結合**と呼ばれる問合せを使用します。結合の表は少なくとも1つの列名を共有している必要があります。
- 問合せにおける演算子およびファンクションの使用
  問合せのselect_listにはSQL式を含めることができ、SQL式にはSQL演算子とSQLファンクションを含めることができます。これらの演算子およびファンクションは、オペランドおよび引数として表データを持つことが可能です。SQL式を評価して、得られた値が問合せの結果に表示されます。

親トピック: Oracle Databaseへの接続および検索

2.6.1 問合せについて

問合せ、つまりSQLのSELECT文は、1つ以上の表またはビューからデータを選択します。

最も単純な形式の問合せは、次のような構文を持ちます。

```
SELECT select_list FROM source_list
```

select_listは、データが選択される列を指定し、source_listは、それらの列を含む表またはビューを指定します。

別のSQL文内にネストされた問合せは、**副問合せ**と呼ばれます。

SQL*Plus環境では、SQL>プロンプトで問合せ(またはその他の任意のSQL文)を入力できます。

SQL Developer環境では、ワークシートで問合せ(またはその他の任意のSQL文)を入力できます。

> **注意:**
>
> 問合せ結果が表示されるとき、ORDER BY 句で順序を指定しないかぎり、レコードを任意の順序で表示できます。詳細は、「選択されたデータのソート」を参照してください。

関連項目:

- 問合せおよび副問合せの詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。
- 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。
- SQL*Plusコマンド・ライン・インタフェースの詳細は、『*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でのワークシートの使用の詳細は、『*Oracle SQL Developerユーザーズ・ガイド*』を参照してください。

親トピック: 表データの選択

2.6.2 SQL Developerにおける問合せの実行

この項では、ワークシートを使用してSQL Developerで問合せを実行する方法を説明します。

> **注意:**
>
> ワークシートは問合せに限定されておらず、任意の SQL 文の実行に使用できます。

SQL Developerにおいて問合せを実行するには、次の手順を実行します。

1. SQL Developerの右側のフレームに、hr_connペインが表示されます。
    a. 「ワークシート」サブペインが表示されない場合は、「ワークシート」タブをクリックします。
    b. 手順4に進みます。
2. 「SQLワークシート」アイコンをクリックします。
3. 「接続の選択」ウィンドウが開かれた場合は、次のようにします。
    a. 「接続」 フィールドの値がhr_connではない場合は、メニューからその値を選択します。
    b. 「OK」をクリックします。

    ペインに、hr_connというラベルのタブと、「ワークシート」および「クエリー・ビルダー」という2つのサブペインが表示されます。ワークシートにはSQL文を入力できます。

4. ワークシートに問合せ(SELECT文)を入力します。
5. 「文の実行」アイコンをクリックします。

   問合せが実行されます。ワークシートの下に「問合せ結果」ペインが表示され、問合せの結果が表示されます。
6. 「hr_conn」タブの下で、「クリア」アイコンをクリックします。

   問合せが消去され、ワークシートに別のSQL文を入力できます。別のSQL文を実行すると、その結果が前に実行したSQL文の結果のかわりに「問合せ結果」ペインに表示されます。

関連項目:

SQL Developerでのワークシートの使用の詳細は、『*Oracle SQL Developerユーザーズ・ガイド*』を参照してください。

親トピック: 表データの選択

2.6.3 チュートリアル: 表のすべての列の選択

このチュートリアルは、EMPLOYEES表の列を選択する方法を示します。

EMPLOYEES表のすべての列を選択するには、次の手順を実行します。

1. hr_connというタブが含まれるペインがあれば、それを選択します。そうでない場合は、「SQL Developerにおける問合せの実行」にあるように、「SQLワークシート」アイコンをクリックします。
2. ワークシートに、次の問合せを入力します。
   ```
   SELECT * FROM EMPLOYEES;
   ```
3. 「文の実行」アイコンをクリックします。

   問合せが実行されます。ワークシートの下に「問合せ結果」ペインが表示され、EMPLOYEES表のすべての列が表示されます。

> **注意:**
>
> パスワードやクレジット・カード情報などの機密データを保存する列が含まれる表で SELECT *を使用する場合は十分に注意してください。

関連項目:

SQL Developerで表データを表示する他の方法の詳細は、「チュートリアル: SQL DeveloperによるEMPLOYEES表のプロパティとデータの表示」を参照してください。

親トピック: 表データの選択

2.6.4 チュートリアル: 表の特定の列の選択

このチュートリアルでは、EMPLOYEES表のFIRST_NAME列、LAST_NAME列およびDEPARTMENT_ID列のみを選択

する方法を示します。

FIRST_NAME、LAST_NAMEおよびDEPARTMENT_IDのみを選択するには、次の手順を実行します。

1. hr_connというタブが含まれるペインがあれば、それを選択します。そうでない場合は、「SQL Developerにおける問合せの実行」にあるように、「SQLワークシート」アイコンをクリックします。
2. 「ワークシート」ペインに問合せが含まれている場合、「クリア」アイコンをクリックして問合せを消去します。
3. ワークシートに、次の問合せを入力します。
   ```
   SELECT FIRST_NAME, LAST_NAME, DEPARTMENT_ID FROM EMPLOYEES;
   ```
4. 「文の実行」アイコンをクリックします。

   問合せが実行されます。ワークシートの下に「問合せ結果」ペインが表示され、次のような問合せの結果が表示されます。

```
FIRST_NAME           LAST_NAME                 DEPARTMENT_ID
-------------------- ------------------------- -------------
Donald               OConnell                             50
Douglas              Grant                                50
Jennifer             Whalen                               10
Michael              Hartstein                            20
Pat                  Fay                                  20
Susan                Mavris                               40
Hermann              Baer                                 70
Shelley              Higgins                             110
William              Gietz                               110
Steven               King                                 90
Neena                Kochhar                              90

FIRST_NAME           LAST_NAME                 DEPARTMENT_ID
-------------------- ------------------------- -------------
Lex                  De Haan                              90
...
Kevin                Feeney                               50

107 rows selected.
```

親トピック: 表データの選択

2.6.5 新規のヘッダーの下での選択された列の表示

問合せ結果を表示する際のデフォルトの列ヘッダーは、列の名前です。新規のヘッダーの下に列が表示されるようにするには、列名の直後に新規のヘッダー(**別名**)を指定します。別名により、問合せの期間に対する列名が変更されますが、データベース内の対応する名前は変更されません。

例2-5の問合せは、「チュートリアル: 表の特定の列の選択」の問合せと同じ列を選択しますが、列の別名も指定します。別名が二重引用符で囲まれていないので、大文字で表示されます。

列の別名を二重引用符で囲むと、例2-6のように、大/小文字の区別が保持され、別名に空白を使用することも可能です。

関連項目:

列の別名(*c_alias*)を含む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。

例2-5 新規のヘッダーの下での選択された列の表示

```
SELECT FIRST_NAME First, LAST_NAME last, DEPARTMENT_ID DepT
```

```
FROM EMPLOYEES;
```

結果は次のようになります。

```
FIRST                LAST                       DEPT
-------------------- ------------------------- ----------
Donald               OConnell                     50
Douglas              Grant                        50
Jennifer             Whalen                       10
Michael              Hartstein                    20
Pat                  Fay                          20
Susan                Mavris                       40
Hermann              Baer                         70
Shelley              Higgins                     110
William              Gietz                       110
Steven               King                         90
Neena                Kochhar                      90

FIRST                LAST                       DEPT
-------------------- ------------------------- ----------
Lex                  De Haan                      90
...
Kevin                Feeney                       50

107 rows selected.
```

例2-6 列の別名における大/小文字の区別の保持および空白の使用

```
SELECT FIRST_NAME "Given Name", LAST_NAME "Family Name"
FROM EMPLOYEES;
```

結果は次のようになります。

```
Given Name           Family Name
-------------------- -------------------------
Donald               OConnell
Douglas              Grant
Jennifer             Whalen
Michael              Hartstein
Pat                  Fay
Susan                Mavris
Hermann              Baer
Shelley              Higgins
William              Gietz
Steven               King
Neena                Kochhar

Given Name           Family Name
-------------------- -------------------------
Lex                  De Haan
...
Kevin                Feeney

107 rows selected.
```

親トピック: 表データの選択

2.6.6 指定された条件を満たすデータの選択

指定された条件に一致するデータのみを選択するには、SELECT文にWHERE句を含めます。

WHERE句には、任意のSQL条件を使用できます(SQL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)。

例2-7の問合せは、部門90に所属する従業員のデータのみを選択します。

部門100、110および120の従業員のデータのみを選択する場合は、このWHERE句を使用します。

```
WHERE DEPARTMENT_ID IN (100, 110, 120);
```

例2-8の問合せは、姓が「Ma」で始まる従業員のデータのみを選択します。

姓にmaが含まれる従業員のデータのみを選択する場合は、このWHERE句を使用します。

```
WHERE LAST_NAME LIKE '%ma%';
```

例2-9の問合せは、給与が少なくとも11000であり、かつ歩合率がNULLではないという2つの条件をテストします。

関連項目:

- WHERE句を含む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。
- SQL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。

例2-7 1つの部門からのデータの選択

```
SELECT FIRST_NAME, LAST_NAME, DEPARTMENT_ID
FROM EMPLOYEES
WHERE DEPARTMENT_ID = 90;
```

結果は次のようになります。

```
FIRST_NAME           LAST_NAME                 DEPARTMENT_ID
-------------------- ------------------------- -------------
Steven               King                                 90
Neena                Kochhar                              90
Lex                  De Haan                              90

3 rows selected.
```

例2-8 同じ部分文字列で始まる姓を持つデータの選択

```
SELECT FIRST_NAME, LAST_NAME
FROM EMPLOYEES
WHERE LAST_NAME LIKE 'Ma%';
```

結果は次のようになります。

```
FIRST_NAME           LAST_NAME
-------------------- -------------------------
Jason                Mallin
Steven               Markle
James                Marlow
Mattea               Marvins
Randall              Matos
Susan                Mavris
```

```
6 rows selected.
```

例2-9 2つの条件を満たすデータの選択

```
SELECT FIRST_NAME, LAST_NAME, SALARY, COMMISSION_PCT "%"
FROM EMPLOYEES
WHERE (SALARY >= 11000) AND (COMMISSION_PCT IS NOT NULL);
```

結果は次のようになります。

```
FIRST_NAME           LAST_NAME                 SALARY          %
-------------------- ------------------------- ---------- ----------
John                 Russell                    14000         .4
Karen                Partners                   13500         .3
Alberto              Errazuriz                  12000         .3
Gerald               Cambrault                  11000         .3
Lisa                 Ozer                       11500        .25
Ellen                Abel                       11000         .3

6 rows selected.
```

親トピック: 表データの選択

2.6.7 選択されたデータのソート

問合せの結果を表示する場合に、ORDER BY句で順序を指定しないと、レコードは任意の順序になります。

例2-10の問合せの結果は、LAST_NAMEを基準に昇順(デフォルト)でソートされます。

また、SQL Developerでは、ORDER BY句を省略し、列の名前をダブルクリックしてソートすることもできます。

例2-11に示すように、ソート基準を選択リストに含める必要はありません。

関連項目:

ORDER BY句を含む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。

例2-10 LAST_NAMEによる選択されたデータのソート

```
SELECT FIRST_NAME, LAST_NAME, HIRE_DATE
FROM EMPLOYEES
ORDER BY LAST_NAME;
```

結果:

```
FIRST_NAME           LAST_NAME                 HIRE_DATE
-------------------- ------------------------- ---------
Ellen                Abel                      11-MAY-04
Sundar               Ande                      24-MAR-08
Mozhe                Atkinson                  30-OCT-05
David                Austin                    25-JUN-05
Hermann              Baer                      07-JUN-02
Shelli               Baida                     24-DEC-05
Amit                 Banda                     21-APR-08
Elizabeth            Bates                     24-MAR-07
...
FIRST_NAME           LAST_NAME                 HIRE_DATE
```

```
-------------------- ------------------------- ---------
Jose Manuel          Urman                     07-MAR-06
Peter                Vargas                    09-JUL-06
Clara                Vishney                   11-NOV-05
Shanta               Vollman                   10-OCT-05
Alana                Walsh                     24-APR-06
Matthew              Weiss                     18-JUL-04
Jennifer             Whalen                    17-SEP-03
Eleni                Zlotkey                   29-JAN-08

107 rows selected
```

例2-11 選択対象ではない列による選択されたデータのソート

```
SELECT FIRST_NAME, HIRE_DATE
FROM EMPLOYEES
ORDER BY LAST_NAME;
```

結果:

```
FIRST_NAME           HIRE_DATE
-------------------- ---------
Ellen                11-MAY-04
Sundar               24-MAR-08
Mozhe                30-OCT-05
David                25-JUN-05
Hermann              07-JUN-02
Shelli               24-DEC-05
Amit                 21-APR-08
Elizabeth            24-MAR-07
...
FIRST_NAME           HIRE_DATE
-------------------- ---------
Jose Manuel          07-MAR-06
Peter                09-JUL-06
Clara                11-NOV-05
Shanta               10-OCT-05
Alana                24-APR-06
Matthew              18-JUL-04
Jennifer             17-SEP-03
Eleni                29-JAN-08

107 rows selected.
```

親トピック: 表データの選択

2.6.8 複数の表からのデータの選択

複数の表からデータを選択するには、**結合**と呼ばれる問合せを使用します。結合の表は少なくとも1つの列名を共有している必要があります。

すべての従業員のFIRST_NAME、LAST_NAMEおよびDEPARTMENT_NAMEを選択する場合を考えます。FIRST_NAMEおよびLAST_NAMEはEMPLOYEES表にあり、DEPARTMENT_NAMEはDEPARTMENTS表にあります。どちらの表にもDEPARTMENT_IDがあります。この場合、例2-12の問合せを使用できます。

表名の修飾子は、結合の一方の表にしかない列名ではオプションですが、両方の表にある列名では必須です。次の問合せは、例2-12の問合せと同等です。

```
SELECT FIRST_NAME "First",
```

```
LAST_NAME "Last",
DEPARTMENT_NAME "Dept. Name"
FROM EMPLOYEES, DEPARTMENTS
WHERE EMPLOYEES.DEPARTMENT_ID = DEPARTMENTS.DEPARTMENT_ID
ORDER BY DEPARTMENT_NAME, LAST_NAME;
```

修飾子付きの列名を使用する問合せをより読みやすくするには、次の例に示すように表の別名を使用します。

```
SELECT FIRST_NAME "First",
LAST_NAME "Last",
DEPARTMENT_NAME "Dept. Name"
FROM EMPLOYEES e, DEPARTMENTS d
WHERE e.DEPARTMENT_ID = d.DEPARTMENT_ID
ORDER BY d.DEPARTMENT_NAME, e.LAST_NAME;
```

FROM句で別名を作成するのではなく、次の例のように、問合せの中で先に作成することも可能です。

```
SELECT e.FIRST_NAME "First",
e.LAST_NAME "Last",
d.DEPARTMENT_NAME "Dept. Name"
FROM EMPLOYEES e, DEPARTMENTS d
WHERE e.DEPARTMENT_ID = d.DEPARTMENT_ID
ORDER BY d.DEPARTMENT_NAME, e.LAST_NAME;
```

関連項目:

結合の詳細は、*『Oracle Database SQL言語リファレンス』*を参照してください。

例2-12 2つの表からのデータの選択(2つの表の結合)

```
SELECT EMPLOYEES.FIRST_NAME "First",
EMPLOYEES.LAST_NAME "Last",
DEPARTMENTS.DEPARTMENT_NAME "Dept. Name"
FROM EMPLOYEES, DEPARTMENTS
WHERE EMPLOYEES.DEPARTMENT_ID = DEPARTMENTS.DEPARTMENT_ID
ORDER BY DEPARTMENTS.DEPARTMENT_NAME, EMPLOYEES.LAST_NAME;
```

結果:

```
First               Last                      Dept. Name
------------------- ------------------------- ------------------------------
William             Gietz                     Accounting
Shelley             Higgins                   Accounting
Jennifer            Whalen                    Administration
Lex                 De Haan                   Executive
Steven              King                      Executive
Neena               Kochhar                   Executive
John                Chen                      Finance
...
Jose Manuel         Urman                     Finance
Susan               Mavris                    Human Resources
David               Austin                    IT
...
Valli               Pataballa                 IT
Pat                 Fay                       Marketing
Michael             Hartstein                 Marketing
Hermann             Baer                      Public Relations
```

```
Shelli                Baida                Purchasing
...
Sigal                 Tobias               Purchasing
Ellen                 Abel                 Sales
...
Eleni                 Zlotkey              Sales
Mozhe                 Atkinson             Shipping
...
Matthew               Weiss                Shipping

106 rows selected.
```

親トピック: 表データの選択

2.6.9 問合せにおける演算子およびファンクションの使用

問合せのselect_listには、SQL演算子およびSQLファンクションを含むSQL式が含まれます。これらの演算子およびファンクションは、オペランドおよび引数として表データを持つことが可能です。SQL式を評価して、得られた値が問合せの結果に表示されます。

- 問合せにおける算術演算子の使用
  基本的な算術演算子として+(加算)、-(減算)、*(乗算)および/(除算)を列値で操作します。
- 問合せにおける数値ファンクションの使用
  数値ファンクションは、数値入力を受け取り、数値を返します。各数値ファンクションは、評価される各行に対し、値を1つ返します。
- 問合せにおける連結演算子の使用
  連結演算子(||)は、2つの文字列の2つ目を1つ目の文字列に追加して1つの文字列にします。たとえば、'a'||'b'='ab'となります。この演算子を使用すると、2つの列または式の情報を問合せ結果の同じ列の中に結合できます。
- 問合せにおける文字ファンクションの使用
  文字ファンクションは、文字入力を受け取ります。多くは文字の値を返しますが、数値を返すものもあります。各文字ファンクションは、評価される各行に対し、値を1つ返します。
- 問合せにおける日付ファンクションの使用
  日時ファンクションは、DATE、タイムスタンプ、間隔の値を操作します。各日付ファンクションは、評価される各行に対し、値を1つ返します。
- 問合せにおける変換ファンクションの使用
  変換ファンクションは、あるデータ型を他のデータ型に変換します。
- 問合せにおける集計ファンクションの使用
  集計ファンクションは、行のグループを取得し、単一の結果行を返します。行のグループは表またはビュー全体です。
- 問合せにおけるNULL関連ファンクションの使用
  NULL関連ファンクションは、NULL値の処理を容易にします。
- 問合せにおけるCASE式の使用
  CASE式を使用すると、サブプログラムを呼び出すことなく、SQL文でIF ... THEN ... ELSE論理を使用できます。CASE式には、単純と検索の2種類あります。
- 問合せにおけるDECODEファンクションの使用
  DECODEファンクションは、式といくつかの検索値と比較します。式の値が検索値に一致すると、DECODEは、検索値に関連付けられている結果を返します。DECODEが一致を検出しない場合、デフォルト値(指定されている場合)またはNULL(デフォルト値が指定されていない場合)が返されます。

関連項目:

- SQL演算子の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。
- SQLファンクションの詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

親トピック: 表データの選択

2.6.9.1 問合せにおける算術演算子の使用

基本的な算術演算子として+(加算)、-(減算)、*(乗算)および/(除算)を列値で操作します。

例2-13の問合せは、部門90に所属する各従業員のLAST_NAME、SALARY(月給)および年間賃金をSALARYの降順で表示します。

例2-13 問合せにおける演算式の使用

```
SELECT LAST_NAME,
SALARY "Monthly Pay",
SALARY * 12 "Annual Pay"
FROM EMPLOYEES
WHERE DEPARTMENT_ID = 90
ORDER BY SALARY DESC;
```

結果:

```
LAST_NAME                 Monthly Pay Annual Pay
------------------------- ----------- ----------
King                            24000     288000
De Haan                         17000     204000
Kochhar                         17000     204000
```

親トピック: 問合せにおける演算子およびファンクションの使用

2.6.9.2 問合せにおける数値ファンクションの使用

数値ファンクションは入力として数値を受け取り、結果として数値を戻します。各数値ファンクションは、評価される各行に対し、値を1つ返します。

SQLでサポートされる数値ファンクションのリストと説明は、*Oracle Database SQL言語リファレンス*に記載されています。

例2-14の問合せは、数値ファンクションROUNDを使用して、部門100の各従業員の日給を1セント未満は四捨五入して表示します。

例2-15の問合せは、数値ファンクションTRUNCを使用して、部門100の各従業員の日給を1ドル未満は切り捨てて表示します。

関連項目:

SQL数値ファンクションの詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

例2-14 数値データの四捨五入

```
SELECT LAST_NAME,
ROUND (((SALARY * 12)/365), 2) "Daily Pay"
FROM EMPLOYEES
WHERE DEPARTMENT_ID = 100
ORDER BY LAST_NAME;
```

結果:

```
LAST_NAME                 Daily Pay
------------------------- ----------
Chen                          269.59
Faviet                        295.89
Greenberg                     394.52
Popp                          226.85
Sciarra                       253.15
Urman                         256.44

6 rows selected.
```

例2-15 数値データの切捨て

```
SELECT LAST_NAME,
TRUNC ((SALARY * 12)/365) "Daily Pay"
FROM EMPLOYEES
WHERE DEPARTMENT_ID = 100
ORDER BY LAST_NAME;
```

結果:

```
LAST_NAME                 Daily Pay
------------------------- ----------
Chen                             269
Faviet                           295
Greenberg                        394
Popp                             226
Sciarra                          253
Urman                            256

6 rows selected.
```

親トピック: 問合せにおける演算子およびファンクションの使用

2.6.9.3 問合せにおける連結演算子の使用

連結演算子(||)は、1番目の文字列に2番目の文字列を追加することによって、2つの文字列を1つに結合します。たとえば、'a'||'b'='ab'となります。この演算子を使用すると、2つの列または式の情報を問合せ結果の同じ列の中に結合できます。

例2-16の問合せでは、選択した各従業員の名、スペースおよび姓を連結します。

関連項目:

連結演算子の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

例2-16 文字データの連結

```
SELECT FIRST_NAME || ' ' || LAST_NAME "Name"
FROM EMPLOYEES
WHERE DEPARTMENT_ID = 100
ORDER BY LAST_NAME;
```

結果:

```
Name
-----------------------------------------
```

```
John Chen
Daniel Faviet
Nancy Greenberg
Luis Popp
Ismael Sciarra
Jose Manuel Urman

6 rows selected.
```

親トピック: 問合せにおける演算子およびファンクションの使用

2.6.9.4 問合せにおける文字ファンクションの使用

文字ファンクションは、文字の入力を受け取ります。多くは文字の値を返しますが、数値を返すものもあります。各文字ファンクションは、評価される各行に対し、値を1つ返します。

SQLでサポートされる文字ファンクションのリストと説明は、*Oracle Database SQL言語リファレンス*に記載されています。

UPPER、INITCAPおよびLOWERファンクションは文字引数をそれぞれ、すべて大文字、頭文字のみ大文字および、すべて小文字で表示します。

例2-17の問合せは、LAST_NAMEを大文字で、FIRST_NAMEは頭文字を大文字で他は小文字で、EMAILを小文字で表示します。

関連項目:

SQL文字ファンクションの詳細は、*『Oracle Database SQL言語リファレンス』*を参照してください。

例2-17 文字データの大/小文字の変更

```
SELECT UPPER(LAST_NAME) "Last",
INITCAP(FIRST_NAME) "First",
LOWER(EMAIL) "E-Mail"
FROM EMPLOYEES
WHERE DEPARTMENT_ID = 100
ORDER BY EMAIL;
```

結果:

```
Last                      First               E-Mail
------------------------- ------------------- -------------------------
FAVIET                    Daniel              dfaviet
SCIARRA                   Ismael              isciarra
CHEN                      John                jchen
URMAN                     Jose Manuel         jmurman
POPP                      Luis                lpopp
GREENBERG                 Nancy               ngreenbe

6 rows selected.
```

親トピック: 問合せにおける演算子およびファンクションの使用

2.6.9.5 問合せにおける日付ファンクションの使用

日時ファンクションは、DATE、タイムスタンプ、間隔の値を操作します。各日付ファンクションは、評価される各行に対し、値を1つ返します。

SQLでサポートされる日付ファンクションのリストと説明は、*Oracle Database SQL言語リファレンス*に記載されています。

各DATEおよびタイムスタンプの値として、Oracle Databaseには、次の情報が格納されています。

- Year
- Month
- Date
- Hour
- Minute
- Second

  タイムスタンプの値として、Oracle Databaseには、指定した精度の秒の小数部分も格納されます。また、タイムゾーンを格納するには、データ型TIMESTAMP WITH TIME ZONEまたはTIMESTAMP WITH LOCAL TIME ZONEを使用します。

DATE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。

TIMESTAMP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。

その他のタイムスタンプ・データ型および間隔データ型の詳細は、*『Oracle Database SQL言語リファレンス』*を参照してください。

例2-18の問合せは、EXTRACTおよびSYSDATEファンクションを使用して、部門100の各従業員が何年間雇用されているかを表示します。SYSDATEファンクションは、システム・クロックの現在日時をDATE値として返します。SYSDATEファンクションの詳細は、*『Oracle Database SQL言語リファレンス』*を参照してください。EXTRACTファンクションの詳細は、*『Oracle Database SQL言語リファレンス』*を参照してください。

例2-19の問合せは、SYSTIMESTAMPファンクションを使用して、現在のシステム日付と時間を表示します。SYSTIMESTAMPファンクションは、TIMESTAMP値を返します。SYSTIMESTAMPファンクションの詳細は、*『Oracle Database SQL言語リファレンス』*を参照してください。

問合せのFROM句内のDUAL表は、Oracle Databaseがデータ・ディクショナリとともに自動的に作成する1行の表です。SELECT文を使用して定数式を計算する場合は、DUALから選択します。 DUALには行が1つしかないため、定数が返されるのは1回のみです。DUALからの選択の詳細は、*『Oracle Database SQL言語リファレンス』*を参照してください。

関連項目:

SQL日付ファンクションの詳細は、*『Oracle Database SQL言語リファレンス』*を参照してください。

例2-18 日付間の年数の表示

```
SELECT LAST_NAME,
(EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM HIRE_DATE)) "Years Employed"
FROM EMPLOYEES
WHERE DEPARTMENT_ID = 100
ORDER BY "Years Employed";
```

結果:

```
LAST_NAME                 Years Employed
------------------------- --------------
Popp                                   5
```

```
Urman                                      6
Chen                                       7
Sciarra                                    7
Greenberg                                 10
Faviet                                    10

6 rows selected.
```

例2-19 システム日付および時間の表示

```
SELECT EXTRACT(HOUR FROM SYSTIMESTAMP) || ':' ||
EXTRACT(MINUTE FROM SYSTIMESTAMP) || ':' ||
ROUND(EXTRACT(SECOND FROM SYSTIMESTAMP), 0) || ', ' ||
EXTRACT(MONTH FROM SYSTIMESTAMP) || '/' ||
EXTRACT(DAY FROM SYSTIMESTAMP) || '/' ||
EXTRACT(YEAR FROM SYSTIMESTAMP) "System Time and Date"
FROM DUAL;
```

結果は、現在のSYSTIMESTAMPの値に依存しますが、次の形式で表示されます。

```
System Time and Date
---------------------------------------------------------------
18:17:53, 12/27/2012
```

親トピック: 問合せにおける演算子およびファンクションの使用

2.6.9.6 問合せにおける変換ファンクションの使用

変換ファンクションは、あるデータ型を別のデータ型に変換します。

SQLでサポートされる変換ファンクションのリストと説明は、*Oracle Database SQL言語リファレンス*に記載されています。

例2-20の問合せは、TO_CHARファンクションを使用して、HIRE_DATE値(DATE型)をFMMonth DD YYYY 書式の文字値に変換します。 FMは、月名から先頭と末尾の空白を削除します。FMMonth DD YYYYは、**日付時間書式モデル**の一例です。日付時間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。

例2-21の問合せは、TO_NUMBERファンクションを使用して、POSTAL_CODE値(VARCHAR2型)を、計算で使用するNUMBER型の値に変換します。

関連項目:

- SQL変換ファンクションの詳細は、*『Oracle Database SQL言語リファレンス』*を参照してください。
- 「NLS_DATE_FORMATパラメータについて」

例2-20 日付から文字列への書式テンプレートを使用した変換

```
SELECT LAST_NAME,
HIRE_DATE,
TO_CHAR(HIRE_DATE, 'FMMonth DD YYYY') "Date Started"
FROM EMPLOYEES
WHERE DEPARTMENT_ID = 100
ORDER BY LAST_NAME;
```

結果:

```
LAST_NAME                 HIRE_DATE Date Started
------------------------- --------- -----------------
```

```
Chen                      28-SEP-05 September 28 2005
Faviet                    16-AUG-02 August 16 2002
Greenberg                 17-AUG-02 August 17 2002
Popp                      07-DEC-07 December 7 2007
Sciarra                   30-SEP-05 September 30 2005
Urman                     07-MAR-06 March 7 2006

6 rows selected.
```

例2-21 文字列から数値への変換

```
SELECT CITY,
POSTAL_CODE "Old Code",
TO_NUMBER(POSTAL_CODE) + 1 "New Code"
FROM LOCATIONS
WHERE COUNTRY_ID = 'US'
ORDER BY POSTAL_CODE;
```

結果:

```
CITY                       Old Code       New Code
-------------------------- ------------ ----------
Southlake                  26192             26193
South Brunswick            50090             50091
Seattle                    98199             98200
South San Francisco        99236             99237

4 rows selected.
```

親トピック: 問合せにおける演算子およびファンクションの使用

2.6.9.7 問合せにおける集計ファンクションの使用

集計ファンクションは、行のグループを取得し、単一の結果行を返します。行のグループは表またはビュー全体です。

SQLでサポートされる集計ファンクションのリストと説明は、*Oracle Database SQL言語リファレンス*に記載されています。

集計ファンクションは、1つまたは複数の列ごとに問合せ結果をグループ化するGROUP BY句を各グループの結果ともに使用する場合に、特に有効です。

例2-22の問合せは、COUNTファンクションおよびGROUP BY句を使用して、各マネージャに対してレポートする人数を表示します。ワイルドカード文字*は、レコード全体を表します。

例2-22は、従業員の1人がマネージャにレポートしないことを示しています。次の問合せは、その従業員の名、姓および役職を選択します。

```
COLUMN FIRST_NAME FORMAT A10;
COLUMN LAST_NAME FORMAT A10;
COLUMN JOB_TITLE FORMAT A10;

SELECT e.FIRST_NAME,
e.LAST_NAME,
j.JOB_TITLE
FROM EMPLOYEES e, JOBS j
WHERE e.JOB_ID = j.JOB_ID
AND MANAGER_ID IS NULL;
```

結果:

```
FIRST_NAME LAST_NAME  JOB_TITLE
```

```
---------- ---------- ----------
Steven     King       President
```

問合せで、集計値が指定された条件を満たす行のみを返すには、問合せのHAVING句で集計ファンクションを使用します。

例2-23の問合せは、毎年給与に$1,000,000を超える額を費やしている部門の部門と給与額を表示します。

例2-24の問合せは、複数の集計ファンクションを使用して、各JOB_IDの給与の統計を表示します。

関連項目:

SQL集計ファンクションの詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

例2-22 各グループの行数のカウント

```
SELECT MANAGER_ID "Manager",
COUNT(*) "Number of Reports"
FROM EMPLOYEES
GROUP BY MANAGER_ID
ORDER BY MANAGER_ID;
```

結果:

```
   Manager Number of Reports
---------- -----------------
       100                14
       101                 5
       102                 1
       103                 4
       108                 5
       114                 5
       120                 8
       121                 8
       122                 8
       123                 8
       124                 8
       145                 6
       146                 6
       147                 6
       148                 6
       149                 6
       201                 1
       205                 1
                           1

19 rows selected.
```

例2-23 条件を満たす行への集計ファンクションの制限

```
SELECT DEPARTMENT_ID "Department",
SUM(SALARY*12) "All Salaries"
FROM EMPLOYEES
HAVING SUM(SALARY * 12) >= 1000000
GROUP BY DEPARTMENT_ID;
```

結果:

```
Department All Salaries
```

```
---------- ------------
        50      1876800
        80      3654000
```

例2-24 統計情報を得るための集計ファンクションの使用

```
SELECT JOB_ID,
COUNT(*) "#",
MIN(SALARY) "Minimum",
ROUND(AVG(SALARY), 0) "Average",
MEDIAN(SALARY) "Median",
MAX(SALARY) "Maximum",
ROUND(STDDEV(SALARY)) "Std Dev"
FROM EMPLOYEES
GROUP BY JOB_ID
ORDER BY JOB_ID;
```

結果:

```
JOB_ID              #    Minimum    Average     Median    Maximum    Std Dev
---------- ---------- ---------- ---------- ---------- ---------- ----------
AC_ACCOUNT          1       8300       8300       8300       8300          0
AC_MGR              1      12008      12008      12008      12008          0
AD_ASST             1       4400       4400       4400       4400          0
AD_PRES             1      24000      24000      24000      24000          0
AD_VP               2      17000      17000      17000      17000          0
FI_ACCOUNT          5       6900       7920       7800       9000        766
FI_MGR              1      12008      12008      12008      12008          0
HR_REP              1       6500       6500       6500       6500          0
IT_PROG             5       4200       5760       4800       9000       1926
MK_MAN              1      13000      13000      13000      13000          0
MK_REP              1       6000       6000       6000       6000          0
PR_REP              1      10000      10000      10000      10000          0
PU_CLERK            5       2500       2780       2800       3100        239
PU_MAN              1      11000      11000      11000      11000          0
SA_MAN              5      10500      12200      12000      14000       1525
SA_REP             30       6100       8350       8200      11500       1524
SH_CLERK           20       2500       3215       3100       4200        548
ST_CLERK           20       2100       2785       2700       3600        453
ST_MAN              5       5800       7280       7900       8200       1066

19 rows selected.
```

親トピック: 問合せにおける演算子およびファンクションの使用

2.6.9.8 問合せにおけるNULL関連ファンクションの使用

NULL関連ファンクションは、NULL値の処理を容易にします。

SQLがサポートするNULL関連ファンクションのリストと説明については、『*Oracle Database SQL言語リファレンス*』を参照してください。

例2-25の問合せは、姓がBで始まる従業員の姓と歩合を返します。従業員が歩合を受け取らない場合(つまり、COMMISSION_PCTがNULLの場合)、NVLファンクションは、NULLを"Not Applicable"で置換します。

例2-26の問合せは、NVL2ファンクションを使用して、姓がBで始まる従業員の姓、給与および収入を返します(COMMISSION_PCTがNULLでない場合、収入は給与と歩合の合計であり、COMMISSION_PCTがNULLの場合、収入は給与のみです)。

関連項目:

- NVLファンクションの詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。
- NVL2ファンクションの詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

例2-25 文字列によるNULL値の置換

```
SELECT LAST_NAME,
NVL(TO_CHAR(COMMISSION_PCT), 'Not Applicable') "COMMISSION"
FROM EMPLOYEES
WHERE LAST_NAME LIKE 'B%'
ORDER BY LAST_NAME;
```

結果:

```
LAST_NAME                 COMMISSION
------------------------- ----------------------------------------
Baer                      Not Applicable
Baida                     Not Applicable
Banda                     .1
Bates                     .15
Bell                      Not Applicable
Bernstein                 .25
Bissot                    Not Applicable
Bloom                     .2
Bull                      Not Applicable

9 rows selected.
```

例2-26 NULL値と非NULL値に対する異なる式の指定

```
SELECT LAST_NAME, SALARY,
NVL2(COMMISSION_PCT, SALARY + (SALARY * COMMISSION_PCT), SALARY) INCOME
FROM EMPLOYEES WHERE LAST_NAME LIKE 'B%'
ORDER BY LAST_NAME;
```

結果:

```
LAST_NAME                   SALARY     INCOME
------------------------- ---------- ----------
Baer                           10000      10000
Baida                           2900       2900
Banda                           6200       6820
Bates                           7300       8395
Bell                            4000       4000
Bernstein                       9500      11875
Bissot                          3300       3300
Bloom                          10000      12000
Bull                            4100       4100

9 rows selected.
```

親トピック: 問合せにおける演算子およびファンクションの使用

2.6.9.9 問合せにおけるCASE式の使用

CASE式を使用すると、サブプログラムを呼び出すことなく、SQL文でIF ... THEN ... ELSE論理を使用できます。CASE式

には、単純と検索の2種類あります。

例2-27の問合せは、単純CASE式を使用し、各国コードの国名を表示します。

例2-28の問合せは、検索CASE式を使用して、勤続年数に基づいて推奨される給与の値上げ(15%、10%、5%または0%)を表示します。

関連項目:

- CASE式の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。
- CASE式の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください。
- 「問合せにおけるDECODEファンクションの使用」
- 「CASE文の使用」

例2-27 問合せでの単純CASE式の使用

```
SELECT UNIQUE COUNTRY_ID ID,
       CASE COUNTRY_ID
         WHEN 'AU' THEN 'Australia'
         WHEN 'BR' THEN 'Brazil'
         WHEN 'CA' THEN 'Canada'
         WHEN 'CH' THEN 'Switzerland'
         WHEN 'CN' THEN 'China'
         WHEN 'DE' THEN 'Germany'
         WHEN 'IN' THEN 'India'
         WHEN 'IT' THEN 'Italy'
         WHEN 'JP' THEN 'Japan'
         WHEN 'MX' THEN 'Mexico'
         WHEN 'NL' THEN 'Netherlands'
         WHEN 'SG' THEN 'Singapore'
         WHEN 'UK' THEN 'United Kingdom'
         WHEN 'US' THEN 'United States'
       ELSE 'Unknown'
       END COUNTRY
FROM LOCATIONS
ORDER BY COUNTRY_ID;
```

結果:

```
ID COUNTRY
-- --------------
AU Australia
BR Brazil
CA Canada
CH Switzerland
CN China
DE Germany
IN India
IT Italy
JP Japan
MX Mexico
NL Netherlands
SG Singapore
UK United Kingdom
US United States
```

```
14 rows selected.
```

例2-28 問合せでの検索CASE式の使用

```
SELECT LAST_NAME "Name",
HIRE_DATE "Started",
SALARY "Salary",
CASE
  WHEN HIRE_DATE < TO_DATE('01-Jan-03', 'dd-mon-yy')
    THEN TRUNC(SALARY*1.15, 0)
  WHEN HIRE_DATE >= TO_DATE('01-Jan-03', 'dd-mon-yy') AND
       HIRE_DATE < TO_DATE('01-Jan-06', 'dd-mon-yy')
    THEN TRUNC(SALARY*1.10, 0)
  WHEN HIRE_DATE >= TO_DATE('01-Jan-06', 'dd-mon-yy') AND
       HIRE_DATE < TO_DATE('01-Jan-07', 'dd-mon-yy')
    THEN TRUNC(SALARY*1.05, 0)
  ELSE SALARY
END "Proposed Salary"
FROM EMPLOYEES
WHERE DEPARTMENT_ID = 100
ORDER BY HIRE_DATE;
```

結果:

```
Name                      Started       Salary Proposed Salary
------------------------- --------- ---------- ---------------
Faviet                    16-AUG-02       9000           10350
Greenberg                 17-AUG-02      12008           13809
Chen                      28-SEP-05       8200            9020
Sciarra                   30-SEP-05       7700            8470
Urman                     07-MAR-06       7800            8190
Popp                      07-DEC-07       6900            6900

6 rows selected.
```

親トピック: 問合せにおける演算子およびファンクションの使用

2.6.9.10 問合せにおけるDECODEファンクションの使用

DECODEファンクションは、式といくつかの検索値と比較します。式の値が検索値に一致すると、DECODEは、検索値に関連付けられている結果を返します。DECODEが一致を検出しない場合、デフォルト値(指定されている場合)またはNULL(デフォルト値が指定されていない場合)が返されます。

例2-29の問合せは、DECODEファンクションを使用して、3つの異なる役職に対して推奨される給与の値上げを表示します。式はJOB_ID、検索値は'PU_CLERK'、'SH_CLERK'、'ST_CLERK'、およびデフォルト値はSALARYです。

> 注意:
>
> DECODE ファンクションの引数は、任意の SQL 数値型または文字型です。式および各検索値は、比較の前に最初の検索値のデータ型に自動的に変換されます。戻り値は、最初の結果と同じデータ型に自動的に変換されます。最初の結果がデータ型 CHAR の場合、または最初の結果が NULL の場合、戻り値はデータ型 VARCHAR2 に変換されます。

関連項目:

- DECODEファンクションの詳細は、『Oracle Database SQL言語リファレンス』を参照してください。
- 「問合せにおけるCASE式の使用」

例2-29 問合せにおけるDECODEファンクションの使用

```
SELECT LAST_NAME, JOB_ID, SALARY,
DECODE(JOB_ID,
  'PU_CLERK', SALARY * 1.10,
  'SH_CLERK', SALARY * 1.15,
  'ST_CLERK', SALARY * 1.20,
  SALARY) "Proposed Salary"
FROM EMPLOYEES
WHERE JOB_ID LIKE '%_CLERK'
AND LAST_NAME < 'E'
ORDER BY LAST_NAME;
```

結果:

```
LAST_NAME                 JOB_ID         SALARY Proposed Salary
------------------------- ---------- ---------- ---------------
Atkinson                  ST_CLERK         2800            3360
Baida                     PU_CLERK         2900            3190
Bell                      SH_CLERK         4000            4600
Bissot                    ST_CLERK         3300            3960
Bull                      SH_CLERK         4100            4715
Cabrio                    SH_CLERK         3000            3450
Chung                     SH_CLERK         3800            4370
Colmenares                PU_CLERK         2500            2750
Davies                    ST_CLERK         3100            3720
Dellinger                 SH_CLERK         3400            3910
Dilly                     SH_CLERK         3600            4140

11 rows selected.
```

親トピック: 問合せにおける演算子およびファンクションの使用

3 DML文とトランザクションについて

データ操作言語(DML)文は、Oracle Database表のデータを追加、変更および削除します。**トランザクション**は、Oracle Databaseが1つの単位として処理する連続した1つ以上のSQL文です(これらの文は、すべて実行されるか、1つも実行されないかのいずれかです)。

- データ操作言語(DML)文について
  データ操作言語(DML)文は、既存の表のデータにアクセスし、操作します。
- トランザクション制御文について
  トランザクションは、Oracle Databaseが1つの単位として扱う、1つ以上の連続するSQL文です。すべての文が実行されるか、1つも実行されないかのいずれかになります。トランザクションは、複数の操作を1つの単位として実行する必要があるビジネス・プロセスをモデル化する場合に必要になります。
- トランザクションのコミット
  トランザクションをコミットすると、変更が永続的になり、セーブポイントが消去され、ロックが解除されます。
- トランザクションのロールバック
  トランザクションをロールバックすると、変更が取り消されます。現在のトランザクション全体をロールバックするか、指定したセーブポイントまでのみロールバックすることができます。
- トランザクションでのセーブポイントの設定
  SAVEPOINT文は、トランザクションに**セーブポイント**をマークします。このポイントからトランザクションをロールバックできます。セーブポイントはオプションで、1つのトランザクションで複数のセーブポイントを設定できます。

3.1 データ操作言語(DML)文について

データ操作言語(DML)文は、既存の表のデータにアクセスし、操作します。

SQL*Plus環境では、SQL>プロンプトに続いてDML文を入力できます。

SQL Developer環境では、ワークシートにDML文を入力できます。または、SQL Developerの「接続」フレームとツールを使用して、データにアクセスして操作することもできます。

SQL DeveloperでDML文の結果を確認するには、「接続」フレームで変更されたオブジェクトのスキーマ・オブジェクト・タイプを選択して、「リフレッシュ」アイコンをクリックします。

DML文を含むトランザクションをコミットするまでは、その効果は永続的ではありません。**トランザクション**は、Oracle Databaseが1つの単位として扱うSQL文の順序です(1つのDML文のこともあります)。トランザクションをコミットするまでは、ロールバックする(元に戻す)ことができます。トランザクションの詳細は、「トランザクション制御文について」を参照してください。

- INSERT文について
  INSERT文は、既存の表に行を挿入します。
- UPDATE文について
  UPDATE文は、既存の表の行のセットを更新(値を変更)します。
- DELETE文について
  DELETE文は、表から行を削除します。

関連項目:

DML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。

3.1.1 INSERT文について

INSERT文は、既存の表に行を挿入します。

推奨されている最も単純なINSERT文の構文は、次のとおりです。

```
INSERT INTO table_name (list_of_columns)
VALUES (list_of_values);
```

list_of_columnsの各列に対して、list_of_valuesの対応する位置に有効な値が必要です。そのため、表に列を挿入する前に、表にどのような列があり、どのような値が有効かを知っておく必要があります。SQL Developerを使用してこの情報を取得するには、「チュートリアル: SQL DeveloperによるEMPLOYEES表のプロパティとデータの表示」を参照してください。
SQL*Plusを使用してこうした情報を取得するには、DESCRIBE文を使用します。次に例を示します。

```
DESCRIBE EMPLOYEES;
```

結果:

```
Name                                      Null?    Type
----------------------------------------- -------- ------------

EMPLOYEE_ID                               NOT NULL NUMBER(6)
FIRST_NAME                                         VARCHAR2(20)
LAST_NAME                                 NOT NULL VARCHAR2(25)
EMAIL                                     NOT NULL VARCHAR2(25)
PHONE_NUMBER                                       VARCHAR2(20)
HIRE_DATE                                 NOT NULL DATE
JOB_ID                                    NOT NULL VARCHAR2(10)
SALARY                                             NUMBER(8,2)
COMMISSION_PCT                                     NUMBER(2,2)
MANAGER_ID                                         NUMBER(6)
DEPARTMENT_ID                                      NUMBER(4)
```

例3-1のINSERT文では、すべての列の値がわかっている従業員として、表EMPLOYEESに行を挿入します。

表に行を挿入するためにすべての列の値を公開する必要はありませんが、NOT NULL列の値はすべて公開します。NULLの可能性がある列の値がわからない場合は、その列をlist_of_columnsから省略することができます。この列の値はデフォルトでNULLとなります。

例3-2のINSERT文では、SALARYを除くすべての列の値がわかっている従業員として、表EMPLOYEESに行を挿入します。ここでは、SALARYをNULL値にすることができます。給与がわかった際に、UPDATE文で更新できます(例3-4を参照)。

例3-3のINSERT文では、LAST_NAMEが不明の従業員として、表EMPLOYEESに行を挿入しようとしています。

例3-1 すべての情報が使用できる場合でのINSERT文の使用

```
INSERT INTO EMPLOYEES (
  EMPLOYEE_ID,
  FIRST_NAME,
  LAST_NAME,
  EMAIL,
  PHONE_NUMBER,
  HIRE_DATE,
  JOB_ID,
  SALARY,
  COMMISSION_PCT,
  MANAGER_ID,
```

```
  DEPARTMENT_ID
)
VALUES (
  10,               -- EMPLOYEE_ID
  'George',         -- FIRST_NAME
  'Gordon',         -- LAST_NAME
  'GGORDON',        -- EMAIL
  '650.506.2222',   -- PHONE_NUMBER
  '01-JAN-07',      -- HIRE_DATE
  'SA_REP',         -- JOB_ID
  9000,             -- SALARY
  .1,               -- COMMISSION_PCT
  148,              -- MANAGER_ID
  80                -- DEPARTMENT_ID
);
```

結果:

```
1 row created.
```

例3-2 情報をすべては使用できない場合でのINSERT文の使用

```
INSERT INTO EMPLOYEES (
  EMPLOYEE_ID,
  FIRST_NAME,
  LAST_NAME,
  EMAIL,
  PHONE_NUMBER,
  HIRE_DATE,
  JOB_ID,           -- Omit SALARY; its value defaults to NULL.
  COMMISSION_PCT,
  MANAGER_ID,
  DEPARTMENT_ID
)
VALUES (
  20,               -- EMPLOYEE_ID
  'John',           -- FIRST_NAME
  'Keats',          -- LAST_NAME
  'JKEATS',         -- EMAIL
  '650.506.3333',   -- PHONE_NUMBER
  '01-JAN-07',      -- HIRE_DATE
  'SA_REP',         -- JOB_ID
  .1,               -- COMMISSION_PCT
  148,              -- MANAGER_ID
  80                -- DEPARTMENT_ID
);
```

結果:

```
1 row created.
```

例3-3 INSERT文の誤った使用

```
INSERT INTO EMPLOYEES (
  EMPLOYEE_ID,
  FIRST_NAME,       -- Omit LAST_NAME (error)
  EMAIL,
  PHONE_NUMBER,
  HIRE_DATE,
  JOB_ID,
```

```
  COMMISSION_PCT,
  MANAGER_ID,
  DEPARTMENT_ID
)
VALUES (
  20,               -- EMPLOYEE_ID
  'John',           -- FIRST_NAME
  'JOHN',           -- EMAIL
  '650.506.3333',   -- PHONE_NUMBER
  '01-JAN-07',      -- HIRE_DATE
  'SA_REP',         -- JOB_ID
  .1,               -- COMMISSION_PCT
  148,              -- MANAGER_ID
  80                -- DEPARTMENT_ID
);
```

結果:

```
ORA-01400: cannot insert NULL into ("HR"."EMPLOYEES"."LAST_NAME")
```

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。
- 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。
- 「チュートリアル: 行の挿入ツールによる表への行の追加」

親トピック: データ操作言語(DML)文について

3.1.2 UPDATE文について

UPDATE文は、既存の表の行のセットを更新(値を変更)します。

単純なUPDATE文の構文は、次のとおりです。

```
UPDATE table_name
SET column_name = value [, column_name = value]...
[ WHERE condition ];
```

各valueは対応するcolumn_nameに対して有効である必要があります。 WHERE句を使用する場合、その文ではconditionを満たす行の列値のみを更新します。

例3-4のUPDATE文では、例3-2で従業員の給与がわかる前にEMPLOYEES表に挿入された行のSALARY列の値を更新します。

例3-5のUPDATE文では、部門80の全従業員の歩合率を更新します。

例3-4 データを追加するUPDATE文の使用

```
UPDATE EMPLOYEES
SET SALARY = 8500
WHERE LAST_NAME = 'Keats';
```

結果:

```
1 row updated.
```

例3-5 複数の行を更新するUPDATE文の使用

```
UPDATE EMPLOYEES
SET COMMISSION_PCT = COMMISSION_PCT + 0.05
WHERE DEPARTMENT_ID = 80;
```

結果:

```
34 rows updated.
```

関連項目:

- UPDATE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。
- 「チュートリアル:「データ」ペインにある表のデータの変更」

親トピック: データ操作言語(DML)文について

3.1.3 DELETE文について

DELETE文は、表から行を削除します。

単純なDELETE文の構文は、次のとおりです。

```
DELETE FROM table_name [ WHERE condition ];
```

WHERE句を含む文では、conditionを満たす行のみが削除されます。 WHERE句を省略した場合、その文では表からすべての行が削除されますが、空の表は残ります。表を削除するには、DROP TABLE文を使用します。

例3-6のDELETE文は、例3-1および例3-2で挿入した行を削除します。

例3-6 DELETE文の使用

```
DELETE FROM EMPLOYEES
WHERE HIRE_DATE = TO_DATE('01-JAN-07', 'dd-mon-yy');
```

結果:

```
2 rows deleted.
```

関連項目:

- DELETE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。
- 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。
- 「チュートリアル: 選択した行の削除ツールを使用した表内の行の削除」

親トピック: データ操作言語(DML)文について

3.2 トランザクション制御文について

トランザクションは、Oracle Databaseが1つの単位として処理する連続した1つ以上のSQL文です(これらの文は、すべて実行されるか、1つも実行されないかのいずれかです)。トランザクションは、複数の操作を1つの単位として実行する必要があるビジネス・プロセスをモデル化する場合に必要になります。

たとえば、マネージャが退職する場合、JOB_HISTORY表に退社日を示す行が挿入され、このマネージャにレポートする全従業員について、EMPLOYEES表でMANAGER_IDを更新する必要があります。アプリケーションでこのプロセスをモデル化するには、INSERT文とUPDATE文を1つのトランザクションとしてグループ化する必要があります。

基本的なトランザクション制御文は、次のとおりです。

- SAVEPOINT: トランザクションに**セーブポイント**をマークします。このポイントからトランザクションをロールバックできます。セーブポイントはオプションで、1つのトランザクションで複数のセーブポイントを設定できます。
- COMMITは、現在のトランザクションを終了して、変更を永続的にし、セーブポイントを消去して、ロックを解除します。
- ROLLBACKは、現在のトランザクション全体か、または指定したセーブポイント以降に行われた変更のみをロールバックします(取り消す)。

SQL*Plus環境では、SQL>プロンプトに続いてトランザクション制御文を入力できます。

SQL Develope環境では、ワークシートにトランザクション制御文を入力できます。また、SQL Developerには、「変更のコミット」および「変更のロールバック」アイコンがあります。これらのアイコンの詳細は、「トランザクションのコミット」および"「トランザクションのロールバック」を参照してください。

> 注意:
>
> 明示的にトランザクションをコミットせずプログラムが異常終了した場合、データベースはコミットされていないトランザクションを自動的にロールバックします。
>
> アプリケーション・プログラム内でトランザクションをコミットまたはロールバックして、トランザクションを明示的に終了することをお薦めします。

関連項目:

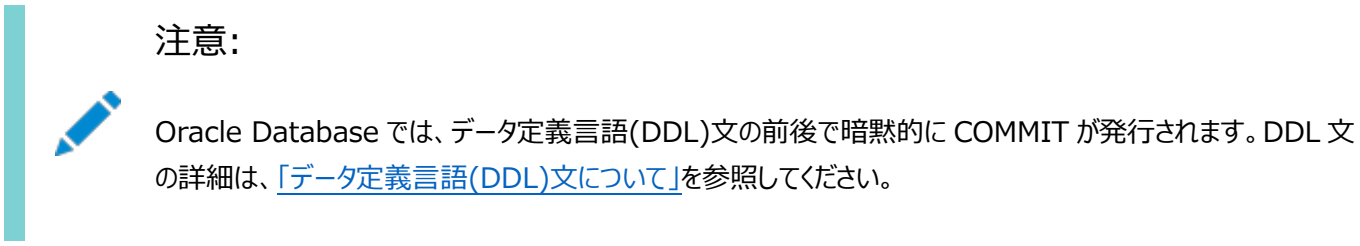
- トランザクション管理の詳細は、『*Oracle Database概要*』を参照してください。
- トランザクション制御文の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

親トピック: DML文とトランザクションについて

3.3 トランザクションのコミット

トランザクションをコミットすると、そのトランザクションの変更を確定して、セーブポイントを消去しロックを解除します。

トランザクションを明示的にコミットするには、COMMIT文を使用するか、(SQL Developer環境で)「変更のコミット」アイコンを使用します。



トランザクションをコミットする前:

- 変更は表示できますが、他のデータベース・インスタンスのユーザーには表示されません。
- 変更は完了しておらず、ROLLBACK文で取り消すことができます。

トランザクションをコミットした後:

- 自分で行った変更が他のユーザーから参照可能になり、トランザクションのコミット後に他のユーザーの文が実行されます。
- 変更は完了しており、ROLLBACK文で取り消すことはできません。

例3-7では、REGIONS表に1行追加し(単純なトランザクション)、結果を確認してから、トランザクションをコミットします。

例3-7 トランザクションのコミット

トランザクションの前:

```
SELECT * FROM REGIONS
ORDER BY REGION_ID;
```

結果:

```
 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
         3 Asia
         4 Middle East and Africa

4 rows selected.
```

トランザクション(表に行を追加):

```
INSERT INTO regions (region_id, region_name) VALUES (5, 'Africa');
```

結果:

```
1 row created.
```

行が追加されたことを確認:

```
SELECT * FROM REGIONS
ORDER BY REGION_ID;
```

結果:

```
 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
```

```
         3 Asia
         4 Middle East and Africa
         5 Africa

5 rows selected.
```

トランザクションのコミット:

```
COMMIT;
```

結果:

```
Commit complete.
```

関連項目:

COMMIT文の詳細は、*『Oracle Database SQL言語リファレンス』* を参照してください。

親トピック: DML文とトランザクションについて

3.4 トランザクションのロールバック

トランザクションのロールバックでは変更は取り消されます。現在のトランザクション全体をロールバックするか、指定したセーブポイントまでのみロールバックすることができます。

現在のトランザクションを指定したセーブポイントまでのみロールバックするには、ROLLBACK文をTO SAVEPOINT句とともに使用します。

現在のトランザクション全体をロールバックするには、ROLLBACK文をTO SAVEPOINT句なしで使用するか、(SQL Developer環境で)「変更のロールバック」アイコンを使用します。

現在のトランザクション全体のロールバック

- トランザクションを終了します
- すべての変更を戻します
- すべてのセーブポイントを消去します
- トランザクションのロックの解除

現在のトランザクションの指定したセーブポイントまでのみのロールバック:

- トランザクションを終了しません
- 指定したセーブポイントの後で行われた変更のみを戻します
- 指定したセーブポイントの後にあるセーブポイントのみを消去します(指定したセーブポイント自体は除く)
- 指定したセーブポイントの後で取得された、すべての表と行のロックを解除します

  指定したセーブポイントの後で、ロックされた行に対するアクセスを要求した他のトランザクションは、トランザクションがコミットまたはロールバックされるまで引き続き待機する必要があります。その行を要求していない他のトランザクションは、直ちにその行を要求してアクセスすることができます。

SQL Developerでロールバックの効果を確認するには、「リフレッシュ」アイコンをクリックする必要のある場合があります。

例3-7の結果、REGIONS表には「Middle East and Africa」という地域と「Africa」という地域があります。例3-8では、この問題を修正(単純なトランザクション)して変更を確認した後、トランザクションをロールバックし、ロールバックされたことを確認します。

例3-8 トランザクション全体のロールバック

トランザクションの前:

```
SELECT * FROM REGIONS
ORDER BY REGION_ID;
```

結果:

```
 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
         3 Asia
         4 Middle East and Africa
         5 Africa

5 rows selected.
```

トランザクション(表を変更):

```
UPDATE REGIONS
SET REGION_NAME = 'Middle East'
WHERE REGION_NAME = 'Middle East and Africa';
```

結果:

```
1 row updated.
```

変更のチェック:

```
SELECT * FROM REGIONS
ORDER BY REGION_ID;
```

結果:

```
 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
         3 Asia
         4 Middle East
         5 Africa

5 rows selected.
```

トランザクションのロールバック:

```
ROLLBACK;
```

結果:

```
Rollback complete.
```

ロールバックのチェック:

```
SELECT * FROM REGIONS
```

```
ORDER BY REGION_ID;
```

結果:

```
 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
         3 Asia
         4 Middle East and Africa
         5 Africa

5 rows selected.
```

関連項目:

ROLLBACK文の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

親トピック: DML文とトランザクションについて

3.5 トランザクションでのセーブポイントの設定

SAVEPOINT文は、トランザクションに**セーブポイント**をマークします。このポイントからトランザクションをロールバックできます。セーブポイントはオプションで、1つのトランザクションで複数のセーブポイントを設定できます。

例3-9では、複数のDML文と複数のセーブポイントがあるトランザクションを実行した後、あるセーブポイントへトランザクションをロールバックして、そのセーブポイントの後で行われた変更のみを元に戻します。

例3-9 セーブポイントへのトランザクションのロールバック

トランザクション前のREGIONS表のチェック:

```
SELECT * FROM REGIONS
ORDER BY REGION_ID;
```

結果:

```
 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
         3 Asia
         4 Middle East and Africa
         5 Africa

5 rows selected.
```

トランザクション前に地域4にある国の確認:

```
SELECT COUNTRY_NAME, COUNTRY_ID, REGION_ID
FROM COUNTRIES
WHERE REGION_ID = 4
ORDER BY COUNTRY_NAME;
```

結果:

```
COUNTRY_NAME                               CO  REGION_ID
---------------------------------------- -- ----------
Egypt                                      EG          4
Israel                                     IL          4
Kuwait                                     KW          4
Nigeria                                    NG          4
Zambia                                     ZM          4
Zimbabwe                                   ZW          4

6 rows selected.
```

トランザクション前に地域5にある国の確認:

```
SELECT COUNTRY_NAME, COUNTRY_ID, REGION_ID
FROM COUNTRIES
WHERE REGION_ID = 5
ORDER BY COUNTRY_NAME;
```

結果:

```
no rows selected
```

複数のセーブポイントがあるトランザクション:

```
UPDATE REGIONS
SET REGION_NAME = 'Middle East'
WHERE REGION_NAME = 'Middle East and Africa';

UPDATE COUNTRIES
  SET REGION_ID = 5
  WHERE COUNTRY_ID = 'ZM';
SAVEPOINT zambia;

UPDATE COUNTRIES
  SET REGION_ID = 5
  WHERE COUNTRY_ID = 'NG';
SAVEPOINT nigeria;

UPDATE COUNTRIES
  SET REGION_ID = 5
  WHERE COUNTRY_ID = 'ZW';
SAVEPOINT zimbabwe;

UPDATE COUNTRIES
  SET REGION_ID = 5
  WHERE COUNTRY_ID = 'EG';
SAVEPOINT egypt;
```

トランザクション後のREGIONS表の確認:

```
SELECT * FROM REGIONS
ORDER BY REGION_ID;
```

結果:

```
 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
         3 Asia
         4 Middle East
```

```
         5 Africa

5 rows selected.
```

トランザクション後に地域4にある国の確認:

```
SELECT COUNTRY_NAME, COUNTRY_ID, REGION_ID
FROM COUNTRIES
WHERE REGION_ID = 4
ORDER BY COUNTRY_NAME;
```

結果:

```
COUNTRY_NAME                             CO  REGION_ID
---------------------------------------- -- ----------
Israel                                   IL          4
Kuwait                                   KW          4

2 rows selected.
```

トランザクション後に地域5にある国の確認:

```
SELECT COUNTRY_NAME, COUNTRY_ID, REGION_ID
FROM COUNTRIES
WHERE REGION_ID = 5
ORDER BY COUNTRY_NAME;
```

結果:

```
COUNTRY_NAME                             CO  REGION_ID
---------------------------------------- -- ----------
Egypt                                    EG          5
Nigeria                                  NG          5
Zambia                                   ZM          5
Zimbabwe                                 ZW          5

4 rows selected.

ROLLBACK TO SAVEPOINT nigeria;
```

ロールバック後のREGIONS表の確認:

```
SELECT * FROM REGIONS
ORDER BY REGION_ID;
```

結果:

```
 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
         3 Asia
         4 Middle East
         5 Africa

5 rows selected.
```

ロールバック後に地域4にある国の確認:

```
SELECT COUNTRY_NAME, COUNTRY_ID, REGION_ID
FROM COUNTRIES
```

```
WHERE REGION_ID = 4
ORDER BY COUNTRY_NAME;
```

結果:

```
COUNTRY_NAME                             CO  REGION_ID
---------------------------------------- -- ----------
Egypt                                    EG          4
Israel                                   IL          4
Kuwait                                   KW          4
Zimbabwe                                 ZW          4

4 rows selected.
```

ロールバック後に地域5にある国の確認:

```
SELECT COUNTRY_NAME, COUNTRY_ID, REGION_ID
FROM COUNTRIES
WHERE REGION_ID = 5
ORDER BY COUNTRY_NAME;
```

結果:

```
COUNTRY_NAME                             CO  REGION_ID
---------------------------------------- -- ----------
Nigeria                                  NG          5
Zambia                                   ZM          5

2 rows selected.
```

関連項目:

SAVEPOINT文の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

親トピック: DML文とトランザクションについて

4 スキーマ・オブジェクトの作成および管理

スキーマ・オブジェクトを作成、変更および削除するには、データ定義言語(DDL)文を使用します。

- データ定義言語(DDL)について
  データ定義言語(DDL)文はスキーマ・オブジェクトを作成、変更、削除します。DDL文の前後に、Oracle Databaseは暗黙的なCOMMIT文を発行するため、DDL文はロールバックできません。
- 表の作成および管理
  表は、Oracle Databaseのデータ記憶域の基本単位です。表は、ユーザーがアクセス可能なすべてのデータを保持します。各表には、それぞれのデータのレコードを表す行が含まれています。行は、レコードのフィールドを表す列で構成されています。
- ビューの作成および管理
  ビューは、表として問合せ結果を表示します。表を使用できるほとんどの場所で、ビューを使用できます。複数の異なる表に格納された情報に何度もアクセスする必要がある場合に、ビューが役立ちます。
- 順序の作成および管理
  順序は一意の連続した値を生成できるスキーマ・オブジェクトであり、一意の主キーが必要な場合に非常に役立ちます。順序は、疑似列CURRVALおよびNEXTVALを通じて使用します。これらの疑似列は、それぞれ順序の現在と次の値を返します。
- シノニムの作成および管理
  シノニムとは別のスキーマ・オブジェクトの別名です。シノニムを使用する理由は、セキュリティ(たとえば、オブジェクトの所有者と位置を分からなくするため)と簡素化のためです。

4.1 データ定義言語(DDL)文について

データ定義言語(DDL)文はスキーマ・オブジェクトを作成、変更、削除します。DDL文の前後に、Oracle Databaseは暗黙的なCOMMIT文を発行するため、DDL文はロールバックできません。

> 注意:
>
> スキーマ・オブジェクトを作成する場合は、『*Oracle Database SQL 言語リファレンス*』に記載されたスキーマ・オブジェクトのネーミング・ルールに注意してください。

SQL*Plus環境では、SQL>プロンプトに続いてDDL文を入力できます。

SQL Developer環境では、ワークシートにDDL文を入力できます。または、SQL Developerのツールを使用して、オブジェクトを作成、変更および削除することもできます。

スキーマ・オブジェクトを作成するDDL文に、オプションのOR REPLACE句が入る場合があります。これらの句を使用すると、DDL文の既存のスキーマ・オブジェクトを同じ名前とタイプの他のスキーマ・オブジェクトに置き換えられます。SQL Developerがこの文に対してコードを生成する場合、常にOR REPLACE句が含まれます。

SQL DeveloperでDDL文の結果を確認するには、「接続」フレームで新しく作成されたオブジェクトのスキーマ・オブジェクト・タイプを選択して、「リフレッシュ」アイコンをクリックします。

関連項目:

- DDL文の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。
- 「トランザクションのコミット」

親トピック: スキーマ・オブジェクトの作成および管理

4.2 表の作成および管理

表は、Oracle Databaseのデータ記憶領域の基本単位です。表は、ユーザーがアクセス可能なすべてのデータを保持します。各表には、それぞれのデータのレコードを表す行が含まれています。行は、レコードのフィールドを表す列で構成されています。

> **注意:**
>
> このマニュアルのチュートリアルを行うには、ユーザーHR として、SQL Developer から Oracle Database に接続している必要があります。

- SQLデータ型について
  表を作成するときは、各列にSQLデータ型を指定します。列がどのような値を持つことができるかは、列のデータ型によって決まります。
- 表の作成
  表を作成するには、SQL Developerの表の作成ツールまたはDDL文のCREATE TABLEを使用します。
- 表のデータ整合性の保証
  ユーザーのアプリケーションが形成するビジネス・ルールを表内のデータが満たすことを保証するために、制約、アプリケーション・ロジック、またはその両方を使用できます。
- チュートリアル: 行の挿入ツールによる表への行の追加
  このチュートリアルでは、行の挿入ツールを使用してPERFORMANCE_PARTS表に移入済の6つの行を追加する方法について説明します。
- チュートリアル: 「データ」ペインにある表のデータの変更
  このチュートリアルでは、「データ」ペインのPERFORMANCE_PARTS表の3つのWEIGHT値を変更する方法を説明します。
- チュートリアル: 選択した行の削除ツールを使用した表内の行の削除
  このチュートリアルでは、選択した行の削除ツールを使用して、PERFORMANCE_PARTS表から行を削除する方法について説明します。
- 索引の管理
  表の1つ以上の列に索引を作成すると、その表に対するSQL文の実行を高速化できます。索引は、適切に使用すれば、ディスク入力/出力(I/O)を削減させる重要な手段となります。
- 表の削除
  表を削除するには、SQL Developerの「接続」フレームと削除ツールまたはDDL文のDROP TABLEを使用します。

関連項目:

- 「チュートリアル: SQL DeveloperによるEMPLOYEES表のプロパティとデータの表示」
- 表の作成および移入について説明しているSQL Developerチュートリアルは、『*Oracle SQL Developerユーザー*

ズ・ガイド』を参照してください。

- 表の一般情報は、『*Oracle Database概要*』を参照してください。

親トピック: スキーマ・オブジェクトの作成および管理

4.2.1 SQLデータ型について

表を作成するときは、各列にSQLデータ型を指定する必要があります。列がどのような値を持つことができるかは、列のデータ型によって決まります。

たとえば、DATE型の列は、'01-MAY-05' という値を持つことはできますが、数値2や文字値'shoe'を持つことはできません SQLのデータ型には、ビルトインとユーザー定義の2種類があります。(PL/SQLには追加のデータ型があります。詳細は、「PL/SQLデータ型について」を参照してください。)

関連項目:

- 埋込みSQLデータ型の要約については、『*Oracle Database SQL言語リファレンス*』を参照してください。
- 各埋込みSQLデータ型の概要については、『*Oracle Database概要*』を参照してください。
- ユーザー定義のデータ型の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。
- 「PL/SQLデータ型について」

親トピック: 表の作成および管理

4.2.2 表の作成

表を作成するには、SQL Developerの表の作成ツールまたはDDL文のCREATE TABLEを使用します。

この項では、従業員の次のような評価に関するデータを含む表を、これらの方法で作成する手順について説明します。

- PERFORMANCE_PARTS。この表には、従業員のパフォーマンスに関して評価対象となるカテゴリおよびカテゴリの相対的な重みが含まれます。
- EVALUATIONS。この表には、従業員情報、評価日、役職、マネージャおよび部門が含まれます。
- SCORES。この表には、各評価の各パフォーマンス・カテゴリに割り当てられたスコアが含まれます。

これらの表は、このドキュメントの多くのチュートリアルおよび例に表示されます。

- チュートリアル: 表の作成ツールを使用した表の作成
  このチュートリアルでは、SQL Developerの表の作成ツールを使用してPERFORMANCE_PARTS表を作成する方法について説明します。
- CREATE TABLE文を使用した表の作成
  ここでは、CREATE TABLE文を使用してEVALUATIONSおよびSCORESの各表を作成する方法について説明します。

親トピック: 表の作成および管理

4.2.2.1 チュートリアル: 表の作成ツールを使用した表の作成

このチュートリアルでは、SQL Developerの表の作成ツールを使用してPERFORMANCE_PARTS表を作成する方法について説明します。

表の作成ツールを使用してPERFORMANCE_PARTS表を作成するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「表」を右クリックします。
3. 選択肢のリストで、「新規の表」をクリックします。

   「表の作成」ウィンドウが開きます。新規の表には行が1つのみあり、デフォルト値が入力されています。

4. 「スキーマ」では、デフォルト値のHR.を受け入れます
5. 「名前」にPERFORMANCE_PARTSと入力します。
6. デフォルトの行:
   - 「PK」(主キー)では、デフォルトのオプション(選択解除)を受け入れます。
   - 「列名」にPERFORMANCE_IDと入力します。
   - 「型」では、デフォルト値のVARCHAR2を受け入れます。
   - 「サイズ」に2と入力します。
   - 「NULL以外」では、デフォルトのオプション(選択解除)を受け入れます。
7. 「列の追加」をクリックします。
8. 「列名」にNAMEと入力します。
9. 「型」では、デフォルト値のVARCHAR2.を受け入れます
10. 「サイズ」に80と入力します。
11. 「列の追加」をクリックします。
12. 「列名」にWEIGHTと入力します。
13. 「タイプ」で、「NUMBER」をメニューから選択します。
14. 「OK」をクリックします。

    表PERFORMANCE_PARTSが作成されます。「接続」フレームの「表」の下にその名前が表示されます。

    この表を作成するためにCREATE TABLE文を表示するには、PERFORMANCE_PARTSを選択して、「SQL」タブをクリックします。

関連項目:

SQL Developerを使用して表を作成する方法の詳細は、『*Oracle SQL Developerユーザーズ・ガイド*』を参照してください。

親トピック: 表の作成

4.2.2.2 CREATE TABLE文を使用した表の作成

ここでは、CREATE TABLE文を使用してEVALUATIONSおよびSCORESの各表を作成する方法について説明します。

例4-1のCREATE TABLE文はEVALUATIONS表を作成します。

例4-2のCREATE TABLE文はSCORES表を作成します。

SQL Developerの「接続」フレームで、EVALUATIONS表とSCORES表を表示するには、「表」を展開します。

例4-1 CREATE TABLEを使用したEVALUATIONS表の作成

```
CREATE TABLE EVALUATIONS (
  EVALUATION_ID    NUMBER(8,0),
  EMPLOYEE_ID      NUMBER(6,0),
  EVALUATION_DATE  DATE,
  JOB_ID           VARCHAR2(10),
  MANAGER_ID       NUMBER(6,0),
  DEPARTMENT_ID    NUMBER(4,0),
  TOTAL_SCORE      NUMBER(3,0)
);
```

結果:

```
Table created.
```

例4-2 CREATE TABLEを使用したSCORES表の作成

```
CREATE TABLE SCORES (
  EVALUATION_ID   NUMBER(8,0),
  PERFORMANCE_ID  VARCHAR2(2),
  SCORE           NUMBER(1,0)
);
```

結果:

```
Table created.
```

関連項目:

CREATE 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。

親トピック: 表の作成

4.2.3 表のデータ整合性の保証

表内のデータが、アプリケーションによってモデル化されるビジネス・ルールを満たすようにするには、制約とアプリケーション・ロジックのいずれかまたは両方を使用します。

ヒント:

可能な場合は常に、アプリケーション・ロジックではなく制約を使用します。Oracle Database は、すべてのデータが制約に従っているかの確認をアプリケーション・ロジックよりも高速に実行できます。

- 制約について
  制約によって、列の値が制限されます。データに対して制約に違反するような変更を試みると、エラーが発生して変更がロールバックされます。移入された表に新しい制約を適用する場合に、既存の行のいずれかが新しい制約に違反していると、エラーが発生します。

- チュートリアル: 既存の表への制約の追加
  このチュートリアルでは、SQL DeveloperツールおよびALTER TABLE文の両方を使用して既存の表に制約を追加する方法を説明します。

関連項目:

- 制約の種類の一般情報は、『*Oracle Database概要*』を参照してください
- 制約の構文情報は、『*Oracle Database SQL言語リファレンス*』を参照してください。
- 制約の有効化および無効化の詳細は、『*Oracle Database開発ガイド*』を参照してください。

親トピック: 表の作成および管理

4.2.3.1 制約について

制約によって、列の値が制限されます。データに対して制約に違反するような変更を試みると、エラーが発生して変更がロールバックされます。移入された表に新しい制約を適用する場合に、既存の行のいずれかが新しい制約に違反していると、エラーが発生します。

制約は有効または無効にできます。デフォルトでは、有効の状態で作成されます。

制約タイプは次のとおりです。

- **Not Null**。値がNULLになることを禁止します。

  EMPLOYEES表では、LAST_NAME列がNOT NULL制約を持ち、全従業員に対して姓を持たせるというビジネス・ルールが適用されます。

- **一意**。同じ列または列の組合せの中で複数の行が同じ値を持つことを禁止しますが、NULLの値を許可します。

  EMPLOYEES表では、EMAIL列がUNIQUE制約を持つため、従業員は電子メール・アドレスを持たなくてもよいが、他の従業員と同じ電子メール・アドレスを持つことはできないというビジネス・ルールが適用されます。

- 主キー。これは、NOT NULLとUNIQUEの組合せです。

  EMPLOYEES表では、EMPLOYEE_ID列がPRIMARY KEY制約を持つため、全従業員に対して一意の識別番号を持たせるというビジネス・ルールが適用されます。

- **外部キー**。ある表の値が、別の表の値と一致することを要求します。

  EMPLOYEES表では、JOB_ID列がJOBS表を参照するFOREIGN KEY制約を持つため、従業員はJOBS表に存在しないJOB_IDを持つことができないというビジネス・ルールが適用されます。

- **チェック**。指定された条件を値が満たすことを要求します。

  EMPLOYEES表にはCHECK制約がありません。ただし、EMPLOYEESに新しい列EMPLOYEE_AGEが必要であり、全従業員が18歳以上である必要があると想定します。制約CHECK (EMPLOYEE_AGE >= 18)によって、そのビジネス・ルールが適用されます。

ヒント:

チェック制約は、必要なチェックを他の制約では実現できない場合にのみ使用します。

- REF: REF列と参照先のオブジェクトの関係をさらに詳しく記述します。

  REF列は、別のオブジェクト・タイプまたはリレーショナル表のオブジェクトを参照します。

  REF制約の詳細は、『*Oracle Database概要*』を参照してください。

関連項目:

- 制約の構文情報は、『*Oracle Database SQL言語リファレンス*』を参照してください。

親トピック: 表のデータ整合性の保証

4.2.3.2 チュートリアル: 既存の表への制約の追加

このチュートリアルでは、SQL DeveloperツールおよびALTER TABLE文の両方を使用して既存の表に制約を追加する方法を説明します。

既存の表に制約を追加するには、SQL DeveloperツールまたはDDL文のALTER TABLEを使用します。このトピックでは、これら2つの方法を使用して、「表の作成」で作成した表に制約を追加する方法を示します。

このチュートリアルでは様々な手順を説明します。最初の手順では、「表の編集」ツールを使用してNOT NULL制約をPERFORMANCE_PARTS表のNAMES列に追加します。その他の手順では、他のツールで制約を追加する方法を示しますが、表の編集ツールを使用して同じ制約を追加することもできます。

注意:

チュートリアルの手順を実行した後、表が持つ制約を表示するには、次のようにします。

1. 「接続」フレームでは、表の名前を選択します。
2. 右側のフレームで、「制約」タブをクリックします。

表のプロパティとデータの表示の詳細は、「チュートリアル: SQL Developer による EMPLOYEES 表のプロパティとデータの表示」を参照してください。

表の編集ツールを使用してNot Null制約を追加するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「表」を展開します。
3. 表のリストで、PERFORMANCE_PARTSを右クリックします。
4. 選択肢のリストで、「編集」をクリックします。
5. 「表の編集」ウィンドウで、「NAME」列をクリックします。
6. 「NULL以外」プロパティを選択します。
7. 「OK」をクリックします。

PERFORMANCE_PARTS表のNAME列にNot Null制約が追加されます。

次の手順では、ALTER TABLE文を使用し、PERFORMANCE_PARTS表のWEIGHT列にNOT NULL制約を追加します。

ALTER TABLE文を使用してNot Null制約を追加するには、次の手順を実行します。

1. hr_connというタブが含まれるペインがあれば、それを選択します。そうでない場合は、「SQL Developerにおける問合せの実行」にあるように、「SQLワークシート」アイコンをクリックします。
2. ワークシート・ペインで、次の文を入力します。

```
ALTER TABLE PERFORMANCE_PARTS
MODIFY WEIGHT NOT NULL;
```

3. 「文の実行」アイコンをクリックします。

   文が実行され、PERFORMANCE_PARTS表のWEIGHT列にNot Null制約が追加されます。

次の手順では、一意を追加ツールを使用してSCORES表に一意制約を追加します。

一意の追加ツールを使用して一意制約を追加するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「表」を展開します。
3. 表のリストで、SCORESを右クリックします。
4. 選択肢のリストで、「制約」を選択します。
5. 選択肢のリストで、「一意の追加」をクリックします。
6. 「一意の追加」ウィンドウで、次の手順を実行します。
   a. 「制約名」にSCORES_EVAL_PERF_UNIQUEと入力します。
   b. 「列1」で、メニューからEVALUATION_IDを選択します。
   c. 「列2」で、メニューからPERFORMANCE_IDを選択します。
   d. 「適用」をクリックします。
7. 「確認」ウィンドウで「OK」をクリックします。

   SCORES_EVAL_PERF_UNIQUEという名前の一意制約が、SCORES表に追加されます。

次の手順では、主キーを追加ツールを使用し、PERFORMANCE_PARTS表のPERFORMANCE_ID列に主キー制約を追加します。

主キーの追加ツールを使用して主キー制約を追加するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「表」を展開します。
3. 表のリストで、PERFORMANCE_PARTSを右クリックします。
4. 選択肢のリストで、「制約」を選択します。
5. 選択肢のリストで、「主キーの追加」をクリックします。
6. 「主キーの追加」ウィンドウで、次の手順を実行します。
   a. 「主キー名」にPERF_PERF_ID_PKと入力します。
   b. 「列1」で、メニューからPERFORMANCE_IDを選択します。

c. 「適用」をクリックします。

7. 「確認」ウィンドウで「OK」をクリックします。

PERFORMANCE_PARTS表のPERFORMANCE_ID列に、PERF_PERF_ID_PKという名前の主キー制約が追加されます。

次の手順では、ALTER TABLE文を使用し、EVALUATIONS表のEVALUATION_ID列に主キー制約を追加します。

ALTER TABLE文を使用して主キー制約を追加するには、次の手順を実行します。

1. hr_connというタブが含まれるペインがあれば、それを選択します。そうでない場合は、「SQL Developerにおける問合せの実行」にあるように、「SQLワークシート」アイコンをクリックします。
2. ワークシート・ペインで、次の文を入力します。

```
ALTER TABLE EVALUATIONS
ADD CONSTRAINT EVAL_EVAL_ID_PK PRIMARY KEY (EVALUATION_ID);
```

3. 「文の実行」アイコンをクリックします。

文が実行され、EVALUATIONS表のEVALUATION_ID列に主キー制約が追加されます。

次の手順では、「外部キーの追加」ツールを使用してSCORES表に2つの外部キー制約を追加します。

外部キーの追加ツールを使用して2つの外部キー制約を追加するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「表」を展開します。
3. 表のリストで、SCORESを右クリックします。
4. 選択肢のリストで、「制約」を選択します。
5. 選択肢のリストで、「外部キーの追加」をクリックします。
6. 「外部キーの追加」ウィンドウで、次の手順を実行します。
   a. 「制約名」にSCORES_EVAL_FKと入力します。
   b. 「列名」で、メニューからEVALUATION_IDを選択します。
   c. 「参照表名」で、メニューからEVALUATIONSを選択します。
   d. 「参照元の列」で、メニューからEVALUATION_IDを選択します。
   e. 「適用」をクリックします。
7. 「確認」ウィンドウで「OK」をクリックします。

EVALUATIONS表のEVALUTION_ID列を参照するSCORES表のEVALUTION_ID列に、SCORES_EVAL_FKという名前の外部キー制約が追加されます。

次の手順では、別の外部キー制約をSCORES表に追加します。

8. 表のリストで、SCORESを右クリックします。
9. 表のリストで、「制約」を選択します。
10. 選択肢のリストで、「外部キーの追加」をクリックします。

「外部キーの追加」ウィンドウが開きます。

11. 「外部キーの追加」ウィンドウで、次の手順を実行します。

a. 「制約名」にSCORES_PERF_FKと入力します。

b. 「列名」で、メニューからPERFORMANCE_IDを選択します。

c. 「参照表名」で、メニューからPERFORMANCE_PARTSを選択します。

d. 「参照元の列」で、メニューからPERFORMANCE_IDを選択します。

e. 「適用」をクリックします。

12. 「確認」ウィンドウで「OK」をクリックします。

EVALUATIONS表のEVALUTION_ID列を参照するSCORES表のEVALUTION_ID列に、SCORES_PERF_FKという名前の外部キー制約が追加されます。

次の手順では、ALTER TABLE文を使用し、EVALUATIONS表のEMPLOYEE_ID列に外部キー制約を追加し、EMPLOYEES表のEMPLOYEE_ID列を参照します。

ALTER TABLE文を使用して外部キー制約を追加するには、次の手順を実行します。

1. hr_connというタブが含まれるペインがあれば、それを選択します。そうでない場合は、「SQL Developerにおける問合せの実行」にあるように、「SQLワークシート」アイコンをクリックします。
2. ワークシート・ペインで、次の文を入力します。

```
ALTER TABLE EVALUATIONS
ADD CONSTRAINT EVAL_EMP_ID_FK FOREIGN KEY (EMPLOYEE_ID)
REFERENCES EMPLOYEES (EMPLOYEE_ID);
```

3. 「文の実行」アイコンをクリックします。

文が実行され、EMPLOYEES表のEMPLOYEE_ID列を参照するEVALUATIONS表のEMPLOYEE_ID列に外部キー制約が追加されます。

次の手順では、チェックを追加ツールを使用してSCORES表にチェック制約を追加します。

チェックの追加ツールを使用してチェック制約を追加するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「表」を展開します。
3. 表のリストで、SCORESを右クリックします。
4. 選択肢のリストで、「制約」を選択します。
5. 選択肢のリストで、「チェックの追加」をクリックします。
6. 「チェックの追加」ウィンドウで、次の手順を実行します。

a. 「制約名」にSCORE_VALIDと入力します。

b. 「CHECK条件」に、score >= 0 and score <+ 9と入力します。

c. 「ステータス」では、デフォルトのENABLEを受け入れます。

d. 「適用」をクリックします。

7. 「確認」ウィンドウで「OK」をクリックします。

SCORE_VALIDという名前のチェック制約が、SCORES表に追加されます。

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。
- SQL Developerで作成する表に制約を追加する方法については、『*Oracle SQL Developerユーザーズ・ガイド*』を参照してください。
- CREATE TABLE文で作成する表に制約を追加する方法については、『Oracle Database SQL言語リファレンス』を参照してください。

親トピック: 表のデータ整合性の保証

4.2.4 チュートリアル: 行の挿入ツールによる表への行の追加

このチュートリアルでは、行の挿入ツールを使用してPERFORMANCE_PARTS表に移入済の6つの行を追加する方法について説明します。

行の挿入ツールを使用してPERFORMANCE_PARTS表に行を追加するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「表」を展開します。
3. 表のリストで、PERFORMANCE_PARTSを選択します。
4. 右側のフレームで、「データ」タブをクリックします。

   「データ」ペインが表示され、PERFORMANCE_PARTS表の列名が表示されますが、行は表示されません。
5. 「データ」ペインで「行の挿入」アイコンをクリックします。

   列が空の新しい行が表示されます。行番号の周りに、挿入がコミットされていないことを示す緑色の境界線が表示されます。
6. 列ヘッダーPERFORMANCE_IDの下にあるセルをクリックします。
7. PERFORMANCE_IDの値にWMを入力します。
8. [Tab]キーを押すか、列ヘッダーNAMEの下にあるセルをクリックします。
9. NAMEの値にWorkload Managementと入力します。
10. [Tab]キーを押すか、列ヘッダーWEIGHTの下にあるセルをクリックします。
11. WEIGHTの値に0.2と入力します。
12. [Enter]キーを押します。
13. 手順5から12を繰り返して、2番目の行を追加し、次の値を入力します。
    - PERFORMANCE_IDにBRと入力します。
    - NAMEにBuilding Relationshipsと入力します。
    - WEIGHTに0.2と入力します。
14. 手順5から12を繰り返して、3番目の行を追加し、次の値を入力します。
    - PERFORMANCE_IDにCFと入力します。
    - NAMEにCustomer Focusと入力します。
    - WEIGHTに0.2と入力します。
15. 手順5から12を繰り返して、4番目の行を追加し、次の値を入力します。
    - PERFORMANCE_IDにCMと入力します。

- NAMEにCommunicationと入力します。
- WEIGHTに0.2と入力します。

16. 手順5から12を繰り返して、5番目の行を追加し、次の値を入力します。
    - PERFORMANCE_IDにTWと入力します。
    - NAMEにTeamworkと入力します。
    - WEIGHTに0.2と入力します。
17. 手順5から12を繰り返して、6番目の行を追加し、次の値を入力します。
    - PERFORMANCE_IDにROと入力します。
    - NAMEにResults Orientationと入力します。
    - WEIGHTに0.2と入力します。
18. 「変更のコミット」アイコンをクリックします。

    行番号の周りの緑色の境界線が消えます。

    「データ」ペインの下に、ラベルメッセージ - ログがあります。

19. メッセージ - ログペインに、「コミットは成功しました」という記述があることを確認します。
20. 「データ」ペインで新しい行を確認します。

関連項目:

「INSERT文について」

親トピック: 表の作成および管理

4.2.5 チュートリアル:「データ」ペインにある表のデータの変更

このチュートリアルでは、「データ」ペインのPERFORMANCE_PARTS表の3つのWEIGHT値を変更する方法を説明します。

PERFORMANCE_PARTS表は、"「チュートリアル: 行の挿入ツールによる表への行の追加」で移入されました。

「データ」ペインを使用してPERFORMANCE_PARTS表のデータを変更するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「表」を展開します。
3. 表のリストで、PERFORMANCE_PARTSを選択します。
4. 右側のフレームで、「データ」タブをクリックします。
5. 「データ」ペインのNAMEが"Workload Management"である行で、次の手順を実行します。
    a. WEIGHT値をクリックします。
    b. 値0.3を入力します。
    c. [Enter]キーを押します。

    行番号の左に、変更がコミットされていないことを示すアスタリスクが表示されます。
6. NAMEが"Building Relationships"である行で、次の手順を実行します。

a. WEIGHT値をクリックします。

b. 値0.15を入力します。

c. [Enter]キーを押します。

行番号の左に、変更がコミットされていないことを示すアスタリスクが表示されます。

7. NAMEが"Customer Focus"である行で、次の手順を実行します。

a. WEIGHT値をクリックします。

b. 値0.15を入力します。

c. [Enter]キーを押します。

行番号の左に、変更がコミットされていないことを示すアスタリスクが表示されます。

8. 「変更のコミット」アイコンをクリックします。

行番号の左のアスタリスクが消えます。

9. 「データ」ペインの下のメッセージ - ログペインに、「コミットは成功しました」という記述があることを確認します。

10. 「データ」ペインで新しいデータを確認します。

関連項目:

「UPDATE文について」

親トピック: 表の作成および管理

4.2.6 チュートリアル: 選択した行の削除ツールを使用した表内の行の削除

このチュートリアルでは、選択した行の削除ツールを使用してPERFORMANCE_PARTS表から行を削除する方法について説明します。

PERFORMANCE_PARTS表は、"「チュートリアル: 行の挿入ツールによる表への行の追加」で移入されました。

選択した行の削除ツールを使用してPERFORMANCE_PARTSから行を削除するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「表」を展開します。
3. 表のリストで、PERFORMANCE_PARTSを選択します。
4. 右側のフレームで、「データ」タブをクリックします。
5. 「データ」ペインで、NAMEが"Results Orientation"である行をクリックします。
6. 「選択した行の削除」アイコンをクリックします。

行番号の周りに、削除がコミットされていないことを示す赤色の境界線が表示されます。

7. 「変更のコミット」アイコンをクリックします。

行が削除されます。

8. 「データ」ペインの下のメッセージ - ログペインに、「コミットは成功しました」という記述があることを確認します。

注意:

表内のすべての行を削除しても、空の表が残ります。表を削除する方法は、「表の削除」を参照してください。

関連項目:

「DELETE文について」

親トピック: 表の作成および管理

4.2.7 索引の管理

表の1つ以上の列に対して索引を作成すると、その表におけるSQL文の実行速度が向上します。索引は、適切に使用すれば、ディスク入力/出力(I/O)を削減させる重要な手段となります。

表に主キーを定義する場合

- 既存の索引が主キー列で始まる場合、Oracle Databaseは主キーのその既存の索引を使用します。既存の索引は一意である必要はありません。

  たとえば、主キー(A, B)を定義する場合、Oracle Databaseは既存の索引(A, B, C)を使用します。
- 既存の索引が主キー列で始まり、制約が即時の場合、Oracle Databaseは主キーに一意索引を作成します。
- 既存の索引が主キー列で始まり、制約が遅延可能の場合、Oracle Databaseは主キーに一意でない索引を作成します。

たとえば、「チュートリアル: 既存の表への制約の追加」では、EVALUATIONS表のEVALUATION_ID列に主キー制約を追加しました。このため、SQL Developerの「接続」フレームで、EVALUATIONS表を選択し、「索引」タブをクリックすると、「索引」ペインに、EVALUATION_ID列に対する一意索引が表示されます。

- チュートリアル: 索引の作成ツールを使用した索引の追加
  このチュートリアルでは、索引の作成ツールを使用してEVALUATIONS表に索引を追加する方法について説明します。
- チュートリアル: 索引の編集ツールを使用した索引の変更
  このチュートリアルでは、索引の編集ツールを使用して、索引EVAL_JOB_IXのソート順序を逆にする方法について説明します。
- チュートリアル: 索引の削除
  このチュートリアルでは、「接続」フレームおよび削除ツールを使用して、索引EVAL_JOB_IXを削除する方法について説明します。

関連項目:

索引の詳細は、次を参照してください。

- 『*Oracle Database*概要』
- 『*Oracle Database*開発ガイド』

親トピック: 表の作成および管理

4.2.7.1 チュートリアル: 索引の作成ツールを使用した索引の追加

このチュートリアルでは、「索引の作成」ツールを使用して、EVALUATIONS表に索引を追加する方法について説明します。

EVALUATIONS表は、例4-1で作成しました。

索引を作成するには、SQL Developerの索引の作成ツールまたはDDL文のCREATE INDEXを使用します。次のようなDDL文を使用します。

```
CREATE INDEX EVAL_JOB_IX
ON EVALUATIONS (JOB_ID ASC) NOPARALLEL;
```

索引の作成ツールを使用してEVALUATIONS表に索引を追加するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「表」を展開します。
3. 表のリストで、EVALUATIONSを右クリックします。
4. 選択肢のリストで、「索引」を選択します。
5. 選択肢のリストで、「索引の作成」を選択します。
6. 「索引の作成」 ウィンドウで、次の手順を実行します。
    a. 「スキーマ」では、デフォルトのHRを受け入れます。
    b. 「名前」にEVAL_JOB_IXと入力します。
    c. 「定義」ペインが表示されない場合は、「定義」タブを選択します。
    d. 「定義」ペインで、「索引のタイプ」に、メニューから「一意」を選択します。
    e. 「式の追加」アイコンをクリックします。

       「順序」が<Not Specified>の式EMPLOYEE_IDが表示されます。
    f. EMPLOYEE_IDを上書きしてJOB_IDと入力します。
    g. 「順序」では、メニューからASC (昇順)を選択します。
    h. 「OK」をクリックします。

       EVALUATIONS表のJOB_ID列に、EVAL_JOB_IXという名前の索引が追加されます。

関連項目:

CREATE INDEX文の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください

親トピック: 索引の管理

4.2.7.2 チュートリアル: 索引の編集ツールを使用した索引の変更

このチュートリアルでは、索引の編集ツールを使用して、索引EVAL_JOB_IXのソート順序を逆にする方法について説明します。

索引を変更するには、SQL Developerの索引の編集ツールまたはDDL文のDROP INDEXとCREATE INDEXを使用します。

次のようなDDL文を使用します。

```
DROP INDEX EVAL_JOB_ID;

CREATE INDEX EVAL_JOB_IX
ON EVALUATIONS (JOB_ID DESC) NOPARALLEL;
```

索引の編集ツールを使用して索引EVAL_JOB_IXのソート順序を逆にするには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「索引」を展開します。
3. 索引のリストで、EVAL_JOB_IXを右クリックします。
4. 選択肢のリストで、「編集」をクリックします。
5. 「索引の編集」ウィンドウで、「順序」をDESCに変更します。
6. 「OK」をクリックします。
7. 「置換の確認」ウィンドウで、「はい」または「いいえ」をクリックします。

関連項目:

ALTER INDEX文の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

親トピック: 索引の管理

4.2.7.3 チュートリアル: 索引の削除

このチュートリアルでは、「接続」フレームおよび削除ツールを使用して、索引EVAL_JOB_IXを削除する方法について説明します。

索引を削除するには、SQL Developerの「接続」フレームと削除ツールまたはDDL文DROP INDEXのいずれかを使用します。次のようなDDL文を使用します。

```
DROP INDEX EVAL_JOB_ID;
```

索引EVAL_JOB_IXを削除するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「索引」を展開します。
3. 索引のリストで、EVAL_JOB_IXを右クリックします。
4. 選択肢のリストで、「削除」をクリックします。
5. 削除ウィンドウで、「適用」をクリックします。
6. 「確認」ウィンドウで「OK」をクリックします。

関連項目:

DROP INDEX文の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

親トピック: 索引の管理

4.2.8 表の削除

表を削除するには、SQL Developerの「接続」フレームと削除ツールまたはDDL文のDROP TABLEを使用します。

> **注意:**
>
> 「表の作成」で作成した表は今後のチュートリアルで必要なため、削除しないでください。表の削除の練習をする際は、簡単な表を作成し、それを削除してください。

削除ツールを使用して表を削除するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「表」を展開します。
3. 表のリストで、削除する表の名前を右クリックします。
4. 選択肢のリストで、「表」を選択します。
5. 選択肢のリストで、「削除」をクリックします。
6. 削除ウィンドウで、「適用」をクリックします。
7. 「確認」ウィンドウで「OK」をクリックします。

関連項目:

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。

親トピック: 表の作成および管理

4.3 ビューの作成および管理

ビューは、表として問合せ結果を表示します。表を使用できるほとんどの場所で、ビューを使用できます。複数の異なる表に格納された情報に何度もアクセスする必要がある場合に、ビューが役立ちます。

- ビューの作成
  ビューを作成するには、SQL Developerのビューの作成ツールまたはDDL文のCREATE VIEWを使用します。
- ビューの問合せの変更
  ビューで問合せを変更するには、OR REPLACE句を持つDDL文CREATE VIEWを使用します。
- チュートリアル: 名前の変更ツールを使用したビュー名の変更
  このチュートリアルでは、名前の変更ツールを使用して、SALESFORCEビューの名前を変更する方法について説明します。
- ビューの削除
  ビューを削除するには、SQL Developerの「接続」フレームと削除ツールまたはDDL文のDROP VIEWを使用します。

関連項目:

- 問合せの詳細は、「表データの選択」を参照してください。
- ビューの一般情報は、『*Oracle Database概要*』を参照してください。

親トピック: スキーマ・オブジェクトの作成および管理

4.3.1 ビューの作成

ビューを作成するには、SQL Developerのビューの作成ツールまたはDDL文のCREATE VIEWを使用します。

このトピックでは、これら両方の方法を使用して、次のようなビューを作成する方法を示します。

- SALESFORCE。営業部に所属する従業員の名前と給与が含まれます。
- EMP_LOCATIONS。全従業員の名前と事業所が含まれます。

  このビューは、「INSTEAD OFトリガーの作成」で使用されます。

- チュートリアル: ビューの作成ツールを使用したビューの作成
  このチュートリアルでは、ビューの作成ツールを使用してSALESFORCE表を作成する方法について説明します。
- CREATE VIEW文を使用したビューの作成
  この例では、CREATE VIEW文を使用して、4つの表を結合するEMP_LOCATIONSビューを作成します。

関連項目:

- SQL Developerを使用してビューを作成する方法の詳細は、『*Oracle SQL Developerユーザーズ・ガイド*』を参照してください。
- CREATE 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。

親トピック: ビューの作成および管理

4.3.1.1 チュートリアル: ビューの作成ツールを使用したビューの作成

このチュートリアルでは、ビューの作成ツールを使用してSALESFORCEビューを作成する方法について説明します。

ビューの作成ツールを使用してSALESFORCEビューを作成するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「ビュー」を右クリックします。
3. 選択肢のリストで、「新規ビュー」をクリックします。

   「ビューの作成」ウィンドウが開きます。新規のビューには、デフォルト値が入力されています。

4. 「スキーマ」では、デフォルト値のHRを受け入れます。
5. 「名前」にSALESFORCEと入力します。
6. 「SQL問合せ」ペインが表示されない場合は、「SQL問合せ」タブをクリックします。
7. 「SQL問合せ」ペインの「SQL問合せ」フィールドで、次の手順を実行します。
   - SELECTの後に次のように入力します。

     ```
     FIRST_NAME || ' ' || LAST_NAME "Name", SALARY*12 "Annual Salary"
     ```

   - FROMの後に次のように入力します。

     ```
     EMPLOYEES WHERE DEPARTMENT_ID = 80
     ```

8. 「構文チェック」をクリックします。

9. 構文の結果の下に、「SQLにエラーはありませんでした。」以外のメッセージが表示された場合は、手順7に戻り、問合せの構文エラーを修正します。
10. 「OK」をクリックします。

ビューSALESFORCEが作成されます。これを表示するには、「接続」フレームで「ビュー」を展開します。

このビューを作成するためにCREATE VIEW文を表示するには、その名前を選択して、「SQL」タブをクリックします。

関連項目:

SQL Developerを使用してビューを作成する方法の詳細は、『*Oracle SQL Developerユーザーズ・ガイド*』を参照してください。

親トピック: ビューの作成

4.3.1.2 CREATE VIEW文を使用したビューの作成

この例では、CREATE VIEW文を使用して、4つの表を結合するEMP_LOCATIONSビューを作成します。

例4-3のCREATE VIEW文は、4つの表を結合するEMP_LOCATIONS表を作成します。(結合の詳細は、「複数の表からのデータの選択」を参照してください。)

例4-3 CREATE VIEW文を使用したEMP_LOCATIONSビューの作成

```
CREATE VIEW EMP_LOCATIONS AS
SELECT e.EMPLOYEE_ID,
  e.LAST_NAME || ', ' || e.FIRST_NAME NAME,
  d.DEPARTMENT_NAME DEPARTMENT,
  l.CITY CITY,
  c.COUNTRY_NAME COUNTRY
FROM EMPLOYEES e, DEPARTMENTS d, LOCATIONS l, COUNTRIES c
WHERE e.DEPARTMENT_ID = d.DEPARTMENT_ID AND
 d.LOCATION_ID = l.LOCATION_ID AND
 l.COUNTRY_ID = c.COUNTRY_ID
ORDER BY LAST_NAME;
```

結果:

```
View EMP_LOCATIONS created.
```

関連項目:

CREATE 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。

親トピック: ビューの作成

4.3.2 ビューの問合せの変更

ビューで問合せを変更するには、OR REPLACE句を持つDDL文CREATE VIEWを使用します。

例4-4のCREATE OR REPLACE VIEW文により、SALESFORCEビューの問合せが変更されます。

例4-4 SALESFORCEビューの問合せの変更

```
CREATE OR REPLACE VIEW SALESFORCE AS
  SELECT FIRST_NAME || ' ' || LAST_NAME "Name",
  SALARY*12 "Annual Salary"
  FROM EMPLOYEES
  WHERE DEPARTMENT_ID = 80 OR DEPARTMENT_ID = 20;
```

結果:

```
View SALESFORCE created.
```

関連項目:

OR REPLACE句を使用したCREATE VIEWの詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

親トピック: ビューの作成および管理

4.3.3 チュートリアル: 名前変更ツールを使用したビュー名の変更

このチュートリアルでは、名前の変更ツールを使用して、SALESFORCEビューの名前を変更する方法について説明します。

ビューの名前を変更するには、SQL Developerの名前変更ツールまたはRENAME文を使用します。次のようなDDL文を使用します。

```
RENAME SALESFORCE to SALES_MARKETING;
```

名前の変更ツールを使用してSALESFORCEビューを変更するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「ビュー」を展開します。
3. ビューのリストで、SALESFORCEを右クリックします。
4. 選択肢のリストで、「名前変更」を選択します。
5. 「名前変更」ウィンドウで、「新規ビュー名」フィールドにSALES_MARKETINGと入力します。
6. 「適用」をクリックします。
7. 「確認」ウィンドウで「OK」をクリックします。

関連項目:

RENAME文の詳細は、*Oracle Database SQLリファレンス*を参照

親トピック: ビューの作成および管理

4.3.4 ビューの削除

ビューを削除するには、SQL Developerの「接続」フレームと削除ツールまたはDDL文のDROP VIEWを使用します。

次のチュートリアルでは、「接続」フレームと削除ツールを使用して、SALES_MARKETINGビュー(「チュートリアル: 名前変更ツールを使用したビュー名の変更」で変更)を削除する方法を示します。次のようなDDL文を使用します。

```
DROP VIEW SALES_MARKETING;
```

削除ツールを使用してSALES_MARKETINGビューを削除するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「ビュー」を展開します。
3. ビューのリストで、SALES_MARKETINGを右クリックします。
4. 選択肢のリストで、「削除」をクリックします。
5. 「削除」ウィンドウで、「適用」をクリックします。
6. 「確認」ウィンドウで「OK」をクリックします。

関連項目:

DROP 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。

親トピック: ビューの作成および管理

4.4 順序の作成および管理

順序は一意の連続した値を生成できるスキーマ・オブジェクトであり、一意の主キーが必要な場合に非常に役立ちます。順序は、疑似列CURRVALおよびNEXTVALを通じて使用します。これらの疑似列は、それぞれ順序の現在と次の値を返します。

順序の作成後、NEXTVALを使用して最初の値を取得し、順序を初期化する必要があります。順序を初期化した後のみ、CURRVALは現在の値を戻します。

HRスキーマにはDEPARTMENTS_SEQUENCE、EMPLOYEES_SEQUENCE、LOCATIONS_SEQUENCEという3つの順序があります。

> ヒント:
>
> 表の主キーを移入するために順序を使用する場合は、順序に、その目的を反映する名前を付けます。(ここでは、ネーミング規則 *TABLE_NAME*_SEQUENCE を使用します。)

- チュートリアル: 順序の作成
  このチュートリアルでは、データベース順序の作成ツールを使用して順序を作成し、それを使用して、EVALUATIONS表に主キーを生成する方法を示します。
- 順序の削除
  順序を削除するには、SQL Developerの「接続」フレームと削除ツールまたはDDL文のDROP SEQUENCEを使用します。

関連項目:

- 順序の概要については、『*Oracle Database概要*』を参照してください。
- CURRVALおよびNEXTVAL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。
- 順序の管理の詳細は、『*Oracle Database管理者ガイド*』を参照してください。
- 「順序を作成するインストール・スクリプトの編集」
- 「順序および同時実行性について」

親トピック: スキーマ・オブジェクトの作成および管理

4.4.1 チュートリアル: 順序の作成

このチュートリアルでは、データベース順序の作成ツールを使用して順序を作成し、それを使用して、EVALUATIONS表に主キーを生成する方法を示します。

EVALUATIONS表は、例4-1で作成しました。

順序を作成するには、SQL Developerの順序の作成ツールまたはDDL文CREATE SEQUENCEを使用します。次のようなDDL文を使用します。

```
CREATE SEQUENCE evaluations_sequence
INCREMENT BY 1
START WITH 1 ORDER;
```

データベース順序の作成ツールを使用してEVALUATIONS_SEQUENCEを作成するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「順序」を右クリックします。
3. 選択肢のリストで、「新規順序」をクリックします。
4. 「順序の作成」ウィンドウで、「名前」フィールドにEVALUATIONS_SEQUENCEと入力して、デフォルト値"SEQUENCE1"を上書きします。
5. 「プロパティ」ペインが表示されない場合は、「プロパティ」タブをクリックします。
6. 「プロパティ」ペインで、次の手順を実行します。
    a. 「増分」フィールドに1を入力します。
    b. 「接続」フィールドに1を入力します。
    c. 残りのフィールドでは、デフォルト値を受け入れます。
    d. 「OK」をクリックします。

    順序EVALUATIONS_SEQUENCEが作成されます。「接続」フレームの「順序」の下にその名前が表示されます。

関連項目:

- SQL Developerを使用して順序を作成する方法の詳細は、『*Oracle SQL Developerユーザーズ・ガイド*』を参照してください。
- CREATE SEQUENCE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。
- EVALUATIONS_SEQUENCEによって作成された主キーをEVALUATIONS表に挿入するトリガーを作成する方法については、「チュートリアル: 行が挿入される前に主キーを生成するトリガーの作成」を参照してください。

親トピック: 順序の作成および管理

4.4.2 順序の削除

順序を削除するには、SQL Developerの「接続」フレームと削除ツールまたはDDL文のDROP SEQUENCEを使用します。

この文は、順序EVALUATIONS_SEQUENCEを削除します。

```
DROP SEQUENCE EVALUATIONS_SEQUENCE;
```

> **注意:**
>
> 例 5-3 で必要なため、順序 EVALUATIONS_SEQUENCE を削除しないでください。順序の削除の練習をする際は、簡単な順序を作成し、それを削除してください。

削除ツールを使用して順序を削除するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「順序」を展開します。
3. 順序のリストで、削除する順序の名前を右クリックします。
4. 選択肢のリストで、「削除」をクリックします。
5. 削除ウィンドウで、「適用」をクリックします。
6. 「確認」ウィンドウで「OK」をクリックします。

関連項目:

DROP SEQUENCE文の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

親トピック: 順序の作成および管理

4.5 シノニムの作成および管理

シノニムとは、他のスキーマ・オブジェクトの別名のことです。シノニムを使用する理由は、セキュリティ(たとえば、オブジェクトの所有者と位置を分からなくするため)と簡素化のためです。

簡素化には、次のような例があります。

- ACME_CO.SALES_DATAのような長いオブジェクト名のかわりに、SALESなどの短いシノニムを使用します。
- あるオブジェクトを使用するすべてのアプリケーションでそのオブジェクトの名前を変更するのではなく、名前を変更したオブジェクトに対してシノニムを使用します。

  たとえば、ユーザーのアプリケーションがDEPARTMENTSという名前の表を使用し、その名前がDIVISIONSに変更された場合、その表に対してDEPARTMENTSシノニムを使用して、元の名前での参照を継続します。

- シノニムの作成
  シノニムを作成するには、SQL Developerのデータベース・シノニムの作成ツールまたはDDL文CREATE SYNONYMを使用します。
- シノニムの削除
  シノニムを削除するには、SQL Developerの「接続」フレームと削除ツールまたはDDL文のDROP SYNONYMを使

用します。

関連項目:

シノニムの一般情報は、『*Oracle Database*概要』を参照してください。

親トピック: スキーマ・オブジェクトの作成および管理

4.5.1 シノニムの作成

シノニムを作成するには、SQL Developerのデータベース・シノニムの作成ツールまたはDDL文CREATE SYNONYM を使用します。

次のチュートリアルでは、データベース・シノニムの作成ツールを使用して、EMPLOYEES表のためのシノニムEMPを作成する方法を示します。次のようなDDL文を使用します。

```
CREATE SYNONYM EMPL FOR EMPLOYEES;
```

データベース・シノニムの作成ツールを使用してシノニムEMPを作成するには、次のステップを実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「シノニム」を右クリックします。
3. 選択肢のリストで、「新規シノニム」をクリックします。
4. 「新規シノニム」ウィンドウで、次の手順を実行します。
    a. 「シノニム名」フィールドにEMPLと入力します。
    b. 「オブジェクト所有者」フィールドで、HRをメニューから選択します。
    c. 「オブジェクト名」フィールドで、EMPLOYEESをメニューから選択します。

        シノニムは特定のスキーマ・オブジェクト、この場合、EMPLOYEES表を参照します。

    d. 「適用」をクリックします。
5. 「確認」ウィンドウで「OK」をクリックします。

    シノニムEMPLが作成されます。これを表示するには、「接続」フレームで「シノニム」を展開します。これによって、EMPLOYEESのかわりにEMPLを使用できるようになります。

関連項目:

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。

親トピック: シノニムの作成および管理

4.5.2 シノニムの削除

シノニムを削除するには、SQL Developerの「接続」フレームと削除ツールまたはDDL文のDROP SYNONYMを使用します。

この文は、シノニムEMPを削除します。

```
DROP SYNONYM EMP;
```

削除ツールを使用してシノニムを削除するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「シノニム」を展開します。
3. シノニムのリストで、削除するシノニムの名前を右クリックします。
4. 選択肢のリストで、「削除」をクリックします。
5. 削除ウィンドウで、「適用」をクリックします。
6. 「確認」ウィンドウで「OK」をクリックします。

関連項目:

DROP 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。

親トピック: シノニムの作成および管理

5 ストアド・サブプログラムおよびパッケージの開発

ストアド・プログラムとパッケージは、多くの様々なデータベース・アプリケーションのビルディング・ブロックとして使用できます。

- ストアド・サブプログラムについて
  ストアド・サブプログラムは、データベースに格納されたサブプログラムです。ストアド・サブプログラムは、データベースに格納されているため、多様なデータベース・アプリケーションのビルディング・ブロックとして使用できます。
- パッケージについて
  パッケージとは、関連するサブプログラムとそれらで使用する宣言カーソルと変数からなるPL/SQLユニットです。サブプログラムをパッケージに配置することをお薦めします。
- PL/SQL識別子について
  PL/SQL、サブプログラム、パッケージ、パラメータ、変数、定数、例外および宣言カーソルには、それぞれ名前があり、それぞれの名前がPL/SQL識別子です。
- PL/SQLデータ型について
  PL/SQLの定数、変数、サブプログラム・パラメータおよび関数の戻り値のそれぞれに、データ型があり、このデータ型により、格納の形式、制約、値の有効範囲および実行される可能性がある操作が決定されます。
- スタンドアロンのサブプログラムの作成および管理
  スタンドアロンのPL/SQLサブプログラムを作成および管理できます。
- パッケージの作成および管理
  PL/SQLサブプログラムを作成および管理できます。
- 変数および定数の宣言と値の割当て
  パッケージ仕様で宣言された変数または定数は、このパッケージにアクセスしている任意のプログラムで使用可能です。パッケージ本体またはサブプログラムで宣言された変数または定数は、そのパッケージまたはサブプログラムに対してローカルです。定数を宣言する場合、その定数に初期値を割り当てる必要があります。
- プログラム・フローの制御
  入力の順序に従って文を実行するSQLとは異なり、PL/SQLには、プログラム・フローを制御できる制御文があります。
- レコードおよびカーソルの使用
  レコードにデータ値を格納し、カーソルを結果セットおよび関連の処理情報へのポインタとして使用できます。
- 連想配列の使用
  連想配列はコレクションの型です。
- 例外の処理(実行時エラー)
  PL/SQLコードで実行時に発生する例外を処理できます。

5.1 ストアド・サブプログラムについて

ストアド・サブプログラムは、データベースに格納されたサブプログラムです。ストアド・サブプログラムは、データベースに格納されているため、多様なデータベース・アプリケーションのビルディング・ブロックとして使用できます。

サブプログラムは、特定の問題を解決したり、関連する一連のタスクを実行するSQL文およびPL/SQL文で構成されているPL/SQLユニットです。サブプログラムはパラメータを持つことができ、値は起動元から提供されます。サブプログラムは、プロシージャの場合もファンクションの場合もあります。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。

ストアド・サブプログラムは、データベースに格納されているため、多様なデータベース・アプリケーションのビルディング・ブロックとして使用できます。別のサブプログラムまたは無名ブロック内で宣言されたサブプログラムは、**ネストされたサブプログラム**または**ローカル・サブプログラム**と呼ばれます。宣言されたサブプログラムまたはブロックの外部から呼び出すことはできません。**無名ブロック**

とは、データベースに格納されていないブロックです。

ストアド・サブプログラムは2種類あります。

- **スキーマ・レベルで作成されたスタンドアロンのサブプログラム**
- パッケージ内で作成される**パッケージ・サブプログラム**

スタンドアロンのサブプログラムは、一部のプログラム・ロジックのテストに有用ですが、確実にこれらを意図したとおりに作動させる場合、これらをパッケージ内に配置することをお薦めします。

関連項目:

- ストアド・サブプログラムの一般情報は、『*Oracle Database概要*』を参照してください。
- 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。

親トピック: ストアド・サブプログラムおよびパッケージの開発

5.2 パッケージについて

パッケージとは、関連するサブプログラムとそれらで使用する宣言カーソルと変数からなるPL/SQLユニットです。サブプログラムをパッケージに配置することをお薦めします。

サブプログラムをパッケージに含めることをお薦めする理由は次のとおりです。

- パッケージでは、クライアント・プログラムから実装の詳細を隠すことができます。

  クライアント・プログラムから実装の詳細を隠すことは、広く指示されるベスト・プラクティスです。Oracleのカスタマの多くは、このプラクティスに厳密に従っていて、クライアント・プログラムでは、PL/SQLサブプログラムを起動したときにのみデータベースにアクセスできます。一部のカスタマは、クライアント・プログラムでSELECT文を使用し、データベース表から情報を取得することを可能にしていますが、この文は、データベースを変更するすべてのビジネス機能に対するPL/SQLサブプログラムを起動する必要があります。

- パッケージ済サブプログラムは、パッケージ外部からの起動時にパッケージ名による修飾が必要であり、これによりこれらのパッケージ名はパッケージ外部からの起動時に常に確実に機能します。

  たとえば、Oracle Database 11*g*以前のバージョンでCONTINUEというスキーマ・レベル・プロシージャを作成したとします。Oracle Database 11*g* はCONTINUE文を導入しました。したがって、コードをOracle Database 11*g* に移植しても、コンパイルされません。ただし、パッケージ内にプロシージャを作成した場合、コードはプロシージャを *package_name*.CONTINUEとして参照するため、コンパイルできます。

> 注意:
>
> Oracle Database は、多くの PL/SQL パッケージを提供してデータベース機能を拡張し、SQL 機能への PL/SQL によるアクセスを可能にしています。提供されたパッケージは、アプリケーションの作成や、独自のストアド・プロシージャを作成するアイデアのために使用できます。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。

関連項目:

- パッケージの一般情報は、『*Oracle Database概要*』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。
- 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。
- Oracleが提供するPL/SQLパッケージの詳細は、『*Oracle Database PL/SQLパッケージおよびタイプ・リファレンス*』を参照してください。

親トピック: ストアド・サブプログラムおよびパッケージの開発

5.3 PL/SQL識別子について

PL/SQL、サブプログラム、パッケージ、パラメータ、変数、定数、例外および宣言カーソルには、それぞれ名前があり、それぞれの名前がPL/SQL識別子です。

識別子は最短で1文字、最長で30文字です。最初は文字である必要がありますが、以降は文字、数字、ドル記号($)、アンダースコア(_)またはシャープ記号(#)を使用できます。たとえば、次に示すのが許容可能な識別子です。

```
X
t2
phone#
credit_limit
LastName
oracle$number
money$$$tree
SN##
try_again_
```

PL/SQLでは、識別子の大/小文字は区別されません。たとえば、PL/SQLは次を同一とみなします。

```
lastname
LastName
LASTNAME
```

PL/SQLの予約語は、識別子として使用できません。PL/SQLキーワードは識別子として使用できますが、推奨されていません。PL/SQLの予約語およびキーワードのリストは、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。

関連項目:

- PL/SQL識別子の追加の一般情報は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。
- PL/SQLのネーミング規則の詳細は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。
- PL/SQL識別子の適用範囲および可視性の詳細は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。
- PL/SQL識別子に関するデータの収集方法の詳細は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。
- PL/SQLが識別子名を解決する方法の詳細は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。

親トピック: ストアド・サブプログラムおよびパッケージの開発

5.4 PL/SQLデータ型について

PL/SQLの定数、変数、サブプログラム・パラメータおよび関数の戻り値のそれぞれに、データ型があり、このデータ型により、格納の形式、制約、値の有効範囲および実行される可能性がある操作が決定されます。

PL/SQLデータ型は、SQLデータ型(VARCHAR2、NUMBER、DATEなど)またはPL/SQLのみのデータ型です。後者には、BOOLEAN、RECORD、REF CURSORに加え、多くの事前定義サブタイプが含まれます。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。

サブタイプは、他のデータ型のサブセットで、**ベース型**と呼ばれます。サブタイプには、そのベース型として同じ有効な操作がありますが、その有効な値のサブセットのみです。サブタイプでは、定数と変数の用途を示すことにより、信頼性の向上、ANSI/ISO型との互換性の提供、および見やすさの改善が可能です。

事前定義された数値のサブタイプPLS_INTEGERは特に便利です。演算に、ベース型が使用するライブラリ算術計算のかわりにハードウェア算術計算が使用されるためです。

スキーマ・レベル(つまり、表内またはスタンドアロン・サブプログラム内)ではPL/SQL-onlyデータ型は使用できません。したがって、ストアド・サブプログラムでこれらのデータ型を使用するには、パッケージに配置する必要があります。

関連項目:

- PL/SQLデータ型の一般情報は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。
- 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。
- 「SQLデータ型について」

親トピック: ストアド・サブプログラムおよびパッケージの開発

5.5 スタンドアロンのサブプログラムの作成および管理

スタンドアロンのPL/SQLサブプログラムを作成および管理できます。

> 注意:
>
> このマニュアルのチュートリアルを行うには、ユーザーHR として、SQL Developer から Oracle Database に接続している必要があります。

- サブプログラム構造について
- チュートリアル: スタンドアロンのプロシージャの作成
  このチュートリアルでは、プロシージャの作成ツールを使用して、行をEVALUATIONS表に追加するADD_EVALUATIONという名前のスタンドアロンのプロシージャを作成する方法を示します。
- チュートリアル: スタンドアロンのファンクションの作成
  このチュートリアルでは、ファンクションの作成ツールを使用して、3つのパラメータがあり、NUMBER型の値を戻す、CALCULATE_SCOREという名前のスタンドアロン・ファンクションを作成する方法を表示します。

- スタンドアロンのサブプログラムの変更
  スタンドアロンのサブプログラムを変更するには、SQL Developerの編集ツールまたはDDL文ALTER PROCEDUREかALTER FUNCTIONのいずれかを使用します。
- チュートリアル: スタンドアロンのファンクションのテスト
  このチュートリアルでは、SQL Developerの実行ツールを使用して、スタンドアロンのファンクションCALCULATE_SCOREをテストする方法について説明します。
- スタンドアロンのサブプログラムの削除
  スタンドアロンのサブプログラムを削除するには、SQL Developerの「接続」フレームと削除ツール、またはDDL文DROP PROCEDUREかDROP FUNCTIONのいずれかを使用します。

親トピック: ストアド・サブプログラムおよびパッケージの開発

5.5.1 サブプログラム構造について

サブプログラムはPL/SQLブロック構造に従います。つまり、サブプログラムには次のものがあります。

- **宣言部分**(オプション)

  宣言部には、型、定数、変数、例外、宣言カーソル、およびネストされたサブプログラムの宣言が含まれます。これらのアイテムは、サブプログラムに対してローカルであり、サブプログラムの実行が完了すると存在しなくなります。

- **実行可能部分**(必須)

  実行可能部分には、値を割り当て、実行を制御し、データを操作する文が含まれます。

- **例外処理部分**(オプション)

  例外処理部には、例外(実行時エラー)を処理するコードがあります。

コメントは、PL/SQLコードの任意の場所に表示可能です。PL/SQLコンパイラには無視されます。プログラムにコメントを追加することで、可読性が向上し、理解を助けます。**単一行コメント**は二重ハイフン(--)で始まり、行の末尾まで拡張されます。**複数行にまたがるコメント**はスラッシュとアスタリスク(/*)で始まり、アスタリスクとスラッシュ(*/)で終わります。

プロシージャの構造は、次のとおりです。

```
PROCEDURE name [ ( parameter_list ) ]
{ IS | AS }
  [ declarative_part ]
BEGIN  -- executable part begins
  statement; [ statement; ]...
[ EXCEPTION -- executable part ends, exception-handling part begins]
  exception_handler; [ exception_handler; ]... ]
END; /* exception-handling part ends if it exists;
        otherwise, executable part ends */
```

ファンクションの構造はプロシージャの構造に似ていますが、RETURN句および少なくとも1つのRETURN文(およびこのマニュアルの範囲外のオプション句)が含まれる点が異なります。

```
FUNCTION name [ ( parameter_list ) ] RETURN data_type [ clauses ]
{ IS | AS }
  [ declarative_part ]
BEGIN  -- executable part begins
  -- at least one statement must be a RETURN statement
  statement; [ statement; ]...
[ EXCEPTION -- executable part ends, exception-handling part begins]
  exception_handler; [ exception_handler; ]... ]
END; /* exception-handling part ends if it exists;
        otherwise, executable part ends */
```

PROCEDUREまたはFUNCTIONで始まりISまたはASの前で終わるコードは、**サブプログラムの署名**です。宣言部分、実行可能部分および例外処理部分は、**サブプログラムの本体**を構成します。例外ハンドラの構文は、「例外および例外ハンドラについて」を参照してください。

関連項目:

サブプログラムの部分の詳細は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。

親トピック: スタンドアロンのサブプログラムの作成および管理

5.5.2 チュートリアル: スタンドアロンのプロシージャの作成

このチュートリアルでは、プロシージャの作成ツールを使用して、EVALUATIONS表に行を追加するADD_EVALUATIONというスタンドアロン・ストアド・プロシージャを作成する方法を示します。

EVALUATIONS表は、例4-1で作成しました。

スタンドアロン・プロシージャを作成するには、SQL Developerツールのプロシージャの作成またはDDL文CREATE PROCEDUREのいずれかを使用します。

プロシージャの作成ツールを使用してスタンドアロン・プロシージャを作成するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「プロシージャ」を右クリックします。
3. 選択肢のリストで、「新規プロシージャ」をクリックします。

   プロシージャの作成ウィンドウが開きます。

4. 「スキーマ」では、デフォルト値のHRを受け入れます。
5. 「名前」では、PROCEDURE1をADD_EVALUATIONに変更します。
6. 「パラメータの追加」アイコンをクリックします。

   列のヘッダーの下に1行表示されます。そのフィールドには、デフォルト値(名前: PARAM1、モード: IN、コピーなし: 選択解除、データ型: VARCHAR2、デフォルト値: 空)が設定されています。

7. 「名前」では、PARAM1をEVALUATION_IDに変更します。
8. 「モード」では、デフォルト値の「IN」を受け入れます。
9. 「「データ型」では、メニューからNUMBERを選択します。
10. 「デフォルト値」は空白のままにします。
11. 「名前」にEMPLOYEE_ID、「データ型」にNUMBERを使用して、手順6から10を繰り返すことにより、2番目のパラメータを追加します。
12. 「名前」にEVALUATION_DATE、「データ型」にDATEを使用して、手順6から10を繰り返すことにより、3番目のパラメータを追加します。
13. 「名前」にJOB_ID、「データ型」にVARCHAR2を使用して、手順6から10を繰り返すことにより、4番目のパラメータを追加します。
14. 「名前」にMANAGER_ID、「データ型」にNUMBERを使用して、手順6から10を繰り返すことにより、5番目のパラメータを追加します。
15. 「名前」にDEPARTMENT_ID、「データ型」にNUMBERを使用して、手順6から10を繰り返すことにより、6番目のパラメータを追加します。
16. 「名前」にTOTAL_SCORE、「データ型」にNUMBERを使用して、手順6から10を繰り返すことにより、7番目の

パラメータを追加します。

17. 「**OK**」をクリックします。

```
CREATE OR REPLACE PROCEDURE ADD_EVALUATION
(
  EVALUATION_ID IN NUMBER
, EMPLOYEE_ID IN NUMBER
, EVALUATION_DATE IN DATE
, JOB_ID IN VARCHAR2
, MANAGER_ID IN NUMBER
, DEPARTMENT_ID IN NUMBER
, TOTAL_SCORE IN NUMBER
) AS
BEGIN
  NULL;
END ADD_EVALUATION;
```

ADD_EVALUATIONペインのタイトルがイタリック・フォントになっています。プロシージャがデータベースに保存されていないことを示しています。

プロシージャの実行部分にある唯一の文がNULLであるため、プロシージャは何も行いません。

18. NULL文を次の文に置換します。

```
INSERT INTO EVALUATIONS (
   evaluation_id,
   employee_id,
   evaluation_date,
   job_id,
   manager_id,
   department_id,
   total_score
)
VALUES (
  ADD_EVALUATION.evaluation_id,
  ADD_EVALUATION.employee_id,
  ADD_EVALUATION.evaluation_date,
  ADD_EVALUATION.job_id,
  ADD_EVALUATION.manager_id,
  ADD_EVALUATION.department_id,
  ADD_EVALUATION.total_score
);
```

(パラメータ名をプロシージャ名で修飾すると、パラメータは同じ名前の列と混同されなくなります。)

19. 「File」メニューから、「Save」を選択します。

Oracle Databaseは、プロシージャをコンパイルして保存します。ADD_EVALUATIONペインのタイトルがイタリック・フォントではなくなります。メッセージ - ログペインには「**コンパイル済**」というメッセージが表示されます。

関連項目:

- SQL Developerを使用してスタンドアロン・プロシージャを作成する別の例は、『*Oracle SQL Developerユーザーズ・ガイド*』を参照してください。
- CREATE PROCEDURE文に適用される一般情報は、"「データ定義言語(DDL)文について」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。

親トピック: スタンドアロンのサブプログラムの作成および管理

5.5.3 チュートリアル: スタンドアロンのファンクションの作成

このチュートリアルでは、ファンクションの作成ツールを使用して、3つのパラメータがあり、NUMBER型の値を戻す、CALCULATE_SCOREという名前のスタンドアロン・ファンクションを作成する方法を表示します。

スタンドアロン・ファンクションを作成するには、SQL Developerツールのファンクションの作成またはDDL文CREATE FUNCTIONのいずれかを使用します。

ファンクションの作成ツールを使用してスタンドアロン・ファンクションを作成するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「ファンクション」を右クリックします。
3. 選択肢のリストで、新規ファンクションをクリックします。

   ファンクションの作成ウィンドウが開きます。

4. 「スキーマ」では、デフォルト値のHRを受け入れます。
5. 「名前」で、FUNCTION1をCALCULATE_SCOREに変更します。
6. 「戻り値の型」で、メニューからNUMBERを選択します。
7. 「パラメータの追加」アイコンをクリックします。

   列のヘッダーの下に1行表示されます。そのフィールドには、デフォルト値(名前: PARAM1、モード: IN、コピーなし: 選択解除、データ型: VARCHAR2、デフォルト値: 空)が設定されています。

8. 「名前」では、PARAM1をcatに変更します。
9. 「モード」では、デフォルト値の「IN」を受け入れます。
10. 「データ型」では、デフォルトのVARCHAR2を受け入れます。
11. 「デフォルト値」は空白のままにします。
12. 「名前」にscore、「データ型」にNUMBERを使用して、手順7から11を繰り返すことにより、2番目のパラメータを追加します。
13. 「名前」にweight、「データ型」にNUMBERを使用して、手順7から11を繰り返すことにより、3番目のパラメータを追加します。
14. 「OK」をクリックします。

    CALCULATE_SCOREペインが開き、ファンクションを作成したCREATE FUNCTION文が表示されます。

```
CREATE OR REPLACE FUNCTION CALCULATE_SCORE
(
  CAT IN VARCHAR2
, SCORE IN NUMBER
, WEIGHT IN NUMBER
) RETURN NUMBER AS
BEGIN
  RETURN NULL;
END CALCULATE_SCORE;
```

    CALCULATE_SCOREペインのタイトルがイタリック・フォントになっています。ファンクションがデータベースに保存されていないことを示しています。

    ファンクションの実行部分にある唯一の文がRETURN NULL文であるため、ファンクションは何も行いません。

15. NULLをscore * weightに置換します。

16. 「File」メニューから、「Save」を選択します。

Oracle Databaseは、ファンクションをコンパイルして保存します。CALCULATE_SCOREペインのタイトルがイタリック・フォントではなくなります。メッセージ - ログペインには「コンパイル済」というメッセージが表示されます。

関連項目:

- CREATE FUNCTION文に適用される一般情報は、「データ定義言語(DDL)文について」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。

親トピック: スタンドアロンのサブプログラムの作成および管理

5.5.4 スタンドアロンのサブプログラムの変更

スタンドアロン・サブプログラムを変更するには、SQL Developerツールの編集またはDDL文ALTER PROCEDUREまたはALTER FUNCTIONのいずれかを使用します。

編集ツールを使用してスタンドアロンのサブプログラムを変更するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「ファンクション」または「プロシージャ」を展開します。

   ファンクションまたはプロシージャのリストが表示されます。

3. 変更するファンクションまたはプロシージャをクリックします。

   「接続」フレームの右側に、フレームが表示されます。上部のタブには変更するサブプログラムの名前が表示されます。「コード」ペインにサブプログラムを作成したコードが表示されます。

   「コード」ペインが書込みモードになっています。(鉛筆アイコンをクリックすると、モードが書込みモードから読取り専用モードに、またはその逆に切り替わります。)

4. 「コード」ペインで、コードを変更します。

   ペインのタイトルはイタリック・フォントになっており、変更がデータベースに保存されていないことを示しています。

5. 「File」メニューから、「Save」を選択します。

   Oracle Databaseは、サブプログラムをコンパイルして保存します。ペインのタイトルがイタリック・フォントではなくなります。メッセージ - ログペインには「コンパイル済」というメッセージが表示されます。

関連項目:

- ALTER PROCEDUREおよびALTER FUNCTION文に適用される一般情報は、「データ定義言語(DDL)文について」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 FUNCTION文の詳細は、*Oracle Database PL/SQLリファレンス*を参照

親トピック: スタンドアロンのサブプログラムの作成および管理

5.5.5 チュートリアル: スタンドアロンのファンクションのテスト

このチュートリアルでは、SQL Developer実行ツールを使用して、スタンドアロンのファンクションCALCULATE_SCOREをテストする方法について説明します。

実行ツールを使用してCALCULATE_SCOREファンクションをテストするには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「ファンクション」を展開します。
3. ファンクションのリストで、CALCULATE_SCOREを右クリックします。
4. 選択肢のリストで、「実行」をクリックします。

   「PL/SQLの実行」ウィンドウが開きます。「PL/SQLブロック」フレームには、次のコードが含まれます。

```
v_Return := CALCULATE_SCORE (
    CAT => CAT,
    SCORE => SCORE,
    WEIGHT => WEIGHT
  );
```

5. SCOREおよびWEIGHTの値を、それぞれ8および0.2に変更します。

```
v_Return := CALCULATE_SCORE (
    CAT => CAT,
    SCORE => 8,
    WEIGHT => 0.2
  );
```

6. 「OK」をクリックします。

   「コード」ペインの下に「実行中」ウィンドウが開き、次の結果が表示されます。

```
Connecting to the database hr_conn.
Process exited.
Disconnecting from the database hr_conn.
```

   「実行中」タブの右に「出力変数」タブが表示されます。

7. 「出力変数」タブをクリックします。

   「変数」および「値」という2つのフレームが表示され、それぞれ<Return Value>および1.6と表示されます。

関連項目:

SQL Developerを使用したプロシージャおよびファンクションの実行とデバッグについては、『*Oracle SQL Developerユーザーズ・ガイド*』を参照してください。

親トピック: スタンドアロンのサブプログラムの作成および管理

5.5.6 スタンドアロンのサブプログラムの削除

スタンドアロンのサブプログラムを削除するには、SQL Developerの「接続」フレームと削除ツール、またはDDL文DROP PROCEDUREかDROP FUNCTIONのいずれかを使用します。

注意:

プロシージャ ADD_EVALUATION またはファンクション CALCULATE_SCORE は今後のチュートリアルで必要なため、削除しないでください。サブプログラムの削除の実習を行う場合は、簡単なサブプログラムを作成してから削除してください。

Dropツールを使用してスタンドアロンのサブプログラムを削除するには、次の操作を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「ファンクション」または「プロシージャ」を展開します。
3. ファンクションまたはプロシージャのリストで、削除するファンクションまたはプロシージャの名前を右クリックします。
4. 選択肢のリストで、「削除」をクリックします。
5. 「削除」ウィンドウで、「適用」をクリックします。
6. 「確認」ウィンドウで「OK」をクリックします。

関連項目:

- DROP PROCEDUREおよびDROP FUNCTION文に適用される一般情報は、「データ定義言語(DDL)文について」を参照してください。
- DROP PROCEDURE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。
- DROP FUNCTION文の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

親トピック: スタンドアロンのサブプログラムの作成および管理

5.6 パッケージの作成および管理

PL/SQLパッケージを作成および管理できます。

- パッケージ構造について
  パッケージには必ず仕様部があり、通常、さらに本体があります。仕様部はパッケージそのものを定義する、Application Program Interface (API)です。パッケージ本体では、宣言されたカーソルの問合せ、およびパッケージ仕様で宣言されているサブプログラムのコードを定義します。
- チュートリアル: パッケージ仕様部の作成
  このチュートリアルでは、パッケージの作成ツールを使用して、このドキュメントの多くのチュートリアルおよび例に示されているEMP_EVALというパッケージの仕様を作成する方法を示します。
- チュートリアル: パッケージ仕様部の変更
  このチュートリアルでは、編集ツールを使用して、このドキュメントの多くのチュートリアルおよび列に示されているEMP_EVALパッケージの仕様を変更する方法を示します。具体的には、このチュートリアルでは、プロシージャ、EVAL_DEPARTMENTおよび関数CALCULATE_SCOREの宣言を追加する方法を表示されます。
- チュートリアル: パッケージ本体の作成
  このチュートリアルでは、本体の作成ツールを使用して、このドキュメントの多くのチュートリアルおよび列に示されているEMP_EVALパッケージの本体を作成する方法を示します。
- パッケージの削除
  パッケージ(仕様および本体)を削除するには、SQL Developerの「接続」フレームと削除ツールまたはDDL文DROP

PACKAGEのいずれかを使用します。

関連項目:

パッケージ本体を変更する方法については、「チュートリアル: サブプログラムでの変数および定数の宣言」を参照してください。

親トピック: ストアド・サブプログラムおよびパッケージの開発

5.6.1 パッケージ構造について

パッケージには常に仕様があり、本体があります。仕様部はパッケージそのものを定義する、Application Program Interface (API)です。パッケージ本体では、宣言されたカーソルの問合せ、およびパッケージ仕様で宣言されているサブプログラムのコードを定義します。

パッケージ仕様は、パッケージを定義し、型、変数、定数、例外、宣言カーソル、およびパッケージ外部から参照される可能性のあるサブプログラムを宣言します。パッケージ仕様は**Application Program Interface (API)**です。クライアント・プログラムからサブプログラムを起動するために必要な情報はすべて含まれていますが、それらの実装に関する情報は含まれません。

パッケージ本体には、パッケージ仕様で宣言されている宣言カーソルやサブプログラムについて、対応する問合せやコードを定義します(そのため、宣言カーソルもサブプログラムもないパッケージについては本体は必要ありません)。また、パッケージ本体は、仕様部で宣言されずパッケージの他のサブプログラムでのみ起動できる**ローカル・サブプログラム**も定義できます。パッケージ本体の内容は、クライアント・プログラムに対して非表示です。パッケージ本体は、パッケージをコールするアプリケーションを無効にすることなく変更できます。

関連項目:

- パッケージ仕様部の詳細は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。
- パッケージ本体の詳細は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。

親トピック: パッケージの作成および管理

5.6.2 チュートリアル: パッケージ仕様部の作成

このチュートリアルでは、パッケージの作成ツールを使用して、このドキュメントの多くのチュートリアルおよび例に示されているEMP_EVALというパッケージの仕様を作成する方法について説明します。

パッケージ仕様部を作成するには、SQL Developerのパッケージの作成ツールまたはDDL文のCREATE PACKAGEを使用します。

パッケージの作成ツールを使用してパッケージ仕様部を作成するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「パッケージ」を右クリックします。
3. 選択肢のリストで、「新規パッケージ」をクリックします。

   「パッケージの作成」ウィンドウが開きます。「スキーマ」フィールドには値HRが、「名前」フィールドにはデフォルト値PACKAGE1が入り、新規ソースを小文字で追加チェック・ボックスは選択解除されています。
4. 「スキーマ」では、デフォルト値のHRを受け入れます。

5. 「名前」では、PACKAGE1をEMP_EVALに変更します。
6. 「OK」をクリックします。

   EMP_EVALペインが開き、パッケージを作成したCREATE PACKAGE文が表示されます。

```
CREATE OR REPLACE PACKAGE emp_eval AS

  /* TODO enter package declarations (types, exceptions, methods etc) here */

END emp_eval;
```

   ペインのタイトルがイタリック・フォントになっています。これは、パッケージがデータベースに保存されていないことを示しています。

7. (オプション)CREATE PACKAGE文で、コメントを宣言に置換します。

   「チュートリアル: パッケージ仕様部の変更」に示すとおり、今この手順を行わない場合、後で行うこともできます。

8. 「File」メニューから、「Save」を選択します。

   Oracle Databaseは、パッケージをコンパイルして保存します。EMP_EVALペインのタイトルがイタリック・フォントではなくなります。

関連項目:

CREATE PACKAGE文(パッケージ仕様部)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。

親トピック: パッケージの作成および管理

5.6.3 チュートリアル: パッケージ仕様部の変更

このチュートリアルでは、編集ツールを使用して、このドキュメントの多くのチュートリアルおよび列に示されているEMP_EVALパッケージの仕様を変更する方法について説明します。具体的には、このチュートリアルでは、プロシージャ、EVAL_DEPARTMENTおよび関数CALCULATE_SCOREの宣言を追加する方法を表示されます。

パッケージ仕様部を変更するには、SQL Developerの編集ツール、またはOR REPLACE句を持つCREATE PACKAGEDDL文を使用します。

編集ツールを使用してEMP_EVALパッケージ仕様部を変更するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「パッケージ」を展開します。
3. パッケージのリストで、EMP_EVALを右クリックします。
4. 選択肢のリストで、「編集」をクリックします。

   EMP_EVALペインが開き、パッケージを作成したCREATE PACKAGE文が表示されます。

```
CREATE OR REPLACE PACKAGE emp_eval AS

  /* TODO enter package declarations (types, exceptions, methods etc) here */

END emp_eval;
```

   ペインのタイトルがイタリック・フォントになっていません。これは、パッケージがデータベースに保存されていることを示しています。

5. EMP_EVALペインで、コメントを次のコードに置換します。

```
PROCEDURE eval_department ( dept_id IN NUMBER );

FUNCTION calculate_score ( evaluation_id IN NUMBER
                         , performance_id IN NUMBER)
                         RETURN NUMBER;
```

EMP_EVALペインのタイトルがイタリック・フォントに変更されます。その変更がまだデータベースに保存されていないことを示しています。

6. 「コンパイル」アイコンをクリックします。

変更されたパッケージ仕様部は、コンパイルされデータベースに保存されます。EMP_EVALペインのタイトルがイタリック・フォントではなくなります。

関連項目:

OR REPLACE句が指定されたCREATE 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。

親トピック: パッケージの作成および管理

5.6.4 チュートリアル: パッケージ本体の作成

このチュートリアルでは、本体の作成ツールを使用して、このドキュメントの多くのチュートリアルおよび列に示されているEMP_EVALパッケージの本体を作成する方法について説明します。

パッケージ本体を作成するには、SQL Developerの本体の作成ツールまたはDDL文のCREATE PACKAGE BODYを使用します。

本体の作成ツールを使用してEMP_EVALパッケージの本体を作成するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「パッケージ」を展開します。
3. パッケージのリストで、EMP_EVALを右クリックします。
4. 選択肢のリストで、「本体の作成」をクリックします。

EMP_EVAL本体ペインが表示され、パッケージ本体の自動生成されたコードが表示されます。

```
CREATE OR REPLACE
PACKAGE BODY EMP_EVAL AS

  PROCEDURE eval_department(dept_id IN NUMBER) AS
  BEGIN
    -- TODO implementation required for PROCEDURE EMP_EVAL.eval_department
    NULL;
  END eval_department;

  FUNCTION calculate_score ( evaluation_id IN NUMBER
                           , performance_id IN NUMBER)
                           RETURN NUMBER AS
  BEGIN
    -- TODO implementation required for FUNCTION EMP_EVAL.calculate_score
    RETURN NULL;
```

```
  END calculate_score;

END EMP_EVAL;
```

ペインのタイトルがイタリック・フォントになっています。コードがデータベースに保存されていないことを示しています。

5. (オプション)CREATE PACKAGE BODY文で、次の手順を実行します。
   - コメントを実行可能文に置換します。
   - (オプション)プロシージャの実行可能部分で、NULLを削除するか、または実行可能文に置換します。
   - (オプション)ファンクションの実行可能部分で、NULLを別の式に置換します。

   「チュートリアル: サブプログラムでの変数および定数の宣言」に示すとおり、今この手順を行わない場合、後で行うこともできます。

6. 「コンパイル」アイコンをクリックします。

   変更されたパッケージ本体は、コンパイルされデータベースに保存されます。EMP_EVAL本体ペインのタイトルがイタリック・フォントではなくなります。

関連項目:

CREATE PACKAGE BODY文(パッケージ本体)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。

親トピック: パッケージの作成および管理

5.6.5 パッケージの削除

パッケージ(仕様および本体)を削除するには、SQL Developerの「接続」フレームと削除ツールまたはDDL文DROP PACKAGEのいずれかを使用します。

> **注意:**
>
> パッケージ EMP_EVAL は今後のチュートリアルで必要なため、削除しないでください。パッケージの削除の実習を行う場合は、簡単なパッケージを作成してから削除してください。

削除ツールを使用してパッケージを削除するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「パッケージ」を展開します。

   パッケージのリストが表示されます。

3. パッケージのリストで、削除するパッケージの名前を右クリックします。
4. 選択肢のリストで、「パッケージの削除」をクリックします。
5. 「削除」ウィンドウで、「適用」をクリックします。
6. 「確認」ウィンドウで「OK」をクリックします。

関連項目:

DROP 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。

親トピック: パッケージの作成および管理

5.7 変数および定数の宣言と値の割当て

パッケージ仕様で宣言された変数または定数は、このパッケージにアクセスしている任意のプログラムで使用可能です。パッケージ本体またはサブプログラムで宣言された変数または定数は、そのパッケージまたはサブプログラムに対してローカルです。定数を宣言する場合、その定数に初期値を割り当てる必要があります。

PL/SQLがSQLより優れている点の1つは、PL/SQLでは変数および定数を宣言して使用できることです。

パッケージ仕様で宣言された変数または定数は、このパッケージにアクセスしている任意のプログラムで使用可能です。パッケージ本体またはサブプログラムで宣言された変数または定数は、そのパッケージまたはサブプログラムに対してローカルです。

変数には特定のデータ型の値が格納されます。ご使用のプログラムで、実行時に値を変更できます。 **定数**には、変更できない値が格納されます。

変数または定数には、任意のPL/SQLデータ型を指定できます。変数を宣言する際に初期値を割り当てることができ、割り当てない場合はこの値がNULLになります。定数を宣言する場合、その定数に初期値を割り当てる必要があります。初期値を変数または定数に割り当てるには、代入演算子(:=)を使用します。

ヒント:

変わらないすべての値を定数として宣言します。これによってコンパイル・コードが最適化され、ソース・コードがメンテナンスしやすくなります。

- チュートリアル: サブプログラムでの変数および定数の宣言
  このチュートリアルでは、SQL Developerの編集ツールを使用してEMP_EVAL.CALCULATE_SCOREファンクションで変数および制約を宣言する方法を示します。(このチュートリアルは、パッケージ本体を変更する例でもあります。)
- 変数、定数およびパラメータのデータ型が正しいことの確認
  変数、定数およびパラメータのデータ型が正しいことを確認するには、これらを%TYPE属性で宣言します。
- チュートリアル: %TYPE属性を使用するための宣言の変更
  このチュートリアルでは、SQL Developerツールの「編集」を使用して、EMP_EVAL.CALCULATE_SCOREファンクションの変数、定数および仮パラメータの宣言を変更して%TYPE属性を使用する方法を示します。
- 変数への値の代入

関連項目:

変数および定数の一般情報は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。

親トピック: ストアド・サブプログラムおよびパッケージの開発

5.7.1 チュートリアル: サブプログラムでの変数および定数の宣言

このチュートリアルでは、SQL Developerの編集ツールを使用してEMP_EVAL.CALCULATE_SCOREファンクションで変数および制約を宣言する方法を示します。(このチュートリアルは、パッケージ本体を変更する例でもあります。)

EMP_EVAL.CALCULATE_SCOREファンクションは、「チュートリアル: パッケージ仕様部の作成」で指定されています。

CALCULATE_SCORE関数の変数および定数を宣言するには、次のようにします。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「パッケージ」を展開します。
3. パッケージのリストで、EMP_EVALを展開します。
4. 選択肢のリストで、EMP_EVALの本体を右クリックします。

   選択肢のリストが表示されます。

5. 選択肢のリストで、「編集」をクリックします。

   EMP_EVAL本体ペインが表示され、パッケージ本体のコードが表示されます。

```
CREATE OR REPLACE
PACKAGE BODY EMP_EVAL AS

  PROCEDURE eval_department ( dept_id IN NUMBER ) AS

  BEGIN
    -- TODO implementation required for PROCEDURE EMP_EVAL.eval_department
    NULL;
  END eval_department;

  FUNCTION calculate_score ( evaluation_id IN NUMBER
                           , performance_id IN NUMBER)
                           RETURN NUMBER AS
  BEGIN
    -- TODO implementation required for FUNCTION EMP_EVAL.calculate_score
    RETURN NULL;
  END calculate_score;

END EMP_EVAL;
```

6. RETURN NUMBER ASとBEGINの間に、次の変数および定数の宣言を追加します。

```
n_score       NUMBER(1,0);                -- variable
n_weight      NUMBER;                     -- variable
max_score     CONSTANT NUMBER(1,0) := 9;  -- constant, initial value 9
max_weight    CONSTANT NUMBER(8,8) := 1;  -- constant, initial value 1
```

   EMP_EVAL本体ペインのタイトルがイタリック・フォントに変更されます。コードがまだデータベースに保存されていないことを示しています。

7. 「File」メニューから、「Save」を選択します。

   Oracle Databaseは、変更されたパッケージ本体をコンパイルおよび保存します。EMP_EVAL本体ペインのタイトルがイタリック・フォントではなくなります。

関連項目:

- 変数および定数の宣言の一般情報は、*『Oracle Database PL/SQL言語リファレンス』*を参照してください。

- "代入演算子を使用した変数への値の割当て

親トピック: 変数および定数の宣言と値の割当て

5.7.2 変数、定数およびパラメータのデータ型が正しいことの確認

変数、定数およびパラメータのデータ型が正しいことを確認するには、これらを%TYPE属性で宣言します。

「チュートリアル: サブプログラムでの変数および定数の宣言」の後のEMP_EVAL.CALCULATE_SCOREファンクションのコードは、次のとおりです。

```
FUNCTION calculate_score ( evaluation_id IN NUMBER
                         , performance_id IN NUMBER )
                         RETURN NUMBER AS
  n_score       NUMBER(1,0);                -- variable
  n_weight      NUMBER;                     -- variable
  max_score     CONSTANT NUMBER(1,0) := 9;  -- constant, initial value 9
  max_weight    CONSTANT NUMBER(8,8) := 1;  -- constant, initial value 1
  BEGIN
    -- TODO implementation required for FUNCTION EMP_EVAL.calculate_score
    RETURN NULL;
  END calculate_score;
```

ファンクションの変数、定数およびパラメータは、(「表の作成」で作成した)表SCORESおよびPERFORMANCE_PARTSの値を表します。

- 変数n_scoreは、SCORE.SCORES列の値を保持し、定数max_scoreは、その値と比較されます。
- 変数n_weightは、PERFORMANCE_PARTS.WEIGHT列の値を保持し、定数max_weightは、その値と比較されます。
- パラメータevaluation_idは、SCORE.EVALUATION_ID列の値を保持します。
- パラメータperformance_idは、SCORE.PERFORMANCE_ID列の値を保持します。

このため、各変数、定数およびパラメータのデータ型は、対応する列と同じです。

列のデータ型が変更されたら、変数、定数およびパラメータのデータ型も同じデータ型に変わる必要があります。そうでないと、CALCULATE_SCOREファンクションが無効になります。

変数、定数およびパラメータのデータ型が常に列のデータ型と一致することを確認するには、これらを%TYPE属性で宣言します。 %TYPE属性は、表の列または別の変数のデータ型を提供し、正しいデータ型の割当てを保証します。

関連項目:

- %TYPE属性の詳細は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。
- %TYPE属性の構文については、『Oracle Database PL/SQL言語リファレンス』を参照してください。

親トピック: 変数および定数の宣言と値の割当て

5.7.3 チュートリアル: %TYPE属性を使用するための宣言の変更

このチュートリアルでは、SQL Developerの編集ツールを使用して、EMP_EVAL.CALCULATE_SCOREファンクションの変

数、定数および仮パラメータの宣言を変更して%TYPE属性を使用する方法を示します。

EMP_EVAL.CALCULATE_SCOREファンクションは、「チュートリアル: サブプログラムでの変数および定数の宣言」で指定されています。

CALCULATE_SCOREの宣言を変更して%TYPEを使用するには、次のようにします。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「パッケージ」を展開します。
3. パッケージのリストで、EMP_EVALを展開します。
4. 選択肢のリストで、EMP_EVALの本体を右クリックします。
5. 選択肢のリストで、「編集」をクリックします。

   EMP_EVAL本体ペインが表示され、パッケージ本体のコードが表示されます。

```
CREATE OR REPLACE
PACKAGE BODY emp_eval AS

  PROCEDURE eval_department ( dept_id IN NUMBER ) AS
  BEGIN
    -- TODO implementation required for PROCEDURE EMP_EVAL.eval_department
    NULL;
  END eval_department;

  FUNCTION calculate_score ( evaluation_id IN NUMBER
                           , performance_id IN NUMBER )
                           RETURN NUMBER AS
  n_score       NUMBER(1,0);                -- variable
  n_weight      NUMBER;                     -- variable
  max_score     CONSTANT NUMBER(1,0) := 9;  -- constant, initial value 9
  max_weight    CONSTANT NUMBER(8,8) := 1;  -- constant, initial value 1
  BEGIN
    -- TODO implementation required for FUNCTION EMP_EVAL.calculate_score
    RETURN NULL;
  END calculate_score;

END emp_eval;
```

6. ファンクションのコードに、太字で示された変更を加えます。

```
  FUNCTION calculate_score ( evaluation_id IN SCORES.EVALUATION_ID%TYPE
                           , performance_id IN SCORES.PERFORMANCE_ID%TYPE)
                           RETURN NUMBER AS
  n_score       SCORES.SCORE%TYPE;
  n_weight      PERFORMANCE_PARTS.WEIGHT%TYPE;
  max_score     CONSTANT SCORES.SCORE%TYPE := 9;
  max_weight    CONSTANT PERFORMANCE_PARTS.WEIGHT%TYPE := 1;
```

7. EMP_EVALを右クリックします。
8. 選択肢のリストで、「編集」をクリックします。

   「EMP_EVAL」ペインが開き、パッケージを作成したCREATE PACKAGE文が表示されます。

```
CREATE OR REPLACE PACKAGE EMP_EVAL AS

PROCEDURE eval_department(dept_id IN NUMBER);
FUNCTION calculate_score(evaluation_id IN NUMBER
                       , performance_id IN NUMBER)
                         RETURN NUMBER;
```

```
END EMP_EVAL;
```

9. ファンクションのコードに、太字で示された変更を加えます。

```
FUNCTION calculate_score(evaluation_id IN scores.evaluation_id%TYPE
                         , performance_id IN scores.performance_id%TYPE)
```

10. EMP_EVALを右クリックします。
11. 選択肢のリストで、「コンパイル」をクリックします。
12. EMP_EVAL Bodyを右クリックします。
13. 選択肢のリストで、「コンパイル」をクリックします。

親トピック: 変数および定数の宣言と値の割当て

5.7.4 変数への値の代入

次の方法で変数に値を割り当てることができます。

- 代入演算子を使用して、式の値を割り当てます。
- SELECT INTO文またはFETCH文を使用して、表の値を割り当てます。
- OUTパラメータまたはIN OUTパラメータとしてサブプログラムに渡し、サブプログラム内で値を代入する方法。
- 変数を値にバインドします。

- 代入演算子を使用した変数への値の割当て
  代入演算子(:=)を使用して、サブプログラムの宣言部分または実行可能部分の変数に式の値を割り当てることができます。
- SELECT INTO文を使用した変数への値の割当て
  サブプログラムまたはパッケージの表の値を使用するには、SELECT INTO文によって変数に値を割り当てる必要があります。

関連項目:

- 変数への値の割当ての詳細は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。
- 変数のバインドの詳細は、『*Oracle Database 2日でJava開発者ガイド*』を参照してください。

親トピック: 変数および定数の宣言と値の割当て

5.7.4.1 代入演算子を使用した変数への値の割当て

代入演算子(:=)を使用して、サブプログラムの宣言部または実行可能部の変数に式の値を割り当てることができます。

サブプログラムの宣言部分では、宣言時に、変数に初期値を割り当てることができます。構文は次のとおりです。

```
variable_name data_type := expression;
```

サブプログラムの実行可能部分では、代入文によって変数に値を割り当てることができます。構文は次のとおりです。

```
variable_name := expression;
```

例5-1では、EMP_EVAL.CALCULATE_SCOREファンクションに対して行う変更が太字で表示されて、変数

running_totalが追加され、この新しい変数がファンクションの戻り値として使用されます。代入演算子は、ファンクションの宣言部と実行可能部の両方に表示されます。(running_totalのデータ型は、異なる精度およびスケールを持つ2つのNUMBER値の積を保持するため、SCORES.SCORE%TYPEまたはPERFORMANCE_PARTS.WEIGHT%TYPEではなく、NUMBERである必要があります。)

関連項目:

- 変数宣言の構文については、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。
- 代入文の構文については、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。

例5-1 代入演算子を使用した変数への値の割当て

```
FUNCTION calculate_score(evaluation_id IN SCORES.EVALUATION_ID%TYPE
                         , performance_id IN SCORES.PERFORMANCE_ID%TYPE)
                         RETURN NUMBER AS
  n_score       SCORES.SCORE%TYPE;
  n_weight      PERFORMANCE_PARTS.WEIGHT%TYPE;
  running_total NUMBER := 0;
  max_score     CONSTANT SCORES.SCORE%TYPE := 9;
  max_weight    CONSTANT PERFORMANCE_PARTS.WEIGHT%TYPE:= 1;
BEGIN
  running_total := max_score * max_weight;
  RETURN running_total;
END calculate_score;
```

親トピック: 変数への値の代入

5.7.4.2 SELECT INTO文を使用した変数への値の代入

サブプログラムまたはパッケージの表の値を使用するには、SELECT INTO文によって変数に値を割り当てる必要があります。

例5-2では、表の値からrunning_totalを計算させるためにEMP_EVAL.CALCULATE_SCOREファンクションに対して加える変更を太字で示しています。

例5-3に示すADD_EVALプロシージャは、EVALUATIONS表への行挿入に、EMPLOYEES表の対応する行の値を使用する場合の例です。ADD_EVALプロシージャは、EMP_EVALパッケージの本体にのみ追加し、仕様には追加しません。ADD_EVALは仕様内には定義しないので、そのパッケージのローカル・プロシージャになり、パッケージ内の他のサブプログラムからのみ起動でき、パッケージ外部からは起動できません。

関連項目:

SELECT INTO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5-2 SELECT INTOを使用した変数への表の値の割当て

```
FUNCTION calculate_score ( evaluation_id IN scores.evaluation_id%TYPE
                         , performance_id IN scores.performance_id%TYPE )
                         RETURN NUMBER AS

  n_score       scores.score%TYPE;
  n_weight      performance_parts.weight%TYPE;
  running_total NUMBER := 0;
  max_score     CONSTANT scores.score%TYPE := 9;
```

```
  max_weight    CONSTANT performance_parts.weight%TYPE:= 1;
BEGIN
  SELECT s.score INTO n_score
  FROM SCORES s
  WHERE evaluation_id = s.evaluation_id
  AND performance_id = s.performance_id;

  SELECT p.weight INTO n_weight
  FROM PERFORMANCE_PARTS p
  WHERE performance_id = p.performance_id;

  running_total := n_score * n_weight;
  RETURN running_total;
END calculate_score;
```

例5-3 他の表からの値を使用した表の行の挿入

```
PROCEDURE add_eval ( employee_id IN EMPLOYEES.EMPLOYEE_ID%TYPE
                   , today IN DATE )
AS
  job_id         EMPLOYEES.JOB_ID%TYPE;
  manager_id     EMPLOYEES.MANAGER_ID%TYPE;
  department_id  EMPLOYEES.DEPARTMENT_ID%TYPE;
BEGIN
  INSERT INTO EVALUATIONS (
    evaluation_id,
    employee_id,
    evaluation_date,
    job_id,
    manager_id,
    department_id,
    total_score
  )
  SELECT
    evaluations_sequence.NEXTVAL,    -- evaluation_id
    add_eval.employee_id,       -- employee_id
    add_eval.today,             -- evaluation_date
    e.job_id,                   -- job_id
    e.manager_id,               -- manager_id
    e.department_id,            -- department_id
    0                           -- total_score
  FROM employees e;

  IF SQL%ROWCOUNT = 0 THEN
    RAISE NO_DATA_FOUND;
  END IF;
END add_eval;
```

親トピック: 変数への値の代入

5.8 プログラム・フローの制御

入力順に実行する文であるSQLとは異なり、PL/SQLには、プログラムのフローを制御できる制御文があります。

- 制御文について
  PL/SQLには、条件付き選択文、繰り返し文および順次制御文の3つのカテゴリの制御文があります。
- IF文の使用
  IF文は、ブール式の値に応じて一連の文を実行またはスキップします。

- CASE文の使用
  CASE文は、一連の条件から選択し、対応する文を実行します。
- FOR LOOP文の使用
  FOR LOOP文は、lower_boundからupper_boundまでの範囲の各整数に対して1回ずつ、一連の文を繰り返します。
- WHILE LOOP文の使用
  WHILE LOOP文は、条件がTRUEであるかぎり一連の文を繰り返します。
- 基本のLOOPおよびEXIT WHEN文の使用
  基本のLOOP文は、一連の文を繰り返します。

親トピック: ストアド・サブプログラムおよびパッケージの開発

5.8.1 制御文について

PL/SQLには、条件付き選択文、繰り返し文および順次制御文の3つのカテゴリの制御文があります。

条件選択文では、異なるデータ値に対して異なる文を実行できます。条件選択文は、IFおよびCASEです。

繰り返し文では、一連の異なるデータ値で同じ文を繰り返すことができます。繰り返し文は、FOR LOOP、WHILE LOOP,および基本のLOOPです。EXIT文は、制御をループの終わりに転送します。 CONTINUE文は、現在のループの反復を終了し、制御を次の反復に転送します。 EXITおよびCONTINUEには、オプションのWHEN句があり、条件を指定できます。

順次制御文では、指定されたラベル付き文に移動するか、または何も処理をしません。順次制御文は、GOTOおよびNULLです。

関連項目:

PL/SQL制御文の概要は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。

親トピック: プログラム・フローの制御

5.8.2 IF文の使用

IF文は、ブール式の値に応じて一連の文を実行またはスキップします。

IF文の構文は、次のとおりです。

```
IF boolean_expression THEN statement [, statement ]
[ ELSIF boolean_expression THEN statement [, statement ] ]...
[ ELSE  statement [, statement ] ]
END IF;
```

企業が雇用の最初の10年は1年に2回、その後は1年に1回のみ、従業員を評価するとします。これには従業員の評価頻度を戻すファンクションが必要です。この場合、例5-4のように、IF文を使用してファンクションの戻り値を判断できます。

EVAL_FREQUENCY関数をEMP_EVALパッケージの本体に追加しますが、仕様には追加しません。
EVAL_FREQUENCYは、仕様にはないため、パッケージに対してローカルであり、パッケージ内の他のサブプログラムでのみ起動でき、パッケージ外からは起動できません。

ヒント:

SQL 文で PL/SQL 変数を使用する場合、例 5-4 の 2 つ目の SELECT 文に示すとおり、変数をサブプログラム名で修飾して、表の列と間違えないようにします。

関連項目:

- IF文の構文については、『Oracle Database PL/SQL言語リファレンス』を参照してください。
- IF文の使用の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください。

例5-4 ファンクションの戻り値を判断するIF文

```
FUNCTION eval_frequency (emp_id IN EMPLOYEES.EMPLOYEE_ID%TYPE)
  RETURN PLS_INTEGER
AS
  h_date     EMPLOYEES.HIRE_DATE%TYPE;
  today      EMPLOYEES.HIRE_DATE%TYPE;
  eval_freq  PLS_INTEGER;
BEGIN
  SELECT SYSDATE INTO today FROM DUAL;

  SELECT HIRE_DATE INTO h_date
  FROM EMPLOYEES
  WHERE EMPLOYEE_ID = eval_frequency.emp_id;

  IF ((h_date + (INTERVAL '120' MONTH)) < today) THEN
    eval_freq := 1;
  ELSE
    eval_freq := 2;
  END IF;

  RETURN eval_freq;
END eval_frequency;
```

親トピック: プログラム・フローの制御

5.8.3 CASE文の使用

CASE文は、一連の条件から選択し、対応する文を実行します。

単純なCASE文は、単一の式を評価して、可能性のある複数の値と比較します。構文は次のとおりです。

```
CASE expression
WHEN value THEN statement
[ WHEN value THEN statement ]...
[ ELSE statement [, statement ]... ]
END CASE;
```

検索CASE文は、複数のブール式を評価して、値がTRUEである最初の式を選択します。検索CASE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。

ヒント:

CASE 文またはネストされた IF 文のいずれも使用できる場合、CASE 文を使用すると、より読みやすく効率

的です。

従業員が1年に1回のみ評価され、JOB_IDに応じて昇給を提案するEVAL_FREQUENCY関数が必要だと仮定します。

例5-5に太字で示されているように、EVAL_FREQUENCYファンクションを変更します。(文字列を出力するプロシージャDBMS_OUTPUT.PUT_LINE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。)

例5-5 出力する文字列を判断するCASE文

```
FUNCTION eval_frequency (emp_id IN EMPLOYEES.EMPLOYEE_ID%TYPE)
  RETURN PLS_INTEGER
AS
  h_date     EMPLOYEES.HIRE_DATE%TYPE;
  today      EMPLOYEES.HIRE_DATE%TYPE;
  eval_freq  PLS_INTEGER;
  j_id       EMPLOYEES.JOB_ID%TYPE;

BEGIN
  SELECT SYSDATE INTO today FROM DUAL;

  SELECT HIRE_DATE, JOB_ID INTO h_date, j_id
  FROM EMPLOYEES
  WHERE EMPLOYEE_ID = eval_frequency.emp_id;

  IF ((h_date + (INTERVAL '12' MONTH)) < today) THEN
    eval_freq := 1;

    CASE j_id
       WHEN 'PU_CLERK' THEN DBMS_OUTPUT.PUT_LINE(
         'Consider 8% salary increase for employee # ' || emp_id);
       WHEN 'SH_CLERK' THEN DBMS_OUTPUT.PUT_LINE(
         'Consider 7% salary increase for employee # ' || emp_id);
       WHEN 'ST_CLERK' THEN DBMS_OUTPUT.PUT_LINE(
         'Consider 6% salary increase for employee # ' || emp_id);
       WHEN 'HR_REP' THEN DBMS_OUTPUT.PUT_LINE(
         'Consider 5% salary increase for employee # ' || emp_id);
       WHEN 'PR_REP' THEN DBMS_OUTPUT.PUT_LINE(
         'Consider 5% salary increase for employee # ' || emp_id);
       WHEN 'MK_REP' THEN DBMS_OUTPUT.PUT_LINE(
         'Consider 4% salary increase for employee # ' || emp_id);
       ELSE DBMS_OUTPUT.PUT_LINE(
         'Nothing to do for employee #' || emp_id);
    END CASE;
  ELSE
    eval_freq := 2;
  END IF;

  RETURN eval_freq;
END eval_frequency;
```

関連項目:

- 「問合せにおけるCASE式の使用」
- CASE文の構文については、『Oracle Database PL/SQL言語リファレンス』を参照してください。

- CASE文の使用の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください。

親トピック: プログラム・フローの制御

5.8.4 FOR LOOP文の使用

FOR LOOP文は、lower_boundからupper_boundまでの範囲の各整数に対して1回ずつ、一連の文を繰り返します。

FOR LOOPの構文は次のとおりです。

```
FOR counter IN lower_bound..upper_bound LOOP
  statement [, statement ]...
END LOOP;
```

LOOPとEND LOOP間の文では、counterを使用できますが、値は変更できません。

給与の値上げを想定するだけでなく、EVAL_FREQUENCYファンクションを使用して、5年間で毎年推定額が増加した場合に給与がどう変わるかをレポートするとします。

例5-6に太字で示されているように、EVAL_FREQUENCYファンクションを変更します。(文字列DBMS_OUTPUT.PUT_LINEを出力する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。)

例5-6 5年後の給与を計算するFOR LOOP文

```
FUNCTION eval_frequency (emp_id IN EMPLOYEES.EMPLOYEE_ID%TYPE)
  RETURN PLS_INTEGER
AS
  h_date      EMPLOYEES.HIRE_DATE%TYPE;
  today       EMPLOYEES.HIRE_DATE%TYPE;
  eval_freq   PLS_INTEGER;
  j_id        EMPLOYEES.JOB_ID%TYPE;
  sal         EMPLOYEES.SALARY%TYPE;
  sal_raise   NUMBER(3,3) := 0;

BEGIN
  SELECT SYSDATE INTO today FROM DUAL;

  SELECT HIRE_DATE, JOB_ID, SALARY INTO h_date, j_id, sal
  FROM EMPLOYEES
  WHERE EMPLOYEE_ID = eval_frequency.emp_id;

  IF ((h_date + (INTERVAL '12' MONTH)) < today) THEN
    eval_freq := 1;

    CASE j_id
      WHEN 'PU_CLERK' THEN sal_raise := 0.08;
      WHEN 'SH_CLERK' THEN sal_raise := 0.07;
      WHEN 'ST_CLERK' THEN sal_raise := 0.06;
      WHEN 'HR_REP'   THEN sal_raise := 0.05;
      WHEN 'PR_REP'   THEN sal_raise := 0.05;
      WHEN 'MK_REP'   THEN sal_raise := 0.04;
      ELSE NULL;
    END CASE;

    IF (sal_raise != 0) THEN
      BEGIN
        DBMS_OUTPUT.PUT_LINE('If salary ' || sal || ' increases by ' ||
          ROUND((sal_raise * 100),0) ||
```

```
          '% each year for 5 years, it will be:');

        FOR i IN 1..5 LOOP
          sal := sal * (1 + sal_raise);
          DBMS_OUTPUT.PUT_LINE(ROUND(sal, 2) || ' after ' || i || ' year(s)');
        END LOOP;
      END;
    END IF;

  ELSE
    eval_freq := 2;
  END IF;

  RETURN eval_freq;
END eval_frequency;
```

関連項目:

- FOR LOOP文の構文については、『Oracle Database PL/SQL言語リファレンス』を参照してください。
- FOR LOOP文の使用の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください。

親トピック: プログラム・フローの制御

5.8.5 WHILE LOOP文の使用

WHILE LOOP文は、条件がTRUEであるかぎり一連の文を繰り返します。

WHILE LOOP文の構文は次のとおりです。

```
WHILE condition LOOP
  statement [, statement ]...
END LOOP;
```

注意:

LOOP と END LOOP の間の文によって condition が FALSE にならない場合、WHILE LOOP 文は無限に実行され続けます。

EVAL_FREQUENCYファンクションでFOR LOOP文ではなくWHILE LOOP文を使用して、推奨した給与がJOB_IDの最大給与を超過したときに停止するようにするとします。

例5-7に太字で示されているように、EVAL_FREQUENCYファンクションを変更します。(文字列を出力するプロシージャDBMS_OUTPUT.PUT_LINE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。)

例5-7 最大値まで給与を計算するWHILE LOOP文

```
FUNCTION eval_frequency (emp_id IN EMPLOYEES.EMPLOYEE_ID%TYPE)
  RETURN PLS_INTEGER
AS
```

```
  h_date      EMPLOYEES.HIRE_DATE%TYPE;
  today       EMPLOYEES.HIRE_DATE%TYPE;
  eval_freq   PLS_INTEGER;
  j_id        EMPLOYEES.JOB_ID%TYPE;
  sal         EMPLOYEES.SALARY%TYPE;
  sal_raise   NUMBER(3,3) := 0;
  sal_max     JOBS.MAX_SALARY%TYPE;

BEGIN
  SELECT SYSDATE INTO today FROM DUAL;

  SELECT HIRE_DATE, j.JOB_ID, SALARY, MAX_SALARY INTO h_date, j_id, sal, sal_max
  FROM EMPLOYEES e, JOBS j
  WHERE EMPLOYEE_ID = eval_frequency.emp_id AND JOB_ID = eval_frequency.j_id;

  IF ((h_date + (INTERVAL '12' MONTH)) < today) THEN
    eval_freq := 1;

    CASE j_id
      WHEN 'PU_CLERK' THEN sal_raise := 0.08;
      WHEN 'SH_CLERK' THEN sal_raise := 0.07;
      WHEN 'ST_CLERK' THEN sal_raise := 0.06;
      WHEN 'HR_REP'   THEN sal_raise := 0.05;
      WHEN 'PR_REP'   THEN sal_raise := 0.05;
      WHEN 'MK_REP'   THEN sal_raise := 0.04;
      ELSE NULL;
    END CASE;

    IF (sal_raise != 0) THEN
      BEGIN
        DBMS_OUTPUT.PUT_LINE('If salary ' || sal || ' increases by ' ||
          ROUND((sal_raise * 100),0) ||
          '% each year, it will be:');

        WHILE sal <= sal_max LOOP
          sal := sal * (1 + sal_raise);
          DBMS_OUTPUT.PUT_LINE(ROUND(sal, 2));
        END LOOP;

        DBMS_OUTPUT.PUT_LINE('Maximum salary for this job is ' || sal_max);
      END;
    END IF;
  ELSE
    eval_freq := 2;
  END IF;

  RETURN eval_freq;
END eval_frequency;
```

関連項目:

- WHILE LOOP文の構文については、『Oracle Database PL/SQL言語リファレンス』を参照してください。
- WHILE LOOP文の使用の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください。

親トピック: プログラム・フローの制御

5.8.6 基本のLOOPおよびEXIT WHEN文の使用

基本のLOOP文は、一連の文を繰り返します。

基本のLOOP文の構文は次のとおりです。

```
LOOP
  statement [, statement ]...
END LOOP;
```

少なくとも1つの文は、EXIT文である必要があります。そうでない場合、LOOP文は無限に実行され続けます。

EXIT WHEN文(オプションのWHEN句を持つEXIT文)は、条件がTRUEのときにループを終了し、制御をループの終わりに転送します。

EVAL_FREQUENCYファンクションでは、WHILE LOOP文の最後の反復で、通常、最後に計算された値が最大給与を超過します。

例5-8に示すように、WHILE LOOP文を、EXIT WHEN文を含む基本のLOOP文に変更します。

例5-8 EXIT WHEN文の使用

```
FUNCTION eval_frequency (emp_id IN EMPLOYEES.EMPLOYEE_ID%TYPE)
  RETURN PLS_INTEGER
AS
  h_date      EMPLOYEES.HIRE_DATE%TYPE;
  today       EMPLOYEES.HIRE_DATE%TYPE;
  eval_freq   PLS_INTEGER;
  j_id        EMPLOYEES.JOB_ID%TYPE;
  sal         EMPLOYEES.SALARY%TYPE;
  sal_raise   NUMBER(3,3) := 0;
  sal_max     JOBS.MAX_SALARY%TYPE;

BEGIN
  SELECT SYSDATE INTO today FROM DUAL;

  SELECT HIRE_DATE, j.JOB_ID, SALARY, MAX_SALARY INTO h_date, j_id, sal, sal_max
  FROM EMPLOYEES e, JOBS j
  WHERE EMPLOYEE_ID = eval_frequency.emp_id AND JOB_ID = eval_frequency.j_id;

  IF ((h_date + (INTERVAL '12' MONTH)) < today) THEN
    eval_freq := 1;

    CASE j_id
      WHEN 'PU_CLERK' THEN sal_raise := 0.08;
      WHEN 'SH_CLERK' THEN sal_raise := 0.07;
      WHEN 'ST_CLERK' THEN sal_raise := 0.06;
      WHEN 'HR_REP'   THEN sal_raise := 0.05;
      WHEN 'PR_REP'   THEN sal_raise := 0.05;
      WHEN 'MK_REP'   THEN sal_raise := 0.04;
      ELSE NULL;
    END CASE;

    IF (sal_raise != 0) THEN
      BEGIN
        DBMS_OUTPUT.PUT_LINE('If salary ' || sal || ' increases by ' ||
          ROUND((sal_raise * 100),0) ||
          '% each year, it will be:');

        LOOP
```

```
          sal := sal * (1 + sal_raise);
          EXIT WHEN sal > sal_max;
          DBMS_OUTPUT.PUT_LINE(ROUND(sal,2));
        END LOOP;

        DBMS_OUTPUT.PUT_LINE('Maximum salary for this job is ' || sal_max);
      END;
    END IF;
  ELSE
    eval_freq := 2;
  END IF;

  RETURN eval_freq;
END eval_frequency;
```

関連項目:

- LOOP文の構文については、『Oracle Database PL/SQL言語リファレンス』を参照してください。
- EXIT文の構文については、『Oracle Database PL/SQL言語リファレンス』を参照してください。
- LOOP文およびEXIT文の使用の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください。

親トピック: プログラム・フローの制御

5.9 レコードおよびカーソルの使用

レコードにデータ値を格納し、カーソルを結果セットおよび関連の処理情報へのポインタとして使用できます。

- レコードについて
  レコードは、様々な型のデータ値を格納できるPL/SQLコンポジット変数です。内部コンポーネント(**フィールド**)はスカラー変数と同様に処理できます。サブプログラム・パラメータとしてレコード全体を渡すことができます。レコードは、表の行からのデータ、または表の行の特定列からのデータを格納するのに便利です。
- チュートリアル: RECORD型の宣言
  このチュートリアルでは、SQL Developerの編集ツールを使用してRECORD型のsal_infoを宣言する方法を示します。フィールドには、ジョブID、そのジョブIDの給与の最小値および最大値、現在の給与、推奨される値上げなどの、従業員の給与情報を格納できます。
- チュートリアル: レコード・パラメータによるサブプログラムの作成および起動
  このチュートリアルでは、SQL Developerの編集ツールを使用してレコード型sal_infoのパラメータでサブプログラムを作成および起動する方法を示します。
- カーソルについて
  Oracle Databaseは、SQL文の実行時に、結果セットおよび処理情報を無名のプライベートSQL領域に保存します。この名前のない領域へのポインタは、**カーソル**と呼ばれ、これを使用して結果セットを1行ずつ取得することができます。 **カーソル属性**は、カーソルの状態に関する情報を戻します。
- チュートリアル: 宣言カーソルを使用して結果セットの行を1行ずつ取得
  宣言カーソルを使用して結果セットの行を1行ずつ取得できます。
- チュートリアル: 宣言カーソルを使用して結果セットの行を1行ずつ取得
  このチュートリアルでは、宣言カーソルemp_cursorを使用するプロシージャEMP_EVAL.EVAL_DEPARTMENTの実装方法を示します。

- カーソル変数について
  カーソル変数は、問合せを1つに制限しないカーソルと似ています。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。カーソル変数は、サブプログラム間で問合せ結果を渡すときに有用です。
- 結果セットの行を1行ずつ取得するためのカーソル変数の使用
  カーソル変数を使用して結果セットの行を1行ずつ取得できます。
- チュートリアル: 結果セット行を1つずつ取得するためのカーソル変数の使用
  このチュートリアルでは、EMP_EVAL.EVAL_DEPARTMENTプロシージャを変更して、宣言カーソル(複数の部門を処理できる)のかわりにカーソル変数を使用する方法と、EMP_EVAL.EVAL_DEPARTMENTおよびEMP_EVAL.ADD_EVALのより効率的な使用方法を示します。

関連項目:

レコードの詳細は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。

親トピック: ストアド・サブプログラムおよびパッケージの開発

5.9.1 レコードについて

レコードは、様々な型のデータ値を格納できるPL/SQLコンポジット変数です。内部コンポーネント(**フィールド**)はスカラー変数と同様に処理できます。サブプログラム・パラメータとしてレコード全体を渡すことができます。レコードは、表の行からのデータ、または表の行の特定列からのデータを格納するのに便利です。

レコードは、C、C++、Javaなどのstruct型に似た、異なる型のデータ値を格納できるPL/SQLの複合変数です。レコードの内部コンポーネントは、**フィールド**と呼ばれます。レコード・フィールドにアクセスするには、**ドット表記法** record_name.field_nameを使用します。

レコード・フィールドは、スカラー変数のように扱うことができます。サブプログラム・パラメータとしてレコード全体を渡すこともできます。

レコードは、表の行からのデータ、または表の行の特定列からのデータを格納するのに便利です。各レコード・フィールドは表の列に対応しています。

レコードを作成するには、3つの方法があります。

- **レコード**型を宣言し、その型の変数を宣言する。

  構文は次のとおりです。

```
TYPE record_name IS RECORD
  ( field_name data_type [:= initial_value]
 [, field_name data_type [:= initial_value ] ]... );

variable_name record_name;
```

- table_name%ROWTYPE型の変数を宣言します。

  レコードのフィールドの名前およびデータ型は、表の列と同じです。

- cursor_name%ROWTYPE型の変数を宣言します。

  レコードのフィールドの名前およびデータ型は、カーソルSELECT文のFROM句の表の列と同じです。

関連項目:

- RECORD型の定義およびその型のレコードの宣言の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください。
- RECORD型の定義の構文については、『Oracle Database PL/SQL言語リファレンス』を参照してください。
- %ROWTYPE属性の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください。
- %ROWTYPE属性の構文については、『Oracle Database PL/SQL言語リファレンス』を参照してください。

親トピック: レコードおよびカーソルの使用

5.9.2 チュートリアル: RECORD型の宣言

このチュートリアルでは、SQL Developerの編集ツールを使用してsal_infoというRECORD型を宣言する方法について説明します。この型には、各従業員の給与情報(ジョブID、最小および最大給与、現在の給与、推奨される増加額など)を保持するためのフィールドを指定します。

RECORD型sal_infoを宣言するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。

   hr_connアイコンの下に、スキーマ・オブジェクト型のリストが表示されます。

2. 「パッケージ」を展開します。

   パッケージのリストが表示されます。

3. EMP_EVALを右クリックします。

   選択肢のリストが表示されます。

4. 「編集」をクリックします。

   EMP_EVALペインが開き、パッケージを作成したCREATE PACKAGE文が表示されます。

```
CREATE OR REPLACE PACKAGE EMP_EVAL AS

PROCEDURE eval_department(dept_id IN NUMBER);
FUNCTION calculate_score(evaluation_id IN NUMBER
                         , performance_id IN NUMBER)
                           RETURN NUMBER;

END EMP_EVAL;
```

5. EMP_EVALペインで、END EMP_EVALの直前に次のコードを追加します。

```
TYPE sal_info IS RECORD
  ( j_id     jobs.job_id%type
  , sal_min  jobs.min_salary%type
  , sal_max  jobs.max_salary%type
  , sal      employees.salary%type
  , sal_raise NUMBER(3,3) );
```

   ペインのタイトルがイタリック・フォントになっています。これは、変更がデータベースに保存されていないことを示しています。

6. 「コンパイル」アイコンをクリックします。

変更されたパッケージ仕様部は、コンパイルされデータベースに保存されます。EMP_EVALペインのタイトルがイタリック・フォントではなくなります。

これで、「チュートリアル: レコード・パラメータによるサブプログラムの作成および起動」に示すようにsal_info型のレコードを宣言できます。

親トピック: レコードおよびカーソルの使用

5.9.3 チュートリアル: レコード・パラメータによるサブプログラムの作成および起動

このチュートリアルでは、SQL Developerの編集ツールを使用してレコード型sal_infoのパラメータでサブプログラムを作成および起動する方法を示します。

レコード型sal_infoは、「チュートリアル: RECORD型の宣言」で作成しました。

このチュートリアルでは、SQL Developerの編集ツールを使用して次を実行する方法を示します。

- 型sal_infoのパラメータを持つプロシージャSALARY_SCHEDULEを作成します。
- EVAL_FREQUENCYファンクションを変更して、レコードemp_salとその型sal_infoを宣言し、このフィールドを移入して、SALARY_SCHEDULEプロシージャに渡します。

SALARY_SCHEDULEはEVAL_FREQUENCYによって起動されるため、SALARY_SCHEDULEの宣言は、EVAL_FREQUENCYの宣言の前に行う必要があります(そうでない場合、パッケージはコンパイルしません)。ただし、SALARY_SCHEDULEの宣言は、パッケージ本体のどこで実行しても構いません。

SALARY_SCHEDULEを作成して、EVAL_FREQUENCYを変更するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「パッケージ」を展開します。
3. パッケージのリストで、EMP_EVALを展開します。
4. 選択肢のリストで、EMP_EVALの本体を右クリックします。
5. 選択肢のリストで、「編集」をクリックします。

   EMP_EVAL本体ペインが表示され、パッケージ本体のコードが表示されます。

6. EMP_EVAL本体ペインで、END EMP_EVALの直前にSALARY_SCHEDULEプロシージャのこの定義を追加します。

```
PROCEDURE salary_schedule (emp IN sal_info) AS
  accumulating_sal  NUMBER;
BEGIN
  DBMS_OUTPUT.PUT_LINE('If salary ' || emp.sal ||
    ' increases by ' || ROUND((emp.sal_raise * 100),0) ||
    '% each year, it will be:');

  accumulating_sal := emp.sal;

  WHILE accumulating_sal <= emp.sal_max LOOP
    accumulating_sal := accumulating_sal * (1 + emp.sal_raise);
    DBMS_OUTPUT.PUT_LINE(ROUND(accumulating_sal,2) ||', ');
  END LOOP;
END salary_schedule;
```

   ペインのタイトルがイタリック・フォントになっています。これは、変更がデータベースに保存されていないことを示しています。

7. EMP_EVAL本体ペインで、太字で示されたコードを次の位置に入力します。

```
CREATE OR REPLACE
PACKAGE BODY EMP_EVAL AS
```

```
FUNCTION eval_frequency (emp_id EMPLOYEES.EMPLOYEE_ID%TYPE)
  RETURN PLS_INTEGER;
PROCEDURE salary_schedule(emp IN sal_info);
PROCEDURE add_eval(employee_id IN employees.employee_id%type, today IN DATE);

PROCEDURE eval_department (dept_id IN NUMBER) AS
```

8. EVAL_FREQUENCYファンクションを編集し、太字フォントで示されている箇所を変更します。

```
FUNCTION eval_frequency (emp_id EMPLOYEES.EMPLOYEE_ID%TYPE)
  RETURN PLS_INTEGER
AS
  h_date     EMPLOYEES.HIRE_DATE%TYPE;
  today      EMPLOYEES.HIRE_DATE%TYPE;
  eval_freq  PLS_INTEGER;
  emp_sal    SAL_INFO;  -- replaces sal, sal_raise, and sal_max

BEGIN
  SELECT SYSDATE INTO today FROM DUAL;

  SELECT HIRE_DATE INTO h_date
  FROM EMPLOYEES
  WHERE EMPLOYEE_ID = eval_frequency.emp_id;

  IF ((h_date + (INTERVAL '120' MONTH)) < today) THEN
     eval_freq := 1;

     /* populate emp_sal */

     SELECT j.JOB_ID, j.MIN_SALARY, j.MAX_SALARY, e.SALARY
     INTO emp_sal.j_id, emp_sal.sal_min, emp_sal.sal_max, emp_sal.sal
     FROM EMPLOYEES e, JOBS j
     WHERE e.EMPLOYEE_ID = eval_frequency.emp_id
     AND j.JOB_ID = eval_frequency.emp_id;

     emp_sal.sal_raise := 0;  -- default

     CASE emp_sal.j_id
       WHEN 'PU_CLERK' THEN emp_sal.sal_raise := 0.08;
       WHEN 'SH_CLERK' THEN emp_sal.sal_raise := 0.07;
       WHEN 'ST_CLERK' THEN emp_sal.sal_raise := 0.06;
       WHEN 'HR_REP' THEN emp_sal.sal_raise := 0.05;
       WHEN 'PR_REP' THEN emp_sal.sal_raise := 0.05;
       WHEN 'MK_REP' THEN emp_sal.sal_raise := 0.04;
       ELSE NULL;
     END CASE;

     IF (emp_sal.sal_raise != 0) THEN
       salary_schedule(emp_sal);
     END IF;
   ELSE
     eval_freq := 2;
   END IF;

   RETURN eval_freq;
 END eval_frequency;
```

9. 「コンパイル」をクリックします。

5.9.4 カーソルについて

Oracle DatabaseによりSQL文が実行される場合、結果セットおよび処理情報は、名前が指定されていないプライベートSQL領域に格納されます。この名前のない領域へのポインタは、**カーソル**と呼ばれ、これを使用して結果セットを1行ずつ取得することができます。 **カーソル属性**は、カーソルの状態に関する情報を戻します。

SQL DML文またはPL/SQL SELECT INTO文を実行するたびに、PL/SQLは**暗黙カーソル**をオープンします。このカーソルに関する情報は属性から得られますが、制御はできません。文の実行後、データベースはカーソルをクローズしますが、属性の値は別のDMLまたはSELECT INTO文が実行されるまで使用可能です。

PL/SQLを使用すれば、カーソルも宣言できます。**宣言カーソル**には名前があり、通常、複数行が返される問合せ(SQL SELECT文)に関連付けられています。カーソルを宣言した後、暗黙的または明示的に処理する必要があります。カーソルを暗黙的に処理するには、カーソルFOR LOOPを使用します。構文は次のとおりです。

```
FOR record_name IN cursor_name LOOP
  statement
  [ statement ]...
END LOOP;
```

カーソルを明示的に処理するには、カーソルを開き(OPEN文)、結果セットから1行ずつまたは一括して行をフェッチし(FETCH文)、カーソルを閉じます(CLOSE文)。カーソルのクローズ後は、結果セットからレコードをフェッチしたり、カーソル属性の値を参照することはできません。

暗黙カーソルの属性値の構文は、SQL%属性です(たとえば、SQL%FOUND)。SQL%属性は常に、最後に実行されたDMLまたはSELECT INTO文を参照します。

宣言カーソルの属性値の構文は、cursor_name%属性です(たとえば、c1%FOUND)。

表5-1に、カーソル属性および戻すことのできる値のリストを示します。(暗黙カーソルには、このマニュアルの範囲外の追加属性があります。)

表5-1 カーソル属性の値

属性	宣言カーソルの値	暗黙カーソルの値
%FOUND	カーソルは開くが脚注 1フェッチが試行されない場合は、NULL です。 最後のフェッチが行を戻した場合、TRUE。 最後のフェッチが行を戻さなかった場合、FALSE。	DML または SELECT INTO 文が実行されていない場合、NULL。 最後の DML または SELECT INTO 文が行を戻した場合、TRUE。 最後の DML または SELECT INTO 文が行を戻さなかった場合、FALSE。
%NOTFOUND	カーソルは開くが脚注 1フェッチが試行されない場合は、NULL です。 最後のフェッチが行を戻した場合、FALSE。	DML または SELECT INTO 文が実行されていない場合、NULL。 最後の DML または SELECT INTO 文が行を戻した場合、FALSE。

属性	宣言カーソルの値	暗黙カーソルの値
	最後のフェッチが行を戻さなかった場合、TRUE。	最後の DML または SELECT INTO 文が行を戻さなかった場合、TRUE。
%ROWCOUNT	カーソルが開く場合脚注 1、0 以上の数字です。	DML または SELECT INTO 文が実行されていない場合、NULL。それ以外の場合は、0 以上の数字。
%ISOPEN	カーソルがオープンされている場合、TRUE。オープンされていない場合、FALSE。	常に FALSE。

脚注1

カーソルが開いていない場合、属性は事前定義済の例外INVALID_CURSORを発生します。

関連項目:

- 「問合せについて」
- 「データ操作言語(DML)文について」
- SELECT INTO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。
- PL/SQLでのカーソルの管理の詳細は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。

親トピック: レコードおよびカーソルの使用

5.9.5 宣言カーソルを使用して結果セットの行を1行ずつ取得

宣言カーソルを使用して結果セットの行を1行ずつ取得できます。

次の手順では、最も単純な形式で必要な各文を使用し、複雑な構文への参照も提示します。

宣言カーソルを使用して、結果セットの行を1行ずつ取得するには、次の手順を実行します。

1. 宣言部分で、次の手順を実行します。
    a. カーソルを宣言します。

    ```
    CURSOR cursor_name IS query;
    ```

    宣言カーソルの宣言構文の詳細は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。
    b. カーソルによって戻された行を格納するレコードを宣言します。

    ```
    record_name cursor_name%ROWTYPE;
    ```

    %ROWTYPE構文の詳細は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。
2. 実行可能部分で、次の手順を実行します。
    a. カーソルをオープンします。

    ```
    OPEN cursor_name;
    ```

OPEN文の構文の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください。

b. 次に、似た構文を持つLOOP文を使用して、カーソルから行(結果セットからの行)を1つずつフェッチします。

```
LOOP
  FETCH cursor_name INTO record_name;
  EXIT WHEN cursor_name%NOTFOUND;
  -- Process row that is in record_name:
  statement;
  [ statement; ]...
END LOOP;
```

FETCH文の構文の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください。

c. カーソルをクローズします。

```
CLOSE cursor_name;
```

CLOSE文の構文の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください。

親トピック: レコードおよびカーソルの使用

5.9.6 チュートリアル: 宣言カーソルを使用して結果セットの行を1行ずつ取得

このチュートリアルでは、宣言カーソルemp_cursorを使用するプロシージャEMP_EVAL.EVAL_DEPARTMENTの実装方法を示します。

EMP_EVAL.EVAL_DEPARTMENTプロシージャを実装するには、次の手順を実行します。

1. EMP_EVALパッケージ仕様部で、太字で示されているように、EVAL_DEPARTMENTプロシージャの宣言を変更します。

```
PROCEDURE eval_department(dept_id IN employees.department_id%TYPE);
```

2. EMP_EVALパッケージ本体で、太字で示されているように、EVAL_DEPARTMENTプロシージャの定義を変更します。

```
PROCEDURE eval_department (dept_id IN employees.department_id%TYPE)
AS
  CURSOR emp_cursor IS
    SELECT * FROM EMPLOYEES
    WHERE DEPARTMENT_ID = eval_department.dept_id;

  emp_record  EMPLOYEES%ROWTYPE;  -- for row returned by cursor
  all_evals   BOOLEAN;  -- true if all employees in dept need evaluations
  today       DATE;

BEGIN
  today := SYSDATE;

  IF (EXTRACT(MONTH FROM today) < 6) THEN
    all_evals := FALSE; -- only new employees need evaluations
  ELSE
    all_evals := TRUE;  -- all employees need evaluations
  END IF;

  OPEN emp_cursor;

  DBMS_OUTPUT.PUT_LINE (
    'Determining evaluations necessary in department # ' ||
```

```
    dept_id );

  LOOP
    FETCH emp_cursor INTO emp_record;
    EXIT WHEN emp_cursor%NOTFOUND;

    IF all_evals THEN
      add_eval(emp_record.employee_id, today);
    ELSIF (eval_frequency(emp_record.employee_id) = 2) THEN
      add_eval(emp_record.employee_id, today);
    END IF;
  END LOOP;

  DBMS_OUTPUT.PUT_LINE('Processed ' || emp_cursor%ROWCOUNT || ' records.');

  CLOSE emp_cursor;
END eval_department;
```

(パッケージ本体を変更する手順の例は、「チュートリアル: サブプログラムでの変数および定数の宣言」を参照してください。)

3. EMP_EVALパッケージ仕様部をコンパイルします。
4. EMP_EVALパッケージ本体をコンパイルします。

親トピック: レコードおよびカーソルの使用

5.9.7 カーソル変数について

カーソル変数は、1つの問合せに限定されないカーソルと似ています。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。カーソル変数は、サブプログラム間で問合せ結果を渡すときに有用です。

カーソルの詳細は、「カーソルについて」を参照してください。

カーソル変数を宣言するには、REF CURSOR型を宣言し、その型の変数を宣言します(このため、カーソル変数は**REF CURSOR**と呼ばれることもあります)。REF CURSOR型には強弱があります。

強い REF CURSOR 型は、カーソル変数の**RECORD**型である戻り型を指定します。PL/SQLコンパイラでは、これらの**強い型指定の**カーソル変数を、戻り型と異なる行を戻す問合せに使用できません。強いREF CURSOR型は、弱い型よりもエラー発生の可能性が少なく、弱い型はより柔軟です。

弱い REF CURSOR 型は、戻り型を指定しません。PL/SQLコンパイラでは、**弱い型指定**のカーソル変数をすべての問合せに使用できます。弱いREF CURSOR型は置換可能なため、弱いREF CURSOR型を作成するかわりに、事前定義型の、弱いカーソル型SYS_REFCURSORを使用できます。

カーソル変数を宣言した後、(OPEN FOR文を使用して)特定の問合せに対して変数をオープンし、(FETCH文を使用して)結果セットから行を1つずつフェッチしてから、(CLOSE文を使用して)カーソルをクローズするか、または(OPEN FOR文を使用して)別の特定の問合せに対してオープンする必要があります。カーソル変数を別の問合せに対してオープンすると、前の問合せに対してはクローズされます。特定の問合せに対してカーソル変数をクローズした後は、その問合せの結果セットからレコードをフェッチしたり、カーソル属性の値を参照することはできません。

関連項目:

- カーソル変数の使用の詳細は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。

- カーソル変数宣言の構文については、『Oracle Database PL/SQL言語リファレンス』を参照してください。

親トピック: レコードおよびカーソルの使用

5.9.8 結果セット行を1つずつ取得するためのカーソル変数の使用

カーソル変数を使用して結果セットの行を1行ずつ取得できます。

次の手順では、最も単純な形式で必要な各文を使用し、複雑な構文への参照も提示します。

カーソル変数を使用して結果セット行を1つずつ取得するには、次の手順を実行します。

1. 宣言部分で、次の手順を実行します。
    a. REF CURSOR型を宣言します。

    ```
    TYPE cursor_type IS REF CURSOR [ RETURN return_type ];
    ```

    REF CURSOR型宣言の構文の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください。

    b. その型のカーソル変数を宣言します。

    ```
    cursor_variable cursor_type;
    ```

    カーソル変数宣言の構文の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください。

    c. カーソルによって戻された行を格納するレコードを宣言します。

    ```
    record_name return_type;
    ```

    レコード宣言の構文の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください。

2. 実行可能部分で、次の手順を実行します。
    a. カーソル変数を特定の問合せに対してオープンします。

    ```
    OPEN cursor_variable FOR query;
    ```

    OPEN FOR文の構文の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください。

    b. 次に、似た構文を持つLOOP文を使用して、カーソル変数から行(結果セットからの行)を1つずつフェッチします。

    ```
    LOOP
      FETCH cursor_variable INTO record_name;
      EXIT WHEN cursor_variable%NOTFOUND;
      -- Process row that is in record_name:
      statement;
      [ statement; ]...
    END LOOP;
    ```

    FETCH文の構文の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください。

    c. カーソル変数をクローズします。

    ```
    CLOSE cursor_variable;
    ```

    または、カーソル変数を別の問合せに対してオープンすることで、現在の問合せに対してクローズすることもでき

ます。

CLOSE文の構文の詳細は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。

親トピック: レコードおよびカーソルの使用

5.9.9 チュートリアル: 結果セット行を1つずつ取得するためのカーソル変数の使用

このチュートリアルでは、EMP_EVAL.EVAL_DEPARTMENTプロシージャを変更して、宣言カーソル(複数の部門を処理できる)のかわりにカーソル変数を使用する方法と、EMP_EVAL.EVAL_DEPARTMENTおよびEMP_EVAL.ADD_EVALのより効率的な使用方法を示します。

このチュートリアルでは、EMP_EVAL.EVAL_DEPARTMENTおよびEMP_EVAL.ADD_EVALのより効率的な使用方法を示します。レコードの1つのフィールドをADD_EVALに渡して、ADD_EVALで同じレコード内の他の3つのフィールドを抽出する3つの問合せを使用するかわりに、EVAL_DEPARTMENTはレコード全体をADD_EVALに渡し、ADD_EVALは、ドット表記法を使用して他の3つのフィールドの値にアクセスします。

EMP_EVAL.EVAL_DEPARTMENTプロシージャを変更して、カーソル変数を使用するには、次の手順を実行します。

1. EMP_EVALパッケージ仕様部で、太字で示されているように、プロシージャの宣言およびREF CURSOR型の定義を追加します。

```
CREATE OR REPLACE
PACKAGE emp_eval AS

  PROCEDURE eval_department (dept_id IN employees.department_id%TYPE);

  PROCEDURE eval_everyone;

  FUNCTION calculate_score(eval_id IN scores.evaluation_id%TYPE
                         , perf_id IN scores.performance_id%TYPE)
                           RETURN NUMBER;
  TYPE SAL_INFO IS RECORD
      ( j_id jobs.job_id%type
      , sal_min jobs.min_salary%type
      , sal_max jobs.max_salary%type
      , salary employees.salary%type
      , sal_raise NUMBER(3,3));


  TYPE emp_refcursor_type IS REF CURSOR RETURN employees%ROWTYPE;
END emp_eval;
```

2. EMP_EVALパッケージ本体で、次に太字フォントで示されているように、プロシージャEVAL_LOOP_CONTROLの前の宣言を追加し、プロシージャADD_EVALの宣言を編集します。

```
CREATE OR REPLACE
PACKAGE BODY EMP_EVAL AS

  FUNCTION eval_frequency (emp_id IN EMPLOYEES.EMPLOYEE_ID%TYPE)
    RETURN PLS_INTEGER;

  PROCEDURE salary_schedule(emp IN sal_info);

  PROCEDURE add_eval(emp_record IN EMPLOYEES%ROWTYPE, today IN DATE);

  PROCEDURE eval_loop_control(emp_cursor IN emp_refcursor_type);
...
```

(パッケージ本体を変更する手順の例は、「チュートリアル: サブプログラムでの変数および定数の宣言」を参照してください。)

3. 次に太字フォントで示されているように、部門ごとの3つの結果セットを取得し、EVAL_LOOP_CONTROLプロシージャを起動するよう、EVAL_DEPARTMENTプロシージャを変更します。

```
PROCEDURE eval_department(dept_id IN employees.department_id%TYPE) AS
  emp_cursor    emp_refcursor_type;
  current_dept  departments.department_id%TYPE;

BEGIN
  current_dept := dept_id;

  FOR loop_c IN 1..3 LOOP
    OPEN emp_cursor FOR
      SELECT *
      FROM employees
      WHERE current_dept = eval_department.dept_id;

    DBMS_OUTPUT.PUT_LINE
      ('Determining necessary evaluations in department #' ||
       current_dept);

    eval_loop_control(emp_cursor);

    DBMS_OUTPUT.PUT_LINE
      ('Processed ' || emp_cursor%ROWCOUNT || ' records.');

    CLOSE emp_cursor;
    current_dept := current_dept + 10;
  END LOOP;
END eval_department;
```

4. 次に太字フォントで示されているように、ADD_EVALを変更します。

```
PROCEDURE add_eval(emp_record IN employees%ROWTYPE, today IN DATE)
AS
-- (Delete local variables)
BEGIN
  INSERT INTO EVALUATIONS (
    evaluation_id,
    employee_id,
    evaluation_date,
    job_id,
    manager_id,
    department_id,
    total_score
  )
  VALUES (
    evaluations_sequence.NEXTVAL,    -- evaluation_id
    emp_record.employee_id,     -- employee_id
    today,                      -- evaluation_date
    emp_record.job_id,          -- job_id
    emp_record.manager_id,      -- manager_id
    emp_record.department_id,   -- department_id
    0                            -- total_score
);
END add_eval;
```

5. END EMP_EVALの前に、結果セットから個々のレコードをフェッチして処理する、次のプロシージャを追加します。

```
PROCEDURE eval_loop_control (emp_cursor IN emp_refcursor_type) AS
   emp_record      EMPLOYEES%ROWTYPE;
   all_evals       BOOLEAN;
   today           DATE;
BEGIN
  today := SYSDATE;

  IF (EXTRACT(MONTH FROM today) < 6) THEN
    all_evals := FALSE;
  ELSE
    all_evals := TRUE;
  END IF;

  LOOP
    FETCH emp_cursor INTO emp_record;
    EXIT WHEN emp_cursor%NOTFOUND;

    IF all_evals THEN
      add_eval(emp_record, today);
    ELSIF (eval_frequency(emp_record.employee_id) = 2) THEN
      add_eval(emp_record, today);
    END IF;
  END LOOP;
END eval_loop_control;
```

6. END EMP_EVALの前に、企業のすべての従業員を含む結果セットを取得する次のプロシージャを追加します。

```
PROCEDURE eval_everyone AS
  emp_cursor emp_refcursor_type;
BEGIN
  OPEN emp_cursor FOR SELECT * FROM employees;
  DBMS_OUTPUT.PUT_LINE('Determining number of necessary evaluations.');
  eval_loop_control(emp_cursor);
  DBMS_OUTPUT.PUT_LINE('Processed ' || emp_cursor%ROWCOUNT || ' records.');
  CLOSE emp_cursor;
END eval_everyone;
```

7. EMP_EVALパッケージ仕様部をコンパイルします。
8. EMP_EVALパッケージ本体をコンパイルします。

親トピック: レコードおよびカーソルの使用

5.10 連想配列の使用

連想配列は、コレクションの型です。

- コレクションについて
  コレクションは、指定した順序で同じ型の要素を格納するPL/SQLのコンポジット変数で、一次元配列に似ています。コレクションの内部コンポーネントは、**要素**と呼ばれます。各要素には、コレクション内での位置を識別する一意のサブスクリプトがあります。
- 連想配列について
  連想配列は、制限のない一連のキーと値のペアです。各キーは一意であり、対応する値を保持する要素のサブスクリプトとして機能します。そのため、配列内の位置がわからなくても、また配列を横断しなくても要素にアクセスできます。
- 連想配列の宣言

連想配列を宣言するには、連想配列の型を宣言し、その型の変数を宣言します。

- 連想配列の移入
  通常、稠密な連想配列を移入する最も効率的な方法は、BULK COLLECT INTO句を含むSELECT文を使用することです。
- 稠密な連想配列のトラバース
  稠密な連想配列(整数による索引付け)には、要素間の差異がなく、最初と最後の要素の間のすべての要素が定義されていて、それぞれに値があります(値はNULLの場合もあります)。
- スパースな連想配列のトラバース
  スパースな連想配列(文字列による索引付け)には、要素間に差異がある可能性があります。

関連項目:

コレクションの詳細は、次を参照してください。

- 『*Oracle Database*概要』
- 『*Oracle Database PL/SQL*言語リファレンス』

親トピック: ストアド・サブプログラムおよびパッケージの開発

5.10.1 コレクションについて

コレクションは、1次元配列に似ていて、同じ型の要素を指定された順序で格納するPL/SQLの複合変数です。コレクションの内部コンポーネントは、**要素**と呼ばれます。各要素には、コレクション内での位置を識別する一意のサブスクリプトがあります。

コレクション要素にアクセスするには、**サブスクリプト表記法**collection_name(element_subscript)を使用します。

コレクション要素は、スカラー変数のように扱うことができます。また、コレクション全体をサブプログラムのパラメータとして渡すこともできます(送信または受信サブプログラムのどちらもスタンドアロンのサブプログラムでない場合)。

コレクション・メソッドは、コレクションに関する情報を戻す、またはコレクション上で動作する埋込みPL/SQLサブプログラムです。コレクション・メソッドを起動するには、**ドット表記法**collection_name.method_nameを使用します。たとえば、collection_name.COUNTはコレクションの要素の数を戻します。

PL/SQLには、次の3つの型のコレクションがあります。

- 連想配列(以前の「PL/SQL表」または「索引付き表」)
- ネストした表
- 可変配列(VARRAY)

このマニュアルでは、連想配列のみを説明します。

関連項目:

- PL/SQLコレクション型の詳細は、『*Oracle Database PL/SQL*言語リファレンス』を参照してください。
- コレクション・メソッドの詳細は、『*Oracle Database PL/SQL*言語リファレンス』を参照してください。

親トピック: 連想配列の使用

5.10.2 連想配列について

連想配列は、制限のない一連のキーと値のペアです。各キーは一意であり、対応する値を保持する要素のサブスクリプトとして機能します。そのため、配列内の位置がわからなくても、また配列を横断しなくても要素にアクセスできます。

キーのデータ型は、PLS_INTEGERまたはVARCHAR2(length).です

キーのデータ型がPLS_INTEGERで、連想配列が**整数で索引付けされ**、**稠密**(つまり、要素間に差異がない)である場合、最初と最後の要素の間のすべての要素が定義されていて、それぞれに値があります(値はNULLの場合もあります)。

キー・タイプがVARCHAR2 (length)の場合、連想配列は**文字列(length文字)で索引付けされ**、**スパース**です(つまり、要素間に差異がある可能性があります)。

稠密な連想配列をトラバースする場合、要素間の差異を考慮する必要はありません。スパースな連想配列をトラバースする場合は、要素間の差異を考慮する必要があります。

連想配列の要素に値を割り当てるには、代入演算子を使用できます。

```
array_name(key) := value
```

キーが配列にない場合、代入文によって配列にキー-値のペアが追加されます。そうでない場合、文がarray_name(key)の値をvalueに変更します。

連想配列は、データの一時的な格納に便利です。連想配列では、表が必要とするディスク領域やネットワーク操作を使用しません。ただし、連想配列は、データを一時的に格納する用途のため、DML文で操作できません。

パッケージ内で連想配列を宣言し、パッケージ本体の変数に値を割り当てると、連想配列はデータベース・セッションの存続期間中に存在します。そうでない場合は、宣言をしたサブプログラムが存続するかぎり存在します。

関連項目:

連想配列の詳細は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。

親トピック: 連想配列の使用

5.10.3 連想配列の宣言

連想配列を宣言するには、連想配列の型を宣言し、その型の変数を宣言します。

次に、最も単純な構文を示します。

```
TYPE array_type IS TABLE OF element_type INDEX BY key_type;

array_name  array_type;
```

連想配列を宣言する効果的な方法は、次の手順を実行して、カーソルを使用することです。この手順では、必要な各文を最も単純な形式で使用しますが、構文の詳細の参照先も示します。

カーソルを使用して連想配列を宣言するには、次の手順を実行します。

1. 宣言部分で、次の手順を実行します。
    a. カーソルを宣言します。

```
CURSOR cursor_name IS query;
```

宣言カーソルの宣言構文の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください。

b. 連想配列型を宣言します。

```
TYPE array_type IS TABLE OF cursor_name%ROWTYPE
  INDEX BY { PLS_INTEGER | VARCHAR2 (length) }
```

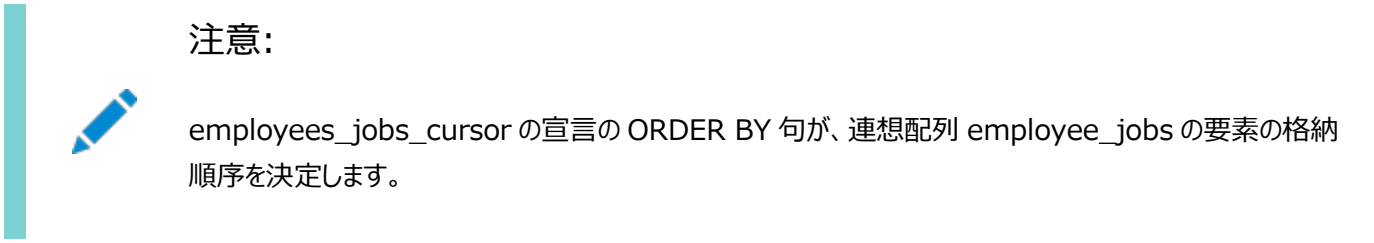
連想配列型宣言の構文の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください。

c. その型の連想配列変数を宣言します。

```
array_name  array_type;
```

変数宣言の構文の詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください。

例5-9では、前述の手順を使用して2つの連想配列employees_jobsおよびjobs_を宣言し、カーソルを使用せずに3つ目の連想配列job_titlesを宣言します。初めの2つの配列は整数によって索引付けされ、3つ目の配列は文字列によって索引付けされます。



注意:

employees_jobs_cursor の宣言の ORDER BY 句が、連想配列 employee_jobs の要素の格納順序を決定します。

例5-9 連想配列の宣言

```
DECLARE
  -- Declare cursor:

  CURSOR employees_jobs_cursor IS
    SELECT FIRST_NAME, LAST_NAME, JOB_ID
    FROM EMPLOYEES
    ORDER BY JOB_ID, LAST_NAME, FIRST_NAME;

  -- Declare associative array type:

  TYPE employees_jobs_type IS TABLE OF employees_jobs_cursor%ROWTYPE
    INDEX BY PLS_INTEGER;

  -- Declare associative array:

  employees_jobs  employees_jobs_type;

  -- Use same procedure to declare another associative array:

  CURSOR jobs_cursor IS
    SELECT JOB_ID, JOB_TITLE
    FROM JOBS;

  TYPE jobs_type IS TABLE OF jobs_cursor%ROWTYPE
    INDEX BY PLS_INTEGER;

  jobs_  jobs_type;

-- Declare associative array without using cursor:

  TYPE job_titles_type IS TABLE OF JOBS.JOB_TITLE%TYPE
```

```
    INDEX BY JOBS.JOB_ID%TYPE;  -- jobs.job_id%type is varchar2(10)

  job_titles  job_titles_type;

BEGIN
  NULL;
END;
/
```

関連項目:

- 「カーソルについて」
- 連想配列宣言の構文については、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。

親トピック: 連想配列の使用

5.10.4 連想配列の移入

通常、稠密な連想配列を移入する最も効率的な方法は、BULK COLLECT INTO句を含むSELECT文を使用することです。

> **注意:**
>
> 稠密な連想配列が非常に大きいため、SELECT 文が大きすぎてメモリーに収まらない結果セットを戻す場合、SELECT 文を使用しないでください。かわりに、カーソルおよび BULK COLLECT INTO および LIMIT 句を含む FETCH 文で配列を移入します。BULK COLLECT INTO 句を含む FETCH 文の使用の詳細は、『*Oracle Database PL/SQL 言語リファレンス*』を参照してください。

SELECT文を使用して、スパースな連想配列(「連想配列の宣言」のjob_titlesなど)を移入できません。かわりに、繰り返し文の中の代入文を使用する必要があります。繰り返し文の詳細は、「プログラム・フローの制御」を参照してください。

例5-10では、SELECT文を使用して、整数で索引付けされるemployees_jobsおよびjobs_の連想配列を移入します。次に、FOR LOOP文内部の代入文を使用して、文字列で索引付けされる連想配列job_titlesを移入します。

例5-10 連想配列の移入

```
-- Declarative part from Example 5-9 goes here.

BEGIN
  -- Populate associative arrays indexed by integer:

SELECT FIRST_NAME, LAST_NAME, JOB_ID BULK COLLECT INTO employees_jobs
  FROM EMPLOYEES ORDER BY JOB_ID, LAST_NAME, FIRST_NAME;

SELECT JOB_ID, JOB_TITLE BULK COLLECT INTO jobs_ FROM JOBS;

  -- Populate associative array indexed by string:

  FOR i IN 1..jobs_.COUNT() LOOP
    job_titles(jobs_(i).job_id) := jobs_(i).job_title;
```

```
  END LOOP;
END;
/
```

関連項目:

「カーソルについて」

親トピック: 連想配列の使用

5.10.5 稠密連想配列の横断

稠密な連想配列(整数による索引付け)には、要素間の差異がなく、最初と最後の要素の間のすべての要素が定義されていて、それぞれに値があります(値はNULLの場合もあります)。

例5-11のように、FOR LOOP文を使用して稠密な配列を横断できます。

例5-11のFOR LOOP文は、例5-10の実行可能部分に挿入すると、employees_jobs配列を移入するコードの後に、employees_jobs配列の要素を、格納された順序で出力します。格納順序は、employees_jobsを宣言するために使用されたemployees_jobs_cursor宣言のORDER BY句によって決定されます(例5-9を参照)。

FOR LOOP文の上限employees_jobs.COUNTは、配列内の要素数を戻すコレクション・メソッドを起動します。COUNT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5-11 稠密連想配列の横断

```
-- Code that populates employees_jobs must precede this code:

FOR i IN 1..employees_jobs.COUNT LOOP
  DBMS_OUTPUT.PUT_LINE(
    RPAD(employees_jobs(i).first_name, 23) ||
    RPAD(employees_jobs(i).last_name,  28) ||     employees_jobs(i).job_id);
  END LOOP;
```

結果:

```
William                 Gietz                       AC_ACCOUNT
Shelley                 Higgins                     AC_MGR
Jennifer                Whalen                      AD_ASST
Steven                  King                        AD_PRES
Lex                     De Haan                     AD_VP
Neena                   Kochhar                     AD_VP
John                    Chen                        FI_ACCOUNT
...
Jose Manuel             Urman                       FI_ACCOUNT
Nancy                   Greenberg                   FI_MGR
Susan                   Mavris                      HR_REP
David                   Austin                      IT_PROG
...
Valli                   Pataballa                   IT_PROG
Michael                 Hartstein                   MK_MAN
Pat                     Fay                         MK_REP
Hermann                 Baer                        PR_REP
Shelli                  Baida                       PU_CLERK
...
Sigal                   Tobias                      PU_CLERK
```

```
Den                  Raphaely                  PU_MAN
Gerald               Cambrault                 SA_MAN
...
Eleni                Zlotkey                   SA_MAN
Ellen                Abel                      SA_REP
...
Clara                Vishney                   SA_REP
Sarah                Bell                      SH_CLERK
...
Peter                Vargas                    ST_CLERK
Adam                 Fripp                     ST_MAN
...
Matthew              Weiss                     ST_MAN
```

親トピック: 連想配列の使用

5.10.6 スパース連想配列の横断

スパース連想配列(文字列によって索引付けされたもの)は、要素間に差分がある場合があります。

例5-12のように、WHILE LOOP文を使用して横断できます。

例5-12でコードを実行し、job_titles配列の要素を出力するには、次の手順を実行します。

1. 例5-9の宣言部分の終わりに、次の変数宣言を挿入します。

   ```
   i jobs.job_id%TYPE;
   ```

2. 例5-10の実行可能部分で、job_titles配列を移入するコードの後に、例5-12のコードを挿入します。

例5-12 スパース連想配列の横断

```
/* Declare this variable in declarative part:

   i jobs.job_id%TYPE;

   Add this code to the executable part,
   after code that populates job_titles:
*/

i := job_titles.FIRST;

WHILE i IS NOT NULL LOOP
  DBMS_OUTPUT.PUT_LINE(RPAD(i, 12) || job_titles(i));
  i := job_titles.NEXT(i);
END LOOP;
```

結果:

```
AC_ACCOUNT  Public Accountant
AC_MGR      Accounting Manager
AD_ASST     Administration Assistant
AD_PRES     President
AD_VP       Administration Vice President
FI_ACCOUNT  Accountant
FI_MGR      Finance Manager
HR_REP      Human Resources Representative
IT_PROG     Programmer
MK_MAN      Marketing Manager
MK_REP      Marketing Representative
PR_REP      Public Relations Representative
```

```
PU_CLERK    Purchasing Clerk
PU_MAN      Purchasing Manager
SA_MAN      Sales Manager
SA_REP      Sales Representative
SH_CLERK    Shipping Clerk
ST_CLERK    Stock Clerk
ST_MAN      Stock Manager
```

例5-12には、job_titles.FIRSTおよびjob_titles.NEXT(i)という2つのコレクション・メソッドの起動が含まれます。job_titles.FIRSTでは、job_titlesの最初の要素が返され、job_titles.NEXT(i)では、iに続くサブスクリプトが返されます。FIRST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。NEXT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。

親トピック: 連想配列の使用

5.11 例外の処理(実行時エラー)

PL/SQLコードで実行時に発生する例外を処理できます。

- 例外および例外ハンドラについて
  PL/SQLコードで実行時エラーが発生すると、**例外**が発生します。例外が発生したサブプログラム(またはブロック)に例外処理部分がある場合は制御が転送され、そうでない場合は実行が停止します。
- 例外ハンドラを使用するタイミング
  次の状況でのみ、例外ハンドラを使用することをお薦めします。
- 事前定義済の例外の処理
  事前定義済の例外を処理できます。
- ユーザー定義の例外の宣言および処理
  ユーザー定義の例外を宣言および処理できます。

関連項目:

PL/SQLエラーの処理の詳細は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。

親トピック: ストアド・サブプログラムおよびパッケージの開発

5.11.1 例外および例外ハンドラについて

PL/SQLコードで実行時エラーが発生すると、**例外**が発生します。例外が発生したサブプログラム(またはブロック)に例外処理部分がある場合は制御が転送され、そうでない場合は実行が停止します。

実行時エラーは、設計障害、コードの誤り、ハードウェア障害およびその他の多くの原因によって発生する可能性があります。

Oracle Databaseには多くの**事前定義済の例外**があり、プログラムがデータベース・ルールに違反したり、システム依存の限度を超えた場合に自動的に発生します。たとえば、SELECT INTO文で行が返されない場合、事前定義済の例外NO_DATA_FOUNDが発生します。PL/SQLの事前定義済の例外の概要は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。

PL/SQLでは、独自の例外を定義(宣言)できます。例外宣言の構文は、次のとおりです。

```
exception_name EXCEPTION;
```

事前定義済の例外と異なり、**ユーザー定義の例外**はRAISE文または DBMS_STANDARD.RAISE_APPLICATION_ERRORプロシージャを使用して、明示的に発生させる必要があります。次に例を示します。

```
IF condition THEN RAISE exception_name;
```

DBMS_STANDARD.RAISE_APPLICATION_ERROR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。

サブプログラムの例外処理部分には、1つ以上の例外ハンドラが含まれています。 **例外ハンドラ**の構文は、次のとおりです。

```
WHEN { exception_name [ OR exception_name ]... | OTHERS } THEN
  statement; [ statement; ]...
```

(「サブプログラム構造について」には、サブプログラムの例外処理部分を挿入する場所が示されています。)

WHEN OTHERS例外ハンドラは、予期しないランタイム・エラーを処理します。これは、最後に使用する必要があります。次に例を示します。

```
EXCEPTION
  WHEN exception_1 THEN
    statement; [ statement; ]...
  WHEN exception_2 OR exception_3 THEN
    statement; [ statement; ]...
  WHEN OTHERS THEN
    statement; [ statement; ]...
    RAISE;  -- Reraise the exception (very important).
END;
```

WHEN OTHERS例外ハンドラのかわりに、EXCEPTION_INITプラグマを使用することもできます。このプラグマによって、ユーザー定義の例外の名前がOracle Databaseのエラー番号に関連付けられます。

関連項目:

- 例外宣言の構文の詳細は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。
- 例外ハンドラの構文の詳細は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。
- EXCEPTION_INIT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。

親トピック: 例外の処理(実行時エラー)

5.11.2 例外ハンドラを使用するタイミング

次の状況でのみ、例外ハンドラを使用することをお薦めします。

- 例外を予期して処理します。

  たとえば、SELECT INTO文で行が返されず、Oracle Databaseで事前定義済の例外NO_DATA_FOUNDが発生するとします。例5-13のとおり、サブプログラムまたはブロックでその例外(エラーではない)を処理して続行します。

- リソースを放棄するか、閉じる必要があります。

  次に例を示します。

  ```
  ...
  file := UTL_FILE.OPEN ...
  ```

```
BEGIN
  statement statement]...  -- If this code fails for any reason,
EXCEPTION
  WHEN OTHERS THEN
    UTL_FILE.FCLOSE(file);     -- then you want to close the file.
    RAISE;                     -- Reraise the exception (very important).
END;
UTL_FILE.FCLOSE(file);
...
```

- コードの最上位レベルで、エラーを記録します。

  たとえば、クライアント・プロセスがこのブロックを発行する場合があります。

```
BEGIN
  proc(...);
EXCEPTION
  WHEN OTHERS THEN
    log_error_using_autonomous_transaction(...);
    RAISE;  -- Reraise the exception (very important).
END;
/
```

  または、クライアントが起動するスタンドアロンのサブプログラムは、同じ例外処理ロジックを含むことができますが、最上位レベルのみです。

親トピック: 例外の処理(実行時エラー)

5.11.3 事前定義済の例外の処理

事前定義済の例外を処理できます。

例5-13では、EMP_EVAL.EVAL_DEPARTMENTプロシージャを変更して事前定義済の例外NO_DATA_FOUNDを処理する方法を太字で示しています。この変更を行い、変更したプロシージャをコンパイルします。(パッケージ本体を変更する方法の例は、「チュートリアル: サブプログラムでの変数および定数の宣言」を参照してください。)

例5-13 事前定義済の例外NO_DATA_FOUNDの処理

```
PROCEDURE eval_department(dept_id IN employees.department_id%TYPE) AS
  emp_cursor    emp_refcursor_type;
  current_dept  departments.department_id%TYPE;

BEGIN
  current_dept := dept_id;

  FOR loop_c IN 1..3 LOOP
    OPEN emp_cursor FOR
      SELECT *
      FROM employees
      WHERE current_dept = eval_department.dept_id;

    DBMS_OUTPUT.PUT_LINE
      ('Determining necessary evaluations in department #' ||
       current_dept);

    eval_loop_control(emp_cursor);

    DBMS_OUTPUT.PUT_LINE
      ('Processed ' || emp_cursor%ROWCOUNT || ' records.');
```

```
    CLOSE emp_cursor;
    current_dept := current_dept + 10;
  END LOOP;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE ('The query did not return a result set');
END eval_department;
```

関連項目:

事前定義済の例外の詳細は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。

親トピック: 例外の処理(実行時エラー)

5.11.4 ユーザー定義の例外の宣言および処理

ユーザー定義の例外を宣言および処理できます。

例5-14では、EMP_EVAL.CALCULATE_SCOREファンクションを変更して2つのユーザー定義の例外wrong_weightおよびwrong_scoreを宣言および処理する方法を太字で示しています。この変更を行い、変更したファンクションをコンパイルします。(パッケージ本体を変更する方法の例は、「チュートリアル: サブプログラムでの変数および定数の宣言」を参照してください。)

例5-14 ユーザー定義の例外の処理

```
FUNCTION calculate_score ( evaluation_id IN scores.evaluation_id%TYPE
                         , performance_id IN scores.performance_id%TYPE )
                         RETURN NUMBER AS

  weight_wrong  EXCEPTION;
  score_wrong   EXCEPTION;
  n_score       scores.score%TYPE;
  n_weight      performance_parts.weight%TYPE;
  running_total NUMBER := 0;
  max_score     CONSTANT scores.score%TYPE := 9;
  max_weight    CONSTANT performance_parts.weight%TYPE:= 1;
BEGIN
  SELECT s.score INTO n_score
  FROM SCORES s
  WHERE evaluation_id = s.evaluation_id
  AND performance_id = s.performance_id;

  SELECT p.weight INTO n_weight
  FROM PERFORMANCE_PARTS p
  WHERE performance_id = p.performance_id;

  BEGIN
    IF (n_weight > max_weight) OR (n_weight < 0) THEN
      RAISE weight_wrong;
    END IF;
  END;

  BEGIN
    IF (n_score > max_score) OR (n_score < 0) THEN
      RAISE score_wrong;
    END IF;
```

```
  END;

  running_total := n_score * n_weight;
  RETURN running_total;

EXCEPTION
  WHEN weight_wrong THEN
    DBMS_OUTPUT.PUT_LINE(
      'The weight of a score must be between 0 and ' || max_weight);
    RETURN -1;
  WHEN score_wrong THEN
    DBMS_OUTPUT.PUT_LINE(
      'The score must be between 0 and ' || max_score);
    RETURN -1;
END calculate_score;
```

関連項目:

ユーザー定義の例外の詳細は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。

親トピック: 例外の処理(実行時エラー)

6 トリガーの使用

トリガーは、指定されたイベントに対応して自動的に実行するストアドPL/SQLユニットです。

- トリガーについて
  トリガーはデータベースに格納され、(有効な状態の場合)指定されたイベントに対して自動的に実行される(起動する)PL/SQLユニットです。
- トリガーの作成
  トリガーを作成するには、SQL Developerのグラフィカル・インタフェースまたはDDL文CREATE TRIGGERのいずれかを使用します。
- トリガーの変更
  トリガーを変更するには、SQL Developerの編集ツールを使用するか、OR REPLACE句を指定してDDL文CREATE TRIGGERを使用します。
- トリガーの無効化および有効化
  使用不可能なオブジェクトを参照するトリガーや、トリガーが原因の遅延なしに大量のデータをアップロードする(リカバリ操作など)トリガーを一時的に無効にする必要があります。参照されたオブジェクトが使用可能になった後、またはデータのアップロード終了後に、トリガーを再度有効にすることができます。
- トリガーのコンパイルおよび依存性について
  コンパイルされたトリガーは、それらが定義されているスキーマ・オブジェクトに依存します。トリガーが依存するオブジェクトが削除または変更され、トリガーとオブジェクトの不一致が生じた場合、そのトリガーは無効になります。
- トリガーの削除

6.1 トリガーについて

トリガーはデータベースに格納され、(有効な状態の場合)指定されたイベントに対して自動的に実行される(起動する)PL/SQLユニットです。

トリガーは次のような構造です。

```
TRIGGER trigger_name
  triggering_event
  [ trigger_restriction ]
BEGIN
  triggered_action;
END;
```

trigger_nameは、スキーマ内のトリガーで一意である必要があります。トリガーには、スキーマ内の別の種類のオブジェクト(表など)と同じ名前を指定できますが、混乱を防ぐネーミング規則を使用することをお薦めします。

トリガーが**有効な**状態の場合、trigger_restrictionがTRUEまたは省略されると、triggering_eventによりデータベースはtriggered_actionを実行します。triggering_eventは、表、ビュー、スキーマ、またはデータベースのいずれかに関連付けられており、次のうちのいずれかとなります。

- DML文(「データ操作言語(DML)文について」を参照)
- DDL文(「データ定義言語(DDL)文について」を参照)
- データベース処理(SERVERERROR、LOGON、LOGOFF、STARTUPまたはSHUTDOWN)

トリガーが**無効**の状態のとき、triggering_eventはデータベースにtriggered_actionを実行させません。これは、trigger_restrictionがTRUEの場合や省略された場合も同様です。

デフォルトでは、トリガーは有効な状態で作成されます。有効なトリガーを無効化したり、無効なトリガーを有効化することができます。

サブプログラムとは異なり、トリガーを直接起動することはできません。トリガーは、ユーザーまたはアプリケーションによって生じるトリガー・イベントによってのみ起動します。正常に処理されないエラーが発生しないかぎり、トリガーが実行されていることに気づかない可能性があります。

単純なトリガーは、次のいずれかの**タイミング・ポイント**で起動できます。

- トリガー・イベントが実行される前(**文レベルのBEFOREトリガー**)
- トリガー・イベントが実行された後(**文レベルのAFTERトリガー**)
- イベントが影響する各行の前(**行レベルのBEFOREトリガー**)
- イベントが影響する各行の後(**行レベルのAFTERトリガー**)

複合トリガーは、複数のタイミング・ポイントで起動できます。複合トリガーの詳細は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。

INSTEAD OFトリガーはビューで定義され、トリガー・イベントはDML文です。DML文を実行するかわりに、Oracle DatabaseはINSTEAD OFトリガーを実行します。詳細は、「INSTEAD OFトリガーの作成」を参照してください。

システム・トリガーは、スキーマまたはデータベースで定義されます。スキーマで定義されたトリガーは、スキーマの所有者(現在のユーザー)に関連付けられた各イベントに対して起動します。データベースで定義されたトリガーは、すべてのユーザーに関連付けられた各イベントに対して起動します。

トリガーの1つの使用目的は、すべてのクライアント・アプリケーションに適用されるビジネス・ルールを実行することです。たとえば、EMPLOYEES表に追加するデータが特定のフォーマットをもっていなければならず、多くのクライアント・アプリケーションがこの表にデータを追加できるとします。表のトリガーを使用すると、追加するすべてのデータが適切なフォーマットになります。任意のクライアントが表にデータを追加するたびにトリガーが実行されるため、クライアントはルールを回避することができず、ルールを実行するコードは、すべてのクライアント・アプリケーションではなくトリガーにのみ格納されて保持されます。トリガーの他の使用目的の詳細は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。

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。

親トピック: トリガーの使用

6.2 トリガーの作成

トリガーを作成するには、SQL Developerのグラフィカル・インタフェースまたはDDL文CREATE TRIGGERのいずれかを使用します。

ここでは、これらの方法を使用してトリガーを作成する方法について説明します。

デフォルトでは、トリガーは有効な状態で作成されます。無効な状態でトリガーを作成するには、DISABLE句のあるCREATE TRIGGER文を使用します。

注意:

トリガーを作成するには、適切な権限を所持する必要がありますが、ここでの説明では、この追加情報は必要ありません。

注意:

このマニュアルのチュートリアルを行うには、ユーザーHR として、SQL Developer から Oracle Database に接続している必要があります。

- OLDおよびNEW擬似レコードについて
  行レベルのトリガーが起動すると、PL/SQL実行時システムによって、OLDおよびNEWという擬似レコードが作成され、それぞれ値を入れられます。レコードのプロパティのすべてではなく、一部を持つため、疑似レコードと呼ばれます。
- チュートリアル: 表の変更を記録するトリガーの作成
  このチュートリアルでは、CREATE TRIGGER文を使用してEVAL_CHANGE_TRIGGERトリガーを作成する方法を示します。このトリガーは、INSERT、UPDATE、またはDELETE文によってEVALUATIONS表が変更されるたびにEVALUATIONS_LOG表に行を追加します。
- チュートリアル: 行を挿入する前に行に対して主キーを生成するトリガーの作成
  このチュートリアルでは、SQL Developerのトリガーの作成ツールを使用して、EVALUATIONS表に行が挿入される前に起動し、EVALUATIONS_SEQUENCEを使用して、その行の主キーに対する一意の番号を生成するトリガーを作成する方法を説明します。
- INSTEAD OFトリガーの作成
  ビューは、表として問合せ結果を表示します。表の変更と同時にビューも変更する場合、INSTEAD OFトリガーを作成する必要があります。このトリガーは、ビューを変更するかわりに基礎となる表を変更します。
- チュートリアル: LOGONおよびLOGOFFイベントを記録するトリガーの作成
  このチュートリアルでは、CREATE TRIGGER文を使用して、HR_LOGON_TRIGGERとHR_LOGOFF_TRIGGERの2つのトリガーを作成する方法を説明します。ユーザー*HR*としてログオンした後、HR_LOGON_TRIGGERはHR_USERS_LOG表に行を追加します。ユーザー*HR*としてログオフする前に、HR_LOGOFF_TRIGGERはHR_USERS_LOG表に行を追加します。

関連項目:

- CREATE 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。
- 「トリガーを作成するインストール・スクリプトの編集」

親トピック: トリガーの使用

6.2.1 OLDおよびNEW疑似レコードについて

行レベルのトリガーが起動すると、PL/SQLランタイム・システムは2つの疑似レコードOLDおよびNEWを作成および移入します。レコードのプロパティのすべてではなく、一部を持つため、疑似レコードと呼ばれます。

トリガーが処理している行では、次のようになります。

- INSERTトリガーの場合、OLDには値が含まれず、NEWには新しい値が含まれます。
- UPDATEトリガーの場合、OLDには古い値が含まれ、NEWには新しい値が含まれます。
- DELETEトリガーの場合、OLDには古い値が含まれ、NEWには値が含まれません。

疑似レコードを参照するには、例6-1のように、:OLDまたは:NEWと名前の前にコロンを付けます。

関連項目:

OLDおよびNEW疑似レコードの詳細は、『Oracle Database PL/SQL言語リファレンス』を参照してください。

親トピック: トリガーの作成

6.2.2 チュートリアル: 表の変更を記録するトリガーの作成

このチュートリアルでは、CREATE TRIGGER文を使用してEVAL_CHANGE_TRIGGERトリガーを作成する方法を示します。このトリガーは、INSERT、UPDATEまたはDELETE文によってEVALUATIONS表が変更されるたびにEVALUATIONS_LOG表に行を追加します。

このトリガーは、トリガー文が実行された後に行を追加し、**条件述語INSERTING**、**UPDATING**および**DELETING**を使用して、使用可能な3つのDML文のうち、トリガーを起動した文を判断します。

EVAL_CHANGE_TRIGGERは、**文レベルのトリガー**で、**AFTERトリガー**です。

EVALUATIONS_LOGおよびEVAL_CHANGE_TRIGGERを作成するには、次の手順を実行します。

1. EVALUATIONS_LOG表を作成します。

```
CREATE TABLE EVALUATIONS_LOG ( log_date DATE
                             , action VARCHAR2(50));
```

2. EVAL_CHANGE_TRIGGERを作成します。

```
CREATE OR REPLACE TRIGGER EVAL_CHANGE_TRIGGER
  AFTER INSERT OR UPDATE OR DELETE
  ON EVALUATIONS
DECLARE
  log_action  EVALUATIONS_LOG.action%TYPE;
BEGIN
  IF INSERTING THEN
    log_action := 'Insert';
  ELSIF UPDATING THEN
    log_action := 'Update';
  ELSIF DELETING THEN
    log_action := 'Delete';
  ELSE
    DBMS_OUTPUT.PUT_LINE('This code is not reachable.');
  END IF;

  INSERT INTO EVALUATIONS_LOG (log_date, action)
    VALUES (SYSDATE, log_action);
END;
```

関連項目:

条件述部の詳細は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。

親トピック: トリガーの作成

6.2.3 チュートリアル: 行を挿入する前に行に対して主キーを生成するトリガーの作成

このチュートリアルでは、SQL Developerのトリガーの作成ツールを使用して、EVALUATIONS表に行が挿入される前に起動し、EVALUATIONS_SEQUENCEを使用して、その行の主キーに対する一意の番号を生成するトリガーを作成する方法を説明します。

(「チュートリアル: 順序の作成」で作成した)順序EVALUATIONS_SEQUENCEは、(「表の作成」で作成した)EVALUATIONS表の主キーを生成します。ただし、これらの主キーは自動的に表に挿入されません。

このチュートリアルでは、SQL Developerのトリガーの作成ツールを使用して、NEW_EVALUATION_TRIGGERというトリガーを作成する方法を説明します。このトリガーは*EVALUATIONS*表に行が挿入される前に起動し、EVALUATIONS_SEQUENCEを使用して、その行の主キーに対する一意の番号を生成します。*INSERT*文のトリガーの影響を受ける行ごとに、このトリガーは一度起動します。

NEW_EVALUATION_TRIGGERは、**行レベルのトリガー**で、**BEFOREトリガー**です。

NEW_EVALUATIONトリガーを作成するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「トリガー」を右クリックします。
3. 選択肢のリストで、「新規トリガー」をクリックします。
4. 「トリガーの作成」ウィンドウで、次の手順を実行します。
    a. 「名前」フィールドで、NEW_EVALUATION_TRIGGERと入力してデフォルト値TRIGGER1を上書きします。
    b. ベース・オブジェクトでは、メニューからEVALUATIONSを選択します。
    c. INSERTを「使用可能なイベント」から「選択済のイベント」に移動します。

    (INSERTを選択して「>」をクリックします。)
    d. 「文レベル」オプションを選択解除します。
    e. 「OK」をクリックします。

    NEW_EVALUATION_TRIGGERペインが開き、トリガーを作成したCREATE TRIGGER文が表示されます。

```
CREATE OR REPLACE
TRIGGER NEW_EVALUATION_TRIGGER
BEFORE INSERT ON EVALUATIONS
FOR EACH ROW
BEGIN
  NULL;
END;
```

    NEW_EVALUATION_TRIGGERペインのタイトルがイタリック・フォントになっています。これは、トリガーがまだデータベースに保存されていないことを示しています。
5. CREATE TRIGGER文で、NULLを次と置き換えます。

```
:NEW.evaluation_id := evaluations_sequence.NEXTVAL
```

6. 「File」メニューから、「Save」を選択します。

   Oracle Databaseは、プロシージャをコンパイルして保存します。NEW_EVALUATION_TRIGGERペインのタイトルがイタリック・フォントではなくなります。

親トピック: トリガーの作成

6.2.4 INSTEAD OFトリガーの作成

ビューは、問合せの出力を表形式で表します。表の変更と同時にビューも変更する場合、INSTEAD OFトリガーを作成する必要があります。このトリガーは、ビューを変更するかわりに基礎となる表を変更します。

たとえば、EMP_LOCATIONSというビューを検討します。 NAME列は、EMPLOYEES表のLAST_NAMEおよびFIRST_NAME列から作成されています。

```
CREATE VIEW EMP_LOCATIONS AS
SELECT e.EMPLOYEE_ID,
  e.LAST_NAME || ', ' || e.FIRST_NAME NAME,
  d.DEPARTMENT_NAME DEPARTMENT,
  l.CITY CITY,
  c.COUNTRY_NAME COUNTRY
FROM EMPLOYEES e, DEPARTMENTS d, LOCATIONS l, COUNTRIES c
WHERE e.DEPARTMENT_ID = d.DEPARTMENT_ID AND
 d.LOCATION_ID = l.LOCATION_ID AND
 l.COUNTRY_ID = c.COUNTRY_ID
ORDER BY LAST_NAME;
```

ビューEMP_LOCATIONS.NAME (「CREATE VIEW文によるビューの作成」で作成)を更新するには、EMPLOYEES.LAST_NAMEおよびEMPLOYEES.FIRST_NAMEを更新する必要があります。これは、例6-1のINSTEAD OFトリガーによって実行されます。

OLDとNEWは**擬似レコード**です。行レベルのトリガーが起動したときにPL/SQL実行時エンジンによって作成され、それぞれ値を入れられます。OLDおよびNEWは、トリガーによって処理されているレコードの元の値と新しい値をそれぞれ格納します。PL/SQLレコードの一部のプロパティを持たないため、疑似レコードと呼ばれます。

例6-1 INSTEAD OFトリガーの作成

```
CREATE OR REPLACE TRIGGER update_name_view_trigger
INSTEAD OF UPDATE ON emp_locations
BEGIN
  UPDATE employees SET
    first_name = substr( :NEW.name, instr( :new.name, ',' )+2),
    last_name = substr( :NEW.name, 1, instr( :new.name, ',')-1)
  WHERE employee_id = :OLD.employee_id;
END;
```

関連項目:

- INSTEAD OF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。
- OLDおよびNEW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。

親トピック: トリガーの作成

6.2.5 チュートリアル: LOGONおよびLOGOFFイベントを記録するトリガーの作成

このチュートリアルでは、CREATE TRIGGER文を使用して、HR_LOGON_TRIGGERとHR_LOGOFF_TRIGGERの2つのトリガーを作成する方法を説明します。ユーザー*HR*としてログオンした後、HR_LOGON_TRIGGERはHR_USERS_LOG表に行を追加します。ユーザー*HR*としてログオフする前に、HR_LOGOFF_TRIGGERはHR_USERS_LOG表に行を追加します。

HR_LOGON_TRIGGERおよびHR_LOGOFF_TRIGGERは**システム・トリガー**です。HR_LOGON_TRIGGERは**AFTERトリガー**、HR_LOGOFF_TRIGGERは**BEFOREトリガー**です。

HR_USERS_LOG、HR_LOGON_TRIGGERおよびHR_LOGOFF_TRIGGERを作成するには、次の手順を実行します。

1. HR_USERS_LOG表を作成します。

```
CREATE TABLE hr_users_log (
  user_name VARCHAR2(30),
  activity VARCHAR2(20),
  event_date DATE
);
```

2. HR_LOGON_TRIGGERを作成します。

```
CREATE OR REPLACE TRIGGER hr_logon_trigger
  AFTER LOGON
  ON HR.SCHEMA
BEGIN
  INSERT INTO hr_users_log (user_name, activity, event_date)
  VALUES (USER, 'LOGON', SYSDATE);
END;
```

3. HR_LOGOFF_TRIGGERを作成します。

```
CREATE OR REPLACE TRIGGER hr_logoff_trigger
  BEFORE LOGOFF
  ON HR.SCHEMA
BEGIN
  INSERT INTO hr_users_log (user_name, activity, event_date)
  VALUES (USER, 'LOGOFF', SYSDATE);
END;
```

関連項目:

システム・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。

親トピック: トリガーの作成

6.3 トリガーの変更

トリガーを変更するには、SQL Developerの編集ツールを使用するか、OR REPLACE句を指定してCREATE TRIGGER DDL文を使用します。

編集ツールを使用してトリガーを変更するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「トリガー」を展開します。

3. トリガーのリストで、変更するトリガーをクリックします。
4. 「接続」フレームの右側の「コード」ペインに、トリガーを作成したコードが表示されます。

   「コード」ペインが書込みモードになっています。(鉛筆アイコンをクリックすると、モードが書込みモードから読取り専用モードに、またはその逆に切り替わります。)

5. 「コード」ペインで、コードを変更します。

   ペインのタイトルはイタリック・フォントになっており、変更がデータベースに保存されていないことを示しています。

6. 「File」メニューから、「Save」を選択します。

   Oracle Databaseは、トリガーをコンパイルして保存します。ペインのタイトルがイタリック・フォントではなくなります。

関連項目:

- CREATE OR REPLACE TRIGGER文に適用される一般情報は、「データ定義言語(DDL)文について」を参照してください。
- CREATE OR REPLACE 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。

親トピック: トリガーの使用

6.4 トリガーの無効化および有効化

使用不可能なオブジェクトを参照するトリガーや、トリガーが原因の遅延なしに大量のデータをアップロードする(リカバリ操作など)トリガーを一時的に無効にする必要があります。参照されたオブジェクトが使用可能になった後、またはデータのアップロード終了後に、トリガーを再度有効にすることができます。

- 単一のトリガーの無効化または有効化
  単一のトリガーを無効化または有効化するには、トリガーの無効化/トリガーの有効化ツール、あるいはDISABLEまたはENABLE句を指定してALTER TRIGGER文を使用します。
- 単一の表のすべてのトリガーの無効化または有効化
  特定の表のすべてのトリガーを無効化または有効化するには、すべてのトリガーの無効化/すべてのトリガーの有効化ツール、あるいはDISABLE ALL TRIGGERSまたはENABLE ALL TRIGGERS句を指定してALTER TABLE文を使用します。

関連項目:

- 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 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。

親トピック: トリガーの使用

6.4.1 単一のトリガーの無効化または有効化

単一のトリガーを無効化または有効化するには、トリガーの無効化/トリガーの有効化ツール、あるいはDISABLEまたはENABLE句を指定してALTER TRIGGER文を使用します。

たとえば、次の文ではeval_change_triggerを無効化および有効化します。

```
ALTER TRIGGER eval_change_trigger DISABLE;
ALTER TRIGGER eval_change_trigger ENABLE;
```

トリガーの無効化/トリガーの有効化ツールを使用するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「トリガー」を展開します。
3. トリガーのリストで、対象のトリガーを右クリックします。
4. 選択肢のリストで、「無効化」または「有効化」を選択します。
5. 「無効化」または「有効化」ウィンドウで、「適用」をクリックします。
6. 「確認」ウィンドウで「OK」をクリックします。

親トピック: トリガーの無効化および有効化

6.4.2 単一の表のすべてのトリガーの無効化または有効化

特定の表のすべてのトリガーを無効化または有効化するには、すべてのトリガーの無効化/すべてのトリガーの有効化ツール、あるいはDISABLE ALL TRIGGERSまたはENABLE ALL TRIGGERS句を指定してALTER TABLE文を使用します。

たとえば、次の文ではevaluations表のすべてのトリガーを無効化および有効化します。

```
ALTER TABLE evaluations DISABLE ALL TRIGGERS;
ALTER TABLE evaluations ENABLE ALL TRIGGERS;
```

すべてのトリガーの無効化/すべてのトリガーの有効化ツールを使用するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「表」を展開します。
3. 表のリストで、対象の表を右クリックします。
4. 選択肢のリストで、「トリガー」を選択します。
5. 選択肢のリストで、「すべて無効化」または「すべて有効化」を選択します。
6. 「すべて無効化」または「すべて有効化」ウィンドウで、「適用」をクリックします。
7. 「確認」ウィンドウで「OK」をクリックします。

親トピック: トリガーの無効化および有効化

6.5 トリガーのコンパイルおよび依存性について

コンパイルされたトリガーは、トリガーが定義されたスキーマ・オブジェクトに依存します。トリガーが依存するオブジェクトが削除または変更され、トリガーとオブジェクトの不一致が生じた場合、そのトリガーは無効になります。

CREATE TRIGGER文を実行すると、作成中のトリガーがコンパイルされます。このコンパイルでエラーが発生した場合、CREATE TRIGGER文は失敗します。コンパイル・エラーを表示するには、このビューを使用します。

```
SELECT * FROM USER_ERRORS WHERE TYPE = 'TRIGGER';
```

コンパイルされたトリガーは、トリガーが定義されたスキーマ・オブジェクトに依存します。たとえば、NEW_EVALUATION_TRIGGERはEVALUATIONS表に依存します。

```
CREATE OR REPLACE
TRIGGER NEW_EVALUATION_TRIGGER
BEFORE INSERT ON EVALUATIONS
FOR EACH ROW
```

```
BEGIN
  :NEW.evaluation_id := evaluations_seq.NEXTVAL;
END;
```

トリガーが依存するスキーマ・オブジェクトを表示するには、次の文を使用します。

```
SELECT * FROM ALL_DEPENDENCIES WHERE TYPE = 'TRIGGER';
```

トリガーが依存するオブジェクトが削除または変更され、トリガーとオブジェクトの不一致が生じた場合、そのトリガーは無効になります。トリガーが次回起動されるとき、トリガーは再コンパイルされます。トリガーをすぐに再コンパイルするには、COMPILE句を含むALTER TRIGGER文を使用します。次に例を示します。

```
ALTER TRIGGER NEW_EVALUATION_TRIGGER COMPILE;
```

関連項目:

トリガーのコンパイルおよび依存性の詳細は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。

親トピック: トリガーの使用

6.6 トリガーの削除

トリガーが依存するオブジェクトを削除する前に、トリガーを削除する必要があります。

トリガーを削除するには、SQL Developerの「接続」フレームと削除ツールまたはDDL文のDROP TRIGGERを使用します。

この文は、トリガーEVAL_CHANGE_TRIGGERを削除します。

```
DROP TRIGGER EVAL_CHANGE_TRIGGER;
```

削除ツールを使用してトリガーを削除するには、次の手順を実行します。

1. 「接続」フレームで、hr_connを展開します。
2. スキーマ・オブジェクト・タイプのリストで、「トリガー」を展開します。
3. トリガーのリストで、削除するトリガーの名前を右クリックします。
4. 選択肢のリストで、「トリガーの削除」をクリックします。
5. 「削除」ウィンドウで、「適用」をクリックします。
6. 「確認」ウィンドウで、yをクリックします。

関連項目:

- DROP TRIGGER文に適用される一般情報は、「データ定義言語(DDL)文について」を参照してください。
- 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。

親トピック: トリガーの使用

7 グローバル環境での作業

グローバリゼーション・サポートにより、多言語アプリケーションを世界のどこからでも同時に実行できます。アプリケーションにより、ユーザー・インタフェースの内容をレンダリングし、各国語およびユーザーのロケール・プリファレンスでデータを処理できます。

- グローバリゼーション・サポート機能について
  グローバリゼーション・サポート機能により、世界のどこからでも同時に実行できる多言語アプリケーションを開発できるようになります。アプリケーションにより、ユーザー・インタフェースの内容をレンダリングし、各国語およびユーザーのロケール・プリファレンスでデータを処理できます。
- NLSパラメータの初期値について
  SQL Developer以外では、NLSパラメータの初期値はデータベース初期化パラメータで設定されます。
- NLSパラメータ値の表示
  NLSパラメータの現在の値を表示するには、SQL Developerの各国語サポート・パラメータ・レポートを使用します。
- NLSパラメータ値の変更
  1つ以上のNLSパラメータの値を、次のいずれかの方法で変更できます。
- 各NLSパラメータについて
  多くのNLSパラメータを利用できます。
- グローバルなアプリケーションでのUnicodeの使用方法
  Unicodeデータを挿入および取得できます。データはデータベースおよびクライアント間で透過的に変換され、クライアント・プログラムがデータベース・キャラクタ・セットおよび各国キャラクタ・セットから独立していることを保証します。

7.1 グローバリゼーション・サポート機能について

グローバリゼーション・サポートにより、世界のどこからでも同時に実行できる多言語アプリケーションを開発できるようになります。アプリケーションにより、ユーザー・インタフェースの内容をレンダリングし、各国語およびユーザーのロケール・プリファレンスでデータを処理できます。

> 注意:
>
> 以前は、グローバリゼーション・サポートを**各国語サポート(NLS)**と呼んでいましたが、実際には NLS はグローバリゼーション・サポートのサブセットです。NLS は、各国の言語を選択し、特定のキャラクタ・セットを使用してデータを格納する機能です。NLS は、NLS パラメータで実装します。

- 言語サポートについて
  Oracle Databaseでは、データを各国語で格納、処理、取得できます。データベースには、Oracleがサポートするキャラクタ・セットでエンコードされたスクリプトに記述されているすべての言語を格納できます。Unicodeデータベースとデータ型を使用することで、Oracle Databaseは、ほとんどの現代言語をサポートしています。
- 地域サポートについて
  デフォルトのローカル時刻書式、日付書式、数値および通貨に関する規則はローカル地域設定によって異なります。
- 日付および時刻書式について
  国によって、時刻、曜日、月および年の表記規則は異なります。
- カレンダ書式について
  国ごとに異なるカレンダを使用します。

- 数値および通貨の書式について
  国によって、数値と通貨の規則は異なります。
- 言語ソートと文字列検索について
  ソート順序(照合順序)は言語によって異なります。また、国や文化が異なると、同じアルファベットでも単語のソート方法が異なります。たとえば、デンマークでは、Zの後にÆがあり、YおよびÜは同じ文字の変形とみなされます。
- 長さセマンティクスについて
  文字列内の文字数をバイト長を使用して計算するには、キャラクタ・セット内の各文字のバイト数を知る必要があります。
- UnicodeおよびSQL各国語キャラクタ・データ型について
  Unicodeは、世界中で話されているほとんどの言語のあらゆる文字を定義する文字エンコード・システムです。Unicodeでは、プラットフォーム、プログラムまたは言語を問わず、すべての文字に一意のコードがあります。

関連項目:

グローバリゼーション・サポート機能の詳細は、『*Oracle Database*グローバリゼーション・サポート・ガイド』を参照してください。

親トピック: グローバル環境での作業

7.1.1 言語サポートについて

Oracle Databaseでは、データの格納、処理および取出しをネイティブ言語で行うことができます。データベースには、Oracleがサポートするキャラクタ・セットでエンコードされたスクリプトに記述されているすべての言語を格納できます。Unicodeデータベースとデータ型を使用することで、Oracle Databaseは、ほとんどの現代言語をサポートしています。

各国語のサブセットに対しては、追加サポートが用意されています。たとえば、データベースでは、翻訳された月の名前を使用して日付を表示し、文化的な慣習に従ってテキスト・データをソートできます。

このマニュアルでは、**言語サポート**という用語は、特定の言語のテキストを格納する機能ではなく、追加の言語依存機能を指します。たとえば、言語サポートには、特定のロケールおよび文化的慣習に従った日付の表示やテキストのソートが含まれます。また、サポートされる言語の一部には、Oracle Databaseによりデータベース・ユーティリティ用に翻訳されたサーバー・メッセージおよび翻訳されたユーザー・インタフェースが提供されます。

関連項目:

- 「NLS_LANGUAGEパラメータについて」
- Oracle Databaseがサポートする言語のリストは、『*Oracle Database*グローバリゼーション・サポート・ガイド』を参照してください。
- Oracle Databaseのメッセージが翻訳されている言語のリストは、『*Oracle Database*グローバリゼーション・サポート・ガイド』を参照してください。

親トピック: グローバリゼーション・サポート機能について

7.1.2 地域サポートについて

デフォルトのローカル時刻書式、日付書式、数値および通貨に関する規則はローカル地域設定によって異なります。

Oracle Databaseでは、地域に固有な文化的慣習をサポートしています。デフォルトのローカル時刻書式、日付書式、数値および通貨に関する規則はローカル地域設定によって異なります。異なるNLSパラメータを設定することにより、データベース・セッションにおいて異なる文化的設定を使用できます。たとえば、地域がAMERICAの場合であっても、指定したデータベース・セッションに対する基本通貨としてユーロ(EUR)を設定し、補助通貨として日本円(JPY)を設定できます。

関連項目:

- 「NLS_TERRITORYパラメータについて」
- Oracle Databaseがサポートする地域のリストは、『*Oracle Database*グローバリゼーション・サポート・ガイド』を参照してください。

親トピック: グローバリゼーション・サポート機能について

7.1.3 日付および時刻書式について

国ごとに、時、日、月および年を表示する規則があります。

たとえば、この表は、5か国の日付および時刻のローカル書式とその例を示しています。

国	日付書式	例	時刻書式	例
中国	yyyy-mm-dd	2005-02-28	hh24:mi:ss	13:50:23
エストニア	dd.mm.yyyy	28.02.2005	hh24:mi:ss	13:50:23
ドイツ	dd.mm.rr	28.02.05	hh24:mi:ss	13:50:23
英国	dd/mm/yyyy	28/02/2005	hh24:mi:ss	13:50:23
米国	mm/dd/yyyy	02/28/2005	hh:mi:ssxff am	1:50:23.555 PM

関連項目:

- 「NLS_DATE_FORMATパラメータについて」
- 「NLS_DATE_LANGUAGEパラメータについて」
- 「NLS_TIMESTAMP_FORMATおよびNLS_TIMESTAMP_TZ_FORMATパラメータについて」
- 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*グローバリゼーション・サポート・ガイド』を参照してください。
- 日付と時刻の書式の詳細は、『*Oracle Database SQL*言語リファレンス』を参照してください。

親トピック: グローバリゼーション・サポート機能について

7.1.4 カレンダ書式について

国ごとに、異なるカレンダを使用します。

Oracle Databaseは、各地域のカレンダ情報を格納しています。

- **週の最初の曜日**

  ある文化では日曜日ですが、別の文化では月曜日です。 NLS_TERRITORYパラメータで設定します。

- **年の最初の暦週**

  一部の国では、週番号を使用してスケジューリング、計画および会計処理を行います。ISO規格では、暦年の週番号とは異なる週番号を使用できます。たとえば、2005年1月1日は、2004年のISOの週番号53となります。ISOの週は月曜日に始まり、日曜日に終わります。ISO規格をサポートするために、OracleにはIW日付書式要素が用意されています。これによりISOの週番号が戻されます。年の最初の暦週はNLS_TERRITORYパラメータで設定します。

- **1年の日数および月数**

  Oracle Databaseでは、デフォルトのグレゴリオ暦の他に、次の6つの暦法をサポートしています。これらの暦法は次のとおりです。

  - Japanese Imperial(日本の元号暦)

    グレゴリオ暦と同じ月数と日数ですが、年は元号ごとに始まります。

  - ROC Official(台湾暦)

    グレゴリオ暦と同じ月数と日数ですが、年は台湾の建国年から始まります。

  - Persian(ペルシャ暦)

    最初の6か月の日数がそれぞれ31日、次の5か月はそれぞれ30日、最後の月は29日または30日(うるう年)です。

  - Thai Buddha(タイ仏教暦): 仏教のカレンダを使用します。

  - Arabic Hijrah(イスラム歴): 月数が12で、日数が354または355です。

  - English Hijrah(英語版イスラム歴): 月数が12で、日数が354または355です。

  暦法は、NLS_CALENDARパラメータで指定します。

- **紀元の年**

  イスラム暦は、ヒジュラ紀元の年から始まります。日本の元号暦は天皇が即位した最初の年から始まります(たとえば、1998年は平成10年となります)。

関連項目:

- 「NLS_TERRITORYパラメータについて」
- 「NLS_CALENDARパラメータについて」
- カレンダ書式の詳細は、『*Oracle Database*グローバリゼーション・サポート・ガイド』を参照してください。

親トピック: グローバリゼーション・サポート機能について

7.1.5 数値および通貨の書式について

数値と通貨の規則は国によって異なります。

この表は、5か国の数値および通貨のローカル書式とその例を示しています。

国	数値書式	通貨書式
中国	1,234,567.89	©1,234.56
エストニア	1 234 567,89	1 234,56 kr
ドイツ	1.234.567,89	1.234,56€
英国	1,234,567.89	£1,234.56
米国	1,234,567.89	$1,234.56

関連項目:

- 「NLS_NUMERIC_CHARACTERSパラメータについて」
- 「NLS_CURRENCYパラメータについて」
- 「NLS_ISO_CURRENCYパラメータについて」
- 「NLS_DUAL_CURRENCYパラメータについて」
- 数値とリストのパラメータの詳細は、『*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。
- 通貨パラメータの詳細は、『*Oracle Database*グローバリゼーション・サポート・ガイド』を参照してください。
- 数値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。

親トピック: グローバリゼーション・サポート機能について

7.1.6 言語ソートと文字列検索について

ソート順序(照合順序)は言語によって異なります。また、国や文化が異なると、同じアルファベットでも単語のソート方法が異なります。たとえば、デンマークでは、Zの後にÆがあり、YおよびÜは同じ文字の変形とみなされます。

関連項目:

- 「NLS_SORTパラメータについて」
- 「NLS_COMPパラメータについて」
- 言語ソートと文字列検索の詳細は、『*Oracle Database*グローバリゼーション・サポート・ガイド』を参照してください。

親トピック: グローバリゼーション・サポート機能について

7.1.7 長さセマンティクスについて

文字列内の文字数をバイト長を使用して計算するには、キャラクタ・セット内の各文字のバイト数を知る必要があります。

シングルバイト・キャラクタ・セットの場合、文字列のバイト数と文字数は同じです。マルチバイト・キャラクタ・セットの場合は、1文字または1つのコード・ポイントが1つ以上のバイトで構成されています。可変幅キャラクタ・セットの場合は、バイト長に基づく文字数の計算が困難な場合があります。列の長さをバイト数単位で計算することを**バイト・セマンティクス**、文字数単位で計算することを**キャラクタ・セマンティクス**と呼びます。

キャラクタ・セマンティクスは、可変幅のマルチバイト文字列の格納要件を指定する場合に役立ちます。たとえば、Unicodeデータベース(AL32UTF8)に、5文字の英語文字とともに最大5文字の中国語文字を格納できるVARCHAR2列が必要であるとします。バイト・セマンティクスを使用すると、この列には、長さ3バイトである中国語文字用に15バイトと、長さ1バイトである英語文字用に5バイト、合計20バイトが必要です。キャラクタ・セマンティクスを使用すると、この列に必要な文字数は10となります。

関連項目:

- 「NLS_LENGTH_SEMANTICSパラメータについて」
- キャラクタ・セットと長さセマンティクスの詳細は、『*Oracle Database*グローバリゼーション・サポート・ガイド』を参照してください。

親トピック: グローバリゼーション・サポート機能について

7.1.8 UnicodeおよびSQL各国語キャラクタ・データ型について

Unicodeは、世界で話されているほとんどの言語のすべての文字を定義した、文字をエンコードするためのシステムです。Unicodeでは、プラットフォーム、プログラムまたは言語を問わず、すべての文字に一意のコードがあります。

Unicode文字は、Oracle Databaseにおいて次の2つの方法で格納できます。

- Unicodeデータベースを作成すると、UTF-8でエンコードされた文字をSQL文字データ型(CHAR、VARCHAR2、CLOBおよびLONG)として格納できます。
- SQL各国語キャラクタ・データ型の列や変数を宣言できます。

SQL各国語キャラクタ・データ型には、NCHAR、NVARCHAR2およびNCLOBがあります。これらはUnicodeデータの格納のみに使用するので、**Unicodeデータ型**とも呼ばれます。

すべてのSQL各国語キャラクタ・データ型に使用する各国語キャラクタ・セットは、データベース作成時に指定します。各国語キャラクタ・セットはUTF8またはAL16UTF16(デフォルト)のいずれかに設定できます。

型がNCHARかNVARCHAR2の列や変数を宣言する場合、指定する長さはバイト数ではなく文字数になります。

関連項目:

- Unicode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。
- Oracle DatabaseでのUnicode文字の格納の詳細は、『*Oracle Database*グローバリゼーション・サポート・ガイド』を参照してください。
- SQL各国語キャラクタ・データ型の詳細は、『*Oracle Database*グローバリゼーション・サポート・ガイド』を参照してくだ

さい。

親トピック: グローバリゼーション・サポート機能について

7.2 NLSパラメータの初期値について

SQL Developerを除いて、NLSパラメータの初期値は、データベース初期化パラメータによって設定されます。

DBAは初期化パラメータ・ファイルで初期化パラメータの値を設定でき、その設定は次回データベースが起動されたときに有効になります。

SQL DeveloperでのNLSパラメータの初期値を表7-1に示します。

表7-1 SQL DeveloperでのNLSパラメータの初期値

パラメータ	初期値
NLS_CALENDAR	GREGORIAN
NLS_CHARACTERSET	AL32UTF8
NLS_COMP	BINARY
NLS_CURRENCY	$
NLS_DATE_FORMAT	DD-MON-RR
NLS_DATE_LANGUAGE	AMERICAN
NLS_DUAL_CURRENCY	$
NLS_ISO_CURRENCY	AMERICA
NLS_LANGUAGE	AMERICAN
NLS_LENGTH_SEMANTICS	BYTE
NLS_NCHAR_CHARACTERSET	AL16UTF16
NLS_NCHAR_CONV_EXCP	FALSE
NLS_NUMERIC_CHARACTERS	.,
NLS_SORT	BINARY

パラメータ	初期値
NLS_TERRITORY	AMERICA
NLS_TIMESTAMP_FORMAT	DD-MON-RR HH.MI.SSXFF AM
NLS_TIMESTAMP_TZ_FORMAT	DD-MON-RR HH.MI.SSXFF AM TZR
NLS_TIME_FORMAT	HH.MI.SSXFF AM
NLS_TIME_TZ_FORMAT	HH.MI.SSXFF AM TZR

関連項目:

初期化パラメータと初期化パラメータ・ファイルの詳細は、『*Oracle Database*管理者ガイド』を参照してください。

親トピック: グローバル環境での作業

7.3 NLSパラメータ値の表示

NLSパラメータの現在の値を表示するには、SQL Developerの各国語サポート・パラメータ・レポートを使用します。

各国語サポート・パラメータ・レポートを表示するには、次の手順を実行します。

1. SQL Developerメニューの「ビュー」メニューで、「レポート」を選択します。
2. 「レポート」ペインで、「データ・ディクショナリ・レポート」を展開します。
3. レポートのリストで、「データベース情報」を展開します。
4. レポートのリストで、「各国語サポート・パラメータ」を選択します。
5. 「接続の選択」ウィンドウで、**hr_conn**を選択します。
6. **「OK」**をクリックします。

   「接続の選択」ウィンドウが閉じて、「各国語サポート・パラメータ」ペインが表示され、NLSパラメータの名前と現在の値が表示されます。

関連項目:

SQL Developerレポートの詳細は、『*Oracle SQL Developer*ユーザーズ・ガイド』を参照してください。

親トピック: グローバル環境での作業

7.4 NLSパラメータ値の変更

1つ以上のNLSパラメータの値を、次のいずれかの方法で変更できます。

- 現在および今後の、すべてのSQL Developer接続の値を変更します。

- クライアントで、対応するNLS環境変数の設定を変更します。

  クライアントでのみ、NLS環境変数の新しい値によって、対応するNLSパラメータの値を上書きします。

  環境変数を使用して、クライアントのロケール依存動作を指定できます。たとえば、Linuxシステムで、次の文はNLS_SORT環境変数の値をFRENCHに設定し、NLS_SORTパラメータの値を上書きします。

```
% setenv NLS_SORT FRENCH
```

注意:

環境変数はプラットフォーム依存である場合があります。

- ALTER SESSION文を次の構文で使用して、現在のセッションの値のみを変更します。

```
ALTER SESSION SET parameter_name=parameter_value
  [ parameter_name=parameter_value ]... ;
```

  現在のセッションでのみ、前述のすべての方法で設定された値を、新しい値で上書きします。

  ALTER SESSIONを使用して、異なるロケールの設定でアプリケーションをテストできます。

- 現在のSQLファンクション起動の値のみを変更します。

  現在のSQLファンクション起動についてのみ、前述のすべての方法で設定された値を、新しい値で上書きします。

- すべてのSQL Developer接続に対するNLSパラメータ値の変更
  次の手順では、現在および今後のすべてのSQL Developer接続に対するNLSパラメータの値を変更できます。
- 現在のSQLファンクション起動に対するNLSパラメータ値の変更
  動作がNLSパラメータ値に依存しているSQLファンクションは、**ロケール依存**と呼ばれます。ロケール依存のSQLファンクションには、オプションのNLSパラメータがあります。

関連項目:

- 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。
- NLSパラメータの設定の詳細は、『*Oracle Databaseグローバリゼーション・サポート・ガイド*』を参照してください。

親トピック: グローバル環境での作業

7.4.1 すべてのSQL Developer接続に対するNLSパラメータ値の変更

現在および今後の、すべてのSQL Developer接続に対するNLSパラメータの値を変更できます。

各国語サポート・パラメータ値を変更するには、次の手順を実行します。

1. SQL Developerメニューの「ツール」メニューで、「プリファレンス」を選択します。
2. 「プリファレンス」ウィンドウの左のフレームで、「データベース」を展開します。
3. データベース・プリファレンスのリストで、「NLS」をクリックします。

   NLSパラメータとその現在値が表示されます。値フィールドはメニューになっています。

4. 値を変更する各パラメータの右側にあるメニューで、必要な値を選択します。
5. 「OK」をクリックします。

これで、NLSパラメータが、指定した値になります。これらの値を確認するには、「NLSパラメータ値の表示」を参照してください。

注意:

NLS パラメータ値に変更が反映されない場合、「レポートの実行」アイコンをクリックします。

関連項目:

SQL Developerのプリファレンスの詳細は、『*Oracle SQL Developerユーザーズ・ガイド*』を参照してください。

親トピック: NLSパラメータ値の変更

7.4.2 現在のSQLファンクション起動に対するNLSパラメータ値の変更

動作がNLSパラメータの値に依存するSQLファンクションは**ロケール依存**と呼ばれます。ロケール依存のSQLファンクションには、オプションのNLSパラメータがあります。

オプションのNLSパラメータがあるロケール依存のファンクションは次のとおりです。

- TO_CHAR
- TO_DATE
- TO_NUMBER
- NLS_UPPER
- NLS_LOWER
- NLS_INITCAP
- NLSSORT

前述のファンクションでは、次のNLSパラメータを指定できます。

- NLS_DATE_LANGUAGE
- NLS_DATE_LANGUAGE
- NLS_NUMERIC_CHARACTERS
- NLS_CURRENCY
- NLS_ISO_CURRENCY
- NLS_DUAL_CURRENCY
- NLS_CALENDAR
- NLS_SORT

NLSSORTファンクションでは、次のNLSパラメータも指定できます。

- NLS_LANGUAGE
- NLS_TERRITORY

- NLS_DATE_FORMAT

ファンクションでNLSパラメータを指定するには、次の構文を使用します。

```
'parameter=value' ['parameter=value']...
```

次の問合せを評価するときに、NLS_DATE_LANGUAGEをAMERICANにすると想定します。

```
SELECT last_name FROM employees WHERE hire_date > '01-JAN-1999';
```

問合せを実行する前に、NLS_DATE_LANGUAGEをAMERICANに設定できます。

```
ALTER SESSION SET NLS_DATE_LANGUAGE=American;
SELECT last_name FROM employees WHERE hire_date > '01-JAN-1999';
```

または、ロケール依存のSQLファンクションTO_DATEにオプションのNLS_DATE_LANGUAGEパラメータを使用して、問合せの内部でNLS_DATE_LANGUAGEをAMERICANに設定できます。

```
SELECT last_name FROM employees
WHERE hire_date > TO_DATE('01-JAN-1999', 'DD-MON-YYYY',
                          'NLS_DATE_LANGUAGE=AMERICAN');
```

> ヒント:
>
> 通常、SQL ファンクションで NLS パラメータにセッションのデフォルト値を使用すると、パフォーマンスが改善されます。このため、デフォルトの NLS パラメータ値を使用する必要がない SQL 文の場合にのみ、ロケール依存の SQL ファンクションでオプションの NLS パラメータを指定します。

関連項目:

オプションのNLSパラメータがある、ロケール依存のSQLファンクションの詳細は、『*Oracle Database*グローバリゼーション・サポート・ガイド』を参照してください。

親トピック: NLSパラメータ値の変更

7.5 各NLSパラメータについて

多くのNLSパラメータを使用できます。

- ロケールとNLS_LANGパラメータについて
  ロケールとは、システムやアプリケーションを実行する言語的および文化的環境のことです。Oracle Database ソフトウェアでロケールを指定する最も簡単な方法は、NLS_LANGパラメータを設定することです。
- NLS_LANGUAGEパラメータについて
  このパラメータは、データベースのデフォルト言語を指定します。
- NLS_TERRITORYパラメータについて
  このパラメータは、日付形書式、タイム・スタンプ書式、小数点文字およびグループ・セパレータ、各国通貨記号、ISO通貨記号、二重通貨記号に対するデフォルト規則を指定します。
- NLS_DATE_FORMATパラメータについて

このパラメータは、TO_CHARおよびTO_DATEファンクションを使用するためのデフォルトの日付書式を指定します。

- NLS_DATE_LANGUAGEパラメータについて
  このパラメータは、SQLファンクションTO_CHARおよびTO_DATEで生成される曜日および月の名前と略称の言語、デフォルトの日付書式(NLS_DATE_FORMATで設定)、およびAM、PM、AD、BCに相当する、デフォルト言語の記号を指定します。
- NLS_TIMESTAMP_FORMATおよびNLS_TIMESTAMP_TZ_FORMATパラメータについて
  このパラメータは、TIMESTAMPオーディオテープおよびTIMESTAMP WITH LOCAL TIME ZONEオーディオテープのデフォルトの日付書式を定義します。
- NLS_CALENDARパラメータについて
  このパラメータは、データベースの暦法を指定します。
- NLS_NUMERIC_CHARACTERSパラメータについて
  このパラメータは、小数点記号(数値の整数部と小数部の区切り)とグループ・セパレータ(千や100万などの桁区切り)を指定します。この文字は数値書式要素Gで戻されます。
- NLS_CURRENCYパラメータについて
  このパラメータは、各国の通貨記号(数値書式要素Lが返す文字列)を指定します。
- NLS_ISO_CURRENCYパラメータについて
  このパラメータは、各国の通貨記号(数値書式要素Cが返す文字列)を指定します。
- NLS_DUAL_CURRENCYパラメータについて
  このパラメータは、二重通貨記号(ユーロ移行期間中にユーロ通貨記号をサポートするために導入)を指定します。
- NLS_SORTパラメータについて
  このパラメータは、ORDER BY句を持つ問合せの言語ソート順序(照合順序)を指定します。
- NLS_COMPパラメータについて
  このパラメータは、SQL操作の文字比較操作を指定します。
- NLS_LENGTH_SEMANTICSパラメータについて
  このパラメータは、文字データ型CHAR、VARCHAR2およびLONGの列の長さセマンティクス、つまりこれらの列がバイト単位または文字単位のどちらで指定されているかを指定します。これらの列がバイトまたは文字のいずれかで指定されるかを指定します(パラメータ設定後に宣言された列にのみ適用可能)。

関連項目:

- グローバリゼーション・サポート環境の設定の詳細は、『*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。
- 「NLSパラメータ値の変更」

親トピック: グローバル環境での作業

7.5.1 ロケールとNLS_LANGパラメータについて

ロケールとは、システムやアプリケーションを実行する言語的および文化的環境のことです。Oracle Database ソフトウェアでロケールを指定する最も簡単な方法は、NLS_LANGパラメータを設定することです。

NLS_LANGパラメータにより、サーバー・セッション(SQL文の処理など)とクライアント・アプリケーション(Oracle Databaseツールの表示の書式など)の両方に対するNLS_LANGUAGEとNLS_TERRITORYパラメータのデフォルト値を設定できます。また、NLS_LANGパラメータで、入力または表示されるデータに対してクライアント・アプリケーションが使用するキャラクタ・セットも設定できます。

NLS_LANGのデフォルト値は、データベースのインストール中に設定されます。 ALTER SESSION文を使用して、NLS_LANGで設定されるものも含めて、セッションに対するNLSパラメータの値を変更できます。ただし、クライアント環境のNLS設定を変更できるのは、そのクライアントのみです。

関連項目:

- NLS_LANGパラメータを使用したロケールの指定の詳細は、『*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。
- Oracle Databaseでサポートされている、言語、地域、キャラクタ・セットなどのロケール・データの詳細は、『*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。
- 「NLS_LANGUAGEパラメータについて」
- 「NLS_TERRITORYパラメータについて」
- 「NLSパラメータ値の変更」

親トピック: 各NLSパラメータについて

7.5.2 NLS_LANGUAGEパラメータについて

このパラメータはデータベースのデフォルト言語を指定します。

指定項目: データベースのデフォルト言語。次の項目に対するデフォルト規則を指定します。

- サーバー・メッセージの言語
- 曜日および月の名前と略称の言語は、SQLファンクションTO_CHARおよびTO_DATEで指定されます
- AM、PM、ADおよびBCに相当する、デフォルト言語の記号
- ORDER BY句が指定された場合の、文字データに対するデフォルトのソート順序
- 記述方向
- 肯定および否定の応答文字列(たとえば、YESとNO)

許容可能な値: Oracleがサポートする言語名。リストは、『*Oracle Database*グローバリゼーション・サポート・ガイド』を参照してください。

デフォルト値: NLS_LANGによって設定されます。詳細は、「ロケールとNLS_LANGパラメータについて」を参照してください。

デフォルト値の設定対象:

- NLS_DATE_LANGUAGE (「NLS_DATE_LANGUAGEパラメータについて」を参照)
- NLS_SORT(「NLS_SORTパラメータについて」を参照)

例7-1では、NLS_LANGUAGEをITALIANおよびGERMANに設定した場合に、サーバー・メッセージや月の略称にどのように影響するかを示します。

この例をSQL Developerで試す場合は、ワークシートに文や問合せを入力します。ワークシートの詳細は、「SQL Developerにおける問合せの実行」を参照してください。ここに示す結果はSQL*Plusによるもので、SQL Developerとは書式が若干異なります。

例7-1 NLS_LANGUAGEによるサーバー・メッセージおよび月の略称への影響

1. NLS_LANGUAGEの現在の値を書き留めます。
2. 手順1の値がITALIANでない場合は変更します。

```
ALTER SESSION SET NLS_LANGUAGE=ITALIAN;
```

3. 存在しない表に問合せをします。

```
SELECT * FROM nonexistent_table;
```

結果:

```
SELECT * FROM nonexistent_table
              *
ERROR at line 1:
ORA-00942: tabella o vista inesistente
```

4. 次の問合せを実行します。

```
SELECT LAST_NAME, HIRE_DATE
FROM EMPLOYEES
WHERE EMPLOYEE_ID IN (111, 112, 113);
```

結果:

```
LAST_NAME                 HIRE_DATE
------------------------- ---------
Sciarra                   30-SET-97
Urman                     07-MAR-98
Popp                      07-DIC-99

3 rows selected.
```

5. NLS_LANGUAGEの値をGERMANに変更します。

```
ALTER SESSION SET NLS_LANGUAGE=GERMAN;
```

6. 手順3の問合せを繰り返します。

結果:

```
SELECT * FROM nonexistent_table
              *
ERROR at line 1:
ORA-00942: Tabelle oder View nicht vorhanden
```

7. 手順4の問合せを繰り返します。

結果:

```
LAST_NAME                 HIRE_DATE
------------------------- ---------
Sciarra                   30-SEP-97
Urman                     07-MRZ-98
Popp                      07-DEZ-99

3 rows selected.
```

8. NLS_LANGUAGEを手順1の値に設定します。

関連項目:

- NLS_LANGUAGEパラメータの詳細は、『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。
- 「言語サポートについて」
- 「NLSパラメータ値の変更」

親トピック: 各NLSパラメータについて

7.5.3 NLS_TERRITORYパラメータについて

このパラメータは、日付形書式、タイム・スタンプ書式、小数点文字およびグループ・セパレータ、各国通貨記号、ISO通貨記号、二重通貨記号に対するデフォルト規則を指定します。

指定項目: 次のデフォルト規則。

- 日付書式
- タイム・スタンプ書式
- 小数点記号および桁グループ・セパレータ
- 各国通貨記号
- ISO通貨記号
- 二重通貨記号

許容可能な値: Oracleがサポートする地域名。リストは、『*Oracle Database*グローバリゼーション・サポート・ガイド』を参照してください。

デフォルト値: NLS_LANGによって設定されます。詳細は、「ロケールとNLS_LANGパラメータについて」を参照してください。

デフォルト値の設定対象:

- NLS_DATE_FORMAT (「NLS_DATE_FORMATパラメータについて」を参照)
- NLS_TIMESTAMP_FORMATおよびNLS_TIMESTAMP_TZ_FORMAT (「NLS_TIMESTAMP_FORMATおよびNLS_TIMESTAMP_TZ_FORMATパラメータについて」を参照)
- NLS_NUMERIC_CHARACTERS(「NLS_NUMERIC_CHARACTERSパラメータについて」を参照)
- NLS_CURRENCY (「NLS_CURRENCYパラメータについて」を参照)
- NLS_ISO_CURRENCY (「NLS_ISO_CURRENCYパラメータについて」を参照)
- NLS_DUAL_CURRENCY (「NLS_DUAL_CURRENCYパラメータについて」を参照)

例7-2では、NLS_TERRITORYをJAPANおよびAMERICAに設定した場合に、通貨記号にどのように影響するかを示します。

この例をSQL Developerで試す場合は、ワークシートに文や問合せを入力します。ワークシートの詳細は、「SQL Developerにおける問合せの実行」を参照してください。ここに示す結果はSQL*Plusによるもので、SQL Developerとは書式が若干異なります。

例7-2 NLS_TERRITORYによる通貨記号への影響

1. NLS_TERRITORYの現在の値を書き留めます。

2. 手順1の値がJAPANでない場合は変更します。

```
ALTER SESSION SET NLS_TERRITORY=JAPAN;
```

3. 次の問合せを実行します。

```
SELECT TO_CHAR(SALARY,'L99G999D99') SALARY
FROM EMPLOYEES
WHERE EMPLOYEE_ID IN (100, 101, 102);
```

結果:

```
SALARY
--------------------
          ¤24,000.00
          ¤17,000.00
          ¤17,000.00

3 rows selected.
```

4. NLS_TERRITORYの値をAMERICAに変更します。

```
ALTER SESSION SET NLS_TERRITORY=AMERICA;
```

5. 手順3の問合せを繰り返します。

結果:

```
SALARY
--------------------
          $24,000.00
          $17,000.00
          $17,000.00

3 rows selected.
```

6. NLS_TERRITORYを手順1の値に設定します。

関連項目:

- NLS_TERRITORYパラメータの詳細は、『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。
- 「地域サポートについて」
- 「NLSパラメータ値の変更」

親トピック: 各NLSパラメータについて

7.5.4 NLS_DATE_FORMATパラメータについて

このパラメータは、TO_CHARおよびTO_DATEファンクションで使用するデフォルトの日付書式を指定します。

指定内容: TO_CHARおよびTO_DATEファンクション(「問合せにおける変換ファンクションの使用」を参照)で使用するデフォルトの日付書式。

許容可能な値: 有効な任意の日付時間書式モデル。次に例を示します。

```
NLS_DATE_FORMAT='MM/DD/YYYY'
```

日付時間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。

デフォルト値: NLS_TERRITORYによって設定されます。詳細は、「NLS_TERRITORYパラメータについて」を参照してください。

デフォルトの日付書式が、特定の地域で使用される規則に対応していない場合があります。ローカライズされた書式で日付を取得するために、'DS'(短い日付)書式および'DL'(長い日付)書式を使用できます。

例7-3では、NLS_TERRITORYをAMERICAおよびFRANCEに設定した場合に、デフォルトの日付書式、短い日付書式および長い日付書式にどのように影響するかを示します。

例7-4では、NLS_DATE_FORMATの値を変更し、NLS_TERRITORYによって設定されたデフォルト値を上書きします。

この例をSQL Developerで試す場合は、ワークシートに文や問合せを入力します。ワークシートの詳細は、「SQL Developerにおける問合せの実行」を参照してください。ここに示す結果はSQL*Plusによるもので、SQL Developerとは書式が若干異なります。

例7-3 NLS_TERRITORYによる日付記号への影響

1. NLS_TERRITORYの現在の値を書き留めます。
2. 手順1の値がAMERICAでない場合は変更します。

```
ALTER SESSION SET NLS_TERRITORY=AMERICA;
```

3. 次の問合せを実行します。

```
SELECT hire_date "Default",
       TO_CHAR(hire_date,'DS') "Short",
       TO_CHAR(hire_date,'DL') "Long"
FROM employees
WHERE employee_id IN (111, 112, 113);
```

結果:

```
Default    Short      Long
---------  ---------- ----------------------------
30-SEP-05 9/30/2005  Friday, September 30, 2005
07-MAR-98 3/7/2006   Tuesday, March 07, 2006
07-DEC-99 12/7/2007  Friday, December 07, 2007

3 rows selected.
```

4. NLS_TERRITORYの値をFRANCEに変更します。

```
ALTER SESSION SET NLS_TERRITORY=FRANCE;
```

5. 手順3の問合せを繰り返します。

結果:

```
Default  Short      Long
-------- ---------- --------------------------
30/09/05 30/09/2005 friday 30 september 2005
07/03/06 07/03/2006 tuesday 7 march 2006
07/12/07 07/12/2007 friday 7 december 2007

3 rows selected.
```

(フランス語の曜日名と月名を取得するには、問合せを実行する前に、NLS_LANGUAGEまたはNLS_DATE_LANGUAGEをFRENCHに設定する必要があります。)

6. NLS_TERRITORYを手順1の値に設定します。

例7-4 NLS_DATE_FORMATによるNLS_TERRITORYの上書き

1. NLS_TERRITORYとNLS_DATE_FORMATの現在の値を書き留めます。
2. 手順1でNLS_TERRITORYの値がAMERICAでない場合は、次のように変更します。

```
ALTER SESSION SET NLS_TERRITORY=AMERICA;
```

3. 手順1でNLS_DATE_FORMATの値が'Day Month ddth'でない場合は、次のように変更します。

```
ALTER SESSION SET NLS_DATE_FORMAT='Day Month ddth';
```

4. 次の問合せを実行します(前述の例の手順3より)。

```
SELECT hire_date "Default",
       TO_CHAR(hire_date,'DS') "Short",
       TO_CHAR(hire_date,'DL') "Long"
FROM employees
WHERE employee_id IN (111, 112, 113);
```

結果:

```
Default                   Short      Long
------------------------- ---------- ------------------------------
Friday    September 30th 9/30/2005  Tuesday, September 30, 2005
Tuesday   March     07th 3/7/2006   Saturday, March 07, 2006
Friday    December  07th 12/7/2007  Tuesday, December 07, 2007

3 rows selected.
```

5. NLS_TERRITORYとNLS_DATE_FORMATを手順1の値に設定します。

関連項目:

- NLS_DATE_FORMATパラメータの詳細は、『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ファンクションの詳細は、『Oracle Database SQL言語リファレンス』を参照してください。
- TO_DATEファンクションの詳細は、『Oracle Database SQL言語リファレンス』を参照してください。
- 「日付および時刻書式について」
- 「NLSパラメータ値の変更」

親トピック: 各NLSパラメータについて

7.5.5 NLS_DATE_LANGUAGEパラメータについて

このパラメータは、SQLファンクションTO_CHARおよびTO_DATEで生成される曜日および月の名前と略称の言語、デフォルトの日付書式(NLS_DATE_FORMATで設定)、およびAM、PM、AD、BCに相当する、デフォルト言語の記号を指定します。

指定項目: 次の方法で作成した曜日と月の名前と略号。

- SQLファンクションTO_CHARおよびTO_DATE (「問合せにおける変換ファンクションの使用」を参照)
- デフォルトの日付書式(「NLS_DATE_FORMATパラメータについて」で説明されているNLS_DATE_FORMATによって設定)
- AM、PM、ADおよびBCに相当する、デフォルト言語の記号

許容可能な値: Oracleがサポートする言語名。リストは、『*Oracle Database*グローバリゼーション・サポート・ガイド』を参照してください。

デフォルト値: NLS_LANGUAGEによって設定されます。詳細は、「NLS_LANGUAGEパラメータについて」を参照してください。

例7-5では、NLS_DATE_LANGUAGEをFRENCHおよびSWEDISHに設定した場合に、表示されるシステム日付にどのように影響するかを示します。

この例をSQL Developerで試す場合は、ワークシートに文や問合せを入力します。ワークシートの詳細は、「SQL Developerにおける問合せの実行」を参照してください。ここに示す結果はSQL*Plusによるもので、SQL Developerとは書式が若干異なります。

例7-5 NLS_DATE_LANGUAGEによる、表示されるSYSDATEへの影響

1. NLS_DATE_LANGUAGEの現在の値を書き留めます。
2. 手順1でNLS_DATE_LANGUAGEの値がFRENCHでない場合は変更します。

```
ALTER SESSION SET NLS_DATE_LANGUAGE=FRENCH;
```

3. 次の問合せを実行します。

```
SELECT TO_CHAR(SYSDATE, 'Day:Dd Month yyyy') "System Date"
FROM DUAL;
```

結果:

```
System Date
--------------------------
Vendredi:28 December   2012
```

4. NLS_DATE_LANGUAGEの値をSWEDISHに変更します。

```
ALTER SESSION SET NLS_DATE_LANGUAGE=SWEDISH;
```

5. 手順3の問合せを繰り返します。

結果:

```
System Date
--------------------------
Fredag :28 December   2012
```

6. NLS_DATE_LANGUAGEを手順1の値に設定します。

関連項目:

- NLS_DATE_LANGUAGEパラメータの詳細は、『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ファンクションの詳細は、『Oracle Database SQL言語リファレンス』を参照してください。
- ロギング属性の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。
- 「日付および時刻書式について」
- 「NLSパラメータ値の変更」

親トピック: 各NLSパラメータについて

7.5.6 NLS_TIMESTAMP_FORMATおよびNLS_TIMESTAMP_TZ_FORMATパラメータについて

このパラメータは、TIMESTAMPオーディオテープおよびTIMESTAMP WITH LOCAL TIME ZONEオーディオテープのデフォルトの日付書式を指定します。

指定項目: 次のデフォルト日付書式:

- TIMESTAMPオーディオテープ
- TIMESTAMP WITH LOCAL TIME ZONEオーディオテープ

許容可能な値: 有効な任意の日付時間書式モデル。次に例を示します。

```
NLS_TIMESTAMP_FORMAT='YYYY-MM-DD HH:MI:SS.FF'
NLS_TIMESTAMP_TZ_FORMAT='YYYY-MM-DD HH:MI:SS.FF TZH:TZM'
```

日付時間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。

デフォルト値: NLS_TERRITORYによって設定されます。詳細は、「NLS_TERRITORYパラメータについて」を参照してください。

関連項目:

- NLS_TIMESTAMP_FORMATパラメータの詳細は、『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。
- NLS_TIMESTAMP_TZ_FORMATパラメータの詳細は、『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。
- 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グローバリゼーション・サポート・ガイド*』を参照してください。
- TIMESTAMPオーディオテープの詳細は、『Oracle Database SQL言語リファレンス』を参照してください。
- TIMESTAMP WITH LOCAL TIME ZONE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。
- 「日付および時刻書式について」
- 「NLSパラメータ値の変更」

親トピック: 各NLSパラメータについて

7.5.7 NLS_CALENDARパラメータについて

このパラメータは、データベースの暦法を指定します。

指定項目: データベースの暦法

許容可能な値: Oracleがサポートする暦法。リストは、『*Oracle Database*グローバリゼーション・サポート・ガイド』を参照してください。

デフォルト値: Gregorian

例7-6では、NLS_CALENDARを'English Hijrah'およびGregorianに設定した場合に、表示されるシステム日付にどのように影響するかを示します。

この例をSQL Developerで試す場合は、ワークシートに文や問合せを入力します。ワークシートの詳細は、「SQL Developerにおける問合せの実行」を参照してください。ここに示す結果はSQL*Plusによるもので、SQL Developerとは書式が若干異なります。

例7-6 NLS_CALENDARによる、表示されるSYSDATEへの影響

1. NLS_CALENDARの現在の値を書き留めます。
2. 手順1で、NLS_CALENDARの値が'English Hijrah'でない場合は変更します。

```
ALTER SESSION SET NLS_CALENDAR='English Hijrah';
```

3. 次の問合せを実行します。

```
SELECT SYSDATE FROM DUAL;
```

結果:

```
SYSDATE
-------------------------
17 Safar              1434
```

4. NLS_CALENDARの値を'Gregorian'に変更します。

```
ALTER SESSION SET NLS_CALENDAR='Gregorian';
```

5. 次の問合せを実行します。

```
SELECT SYSDATE FROM DUAL;
```

結果:

```
SYSDATE
---------
31-DEC-12
```

6. NLS_CALENDARを手順1の値に設定します。

関連項目:

- NLS_CALENDARパラメータの詳細は、『*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。
- 「カレンダ書式について」
- 「NLSパラメータ値の変更」

親トピック: 各NLSパラメータについて

7.5.8 NLS_NUMERIC_CHARACTERSパラメータについて

このパラメータは、小数点記号(数値の整数部と小数部の区切り)とグループ・セパレータ(千や100万などの桁区切り)を指定します。この文字は数値書式要素Gで戻されます。

指定項目: 小数点記号(数値の整数部と小数部の区切り)とグループ・セパレータ(千や100万などの桁区切り)。この文字は数値書式要素Gで戻されます。

許容可能な値: 次の文字以外の、任意の異なる2つのシングルバイト文字。

- 数字
- プラス(+)
- マイナス(-)
- 小なり(<)
- 大なり(>)

デフォルト値: NLS_TERRITORYによって設定されます。詳細は、「NLS_TERRITORYパラメータについて」を参照してください。

SQL文では数値を次のいずれかの方法で表現できます。

- 数値リテラル

  数値リテラルは引用符で囲まず、小数点文字として常にピリオド(.)を使用し、グループ・セパレータは含めません。

- テキスト・リテラル

  テキスト・リテラルは一重引用符で囲みます。テキスト・リテラルは、必要に応じて、現行のNLS設定に従って暗黙的または明示的に数値に変換されます。

例7-7では、2つの異なるNLS_NUMERIC_CHARACTERSの設定により、同じ問合せの表示結果にどのような影響があるかを示します。

この例をSQL Developerで試す場合は、ワークシートに文や問合せを入力します。ワークシートの詳細は、「SQL Developerにおける問合せの実行」を参照してください。ここに示す結果はSQL*Plusによるもので、SQL Developerとは書式が若干異なります。

例7-7 NLS_NUMERIC_CHARACTERSによる小数点文字とグループ・セパレータへの影響

1. NLS_NUMERIC_CHARACTERSの現在の値を書き留めます。
2. 手順1で、NLS_NUMERIC_CHARACTERSの値が",."(小数点文字はカンマ、グループ・セパレータはピリオド)でない場合は変更します。

   ```
   ALTER SESSION SET NLS_NUMERIC_CHARACTERS=",.";
   ```

3. 次の問合せを実行します。

   ```
   SELECT TO_CHAR(4000, '9G999D99') "Number" FROM DUAL;
   ```

   結果:

   ```
   Number
   ----------
   ```

```
 4.000,00
```

4. NLS_NUMERIC_CHARACTERSの値を",."(小数点文字はピリオド、グループ・セパレータはカンマ)に変更します。

```
ALTER SESSION SET NLS_NUMERIC_CHARACTERS=".,";
```

5. 次の問合せを実行します。

```
SELECT TO_CHAR(4000, '9G999D99') "Number" FROM DUAL;
```

結果:

```
Number
---------
 4,000.00
```

6. NLS_NUMERIC_CHARACTERSを手順1の値に設定します。

関連項目:

- NLS_NUMERIC_CHARACTERSパラメータの詳細は、『Oracle Databaseグローバリゼーション・サポート・ガイド』を参照してください。
- 「数値および通貨の書式について」
- 「NLSパラメータ値の変更」

親トピック: 各NLSパラメータについて

7.5.9 NLS_CURRENCYパラメータについて

このパラメータは、各国の通貨記号(数値書式要素Lが返す文字列)を指定します。

指定内容: 各国通貨記号(数値書式要素Lで戻される文字列)。

許容可能な値: 任意の有効な通貨記号文字列。

デフォルト値: NLS_TERRITORYによって設定されます。詳細は、「NLS_TERRITORYパラメータについて」を参照してください。

例7-8では、NLS_CURRENCYの値を変更し、NLS_TERRITORYによって設定されたデフォルト値を上書きします。この例をSQL Developerで試す場合は、ワークシートに文や問合せを入力します。ワークシートの詳細は、「SQL Developerにおける問合せの実行」を参照してください。ここに示す結果はSQL*Plusによるもので、SQL Developerとは書式が若干異なります。

例7-8 NLS_CURRENCYによるNLS_TERRITORYの上書き

1. NLS_TERRITORYとNLS_CURRENCYの現在の値を書き留めます。
2. 手順1でNLS_TERRITORYの値がAMERICAでない場合は、次のように変更します。

```
ALTER SESSION SET NLS_TERRITORY=AMERICA;
```

3. 次の問合せを実行します。

```
SELECT TO_CHAR(salary, 'L099G999D99') "Salary"
FROM EMPLOYEES
```

```
WHERE salary > 13000;
```

結果:

```
Salary
----------------------
          $024,000.00
          $017,000.00
          $017,000.00
          $014,000.00
          $013,500.00
```

4. NLS_CURRENCYの値を'©'に変更します。

```
ALTER SESSION SET NLS_CURRENCY='©';
```

5. 次の問合せを実行します。

```
SELECT TO_CHAR(salary, 'L099G999D99') "Salary"
FROM EMPLOYEES
WHERE salary > 13000;
```

結果:

```
Salary
----------------------
          ©024,000.00
          ©017,000.00
          ©017,000.00
          ©014,000.00
          ©013,500.00
```

6. NLS_TERRITORYとNLS_CURRENCYを手順1の値に設定します。

関連項目:

- NLS_CURRENCYパラメータの詳細は、『Oracle Databaseグローバリゼーション・サポート・ガイド』を参照してください。
- 「数値および通貨の書式について」
- 「NLSパラメータ値の変更」

親トピック: 各NLSパラメータについて

7.5.10 NLS_ISO_CURRENCYパラメータについて

このパラメータは、ISO通貨記号(数値書式要素Cが返す文字列)を指定します。

指定内容: ISO通貨記号(数値書式要素Cで戻される文字列)。

許容可能な値: 任意の有効な通貨記号文字列。

デフォルト値: NLS_TERRITORYによって設定されます。詳細は、「NLS_TERRITORYパラメータについて」を参照してください。

各国通貨記号は不明確なことがありますが、ISO通貨記号は一意です。

例7-9では、地域AUSTRALIAおよびAMERICAは各国通貨記号が同じですが、ISO通貨記号は異なることを示します。

この例をSQL Developerで試す場合は、ワークシートに文や問合せを入力します。ワークシートの詳細は、「SQL Developerにおける問合せの実行」を参照してください。ここに示す結果はSQL*Plusによるもので、SQL Developerとは書式が若干異なります。

例7-9 NLS_ISO_CURRENCY

1. NLS_TERRITORYとNLS_ISO_CURRENCYの現在の値を書き留めます。
2. 手順1でNLS_TERRITORYの値がAUSTRALIAでない場合は変更します。

```
ALTER SESSION SET NLS_TERRITORY=AUSTRALIA;
```

3. 次の問合せを実行します。

```
SELECT TO_CHAR(salary, 'L099G999D99') "Local",
       TO_CHAR(salary, 'C099G999D99') "ISO"
FROM EMPLOYEES
WHERE salary > 15000;
```

結果:

```
Local                ISO
-------------------- ------------------
          $024,000.00      AUD024,000.00
          $017,000.00      AUD017,000.00
          $017,000.00      AUD017,000.00
```

4. NLS_TERRITORYの値をAMERICAに変更します。

```
ALTER SESSION SET NLS_TERRITORY=AMERICA;
```

5. 次の問合せを実行します。

```
SELECT TO_CHAR(salary, 'L099G999D99') "Local",
       TO_CHAR(salary, 'C099G999D99') "ISO"
FROM EMPLOYEES
WHERE salary > 15000;
```

結果:

```
Local                ISO
-------------------- ------------------
          $024,000.00      USD024,000.00
          $017,000.00      USD017,000.00
          $017,000.00      USD017,000.00
```

6. NLS_TERRITORYとNLS_ISO_CURRENCYを手順1の値に設定します。

関連項目:

- NLS_ISO_CURRENCYパラメータの詳細は、『Oracle Databaseグローバリゼーション・サポート・ガイド』を参照してください。
- 「数値および通貨の書式について」
- 「NLSパラメータ値の変更」

親トピック: 各NLSパラメータについて

7.5.11 NLS_DUAL_CURRENCYパラメータについて

このパラメータは、二重通貨記号(ユーロ移行期間中にユーロ通貨記号をサポートするために導入)を指定します。

指定項目: 2重通貨記号(ユーロ移行期間中にユーロ通貨記号をサポートするために導入)

許容可能な値: 任意の有効な通貨記号文字列。

デフォルト値: NLS_TERRITORYによって設定されます。詳細は、「NLS_TERRITORYパラメータについて」を参照してください。

関連項目:

- NLS_DUAL_CURRENCYパラメータの詳細は、『Oracle Databaseグローバリゼーション・サポート・ガイド』を参照してください。
- 「数値および通貨の書式について」
- 「NLSパラメータ値の変更」

親トピック: 各NLSパラメータについて

7.5.12 NLS_SORTパラメータについて

このパラメータは、ORDER BY句を持つ問合せの言語ソート順序(照合順序)を指定します。

指定内容: ORDER BY句がある問合せの言語ソート順序(照合順序)。

許容可能な値:

- BINARY

  ソート順序は、データ型に応じて、データベース・キャラクタ・セットまたは各国キャラクタ・セットのバイナリ・ソート順序に基づいています。

- Oracleがサポートする任意の言語ソート名

  ソート順序は、指定した言語ソート名の順序に基づいています。通常、言語ソート名は言語名と同じですが、異なる場合もあります。サポートされている言語ソート名のリストは、『*Oracle Database*グローバリゼーション・サポート・ガイド』を参照してください。

デフォルト値: NLS_LANGUAGEによって設定されます。詳細は、「NLS_LANGUAGEパラメータについて」を参照してください。

例7-10では、2つの異なるNLS_SORTの設定により、同じ問合せの表示結果にどのような影響があるかを示します。設定は、BINARYと従来のスペイン語(SPANISH_M)です。従来のスペイン語では、ch、llおよびñをそれぞれc、lおよびnに続く文字として扱います。

この例をSQL Developerで試す場合は、ワークシートに文や問合せを入力します。ワークシートの詳細は、「SQL Developerにおける問合せの実行」を参照してください。ここに示す結果はSQL*Plusによるもので、SQL Developerとは書式が若干異なります。

大/小文字およびアクセントを区別しないソート

Oracle Databaseの操作では、大/小文字および文字のアクセントが区別されます。大/小文字を区別しないソートを実行するには、NLS_SORTパラメータの値に_CIを付け加えます(BINARY_CIやGERMAN_CIなど)。大/小文字および文字のアクセントのどちらも区別しないソートを実行するには、NLS_SORTパラメータの値に_AIを付け加えます(BINARY_AIやFRENCH_M_AIなど)。

例7-10 NLS_SORTによる言語ソート順序への影響

1. スペイン語の単語の表を作成します。

```
CREATE TABLE temp (name VARCHAR2(15));
```

2. 表にスペイン語の単語を移入します。

```
INSERT INTO temp (name) VALUES ('laguna');
INSERT INTO temp (name) VALUES ('llama');
INSERT INTO temp (name) VALUES ('loco');
```

3. NLS_SORTの現在の値を書き留めます。
4. 手順3でNLS_SORTの値がBINARYでない場合は、次のように変更します。

```
ALTER SESSION SET NLS_SORT=BINARY;
```

5. 次の問合せを実行します。

```
SELECT * FROM temp ORDER BY name;
```

結果:

```
NAME
---------------
laguna
llama
loco
```

6. NLS_SORTの値をSPANISH_M(従来のスペイン語)に変更します。

```
ALTER SESSION SET NLS_SORT=SPANISH_M;
```

7. 手順5の問合せを繰り返します。

結果:

```
NAME
---------------
laguna
loco
llama
```

8. 表を削除します。

```
DROP TABLE temp;
```

9. NLS_SORTを手順3の値に設定します。

関連項目:

- NLS_SORTパラメータの詳細は、『*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。

- 大/小文字およびアクセントを区別しないソートの詳細は、『*Oracle Database*グローバリゼーション・サポート・ガイド』を参照してください。
- 「言語ソートと文字列検索について」
- 「NLSパラメータ値の変更」

親トピック: 各NLSパラメータについて

7.5.13 NLS_COMPパラメータについて

このパラメータは、SQL操作の文字比較動作を指定します。

指定項目: SQL操作の文字比較動作。

許容可能な値:

- BINARY

  SQLで、文字のバイナリ・コードを比較します。バイナリ・コードが高い方の文字が、もう一方の文字よりも大きくなります。
- LINGUISTIC

  SQLで、NLS_SORTパラメータの値に基づいた言語比較を実行します。詳細は、「NLS_SORTパラメータについて」を参照してください。
- ANSI

  この値は、下位互換性のみを目的として用意されています。

デフォルト値: BINARY

例7-11では、NLS_COMP設定に応じた問合せの結果を示します。

この例をSQL Developerで試す場合は、ワークシートに文や問合せを入力します。ワークシートの詳細は、「SQL Developerにおける問合せの実行」を参照してください。ここに示す結果はSQL*Plusによるもので、SQL Developerとは書式が若干異なります。

例7-11 NLS_COMPによるSQL文字比較への影響

1. NLS_SORTとNLS_COMPの現在の値を書き留めます。
2. 手順1で、NLS_SORTとNLS_COMPの値がそれぞれSPANISH_M(従来のスペイン語)とBINARYでない場合は変更します。

   ```
   ALTER SESSION SET NLS_SORT=SPANISH_M NLS_COMP=BINARY;
   ```

3. *次の問合せを実行します。

   ```
   SELECT LAST_NAME FROM EMPLOYEES
   WHERE LAST_NAME LIKE 'C%';
   ```

   結果:

   ```
   LAST_NAME
   -------------------------
   Cabrio
   Cambrault
   Cambrault
   Chen
   Chung
   ```

```
Colmenares

6 rows selected
```

4. NLS_COMPの値をLINGUISTICに変更します。

```
ALTER SESSION SET NLS_COMP=LINGUISTIC;
```

5. 手順3の問合せを繰り返します。

   結果:

```
LAST_NAME
------------------------
Cabrio
Cambrault
Cambrault
Colmenares

4 rows selected
```

   従来のスペイン語ではchをcに続く1文字として扱うため、ここではChenとChungは返されません。

6. NLS_SORTとNLS_COMPを手順1の値に設定します。

関連項目:

- NLS_COMPパラメータの詳細は、『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。
- 「言語ソートと文字列検索について」
- 「NLSパラメータ値の変更」

親トピック: 各NLSパラメータについて

7.5.14 NLS_LENGTH_SEMANTICSパラメータについて

このパラメータは、文字データ型CHAR、VARCHAR2およびLONGの列の長さセマンティクス、つまりこれらの列がバイト単位または文字単位のどちらで指定されているかを指定します。これらの列がバイトまたは文字のいずれかで指定されるかを指定します(パラメータ設定後に宣言された列にのみ適用可能)。

指定広告: 文字データ型CHAR、VARCHAR2、LONGの列の長さセマンティクス。これらの列がバイトまたは文字のいずれかで指定されるかを指定します(パラメータ設定後に宣言された列にのみ適用可能)。

許容可能な値:

- BYTE

  新しいCHAR、VARCHAR2およびLONGの列をバイト単位で指定します。

- CHAR

  新しいCHAR、VARCHAR2およびLONGの列を文字単位で指定します。

デフォルト値: BYTE

この例をSQL Developerで試す場合は、ワークシートに文や問合せを入力します。ワークシートの詳細は、「SQL Developerにおける問合せの実行」を参照してください。ここに示す結果はSQL*Plusによるもので、SQL Developerとは書

式が若干異なります。

例7-12 NLS_LENGTH_SEMANTICSがVARCHAR2列の格納に与える影響

1. NLS_LENGTH_SEMANTICSの現在の値を書き留めます。
2. 手順1で、NLS_LENGTH_SEMANTICSの値がBYTEでない場合は変更します。

```
ALTER SESSION SET NLS_LENGTH_SEMANTICS=BYTE;
```

3. VARCHAR2列のある表を作成します。

```
CREATE TABLE SEMANTICS_BYTE(SOME_DATA VARCHAR2(20));
```

4. 「接続」タブをクリックします。
5. 「接続」フレームで、hr_connを展開します。
6. スキーマ・オブジェクト・タイプのリストで、「表」を展開します。
7. 表のリストで、SEMANTICS_BYTEを選択します。

   「接続」フレームの右側にある「列」ペインに、「列名」 SOME_DATAの「データ型」がVARCHAR2(20 BYTE)であると表示されます。
8. NLS_LENGTH_SEMANTICSの値をCHARに変更します。

```
ALTER SESSION SET NLS_LENGTH_SEMANTICS=CHAR;
```

9. VARCHAR2列のある別の表を作成します。

```
CREATE TABLE SEMANTICS_CHAR(SOME_DATA VARCHAR2(20));
```

10. 「接続」フレームで、「リフレッシュ」アイコンをクリックします。

    今回はSEMANTICS_CHARを含む表のリストが表示されます。
11. SEMANTICS_CHARを選択します。

    「列」ペインに、「列名」 SOME_DATAの「データ型」がVARCHAR2(20 CHAR)であると表示されます。
12. SEMANTICS_BYTEを再び選択します。

    「列」ペインに、「列名」SOME_DATAの「データ名データ型」がVARCHAR2(20 BYTE)であると表示されます。
13. NLS_LENGTH_SEMANTICSを手順1の値に設定します。

関連項目:

- NLS_LENGTH_SEMANTICSパラメータの詳細は、『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。
- 「長さセマンティクスについて」
- 「NLSパラメータ値の変更」

親トピック: 各NLSパラメータについて

7.6 グローバルなアプリケーションでのUnicodeの使用方法

Unicodeデータは挿入および取得できます。データはデータベースおよびクライアント間で透過的に変換され、クライアント・プログラムがデータベース・キャラクタ・セットおよび各国キャラクタ・セットから独立していることを保証します。

- SQLおよびPL/SQLにおけるUnicode文字列リテラルの表現
  SQLとPL/SQLでUnicode文字列リテラルを表現する方法は3つあります。
- キャラクタ・セット変換中のデータ消失の回避
  SQLまたはPL/SQL文の一部として、リテラル(接頭辞Nあり、またはなし)が、文の残りとして、同じキャラクタ・セットでエンコードされます。クライアント側では、文はNLS_LANGパラメータで決定されるクライアントのキャラクタ・セットでエンコードされます。サーバー側では、文はデータベースのキャラクタ・セットでエンコードされます。

関連項目:

- Unicodeを使用したSQLおよびPL/SQLのプログラミングの詳細は、『*Oracle Database*グローバリゼーション・サポート・ガイド』を参照してください。
- Unicodeを使用したプログラミングの一般的な情報は、『*Oracle Database*グローバリゼーション・サポート・ガイド』を参照してください。

親トピック: グローバル環境での作業

7.6.1 SQLおよびPL/SQLにおけるUnicode文字列リテラルの表現

SQLとPL/SQLでUnicode文字列リテラルを表現する方法は3つあります。

SQLとPL/SQLでUnicode文字列リテラルを表現する方法は3つあります。

- N'string'

  例: N'résumé'。

  制限: 「キャラクタ・セット変換中のデータ消失の回避」を参照してください。

- NCHR(number)

  SQLファンクションNCHRは、各国語キャラクタ・セットでバイナリがnumberに相当する文字を返します。返される文字のデータ型はNVARCHAR2です。

  例: NCHR(36)は、デフォルトの各国語キャラクタ・セットであるAL16UTF16では、$を表します。

  制限: NCHR(number)の値の移植性は、同じ各国語キャラクタ・セットを使用するアプリケーションに限定されます。

- UNISTR('string')

  SQLファンクションUNISTRでは、stringを各国語キャラクタ・セットに変換します。

  移植性とデータ保護のために、stringに含めるのはASCII文字とUnicodeエンコーディング値のみとすることをお薦めします。Unicodeエンコーディング値は¥xxxxという形式をとり、xxxxはUCS-2エンコーディング形式で文字コード値を表す16進値です。

  例: UNISTR('G¥0061ry')は'Gary'を表します。

  ASCII文字は、データベース・キャラクタ・セットに変換されてから、各国語キャラクタ・セットに変換されます。Unicodeエンコーディング値は、各国語キャラクタ・セットに直接変換されます。

関連項目:

- Unicode文字列リテラルの詳細は、『*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。
- NCHRファンクションの詳細は、『Oracle Database SQL言語リファレンス』を参照してください。
- UNISTRファンクションの詳細は、『Oracle Database SQL言語リファレンス』を参照してください。

親トピック: グローバルなアプリケーションでのUnicodeの使用方法

7.6.2 キャラクタ・セット変換中のデータ消失の回避

SQLまたはPL/SQL文の一部として、リテラル(接頭辞Nあり、またはなし)が、文の残りとして、同じキャラクタ・セットでエンコードされます。クライアント側では、文はNLS_LANGパラメータで決定されるクライアントのキャラクタ・セットでエンコードされます。サーバー側では、文はデータベースのキャラクタ・セットでエンコードされます。

SQLまたはPL/SQL文がクライアントからデータベースに送信されると、それに応じてキャラクタ・セットが変換されます。クライアントがテキスト・リテラルで使用しているすべての文字がデータベース・キャラクタ・セットに含まれていない場合、データはこの変換で消失します。NCHAR文字列リテラルはデータベース・キャラクタ・セットに依存しないよう設計されているため、CHARテキスト・リテラルよりも影響を受けやすくなります。

互換性のないデータベース・キャラクタ・セットへの変換時のデータ消失を回避するために、NCHARリテラル置換機能を有効にすることができます。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。

親トピック: グローバルなアプリケーションでのUnicodeの使用方法

8 有効なアプリケーションの作成

有効なアプリケーションはスケーラブルで、推奨されるプログラミング・プラクティスとセキュリティ・プラクティスを使用します。

- スケーラブルなアプリケーションの作成
  ユーザー移入およびデータ量にかかわらず、同じリソースを使用し、システム・リソースに過負荷をかけないようにアプリケーションを設計してください。
- 推奨されるプログラミング・プラクティス
  次の推奨されるプログラミング・プラクティスを使用します。
- 推奨されるセキュリティ・プラクティス
  アプリケーションを構成するスキーマ・オブジェクトに対する権限を付与するときは、**最小限の権限という原則**を使用します。

関連項目:

Oracle Database用に最適化されたアプリケーションを作成およびデプロイする方法の詳細は、*『Oracle Database開発ガイド』*を参照してください。

8.1 スケーラブルなアプリケーションの作成

ユーザー移入およびデータ量にかかわらず、同じリソースを使用し、システム・リソースに過負荷をかけないようにアプリケーションを設計してください。

- スケーラブルなアプリケーションについて
  スケーラブルなアプリケーションは、システム・リソースの使用量の増加に比例して、より多くのワークロードを処理できます。
- バインド変数を使用したスケーラビリティの向上
  バインド変数を正しく使用すると、効率的でスケーラブルなアプリケーションを開発できます。
- PL/SQLを使用したスケーラビリティの向上
  特定のPL/SQL機能を使用して、アプリケーションのスケーラビリティを向上させることができます。
- 同時実行性とスケーラビリティについて
  同時実行性とは、複数のトランザクションを同時に実行することです。アプリケーションの処理の同時実行性が高くなれば、拡張性も高くなります。**スケーラブルな**アプリケーションは、システム・リソースの使用量の増加に比例して、より多くのワークロードを処理できます。
- 同時セッション数の制限
  同時セッション数が増えると、同時実行に基づく待機が増加し、レスポンス時間が遅くなります。
- Runstatsによるプログラミング手法の比較
  Runstatsツールを使用すると、2つのプログラミング手法のパフォーマンスを比較して、いずれの手法が優れているかを確認できます。
- Real-World Performanceおよびデータ処理手法
  データ・ウェアハウス環境内のデータベース・アプリケーションでの一般的なタスクは、大きなデータ・セットの問合せと変更です。アプリケーション開発者の課題は、大きなデータ・セットを処理する際に高パフォーマンスを実現する方法です。

親トピック: 有効なアプリケーションの作成

8.1.1 スケーラブルなアプリケーションについて

スケーラブルなアプリケーションは、システム・リソースの使用量の増加に比例して、より多くのワークロードを処理できます。

スケーラブルなアプリケーションは、システム・リソースの使用量の増加に比例して、より多くのワークロードを処理できます。たとえば、ワークロードが倍増した場合、スケーラブルなアプリケーションでは、システム・リソースが2倍使用されます。

非スケーラブルなアプリケーションでは、システム・リソースが完全に消耗されるため、アプリケーション・ワークロードを増やした場合、スループットは向上しません。非スケーラブルなアプリケーションの場合、スループットが固定され、レスポンス時間が低下します。

リソースの消耗の例:

- ハードウェアの消耗
- ディスクの入力/出力(I/O)不足を必然的に引き起こす、大規模トランザクションでの表スキャン
- ネットワークおよびスケジューリングのボトルネックを引き起こす過剰なネットワーク・リクエスト
- メモリー割当てによって、ページングとスワッピングが発生する場合
- プロセスやスレッドの過剰な割当てによって、オペレーティング・システムのスラッシングが発生する場合

ユーザー移入およびデータ量にかかわらず、同じリソースを使用し、システム・リソースに過負荷をかけないようにアプリケーションを設計してください。

親トピック: スケーラブルなアプリケーションの作成

8.1.2 バインド変数を使用したスケーラビリティの向上

バインド変数を正しく使用すると、効率的でスケーラブルなアプリケーションを開発できます。

バインド変数はSQL文の中のプレースホルダで、SQL文を正常に実行するために、有効な値または値のアドレスと置換される必要があります。バインド変数を使用すると、実行時に入力データまたはパラメータを受け取るSQL文を作成できます。

サブプログラムがパラメータを持ち、その値をインボーカが指定するのと同じように、SQL文はバインド変数のプレースホルダを持ち、その値(バインド変数と呼ばれる)は実行時に指定されます。サブプログラムがコンパイルされると、様々なパラメータで何度も実行されるのと同じように、バインド変数のプレースホルダを持つSQL文は、ハード解析されると、様々なバインド変数でソフト解析されます。

最適化および行ソース生成を含む**ハード解析**は、CPUに大きく負担をかける処理です。**最適化と行ソース生成がスキップされ、ただちに実行されるソフト解析**は、同じ文のハード解析より通常大幅に処理が速くなります。(ハード解析とソフト解析の違いを含む、SQL処理の概要は、『*Oracle Database*概要』を参照してください。)

ハード解析はCPUに負担をかける処理であるだけでなく、他の多くの処理と同時に実行できないため非スケーラブルな処理です。同時実行性およびスケーラビリティの詳細は、「同時実行性およびスケーラビリティについて」を参照してください。

例8-1は、バインド変数のない問合せとバインド変数のある意味的に同等の問合せのパフォーマンスの違いを示しています。前者は、速度が遅く、より多くのラッチが使用されます(ラッチがスケーラビリティに与える影響の詳細は、「ラッチおよび同時実行性について」を参照)。パフォーマンス統計を収集し表示するために、この例では、Runstatsツール(「Runstatsによるプログラミング手法の比較」で説明)が使用されています。

注意:

- 例 8-1 は、単一のユーザーのパフォーマンス・コストを示しています。ユーザーが追加されると、コストは急

速に増大します。

- 例 8-1 の結果は、この設定で作成されました。

```
SET SERVEROUTPUT ON FORMAT TRUNCATED
```

注意:

- 文字列リテラルではなくバインド変数を使用すると、SQL インジェクション攻撃に強いコードを最も効率よく作成できます。詳細は、「*Oracle Database PL/SQ 言語 L リファレンス*」を参照してください。

- バインド変数は、データ・ウェアハウス・システムの効率性を損なう場合があります。ほとんどの問合せは非常に多くの時間を要するため、オプティマイザは、最適な汎用問合せではなく、問合せごとに最適なプランの作成を試みます。バインド変数を使用した場合、オプティマイザにより最適な汎用問合せが強制的に作成されることがあります。データ・ウェアハウス・システムのパフォーマンスの向上の詳細は、『*Oracle Database* データ・ウェアハウス・ガイド』を参照してください。

ソフト解析はハード解析よりも効率的ですが、文を何度もソフト解析するコストは依然として非常に高くなります。アプリケーションの効率性およびスケーラビリティを最大化するには、解析を最小化してください。解析を最小化する最も簡単な方法は、PL/SQLの使用です。

例8-1 パフォーマンスを向上させるバインド変数

```
CREATE TABLE t ( x VARCHAR2(5) );

DECLARE
  TYPE rc IS REF CURSOR;
  l_cursor rc;
BEGIN
  runstats_pkg.rs_start;  -- Collect statistics for query without bind variable

  FOR i IN 1 .. 5000 LOOP
    OPEN l_cursor FOR 'SELECT x FROM t WHERE x = ' || TO_CHAR(i);
    CLOSE l_cursor;
  END LOOP;

  runstats_pkg.rs_middle;  -- Collect statistics for query with bind variable

  FOR i IN 1 .. 5000 LOOP
    OPEN l_cursor FOR 'SELECT x FROM t WHERE x = :x' USING i;
    CLOSE l_cursor;
  END LOOP;

  runstats_pkg.rs_stop(500);  -- Stop collecting statistics
end;
/
```

結果は次のようになります。

```
Run 1 ran in 740 hsec
Run 2 ran in 30 hsec
Run 1 ran in 2466.67% of the time of run 2
```

```
Name                                  Run 1         Run 2    Difference
STAT...recursive cpu usage              729            19          -710
STAT...CPU used by this sessio          742            30          -712
STAT...parse time elapsed             1,051             4        -1,047
STAT...parse time cpu                 1,066             2        -1,064
STAT...session cursor cache hi            1         4,998         4,997
STAT...table scans (short tabl        5,000             1        -4,999
STAT...parse count (total)           10,003         5,004        -4,999
LATCH.session idle bit                5,003             3        -5,000
LATCH.session allocation              5,003             3        -5,000
STAT...execute count                 10,003         5,003        -5,000
STAT...opened cursors cumulati       10,003         5,003        -5,000
STAT...parse count (hard)            10,001             5        -9,996
STAT...CCursor + sql area evic       10,000             1        -9,999
STAT...enqueue releases              10,008             7       -10,001
STAT...enqueue requests              10,009             7       -10,002
STAT...calls to get snapshot s       20,005         5,006       -14,999
STAT...calls to kcmgcs               20,028            35       -19,993
STAT...consistent gets pin (fa       20,013            17       -19,996
LATCH.call allocation                20,002             6       -19,996
STAT...consistent gets from ca       20,014            18       -19,996
STAT...consistent gets               20,014            18       -19,996
STAT...consistent gets pin           20,013            17       -19,996
LATCH.simulator hash latch           20,014            11       -20,003
STAT...session logical reads         20,080            75       -20,005
LATCH.shared pool simulator          20,046             5       -20,041
LATCH.enqueue hash chains            20,343            15       -20,328
STAT...recursive calls               40,015        15,018       -24,997
LATCH.cache buffers chains           40,480           294       -40,186
STAT...session pga memory max       131,072        65,536       -65,536
STAT...session pga memory           131,072        65,536       -65,536
LATCH.row cache objects             165,209           139      -165,070
STAT...session uga memory max       219,000             0      -219,000
LATCH.shared pool                   265,108           152      -264,956
STAT...logical read bytes from  164,495,360       614,400  -163,880,960

Run 1 latches total compared to run 2 -- difference and percentage
        Run 1         Run 2          Diff       Pct
      562,092           864      -561,228  2,466.67%

PL/SQL procedure successfully completed.
```

親トピック: スケーラブルなアプリケーションの作成

8.1.3 PL/SQLを使用したスケーラビリティの向上

特定のPL/SQL機能を使用して、アプリケーションのスケーラビリティを改善できます。

- PL/SQLによる解析の最小化の方法
  データベース・アクセスに最適化されたPL/SQLは、文を暗黙的にキャッシュします。PL/SQLでは、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)、実際にはPL/SQLはカーソルを開いたままにし、その文をキャッシュします。
- EXECUTE IMMEDIATE文について
  EXECUTE IMMEDIATE文は、動的SQL文を一度の操作で作成して実行します。
- OPEN FOR文について
  OPEN FOR文の基本構文は次のようになります。

- DBMS_SQLパッケージについて
  DBMS_SQLパッケージは、動的SQL文を作成、実行および説明するためのAPIです。PL/SQLコンパイラがコンパイル時に出力ホスト変数(選択リスト項目)または入力バインド変数の数値またはタイプを決定できない場合に、EXECUTE IMMEDIATE文ではなくDBMS_SQLパッケージを使用する必要があります。
- バルクSQLについて
  バルクSQLは、PL/SQLとSQLの間のラウンド・トリップ数を削減し、これによりリソースの使用が減少します。

親トピック: スケーラブルなアプリケーションの作成

8.1.3.1 PL/SQLによる解析の最小化の方法

データベース・アクセスに最適化されたPL/SQLは、文を暗黙的にキャッシュします。PL/SQLでは、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)、実際にはPL/SQLはカーソルを開いたままにし、その文をキャッシュします。

キャッシュされた文を再度使用する場合、PL/SQLは同じカーソルを使用することで、解析を回避します。(必要に応じてPL/SQLはキャッシュされた文を閉じます。たとえば、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です。)

PL/SQLは、実行時に変化しないSQL文のみをキャッシュできます。

親トピック: PL/SQLを使用したスケーラビリティの向上

8.1.3.2 EXECUTE IMMEDIATE文について

EXECUTE IMMEDIATE文は、動的SQL文を一度の操作で作成して実行します。

EXECUTE IMMEDIATE文の基本構文は次のとおりです。

```
EXECUTE IMMEDIATE sql_statement
```

sql_statementは、SQL文を表す文字列です。sql_statementがEXECUTE IMMEDIATE文の実行ごとに同じ値を持つ場合、PL/SQLはEXECUTE IMMEDIATE文をキャッシュできます。sql_statementがEXECUTE IMMEDIATE文の実行ごとに異なる可能性がある場合、PL/SQLはEXECUTE IMMEDIATE文をキャッシュできません。

関連項目:

- EXECUTE IMMEDIATE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_SQLパッケージについて」

親トピック: PL/SQLを使用したスケーラビリティの向上

8.1.3.3 OPEN FOR文について

OPEN FOR文の基本構文は次のようになります。

OPEN FOR文の基本構文は次のとおりです。

```
OPEN cursor_variable FOR query
```

アプリケーションは、cursor_variableを閉じる前に様々な問合せに対して開くことができます。PL/SQLは、実行時まで様々な問合せの数を特定できないため、PL/SQLはOPEN FOR文をキャッシュできません。

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、宣言されたカーソルを使用してください。詳細は、*Oracle Database*開発ガイドを参照してください。

関連項目:

- OPEN FOR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。
- 「カーソル変数について」
- 「カーソルについて」

親トピック: PL/SQLを使用したスケーラビリティの向上

8.1.3.4 DBMS_SQLパッケージについて

DBMS_SQLパッケージは、動的SQL文を作成、実行および説明するためのAPIです。PL/SQLコンパイラがコンパイル時に出力ホスト変数(選択リスト項目)または入力バインド変数の数値またはタイプを決定できない場合に、EXECUTE IMMEDIATE文ではなくDBMS_SQLパッケージを使用する必要があります。

DBMS_SQLパッケージは、動的SQL文を作成、実行および説明するためのAPIです。DBMS_SQLパッケージの使用はEXECUTE IMMEDIATE文の使用よりも手間がかかりますが、PL/SQLコンパイラがコンパイル時に出力ホスト変数(選択リスト項目)または入力バインド変数の数値またはタイプを決定できない場合にDBMS_SQLパッケージを使用する必要があります。

関連項目:

- DBMS_SQLパッケージをどのようなときに使用するかの詳細は、『*Oracle Database PL/SQL言語リファレンス*』を参照してください。
- DBMS_SQLパッケージの詳細は、『*Oracle Database PL/SQLパッケージおよびタイプ・リファレンス*』を参照してください。
- 「EXECUTE IMMEDIATE文について」

親トピック: PL/SQLを使用したスケーラビリティの向上

8.1.3.5 バルクSQLについて

バルクSQLは、PL/SQLとSQLの間のラウンド・トリップ数を削減し、これによりリソースの使用が減少します。

バルクSQLを使用しない場合、データベースから1度に1行取得し(SQL)、処理して(PL/SQL)、データベースに返します(SQL)。バルクSQLを使用する場合、データベースから行セットを取得し、行セットを処理して、データベースにセット全体を返します。

データベースから複数の行を取得し、データベースに返す場合は、例8-2に示すように、バルクSQLを使用することをお薦めします。次の例に示すように、複数の行を取得するが、それらを返す必要がない場合は、バルクSQLは必要ありません。

```
FOR x IN (SELECT * FROM t WHERE ... )  -- Retrieve row set (implicit array fetch)
  LOOP
    DBMS_OUTPUT.PUT_LINE(t.x);          -- Process rows but do not return them
  END LOOP;
```

例8-2は、表t、列object_nameをループし、100行のセットを取得し、処理して、データベースに返します。(バルクFETCH

文を100行に制限する場合、明示的なカーソルが必要です。)

例8-3は、バルクSQLを使用しないで例8-2と同じジョブを実行します。

例8-2および例8-3のTKPROFレポートが示すとおり、このジョブでバルクSQLを使用すると、CPU時間が約50%減ります。

```
SELECT ROWID RID, OBJECT_NAME FROM T T_BULK

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      721      0.17       0.17          0      22582          0       71825
********************************************************************************
UPDATE T SET OBJECT_NAME = :B1 WHERE ROWID = :B2

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute    719     12.83      13.77          0      71853      74185       71825
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      720     12.83      13.77          0      71853      74185       71825


SELECT ROWID RID, OBJECT_NAME FROM T T_SLOW_BY_SLOW

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total      721      0.17       0.17          0      22582          0       71825
********************************************************************************
UPDATE T SET OBJECT_NAME = :B2 WHERE ROWID = :B1

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute  71824     21.25      22.25          0      71836      73950       71824
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    71825     21.25      22.25          0      71836      73950       71824
```

ただし、このジョブのバルクSQLの使用は、TKPROFレポートに示すとおり、単一のSQL文の使用よりもCPU時間およびコードを多く使用します。

```
UPDATE T SET OBJECT_NAME = SUBSTR(OBJECT_NAME,2) || SUBSTR(OBJECT_NAME,1,1)

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      1.30       1.44          0       2166      75736       71825
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        2      1.30       1.44          0       2166      75736       71825
```

例8-2 バルクSQL

```
CREATE OR REPLACE PROCEDURE bulk AS
  TYPE ridArray IS TABLE OF ROWID;
  TYPE onameArray IS TABLE OF t.object_name%TYPE;

  CURSOR c is SELECT ROWID rid, object_name  -- explicit cursor
              FROM t t_bulk;
```

```
  l_rids    ridArray;
  l_onames  onameArray;
  N         NUMBER := 100;
BEGIN
  OPEN c;
  LOOP
    FETCH c BULK COLLECT
    INTO l_rids, l_onames LIMIT N;    -- retrieve N rows from t

    FOR i in 1 .. l_rids.COUNT
      LOOP                            -- process N rows
        l_onames(i) := substr(l_onames(i),2) || substr(l_onames(i),1,1);
      END LOOP;

      FORALL i in 1 .. l_rids.count  -- return processed rows to t
        UPDATE t
        SET object_name = l_onames(i)
        WHERE ROWID = l_rids(i);
        EXIT WHEN c%NOTFOUND;
  END LOOP;
  CLOSE c;
END;
/
```

例8-3 バルクSQLを使用しない場合

```
CREATE OR REPLACE PROCEDURE slow_by_slow AS
BEGIN
  FOR x IN (SELECT rowid rid, object_name FROM t t_slow_by_slow)
    LOOP
      x.object_name := substr(x.object_name,2) || substr(x.object_name,1,1);

      UPDATE t
      SET object_name = x.object_name
      WHERE rowid = x.rid;
    END LOOP;
END;
```

関連項目:

- バルクSQLの概要は、『*Oracle Database開発ガイド*』を参照してください。
- バルクSQLをいつ使用するかについての詳細は、『*Oracle Database開発ガイド*』を参照してください。
- バルク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。

親トピック: PL/SQLを使用したスケーラビリティの向上

8.1.4 同時実行性およびスケーラビリティについて

同時実行性とは、複数のトランザクションを同時に実行することです。アプリケーションの処理の同時実行性が高くなれば、拡張性も高くなります。**スケーラブルな**アプリケーションは、システム・リソースの使用量の増加に比例して、より多くのワークロードを処理できます。

同時実行性とは、複数のトランザクションを同時に実行することです。同時実行されるトランザクション内の文は、同じデータを更新できます。アプリケーションの処理の同時実行性が高くなれば、拡張性も高くなります。**スケーラブルな**アプリケーションは、

システム・リソースの使用量の増加に比例して、より多くのワークロードを処理できます。たとえば、ワークロードが倍増した場合、スケーラブルなアプリケーションでは、システム・リソースが2倍使用されます。

同時実行されるトランザクションは、意味のある一貫した結果を作成する必要があります。このため、マルチユーザー・データベースには次の機能が備わっている必要があります。

- **複数のユーザーの同時データ・アクセスを可能にするデータの同時実行性**。
- **ユーザー自身のトランザクションおよび他のユーザーのコミット・トランザクションによる見える変更を含む、一貫したデータ・ビューを各ユーザーに表示する、データの一貫性**。

Oracle Databaseは、複数バージョン一貫性モデル、様々なロック・タイプ、およびトランザクション分離レベルを使用して、データの一貫性を維持します。Oracle Databaseのロック・メカニズムの概要は、『*Oracle Database概要*』を参照してください。Oracle Databaseのトランザクション分離レベルの概要は、『*Oracle Database概要*』を参照してください。

トランザクションが同時に実行されるときに一貫したトランザクション動作を表すために、データベース調査者は、**シリアライズ可能**と呼ばれるトランザクション分離カテゴリを定義しています。**シリアライズ可能トランザクション**は、単一ユーザー・データベースである環境で動作します。シリアライズ可能トランザクションは特定の場合に望ましいですが、作業負荷の99%の場合、読取りコミット済分離で十分です。

Oracle Databaseには、同時実行性およびスケーラビリティを向上させる機能(順序、ラッチ、非ブロック読取り/書込み、および共有SQLなど)があります。

- 順序および同時実行性について
  順序を使用すると、シリアライズ化が不要になり、アプリケーションの同時実行性およびスケーラビリティが向上します。
- ラッチおよび同時実行性について
  ラッチを増やすと、同時実行性に基づく待機が増加するため、スケーラビリティが低下します。
- 非ブロック読取り/書込みおよび同時実行性について
  Oracle Databaseでは、**非ブロック読取り/書込み**を使用すると、問合せを同時に実行し、ブロックまたは停止することなく読取り中のデータを変更できます。非ブロック読取り/書込みでは、あるセッションがデータの変更中に、別のセッションがそのデータを読み取ることができます。
- 共有SQLおよび同時実行性について
  Oracle DatabaseがSQL文を実行可能なオブジェクトにコンパイルすると、そのオブジェクトが存在するかぎりそのオブジェクトを他のセッションが再利用できるようになります。**共有SQL**と呼ばれるこのOracle Database機能を使用すると、リソースに大きな負担をかける操作であるSQL文のコンパイルおよび最適化を、同じSQL文がセッションで使用されるたびに行うのではなく、1度実行するだけで済みます。

関連項目:

データの同時実行性および一貫性の詳細は、『*Oracle Database概要*』を参照してください。

親トピック: スケーラブルなアプリケーションの作成

8.1.4.1 順序および同時実行性について

順序を使用すると、シリアライズ化が不要になり、アプリケーションの同時実行性およびスケーラビリティが向上します。

順序は、複数のユーザーが一意の整数を生成する際に使用するスキーマ・オブジェクトであり、一意の主キーが必要な場合に大変便利です。

順序を使用しない場合、一意の主キー値はプログラム的に作成する必要があります。ユーザーは、最近作成された値を選択し、

増分することで、新しい主キー値を取得します。この手法はトランザクション中にロックが必要で、複数のユーザーが次の主キー値を待機します(つまり、トランザクションのシリアル化)。順序を使用すると、シリアライズ化が不要になり、アプリケーションの同時実行性およびスケーラビリティが向上します。

関連項目:

- 順序への同時アクセスの詳細は、『*Oracle Database概要*』を参照してください。
- 「順序の作成および管理」

親トピック: 同時実行性およびスケーラビリティについて

8.1.4.2 ラッチおよび同時実行性について

ラッチを増やすと、同時実行性に基づく待機が増加するため、スケーラビリティが低下します。

ラッチは、共有データ構造へのマルチユーザー・アクセスを調整する、単純な低レベルのシリアライズ化メカニズムです。ラッチによって、複数のプロセスからアクセスされる共有メモリー・リソースが破損しないように保護されます。

ラッチを増やすと、同時実行性に基づく待機が増加するため、スケーラビリティが低下します。開発中に少し速く実行するアプローチか、より少ないラッチを使用するアプローチのいずれかを使用できる場合は、後者を使用してください。

関連項目:

- ラッチの詳細は、『*Oracle Database概要*』を参照してください。
- 単一オブジェクトに対するラッチに類似するmutexの詳細は、『*Oracle Database概要*』を参照してください。

親トピック: 同時実行性およびスケーラビリティについて

8.1.4.3 非ブロック読取り/書込みおよび同時実行性について

Oracle Databaseでは、**非ブロック読取り/書込み**を使用すると、問合せを同時に実行し、ブロックまたは停止することなく読取り中のデータを変更できます。非ブロック読取り/書込みでは、あるセッションがデータの変更中に、別のセッションがそのデータを読み取ることができます。

親トピック: 同時実行性およびスケーラビリティについて

8.1.4.4 共有SQLおよび同時実行性について

Oracle DatabaseがSQL文を実行可能なオブジェクトにコンパイルすると、そのオブジェクトが存在するかぎりそのオブジェクトを他のセッションが再利用できるようになります。**共有SQL**と呼ばれるこのOracle Database機能を使用すると、リソースに大きな負担をかける操作であるSQL文のコンパイルおよび最適化を、同じSQL文がセッションで使用されるたびに行うのではなく、1度実行するだけで済みます。

関連項目:

共有SQLの詳細は、『*Oracle Database概要*』を参照してください。

親トピック: 同時実行性およびスケーラビリティについて

8.1.5 同時セッション数の制限

同時セッション数が増えると、同時実行に基づく待機が増加し、レスポンス時間が遅くなります。

使用しているコンピュータに*n* CPUコアがある場合、実際には最高で*n*セッションが同時にアクティブになります。追加の同時セッションがそれぞれアクティブになるには、CPUコアが使用可能になるまで待機する必要があります。一部の待機セッションがI/Oのみを待機している場合、同時セッション数が*n*をわずかに超えて増加すると、実行時のパフォーマンスがわずかに向上する場合があります。ただし、同時セッション数が大幅に増加すると、実行時パフォーマンスが著しく低下します。

SESSIONS初期化パラメータは、システム内の同時ユーザーの最大数を決定します。詳細は、『*Oracle Database*リファレンス』を参照してください。

関連項目:

12 CPUコアのコンピュータで同時セッション数を数千から96に減らした場合の効果の動画については、http://www.youtube.com/watch?v=xNDnVOCdvQ0を参照してください。

親トピック: スケーラブルなアプリケーションの作成

8.1.6 Runstatsによるプログラミング手法の比較

Runstatsツールを使用すると、2つのプログラミング手法のパフォーマンスを比較して、いずれの手法が優れているかを確認できます。

- Runstatsについて
  Runstatsツールを使用すると、2つのプログラミング手法のパフォーマンスを比較して、いずれの手法が優れているかを確認できます。
- Runstatsの設定
  Runstatsツールは、ビューおよび一時表を使用するパッケージとして実装されます。
- Runstatsの使用
  このトピックでは、Runstatsツールを使用するための構文を示します。

親トピック: スケーラブルなアプリケーションの作成

8.1.6.1 Runstatsについて

Runstatsツールを使用すると、2つのプログラミング手法のパフォーマンスを比較して、いずれの手法が優れているかを確認できます。

Runstatsでは、次を測定できます。

- 1/100秒(hsec)単位での各手法の経過時間
- 1つ目の手法の経過時間を2つ目の手法の経過時間に対するパーセントで表した値
- 2つの手法のシステム統計(解析コールなど)
- 2つの手法のラッチ

先行する計測の中で最も重要なことはラッチです(「ラッチおよび同時実行性について」を参照)。

関連項目:

Runstatsの使用例である例8-1

親トピック: Runstatsによるプログラミング手法の比較

8.1.6.2 Runstatsの設定

Runstatsツールは、ビューおよび一時表を使用するパッケージとして実装されます。

> 注意:
>
> 次の手順のステップ 1 では、動的パフォーマンス・ビューV$STATNAME、V$MYSTAT および V$LATCH の SELECT 権限が必要です。この権限を取得できない場合、この権限を持つ別のユーザーにステップ 1 のビューの作成、およびそのビューに対する SELECT 権限の付与を依頼してください。

Runstatsツールを設定する手順:

1. Runstatsで使用されるビューを作成します。

```
CREATE OR REPLACE VIEW stats
AS SELECT 'STAT...' || a.name name, b.value
FROM V$STATNAME a, V$MYSTAT b
WHERE a.statistic# = b.statistic#
UNION ALL
SELECT 'LATCH.' || name, gets
FROM V$LATCH;
```

2. Runstatsで使用される一時表を作成します。

```
DROP TABLE run_stats;

CREATE GLOBAL TEMPORARY TABLE run_stats
( runid VARCHAR2(15),
  name VARCHAR2(80),
  value INT )
ON COMMIT PRESERVE ROWS;
```

3. このパッケージ仕様を作成します。

```
CREATE OR REPLACE PACKAGE runstats_pkg
AS
  PROCEDURE rs_start;
  PROCEDURE rs_middle;
  PROCEDURE rs_stop( p_difference_threshold IN NUMBER DEFAULT 0 );
end;
/
```

   パラメータ**p_difference_threshold**は、Runstatsによって表示される統計の量およびラッチ・データを制御します。Runstatsは、2つの手法の差異が**p_difference_threshold**よりも大きい場合のみデータを表示します。デフォルトでは、Runstatsはすべてのデータを表示します。

4. このパッケージ本体を作成します。

```
CREATE OR REPLACE PACKAGE BODY runstats_pkg
AS
  g_start NUMBER;
  g_run1 NUMBER;
  g_run2 NUMBER;

  PROCEDURE rs_start
  IS
  BEGIN
    DELETE FROM run_stats;

    INSERT INTO run_stats
    SELECT 'before', stats.* FROM stats;

    g_start := DBMS_UTILITY.GET_TIME;
  END rs_start;

  PROCEDURE rs_middle
  IS
  BEGIN
    g_run1 := (DBMS_UTILITY.GET_TIME - g_start);

    INSERT INTO run_stats
    SELECT 'after 1', stats.* FROM stats;

    g_start := DBMS_UTILITY.GET_TIME;
  END rs_middle;

  PROCEDURE rs_stop( p_difference_threshold IN NUMBER DEFAULT 0 )
  IS
  BEGIN
    g_run2 := (DBMS_UTILITY.GET_TIME - g_start);

    DBMS_OUTPUT.PUT_LINE
      ('Run 1 ran in ' || g_run1 || ' hsec');

    DBMS_OUTPUT.PUT_LINE
      ('Run 2 ran in ' || g_run2 || ' hsec');

    DBMS_OUTPUT.PUT_LINE
      ('Run 1 ran in ' || round(g_run1/g_run2*100, 2) || '% of the time of run 2');

    DBMS_OUTPUT.PUT_LINE( CHR(9) );

    INSERT INTO run_stats
    SELECT 'after 2', stats.* FROM stats;

    DBMS_OUTPUT.PUT_LINE
      ( RPAD( 'Name', 30 ) ||
        LPAD( 'Run 1', 14) ||
        LPAD( 'Run 2', 14) ||
        LPAD( 'Difference', 14)
      );

    FOR x IN
    ( SELECT RPAD( a.name, 30 ) ||
             TO_CHAR( b.value - a.value, '9,999,999,999' ) ||
             TO_CHAR( c.value - b.value, '9,999,999,999' ) ||
             TO_CHAR( ( (c.value - b.value) - (b.value - a.value)),
               '9,999,999,999' ) data
```

```
      FROM run_stats a, run_stats b, run_stats c
      WHERE a.name = b.name
        AND b.name = c.name
        AND a.runid = 'before'
        AND b.runid = 'after 1'
        AND c.runid = 'after 2'
        AND (c.value - a.value) > 0
        AND abs((c.value - b.value) - (b.value - a.value)) >
          p_difference_threshold
     ORDER BY ABS((c.value - b.value) - (b.value - a.value))
    ) LOOP
        DBMS_OUTPUT.PUT_LINE( x.data );
    END LOOP;

    DBMS_OUTPUT.PUT_LINE( CHR(9) );

    DBMS_OUTPUT.PUT_LINE(
      'Run 1 latches total compared to run 2 -- difference and percentage' );

    DBMS_OUTPUT.PUT_LINE
      ( LPAD( 'Run 1', 14) ||
        LPAD( 'Run 2', 14) ||
        LPAD( 'Diff', 14) ||
        LPAD( 'Pct', 10)
      );

    FOR x IN
    ( SELECT TO_CHAR( run1, '9,999,999,999' ) ||
             TO_CHAR( run2, '9,999,999,999' ) ||
             TO_CHAR( diff, '9,999,999,999' ) ||
             TO_CHAR( ROUND( g_run1/g_run2*100, 2), '99,999.99' ) || '%' data
      FROM ( SELECT SUM (b.value - a.value) run1,
                    SUM (c.value - b.value) run2,
                    SUM ( (c.value - b.value) - (b.value - a.value)) diff
             FROM run_stats a, run_stats b, run_stats c
             WHERE a.name = b.name
               AND b.name = c.name
               AND a.runid = 'before'
               AND b.runid = 'after 1'
               AND c.runid = 'after 2'
               AND a.name like 'LATCH%'
           )
    ) LOOP
        DBMS_OUTPUT.PUT_LINE( x.data );
    END LOOP;

  END rs_stop;

END;
/
```

関連項目:

- 「ビューの作成」
- 「表の作成」
- 「チュートリアル: パッケージ仕様部の作成」

- 「チュートリアル: パッケージ本体の作成」
- 動的パフォーマンス・ビューの詳細は、『*Oracle Database*リファレンス』を参照してください。

親トピック: Runstatsによるプログラミング手法の比較

8.1.6.3 Runstatsの使用

このトピックでは、Runstatsツールを使用するための構文を示します。

Runstatsを使用して2つのプログラミング手法を比較するには、次の構文を使用してrunstats_pkgプロシージャを匿名ブロックから起動します。

```
[ DECLARE local_declarations ]
BEGIN
  runstats_pkg.rs_start;
  code_for_first_technique
  runstats_pkg.rs_middle;
  code_for_second_technique
  runstats_pkg.rs_stop(n);
END;
/
```

関連項目:

Runstatsの使用例である例8-1

親トピック: Runstatsによるプログラミング手法の比較

8.1.7 Real-World Performanceおよびデータ処理手法

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・セットを問い合せているか変更しています。アプリケーション開発者の課題は、大きなデータ・セットを処理する際に高パフォーマンスを実現する方法です。

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、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。

主要な内容は次のとおりです。

- 繰返しデータ処理について
  繰返し処理では、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。
- セット・ベース処理について
  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。

親トピック: スケーラブルなアプリケーションの作成

8.1.7.1 繰返しデータ処理について

繰返し処理では、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。

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。

繰返し処理: 行ごと

行ごとの処理で、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だとします。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。
- 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。
- 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。

関連項目: RWP #7 セット・ベース処理

繰返し処理: 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。

機能要件は例X-X: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だとします。次のPL/SQLブロックでは、この要件を満たすために配列手法を使用しています。

```
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。

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。
- 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。

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。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。

繰返し処理: 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・プロセスが作業を分割し、並列に実行できるようにします。

行ごとおよび配列の例と同じ機能要件だとします。主な違いを次に示します。

- スキャン・イベント・レコードは大量のフラット・ファイルに格納されます。
- 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せます。
- 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。

次のPL/SQLコードでは、手動の並列度を使用しています。

```
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;
```

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。バケット内のファイル名ごとに、プログラムによって外部表の場所がこのファイル名に設定されます。次にプログラムではバッチ処理を使用して外部表を問合せ、ステージング表に挿入した後に、コミットします。

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。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取り、サブセットから同じステージング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。
- 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。つまり、並列プロセスは同時に終了できます。

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。
- 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。
- 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。
- 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。

関連項目: RWP #8: セット・ベース・パラレル処理

親トピック: Real-World Performanceおよびデータ処理手法

8.1.7.2 セット・ベース処理について

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。

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。

- Real-World Performanceデモおよびクラスで示しているように、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。
- 処理速度の大幅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。
- 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。
- 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。
- 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。

セット・ベース処理には、次のようないくつかの潜在的なデメリットがあります。

- 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る場合があります。
- 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・コードを完全に書き直すことになります。

関連項目: 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--データ重複除外、RWP #10: セット・ベース処理--データ変換、およびRWP #11: セット・ベース処理--データ集計

親トピック: Real-World Performanceおよびデータ処理手法

8.2 推奨されるプログラミング・プラクティス

次の推奨されるプログラミング・プラクティスを使用します。

- インスツルメンテーション・パッケージの使用
  Oracle Databaseにはインスツルメンテーション・パッケージが用意されており、このパッケージに含まれるサブプログラムを使用して、必要なときにいつでもアプリケーション・トレース情報を生成できます。このトレース情報を使用すると、デバッガを使用せずにアプリケーションをデバッグでき不正なコードを特定できます。
- 統計の収集およびアプリケーション・トレース
  データベース統計ではデータベースの負荷のタイプ、およびデータベースで使用する内部および外部リソースに関する情報が提供されます。ADDMを使用したデータベースでパフォーマンスの問題を正確に診断するには、統計が使用可能である必要があります。
- 既存の機能の使用
  既存の機能を使用しているアプリケーションは、そうではないアプリケーションよりも開発およびメンテナンスが簡単であり、また実行速度も速くなります。
- エディショニング・ビューによるデータベース表のカバー

アプリケーションでデータベース表が使用されている場合、停止時間を最小限にする、または発生させないように、エディションベース再定義(EBR)を使用して、使用中のアプリケーションのデータベース・コンポーネントをアップグレードできるように各表をエディショニング・ビューでカバーします。

親トピック: 有効なアプリケーションの作成

8.2.1 インスツルメンテーション・パッケージの使用

Oracle Databaseにはインスツルメンテーション・パッケージが用意されており、このパッケージに含まれるサブプログラムを使用して、必要なときにいつでもアプリケーション・トレース情報を生成できます。このトレース情報を使用すると、デバッガを使用せずにアプリケーションをデバッグでき不正なコードを特定できます。

インスツルメンテーションは、アプリケーションに多数の機能を提供しているため、オーバーヘッドではありません。オーバーヘッドは、多くの利点を失わずに削除できる部分です。

Oracle Databaseが提供しているインスツルメンテーション・パッケージの一部を次に示します。

- DBMS_APPLICATION_INFO: システム管理者は、アプリケーションのパフォーマンスをモジュール別に追跡できます。

  DBMS_APPLICATION_INFOの詳細は、『*Oracle Database PL/SQLパッケージおよびタイプ・リファレンス*』を参照してください。

- DBMS_SESSION: アプリケーションが、セッション情報にアクセスし、プリファレンスとセキュリティ・レベルを設定できるようにします。

  DBMS_SESSION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_FILE: アプリケーションが、オペレーティング・システムのテキスト・ファイルを読取りおよび書込みできるようにします。

  UTL_FILE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。

関連項目:

Oracle Databaseに含まれるPL/SQLパッケージの概要は『*Oracle Database PL/SQLパッケージおよびタイプ・リファレンス*』を参照してください。

親トピック: 推奨されるプログラミング・プラクティス

8.2.2 統計の収集およびアプリケーション・トレース

データベース統計ではデータベースの負荷のタイプ、およびデータベースで使用する内部および外部リソースに関する情報が提供されます。ADDMを使用したデータベースでパフォーマンスの問題を正確に診断するには、統計が使用可能である必要があります。

統計の収集の詳細は、『*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。

注意:

Oracle Enterprise Manager を使用できない場合、『Oracle Database PL/SQL パッケージおよびタイ

プ・リファレンス』の記載に従い、DBMS_MONITOR サブプログラムを使用して統計を収集できます。

Oracle Databaseには、Oracle Databaseアプリケーションの監視および分析に使用できるトレース・ツールがいくつか用意されています。詳細は、『*Oracle Database SQLチューニング・ガイド*』を参照してください。

親トピック: 推奨されるプログラミング・プラクティス

8.2.3 既存機能の使用

既存の機能を使用しているアプリケーションは、そうではないアプリケーションよりも開発およびメンテナンスが簡単であり、また実行速度も速くなります。

アプリケーションを開発するときは、プログラミング言語、オペレーティング・システム、Oracle Database、およびOracle Databaseで提供されているPL/SQLパッケージおよびタイプの既存の機能を可能なかぎり使用してください。

多くの開発者が開発に再利用している既存の機能は次のとおりです。

- **制約**

  制約についての概要は、「表でのデータ整合性の保証」を参照してください。

- **SQLファンクション**(SQLに組み込まれているファンクション)

  SQLファンクションの詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

- **順序**(一意の連続的な値の生成が可能)

  「順序の作成および管理」を参照してください。

- **監査**(選択したユーザーのデータベース・アクションの監視および記録)

  監査の概要は、『*Oracle Databaseセキュリティ・ガイド*』を参照してください。

- **レプリケーション**(分散データベース・システムを構成する複数のデータベースで、表などのデータベース・オブジェクトをコピーし、メンテナンスするプロセス)

  レプリケーションの詳細は、Oracle GoldenGateのドキュメントを参照してください。

- **メッセージ・キューイング**(Webベースのビジネス・アプリケーション間の通信方法)

  Oracle Databaseアドバンスト・キューイング(AQ)の概要は、『*Oracle Databaseアドバンスト・キューイング・ユーザーズ・ガイド*』を参照してください。

- **レコード変更履歴のメンテナンス**

  ワークスペース・マネージャの概要は、『*Oracle Database Workspace Manager開発者ガイド*』を参照してください。

例8-4では、2つの同時トランザクションが、表に格納されたメッセージをデキューします(つまり、各トランザクションが、表の次の未処理行を検出しロックします)。(『Oracle Database PL/SQLパッケージおよびタイプ・リファレンス』に記載されているように)DBMS_AQ.DEQUEUEプロシージャを単純に起動するのではなく、この例では、表のファンクション索引が作成され、次にそのファンクションが各トランザクションで使用され、行が取得されてメッセージが表示されます。

例8-4のコードは、制限されたDBMS_AQ.DEQUEUE呼出しのような機能を実装します。既存の機能(この場合、ファンクション・ベース索引)を使用して節約された開発時間が大きくなる場合があります。

例8-4 同時デキュー・トランザクション

表の作成:

```
DROP TABLE t;
CREATE TABLE t
  ( id             NUMBER PRIMARY KEY,
    processed_flag VARCHAR2(1),
    payload        VARCHAR2(20)
  );
```

表の索引の作成:

```
CREATE INDEX t_idx ON
  t( DECODE( processed_flag, 'N', 'N' ) );
```

表への移入:

```
INSERT INTO t
  SELECT r,
         CASE WHEN MOD(r,2) = 0 THEN 'N' ELSE 'Y' END,
         'payload ' || r
  FROM (SELECT LEVEL r FROM DUAL CONNECT BY LEVEL <= 5);
```

表の表示:

```
SELECT * FROM t;
```

結果:

```
        ID P PAYLOAD
---------- - --------------------
         1 Y payload 1
         2 N payload 2
         3 Y payload 3
         4 N payload 4
         5 Y payload 5

5 rows selected.
```

1つ目のトランザクション:

```
DECLARE
  l_rec t%ROWTYPE;
  CURSOR c IS
    SELECT *
    FROM t
    WHERE DECODE(processed_flag,'N','N') = 'N'
    FOR UPDATE
    SKIP LOCKED;
BEGIN
  OPEN c;

  FETCH c INTO l_rec;

  IF ( c%FOUND ) THEN
    DBMS_OUTPUT.PUT_LINE( 'Got row ' || l_rec.id || ', ' || l_rec.payload );
  END IF;

  CLOSE c;
END;
/
```

結果:

```
Got row 2, payload 2
```

同時トランザクション:

```
DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
  l_rec t%ROWTYPE;
  CURSOR c IS
    SELECT *
    FROM t
    WHERE DECODE(processed_flag,'N','N') = 'N'
    FOR UPDATE
    SKIP LOCKED;
BEGIN
  OPEN c;

  FETCH c INTO l_rec;

  IF ( c%FOUND ) THEN
    DBMS_OUTPUT.PUT_LINE( 'Got row ' || l_rec.id || ', ' || l_rec.payload );
  END IF;

  CLOSE c;
  COMMIT;
END;
/
```

結果:

```
Got row 4, payload 4
```

関連項目:

- (各リリースの)『*Oracle Database*新機能ガイド』
- (各リリースの)『*Oracle Database*概要』

親トピック: 推奨されるプログラミング・プラクティス

8.2.4 エディショニング・ビューによるデータベース表のカバー

アプリケーションでデータベース表が使用されている場合、停止時間を最小限にする、または発生させないように、エディションベース再定義(EBR)を使用して、使用中のアプリケーションのデータベース・コンポーネントをアップグレードできるように各表をエディショニング・ビューでカバーします。

エディション・ベースの再定義の詳細は、『*Oracle Database*開発ガイド』を参照してください。

親トピック: 推奨されるプログラミング・プラクティス

8.3 推奨されるセキュリティ・プラクティス

アプリケーションを構成するスキーマ・オブジェクトの権限を付与する際に、**最小限の権限のみを付与するという原則**に従ってください。

つまり、不慮または不正による無許可のアクティビティの危険性を排除するため、ユーザーと中間層には、それぞれのアクションを実行するために必要最小限の権限を与える必要があります。

関連項目:

コードに対するSQLインジェクション攻撃に最も効果的な方法である、文字列リテラルのかわりにバインド変数を使用する方法の詳細は、「バインド変数を使用したスケーラビリティの向上」を参照してください。

親トピック: 有効なアプリケーションの作成

9 簡易的なOracle Databaseアプリケーションの開発

この簡易なアプリケーションを開発する手順に従って、Oracle Databaseアプリケーションの一般的な開発手順を学習します。

- アプリケーションについて
  アプリケーションには、次の目的、構造、およびネーミング規則があります。
- アプリケーションのスキーマの作成
  この項の手順を使用して、アプリケーションのスキーマを作成します。
- スキーマへの権限の付与
  スキーマに権限を付与するには、SQL文GRANTを使用します。
- スキーマ・オブジェクトの作成およびデータのロード
  この項では、アプリケーションの表、エディショニング・ビュー、トリガーおよび順序の作成方法、表へのデータのロード方法、およびこれらのスキーマ・オブヘクトの権限を必要とするユーザーへの付与方法について説明します。
- employees_pkgパッケージの作成
  この項では、employees_pkgパッケージの作成方法、サブプログラムの動作内容、必要とするユーザーへのパッケージ実行権限の付与方法、およびサブプログラムの起動方法を説明します。
- admin_pkgパッケージの作成
  この項では、admin_pkgパッケージの作成方法、サブプログラムの動作内容、必要とするユーザーへのパッケージ実行権限の付与方法、およびサブプログラムの起動方法を説明します。

9.1 アプリケーションについて

アプリケーションには、次の目的、構造、およびネーミング規則があります。

- アプリケーションの目的
  アプリケーションは、会社内の2種類のユーザーを対象としています。
- アプリケーションの構造
  アプリケーションでは、次のスキーマ・オブジェクトとスキーマが使用されます。
- アプリケーションのネーミング規則
  アプリケーションでは、次のネーミング規則が使用されます。

親トピック: 簡易的なOracle Databaseアプリケーションの開発

9.1.1 アプリケーションの目的

アプリケーションは、会社内の2種類のユーザーを対象としています。

- 一般ユーザー(従業員のマネージャ)
- アプリケーション管理者

一般ユーザーは、次の操作を行えます。

- 指定した部門の従業員の取得
- 指定した従業員の職務履歴の取得
- 指定した従業員の一般情報の表示(名前、部門、職務、マネージャ、給与など)
- 指定した従業員の給与の変更
- 指定した従業員の職務の変更

アプリケーション管理者は、次の操作を行えます。

- 既存の職務のID、タイトルまたは給与範囲の変更
- 新規職務の追加
- 既存部門のID、名前またはマネージャの変更
- 新規部門の追加

親トピック: アプリケーションについて

9.1.2 アプリケーションの構造

アプリケーションでは、次のスキーマ・オブジェクトとスキーマが使用されます。

- アプリケーションのスキーマ・オブジェクト
- アプリケーションのスキーマ

親トピック: アプリケーションについて

9.1.2.1 アプリケーションのスキーマ・オブジェクト

アプリケーションは、次のスキーマ・オブジェクトで構成されています。

- 次のデータを格納している4つの表
    - ジョブ
    - 部門
    - 従業員
    - 従業員の職務履歴
- 表をカバーし、エディションベース再定義(EBR)を使用して、使用中の完成アプリケーションをアップグレードできるようにする、4つのエディショニング・ビュー
- ビジネス・ルールを実施する2つのトリガー
- 新規部門および新規従業員用の一意の主キーを生成する2つの順序
- 2つのパッケージ
    - employees_pkg(一般ユーザー用のアプリケーション・プログラム・インタフェース(API))
    - admin_pkg(アプリケーション管理者用のAPI)

    一般ユーザーおよびアプリケーション管理者は、APIを介してのみアプリケーションにアクセスします。このため、パッケージ・サブプログラムを起動することによってのみデータを変更できます。

関連項目:

- スキーマ・オブジェクトの詳細は、「Oracle Databaseについて」を参照してください。
- EBRの詳細については、『*Oracle Database*開発ガイド』を参照してください。

親トピック: アプリケーションの構造

9.1.2.2 アプリケーションのスキーマ

セキュリティのために、アプリケーションでは、次の5つのスキーマ(またはユーザー)が使用され、それぞれに必要な権限のみが付与されています。

- app_data: パッケージ以外のすべてのスキーマ・オブジェクトを所有し、その表に、サンプル・スキーマHRの表のデータをロードするスキーマ

  パッケージを作成する開発者は、このスキーマでは作業しません。このため、開発者が誤ってアプリケーション・スキーマ・オブジェクトを変更したり削除したりすることはありません。

- app_code: パッケージemployees_pkgのみを所有するスキーマ

  employees_pkgの開発者はこのスキーマで作業します。

- app_admin: パッケージadmin_pkgのみを所有するスキーマ

  admin_pkgの開発者はこのスキーマで作業します。

- app_user: 何も所有せず、employees_pkgのみを実行できる一般アプリケーション・ユーザー

  中間層アプリケーション・サーバーは、接続プールのデータベースにapp_userとして接続します。このスキーマが、SQLインジェクション・バグなどによって危険にさらされた場合、攻撃者は、employees_pkgサブプログラムによって表示と変更が許可されている内容のみを表示および変更できます。攻撃者は、表の削除、権限のエスカレート、スキーマ・オブジェクトの作成または変更など、他の作業は行えません。

- app_admin_user: 何も所有せず、admin_pkgおよびemployees_pkgのみを実行できるアプリケーション管理者

  このスキーマの接続プールは、非常に小さく、権限を持つユーザーのみがアクセスできます。このスキーマが危険にさらされた場合、攻撃者は、 admin_pkgおよびemployees_pkgサブプログラムによって表示と変更が許可されている内容のみを表示および変更できます。

app_userおよびapp_admin_userではなく、アプリケーションは、何も所有せず、employees_pkgおよびadmin_pkgの両方を実行できるスキーマのみを持つとします。このスキーマの接続プールには、一般ユーザーとアプリケーション管理者の両方に対して十分な大きさが必要です。SQLインジェクション・バグがemployees_pkgにあった場合、そのバグを利用した一般ユーザーはadmin_pkgにアクセスできます。

app_data、app_codeおよびapp_adminではなく、アプリケーションは、パッケージを含むすべてのスキーマ・オブジェクトを所有するスキーマを1つのみ持つとします。この場合、パッケージは、不要な権限と好ましくない権限の両方を含む、表のすべての権限を持ちます。

たとえば、監査証跡の表AUDIT_TRAILを使用しているとします。employees_pkgの開発者はAUDIT_TRAILへの書込みはできるが、読取りや変更はできないようにするとします。admin_pkgの開発者はAUDIT_TRAILの読取りと書込みはできるが、変更はできないようにするとします。AUDIT_TRAIL、employees_pkgおよびadmin_pkgが同じスキーマに属している場合、2つのパッケージの開発者はAUDIT_TRAILに対するすべての権限を持ちます。ただし、AUDIT_TRAILがapp_dataに属し、employees_pkgがapp_codeに属し、admin_pkgがapp_adminに属す場合、app_dataとしてデータベースにアクセスし、次を実行できます。

```
GRANT INSERT ON AUDIT_TRAIL TO app_code;
GRANT INSERT, SELECT ON AUDIT_TRAIL TO app_admin;
```

関連項目:

- スキーマの詳細は、「Oracle Databaseについて」を参照してください。

- サンプル・スキーマHRの詳細は、「サンプル・スキーマHRについて」を参照してください。
- 「推奨されるセキュリティ・プラクティス」

親トピック: アプリケーションの構造

9.1.3 アプリケーションのネーミング規則

アプリケーションでは、次のネーミング規則が使用されます。

アイテム	名前
表	*table#*
*table#*のエディションニング・ビュー	*table*
エディションニング・ビュー*table* のトリガー	*table*_{a|b}*event*[_fer] • a は、AFTER トリガーを示します。 • b は、BEFORE トリガーを示します。 • fer は、FOR EACH ROW トリガーを示しています。 • *event* は、トリガーを起動するイベントを示します。たとえば、INSERT の場合 i、INSERT または UPDATE の場合 iu、DELETE の場合 d です。
table#内の PRIMARY KEY 制約	*table*_pk
table#.column の *NOT* NULL 制約	*table_column*_not_null 脚注 1
table#.*column* の UNIQUE 制約	*table_column*_unique 脚注 1
table#.*column* の CHECK 制約	*table_column*_check 脚注 1
table1#.*column* から table2#.*column* への REF 制約	*table1*_to_*table2*_fk 脚注 1
table1#.*column1* から table2#.*column2* への REF 制約	*table1_col1*_to_*table2_col2*_fk 脚注 1 脚注 2
*table#*の順序	*table_sequence*
パラメータ名	p_*name*

アイテム	名前
ローカル変数名	l_*name*

脚注1

table、*table1*および*table2*は、employeesの場合はempに、departmentsの場合はdeptに、job_historyの場合はjob_histに短縮されます。

脚注2

*col1*および*col2*は、列名*column1*および*column2*の略語です。制約名は、30文字以下にする必要があります。

親トピック: アプリケーションについて

9.2 アプリケーションのスキーマの作成

この項の手順を使用して、アプリケーションのスキーマを作成します。

スキーマの名前は次のとおりです。

- app_data
- app_code
- app_admin
- app_user
- app_admin_user

> **注意:**
>
> 次の手順では、CREATE USER および DROP USER システム権限を持つユーザーの名前とパスワードが必要です。

スキーマ(またはユーザー)*schema_name*を作成する手順:

1. SQL*Plusを使用して、CREATE USERおよびDROP USERシステム権限を持つユーザーとしてOracle Databaseに接続します。

   SQL>プロンプトが表示されます。

2. スキーマが存在する場合、スキーマとそのオブジェクトを次のSQL文を使用して削除します。

   ```
   DROP USER schema_name CASCADE;
   ```

   スキーマが存在していた場合、システムは次のように応答します。

   ```
   User dropped.
   ```

   スキーマが存在しなかった場合、システムは次のように応答します。

   ```
   DROP USER schema_name CASCADE
             *
   ERROR at line 1:
   ```

```
ORA-01918: user 'schema_name' does not exist
```

3. *schema_name*が、app_data、app_codeまたはapp_adminのいずれかの場合、次のSQL文を使用してスキーマを作成します。

```
CREATE USER schema_name IDENTIFIED BY password
DEFAULT TABLESPACE USERS
QUOTA UNLIMITED ON USERS
ENABLE EDITIONS;
```

それ以外の場合、次のSQL文を使用してスキーマを作成します。

```
CREATE USER schema_name IDENTIFIED BY password
ENABLE EDITIONS;
```

注意:

安全なパスワードを選択してください。セキュアなパスワードの詳細は、『*Oracle Database セキュリティ・ガイド*』を参照してください。

システムは次のように応答します。

```
User created.
```

4. (オプション)「SQL DeveloperからOracle Databaseへの接続」の説明に従って、SQL Developerで、スキーマの接続を作成します。

関連項目:

- 「アプリケーションについて」
- 「SQL*PlusからOracle Databaseへの接続」
- DROP USER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 USER文の詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

親トピック: 簡易的なOracle Databaseアプリケーションの開発

9.3 スキーマへの権限の付与

スキーマに権限を付与するには、SQL文GRANTを使用します。

SQL*PlusまたはSQL Developerのワークシートのいずれかでで GRANT文を入力できます。セキュリティのために、各スキーマには必要な権限のみを付与してください。

- app_dataスキーマへの権限の付与
- app_codeスキーマへの権限の付与
- app_adminスキーマへの権限の付与
- app_userおよびapp_admin_userスキーマへの権限の付与

関連項目:

- 「アプリケーションについて」
- 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。

親トピック: 簡易的なOracle Databaseアプリケーションの開発

9.3.1 app_dataスキーマへの権限の付与

app_dataスキーマには、次を行う権限のみを付与します。

- Oracle Databaseに接続します。

```
GRANT CREATE SESSION TO app_data;
```

- アプリケーションの表、ビュー、トリガーおよび順序の作成

```
GRANT CREATE TABLE, CREATE VIEW, CREATE TRIGGER, CREATE SEQUENCE TO app_data;
```

- サンプル・スキーマHRの4つの表からそれ自体の表へのデータのロード

```
GRANT SELECT ON HR.DEPARTMENTS TO app_data;
GRANT SELECT ON HR.EMPLOYEES TO app_data;
GRANT SELECT ON HR.JOB_HISTORY TO app_data;
GRANT SELECT ON HR.JOBS TO app_data;
```

親トピック: スキーマへの権限の付与

9.3.2 app_codeスキーマへの権限の付与

app_codeスキーマには、次を行う権限のみを付与します。

- Oracle Databaseに接続します。

```
GRANT CREATE SESSION TO app_code;
```

- パッケージemployees_pkgの作成

```
GRANT CREATE PROCEDURE TO app_code;
```

- シノニムの作成(利便性のため):

```
GRANT CREATE SYNONYM TO app_code;
```

親トピック: スキーマへの権限の付与

9.3.3 app_adminスキーマへの権限の付与

app_adminスキーマには、次を行う権限のみを付与します。

- Oracle Databaseに接続します。

```
GRANT CREATE SESSION TO app_admin;
```

- パッケージadmin_pkgの作成

```
GRANT CREATE PROCEDURE TO app_admin;
```

- シノニムの作成(利便性のため):

```
GRANT CREATE SYNONYM TO app_admin;
```

親トピック: スキーマへの権限の付与

9.3.4 app_userおよびapp_admin_userスキーマへの権限の付与

app_userおよびapp_admin_userスキーマには、次を行う権限のみを付与します。

- Oracle Databaseに接続します。

```
GRANT CREATE SESSION TO app_user;
GRANT CREATE SESSION TO app_admin_user;
```

- シノニムの作成(利便性のため)

```
GRANT CREATE SYNONYM TO app_user;
GRANT CREATE SYNONYM TO app_admin_user;
```

親トピック: スキーマへの権限の付与

9.4 スキーマ・オブジェクトの作成およびデータのロード

この項では、アプリケーションの表、エディショニング・ビュー、トリガーおよび順序の作成方法、表へのデータのロード方法、およびこれらのスキーマ・オブヘクトの権限を必要とするユーザーへの付与方法について説明します。

スキーマ・オブジェクトの作成およびデータのロード手順:

1. Oracle Databaseにユーザーapp_data.として接続します

   手順は、「SQL*PlusからOracle Databaseへの接続」または「SQL DeveloperからOracle Databaseへの接続」を参照してください。

2. データをロードした後に追加する必要がある外部キー制約を除き、すべての必要な制約を使用して表を作成します。
3. エディショニング・ビューを作成します。
4. トリガーを作成します。
5. 順序を作成します。
6. データを表にロードします。
7. 外部キー制約を追加します。

- 表の作成
- エディショニング・ビューの作成
- トリガーの作成
- 順序の作成
- データのロード
- 外部キー制約の追加
- ユーザーへのスキーマ・オブジェクトの権限の付与

親トピック: 簡易的なOracle Databaseアプリケーションの開発

9.4.1 表の作成

この項では、データをロードした後に追加する必要がある制約を除き、すべての必要な制約を使用してアプリケーションの表を作成する方法を示します。

注意:

Oracle Database にはユーザーapp_data として接続する必要があります。

次の手順では、SQL*PlusまたはSQL Developerのワークシートのいずれかで文を入力できます。または、SQL Developerツールの Create Tableを使用して表を作成できます。

表を作成する手順:

1. 社内の職務に関する情報を格納するjobs#を作成します(職務ごとに1行)。

```
CREATE TABLE jobs#
( job_id      VARCHAR2(10)
              CONSTRAINT jobs_pk PRIMARY KEY,
  job_title   VARCHAR2(35)
              CONSTRAINT jobs_job_title_not_null NOT NULL,
  min_salary  NUMBER(6)
              CONSTRAINT jobs_min_salary_not_null NOT NULL,
  max_salary  NUMBER(6)
              CONSTRAINT jobs_max_salary_not_null NOT NULL
)
/
```

2. 社内の部門に関する情報を格納するdepartments#を作成します(部門ごとに1行)。

```
CREATE TABLE departments#
( department_id    NUMBER(4)
                   CONSTRAINT departments_pk PRIMARY KEY,
  department_name  VARCHAR2(30)
                   CONSTRAINT department_name_not_null NOT NULL
                   CONSTRAINT department_name_unique UNIQUE,
  manager_id       NUMBER(6)
)
/
```

3. 社内の従業員に関する情報を格納するemployees#を作成します(従業員ごとに1行)。

```
CREATE TABLE employees#
( employee_id     NUMBER(6)
                  CONSTRAINT employees_pk PRIMARY KEY,
  first_name      VARCHAR2(20)
                  CONSTRAINT emp_first_name_not_null NOT NULL,
  last_name       VARCHAR2(25)
                  CONSTRAINT emp_last_name_not_null NOT NULL,
  email_addr      VARCHAR2(25)
                  CONSTRAINT emp_email_addr_not_null NOT NULL,
  hire_date       DATE
                  DEFAULT TRUNC(SYSDATE)
                  CONSTRAINT emp_hire_date_not_null NOT NULL
                  CONSTRAINT emp_hire_date_check
                    CHECK(TRUNC(hire_date) = hire_date),
```

```
  country_code    VARCHAR2(5)
                  CONSTRAINT emp_country_code_not_null NOT NULL,
  phone_number    VARCHAR2(20)
                  CONSTRAINT emp_phone_number_not_null NOT NULL,
  job_id          CONSTRAINT emp_job_id_not_null NOT NULL
                  CONSTRAINT emp_jobs_fk REFERENCES jobs#,
  job_start_date  DATE
                  CONSTRAINT emp_job_start_date_not_null NOT NULL,
                  CONSTRAINT emp_job_start_date_check
                    CHECK(TRUNC(JOB_START_DATE) = job_start_date),
  salary          NUMBER(6)
                  CONSTRAINT emp_salary_not_null NOT NULL,
  manager_id      CONSTRAINT emp_mgr_to_empno_fk REFERENCES employees#,
  department_id   CONSTRAINT emp_to_dept_fk REFERENCES departments#
)
/
```

REF制約の理由は次のとおりです。

- 従業員には既存の職務が必要です。つまり、列employees#.job_idの値は、列jobs#.job_idの値でもある必要があります。
- 従業員には、従業員でもあるマネージャが必要です。つまり、列employees#.manager_idの値は、列employees#.employee_idの値でもある必要があります。
- 従業員は、既存の部門で勤務している必要があります。つまり、列employees#.department_idの値は、列departments#.department_idの値でもある必要があります。

また、従業員のマネージャは、従業員が勤務している部門のマネージャである必要があります。つまり、列employees#.manager_idの値は、列departments#.manager_idの値でもある必要があります。ただし、employees#がまだ存在していなかったため、departments#を作成したときに、必要な制約を指定できませんでした。したがって、外部キー制約をdepartments#に後で追加する必要があります(「外部キー制約の追加」を参照)。

4. 社内の従業員の職務履歴を格納するjob_history#を作成します(従業員の職務ごとに1行。)

```
CREATE TABLE job_history#
( employee_id  CONSTRAINT job_hist_to_employees_fk REFERENCES employees#,
  job_id       CONSTRAINT job_hist_to_jobs_fk REFERENCES jobs#,
  start_date   DATE
               CONSTRAINT job_hist_start_date_not_null NOT NULL,
  end_date     DATE
               CONSTRAINT job_hist_end_date_not_null NOT NULL,
  department_id
            CONSTRAINT job_hist_to_departments_fk REFERENCES departments#
            CONSTRAINT job_hist_dept_id_not_null NOT NULL,
            CONSTRAINT job_history_pk PRIMARY KEY(employee_id,start_date),
            CONSTRAINT job_history_date_check CHECK( start_date < end_date )
)
/
```

REF制約の理由は、従業員、職務および部門が存在している必要があることです。つまり、次のようになります。

- 列job_history#.employee_idの値は、列employees#.employee_idの値でもある必要があります。
- 列job_history#.job_idの値は、列jobs#.job_idの値でもある必要があります。
- 列job_history#.department_idの値は、列departments#.department_idの値でもある必要があります。

関連項目:

「表の作成」

親トピック: スキーマ・オブジェクトの作成およびデータのロード

9.4.2 エディションニング・ビューの作成

注意:

Oracle Database にはユーザーapp_data として接続する必要があります。

エディションニング・ビューを作成するには、次の文を使用します(順序は任意)。SQL*PlusまたはSQL Developerのワークシートのいずれかで文を入力できます。または、SQL Developerのビューの作成ツールを使用して表を作成できます。

```
CREATE OR REPLACE EDITIONING VIEW jobs AS SELECT * FROM jobs#
/
CREATE OR REPLACE EDITIONING VIEW departments AS SELECT * FROM departments#
/
CREATE OR REPLACE EDITIONING VIEW employees AS SELECT * FROM employees#
/
CREATE OR REPLACE EDITIONING VIEW job_history AS SELECT * FROM job_history#
/
```

注意:

アプリケーションは、必ずエディションニング・ビューを介して元表を参照する必要があります。そうではない場合、エディションニング・ビューによって表はカバーされず、EBR を使用して完成アプリケーションをその使用中にアップグレードできません。

関連項目:

- 「ビューの作成」
- エディションニング・ビューの一般的な情報は、『*Oracle Database*開発ガイド』を参照してください。
- エディションニング・ビューを使用するためのアプリケーションの準備の詳細は、『*Oracle Database*開発ガイド』を参照してください。

親トピック: スキーマ・オブジェクトの作成およびデータのロード

9.4.3 トリガーの作成

注意:

Oracle Database にはユーザーapp_data として接続する必要があります。

アプリケーション内のトリガーによって、ビジネス・ルールが実施されます。

- 職務*j*の従業員は、職務*j*の最低給与および最高給与の範囲内の給与を得ている必要があります。
- 職務*j*の従業員の給与が*s*の場合、*j*の最低給与を*s*を超える値、または*j*の最高給与を*s*未満の値に変更できません。(これを行うと、既存のデータが無効になります。)

- 1つ目のビジネス・ルールを実施するトリガーの作成
  1つ目のビジネス・ルールは、職務*j*の従業員は、職務*j*の最低給与および最高給与の範囲内の給与を得ている必要があるというものです。
- 2つ目のビジネス・ルールを実施するトリガーの作成
  2つ目のビジネス・ルールは、職務*j*の従業員の給与が*s*の場合、*j*の最低給与を*s*を超える値、または*j*の最高給与を*s*未満の値に変更できないというものです。(これを行うと、既存のデータが無効になります。)

関連項目:

トリガーの詳細は、「トリガーの使用」を参照してください。

親トピック: スキーマ・オブジェクトの作成およびデータのロード

9.4.3.1 1つ目のビジネス・ルールを実施するトリガーの作成

1つ目のビジネス・ルールは、職務*j*の従業員は、職務*j*の最低給与および最高給与の範囲内の給与を得ている必要があるというものです。

このルールは、新規行がemployees表に挿入されるとき、またはemployees表のsalaryまたはjob_id列が更新されたときに違反となる可能性があります。

ルールを実施するには、エディショニング・ビューemployeesに次のトリガーを作成します。SQL*PlusまたはSQL Developerのワークシートのいずれかでで CREATE TRIGGER文を入力できます。または、SQL DeveloperツールのCreate Triggerを使用してトリガーを作成できます。

```
CREATE OR REPLACE TRIGGER employees_aiufer
AFTER INSERT OR UPDATE OF salary, job_id ON employees FOR EACH ROW
DECLARE
  l_cnt NUMBER;
BEGIN
  LOCK TABLE jobs IN SHARE MODE;  -- Ensure that jobs does not change
                                  -- during the following query.
  SELECT COUNT(*) INTO l_cnt
  FROM jobs
  WHERE job_id = :NEW.job_id
  AND :NEW.salary BETWEEN min_salary AND max_salary;
```

```
  IF (l_cnt<>1) THEN
    RAISE_APPLICATION_ERROR( -20002,
      CASE
        WHEN :new.job_id = :old.job_id
        THEN 'Salary modification invalid'
        ELSE 'Job reassignment puts salary out of range'
      END );
  END IF;
END;
/
```

LOCK TABLE jobs IN SHARE MODEは、トリガーが問合せしている表jobsを他のユーザーが変更できないようにします。トリガーは、employeesを変更中に、非ブロック読取りによりその他のユーザーによってjobsに行われた変更を確認できなくなるため(および、ユーザーはトリガーによりemployeesに行われた変更を確認できなくなるため)、問合せ中にjobsを変更できなくする必要があります。

問合せ中にjobsが変更されないようにするもう1つの方法は、SELECT文にFOR UPDATE句を含めることです。ただし、SELECT FOR UPDATEでは、LOCK TABLE jobs IN SHARE MODEよりも同時実行性が制限されます。

LOCK TABLE jobs IN SHARE MODEは、jobsをユーザーが変更できないようにしますが、共有モードのjobs自体はロックされません。通常、jobsへの変更は、employeesへの変更よりもまれにしか行われません。このため、共有モードのjobsをロックすることにより、排他モードのjobsの単一行をロックすることよりも同時接続性が高くなります。

関連項目:

- IN SHARE MODEでの表のロックの詳細は、『*Oracle Database*開発ガイド』を参照してください。
- SELECT FOR UPDATE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。
- 「トリガーの作成」
- employees_aiuferトリガーの動作の詳細は、「チュートリアル: employees_pkgサブプログラムの動作内容の表示」を参照してください。

親トピック: トリガーの作成

9.4.3.2 2つ目のビジネス・ルールを実施するトリガーの作成

2つ目のビジネス・ルールは、職務*j*の従業員の給与が*s*の場合、*j*の最低給与を*s*を超える値、または*j*の最高給与を*s*未満の値に変更できないというものです。(これを行うと、既存のデータが無効になります。)

このルールは、jobs表のmin_salaryまたはmax_salary列が更新されたときに違反となる可能性があります。

ルールを実施するには、エディショニング・ビューjobsに次のトリガーを作成します。SQL*PlusまたはSQL Developerのワークシートのいずれかでで CREATE TRIGGER文を入力できます。または、SQL DeveloperツールのCreate Triggerを使用してトリガーを作成できます。

```
CREATE OR REPLACE TRIGGER jobs_aufer
AFTER UPDATE OF min_salary, max_salary ON jobs FOR EACH ROW
WHEN (NEW.min_salary > OLD.min_salary OR NEW.max_salary < OLD.max_salary)
DECLARE
  l_cnt NUMBER;
BEGIN
  LOCK TABLE employees IN SHARE MODE;
```

```
  SELECT COUNT(*) INTO l_cnt
  FROM employees
  WHERE job_id = :NEW.job_id
  AND salary NOT BETWEEN :NEW.min_salary and :NEW.max_salary;

  IF (l_cnt>0) THEN
    RAISE_APPLICATION_ERROR( -20001,
      'Salary update would violate ' || l_cnt || ' existing employee records' );
  END IF;
END;
/
```

LOCK TABLE employees IN SHARE MODEは、トリガーが問合せしている表employeesを他のユーザーが変更できないようにします。トリガーは、jobsを変更中に、非ブロック読取りによりその他のユーザーによってemployeesに行われた変更を確認できなくなるため(および、ユーザーはトリガーによりjobsに行われた変更を確認できなくなるため)、問合せ中にemployeesを変更できなくする必要があります。

このトリガーの場合、SELECT FOR UPDATEは、LOCK TABLE IN SHARE MODEの代替手段になりません。ユーザーがこの職務の給与範囲の変更を試行したときに、このトリガーは、他のユーザーが給与を新規範囲外に変更しないようにする必要があります。このため、トリガーは、このjob_idを持つemployees表のすべての行をロックし、このjob_idを持つように変更される可能性のあるすべての行をロックする必要があります。

LOCK TABLE employees IN SHARE MODEの代替手段の1つは、DBMS_LOCKパッケージを使用して、job_idの名前で名前付きロックを作成し、employees表とjobs表の両方でトリガーを使用して、この名前付きロックを使用し同時更新を回避する方法です。ただし、DBMS_LOCKおよび複数のトリガーを使用すると、実行時のパフォーマンスに悪影響を及ぼします。

LOCK TABLE employees IN SHARE MODEの代替手段の1つは、employeesの各変更行について、対応するjobsの職務行をロックするトリガーをemployees表で作成することです。ただし、この手法では、頻繁に起こるemployees表の更新時に大量の作業が発生します。

LOCK TABLE employees IN SHARE MODEは、前述の代替手段よりも単純で、jobs表の変更はまれであり、通常、表をロックすることがユーザーにとって不便ではない、アプリケーションのメンテナンス時に発生します。

関連項目:

- SHARE MODEでの表のロックの詳細は、『*Oracle Database開発ガイド*』を参照してください。
- DBMS_LOCK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。
- 「トリガーの作成」
- 「チュートリアル: admin_pkgサブプログラムの動作内容の表示」

親トピック: トリガーの作成

9.4.4 順序の作成

注意:

Oracle Database にはユーザーapp_data として接続する必要があります。

新規部門および新規従業員用の一意の主キーを生成する順序を作成するには、次の文を(いずれかの順番で)使用しますSQL*PlusまたはSQL Developerのワークシートのいずれかで文を入力できます。または、SQL Developerツールの Create Sequenceを使用して順序を作成できます。

```
CREATE SEQUENCE employees_sequence START WITH 210;
CREATE SEQUENCE departments_sequence START WITH 275;
```

サンプル・スキーマHRの表からロードするデータとの競合を回避するために、employees_sequenceおよびdepartments_sequenceの開始番号は、employees.employee_idとdepartments.department_idの最大値をそれぞれ超える値である必要があります。データのロード後、この問合せにより、これらの最大値が表示されます。

```
SELECT MAX(e.employee_id), MAX(d.department_id)
FROM employees e, departments d;
```

結果:

```
MAX(E.EMPLOYEE_ID) MAX(D.DEPARTMENT_ID)
------------------ --------------------
               206                  270
```

関連項目:

「順序の作成および管理」

親トピック: スキーマ・オブジェクトの作成およびデータのロード

9.4.5 データのロード

注意:

Oracle Database にはユーザーapp_data として接続する必要があります。

サンプル・スキーマHRの表のデータをアプリケーションの表にロードします。

注意:

次の手順は、エディショニング・ビューを介してアプリケーションの表を参照します。

次の手順では、SQL*PlusまたはSQL Developerのワークシートのいずれかで文を入力できます。

表にデータをロードする手順:

1. 表HR.JOBSのデータをjobsにロードします。

```
INSERT INTO jobs (job_id, job_title, min_salary, max_salary)
SELECT job_id, job_title, min_salary, max_salary
  FROM HR.JOBS
/
```

結果:

```
19 rows created.
```

2. 表HR.DEPARTMENTSのデータをdepartmentsにロードします。

```
INSERT INTO departments (department_id, department_name, manager_id)
SELECT department_id, department_name, manager_id
  FROM HR.DEPARTMENTS
/
```

結果:

```
27 rows created.
```

3. HR.phone_numberからemployees.country_codeおよびemployees.phone_numberを取得する検索CASE式およびSQLファンクション、およびHR.JOB_HISTORYからemployees.job_start_dateを取得するSQLファンクションおよびスカラー副問合せを使用して、表HR.EMPLOYEESおよびHR.JOB_HISTORYのデータをemployeesにロードします。

```
INSERT INTO employees (employee_id, first_name, last_name, email_addr,
  hire_date, country_code, phone_number, job_id, job_start_date, salary,
  manager_id, department_id)
SELECT employee_id, first_name, last_name, email, hire_date,
  CASE WHEN phone_number LIKE '011.%'
    THEN '+' || SUBSTR( phone_number, INSTR( phone_number, '.' )+1,
      INSTR( phone_number, '.', 1, 2 ) -  INSTR( phone_number, '.' ) - 1 )
    ELSE '+1'
  END country_code,
  CASE WHEN phone_number LIKE '011.%'
    THEN SUBSTR( phone_number, INSTR(phone_number, '.', 1, 2 )+1 )
    ELSE phone_number
  END phone_number,
  job_id,
  NVL( (SELECT MAX(end_date+1)
        FROM HR.JOB_HISTORY jh
        WHERE jh.employee_id = employees.employee_id), hire_date),
  salary, manager_id, department_id
  FROM HR.EMPLOYEES
/
```

結果:

```
107 rows created.
```

注意:

前述の INSERT 文は、「1 つ目のビジネス・ルールを実施するトリガーの作成」で作成されたトリガーを起動します。

4. 表HR.JOB_HISTORYのデータをjob_historyにロードします。

```
INSERT INTO job_history (employee_id, job_id, start_date, end_date,
  department_id)
SELECT employee_id, job_id, start_date, end_date, department_id
  FROM HR.JOB_HISTORY
/
```

結果:

```
10 rows created.
```

5. 変更をコミットします。

```
COMMIT;
```

関連項目:

- 「INSERT文について」
- 「サンプル・スキーマHRについて」
- 「問合せにおけるCASE式の使用」
- NVLファンクションの詳細は、「問合せにおけるNULL関連ファンクションの使用」を参照してください。
- SQLファンクションの詳細は、『*Oracle Database SQL言語リファレンス*』を参照してください。

親トピック: スキーマ・オブジェクトの作成およびデータのロード

9.4.6 外部キー制約の追加

注意:

Oracle Database にはユーザーapp_data として接続する必要があります。

表departmentsおよびemployeesにデータをロードしたら、次のALTER TABLE文を使用して外部キー制約を追加します。SQL*PlusまたはSQL Developerのワークシートのいずれかで文を入力できます。または、SQL DeveloperツールのAdd Foreign Keyを使用して制約を追加できます。

```
ALTER TABLE departments#
ADD CONSTRAINT dept_to_emp_fk
FOREIGN KEY(manager_id) REFERENCES employees#;
```

departments#およびemployees#にデータをロードする前に、この外部キー制約を追加すると、いずれかにデータをロードしようとしたときに次のエラーが表示されます。

```
ORA-02291: integrity constraint (APP_DATA.JOB_HIST_TO_DEPT_FK) violated - parent key not found
```

関連項目:

「チュートリアル: 既存の表への制約の追加」

親トピック: スキーマ・オブジェクトの作成およびデータのロード

9.4.7 ユーザーへのスキーマ・オブジェクトの権限の付与

注意:

Oracle Database にはユーザーapp_data として接続する必要があります。

ユーザーに権限を付与するには、SQL文GRANTを使用します。SQL*PlusまたはSQL DeveloperのワークシートのいずれかでGRANT文を入力できます。

employees_pkgを作成するために必要な権限のみをapp_codeに付与します。

```
GRANT SELECT, INSERT, UPDATE, DELETE ON employees TO app_code;
GRANT SELECT ON departments TO app_code;
GRANT SELECT ON jobs TO app_code;
GRANT SELECT, INSERT on job_history TO app_code;
GRANT SELECT ON employees_sequence TO app_code;
```

admin_pkgを作成するために必要な権限のみをapp_adminに付与します。

```
GRANT SELECT, INSERT, UPDATE, DELETE ON jobs TO app_admin;
GRANT SELECT, INSERT, UPDATE, DELETE ON departments TO app_admin;
GRANT SELECT ON employees_sequence TO app_admin;
GRANT SELECT ON departments_sequence TO app_admin;
```

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。

親トピック: スキーマ・オブジェクトの作成およびデータのロード

9.5 employees_pkgパッケージの作成

この項では、employees_pkgパッケージの作成方法、サブプログラムの動作内容、必要とするユーザーへのパッケージ実行権限の付与方法、およびサブプログラムの起動方法を説明します。

employees_pkgパッケージの作成手順:

1. Oracle Databaseにユーザーapp_code.として接続します

   手順は、「SQL*PlusからOracle Databaseへの接続」または「SQL DeveloperからOracle Databaseへの接続」を参照してください。

2. 次のシノニムを作成します。

```
CREATE OR REPLACE SYNONYM employees FOR app_data.employees;
CREATE OR REPLACE SYNONYM departments FOR app_data.departments;
CREATE OR REPLACE SYNONYM jobs FOR app_data.jobs;
CREATE OR REPLACE SYNONYM job_history FOR app_data.job_history;
```

SQL*PlusまたはSQL Developerのワークシートのいずれかでで CREATE SYNONYM文を入力できます。または、SQL Developerのシノニムの作成ツールを使用して表を作成できます。

3. パッケージ仕様を作成します。
4. パッケージ本体を作成します。

- employees_pkgのパッケージ仕様の作成
- employees_pkgのパッケージ本体の作成
- チュートリアル: employees_pkgサブプログラムの動作内容の表示
  このチュートリアルでは、SQL*Plusを使用して、employees_pkgパッケージのサブプログラムの動作内容を表示します。また、チュートリアルでは、トリガーemployees_aiuferおよびCHECK制約job_history_date_checkの動作内容も表示します。
- app_userおよびapp_admin_userへの実行権限の付与
- チュートリアル: app_userまたはapp_admin_userとしてのget_job_historyの起動
  このチュートリアルでは、SQL*Plusを使用して、サブプログラムapp_code.employees_pkg.get_job_historyをユーザーapp_user (通常はマネージャ)またはapp_admin_user(アプリケーション管理者)として起動する方法を示します。

関連項目:

- 「シノニムの作成」
- 「パッケージについて」

親トピック: 簡易的なOracle Databaseアプリケーションの開発

9.5.1 employees_pkgのパッケージ仕様の作成

注意:

Oracle Database にはユーザーapp_code として接続する必要があります。

マネージャ用のAPIである、employees_pkgのパッケージ仕様を作成するには、次のCREATE PACKAGE文を使用します。SQL*PlusまたはSQL Developerのワークシートのいずれかで文を入力できます。または、SQL Developerツールのパッケージの作成を使用してパッケージを作成できます。

```
CREATE OR REPLACE PACKAGE employees_pkg
AS
  PROCEDURE get_employees_in_dept
    ( p_deptno     IN     employees.department_id%TYPE,
      p_result_set IN OUT SYS_REFCURSOR );

  PROCEDURE get_job_history
```

```
    ( p_employee_id  IN     employees.department_id%TYPE,
      p_result_set   IN OUT SYS_REFCURSOR );

  PROCEDURE show_employee
    ( p_employee_id  IN     employees.employee_id%TYPE,
      p_result_set   IN OUT SYS_REFCURSOR );

  PROCEDURE update_salary
    ( p_employee_id IN employees.employee_id%TYPE,
      p_new_salary  IN employees.salary%TYPE );

  PROCEDURE change_job
    ( p_employee_id IN employees.employee_id%TYPE,
      p_new_job     IN employees.job_id%TYPE,
      p_new_salary  IN employees.salary%TYPE := NULL,
      p_new_dept    IN employees.department_id%TYPE := NULL );

END employees_pkg;
/
```

関連項目:

- 「アプリケーションについて」
- 「パッケージの作成および管理」
- CREATE 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。

親トピック: employees_pkgパッケージの作成

9.5.2 employees_pkgのパッケージ本体の作成

注意:

Oracle Database にはユーザーapp_code として接続する必要があります。

マネージャ用のAPIである、employees_pkgのパッケージ本体を作成するには、次のCREATE PACKAGE BODY文を使用します。SQL*PlusまたはSQL Developerのワークシートのいずれかで文を入力できます。または、SQL DeveloperツールのCreate Bodyを使用してパッケージを作成できます。

```
CREATE OR REPLACE PACKAGE BODY employees_pkg
AS
  PROCEDURE get_employees_in_dept
    ( p_deptno     IN     employees.department_id%TYPE,
      p_result_set IN OUT SYS_REFCURSOR )
  IS
     l_cursor SYS_REFCURSOR;
  BEGIN
    OPEN p_result_set FOR
      SELECT e.employee_id,
        e.first_name || ' ' || e.last_name name,
        TO_CHAR( e.hire_date, 'Dy Mon ddth, yyyy' ) hire_date,
```

```
      j.job_title,
      m.first_name || ' ' || m.last_name manager,
      d.department_name
    FROM employees e INNER JOIN jobs j ON (e.job_id = j.job_id)
      LEFT OUTER JOIN employees m ON (e.manager_id = m.employee_id)
      INNER JOIN departments d ON (e.department_id = d.department_id)
    WHERE e.department_id = p_deptno ;
END get_employees_in_dept;

PROCEDURE get_job_history
  ( p_employee_id  IN     employees.department_id%TYPE,
    p_result_set   IN OUT SYS_REFCURSOR )
IS
BEGIN
  OPEN p_result_set FOR
    SELECT e.First_name || ' ' || e.last_name name, j.job_title,
      e.job_start_date start_date,
      TO_DATE(NULL) end_date
    FROM employees e INNER JOIN jobs j ON (e.job_id = j.job_id)
    WHERE e.employee_id = p_employee_id
    UNION ALL
    SELECT e.First_name || ' ' || e.last_name name,
      j.job_title,
      jh.start_date,
      jh.end_date
    FROM employees e INNER JOIN job_history jh
      ON (e.employee_id = jh.employee_id)
      INNER JOIN jobs j ON (jh.job_id = j.job_id)
    WHERE e.employee_id = p_employee_id
    ORDER BY start_date DESC;
END get_job_history;

PROCEDURE show_employee
  ( p_employee_id  IN     employees.employee_id%TYPE,
    p_result_set   IN OUT sys_refcursor )
IS
BEGIN
  OPEN p_result_set FOR
    SELECT *
    FROM (SELECT TO_CHAR(e.employee_id) employee_id,
            e.first_name || ' ' || e.last_name name,
            e.email_addr,
            TO_CHAR(e.hire_date,'dd-mon-yyyy') hire_date,
            e.country_code,
            e.phone_number,
            j.job_title,
            TO_CHAR(e.job_start_date,'dd-mon-yyyy') job_start_date,
            to_char(e.salary) salary,
            m.first_name || ' ' || m.last_name manager,
            d.department_name
          FROM employees e INNER JOIN jobs j on (e.job_id = j.job_id)
            RIGHT OUTER JOIN employees m ON (m.employee_id = e.manager_id)
            INNER JOIN departments d ON (e.department_id = d.department_id)
          WHERE e.employee_id = p_employee_id)
    UNPIVOT (VALUE FOR ATTRIBUTE IN (employee_id, name, email_addr, hire_date,
      country_code, phone_number, job_title, job_start_date, salary, manager,
      department_name) );
END show_employee;

PROCEDURE update_salary
```

```
    ( p_employee_id IN employees.employee_id%type,
      p_new_salary  IN employees.salary%type )
  IS
  BEGIN
    UPDATE employees
    SET salary = p_new_salary
    WHERE employee_id = p_employee_id;
  END update_salary;

  PROCEDURE change_job
    ( p_employee_id IN employees.employee_id%TYPE,
      p_new_job     IN employees.job_id%TYPE,
      p_new_salary  IN employees.salary%TYPE := NULL,
      p_new_dept    IN employees.department_id%TYPE := NULL )
  IS
  BEGIN
    INSERT INTO job_history (employee_id, start_date, end_date, job_id,
      department_id)
    SELECT employee_id, job_start_date, TRUNC(SYSDATE), job_id, department_id
      FROM employees
      WHERE employee_id = p_employee_id;

    UPDATE employees
    SET job_id = p_new_job,
      department_id = NVL( p_new_dept, department_id ),
      salary = NVL( p_new_salary, salary ),
      job_start_date = TRUNC(SYSDATE)
    WHERE employee_id = p_employee_id;
  END change_job;
END employees_pkg;
/
```

関連項目:

- 「アプリケーションについて」
- 「パッケージの作成および管理」
- CREATE PACKAGE 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。

親トピック: employees_pkgパッケージの作成

9.5.3 チュートリアル: employees_pkgサブプログラムの動作内容の表示

このチュートリアルでは、SQL*Plusを使用して、employees_pkgパッケージのサブプログラムの動作内容を表示します。また、チュートリアルでは、トリガーemployees_aiuferおよびCHECK制約job_history_date_checkの動作内容も表示します。

注意:

Oracle Database には、SQL*Plus でユーザーapp_code として接続する必要があります。

SQL*Plusを使用してemployees_pkgサブプログラムの動作内容を表示する手順:

1. 書式設定コマンドを使用して、出力を読みやすくします。次に例を示します。

```
SET LINESIZE 80
SET RECSEP WRAPPED
SET RECSEPCHAR "="
COLUMN NAME FORMAT A15 WORD_WRAPPED
COLUMN HIRE_DATE FORMAT A20 WORD_WRAPPED
COLUMN DEPARTMENT_NAME FORMAT A10 WORD_WRAPPED
COLUMN JOB_TITLE FORMAT A29 WORD_WRAPPED
COLUMN MANAGER FORMAT A11 WORD_WRAPPED
```

2. サブプログラム・パラメータ**p_result_set**の値のバインド変数を宣言します。

```
VARIABLE c REFCURSOR
```

3. 部門90の従業員を表示します。

```
EXEC employees_pkg.get_employees_in_dept( 90, :c );
PRINT c
```

結果:

```
EMPLOYEE_ID NAME            HIRE_DATE            JOB_TITLE
----------- --------------- -------------------- ---------------------------
MANAGER     DEPARTMENT
----------- ----------
        100 Steven King     Tue Jun 17th, 2003   President
            Executive
================================================================================
        102 Lex De Haan     Sat Jan 13th, 2001   Administration Vice President
Steven King Executive
================================================================================
        101 Neena Kochhar   Wed Sep 21st, 2005   Administration Vice President
Steven King Executive
================================================================================
```

4. 従業員の101職務履歴を表示します。

```
EXEC employees_pkg.get_job_history( 101, :c );
PRINT c
```

結果:

```
NAME            JOB_TITLE                     START_DAT END_DATE
--------------- ----------------------------- --------- ---------
Neena Kochhar   Administration Vice President 16-MAR-05
Neena Kochhar   Accounting Manager            28-OCT-01 15-MAR-05
Neena Kochhar   Public Accountant             21-SEP-97 27-OCT-01
```

5. 従業員101の一般情報を表示します。

```
EXEC employees_pkg.show_employee( 101, :c );
PRINT c
```

結果:

```
ATTRIBUTE       VALUE
--------------- --------------------------------------------
EMPLOYEE_ID     101
NAME            Neena Kochhar
```

```
EMAIL_ADDR      NKOCHHAR
HIRE_DATE       21-sep-2005
COUNTRY_CODE    +1
PHONE_NUMBER    515.123.4568
JOB_TITLE       Administration Vice President
JOB_START_DATE  16-mar-05
SALARY          17000
MANAGER         Steven King
DEPARTMENT_NAME Executive

11 rows selected.
```

6. 職務管理副社長の情報を表示します。

```
SELECT * FROM jobs WHERE job_title = 'Administration Vice President';
```

結果:

```
JOB_ID     JOB_TITLE                     MIN_SALARY MAX_SALARY
---------- ----------------------------- ---------- ----------
AD_VP      Administration Vice President      15000      30000
```

7. 従業員101に、職務の範囲外の新規給与の割当てを試行します。

```
EXEC employees_pkg.update_salary( 101, 30001 );
```

結果:

```
SQL> EXEC employees_pkg.update_salary( 101, 30001 );
BEGIN employees_pkg.update_salary( 101, 30001 ); END;

*
ERROR at line 1:
ORA-20002: Salary modification invalid
ORA-06512: at "APP_DATA.EMPLOYEES_AIUFER", line 13
ORA-04088: error during execution of trigger 'APP_DATA.EMPLOYEES_AIUFER'
ORA-06512: at "APP_CODE.EMPLOYEES_PKG", line 77
ORA-06512: at line 1
```

8. 従業員101に、職務の範囲内の新規給与を割り当て、再度従業員101の一般情報を表示します。

```
EXEC employees_pkg.update_salary( 101, 18000 );
EXEC employees_pkg.show_employee( 101, :c );
PRINT c
```

結果:

```
ATTRIBUTE       VALUE
--------------- ----------------------------------------------
EMPLOYEE_ID     101
NAME            Neena Kochhar
EMAIL_ADDR      NKOCHHAR
HIRE_DATE       21-sep-2005
COUNTRY_CODE    +1
PHONE_NUMBER    515.123.4568
JOB_TITLE       Administration Vice President
JOB_START_DATE  16-mar-05
SALARY          18000
MANAGER         Steven King
DEPARTMENT_NAME Executive
```

```
11 rows selected.
```

9. 従業員101について、現在の職務のまま給与を低くします。

```
EXEC employees_pkg.change_job( 101, 'AD_VP', 17500, 90 );
```

結果:

```
SQL> exec employees_pkg.change_job( 101, 'AD_VP', 17500, 90 );
BEGIN employees_pkg.change_job( 101, 'AD_VP', 17500, 80 ); END;

*
ERROR at line 1:
ORA-02290: check constraint (APP_DATA.JOB_HISTORY_DATE_CHECK) violated
ORA-06512: at "APP_CODE.EMPLOYEES_PKG", line 101
ORA-06512: at line 1
```

10. 従業員に関する情報が表示されます。(給与が前の手順の文で変更され、17500ではなく18000になっていることに注意してください)。

```
exec employees_pkg.show_employee( 101, :c );
print c
```

結果:

```
ATTRIBUTE       VALUE
--------------- --------------------------------------------
EMPLOYEE_ID     101
NAME            Neena Kochhar
EMAIL_ADDR      NKOCHHAR
HIRE_DATE       21-sep-2005
COUNTRY_CODE    +1
PHONE_NUMBER    515.123.4568
JOB_TITLE       Administration Vice President
JOB_START_DATE  10-mar-2015
SALARY          18000
MANAGER         Steven King
DEPARTMENT_NAME Executive

11 rows selected.
```

関連項目:

- SQL*Plusコマンドの詳細は、『*SQL*Plusユーザーズ・ガイドおよびリファレンス*』を参照してください。
- 「パッケージの作成および管理」

親トピック: employees_pkgパッケージの作成

9.5.4 app_userおよびapp_admin_userへの実行権限の付与

注意:

Oracle Database にはユーザーapp_code として接続する必要があります。

パッケージemployees_pkgの実行権限をapp_user(通常はマネージャ)およびapp_admin_user(アプリケーション管理者)に付与するには、次のGRANT文を使用します(順番は任意)。SQL*PlusまたはSQL Developerのワークシートのいずれかで文を入力できます。

```
GRANT EXECUTE ON employees_pkg TO app_user;
GRANT EXECUTE ON employees_pkg TO app_admin_user;
```

関連項目:

- 「アプリケーションのスキーマ」
- 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。

親トピック: employees_pkgパッケージの作成

9.5.5 チュートリアル: app_userまたはapp_admin_userとしてのget_job_historyの起動

このチュートリアルでは、SQL*Plusを使用して、サブプログラムapp_code.employees_pkg.get_job_historyをユーザーapp_user (通常はマネージャ)またはapp_admin_user(アプリケーション管理者)として起動する方法を示します。

app_userまたはapp_admin_userとしてget_job_historyを起動する手順:

1. ユーザーapp_userまたはapp_admin_userとしてOracle DatabaseにSQL*Plusから接続します。

   手順は、「SQL*PlusからOracle Databaseへの接続」を参照してください。

2. 次のシノニムを作成します。

   ```
   CREATE SYNONYM employees_pkg FOR app_code.employees_pkg;
   ```

3. 従業員の101職務履歴を表示します。

   ```
   EXEC employees_pkg.get_job_history( 101, :c );
   PRINT c
   ```

   結果:

   ```
   NAME            JOB_TITLE                     START_DAT END_DATE
   --------------- ----------------------------- --------- ---------
   Neena Kochhar   Administration Vice President 16-MAR-05 15-MAY-12
   Neena Kochhar   Accounting Manager            28-OCT-01 15-MAR-05
   Neena Kochhar   Public Accountant             21-SEP-97 27-OCT-01
   ```

親トピック: employees_pkgパッケージの作成

9.6 admin_pkgパッケージの作成

この項では、admin_pkgパッケージの作成方法、サブプログラムの動作内容、必要とするユーザーへのパッケージの実行権限の付与方法、およびサブプログラムの起動方法を説明します。

admin_pkgパッケージの作成手順:

1. Oracle Databaseにユーザーapp_admin.として接続します

   手順は、「SQL*PlusからOracle Databaseへの接続」または「SQL DeveloperからOracle Databaseへの接続」を参照してください。

2. 次のシノニムを作成します。

   ```
   CREATE SYNONYM departments FOR app_data.departments;
   CREATE SYNONYM jobs FOR app_data.jobs;
   CREATE SYNONYM departments_sequence FOR app_data.departments_sequence;
   ```

   SQL*PlusまたはSQL DeveloperのワークシートのいずれかでCREATE SYNONYM文を入力できます。または、SQL DeveloperツールのCreate Synonymを使用して表を作成できます。

3. パッケージ仕様を作成します。
4. パッケージ本体を作成します。

- admin_pkgのパッケージ仕様の作成
- admin_pkgのパッケージ本体の作成
- チュートリアル: admin_pkgサブプログラムの動作内容の表示
  このチュートリアルでは、SQL*Plusを使用して、admin_pkgパッケージのサブプログラムの動作内容を表示します。また、チュートリアルでは、トリガーjobs_auferの動作内容も示します。
- app_user_adminへの実行権限の付与
- チュートリアル: app_admin_userとしてのadd_departmentの起動
  このチュートリアルでは、SQL*Plusを使用して、ファンクションapp_admin.admin_pkg.add_departmentをユーザーapp_admin_user(アプリケーション管理者)として起動し、新規部門の情報を表示する方法を示します。

関連項目:

- 「シノニムの作成および管理」
- 「パッケージについて」

親トピック: 簡易的なOracle Databaseアプリケーションの開発

9.6.1 admin_pkgのパッケージ仕様の作成

注意:

Oracle Database にはユーザーapp_admin として接続する必要があります。

アプリケーション管理者用のAPIである、admin_pkgのパッケージ仕様を作成するには、次のCREATE PACKAGE文を使用します。SQL*PlusまたはSQL Developerのワークシートのいずれかで文を入力できます。または、SQL Developerツールのパッケージの作成を使用してパッケージを作成できます。

```
CREATE OR REPLACE PACKAGE admin_pkg
AS
```

```
  PROCEDURE update_job
    ( p_job_id      IN jobs.job_id%TYPE,
      p_job_title   IN jobs.job_title%TYPE := NULL,
      p_min_salary  IN jobs.min_salary%TYPE := NULL,
      p_max_salary  IN jobs.max_salary%TYPE := NULL );

  PROCEDURE add_job
    ( p_job_id      IN jobs.job_id%TYPE,
      p_job_title   IN jobs.job_title%TYPE,
      p_min_salary  IN jobs.min_salary%TYPE,
      p_max_salary  IN jobs.max_salary%TYPE );

  PROCEDURE update_department
    ( p_department_id     IN departments.department_id%TYPE,
      p_department_name   IN departments.department_name%TYPE := NULL,
      p_manager_id        IN departments.manager_id%TYPE := NULL,
      p_update_manager_id IN BOOLEAN := FALSE );

  FUNCTION add_department
    ( p_department_name   IN departments.department_name%TYPE,
      p_manager_id        IN departments.manager_id%TYPE )
    RETURN departments.department_id%TYPE;

END admin_pkg;
/
```

関連項目:

- 「アプリケーションについて」
- 「パッケージの作成および管理」
- CREATE 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。

親トピック: admin_pkgパッケージの作成

9.6.2 admin_pkgのパッケージ本体の作成

注意:

Oracle Database にはユーザーapp_admin として接続する必要があります。

アプリケーション管理者用のAPIである、admin_pkgのパッケージ本体を作成するには、次のCREATE PACKAGE BODY文を使用します。SQL*PlusまたはSQL Developerのワークシートのいずれかで文を入力できます。または、SQL DeveloperツールのCreate Bodyを使用してパッケージを作成できます。

```
CREATE OR REPLACE PACKAGE BODY admin_pkg
AS
  PROCEDURE update_job
    ( p_job_id      IN jobs.job_id%TYPE,
      p_job_title   IN jobs.job_title%TYPE := NULL,
      p_min_salary  IN jobs.min_salary%TYPE := NULL,
```

```
      p_max_salary  IN jobs.max_salary%TYPE := NULL )
  IS
  BEGIN
    UPDATE jobs
    SET job_title  = NVL( p_job_title, job_title ),
        min_salary = NVL( p_min_salary, min_salary ),
        max_salary = NVL( p_max_salary, max_salary )
    WHERE job_id = p_job_id;
  END update_job;

  PROCEDURE add_job
    ( p_job_id      IN jobs.job_id%TYPE,
      p_job_title   IN jobs.job_title%TYPE,
      p_min_salary  IN jobs.min_salary%TYPE,
      p_max_salary  IN jobs.max_salary%TYPE )
  IS
  BEGIN
    INSERT INTO jobs ( job_id, job_title, min_salary, max_salary )
    VALUES ( p_job_id, p_job_title, p_min_salary, p_max_salary );
  END add_job;

  PROCEDURE update_department
    ( p_department_id     IN departments.department_id%TYPE,
      p_department_name   IN departments.department_name%TYPE := NULL,
      p_manager_id        IN departments.manager_id%TYPE := NULL,
      p_update_manager_id IN BOOLEAN := FALSE )
  IS
  BEGIN
    IF ( p_update_manager_id ) THEN
      UPDATE departments
      SET department_name = NVL( p_department_name, department_name ),
          manager_id = p_manager_id
      WHERE department_id = p_department_id;
    ELSE
      UPDATE departments
      SET department_name = NVL( p_department_name, department_name )
      WHERE department_id = p_department_id;
    END IF;
  END update_department;

  FUNCTION add_department
    ( p_department_name   IN departments.department_name%TYPE,
      p_manager_id        IN departments.manager_id%TYPE )
    RETURN departments.department_id%TYPE
  IS
    l_department_id departments.department_id%TYPE;
  BEGIN
    INSERT INTO departments ( department_id, department_name, manager_id )
      VALUES ( departments_sequence.NEXTVAL, p_department_name, p_manager_id )
      RETURNING department_id INTO l_department_id;
    RETURN l_department_id;
  END add_department;

END admin_pkg;
/
```

関連項目:

- 「アプリケーションについて」
- 「パッケージの作成および管理」
- CREATE PACKAGE 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。

親トピック: admin_pkgパッケージの作成

9.6.3 チュートリアル: admin_pkgサブプログラムの動作内容の表示

このチュートリアルでは、SQL*Plusを使用して、admin_pkgパッケージのサブプログラムの動作内容を表示します。また、チュートリアルでは、トリガーjobs_auferの動作内容も示します。

注意:

Oracle Database には、SQL*Plus でユーザーapp_admin として接続する必要があります。

admin_pkgサブプログラムの動作内容を表示する手順:

1. IDがAD_VPである職務の情報を表示します。

```
SELECT * FROM jobs WHERE job_id = 'AD_VP';
```

結果:

```
JOB_ID     JOB_TITLE                           MIN_SALARY MAX_SALARY
---------- ----------------------------------- ---------- ----------
AD_VP      Administration Vice President            15000      30000
```

2. この職務の最高給与を高くし、再度情報を表示します。

```
EXEC admin_pkg.update_job( 'AD_VP', p_max_salary => 31000 );
SELECT * FROM jobs WHERE job_id = 'AD_VP';
```

結果:

```
JOB_ID     JOB_TITLE                           MIN_SALARY MAX_SALARY
---------- ----------------------------------- ---------- ----------
AD_VP      Administration Vice President            15000      31000
```

3. IDがIT_PROGである職務の情報を表示します。

```
SELECT * FROM jobs WHERE job_id = 'IT_PROG';
```

結果:

```
JOB_ID     JOB_TITLE                           MIN_SALARY MAX_SALARY
---------- ----------------------------------- ---------- ----------
IT_PROG    Programmer                                4000      10000
```

4. この職務の最低給与を高くすることを試みます。

```
EXEC admin_pkg.update_job( 'IT_PROG', p_max_salary => 4001 );
```

結果(SQL*Plusから)

```
SQL> EXEC admin_pkg.update_job( 'IT_PROG', p_max_salary => 4001 );
BEGIN admin_pkg.update_job( 'IT_PROG', p_max_salary => 4001 ); END;

*
ERROR at line 1:
ORA-20001: Salary update would violate 5 existing employee records
ORA-06512: at "APP_DATA.JOBS_AUFER", line 12
ORA-04088: error during execution of trigger 'APP_DATA.JOBS_AUFER'
ORA-06512: at "APP_ADMIN.ADMIN_PKG", line 10
ORA-06512: at line 1
```

5. 新規職務を追加し、その情報を表示します。

```
EXEC admin_pkg.add_job( 'AD_CLERK', 'Administrative Clerk', 3000, 7000 );
SELECT * FROM jobs WHERE job_id = 'AD_CLERK';
```

結果:

```
JOB_ID     JOB_TITLE                          MIN_SALARY MAX_SALARY
---------- ---------------------------------- ---------- ----------
AD_CLERK   Administrative Clerk                     3000       7000
```

6. 部門100に関する情報を表示します。

```
SELECT * FROM departments WHERE department_id = 100;
```

結果:

```
DEPARTMENT_ID DEPARTMENT_NAME                MANAGER_ID
------------- ------------------------------ ----------
          100 Finance                               108
```

7. 部門100の名前およびマネージャを変更し、その情報を表示します。

```
EXEC admin_pkg.update_department( 100, 'Financial Services' );
EXEC admin_pkg.update_department( 100, p_manager_id => 111,
  p_update_manager_id => true );
SELECT * FROM departments WHERE department_id = 100;
```

結果:

```
DEPARTMENT_ID DEPARTMENT_NAME                MANAGER_ID
------------- ------------------------------ ----------
          100 Financial Services                    111
```

関連項目:

「パッケージの作成および管理」

親トピック: admin_pkgパッケージの作成

9.6.4 app_admin_userへの実行権限の付与

注意:

Oracle Database にはユーザーapp_admin として接続する必要があります。

パッケージadmin_pkgの実行権限をapp_admin_user(アプリケーション管理者)に付与するには、次のGRANT文を使用します。SQL*PlusまたはSQL Developerのワークシートのいずれかで文を入力できます。

```
GRANT EXECUTE ON admin_pkg TO app_admin_user;
```

関連項目:

- 「アプリケーションのスキーマ」
- 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。

親トピック: admin_pkgパッケージの作成

9.6.5 チュートリアル: app_admin_userとしてのadd_departmentの起動

このチュートリアルでは、SQL*Plusを使用して、ファンクションapp_admin.admin_pkg.add_departmentをユーザーapp_admin_user(アプリケーション管理者)として起動し、新規部門の情報を表示する方法を示します。

admin_pkg.add_departmentをapp_admin_userとして起動する手順:

1. ユーザーapp_admin_userとしてOracle DatabaseにSQL*Plusから接続します。

   手順は、「SQL*PlusからOracle Databaseへの接続」を参照してください。

2. 次のシノニムを作成します。

   ```
   CREATE SYNONYM admin_pkg FOR app_admin.admin_pkg;
   ```

3. ファンクションの戻り値のバインド変数を宣言します。

   ```
   VARIABLE n NUMBER
   ```

4. マネージャなしで新規部門を追加します。

   ```
   EXEC :n := admin_pkg.add_department( 'New department', NULL );
   ```

5. 新規部門のマネージャIDを表示します。

   ```
   PRINT :n
   ```

   結果:

   ```
            N
   ----------
         275
   ```

新規部門の情報を表示する手順:

1. Oracle Databaseにユーザーapp_admin.として接続します
2. 新規部門に関する情報を表示します。

```
SELECT * FROM departments WHERE department_name LIKE 'New department%';
```

結果:

```
DEPARTMENT_ID DEPARTMENT_NAME                MANAGER_ID
------------- ------------------------------ ----------
          275 New department
```

親トピック: admin_pkgパッケージの作成

10 Oracle Databaseアプリケーションのデプロイ

アプリケーションを開発した後、他のユーザーが実行できるデプロイメント環境と呼ばれる他のデータベースにインストールできます。

- 開発および開発環境について
  アプリケーションを開発するデータベースは、**開発環境**と呼ばれます。アプリケーションを開発した後、他のユーザーが実行できる**デプロイメント環境**と呼ばれる他のデータベースにインストールできます。
- インストール・スクリプトについて
  インストール・スクリプトには、アプリケーションの作成に必要なすべてのSQL文または他のスクリプトを実行する**マスター・スクリプト**を設定できます。
- インストール・スクリプトの作成
  インストール・スクリプトはテキスト・エディタまたはSQL Developerで作成できます。
- サンプル・アプリケーションのデプロイ
  インストール・スクリプトを使用してサンプル・アプリケーションをデプロイできます。
- インストールの有効性のチェック
  デプロイメント環境にアプリケーションをインストールした後、SQL Developerで次の方法により有効性をチェックできます。
- インストール・スクリプトのアーカイブ
  アプリケーションのインストールが有効であることを確認したら、ソース・コード制御システムでインストール・スクリプトをアーカイブすることをお薦めします。

10.1 開発およびデプロイメント環境について

アプリケーションを開発するデータベースは、**開発環境**と呼ばれます。アプリケーションを開発した後、他のユーザーが実行できる**デプロイメント環境**と呼ばれる他のデータベースにインストールできます。

第一のデプロイメント環境は**テスト環境**です。テスト環境では、アプリケーションの機能を詳細にテストして、正確に構造化されているかどうかを判断し、**本番環境**にデプロイする前に問題を修正できます。

アプリケーションは本番環境へのデプロイ前後に、**教育環境**にもデプロイすることができます。教育環境は、他の環境に影響を及ぼさずにアプリケーションの実行方法を練習する場をユーザーに提供します。

必要なデプロイメント環境が編成内に存在しない場合は、作成できます。

親トピック: Oracle Databaseアプリケーションのデプロイ

10.2 インストール・スクリプトについて

インストール・スクリプトには、アプリケーションの作成に必要なすべてのSQL文または他のスクリプトを実行する**マスター・スクリプト**を設定できます。

スクリプトは、名前が.sqlで終わるファイル(たとえば、create_app.sql)内の一連のSQL文です。SQL*PlusまたはSQL Developerなどのクライアント・プログラムのスクリプトを実行すると、SQL文がスクリプトの表示順に実行されます。SQL文がアプリケーションを作成するスクリプトは、**インストール・スクリプト**と呼ばれます。

アプリケーションをデプロイするには、デプロイメント環境で１つ以上のインストール・スクリプトを実行します。新しいアプリケーションでは、インストール・スクリプトを作成する必要があります。古いアプリケーションでは、インストール・スクリプトが存在する可能性がありますが、存在しない場合は作成できます。

- DDL文とスキーマ・オブジェクトの依存性について

インストール・スクリプトには、スキーマ・オブジェクトを作成するDDL文およびオプションでDDL文が作成する表にデータをロードするINSERT文が含まれます。インストール・スクリプトを正確に作成し、正しい順序で複数のインストール・スクリプトを実行するには、アプリケーションのスキーマ・オブジェクト間の依存性を理解する必要があります。

- INSERT文と制約について
  INSERT文を含むインストール・スクリプトを実行する場合、ソース表(開発環境)のデータをデプロイメント環境の対応する新規表に挿入するときに違反となる可能性のある制約がないか確認する必要があります。

親トピック: Oracle Databaseアプリケーションのデプロイ

10.2.1 DDL文とスキーマ・オブジェクトの依存性について

インストール・スクリプトには、スキーマ・オブジェクトを作成するDDL文およびオプションでDDL文が作成する表にデータをロードするINSERT文が含まれます。インストール・スクリプトを正確に作成し、正しい順序で複数のインストール・スクリプトを実行するには、アプリケーションのスキーマ・オブジェクト間の依存性を理解する必要があります。

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。そのため、Aを作成する前に、Bを作成する必要があります。これを行わないと、オブジェクト・タイプによって、Bを作成する文が失敗したり、Bが無効な状態で作成されます。

複雑なアプリケーションの場合、オブジェクトを作成する順序ははっきりしていません。通常は、データベース設計者に相談するか、設計のダイアグラムを考慮する必要があります。

関連項目:

- スキーマ・オブジェクトの依存性の詳細は、『*Oracle Database*開発ガイド』を参照してください。
- 「データ定義言語(DDL)文について」

親トピック: インストール・スクリプトについて

10.2.2 INSERT文と制約について

INSERT文を含むインストール・スクリプトを実行する場合、ソース表(開発環境)のデータをデプロイメント環境の対応する新規表に挿入するときに違反となる可能性のある制約がないか確認する必要があります。

アプリケーションの各ソース表に対して、ソース表のデータを新規表に挿入したとき、制約を違反するかどうかを判断する必要があります。そのような制約がある場合は、まずその制約を無効にし、データを挿入してから、制約を再度有効にする必要があります。データ項目に制約違反がある場合、そのデータ項目を修正するまで、制約を再度有効にすることはできません。

「データのロード」に示すように、正しい順序で参照データを単純に挿入する場合は、制約違反は起こりません。そのため、制約を無効にしておく必要はありません。

外部のソース(ファイル、スプレッドシート、以前のアプリケーションなど)や、依存データを多く含む大量の表からデータを挿入する場合は、データを挿入する前に制約を無効にしてください。

制約を無効にし、再度有効にするには、次の方法があります。

- SQL Developerを使用して、制約の無効化と再度有効化を一度に実行します。
  1. 「接続」フレームで、適切な表を選択します。
  2. 表名のラベルが付いたペインで、サブタブ「制約」を選択します。
  3. すべての表の制約のリストで、ENABLEDをDISABLED (またはその逆)に変更します。

- インストール・スクリプトを編集して、各制約を無効化し再度有効化するSQL文を追加します。
- 各制約を無効にし、再度有効にするSQL文を含んだSQLスクリプトを作成します。
- Oracle Databaseデータ・ディクショナリで制約を検索して、各制約を無効化および有効化するSQL文を使用してSQLスクリプトを作成します。

  たとえば、「表の作成」からEVALUATIONS、PERFORMANCE_PARTS、およびSCORES表で使用される制約を検索および有効化するには、ワークシートに次の文を入力します。

```
SELECT 'ALTER TABLE '|| TABLE_NAME || ' DISABLE CONSTRAINT '||
  CONSTRAINT_NAME ||';'
  FROM user_constraints
 WHERE table_name IN ('EVALUATIONS','PERFORMANCE_PARTS','SCORES');

SELECT 'ALTER TABLE '|| TABLE_NAME || ' ENABLE CONSTRAINT '||
  CONSTRAINT_NAME ||';'
  FROM user_constraints
 WHERE table_name IN ('EVALUATIONS','PERFORMANCE_PARTS','SCORES');
```

関連項目:

- 「INSERT文について」
- 「表のデータ整合性の保証」

親トピック: インストール・スクリプトについて

10.3 インストール・スクリプトの作成

インストール・スクリプトはテキスト・エディタまたはSQL Developerで作成できます。

インストール・スクリプトでDDLおよびINSERT文のみが必要な場合、SQL Developerまたは任意のテキスト・エディタで作成できます。SQL Developerで、カートまたはデータベース・エクスポート・ウィザードを使用できます。複数のデプロイメント環境で実行する予定のインストール・スクリプトの場合はカート、単一のデプロイメント環境でのみ実行する予定のインストール・スクリプトの場合はデータベース・エクスポート・ウィザードを使用することをお薦めします。

インストール・スクリプトでDDLおよびINSERT文ではないSQL文が必要な場合、テキスト・エディタで作成できます。

この項では、カートおよびデータベース・エクスポート・ウィザードを使用してインストール・スクリプトを作成する方法、順序およびトリガーを作成するインストール・スクリプトを編集するタイミングおよび方法および「単純なOracle Databaseアプリケーションの開発」(「サンプル・アプリケーション」)のアプリケーションのインストール・スクリプトを作成する方法について説明します。

- カートによるインストール・スクリプトの作成
  SQL Developerのカートは、1つ以上のデータベース接続から宛先接続にOracle Databaseオブジェクトをデプロイするための便利なツールです。
- データベース・エクスポート・ウィザードによるインストール・スクリプトの作成
  データベース・エクスポート・ウィザードを使用してSQL Developerでインストール・スクリプトを作成するには、インストール・スクリプトの名前、エクスポートするオブジェクトとデータおよび必要なオプションを指定すると、ウィザードがインストール・スクリプトを生成します。
- 順序を作成するインストール・スクリプトの編集
  アプリケーションが一意のキーを生成する順序を使用しており、ソース表から関連する新規表にデータを挿入しない場

合、インストール・スクリプトのSTART WITH値を編集できます。

- トリガーを作成するインストール・スクリプトの編集
  ソース表にBEFORE INSERTトリガーがあるアプリケーションで、ソース表から関連する新規表にデータを挿入する場合、インストール・スクリプトの各INSERT文による新規表へのデータの挿入の前に、トリガーを起動するかどうか決定する必要があります。
- サンプル・アプリケーションのインストール・スクリプトの作成
  サンプル・アプリケーションのインストール・スクリプトを作成できます。

親トピック: Oracle Databaseアプリケーションのデプロイ

10.3.1 カートによるインストール・スクリプトの作成

SQL Developerカートは、1つ以上のデータベース接続から宛先接続にOracle Databaseオブジェクトをデプロイするための便利なツールです。

ナビゲータ・フレームから「カート」ウィンドウにオブジェクトをドラッグ・アンド・ドロップし、必要なオプションを指定し、「カートのエクスポート」アイコンをクリックして「オブジェクトのエクスポート」ダイアログ・ボックスを表示します。このダイアログ・ボックスに情報を入力した後、SQL Developerは、スクリプト(マスター・スクリプトを含む)を含む.zipファイルを作成して、必要な宛先の接続のスキーマのオブジェクトを作成します。

カートを使用してインストール・スクリプトを作成するには、次の手順を実行します。

1. SQL Developerウィンドウで、「表示」メニューをクリックします。
2. 「表示」メニューから、「カート」を選択します。

   カート・ウィンドウが開きます。「カートのエクスポート」アイコンは、非アクティブ(グレー)です。

   ヒント:

   「カート」ウィンドウで、カート・ユーザーのプリファレンスの情報を取得するには、[F1]キーを押します。

3. 「接続」フレームで、インストール・スクリプトで作成するスキーマ・オブジェクトを選択し、「カート」ウィンドウにドラッグします。

   「カート」ウィンドウで、「カートのエクスポート」アイコンがアクティブ(グレーではない)になります。

4. インストール・スクリプトでデータをエクスポートする場合は、TABLE型の各選択済オブジェクトで、「データ」オプションを選択します。
5. 「カートのエクスポート」をクリックします。
6. 「オブジェクトのエクスポート」ダイアログ・ボックスで、フィールドに必要な値を入力します。

   これらのフィールドの詳細は、『*Oracle SQL Developerユーザーズ・ガイド*』を参照してください。

7. 「適用」をクリックします。

   SQL Developerは、スクリプト(マスター・スクリプトを含む)を含む.zipファイルを作成して、必要な宛先の接続のスキーマのオブジェクトを作成します。

8. マスター・スクリプトおよびその実行するスクリプトで、次をチェックします。
   - 参照されているオブジェクトが、依存オブジェクトの前に作成されること。
   - データを挿入する前に、挿入先の表が作成されること。

インストール・スクリプトで順序を作成する場合、「順序を作成するインストール・スクリプトの編集」を参照してください。

インストール・スクリプトでトリガーを作成する場合、「順序を作成するインストール・スクリプトの編集」を参照してください。

必要な場合は、ワークシートまたは任意のテキスト・エディタでインストール・ファイルを編集します。

関連項目:

カートの詳細は、『*Oracle SQL Developerユーザーズ・ガイド*』を参照してください。

親トピック: インストール・スクリプトの作成

10.3.2 データベース・エクスポート・ウィザードによるインストール・スクリプトの作成

データベース・エクスポート・ウィザードを使用してSQL Developerでインストール・スクリプトを作成するには、インストール・スクリプトの名前、エクスポートするオブジェクトとデータおよび必要なオプションを指定すると、ウィザードがインストール・スクリプトを生成します。

注意:

次の手順では、すべてのフィールドとオプションが表示されるように、SQL Developer ウィンドウを拡大する必要のある場合があります。

データベース・エクスポート・ウィザードを使用してインストール・スクリプトを作成するには、次の手順を実行します。

1. そのようにしたことがない場合、Oracle Databaseインストール・ディレクトリとは別にインストール・スクリプト用のディレクトリを作成します(例: C:¥my_exports)。
2. SQL Developerウィンドウで「ツール」メニューをクリックします。
3. メニューから、「データベース・エクスポート」を選択します。
4. エクスポート・ウィザード - ステップ1/5 (ソース/宛先)ウィンドウで、次の手順を実行します。
    a. 「接続」フィールドで、開発環境への接続を選択します。
    b. 必要な「DDLのエクスポート」オプションを選択します(不要なオプションは選択を解除します)。

注意:

「終了文字」の選択を解除すると、インストール・スクリプトは失敗します。

    c. インストール・スクリプトでデータをエクスポートしない場合、「データのエクスポート」の選択を解除します。
    d. 「別名保存」フィールドで、デフォルトの「単一ファイル」を受け入れて、インストール・スクリプトのフルパス名(たとえば、C:¥my_exports¥hr_export.sql)を入力します。

    ファイル名の末尾は.sqlである必要があります。

e. 「次へ」をクリックします。

5. エクスポート・ウィザード - ステップ2/5 (エクスポートするタイプ)ウィンドウで、次の手順を実行します。

   a. エクスポートしない型のチェック・ボックスを選択解除します。

      「すべて設定」を選択または選択を解除すると、すべてのチェック・ボックスが選択または選択解除されます。

   b. 「次へ」をクリックします。

6. エクスポート・ウィザード - ステップ3/5 (オブジェクトの指定)ウィンドウで、次の手順を実行します。

   a. 「詳細」をクリックします。

   b. 「Schema」フィールドで、メニューからスキーマを選択します。

   c. 「タイプ」フィールドで、メニューからALL OBJECTSまたは特定のオブジェクト・タイプ(TABLEなど)のいずれかを選択します。

   d. 「参照」をクリックします。

      左側のフレームにオブジェクトのリストが表示されます。「タイプ」フィールドの値がALL OBJECTS場合、リストには選択したスキーマのすべてのオブジェクトが含まれます。「タイプ」フィールドの値が特定のオブジェクト・タイプの場合、リストには選択したスキーマのそのタイプのすべてのオブジェクトが含まれます。

   e. エクスポートするオブジェクトを、左のフレームから右のフレームに移動します。

      すべてのオブジェクトを移動するには、「>>」をクリックします。(すべてのオブジェクトを元に戻すには、「<<」をクリックします。)

      選択したオブジェクトを移動するには、オブジェクトを選択して「>」をクリックします。(選択したオブジェクトを元に戻すには、オブジェクトを選択して「<」をクリックします。)

   f. (オプション)他のオブジェクト・タイプについて、手順6.cから6.eまでを繰り返します。

   g. 「次へ」をクリックします。

      「ソース/宛先」 ウィンドウで **「データのエクスポート」** の選択を解除した場合、「エクスポートのサマリー」ウィンドウが表示されるので、手順8に進みます。

      「ソース/宛先」ウィンドウで **「データのエクスポート」** の選択を解除しなかった場合、エクスポート・ウィザード - ステップ4/5 (データの指定)ウィンドウが表示されます。下部フレームには、「オブジェクトの指定」ウィンドウで指定したオブジェクトがリスト表示されます。

7. 「データの指定」ウィンドウで、次の手順を実行します。

   a. データをエクスポートしないオブジェクトを、下部フレームから上部フレームに移動します。

      すべてのオブジェクトを移動するには、上向きの二重矢印アイコンをクリックします。(すべてのオブジェクトを元に戻すには、下向きの二重矢印アイコンをクリックします。)

      選択したオブジェクトを移動するには、オブジェクトを選択して上向きの一重矢印アイコンをクリックします。

   b. 「次へ」をクリックします。

8. エクスポート・ウィザード - ステップ5/5 (エクスポートのサマリー)Export Wizard - Step 5 of 5 (Export Summary)ウィンドウで、「終了」をクリックします。

   エクスポート中であることを示す「エクスポート中」ウィンドウが開きます。エクスポートが完了すると、「エクスポート中」ウィンドウが閉じ、ワークシートに「ソース/宛先」ウィンドウで指定したインストール・スクリプトのコンテンツが表示されます。

9. インストール・スクリプトで次をチェックします。

    a. 参照されているオブジェクトが、依存オブジェクトの前に作成されること。

    b. データを挿入する前に、挿入先の表が作成されること。

  必要に応じて、ワークシートまたはテキスト・エディタでファイルを編集します。

関連項目:

データ・エクスポート・ウィザードの詳細は、『*Oracle SQL Developerユーザーズ・ガイド*』を参照してください。

親トピック: インストール・スクリプトの作成

10.3.3 順序を作成するインストール・スクリプトの編集

アプリケーションが一意のキーを生成する順序を使用しており、ソース表から関連する新規表にデータを挿入していない場合、インストール・スクリプトのSTART WITH値を編集できます。

順序については、START WITHの値が開発環境における順序の現在の値に関連する順序となっているCREATE SEQUENCE文が、SQL Developerで生成されます。

アプリケーションが一意のキーを生成する順序を使用しており、ソース表から関連する新規表にデータを挿入していない場合、インストール・スクリプトのSTART WITH値を編集できます。

インストール・スクリプトはワークシートまたはテキスト・エディタで編集できます。

関連項目:

「チュートリアル: 順序の作成」

親トピック: インストール・スクリプトの作成

10.3.4 トリガーを作成するインストール・スクリプトの編集

ソース表にBEFORE INSERTトリガーがあるアプリケーションで、ソース表から関連する新規表にデータを挿入する場合、インストール・スクリプトの各INSERT文による新規表へのデータの挿入の前に、トリガーを起動するかどうか決定する必要があります。

たとえば、(「チュートリアル: 行を挿入する前に行に対して主キーを生成するトリガーの作成」で作成した) NEW_EVALUATION_TRIGGERは、行がEVALUATIONS表に挿入される前に起動します。トリガーは、EVALUATIONS_SEQUENCEを使用して、その行の主キーに対する一意の番号を生成します。

ソースのEVALUATIONS表は主キーとともに移入されています。インストール・スクリプトを新規EVALUATIONS表の新規主キーしない場合、インストール・スクリプトのCREATE TRIGGER文を次のように編集する必要があります。

```
CREATE OR REPLACE
TRIGGER NEW_EVALUATION_TRIGGER
BEFORE INSERT ON EVALUATIONS
FOR EACH ROW
BEGIN
  IF :NEW.evaluation_id IS NULL THEN
    :NEW.evaluation_id := evaluations_sequence.NEXTVAL
```

```
  END IF;
END;
```

また、順序の現在の値が主キー列の最大値より大きくない場合、大きくする必要があります。

インストール・スクリプトはワークシートまたはテキスト・エディタで編集できます。

インストール・スクリプトの編集には2つの方法があります。

- ソース・ファイルのトリガー定義を変更し、インストール・スクリプトを再作成します。

  トリガーの変更の詳細は、「トリガーの変更」を参照してください。

- トリガーを無効にしてからデータ・インストール・スクリプトを実行し、トリガーを再度有効にします。

  トリガーの無効化および有効化の詳細は、「トリガーの無効化および有効化」を参照してください。

関連項目:

「トリガーの作成」

親トピック: インストール・スクリプトの作成

10.3.5 サンプル・アプリケーションのインストール・スクリプトの作成

サンプル・アプリケーションのインストール・スクリプトを作成できます。

これらのスクリプトは、「簡易的なOracle Databaseアプリケーションの開発」用です。

- 「アプリケーションのスキーマの作成」および「スキーマへの権限の付与」の開発環境の実行内容をデプロイメント環境で実行する**schemas.sql**
- 「スキーマ・オブジェクトの作成およびデータのロード」の開発環境の実行内容をデプロイメント環境で実行する**objects.sql**
- 「employees_pkgパッケージの作成」の開発環境の実行内容をデプロイメント環境で実行する**employees.sql**
- "「admin_pkgパッケージの作成」の開発環境の実行内容をデプロイメント環境で実行する**admin.sql**
- 前のスクリプトを実行し、デプロイメント環境にサンプル・アプリケーションをデプロイするマスター・スクリプトの**create_app.sql**

任意の順序でスクリプトを作成できます。schemas.sqlおよびcreate_app.sqlを作成するには、テキスト・エディタを使用する必要があります。他のスクリプトを作成するには、テキスト・エディタまたはSQL Developerを使用できます。

- インストール・スクリプトschemas.sqlの作成
- インストール・スクリプトobjects.sqlの作成
- インストール・スクリプトemployees.sqlの作成
- インストール・スクリプトadmin.sqlの作成
- マスター・インストール・スクリプトcreate_app.sqlの作成

親トピック: インストール・スクリプトの作成

10.3.5.1 インストール・スクリプトschemas.sqlの作成

インストール・スクリプトschemas.sqlは、"「アプリケーションのスキーマの作成」および「スキーマへの権限の付与」の開発環境

の実行内容をデプロイメント環境で実行します。

schemas.sqlを作成するには、任意のテキスト・エディタで次のテキストを入力し、schemas.sqlとしてファイルを保存します。

> **注意:**
>
> セキュアなパスワードを選択します。セキュアなパスワードの詳細は、『*Oracle Database* セキュリティ・ガイド』を参照してください。

```
----------------
```

-- Create schemas ---------- DROP USER app_data CASCADE; CREATE USER app_data IDENTIFIED BY *password* DEFAULT TABLESPACE USERS QUOTA UNLIMITED ON USERS ENABLE EDITIONS; DROP USER app_code CASCADE; CREATE USER app_code IDENTIFIED BY *password* DEFAULT TABLESPACE USERS QUOTA UNLIMITED ON USERS ENABLE EDITIONS; DROP USER app_admin CASCADE; CREATE USER app_admin IDENTIFIED BY *password* DEFAULT TABLESPACE USERS QUOTA UNLIMITED ON USERS ENABLE EDITIONS; DROP USER app_user CASCADE; CREATE USER app_user IDENTIFIED BY *password* ENABLE EDITIONS; DROP USER app_admin_user CASCADE; CREATE USER app_admin_user IDENTIFIED BY *password* ENABLE EDITIONS; ---------- -- Grant privileges to schemas ---------- GRANT CREATE SESSION TO app_data; GRANT CREATE TABLE, CREATE VIEW, CREATE TRIGGER, CREATE SEQUENCE TO app_data; GRANT SELECT ON HR.DEPARTMENTS TO app_data; GRANT SELECT ON HR.EMPLOYEES TO app_data; GRANT SELECT ON HR.JOB_HISTORY TO app_data; GRANT SELECT ON HR.JOBS TO app_data; GRANT CREATE SESSION, CREATE PROCEDURE, CREATE SYNONYM TO app_code; GRANT CREATE SESSION, CREATE PROCEDURE, CREATE SYNONYM TO app_admin; GRANT CREATE SESSION, CREATE SYNONYM TO app_user; GRANT CREATE SESSION, CREATE SYNONYM TO app_admin_user;

関連項目:

サンプル・アプリケーションのスキーマの説明の詳細は、「アプリケーションのスキーマ」を参照してください。

親トピック: サンプル・アプリケーションのインストール・スクリプトの作成

10.3.5.2 インストール・スクリプトobjects.sqlの作成

インストール・スクリプトobjects.sqlは、「スキーマ・オブジェクトの作成およびデータのロード」の開発環境の実行内容をデプロイメント環境で実行します。

テキスト・エディタまたはSQL Developerを使用して、objects.sqlを作成できます。

任意のテキスト・エディタでobjects.sqlを作成するには、次のテキストを入力し、objects.sqlとしてファイルを保存します。パスワードに、ユーザーapp_dataの作成時にschema.sqlが指定するパスワードを使用します。

> **注意:**

デプロイメント環境に一般的な HR スキーマが存在する場合のみ、データをロードする INSERT 文は有効です。存在しない場合、SQL Developer を使用してソース表のデータ(開発環境)とともに新規表(デプロイメント環境)をロードするスクリプトを作成するか、次のスクリプトで INSERT 文を変更します。

```
----------------------
```

-- Create schema objects ---------- CONNECT app_data/*password* CREATE TABLE jobs# (job_id VARCHAR2(10) CONSTRAINT jobs_pk PRIMARY KEY, job_title VARCHAR2(35) CONSTRAINT jobs_job_title_not_null NOT NULL, min_salary NUMBER(6) CONSTRAINT jobs_min_salary_not_null NOT NULL, max_salary NUMBER(6) CONSTRAINT jobs_max_salary_not_null NOT NULL) / CREATE TABLE departments# (department_id NUMBER(4) CONSTRAINT departments_pk PRIMARY KEY, department_name VARCHAR2(30) CONSTRAINT dept_department_name_not_null NOT NULL CONSTRAINT dept_department_name_unique UNIQUE, manager_id NUMBER(6)) / CREATE TABLE employees# (employee_id NUMBER(6) CONSTRAINT employees_pk PRIMARY KEY, first_name VARCHAR2(20) CONSTRAINT emp_first_name_not_null NOT NULL, last_name VARCHAR2(25) CONSTRAINT emp_last_name_not_null NOT NULL, email_addr VARCHAR2(25) CONSTRAINT emp_email_addr_not_null NOT NULL, hire_date DATE DEFAULT TRUNC(SYSDATE) CONSTRAINT emp_hire_date_not_null NOT NULL CONSTRAINT emp_hire_date_check CHECK(TRUNC(hire_date) = hire_date), country_code VARCHAR2(5) CONSTRAINT emp_country_code_not_null NOT NULL, phone_number VARCHAR2(20) CONSTRAINT emp_phone_number_not_null NOT NULL, job_id CONSTRAINT emp_job_id_not_null NOT NULL CONSTRAINT emp_to_jobs_fk REFERENCES jobs#, job_start_date DATE CONSTRAINT emp_job_start_date_not_null NOT NULL, CONSTRAINT emp_job_start_date_check CHECK(TRUNC(JOB_START_DATE) = job_start_date), salary NUMBER(6) CONSTRAINT emp_salary_not_null NOT NULL, manager_id CONSTRAINT emp_mgrid_to_emp_empid_fk REFERENCES employees#, department_id CONSTRAINT emp_to_dept_fk REFERENCES departments#) / CREATE TABLE job_history# (employee_id CONSTRAINT job_hist_to_emp_fk REFERENCES employees#, job_id CONSTRAINT job_hist_to_jobs_fk REFERENCES jobs#, start_date DATE CONSTRAINT job_hist_start_date_not_null NOT NULL, end_date DATE CONSTRAINT job_hist_end_date_not_null NOT NULL, department_id CONSTRAINT job_hist_to_dept_fk REFERENCES departments# CONSTRAINT job_hist_dept_id_not_null NOT NULL, CONSTRAINT job_history_pk PRIMARY KEY(employee_id,start_date), CONSTRAINT job_history_date_check CHECK(start_date < end_date)) / CREATE EDITIONING VIEW jobs AS SELECT * FROM jobs# / CREATE EDITIONING VIEW departments AS SELECT * FROM departments# / CREATE EDITIONING VIEW employees AS SELECT * FROM employees# / CREATE EDITIONING VIEW job_history AS SELECT * FROM job_history# / CREATE OR REPLACE TRIGGER employees_aiufer AFTER INSERT OR UPDATE OF salary, job_id ON employees FOR EACH ROW DECLARE l_cnt NUMBER; BEGIN LOCK TABLE jobs IN SHARE MODE; -- Ensure that jobs does not change -- during the following query. SELECT COUNT(*) INTO l_cnt FROM jobs WHERE job_id = :NEW.job_id AND :NEW.salary BETWEEN min_salary AND max_salary; IF (l_cnt<>1) THEN RAISE_APPLICATION_ERROR(-20002, CASE WHEN :new.job_id = :old.job_id THEN 'Salary modification invalid' ELSE 'Job reassignment puts salary out of range' END); END IF; END; / CREATE OR REPLACE TRIGGER jobs_aufer AFTER UPDATE OF min_salary, max_salary ON jobs FOR EACH ROW WHEN (NEW.min_salary > OLD.min_salary OR NEW.max_salary <

OLD.max_salary) DECLARE l_cnt NUMBER; BEGIN LOCK TABLE employees IN SHARE MODE; SELECT COUNT(*) INTO l_cnt FROM employees WHERE job_id = :NEW.job_id AND salary NOT BETWEEN :NEW.min_salary and :NEW.max_salary; IF (l_cnt>0) THEN RAISE_APPLICATION_ERROR(-20001, 'Salary update would violate ' || l_cnt || ' existing employee records'); END IF; END; / CREATE SEQUENCE employees_sequence START WITH 210; CREATE SEQUENCE departments_sequence START WITH 275; ------------ -- Load data ------------ **INSERT** INTO jobs (job_id, job_title, min_salary, max_salary) SELECT job_id, job_title, min_salary, max_salary **FROM HR.JOBS** / **INSERT** INTO departments (department_id, department_name, manager_id) SELECT department_id, department_name, manager_id **FROM HR.DEPARTMENTS** / **INSERT** INTO employees (employee_id, first_name, last_name, email_addr, hire_date, country_code, phone_number, job_id, job_start_date, salary, manager_id, department_id) SELECT employee_id, first_name, last_name, email, hire_date, CASE WHEN phone_number LIKE '011.%' THEN '+' || SUBSTR(phone_number, INSTR(phone_number, '.')+1, INSTR(phone_number, '.', 1, 2) - INSTR(phone_number, '.') - 1) ELSE '+1' END country_code, CASE WHEN phone_number LIKE '011.%' THEN SUBSTR(phone_number, INSTR(phone_number, '.', 1, 2)+1) ELSE phone_number END phone_number, job_id, NVL((SELECT MAX(end_date+1) FROM HR.JOB_HISTORY jh WHERE jh.employee_id = employees.employee_id), hire_date), salary, manager_id, department_id **FROM HR.EMPLOYEES** / **INSERT** INTO job_history (employee_id, job_id, start_date, end_date, department_id) SELECT employee_id, job_id, start_date, end_date, department_id **FROM HR.JOB_HISTORY** / COMMIT; ---------- -- Add foreign key constraint ---------- ALTER TABLE departments# ADD CONSTRAINT dept_to_emp_fk FOREIGN KEY(manager_id) REFERENCES employees#; ---------- ---------- -- Grant privileges on schema objects to users ---------- ------ GRANT SELECT, INSERT, UPDATE, DELETE ON employees TO app_code; GRANT SELECT ON departments TO app_code; GRANT SELECT ON jobs TO app_code; GRANT SELECT, INSERT on job_history TO app_code; GRANT SELECT ON employees_sequence TO app_code; GRANT SELECT, INSERT, UPDATE, DELETE ON jobs TO app_admin; GRANT SELECT, INSERT, UPDATE, DELETE ON departments TO app_admin; GRANT SELECT ON employees_sequence TO app_admin; GRANT SELECT ON departments_sequence TO app_admin; GRANT SELECT ON jobs TO app_admin_user; GRANT SELECT ON departments TO app_admin_user;

関連項目:

- サンプル・アプリケーションのスキーマ・オブジェクトの説明の詳細は、「アプリケーションのスキーマ・オブジェクト」を参照してください。
- 「カートによるインストール・スクリプトの作成」
- 「データベース・エクスポート・ウィザードによるインストール・スクリプトの作成」

親トピック: サンプル・アプリケーションのインストール・スクリプトの作成

10.3.5.3 インストール・スクリプトemployees.sqlの作成

インストール・スクリプトemployees.sqlは、「employees_pkgパッケージの作成」の開発環境の実行内容をデプロイメント環境で実行します。

テキスト・エディタまたはSQL Developerを使用して、employees.sqlを作成できます。

任意のテキスト・エディタでemployees.sqlを作成するには、次のテキストを入力し、employees.sqlとしてファイルを保存します。パスワードに、ユーザーapp_codeの作成時にschema.sqlが指定するパスワードを使用します。

```
-----------------------
-- Create employees_pkg
-----------------------

CONNECT app_code/password

CREATE SYNONYM employees FOR app_data.employees;
CREATE SYNONYM departments FOR app_data.departments;
CREATE SYNONYM jobs FOR app_data.jobs;
CREATE SYNONYM job_history FOR app_data.job_history;

CREATE OR REPLACE PACKAGE employees_pkg
AS
  PROCEDURE get_employees_in_dept
    ( p_deptno     IN     employees.department_id%TYPE,
      p_result_set IN OUT SYS_REFCURSOR );

  PROCEDURE get_job_history
    ( p_employee_id  IN     employees.department_id%TYPE,
      p_result_set   IN OUT SYS_REFCURSOR );

  PROCEDURE show_employee
    ( p_employee_id  IN     employees.employee_id%TYPE,
      p_result_set   IN OUT SYS_REFCURSOR );

  PROCEDURE update_salary
    ( p_employee_id IN employees.employee_id%TYPE,
      p_new_salary  IN employees.salary%TYPE );

  PROCEDURE change_job
    ( p_employee_id IN employees.employee_id%TYPE,
      p_new_job     IN employees.job_id%TYPE,
      p_new_salary  IN employees.salary%TYPE := NULL,
      p_new_dept    IN employees.department_id%TYPE := NULL );
END employees_pkg;
/

CREATE OR REPLACE PACKAGE BODY employees_pkg
AS
  PROCEDURE get_employees_in_dept
    ( p_deptno     IN     employees.department_id%TYPE,
      p_result_set IN OUT SYS_REFCURSOR )
  IS
     l_cursor SYS_REFCURSOR;
  BEGIN
    OPEN p_result_set FOR
      SELECT e.employee_id,
        e.first_name || ' ' || e.last_name name,
        TO_CHAR( e.hire_date, 'Dy Mon ddth, yyyy' ) hire_date,
        j.job_title,
        m.first_name || ' ' || m.last_name manager,
        d.department_name
      FROM employees e INNER JOIN jobs j ON (e.job_id = j.job_id)
        LEFT OUTER JOIN employees m ON (e.manager_id = m.employee_id)
        INNER JOIN departments d ON (e.department_id = d.department_id)
```

```
    WHERE e.department_id = p_deptno ;
END get_employees_in_dept;

PROCEDURE get_job_history
  ( p_employee_id  IN     employees.department_id%TYPE,
    p_result_set   IN OUT SYS_REFCURSOR )
IS
BEGIN
  OPEN p_result_set FOR
    SELECT e.First_name || ' ' || e.last_name name, j.job_title,
      e.job_start_date start_date,
      TO_DATE(NULL) end_date
    FROM employees e INNER JOIN jobs j ON (e.job_id = j.job_id)
    WHERE e.employee_id = p_employee_id
    UNION ALL
    SELECT e.First_name || ' ' || e.last_name name,
      j.job_title,
      jh.start_date,
      jh.end_date
    FROM employees e INNER JOIN job_history jh
      ON (e.employee_id = jh.employee_id)
      INNER JOIN jobs j ON (jh.job_id = j.job_id)
    WHERE e.employee_id = p_employee_id
    ORDER BY start_date DESC;
END get_job_history;

PROCEDURE show_employee
  ( p_employee_id  IN     employees.employee_id%TYPE,
    p_result_set   IN OUT sys_refcursor )
IS
BEGIN
  OPEN p_result_set FOR
    SELECT *
    FROM (SELECT TO_CHAR(e.employee_id) employee_id,
            e.first_name || ' ' || e.last_name name,
            e.email_addr,
            TO_CHAR(e.hire_date,'dd-mon-yyyy') hire_date,
            e.country_code,
            e.phone_number,
            j.job_title,
            TO_CHAR(e.job_start_date,'dd-mon-yyyy') job_start_date,
            to_char(e.salary) salary,
            m.first_name || ' ' || m.last_name manager,
            d.department_name
          FROM employees e INNER JOIN jobs j on (e.job_id = j.job_id)
            RIGHT OUTER JOIN employees m ON (m.employee_id = e.manager_id)
            INNER JOIN departments d ON (e.department_id = d.department_id)
          WHERE e.employee_id = p_employee_id)
    UNPIVOT (VALUE FOR ATTRIBUTE IN (employee_id, name, email_addr, hire_date,
      country_code, phone_number, job_title, job_start_date, salary, manager,
      department_name) );
END show_employee;

PROCEDURE update_salary
  ( p_employee_id IN employees.employee_id%type,
    p_new_salary  IN employees.salary%type )
IS
BEGIN
  UPDATE employees
  SET salary = p_new_salary
```

```
    WHERE employee_id = p_employee_id;
  END update_salary;

  PROCEDURE change_job
    ( p_employee_id IN employees.employee_id%TYPE,
      p_new_job     IN employees.job_id%TYPE,
      p_new_salary  IN employees.salary%TYPE := NULL,
      p_new_dept    IN employees.department_id%TYPE := NULL )
  IS
  BEGIN
    INSERT INTO job_history (employee_id, start_date, end_date, job_id,
      department_id)
    SELECT employee_id, job_start_date, TRUNC(SYSDATE), job_id, department_id
      FROM employees
      WHERE employee_id = p_employee_id;

    UPDATE employees
    SET job_id = p_new_job,
      department_id = NVL( p_new_dept, department_id ),
      salary = NVL( p_new_salary, salary ),
      job_start_date = TRUNC(SYSDATE)
    WHERE employee_id = p_employee_id;
  END change_job;
END employees_pkg;
/

---------------------------------------------
-- Grant privileges on employees_pkg to users
---------------------------------------------

GRANT EXECUTE ON employees_pkg TO app_user;
GRANT EXECUTE ON employees_pkg TO app_admin_user;
```

関連項目:

- 「カートによるインストール・スクリプトの作成」
- 「データベース・エクスポート・ウィザードによるインストール・スクリプトの作成」

親トピック: サンプル・アプリケーションのインストール・スクリプトの作成

10.3.5.4 インストール・スクリプトadmin.sqlの作成

インストール・スクリプトadmin.sqlは、「admin_pkgパッケージの作成」の開発環境の実行内容をデプロイメント環境で実行します。

テキスト・エディタまたはSQL Developerを使用して、admin.sqlを作成できます。

任意のテキスト・エディタでadmin.sqlを作成するには、次のテキストを入力し、admin.sqlとしてファイルを保存します。パスワードに、ユーザーapp_adminの作成時にschema.sqlが指定するパスワードを使用します。

```
--------------------
-- Create admin_pkg
--------------------

CONNECT app_admin/password
```

```
CREATE SYNONYM departments FOR app_data.departments;
CREATE SYNONYM jobs FOR app_data.jobs;
CREATE SYNONYM departments_sequence FOR app_data.departments_sequence;

CREATE OR REPLACE PACKAGE admin_pkg
AS
  PROCEDURE update_job
    ( p_job_id      IN jobs.job_id%TYPE,
      p_job_title   IN jobs.job_title%TYPE := NULL,
      p_min_salary  IN jobs.min_salary%TYPE := NULL,
      p_max_salary  IN jobs.max_salary%TYPE := NULL );

  PROCEDURE add_job
    ( p_job_id      IN jobs.job_id%TYPE,
      p_job_title   IN jobs.job_title%TYPE,
      p_min_salary  IN jobs.min_salary%TYPE,
      p_max_salary  IN jobs.max_salary%TYPE );

  PROCEDURE update_department
    ( p_department_id     IN departments.department_id%TYPE,
      p_department_name   IN departments.department_name%TYPE := NULL,
      p_manager_id        IN departments.manager_id%TYPE := NULL,
      p_update_manager_id IN BOOLEAN := FALSE );

  FUNCTION add_department
    ( p_department_name   IN departments.department_name%TYPE,
      p_manager_id        IN departments.manager_id%TYPE )
    RETURN departments.department_id%TYPE;

END admin_pkg;
/

CREATE OR REPLACE PACKAGE BODY admin_pkg
AS
  PROCEDURE update_job
    ( p_job_id      IN jobs.job_id%TYPE,
      p_job_title   IN jobs.job_title%TYPE := NULL,
      p_min_salary  IN jobs.min_salary%TYPE := NULL,
      p_max_salary  IN jobs.max_salary%TYPE := NULL )
  IS
  BEGIN
    UPDATE jobs
    SET job_title  = NVL( p_job_title, job_title ),
        min_salary = NVL( p_min_salary, min_salary ),
        max_salary = NVL( p_max_salary, max_salary )
    WHERE job_id = p_job_id;
  END update_job;

  PROCEDURE add_job
    ( p_job_id      IN jobs.job_id%TYPE,
      p_job_title   IN jobs.job_title%TYPE,
      p_min_salary  IN jobs.min_salary%TYPE,
      p_max_salary  IN jobs.max_salary%TYPE )
  IS
  BEGIN
    INSERT INTO jobs ( job_id, job_title, min_salary, max_salary )
    VALUES ( p_job_id, p_job_title, p_min_salary, p_max_salary );
  END add_job;

  PROCEDURE update_department
```

```
    ( p_department_id     IN departments.department_id%TYPE,
      p_department_name   IN departments.department_name%TYPE := NULL,
      p_manager_id        IN departments.manager_id%TYPE := NULL,
      p_update_manager_id IN BOOLEAN := FALSE )
  IS
  BEGIN
    IF ( p_update_manager_id ) THEN
      UPDATE departments
      SET department_name = NVL( p_department_name, department_name ),
          manager_id = p_manager_id
      WHERE department_id = p_department_id;
    ELSE
      UPDATE departments
      SET department_name = NVL( p_department_name, department_name )
      WHERE department_id = p_department_id;
    END IF;
  END update_department;

  FUNCTION add_department
    ( p_department_name   IN departments.department_name%TYPE,
      p_manager_id        IN departments.manager_id%TYPE )
    RETURN departments.department_id%TYPE
  IS
    l_department_id departments.department_id%TYPE;
  BEGIN
    INSERT INTO departments ( department_id, department_name, manager_id )
      VALUES ( departments_sequence.NEXTVAL, p_department_name, p_manager_id )
      RETURNING department_id INTO l_department_id;
    RETURN l_department_id;
  END add_department;

END admin_pkg;
/

----------------------------------------
-- Grant privileges on admin_pkg to user
----------------------------------------

GRANT EXECUTE ON admin_pkg TO app_admin_user;
```

関連項目:

- 「カートによるインストール・スクリプトの作成」
- 「データベース・エクスポート・ウィザードによるインストール・スクリプトの作成」

親トピック: サンプル・アプリケーションのインストール・スクリプトの作成

10.3.5.5 マスター・インストール・スクリプトcreate_app.sqlの作成

マスター・インストール・スクリプトcreate_app.sqlは、サンプル・アプリケーションの他の4つのインストール・スクリプトを正しい順序で実行して、デプロイメント環境にサンプル・アプリケーションをデプロイします。

create_app.sqlを作成するには、任意のテキスト・エディタで次のテキストを入力し、create_app.sqlとしてファイルを保存します。

```
@schemas.sql
```

```
@objects.sql
@employees.sql
@admin.sql
```

親トピック: サンプル・アプリケーションのインストール・スクリプトの作成

10.4 サンプル・アプリケーションのデプロイ

インストール・スクリプトを使用してサンプル・アプリケーションをデプロイできます。

「サンプル・アプリケーションのインストール・スクリプトの作成」で作成したインストール・スクリプトを使用します。

注意:

次の手順では、CREATE USER および DROP USER システム権限を持つユーザーの名前とパスワードが必要です。

SQL*Plusを使用してサンプル・アプリケーションをデプロイするには、次の手順を実行します。

1. 「サンプル・アプリケーションのインストール・スクリプトの作成」で作成したインストール・スクリプトをデプロイメント環境にコピーします。
2. デプロイメント環境で、CREATE USERおよびDROP USERシステム権限を持つユーザーとしてOracle Databaseに接続します。
3. SQL>プロンプトで、マスター・インストール・スクリプトを実行します。

   ```
   @create_app.sql
   ```

   マスター・インストール・スクリプトは、サンプル・アプリケーションの他の4つのインストール・スクリプトを正しい順序で実行して、デプロイメント環境にサンプル・アプリケーションをデプロイします。

SQL Developerを使用してサンプル・アプリケーションをデプロイするには、次の手順を実行します。

1. 必要であれば、デプロイメント環境への接続を作成します。

   「接続名」に、開発環境の接続名ではない名前を入力します。

2. 「サンプル・アプリケーションのインストール・スクリプトの作成」で作成したインストール・スクリプトをデプロイメント環境にコピーします。
3. デプロイメント環境でCREATE USERおよびDROP USERシステム権限を持つユーザーとしてOracle Databaseに接続します。

   新しいペインが表示されます。タブにはデプロイメント環境への接続の名前が表示されます。ペインには、「ワークシート」および「クエリー・ビルダー」の2つのサブペインが表示されます。

4. 「ワークシート」ペインで、マスター・インストール・スクリプトを実行するコマンドを入力します。

   ```
   @create_app.sql
   ```

5. 「スクリプトの実行」アイコンをクリックします。

   マスター・インストール・スクリプトは、サンプル・アプリケーションの他の4つのインストール・スクリプトを正しい順序で実行して、デプロイメント環境にサンプル・アプリケーションをデプロイします。出力が「ワークシート」 ペインの下の「スクリプトの

出力」ペインに表示されます。

「接続」フレームでは、デプロイメント環境への接続を展開し、次にサンプル・アプリケーションが使用する各オブジェクトのタイプを展開すると、サンプル・アプリケーションのオブジェクトが表示されます。

関連項目:

- SQL*Plusにおけるスクリプトの使用の詳細は、『*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でのスクリプトの実行の詳細は、『*Oracle SQL Developerユーザーズ・ガイド*』を参照してください。

親トピック: Oracle Databaseアプリケーションのデプロイ

10.5 インストールの有効性のチェック

デプロイメント環境でアプリケーションをインストールした後、SQL Developerでインストールの有効性を確認できます。

- 「接続」フレームで、次の手順を実行します。
    1. デプロイメント環境への接続を展開します。
    2. 新しいオブジェクトの定義を確認します。
- 「レポート」ペインで、次の手順を実行します。
    1. 「データ・ディクショナリ・レポート」を展開します。

       データ・ディクショナリ・レポートのリストが表示されます。
    2. 「すべてのオブジェクト」を展開します。

       オブジェクト・レポートのリストが表示されます。
    3. 「すべてのオブジェクト」を選択します。

       「接続の選択」ウィンドウが表示されます。
    4. 「接続」フィールドで、デプロイメント環境への接続をメニューから選択します。
    5. 「OK」をクリックします。
    6. 「バインド値の入力」ウィンドウで、「**所有者**」または「**オブジェクト名**」のいずれかを選択します。
    7. 「適用」をクリックします。

       「**結果の表示**」メッセージの後に、結果が表示されます。

       このレポートではオブジェクトごとに、「所有者」、「オブジェクト・タイプ」、「オブジェクト名」、「ステータス」(「有効」または「無効」)、「作成日」および「最終DDL」のリストを出力します。「最終DDL」は、オブジェクトに影響を与えた最後のDDL操作の日付です。
    8. 「レポート」ペインで、「無効なオブジェクト」を選択します。
    9. 「バインド値の入力」ウィンドウで、「適用」をクリックします。

       このレポートでは、「ステータス」が「無効」の各オブジェクトについて、「所有者」、「オブジェクト・タイプ」および「オブジェクト名」のリストを出力します。

関連項目:

SQL Developerレポートの詳細は、『*Oracle SQL Developerユーザーズ・ガイド*』を参照してください。

親トピック: Oracle Databaseアプリケーションのデプロイ

10.6 インストール・スクリプトのアーカイブ

アプリケーションのインストールが有効であることを確認したら、ソース・コード制御システムでインストール・スクリプトをアーカイブすることをお薦めします。

アーカイブする前に、各ファイルに作成日と目的を示すコメントを追加します。同じアプリケーションを他の環境にデプロイする必要が生じた場合に、このアーカイブ・ファイルを使用できます。

関連項目:

データベース間におけるデータとメタデータの高速移動を可能にするOracle Data Pumpの詳細は、『*Oracle Database ユーティリティ*』を参照してください。

親トピック: Oracle Databaseアプリケーションのデプロイ

索引

記号

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W