Oracle® Database

2 日で開発者ガイド

11*g* リリース 1（11.1）

部品番号 : E05694-03

2008 年 10 月

ORACLE®

Oracle Database 2 日で開発者ガイド , 11*g* リリース 1（11.1）

部品番号 : E05694-03

Oracle Database 2 Day Developer's Guide, 11*g* Release 1 (11.1)

原本部品番号 : B28843-04

原本著者 : Roza Leyderman

原本協力者 : Pat Huey Sharon Kennedy、Simon Law、Bryn Llewellen、Chuck Murray、Mark Townsend

目次

5 トリガーの使用

6 グローバル環境での作業

7 データベース・アプリケーションのデプロイ

はじめに

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独自に機能拡張した Procedural Language/Structured Query Language（PL/SQL）を介して、Oracle Database 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 製品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。

このマニュアルを使用する前に、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おく必要があります。

このマニュアルで説明するテクノロジについて理解を深めた後、他の Oracle Database 開発ガイド（特に、『Oracle Database 2 日で Application Express 開発者ガイド』、『Oracle Database 2 日で Java 開発者ガイド』、『Oracle Database 2 日で .Net 開発者ガイド』および『Oracle Database 2 日で PHP 開発者ガイド』）を参照することをお薦めします。

ドキュメントのアクセシビリティについて

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

ドキュメント内のサンプル・コードのアクセシビリティについて

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

外部 Web サイトのドキュメントのアクセシビリティについて

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

Oracle サポート・サービスへの TTY アクセス

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

関連ドキュメント

詳細は、Oracle Database 11*g* リリース 1（11.1）ライブラリの次のマニュアルを参照してください。

- 『Oracle Database アドバンスト・アプリケーション開発者ガイド』
- 『Oracle Database 概要』
- 『Oracle Database SQL 言語リファレンス』
- 『Oracle Database PL/SQL 言語リファレンス』

表記規則

この項では、このマニュアルの本文およびコード例で使用される表記規則について説明します。

規則	意味
太字	太字は、操作に関連する 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、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。

サポートおよびサービス

次の各項に、各サービスに接続するための URL を記載します。

Oracle サポート・サービス

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

```
http://www.oracle.com/lang/jp/support/index.html
```

製品マニュアル

製品のマニュアルは、次の URL にあります。

```
http://www.oracle.com/technology/global/jp/documentation/index.html
```

研修およびトレーニング

研修に関する情報とスケジュールは、次の URL で入手できます。

```
http://education.oracle.com/pls/web_prod-plq-dad/db_pages.getpage?page_id=3
```

その他の情報

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

```
http://www.oracle.com/lang/jp/index.html
http://www.oracle.com/technology/global/jp/index.html
```

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

1

Oracle Database および開発の概要

この章では Oracle Database によるアプリケーション開発について説明します。

この章の内容は次のとおりです。

ロードマップ

ユーザーは、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の責任がある Oracle Database 開発者です。この項および次の項の説明では、ユーザーまたは組織内の責任者がマルチユーザー・アプリケーション（2 層または複数の層）を構築する方法を確認し、データの整合性のある層のリレーショナル・データベースを使用する利点を理解する必要があります。

データベース開発者として、アプリケーションに必要なデータ・モデルの実装方法、データの整合性のルールの実装方法、およびアプリケーション・データのアクセスおよび操作に対して指定したファンクションの実装方法を知る必要があります。

クライアント・プログラムを介してのみ Oracle Database へのアクセスが可能なこと、および SQL 言語が Oracle Database に対するクライアント・プログラム・インタフェースであることは確認済です。Oracle Database でパッケージされ、開発者用に設計された 2 つのクライアントを使用して、Oracle Database へのアクセス方法を学びます。SQL Developer および SQL*Plus を使用すると、クライアント・プログラミングすることなくアプリケーション・データベース・コンポーネントのテストおよび作成のために必要な SQL 文を発行できます。クライアントなどのプログラミングは、ここでの説明の範囲外になります。

ソフトウェア・エンジニアリングにおいて広く指示されるベスト・プラクティスを簡単に確認するには、ビジネス機能のモデルである API を定義し、実装を隠す必要があります。API を PL/SQL サブプログラムとして指定でき、それによって Oracle Database でこのプラクティスがサポートされます。表、索引、制約、トリガー、および表の行を変更およびフェッチする様々な SQL 文が実装されます。これらの SQL 文を PL/SQL サブプログラムに埋め込むこと、および Oracle スキーマと権限メカニズムを使用することによって、クライアント・プログラムから実装を安全に隠すことができます。多くの Oracle の主要なカスタマ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 サブプログラムをコール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に *RAW* SQL `SELECT` 文を発行させることでこのルールを受け入れています。ただし、この文は、データベースに変更を加えるすべてのビジネス機能に対する PL/SQL サブプログラムをコールするために必要です。

ここの一般的な説明では、Oracle Database 開発者としての作業に関する章を設定します。

- データベースで作成できるオブジェクトの様々なタイプに関して知る必要があります。2-2 ページの「データベース・オブジェクトの参照」を参照してください。
- オブジェクト `CREATE`、`ALTER`、`TRUNCATE` および `DROP` を管理するために使用する SQL を知る必要があります。この SQL は**データ定義言語**（**DDL**）と呼ばれます。3-1 ページの「データベース・オブジェクトの作成および使用」を参照してください。
- アプリケーション・データ `INSERT`、`UPDATE`、`DELETE` および `MERGE` をメンテナンスするために使用する SQL を知る必要があります。この SQL は**データ操作言語**（**DML**）と呼ばれます。2-1 ページの「データの問合せおよび操作」を参照してください。
- **問合せ**データ `SELECT` 文および様々な句の SQL 言語を知る必要があります。2-7 ページの「問合せを使用したデータの取得」を参照してください。
- `COMMIT`、`SAVEPOINT` および `ROLLBACK` を管理する SQL 言語およびトランザクションを知る必要があります。2-30 ページの「トランザクションの制御」を参照してください。
- DDL、DML、トランザクション制御および問合せを使用するプロシージャ・コードおよび PL/SQL サブプログラムを書く方法を知る必要があります。4-1 ページの「ストアド・プロシージャの開発および使用」および 5-1 ページの「トリガーの使用」を参照してください。
- 最終的に本番環境でのアプリケーションをデプロイするためにデータベースの開発、ユニットのテスト、統合のテスト、エンド・ユーザー受入れのテストおよび教育を目的とする多くの異なるデータベースのアプリケーションをインスタント化する方法および提供するデータベースを管理する方法を知る必要があります。詳細は、7-1 ページの「データベース・アプリケーションのデプロイ」を参照してください。

参照：

- アプリケーション・アーキテクチャの詳細は、『Oracle Database 概要』を参照してください。

Oracle Database スキーマの概要

この項では、Oracle Database スキーマについて説明します。

参照 :

- 『Oracle Database 概要』

スキーマ・オブジェクトの概要

Oracle Database には、関連情報をグループ化した**スキーマ**と呼ばれる論理構造が含まれます。**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と、ユーザー名によってスキーマが命名され、スキーマの所有者であることが示されます。スキーマにはデータベースにおけるデータ記憶領域の基本単位である**表**が含まれています。表を使用して、情報の問合せや更新、あるいは追加データの挿入や削除を実行できます。各表には個別のデータの**レコード**を表す**行**が含まれています。表の行は様々な**フィールド**を表す**列**で構成されます。

スキーマには、表の他にも非常に便利なオブジェクトが数多く含まれています。**索引**は表からのデータ取得のパフォーマンスを改善するために作成できるオプションの構造です。索引は、表の 1 つ以上の列に対して作成され、Oracle Database により自動的に維持されます。3-3 ページの「表の作成および使用」を参照してください。

ビジネスのニーズに応じて、別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、表および他のビューの情報に依存する可能性があります。3-20 ページの「ビューの使用」を参照してください。

表のすべてのレコードが一意である必要があるアプリケーションでは、**順序**により、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。3-25 ページの「順序の使用」を参照してください。

シノニムとは、表、ビュー、順序、プロシージャなどの別名です。シノニムはしばしば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。3-28 ページの「シノニムの使用」を参照してください。

スキーマ・レベル・**プロシージャ**、**ファンクション**および**パッケージ**は、総称して**ストアド・プロシージャ**と呼ばれます。ストアド・プロシージャは、データベースに実際に格納されたコードのブロックです。リレーショナル・データベース・システムにアクセスするクライアント・アプリケーションからコール可能です。4-1 ページの「ストアド・プロシージャの開発および使用」を参照してください。

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プロシージャ・コードです。トリガーは、特定のデータへのアクセス、ロギングの実行またはデータの監視を制限できます。5-1 ページの「トリガーの使用」を参照してください。

参照 :

- すべてのスキーマ・オブジェクトの詳細は、『Oracle Database 概要』を参照してください。

HR スキーマの概要

hr スキーマはサンプル・スキーマの 1 つで、Oracle Database の一部としてインストールされます。hr サンプル・スキーマには従業員、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 スキーマにも表、ビュー、索引、パッケージ、プロシージャ、ファンクション、その他すべての Oracle Database スキーマの属性があります。

hr スキーマを使用および拡張して、Oracle Database でのアプリケーション開発を習得します。

参照 :

- hr サンプル・スキーマの詳細は、『Oracle Database サンプル・スキーマ』を参照してください。

Oracle Database によるアプリケーション開発の概要

この項では、データに直接アクセスできる 2 つの開発言語（SQL および PL/SQL）、2 つの開発ツール（SQL Developer および SQL*Plus）、サンプル・データ・セット（hr スキーマ）および Oracle Database のインスタンスへの接続方法について説明します。

参照：

- 『Oracle Database アドバンスト・アプリケーション開発者ガイド』

SQL および PL/SQL 開発言語の概要

広義のコンピュータ言語には何をする必要があるかを説明する宣言型言語と、どのように行うかを説明する命令型言語の 2 グループに分けられます。データベースに依存しない言語である**構造化問合せ言語**、つまり **SQL** はすでに熟知しているでしょう。SQL はセット・ベースの高度な**宣言型言語**で、目的のデータの条件を記述することで問題を説明します。SQL 文によって、表を問い合せてデータを表示したり、オブジェクトを作成および修正したり、様々な管理タスクを実行できます。SQL コマンドを発行すると、SQL 言語コンパイラが自動的にプロシージャを生成し、データベースにアクセスして目的のタスクを実行できます。

一方、C、C++、Java などの**命令型言語**は必要なデータを見つけることで、問題の解決方法を指定します。つまり計算を、プログラムの状態の変更や、より広範囲な問題の解決を行う文として記述します。

手続き型言語の SQL、つまり **PL/SQL** は、オラクル社が機能を拡張した SQL です。条件付き制御や反復フロー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。SQL と同様に、PL/SQL にもリレーショナル・データベース・ドメインの埋込み型があります。PL/SQL により、定数および変数の宣言、プロシージャとファンクションの定義、コレクション・タイプおよびオブジェクト・タイプの使用、ランタイム・エラーの通知などを実行できます。また、Oracle のプログラム・インタフェースで記述された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ファンクション、パッケージ、プロシージャおよびトリガーを作成できます。

PL/SQL の詳細は、次の Oracle Technology Network の PL/SQL のサイトを参照してください。

```
http://www.oracle.com/technology/tech/pl_sql/
```

参照：

- 『Oracle Database SQL 言語リファレンス』
- 『Oracle Database PL/SQL 言語リファレンス』
- 『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』
- 「その他の開発環境の概要」（1-10 ページ）

SQL Developer の概要

SQL Developer は、Oracle Database のインスタンスにアクセスするためのグラフィカル・ユーザー・インタフェースです。**SQL Developer** は、SQL および PL/SQL 言語の開発をサポートします。Oracle Database のデフォルトのインストールで使用可能です。ナビゲーション階層および SQL ワークシートを介して SQL Developer を使用します。

SQL Developer を実行する前に、Java1.5.0 がインストールされていることを確認します。コマンド・プロンプトで次のコマンドを入力します。

```
java -version
```

次のような出力が表示されます。

```
java version "1.5.0_06"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_06-b05)
Java HotSpot(TM) Client VM (build 1.5.0_06-b05, mixed mode, sharing)
```

SQL Developer を起動するには、次の手順を実行します。

1. Linux の場合 :

   - 「**アプリケーション**」メニューをクリックするか（Gnome の場合）、または「**K**」メニューをクリックします（KDE の場合）。
   - 「**Oracle - *ORACLE_HOME***」、「**Application Development**」、「**SQL Developer**」の順に選択します。

   Windows の場合 :

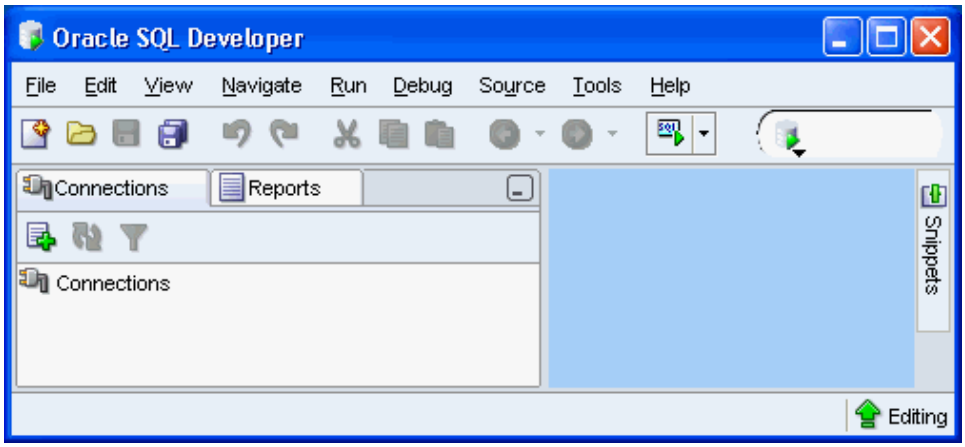
   - 「**スタート**」メニューから「**すべてのプログラム**」を選択します。
   - 「**Oracle - *ORACLE_HOME***」、「**Application Development**」、「**SQL Developer**」の順に選択します。

2. プロンプトが表示され、Java 実行可能ファイルへのフルパスを入力します。

   たとえば、`C:¥jdk1.5.0¥bin¥java.exe` です。

   このパスを指定する必要があるのは、SQL Developer の初回起動時のみです。

   スプラッシュ画面が表示された後に、SQL Developer が起動します。



SQL Developer の詳細は、次の Oracle Technology Network の SQL Developer のサイトを参照してください。

`http://www.oracle.com/technology/products/database/sql_developer/index.html`

参照 :

- 『Oracle Database SQL Developer ユーザーズ・ガイド』

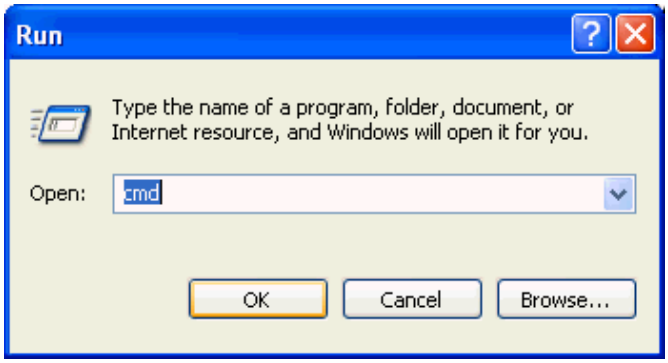
SQL*Plus の概要

SQL*Plus は Oracle Database と一緒にインストールされます。SQL*Plus は Oracle Database にアクセスするためのコマンドライン・ユーザー・インタフェースを持っています。また、SQL Developer 内でも SQL*Plus にアクセスできます。

Windows で SQL*Plus を使用するには、次の手順を実行します。

1. 画面の左下隅にある「**起動**」アイコンをクリックして「**実行**」を選択します。

2. 「Run」ウィンドウのテキスト・プロンプトで cmd と入力します。「**OK**」をクリックします。



3. cmd.exe ウィンドウの c:>¥ プロンプトで、sqlplus を入力しキーボードの [Enter] を押します。

   SQL*Plus が起動すると、データベースへの接続を認証するように要求されます。

   画面が次のように表示されます。

```
C:¥>sqlplus
SQL*Plus: Release 11.1.0.1.0 - Production on Tue April 3 10:10:11 2007
Copyright (c) 1982, 2007, Oracle. All rights reserved.
Enter user-name:
```

4. ユーザー名を入力し、[Enter] を押します。

   画面が次のように表示されます。

```
Enter password:
```

5. パスワードを入力し、[Enter] を押します。この方法でユーザー名およびパスワードを入力すると、パスワードがスクリーンに表示されないため安全です。

   データベース・インスタンスに接続され、SQL プロンプトが表示されます。

   画面が次のように表示されます。

```
Connected to:
Oracle Database 11g Enterprise Edition Release 11.1.0.1.0 - Production
With the Partitioning, OLAP and Data Mining options
```

   これで SQL コマンド・プロンプトを使用できます。

6. SQL*Plus セッションをクローズするには、SQL プロンプトに exit コマンドを入力します。Oracle Database インスタンスは停止しないように注意してください。

```
SQL> exit
```

   画面が次のように表示されます。

```
Disconnected from Oracle Database 11g Enterprise Edition Release 11.1.0.1.0
With the Partitioning, OLAP and Data Mining options
```

参照：

- 『Oracle Database 2 日でセキュリティ・ガイド』
- 『SQL*Plus ユーザーズ・ガイドおよびリファレンス』

データベースへの接続

Oracle Database ではユーザーとユーザーが接続するスキーマの名前は同じです。この項では Oracle Database に同梱されているサンプル・スキーマの 1 つ、hr スキーマへの接続の作成方法を示します。まず、hr アカウントをロック解除します。

この項の内容は次のとおりです。

ユーザー・アカウントのロック解除

デフォルトでは、hr スキーマがインストールされた際にロックされ、パスワードが期限切れになります。hr スキーマを使用して Oracle Database に接続する前に、管理者権限を持つユーザーは 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必要があります。

次の手順は、hr アカウントのロック解除の方法およびパスワード変更の方法を説明します。

hr アカウントをロック解除し、パスワードを変更するには、次の手順を実行します。

1. 新規の SQL*Plus セッションを開始し、ユーザー SYSTEM などの管理者権限を持つユーザーとしてログインします。1-5 ページの「SQL*Plus の概要」を参照してください。
2. SQL プロンプトで、次の文を入力します。

   安全なパスワードを選択します。パスワード選択のガイドラインに関しては『Oracle Database セキュリティ・ガイド』を参照してください。

   ```
   SQL> ALTER USER hr ACCOUNT UNLOCK IDENTIFIED BY password;
   ```

   hr アカウントがロック解除され、パスワードが変更されたことが確認されます。

   ```
   User altered
   ```

SQL*Plus から Oracle Database への接続

hr アカウントがロック解除される際、1-7 ページの「ユーザー・アカウントのロック解除」で設定した新しいパスワードを使用して新しい hr 接続を作成できます。

SQL*Plus で、HR 接続を作成するには、次の手順を実行します。

1. Oracle データベースへの現行の接続をクローズします。1-5 ページの「SQL*Plus の概要」の手順 6 を参照してください。
2. SQL*Plus を起動します。cmd.exe ウィンドウの c:>¥ プロンプトで、sqlplus を入力し、キーボードの [Enter] を押します。
3. SQL プロンプトで、hr を入力した後にパスワードを入力します。

   hr スキーマを介してデータベース・インスタンスに接続されます。

接続およびコマンド・ウィンドウ両方をクローズできます。

参照 :

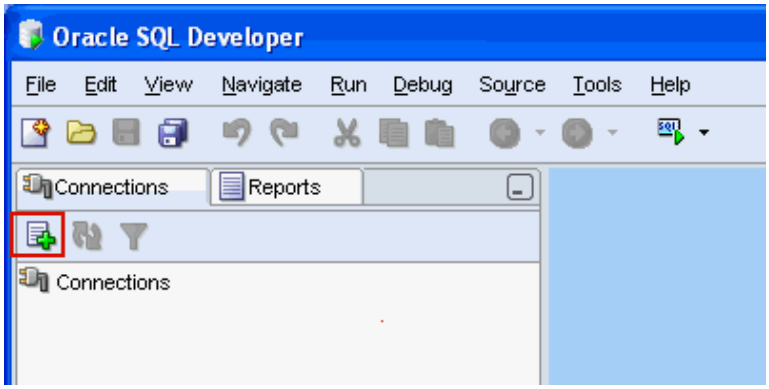
- 『SQL*Plus ユーザーズ・ガイドおよびリファレンス』

SQL Developer から Oracle Database への接続

hr アカウントがロック解除される際、Oracle Database 内の hr スキーマへのアクセスに使用できます。この項では、Oracle SQL Developer を使用して作業を行います。

SQL Developer で HR 接続を作成するには、次の手順を実行します。

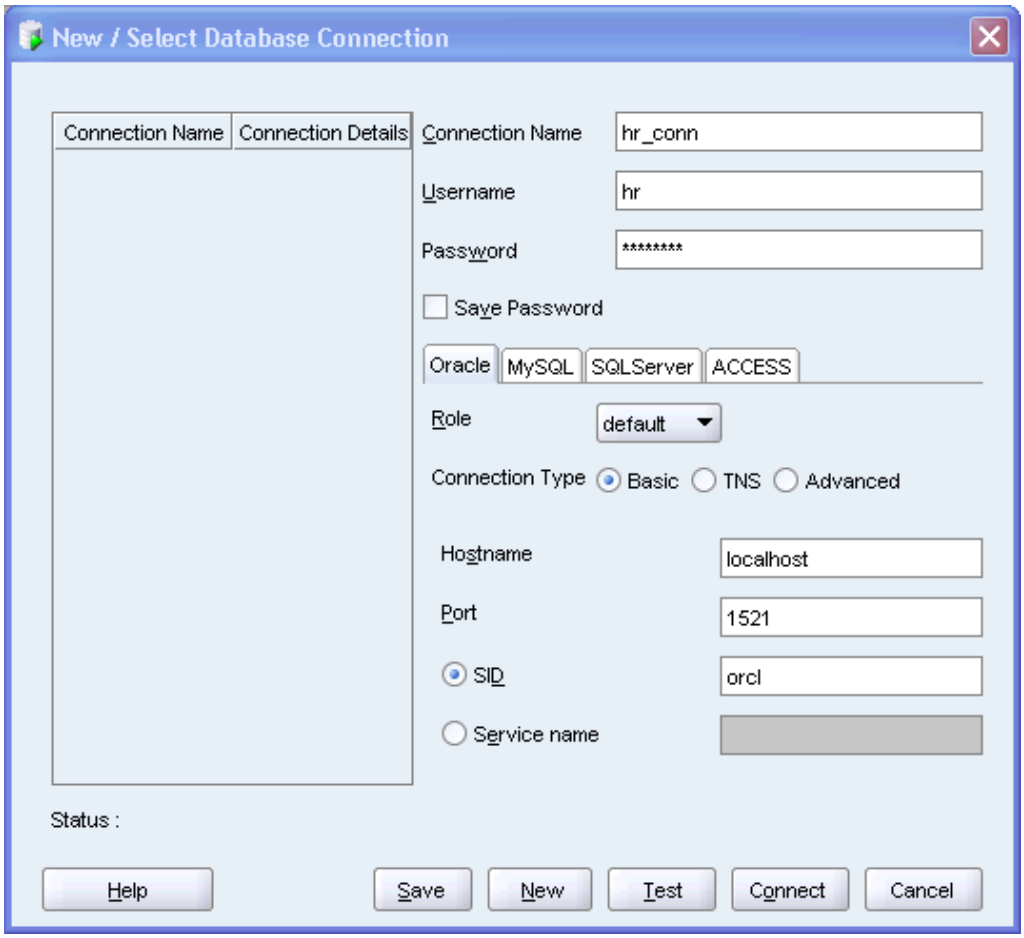
1. SQL Developer を起動します。
2. 「Connections」ペインで、**「New Connections」**アイコンをクリックします。



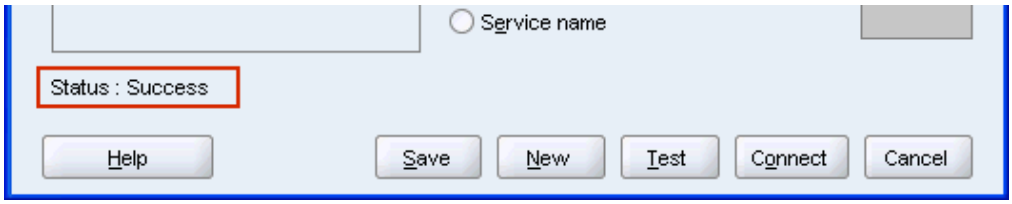
3. 「New/Select Database Connection」ウィンドウ上部で、次の情報を入力します。
   - **「Connection Name」**には、hr_conn と入力します。
   - **「Username」**には、hr と入力します。
   - **「Password」**には、hr アカウントのロック解除後にシステム管理者が作成したパスワードを入力します。パスワード・テキストはマスクされていることに注意してください。
   - **「Save Password」**オプションは、選択を解除したままにします。

   「New/Select Database Connection」ウィンドウの「Oracle」タブで、次の情報を入力します。
   - **「Role」**は「Default」を選択します。
   - **「Connection Type」**は「Basic」を選択します。
   - **「Hostname」**には、localhost と入力します。
   - **「Port」**には、1521 と入力します。
   - **「SID」**には、orcl と入力します。

   「New/Select Database Connection」ウィンドウ下部で、**「Test」**をクリックします。

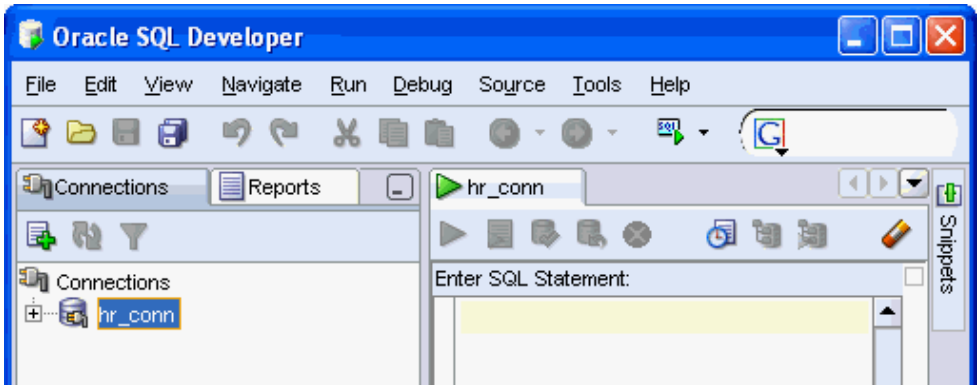


4. 接続がテストされます。「New/Select Database Connection」ウィンドウ下部のステータスが「Success」に変わります。



5. 「New/Select Database Connection」ウィンドウ下部で**「Connects」**をクリックします。

   新規の hr_conn 接続で「Oracle SQL Developer」ウィンドウが表示されます。



hr スキーマへの接続を正常に確立しました。

その他の開発環境の概要

この項では、独自のアプリケーション開発に使用できる他の開発環境および開発言語について説明します。

Visual Studio .NET には Oracle Data Provider for .NET、Oracle Database Extensions for .NET および Oracle Developer Tools が対応

Oracle Data Provider for .NET（ODP.NET）は、Microsoft .NET Framework クラス・ライブラリを使用および拡張して、.NET データを提供します。ODP.NET は .NET Framework を使用してプロバイダ固有の機能とデータ型を公開し、独自の Oracle Database API を使用して Oracle Database の強力な機能を .NET のアプリケーションに提供します。

Oracle Database Extensions for .NET は、Oracle Database 用 Microsoft Common Language Runtime（CLR）ホスト、ODP.NET クラスを介したデータ・アクセスおよび Oracle Deployment Wizard for .NET を提供します。CLR は Oracle Database サーバーの外部プロセスとして実行されるため、この統合により .NET ストアド・プロシージャおよびファンクションを Microsoft Windows XP、2000 および 2003 の Oracle Database 上で実行できます。これらのストアド・プロシージャおよびファンクションは、C# や VB.NET など、任意の .NET 準拠言語を使用して記述でき、PL/SQL または Java ストアド・プロシージャと同様に Oracle Deployment Wizard for .NET を使用して Oracle Database にデプロイできます。

Oracle Developer Tools は Visual Studio .NET を介して Oracle Database 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ます。Oracle Developer Tools にはデータベース・スキーマを参照するための Oracle Explorer、スキーマ・オブジェクトを作成および変更するためのウィザードとデザイナ、.NET デザイン・フォーム上にスキーマ・オブジェクトをドラッグしてコードを自動生成できる機能、および統合された状況依存ヘルプを使用した PL/SQL エディタなどの機能が含まれています。さらに Oracle Data Window を使用してデータベースの定型作業やストアド・プロシージャのテストを Visual Studio 環境で実行できる他、「SQL 問合せ」ウィンドウを使用して SQL 文やスクリプトを実行できます。

Oracle Database による .NET アプリケーション開発の概要は、『Oracle Database 2 日で .Net 開発者ガイド』を参照してください。

Oracle Database による .NET アプリケーション開発に関するその他のドキュメントには、『Oracle Data Provider for .NET 開発者ガイド』および『Oracle Database Extensions for .Net 開発者ガイド』があります。

Oracle Database の .NET API、ODP.NET、Oracle Developer Tools、ダウンロード、チュートリアルおよび関連情報の詳細は、次の Oracle Technology Network の .NET サイトを参照してください。

```
http://www.oracle.com/technology/tech/dotnet/
```

PHP

Hypertext Preprocessor、いわゆる PHP は、ウェブ・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 型ライセンスでデプロイされています。PHP は Oracle Database 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 Database による PHP アプリケーション開発の概要は、『Oracle Database 2 日で PHP 開発者ガイド』を参照してください。

Oracle Database の PHP API および関連情報の詳細は、次の Oracle Technology Network の PHP のサイトを参照してください。

```
http://www.oracle.com/technology/tech/php/
```

Oracle Application Express

Oracle Application Express、いわゆる APEX 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デプロイできるツールです。Application Builder が組み込まれており、HTML のインタフェースや、表またはストアド・プロシージャを使用するアプリケーションを、タブやボタン、ハイパーテキスト・リンクを介してリンクされたページに組み立てます。APEX の詳細は、『Oracle Database 2 日で Application Express 開発者ガイド』を参照してください。

APEX および関連情報の詳細は、次の Oracle Technology Network の APEX のサイトを参照してください。

`http://www.oracle.com/technology/products/database/application_express/`

Oracle Call Interface および Oracle C++ Call Interface

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

Oracle C++ Call Interface（OCCI）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 と非常に似ており、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。C++ の詳細は、『Oracle C++ Call Interface プログラマーズ・ガイド』を参照してください。

OCI および 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修正せずに使用でき、ディスク領域の使用を大幅に抑えることができます。Oracle Instant Client は、次の Oracle Technology Network の Oracle Instant Client のサイトから利用できます。

`http://www.oracle.com/technology/tech/oci/instantclient/`

Oracle Database の OCI および関連情報の詳細は、次の Oracle Technology Network の OCI のサイトを参照してください。

`http://www.oracle.com/technology/tech/oci/`

Oracle Database の OCCI および関連情報の詳細は、次の Oracle Technology Network の OCCI のサイトを参照してください。

`http://www.oracle.com/technology/tech/oci/occi/`

Oracle Java Database Connectivity

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。

OCI および OCCI と同様に、JDBC は Oracle Instant Client Installation の一部です。これは、次の Oracle Technology Network の Instant Client のサイトから利用できます。

`http://www.oracle.com/technology/tech/oci/instantclient/`

JDBC API の詳細は、次の Sun 社の Developer Network を参照してください。

`http://java.sun.com/javase/technologies/database/`

Oracle DatabaseJDBC API、ドライバ、サポートの有無の通知および同様の情報の詳細は、次の Oracle Technology Network を参照してください。

`http://www.oracle.com/technology/tech/java/sqlj_jdbc/`

Java を使用して Oracle Database のデータにアクセスおよびデータを修正する方法の概要は、『Oracle Database 2 日で Java 開発者ガイド』を参照してください。

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、Oracle Database 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。Oracle Database の UNIX プラットフォーム用ドライバは、次の Oracle Technology Network の ODBC のサイトから利用できます。

`http://www.oracle.com/technology/tech/windows/odbc/`

unixODBC のスタンダードおよび最新のドライバ・マネージャの詳細は、次の unixODBC のサイトを参照してください。

`http://www.unixodbc.org/`

Windows の Oracle ODBC ドライバの使用方法は、『Oracle Services for Microsoft Transaction Server 開発者ガイド』を参照してください。

Linux 上での Oracle ODBC ドライバの使用方法は、Oracle データベースの管理者リファレンスを参照してください。

OCI、OCCI および JDBC と同様に、ODBC は Oracle Instant Client Installation の一部です。これは、次の Oracle Technology Network の Instant Client のサイトから利用できます。

`http://www.oracle.com/technology/tech/oci/instantclient/`

2

データの問合せおよび操作

この章では、データベースを参照し、情報を引き出し、既存表の情報を変更し、トランザクション処理を制御する方法を示します。

この章の内容は次のとおりです。

データベース・オブジェクトの参照

表以外にも、Oracle Database には様々なオブジェクト型が用意されています。拡張性の高い管理オプションが多く含まれているオブジェクトもあり、それらには類似したプロパティが含まれます。たとえば、データベースの各オブジェクトは、1 つのスキーマのみに所属しており、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。このため、オブジェクトのネーミング規則および方法は、適切なスキーマを使用して新規のオブジェクトおよびオブジェクト型の識別を明確にすることをお薦めします。ここでユーザーが使用するオブジェクトは、すべて同一の hr スキーマに属しています。一般的に、アプリケーションは同一スキーマ内のオブジェクトを使用して動作します。

独自のオブジェクトを作成する場合は、オブジェクト名は 30 文字を超えることはできず、文字で始まる必要があります。

- 「スキーマ・オブジェクト型の確認」(2-2 ページ)
- 「表の参照およびデータの表示」(2-4 ページ)

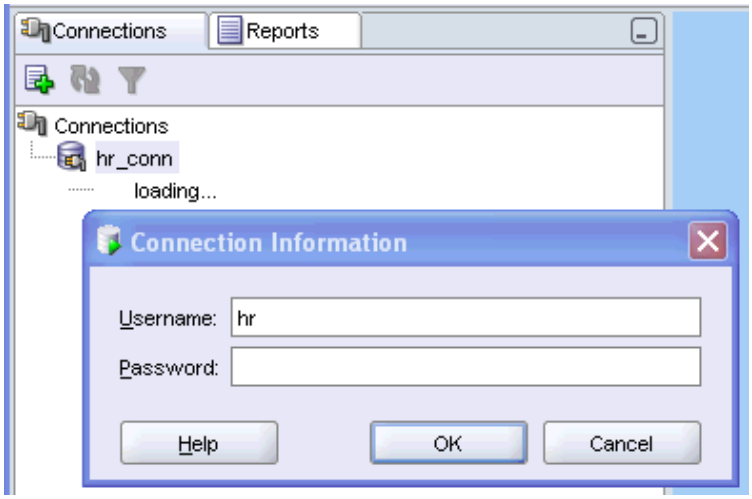
スキーマ・オブジェクト型の確認

この項では、hr サンプルおよびその属性、またはデータベース・オブジェクトに関する理解をさらに深めることができます。Oracle SQL Developer で参照してこれらのオブジェクトを表示する方法について紹介します。

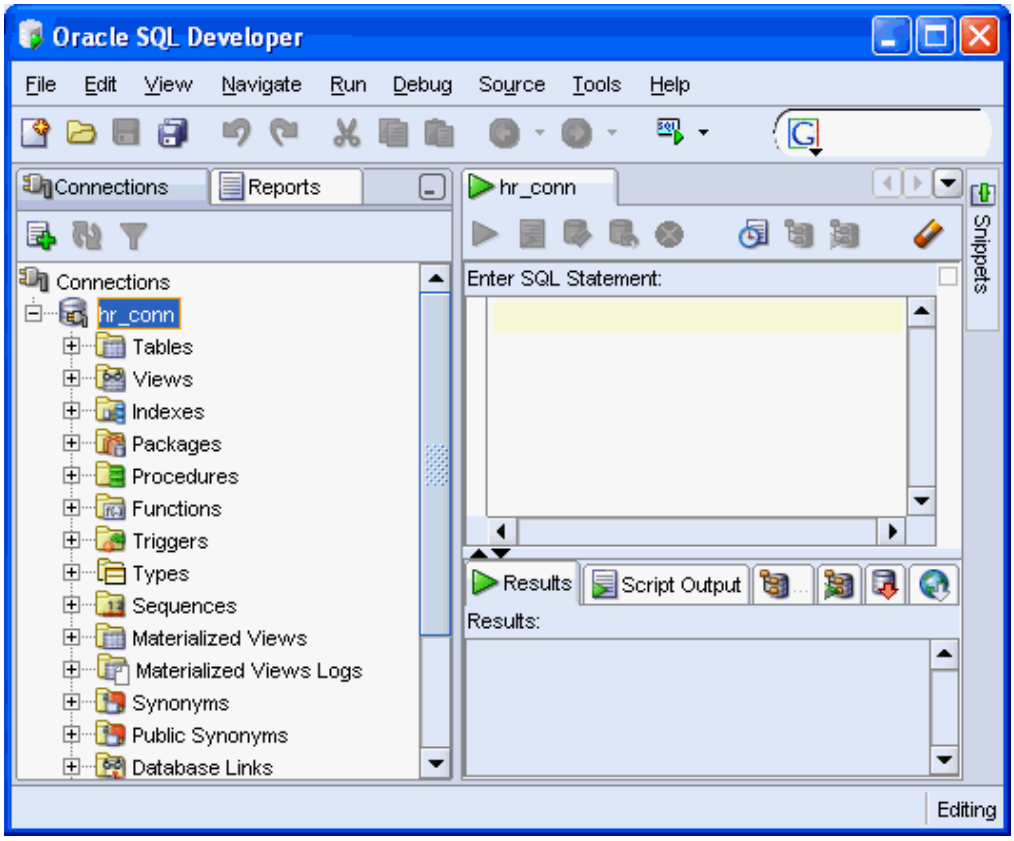
まず、各スキーマに含まれるオブジェクトのタイプを調べます。

HR スキーマを参照するには、次の手順を実行します。

1. Oracle SQL Developer を起動します。
2. SQL Developer ナビゲーション階層の「Connections」タブで、hr_conn の横にあるプラス記号をクリックします。
3. 「Connection Information」ダイアログでパスワードを入力し、hr スキーマへの接続を認証します。**「OK」**をクリックします。



4. 「Connections」ナビゲーション階層で、hr_conn の横にあるプラス記号をクリックし、hr スキーマ・データベース・オブジェクトのビューを拡張表示します。



表、ビュー、索引、パッケージ、プロシージャ、ファンクション、トリガー、タイプ、順序などの多くのオブジェクトがスキーマに含まれています。ここでは、使用する頻度の高いデータベース・オブジェクトの各タイプに関する定義を簡単に説明します。

- 表は Oracle Database 内のデータ記憶領域の基本的な単位であり、すべてのユーザーがアクセス可能であるデータを保持します。
- ビューは、1 つ以上の表または他のビューをデータの表示に関してカスタマイズしたものです。
- 索引は、オプションの構造であり、表のデータ取得のパフォーマンスを改善するために作成されます。
- ファンクションは、PL/SQL プログラミング・オブジェクトで、データベース内で格納および実行されます。ファンクションにより値が戻されます。
- プロシージャは、PL/SQL プログラミング・オブジェクトで、データベース内で格納および実行されます。プロシージャは値を戻しません。
- パッケージには、データベース内で格納および実行されるプロシージャまたはファンクションが含まれます。
- トリガーとは、表、ビュー、イベントに関連付けられたストアド・プロシージャまたはストアド・ファンクションです。トリガーはアクションを追跡するためにイベントの前後にコールされることで、誤操作を回避したり、明示的にビジネス・ルールに従うために新規データを変更したり、操作またはイベントのレコードを記録できます。
- タイプは、表の列、またはプロシージャやファンクションの引数に使用できる値とプロパティの固定セットを関連付けます。Oracle Database では、他のデータ型の値とは別に 1 つのデータ型の値を扱います。
- 順序は一意の整数を生成するために使用します。順序を使用することで自動的に主キーの値を生成できます。

表の参照およびデータの表示

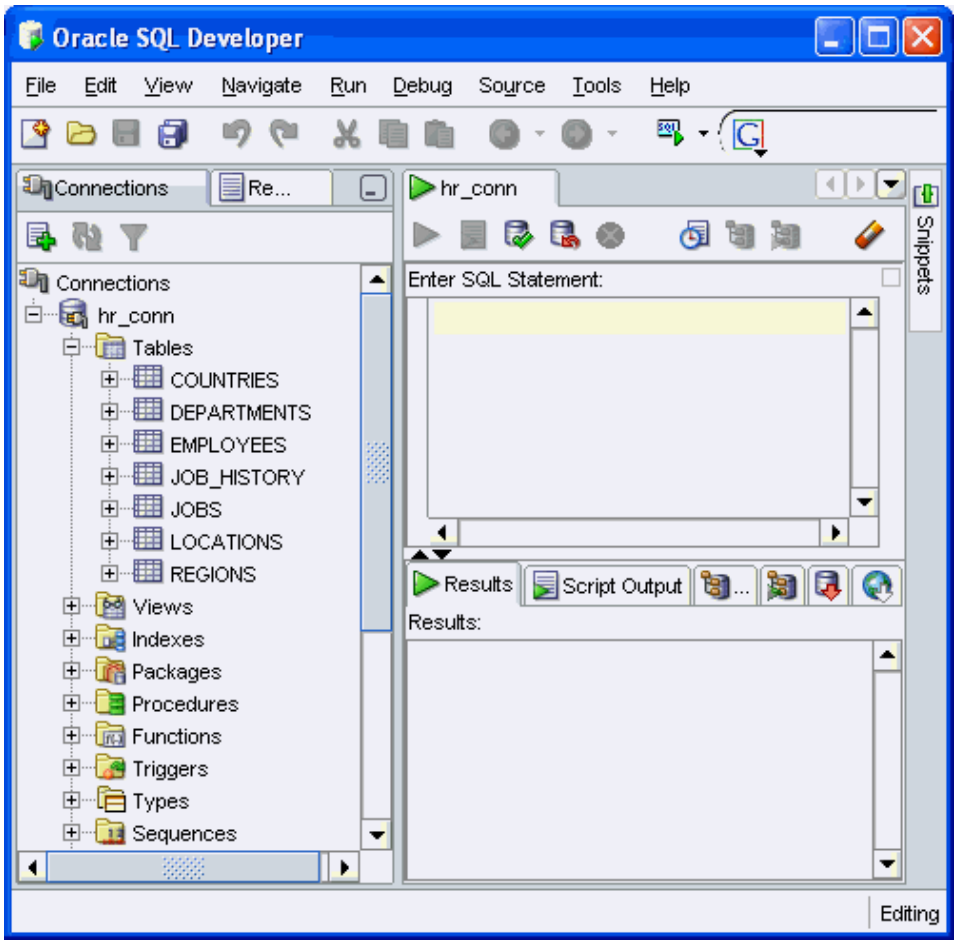
この項では、データベース表のプロパティの検索方法および、表に含まれるデータの表示方法について紹介します。

Oracle Database 表は、基本的なデータ・コンテナです。ユーザーがアクセスできるすべてのデータは、データベース・スキーマの表に含まれています。各表は個々のレコードである行、および各レコードの様々なフィールドを表す列を持つ 2 次元オブジェクトです。

表を表示するには、次の手順を実行します。

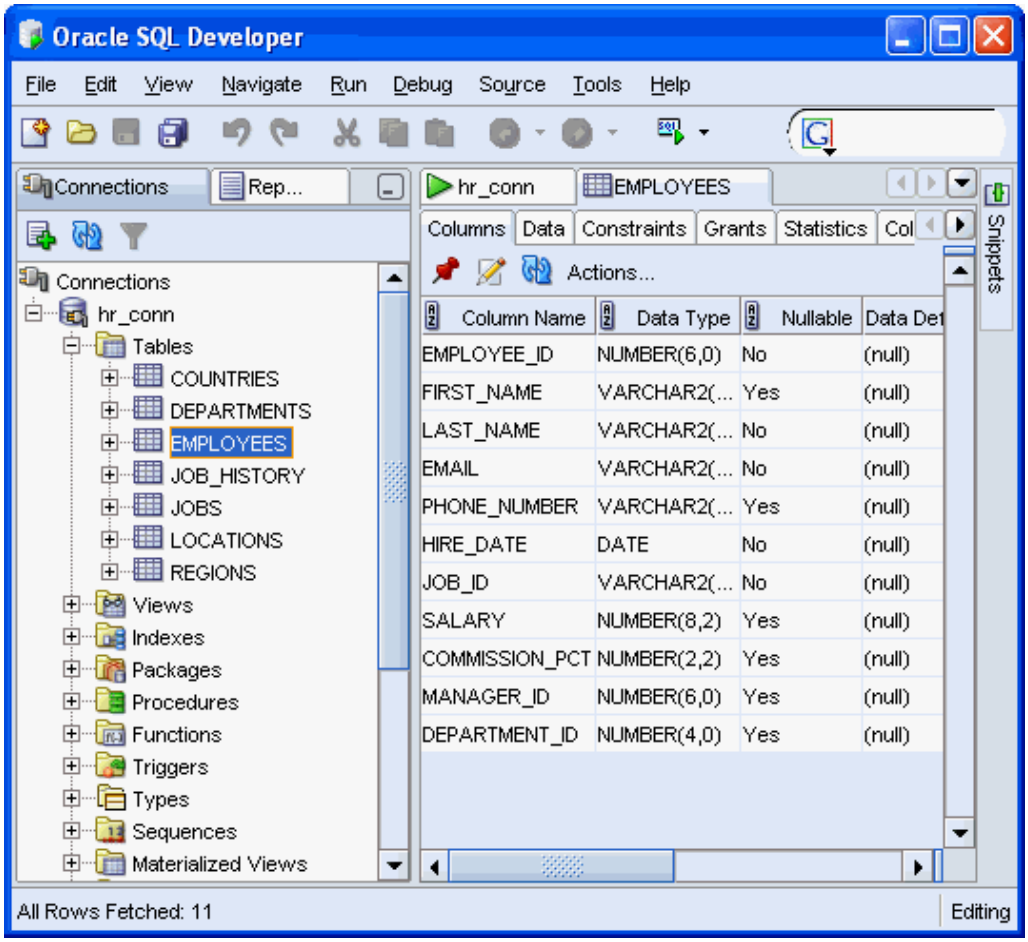
1. 「Connections」ナビゲーション階層で、表の横にあるプラス記号をクリックし、hr スキーマ内の表のリストを拡張表示します。

   表の拡張リストには、表 countries、departments、employees、job_history、jobs、locations および regions が含まれます。



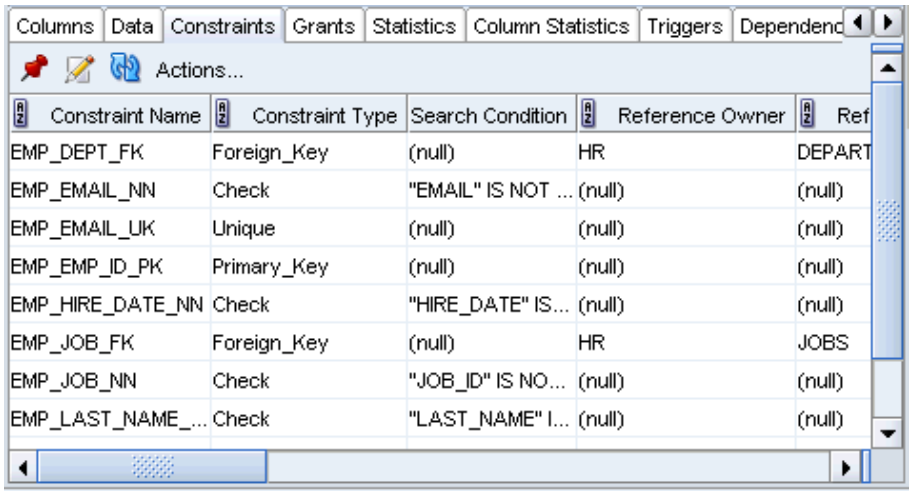
2. employees 表をクリックします。

   「Oracle SQL Developer」ウィンドウの右側にある「Columns」タブに、この表のすべての列（EMPLOYEE_ID、FIRST_NAME、LAST_NAME、EMAIL、PHONE_NUMBER、HIRE_DATE、JOB_ID、SALARY、COMMISSION_PCT、MANAGER_ID および DEPARTMENT_ID）のリストが表示されます。表の各列には関連付けられたデータ型が含まれていて、それぞれ文字データ、整数、浮動小数点数、データまたは時間に関する情報として定義されます。列のすべてのプロパティを表示するには、水平スクロール・バーを右に動かします。



3. 「**Constraints**」タブをクリックします。

   この表で使用されるすべての制約が表示されます。制約のタイプおよび制約の参照表や、制約が有効かどうか、その他のプロパティなどが含まれます。



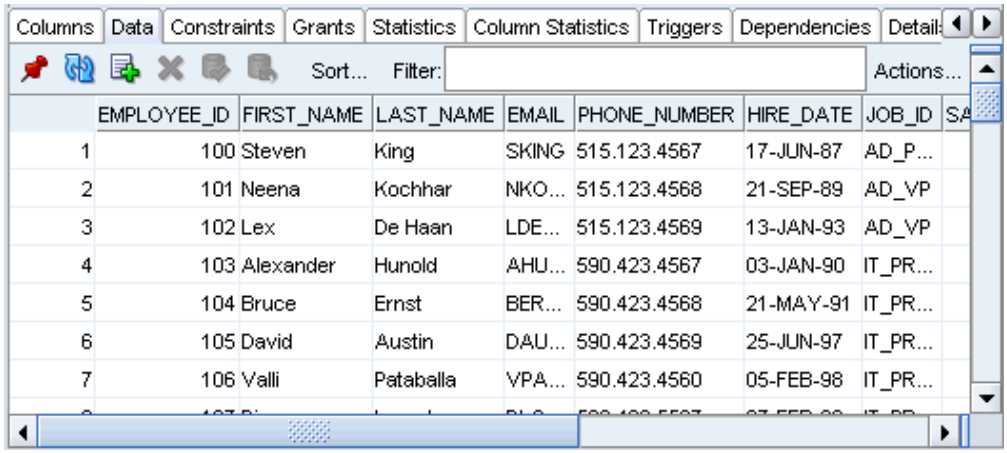
4. 同様に、適切なタブをクリックして、その他の表のプロパティを参照できます。

   - 「**Grants**」タブには、表の権限についての説明があります。
   - 「**Statistics**」タブには、レコード数、表が使用するメモリーのブロック数、行の長さの平均などの表内のデータのプロパティについての説明があります。
   - 「**Column Statistics**」タブには、各列に対する明確なエントリの数、下限値および上限値などがリストされています。
   - 「**Triggers**」タブには、表に関連付けられるトリガーが、トリガーのタイプおよびトリガー・イベントなどと一緒にリストされています。
   - 「**Dependencies**」タブには、トリガーやビューなど、この表に依存するすべてのオブジェクトがリストされています。
   - 「**Details**」タブには、作成日、所有者（hr）、名前、パーティション化情報などの表のその他の詳細がリストされています。
   - 「**Indexes**」タブには、表の列に定義される索引が状態やタイプと一緒にリストされています。
   - 「**SQL**」タブには、表 `employees` の定義の前述の情報が要約されています。表には、列定義、索引などが含まれます。

表内のデータを表示するには、次の手順を実行します。

「Oracle SQL Developer」ウィンドウの右側にある「**Data**」タブをクリックします。

この表のすべてのレコードのリストが表示されます。表の各列には文字データ、整数、浮動小数点、日付、または時間情報として定義する関連付けされたデータ型が含まれます。列のすべてのプロパティを表示するには、横軸のスクロール・バーを右へ動かします。



問合せを使用したデータの取得

問合せとは、データを 1 つ以上の表またはビューから取得する操作のことです。最上位の SELECT 文により問合せの結果が戻され、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。

この項では、問合せおよび副問合せのいくつかの型について紹介します。

参照 :

- 『Oracle Database SQL 言語リファレンス』

表からのデータの選択

次は、単純な問合せ形式の一例です。

```
SELECT select_list FROM source_list
```

この例では、*select_list* はデータが取得される列を指定し、*source_list* はこれらの列を含む表またはビューを指定しています。列数、各列のデータ型および長さは選択リストの要素で決定されます。選択リストは SQL ファンクションを使用できることに注意してください。

表にあるすべての列を表示するには、*select_list* に対して * を使用します。

例 2-1 では、「Data」ウィンドウの employees 表の表示によって、以前に表示した情報を戻すために SELECT 文を使用します。

例 2-1 表のすべての列の選択

1. 「SQL Worksheet」ペインで、次を入力します。

```
SELECT * FROM employees;
```

2. 「SQL Worksheet」ウィンドウで、**「Run Script」**アイコンをクリックします。あるいは [F5] のショートカット・キーも使用できます。

3. **「Script Output」**タブをクリックすると、「SQL Worksheet」ペインの下に問合せの結果が表示されます。

```
EMPLOYEE_ID          FIRST_NAME          LAST_NAME                ...
-------------------- ------------------- ------------------------ ...
100                  Steven              King                     ...
101                  Neena               Kochhar                  ...
102                  Lex                 De Haan                  ...
...
107 rows selected
```

異なる問合せの実行間に、ツールバーの消しゴムのアイコンをクリックすると「SQL Worksheet」ペインおよび「Script Output」ペインの両方をクリアできます。

例 2-2 は、問合せで要求した列、FIRST_NAME、LAST_NAME および HIRE_DATE のみを戻すための SELECT 文の使用方法を示しています。

例 2-2 表からの特定の列の選択

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

問合せの結果があります。

```
FIRST_NAME           LAST_NAME          HIRE_DATE
-------------------- ------------------ -------------------------
Steven               King               17-JUN-87
Neena                Kochhar            21-SEP-89
Lex                  De Haan            13-JAN-93
...
107 rows selected
```

列の別名の使用

新規のヘッダーを含む列を表示するには、列の正しい名前の直後に別名を使用して、レポート内の列の名前を変更できます。この別名により、問合せの期間に対する項目名が効果的に変更されます。

例 2-3 では、SELECT 文は問合せで要求した列を戻しますが、別名 name1、name2 および hired として指定した列のヘッダーを含みます。

例 2-3 単純な列の別名の使用

```
SELECT first_name name1, last_name name2, hire_date hired FROM employees;
```

問合せの結果は次のとおりです。

```
NAME1                NAME2               HIRED
-------------------- ------------------- -------------------------
Steven               King                17-JUN-87
Neena                Kochhar             21-SEP-89
Lex                  De Haan             13-JAN-93
...
107 rows selected
```

使用する別名に大文字、小文字、スペースまたは組合せが含まれる場合、二重引用符（"）を使用する必要があります。

例 2-4 では、SELECT 文を使用して指定した別名の列のヘッダー、First、Last および Date Started を含む列を戻します。

例 2-4 引用符で囲まれた別名の列の使用

```
SELECT first_name "First", last_name "Last", hire_date "Date Started"
FROM employees;
```

問合せの結果は次のとおりです。

```
First                Last                 Date Started
-------------------- -------------------- -------------------------
Steven               King                 17-JUN-87
Neena                Kochhar              21-SEP-89
Lex                  De Haan              13-JAN-93
...
107 rows selected
```

特定の条件に一致させるためのデータの制限

SELECT および FROM キーワード以外に、その他の一般的な句も問合せで使用できます。WHERE 句を比較演算子と一緒に使用すると、表内のすべての行を戻すかわりに取得する行を選択できます。

この表に WHERE 句で使用される比較演算子をリストします。

比較演算子	定義
=	等価かどうかをテストします。
!=, <>	非等価かどうかをテストします。
>	より大きいかどうかをテストします。
>=	以上かどうかをテストします。
<	より小さいかどうかをテストします。
<=	以下かどうかをテストします。
BETWEEN *a* AND *b*	2 つの値の範囲内に適合するかどうかをテストします
LIKE	ゼロまたは複数の文字に対するワイルドカード記号（%）、または単一の文字に対してアンダースコア（_）を使用して、文字列の一致をテストします。
IN()	指定した値リストに一致しているかどうかをテストします。
NOT IN()	指定した値リストに一致がないことをテストします。
IS NULL	値が NULL であることをテストします。
IS NOT NULL	値が NULL でないことをテストします。

WHERE 句を使用すると単一条件をテストでき、AND 句を使用すると複数のテストを結合できます。

例 2-5 は、department_id に 90 が含まれる単一の部門を制限する列の値を戻す WHERE 句の使用方法を示しています。

例 2-5 単一条件のテスト

```
SELECT first_name "First", last_name "Last"
FROM employees
WHERE department_id=90;
```

問合せの結果が表示されます。

```
First                Last
-------------------- -------------------------
Steven               King
Neena                Kochhar
Lex                  De Haan

3 rows selected
```

例 2-6 は、11,000 と同一またはそれ以上の給与および割り当てられた（NULL ではない）手数料率を一致させるために 2 つの異なる条件を制限する行を戻す WHERE ... AND 句の使用方法を示しています。

例 2-6 複数の条件のテスト

```
SELECT first_name "First", last_name "Last",
SALARY "Salary", COMMISSION_PCT "%"
FROM employees
WHERE salary >=11000 AND commission_pct IS NOT NULL;
```

問合せの結果が表示されます。

```
First                Last                 Salary               %
-------------------- -------------------- -------------------- -----
John                 Russell              14000                0.4
Karen                Partners             13500                0.3
Alberto              Errazuriz            12000                0.3
...
6 rows selected
```

例 2-7 では、WHERE 句を使用して姓が Ma で始まる Mallin、Markle、Marlow、Marvins、Matos および Mavris の 6 行が戻されます。一致式 %ma%（テキスト ma が列のどこにある場合も可という意味）を使用した場合には、結果には Kumar、Urman および Vollman の 3 行のみが含まれます。

例 2-7 一致する文字列のテスト

```
SELECT first_name "First", last_name "Last"
FROM employees
WHERE last_name LIKE 'Ma%';
```

問合せの結果が表示されます。

```
First                Last
-------------------- -------------------------
Jason                Mallin
Steven               Markle
James                Marlow
...
6 rows selected
```

例 2-8 は、100、110、120 の値のリストと department_id が一致する複数の異なる部門で勤務する従業員を検索するための WHERE ... IN 句の使用方法を示しています。結果は 8 行を含み、リストの最初の値と一致した 4 行、リストの 2 番目の値と一致した 2 行が含まれます。120 と一致した値はありません。

例 2-8 値リストでの一致のテスト

```
SELECT first_name "First", last_name "Last", department_id "Department"
FROM employees
WHERE department_id IN (100, 110, 120);
```

問合せの結果が表示されます。

```
First               Last                     Department
------------------- ------------------------ ----------------------
John                Chen                     100
Daniel              Faviet                   100
William             Gietz                    110
...
8 rows selected
```

対応する department_id 値のわからない、特定の部門で働く従業員を検索する場合、employees および departments 表の両方を検索する必要があります。JOIN 操作で、2 つの表を結合して確認できます。

employees.employee_id のような完全修飾された列名はオプションです。ただし、同一の列名を持つ 2 つ以上の表を問合せに使用する場合、表に対してこれらの列を識別する必要があります。たとえば、employees.department_id および departments.department_id を同時に使用すると、従業員が働く部門の名前を決定できます。

完全修飾された列名の使用時には、表の名称 departments に d などの別名を使用した場合、問合せがさらに読みやすくなります。列 departments.department_id は d.department_id に、employees.department_id は e.department_id になります。問合せの FROM 句でこれらの表の別名を作成する必要があります。

例 2-9 では、この結果セットに 2 つの異なる表の列が含まれます。レポートの列名は一意であるため、表名で修飾する必要はありません。ただし、WHERE 句では、2 つの異なる表から同じ列名を使用したため、修飾が必要になります。

例 2-9 別の表の値のテスト

```
SELECT e.first_name "First", e.last_name "Last", d.department_name "Department"
FROM employees e, departments d
WHERE e.department_id = d.department_id;
```

問合せの結果が表示されます。

```
First               Last                     Department
------------------- ------------------------ ------------------------------
Jennifer            Whalen                   Administration
Michael             Hartstein                Marketing
Pat                 Fay                      Marketing
...
106 rows selected
```

データのパターン検索

正規表現を使用することにより、一般的な構文規則を使用して文字順序における複雑なパターンを検索できます。正規表現は、検索アルゴリズムを指定する**メタ文字**および文字を指定する**リテラル**を使用して検索パターンを定義します。

正規表現ファンクションには、REGEXP_INSTR、REGEXP_LIKE、REGEXP_REPLACE および REGEXP_SUBSTR が含まれます。

例 2-10 は、すべてのマネージャの検出方法を示しています。メタ文字 | は OR 条件を表し、マネージャの位置が部門に応じて %_MGR または %_MAN のどちらか一方に指定されるために使用する必要があります。オプション i は大 / 小文字を区別しない一致を指定します。

例 2-10 一致するデータ・パターンの検索

```
SELECT first_name "First", last_name "Last", job_id "Job"
FROM employees
WHERE REGEXP_LIKE (job_id, '(_m[an|gr])', 'i');
```

問合せの結果が表示されます。

```
First                Last                     Job
-------------------- ------------------------ ----------
Nancy                Greenberg                FI_MGR
Den                  Raphaely                 PU_MAN
Matthew              Weiss                    ST_MAN
...
14 rows selected
```

例 2-11 は、REGEXPR_LIKE 式で last_name に二重母音（a、e、i、o または u のいずれか 2 つが隣り合って発生する場合）が含まれている行を選択する方法を示しています。REGEXP_LIKE 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 2-11 一致するデータ・パターンの検索（隣接する文字）

```
SELECT first_name "First", last_name "Last"
FROM employees
WHERE REGEXP_LIKE (last_name, '([aeiou])¥1', 'i');
```

問合せの結果が表示されます。

```
First                Last
-------------------- -------------------------
Harrison             Bloom
Lex                  De Haan
Kevin                Feeney
...
8 rows selected
```

データ・パターンを検出して別のデータ・パターンに置き換えるには、REGEXPR_REPLACE を使用します。例 2-12 では、電話番号の形式 'nnn.nnn.nnnn' を '(nnn) nnn-nnnn' に置き換えます。数字はメタ文字 [:digit] と一致し、メタ文字 {n} は発生数を表します。メタ文字 '.' は通常、式のあらゆる文字を示します。そのため、メタ文字 ¥ はエスケープ文字として使用され、次のパターンの文字はリテラルになります。この結果セットには新しい形式で電話番号が表示されます。REGEXP_REPLACE 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 2-12 データ・パターンの置換

```
SELECT first_name "First", last_name "Last",
phone_number "Old Number",
REGEXP_REPLACE(phone_number,
                '([[:digit:]]{3})¥.([[:digit:]]{3})¥.([[:digit:]]{4})',
                '(¥1) ¥2-¥3') "New Number"
FROM employees
WHERE department_id = 90;
```

問合せの結果が表示されます。

```
First                Last                 Old Number           New Number
-------------------- -------------------- -------------------- --------------
Steven               King                 515.123.4567         (515) 123-4567
Neena                Kochhar              515.123.4568         (515) 123-4568
Lex                  De Haan              515.123.4569         (515) 123-4569

3 rows selected
```

例 2-13 は、REGEXPR_SUBSTR ファンクションを使用してパターンと一致する最初の部分文字列を検索する方法を示します。メタ文字 + はパターンの複数の発生を示しています。この結果セットは数字およびダッシュ記号を street_address 列から抽出します。REGEXP_SUBSTR 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。

例 2-13 部分文字列の戻り

```
SELECT street_address, REGEXP_SUBSTR(street_address,
'[[:digit:]-]+', 1, 1) "Street Numbers"
FROM locations;
```

問合せの結果が表示されます。

```
STREET_ADDRESS                         Street Numbers
-------------------------------------- ----------------------
1297 Via Cola di Rie                   1297
93091 Calle della Testa                93091
2017 Shinjuku-ku                       2017
...
23 rows selected
```

REGEXPR_INSTR ファンクションを使用すると、パターンと一致する最初の部分文字列の位置を検索できます。例 2-14 では、空白文字「 」を検索するために REGEXPR_INSTR を使用します。メタ文字 +' はパターンの複数の発生を示しています。この結果セットは各アドレスの最初の空白を示します。REGEXP_INSTR 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。

例 2-14 部分文字列の位置の戻り

```
SELECT street_address, REGEXP_INSTR(street_address, '[ ]+', 1, 1) "Position"
FROM locations;
```

問合せの結果が表示されます。

```
STREET_ADDRESS                         Position
-------------------------------------- ----------------------
1297 Via Cola di Rie                   5
93091 Calle della Testa                6
2017 Shinjuku-ku                       5
...
23 rows selected
```

ファンクション REGEXPR_COUNT は文字列で指定した文字パターンの繰返しの回数を決定します。例 2-15 では、REGEXPR_COUNT は表 locations の street_address 列で発生した空白文字の回数を戻します。REGEXP_COUNT 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。

例 2-15 部分文字列の発生回数の戻り

```
SELECT street_address, REGEXP_COUNT(street_address, ' ', 1) "Number of Spaces"
FROM locations;
```

問合せの結果が表示されます。

```
STREET_ADDRESS                           Number of Spaces
---------------------------------------- ----------------------
1297 Via Cola di Rie                     4
93091 Calle della Testa                  3
2017 Shinjuku-ku                         1
...
23 rows selected
```

この結果セットに、各住所の空白の数が表示されます。

参照：

- 正規表現の構文の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

データのソート

SQL では、**ORDER BY** 句は、結果データのソートに使用される列の識別に使用します。ソート条件は結果セットに含まれる必要はなく、式、列名、計算操作、ユーザー定義ファンクションなどが含まれます。

例 2-16 は、昇順で last_name 順にソートされた結果セットを戻す ORDER BY 句を示しています。

例 2-16 引用符で囲まれた別名の列の使用

```
SELECT first_name "First", last_name "Last", hire_date "Date Started"
FROM employees
ORDER BY last_name;
```

問合せの結果が表示されます。

```
First                Last                     Date Started
-------------------- ------------------------ -------------------------
Ellen                Abel                     11-MAY-96
Sundar               Ande                     24-MAR-00
Mozhe                Atkinson                 30-OCT-97
...
107 rows selected
```

ビルトイン・ファンクションおよび集計ファンクションの使用

SQL 算術演算子およびその他のビルトイン・ファンクションを使用することで、表内でソートされたデータを直接計算できます。

参照 :

- 使用可能なすべての 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。

算術演算子の使用

Oracle Database SQL では、加算に使用するプラス記号（+）、減算に使用するマイナス記号（-）、乗算に使用するアスタリスク（*）、除算に使用するフォワードスラッシュ（/）などの基本的な算術演算子がサポートされています。これらは、評価順の標準的な算術ルールに従って評価されます。

例 2-17 では、歩合制に適格歩合による賃金の受取りに見合った従業員によって稼がれた給与が雇用日の順序で結果セットに表示されます。

例 2-17 演算式の評価

```
SELECT first_name "First", last_name "Last", salary * 12 "Annual Compensation"
FROM employees
WHERE commission_pct IS NOT NULL
ORDER BY hire_date;
```

問合せの結果が表示されます。

```
First                Last                      Annual Compensation
-------------------- ------------------------- ----------------------
Janette              King                      120000
Patrick              Sully                     114000
Ellen                Abel                      132000
...
35 rows selected
```

数値ファンクションの使用

Oracle Database には、数値操作のための数値ファンクションが数多く用意されています。たとえば、指定した小数点以下で切り下げる ROUND、指定した小数点で切り捨てる TRUNC などです。これらのすべてのファンクションにより、評価された各行に対する単一の値が戻されます。

例 2-18 は、セントの位までの概数にした日給の決定方法を示しています。

例 2-18 数値データの概数化

```
SELECT first_name "First", last_name "Last",
ROUND(salary/30, 2) "Daily Compensation"
FROM employees;
```

問合せの結果が表示されます。

```
First                Last                      Daily Compensation
-------------------- ------------------------- ----------------------
Steven               King                      800
Neena                Kochhar                   566.67
Lex                  De Haan                   566.67
...
107 rows selected
```

例 2-19 は、ドルの単位まで切り捨てた日給の決定方法を示しています。TRUNC ファンクションは値を繰り上げしないことに注意してください。

例 2-19 数値データの切捨て

```
SELECT first_name "First", last_name "Last",
  TRUNC(salary/30, 0) "Daily Compensation"
FROM employees;
```

問合せの結果が表示されます。

```
First                Last                      Daily Compensation
-------------------- ------------------------- ----------------------
Steven               King                      800
Neena                Kochhar                   566
Lex                  De Haan                   566
...
107 rows selected
```

参照 :

- 数値の 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。

文字ファンクションの使用

Oracle Database には、文字の値をカスタマイズするために、拡張性の高い文字ファンクションのリストが用意されています。

これらのファンクションは、文字式を大文字または小文字に変更し、空白を削除し、文字列を連結し、部分文字列を抽出または削除できます。

例 2-20 は、式の大 / 小文字の変更方法を示しています。結果セットには UPPER、LOWER および INITCAP ファンクションの結果が表示されます。

例 2-20 文字データの大 / 小文字の変更

```
SELECT UPPER(first_name) "First upper",
LOWER(last_name) "Last lower",
INITCAP(email) "E-Mail"
FROM employees;
```

問合せの結果が表示されます。

```
First upper          Last lower                E-Mail
-------------------- ------------------------- -------------------------
STEVEN               king                      Sking
NEENA                kochhar                   Nkochhar
LEX                  de haan                   Ldehaan
```

レポートの同一の列で 2 つの個別の列または式から情報を生成するには、連結演算子 || を使用して個別の結果を連結できます。例 2-21 では、4 つの連結演算子が実行されていることにも注意してください。この結果セットには、first_name 値の直後に last_name 値がリストされた列 Name に簡単な連結ファンクションが表示され、列 Location でネストした連結ファンクションは、city および country_name 値に区別されます。

例 2-21 文字データの連結

```
SELECT e.first_name || ' ' || e.last_name "Name",
  l.city || ', ' || c.country_name "Location"
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;
```

問合せの結果が表示されます。

```
Name                          Location
----------------------------  ----------------------------------------
Ellen Abel                    Oxford, United Kingdom
Sundar Ande                   Oxford, United Kingdom
Mozhe Atkinson                South San Francisco, United States of America
...
106 rows selected
```

RTRIM および LTRIM ファンクションを使用して、文字データの先頭または末尾から文字（デフォルトでは空白）を削除できます。TRIM ファンクションは最初の文字およびそれに続く文字の両方を削除できます。例 2-22 では、タイプ変換ファンクション TO_CHAR を使用します。この結果セットには、M で始まらない last_name 値を持ち、job_id 値の末尾に MAN を含まず、date_hired 値が 0 で始まらないすべての従業員が表示されます。

例 2-22　文字データの切捨て

```
SELECT LTRIM(last_name, 'M') "Last Name",
RTRIM(job_id, 'MAN') "Job",
TO_CHAR(TRIM(LEADING 0 FROM hire_date)) "Hired"
FROM employees
WHERE department_id=50;
```

問合せの結果が表示されます。

```
Last Name                 Job        Hired
------------------------- ---------- ---------
Weiss                     ST_        18-JUL-96
Fripp                     ST_        10-APR-97
Kaufling                  ST_        1-MAY-95
Vollman                   ST_        10-OCT-97
ourgos                    ST_        16-NOV-99
...
ikkilineni                ST_CLERK   28-SEP-98
Landry                    ST_CLERK   14-JAN-99
arkle                     ST_CLERK   8-MAR-00
...
arlow                     ST_CLERK   16-FEB-97
...
allin                     ST_CLERK   14-JUN-96
...
Philtanker                ST_CLERK   6-FEB-00
...
Patel                     ST_CLERK   6-APR-98
...
atos                      ST_CLERK   15-MAR-98
Vargas                    ST_CLERK   9-JUL-98
Taylor                    SH_CLERK   24-JAN-98
...
Geoni                     SH_CLERK   3-FEB-00
...
Cabrio                    SH_CLERK   7-FEB-99
...
Bell                      SH_CLERK   4-FEB-96
Everett                   SH_CLERK   3-MAR-97
cCain                     SH_CLERK   1-JUL-98
...
45 rows selected
```

RPAD を使用して文字を追加できます。デフォルトでは、文字データの末尾に空白が追加されます。LPAD ファンクションを使用すると、文字データの先頭に文字が追加されます。

例 2-23 では、この結果セットに、給与の相対値の単純なヒストグラムが表示されます。

例 2-23 文字データの埋込み

```
SELECT first_name || ' ' || last_name "Name",
RPAD(' ', salary/1000, '$') "Salary"
FROM employees;
```

問合せの結果が表示されます。

```
Name                                Salary
----------------------------------- ---------------
Steven King                         $$$$$$$$$$$$$$$$$$$$$$$
Neena Kochhar                       $$$$$$$$$$$$$$$$
Lex De Haan                         $$$$$$$$$$$$$$$$
...
107 rows selected
```

SUBSTR を使用して、開始文字位置および文字の総数により指定されたデータの部分文字列のみを抽出できます。

例 2-24 では、SUBSTR を使用して first_name 値を頭文字に短縮し、phone_number 値から地域コードを抜き出します。

例 2-24 文字データの部分文字列の抽出

```
SELECT SUBSTR(first_name, 1, 1) || '. ' || last_name "Name",
  SUBSTR(phone_number, 5, 8) "Phone"
FROM employees;
```

問合せの結果が表示されます。

```
Name                        Phone
--------------------------- --------
S. King                     123.4567
N. Kochhar                  123.4568
L. De Haan                  123.4569
...
107 rows selected
```

この結果セットに、頭文字に短縮された first_name 値、および語頭の地域コード・コンポーネントを含まない phone_number 値が表示されます。

文字データの関連する位置が判明している場合、REPLACE を SUBSTR と組み合せて使用して、特定の部分文字列を置換できます。

例 2-25 では、WHERE 句の SUBSTR を使用して、ジョブ・コードを略称に置換します。

例 2-25 文字データの部分文字列の置換

```
SELECT SUBSTR(first_name, 1, 1) || '. ' || last_name "Name",
REPLACE(job_id, 'SH', 'SHIPPING') "Job"
FROM employees
WHERE SUBSTR(job_id, 1, 2) = 'SH';
```

問合せの結果が表示されます。

```
Name                 Job
-------------------  ------------------------
W. Taylor            SHIPPING_CLERK
J. Fleaur            SHIPPING_CLERK
M. Sullivan          SHIPPING_CLERK
...
20 rows selected
```

この結果セットに、頭文字に短縮された first_name 値、および置換された job_id 値が表示されます。

参照：

- 文字の 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。

日付ファンクションの使用

Oracle Database には、間隔ファンクションなどのデータおよび日付データを操作および計算できるデータ・ファンクションが含まれています。

例 2-26 では、異なる役職に就いた従業員の特定の役職での雇用期間を決定します。従業員は一定期間に 2 つ以上の異なる役職に就いていた場合があるため、名称は一意ではないことに注意してください。MONTHS_BETWEEN 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 2-26 日付間の月数の決定

```
SELECT e.first_name || ' ' || e.last_name "Name",
TRUNC(MONTHS_BETWEEN(j.end_date, j.start_date)) "Months Worked"
FROM employees e, job_history j
WHERE e.employee_id = j.employee_id
ORDER BY "Months Worked";
```

問合せの結果が表示されます。

```
Name                                       Months Worked
------------------------------------------ -------------
Jonathon Taylor                                        9
Payam Kaufling                                        11
Jonathon Taylor                                       11
...
10 rows selected
```

この結果に、退社した従業員の雇用期間が最短で 9 か月、最長で 69 か月であると表示されます。

例 2-27 では、EXTRACT ファンクションを使用して、従業員が、連続する雇用において暦年の年目であるかどうかを決定します。EXTRACT ファンクションは、MONTH や DATE などと組み合せても使用できます。

SYSDATE ファンクションを使用するとシステム時計の現在の日付がわかります。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。

例 2-27 日付間の年の決定

```
SELECT first_name || ' ' || last_name "Name",
(EXTRACT(YEAR from SYSDATE) - EXTRACT(YEAR FROM hire_date)) "Years Employed"
FROM employees;
```

問合せの結果が表示されます。

```
Name                                       Years Employed
------------------------------------------ --------------
Steven King                                            20
Neena Kochhar                                          18
Lex De Haan                                            14
...
107 rows selected
```

従業員 'Steven King' の雇用年数（20 年）が最も長いことが結果に表示されます。

例 2-28 では、last_day ファンクションを使用して、従業員が雇用された月の末日を決定します。

例 2-28 特定の日付に対する月の末尾の取得

```
SELECT first_name || ' ' || last_name "Name", hire_date "Date Started",
  LAST_DAY(hire_date) "End of Month"
FROM employees;
```

問合せの結果が表示されます。

```
Name                          Date Started       End of Month
----------------------------- ------------------ ------------------------
Steven King                   17-JUN-87          30-JUN-87
Neena Kochhar                 21-SEP-89          30-SEP-89
Lex De Haan                   13-JAN-93          31-JAN-93
...
107 rows selected
```

各 hire_date 値に対する月の正確な末日が結果に表示されます。

例 2-29 では、ADD_MONTHS ファンクションを使用して、従業員が雇用された日付に 6 か月を追加します。ADD_MONTH 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 2-29 日付への月の追加

```
SELECT first_name || ' ' || last_name "Name", hire_date "Date Started",
  ADD_MONTHS(hire_date, 6) "New Date"
FROM employees;
```

問合せの結果が表示されます。

```
Name                  Date Started             New Date
--------------------- ------------------------ ------------------------
Steven King           17-JUN-87                17-DEC-87
Neena Kochhar         21-SEP-89                21-MAR-90
Lex De Haan           13-JAN-93                13-JUL-93
...
107 rows selected
```

例 2-30 では、SYSTIMESTAMP ファンクションを使用して現在のシステム時間および日付を決定します。SYSTIMESTAMP は SYSDATE と類似していますが、タイムゾーンおよび秒数を含んだ日の情報の時間を含みます。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。

hr スキーマ表のかわりに、既知の結果を保証するために参照できる、データ・ディクショナリの小表 DUAL を使用することに注意してください。DUAL 表の詳細は、『Oracle Database 概要』を、DUAL 表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。

例 2-30 システム日付および時間の取得

```
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;
```

問合せの結果が表示されます。

```
System Time and Date
------------------------------------------------------------
18:25:56, 4/5/2007
```

現在の SYSTIMESTAMP 値によっては結果が変更する可能性もあります。

参照 :

- 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。

データ型変換ファンクションの使用

Oracle Database には、異なるデータ型間で変換できるデータ・ファンクションが含まれています。これは、同一列内にある異なるデータ型のデータを表示する必要がある場合に特に有効です。

一般的に 3 つの変換ファンクションがあります。文字に対しては TO_CHAR ファンクション、数値に対しては TO_NUMBER ファンクション、日付に対しては TO_DATE ファンクション、そしてタイムスタンプに対しては TO_TIMESTAMP ファンクションです。

TO_CHAR ファンクションを使用して目的の書式にデータを変換します。例 2-31 では、HIRE_DATE 値を 'FMMonth DD YYYY' 書式に変換します。FM オプションを使用すると月名から先頭および末尾の空白をすべて削除でき、その他のオプションでは、'DD-MON-YYYY AD' や 'MM-DD-YYYY HH24:MI:SS' などを含めて使用できます。

例 2-31 書式テンプレートを使用したデータを変換する TO_CHAR の使用

```
SELECT first_name || ' ' || last_name "Name",
  TO_CHAR(hire_date, 'FMMonth DD YYYY') "Date Started"
FROM employees;
```

問合せの結果が表示されます。

```
Name                                       Date Started
------------------------------------------ -----------------
Steven King                                June 17 1987
Neena Kochhar                              September 21 1989
Lex De Haan                                January 13 1993
...
107 rows selected
```

新しい書式ですべての hire_date 値が結果セットにリストされます。

例 2-32 は、短い日付（DS）および長い日付（DL）という 2 つの標準書式タグを使用して、日付の書式を設定する方法を示しています。

例 2-32 標準書式を使用したデータを変換する TO_CHAR の使用

```
SELECT first_name || ' ' || last_name "Name",
  TO_CHAR(hire_date, 'DS') "Short Date",
  TO_CHAR(hire_date, 'DL') "Long Date"
FROM employees;
```

問合せの結果が表示されます。

```
Name                       Short Date  Long Date
-------------------------  ----------  -------------------------
Steven King                6/17/1987   Wednesday, June 17, 1987
Neera Kochhar              9/21/19889  Thursday, September 21, 1989
Lex De Haen                1/13/1993   Wednesday, January 13, 1993
...
107 rows selected
```

TO_CHAR ファンクションを使用して目的の通貨書式に数字を変換できます。例 2-33 では、SALARY 値を '$99,999.99' 書式に変換します。TO_CHAR の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

例 2-33 数字を通貨テンプレートに変換する TO_CHAR の使用

```
SELECT first_name || ' ' || last_name "Name",
  TO_CHAR(salary, '$99,999.99') "Salary"
FROM employees;
```

問合せの結果が表示されます。

```
Name                                       Salary
------------------------------------------ -----------
Steven King                                 $24,000.00
Neena Kochhar                               $17,000.00
Lex De Haan                                 $17,000.00
...
107 rows selected
```

例 2-34 は、TO_NUMBER ファンクションを使用して、その後の計算に使用できるように文字を数字に変換する方法を示しています。TO_NUMBER 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 2-34 文字を数字に変換する TO_NUMBER の使用

```
SELECT first_name || ' ' || last_name "Name",
  TO_NUMBER('300') + salary  "Proposed Salary"
FROM employees
WHERE SUBSTR(job_id, 4, 5) = 'CLERK';
```

問合せの結果が表示されます。

```
Name                                       Proposed Salary
------------------------------------------ ---------------
Alexander Khoo                                        3400
Shelli Baida                                          3200
Sigal Tobias                                          3100
...
45 rows selected
```

選択した従業員のサブセットに対して提案されたすべての salary 値が結果セットにリストされます。

TO_DATE ファンクションを使用して、指定された書式モードを含む文字データを日付データに変換できます。例 2-35 で使用する書式モデルは、'Month dd, YYYY, HH:MI A.M. '、'DD-MON-RR'、'FF-Mon-YY HH24:MI:SI' などです。

例 2-35 文字データを日付に変換する TO_DATE の使用

```
SELECT TO_DATE('January 5, 2007, 8:43 A.M.',
  'Month dd, YYYY, HH:MI A.M.') "Date"
FROM DUAL;
```

問合せの結果が表示されます。

```
Date
---------
05-JAN-07
```

指定した書式文字列によって解釈され、文字データが DATE 型に変換されます。

例 2-36 は、'DD-Mon-RR HH24:MI:SS.FF' などの書式モデルを含む 'TO_TIMESTAMP' メソッドを使用する方法を示しています。TO_DATE の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

例 2-36 文字データをタイムスタンプに変換する TO_TIMESTAMP の使用

```
SELECT TO_TIMESTAMP('May 5, 2007, 8:43 A.M.',
  'Month dd, YYYY, HH:MI A.M.') "Timestamp"
FROM DUAL;
```

問合せの結果が表示されます。

```
Timestamp
-------------------------------------------
05-MAY-07 08.43.00.000000000 AM
```

指定した書式文字列によって解釈され、文字データが TIMESTAMP 型に変換されます。

参照 :

- データ型交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。

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または表またはビュー全体で操作されます。本来、これらのファンクションは、セットに対する統計結果を提供し、平均（AVG）、集計（COUNT）、最大（MAX）、最小（MIN）、標準偏差（STDEV）、合計（SUM）などが含まれます。

GROUP BY 句と組み合せて使用すると、集計ファンクションは特に有効です。この句を使用することで、問合せでは各グループに明確な結果を持つ 1 つ以上の列でグループ化されたリストが戻されます。

指定した条件に一致する集計値を含む行のみが問合せで戻されるよう指定する **HAVING** 句も使用できます。

例 2-37 は、COUNT ファンクションおよび GROUP BY 句を使用して、マネージャにレポートする人数を決定する方法を示しています。レコード全体の数を示すためにワイルドカード * が使用されています。

例 2-37 式を満たす行数のカウント

```
SELECT manager_id "Manager",
  COUNT(*) "Number of Reports"
FROM employees
GROUP BY manager_id;
```

問合せの結果が表示されます。

```
  Manager Number of Reports
--------- -----------------
                          1
      100                14
      123                 8
...
19 rows selected
```

結果として各マネージャにレポートする人数が表示されます。ここで、誰にもレポートしない人がいることに注意してください。データを調査すると、Steven King にはスーパーバイザがいないことがわかります。

例 2-38 は、COUNT ファンクションを DISTINCT オプションと一緒に使用して、データ・セット内にある個別値の数を決定する方法を示しています。この例では、従業員を抱える部門の数を数えます。

例 2-38 セット内の個別値の数のカウント

```
SELECT COUNT(DISTINCT department_id) "Number of Departments"
FROM employees;
```

問合せの結果が表示されます。

```
Number of Departments
---------------------
11
```

11 部門に従業員が所属しているという結果が表示されます。departments 表を確認すると、27 部門がリストされているのがわかります。

MIN、MAX、MEDIAN、AVG などの基本的な統計ファンクションを使用して、セット間の給与の範囲を決定できます。例 2-39 では、job_id でグループ化した給与を調査でき、同様の問合せを使用して、部門、事業所などの間での給与調査も可能です。

例 2-39 統計情報の決定

```
SELECT job_id "Job", 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;
```

問合せの結果が表示されます。

```
Job                 #    Minimum    Average     Median    Maximum    Std Dev
---------- ---------- ---------- ---------- ---------- ---------- ----------
AC_ACCOUNT          1       8300       8300       8300       8300          0
AC_MGR              1      12000      12000      12000      12000          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      12000      12000      12000      12000          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
...
19 rows selected
```

結果として 19 の異なるジョブに対する統計が表示されます。

HAVING 句を使用すると、結果セットを目的の値のみに限定できます。例 2-40 では、給与合計が年間で $1,000,000 を超える部門に対する給与予算を表示します。

例 2-40 HAVING 句を使用する結果の制限

```
SELECT Department_id "Department", SUM(salary*12) "All Salaries"
FROM employees
HAVING SUM(salary * 12) >= 1000000
GROUP BY department_id;
```

問合せの結果が表示されます。

```
Department All Salaries
---------- ------------
        50      1876800
        80      3654000
```

結果として給与予算が $1,000,000 を超過する 2 部門が表示されます。

RANK ファンクションを使用すると相対的な順序付けランクを決定でき、PERCENT_RANK ファンクションを使用するとその百分率を決定できます。例 2-41 では、job_id に CLERK 指定のあるすべての従業員のサブセット内で給与 $3,000 に対するこれらの値を決定できます。

WITHIN GROUP ファンクションを使用するとグループを調査することもできます。

例 2-41 RANK および PERCENT_RANK の決定

```
SELECT RANK(3000) WITHIN GROUP (ORDER BY salary DESC) "Rank",
  ROUND(100 * (PERCENT_RANK(3000)
  WITHIN GROUP (ORDER BY salary DESC)), 0) "Percentile"
FROM employees
WHERE job_id LIKE '%CLERK';
```

問合せの結果が表示されます。

```
      Rank Percentile
---------- ----------
        20         42
```

CLERK 指定のあるすべての従業員の間で、給与 $3,000 は 20 番目に高く、百分率では 42% であることが結果に表示されます。

DENSE_RANK ファンクションは、RANK ファンクションとほぼ同様の働きをしますが、同じ値が同一ランクを受け取り、ランキングに差分が発生しません。例 2-42 では、job_id に CLERK 指定のあるすべての従業員のサブセット内で $3,000 の DENSE_RANK を決定します。

例 2-42 DENSE_RANK の決定

```
SELECT DENSE_RANK(3000) WITHIN GROUP (ORDER BY salary DESC) "Rank"
FROM employees
WHERE job_id LIKE '%CLERK';
```

問合せの結果が表示されます。

```
      Rank
----------
        12
```

DESNE_RANK ファンクションを使用した結果、給与 $3,000 が 12 番目であることが表示されます。RANK ファンクションを使用した前の例で取得した 20 番目のランクと対比します。

参照 :

- 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。

NULL 値ファンクションの使用

NULL 値を処理できるように、Oracle Database には 2 つのファンクションが用意されています。NULL があった場合、NVL ファンクションは指定した値に置き換え、NVL2 ファンクションは考えられる 2 つの評価可能な式（コンポーネント変数のいずれも NULL でない場合と、少なくとも 1 つの変数が NULL である場合）を指定します。

例 2-43 では、NVL および NVL2 ファンクションを使用して、$300,000 の売上に関係している従業員の場合の各従業員に対する全体の年間報酬を決定します。コミッション率は売上量の乗算であり、基本給与の乗算ではないことに注意してください。また、WHERE 句はマネージャに対して結果セットを制限することに注意してください。

例 2-43 NVL および NVL2 ファンクションの使用

```
SELECT first_name || ' ' || last_name "Name",
  NVL((commission_pct * 100), 0) "Comm Rate",
  NVL2(commission_pct,
    ROUND(salary * 12 + commission_pct * 300000, 2),
    salary * 12) "With $300K Sales"
FROM employees
WHERE job_id LIKE '%_M%' AND department_id = 80;
```

問合せの結果が表示されます。

```
Name                              Comm Rate            With $300K Sales
--------------------------------- -------------------- -----------------
John Russell                      40                   288000
Karen Partners                    30                   252000
Alberto Errazuriz                 30                   234000
Gerald Cambrault                  30                   222000
Eleni Zlotkey                     20                   186000

5 rows selected
```

Comm Rate 列で、NVL ファンクションにより NULL 値が 0 に置き換えられます。With $300K Sales 列には、NVL2 ファンクションにより、COMMISSION_PCT 値の値によって 2 つの異なる式から値が生成されます。

条件付きファンクションの使用

Oracle Database には、複数の条件値に基づいた値を戻すことができる 2 つのファンクションが用意されています。

CASE ファンクションは、値、式または検索条件を対比し、一致を検出した場合に結果を戻すという点で、ネストされた IF ... THEN ... ELSE 文と同等です。

例 2-44 では、CASE ファンクションを使用して、勤続年数に応じて支払われる予想給与増加を表示します。

例 2-44 CASE ファンクションの使用

```
SELECT first_name || ' ' || last_name "Name",
hire_date "Date Started", salary "Current Pay",
CASE
  WHEN hire_date < TO_DATE('01-Jan-90') THEN TRUNC(salary*1.15, 0)
  WHEN hire_date < TO_DATE('01-Jan-95') THEN TRUNC(salary*1.10, 0)
  WHEN hire_date < TO_DATE('01-Jan-00') THEN TRUNC(salary*1.05, 0)
  ELSE salary END "Proposed Salary"
FROM employees;
```

問合せの結果が表示されます。

```
Name                      Date Started      Current Pay      Proposed Salary
------------------------  ---------------  ---------------  ------------------
Steven King               17-JUN-87         24000            27600
Neena Kochhar             21-SEP-89         17000            19550
Lex De Haen               13-JAN-93         17000            18700
...
107 rows selected
```

Proposed Salary 列の値が、Date Started 値に基づいて調整されていることが結果に表示されます。

DECODE ファンクションは、値または式を検索値と対比し、一致を検出した場合に結果を戻します。一致が検出されない場合、DECODE ファンクションはデフォルト値、またはデフォルト値が指定されていない場合は NULL に戻します。

例 2-45 では、DECODE ファンクションを使用して job_id 値に基づいて考えられる給与増加を割り当てます。

例 2-45 DECODE ファンクションの使用

```
SELECT first_name || ' ' || last_name "Name",
job_id "Job", salary "Current Pay",
DECODE(job_id,
 'PU_CLERK', salary * 1.10,
 'SH_CLERK', salary * 1.15,
 'ST_CLERK', salary * 1.20,
             salary) "Proposed Salary"
FROM employees;
```

問合せの結果が表示されます。

```
Name                      Job              Current Pay      Proposed Salary
------------------------  ---------------  ---------------  ------------------
...
Alexander Khoo            PU-CLERK         3100             3410
...
Julia Nayer               ST_CLERK         3200             3840
...
Winston Taylor            SH_CLERK         3200             3680
...
107 rows selected
```

Proposed Salary 列の値が、job_id 値に基づいて調整されていることが結果に表示されます。

参照 :

- CASE 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。
- 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。

データの追加、変更および削除

データベースにおける追加、変更および削除操作は、一般的にデータ操作言語（DML）文と呼ばれます。

- INSERT 文は、既存の表に新規の行を追加します。
- UPDATE 文は、既存の表の行のセットの値を変更します。
- DELETE 文は、表から既存の行を削除します。

これらの文は表内のデータを変更するため、トランザクション管理を使用してすべての依存する DML 文をグループ化することをお薦めします。

情報の挿入

INSERT 文を使用して表にデータの行を追加する場合、挿入されたデータは表の各列のデータ型およびサイズに対して有効である必要があります。

INSERT コマンドの一般構文は次のとおりです。値リストは表の列と同じ順序である必要があることに注意してください。

```
INSERT INTO table_name VALUES
(list_of_values_for_new_row);
```

例 2-46 では、INSERT ファンクションを使用して employees 表に新規の行を追加します。

例 2-46 すべての情報が使用できる場合での INSERT 文の使用

```
INSERT INTO employees VALUES
  (10, 'George', 'Gordon', 'GGORDON', '650.506.2222',
   '01-JAN-07', 'SA_REP', 9000, .1, 148, 80);
```

問合せの結果が表示されます。

```
1 row created.
```

新規の行が employees 表に正常に追加されたことが結果に表示されます。

例 2-47 は、新しいレコードがデータベースに追加される時点ですべての情報が使用不可の場合、特定の既知の表の列に値を挿入し、残っている列を NULL に設定する方法を示しています。

NULL に設定された列が NOT NULL 制約に指定される場合、エラーが生成されます。

例 2-47 一部の情報が使用できない場合での INSERT 文の使用

```
INSERT INTO employees VALUES
  (20, 'John', 'Keats', 'JKEATS', '650.506.3333',
   '01-JAN-07', 'SA_REP', NULL, .1, 148, 80);
```

問合せの結果が表示されます。

```
1 row created.
```

新規の行が employees 表に正常に追加されたことが結果に表示されます。

参照：

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

情報の更新

UPDATE 文を使用して表の行を更新する場合、新規のデータは表の各列のデータ型およびサイズに対して有効である必要があります。

UPDATE コマンドの一般構文は次のとおりです。変更された列は識別される必要があり、一致条件に一致している必要があります。

```
UPDATE table_name
SET column_name = value;
WHERE condition;
```

データが欠落している行の情報を更新するには、欠落しているデータの列を指定する必要があります。例 2-48 では以前挿入したレコードに対して salary 列を更新します。

例 2-48 欠落した情報を追加する UPDATE 文の使用

```
UPDATE employees
SET salary = 8500
WHERE last_name = 'Keats';
```

問合せの結果が表示されます。

```
1 row updated.
```

結果として一致行が更新されたことが表示されます。

例 2-49 は、UPDATE 文を使用して複数行を更新する方法を示しています。

例 2-49 データを変更する UPDATE 文の使用

```
UPDATE employees
SET commission_pct=commission_pct + 0.05
WHERE department_id = 80;
```

問合せの結果が表示されます。

```
36 rows updated.
```

結果として指定した行が更新されたことが表示されます。

参照 :

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

情報の削除

DELETE 文を使用して、表の特定の行を削除できます。表のすべての行を削除する場合は、空の表を使用します。データベースから表全体を削除する場合は、DROP TABLE 文を使用します。

行を誤って削除した場合は、ROLLBACK 文を使用して行をリストアできます。

例 2-50 は、DELETE 文を使用して、以前追加したデータを削除する方法を示しています。

WHERE 句を使用しないとすべての行が削除されるので注意してください。

例 2-50 DELETE 文の使用

```
DELETE FROM employees
WHERE hire_date = '1-Jan-2007';
```

問合せの結果が表示されます。

```
2 rows deleted.
```

結果として指定した行が削除されたことが表示されます。

参照 :

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

トランザクションの制御

ビジネス・プロセスをモデル化する多くのアプリケーションでは、複数の異なる操作が同時に実行される必要があります。たとえば、マネージャが退職する場合、job_history 表に行が挿入され、退社日が表示され、このマネージャにレポートする全従業員は employees 表内で再度割り当てられる必要があります。この一連の操作は、単一の単位または**トランザクション**として扱われる必要があります。

次のようなトランザクションを制御する文は、DML 文による変更を管理し、トランザクションにグループ化します。

- **COMMIT** 文は現在のトランザクションを終了し、トランザクション内のすべての変更を永続的にします。COMMIT 文は、トランザクション内のすべてのセーブポイントを消去し、トランザクションのロックを解除します。
- **ROLLBACK** 文は、現在のトランザクションで実行された作業を戻します。つまり、最後の COMMIT または ROLLBACK からのすべてのデータ変更が破棄されます。データの状態は、要求された変更前の状態にロールバックされます。
- **SAVEPOINT** 文は、後でロールバックする際のトランザクションのポイントを識別します。

COMMIT または ROLLBACK 文のいずれかを使用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。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、Oracle Database 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。

トランザクションの変更のコミット

明示的な COMMIT 文はトランザクションを終了し、データベース内のすべての変更を永続的にします。トランザクションをコミットするまでは、データベースに対するすべての変更を表示できますが、これらの変更は完了していないか、またはデータベース・インスタンスの他のユーザーに対して参照可能ではありません。トランザクションをコミットすると、すべての変更は他のユーザーに対して参照可能になり、他のユーザーの文はトランザクション終了後に実行されます。

明示的な COMMIT または ROLLBACK 文より前の変更は取り消すことが可能です。

例 2-51 は、regions 表に新規の行を追加した後に、COMMIT 文を使用する方法を示しています。

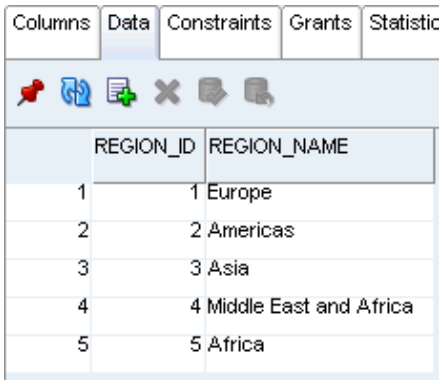
例 2-51 COMMIT 文の使用

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

問合せの結果および COMMIT 文が表示されます。

```
Commit complete.
```

手動で regions 表の内容を確認する際に、新規の行が含まれていることを確認します。



参照 :

- 『Oracle Database SQL 言語リファレンス』

トランザクションの変更のロールバック

ROLLBACK 文は、最後の COMMIT 文以降のトランザクションのすべてをロールバックします。前の COMMIT 文がプログラムに残っていない場合は、すべての操作がロールバックされます。

例 2-52 および例 2-53 は、ROLLBACK 文を使用して regions 表への変更を取り消す方法を示しています。

例 2-52 REGIONS 表の変更

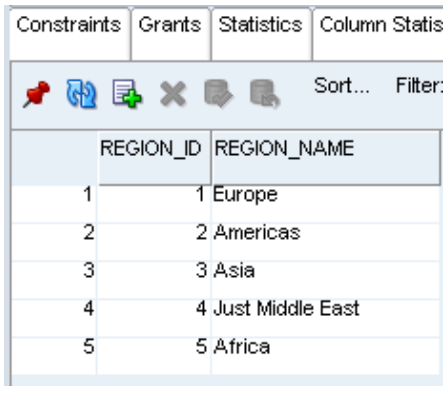
```
UPDATE regions
SET region_name = 'Just Middle East'
WHERE region_name = 'Middle East and Africa';
```

問合せの結果が表示されます。

```
1 row updated.
```

手動で regions 表の内容を確認します。

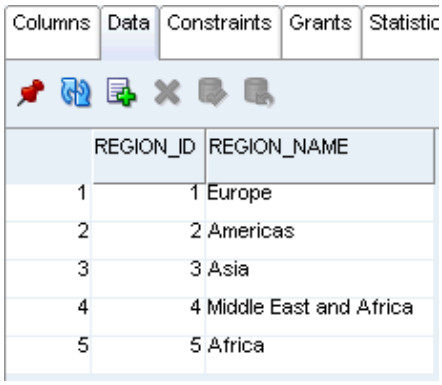
region_name 値が更新されていることを確認します。



例 2-53 REGIONS 表への変更での ROLLBACK の実行

```
ROLLBACK;
```

regions 表の内容を「**Refresh**」アイコンをクリックして手動でチェックします。region_name 値が元の値に戻っていることを確認します。



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

参照：

- 『Oracle Database SQL 言語リファレンス』

セーブポイントの設定

SAVEPOINT 文を使用して後でロールバックできるトランザクションのポイントを識別できます。アプリケーションに必要なだけセーブポイントを使用できるため、アプリケーションにより多くのトランザクション制御を実装できます。

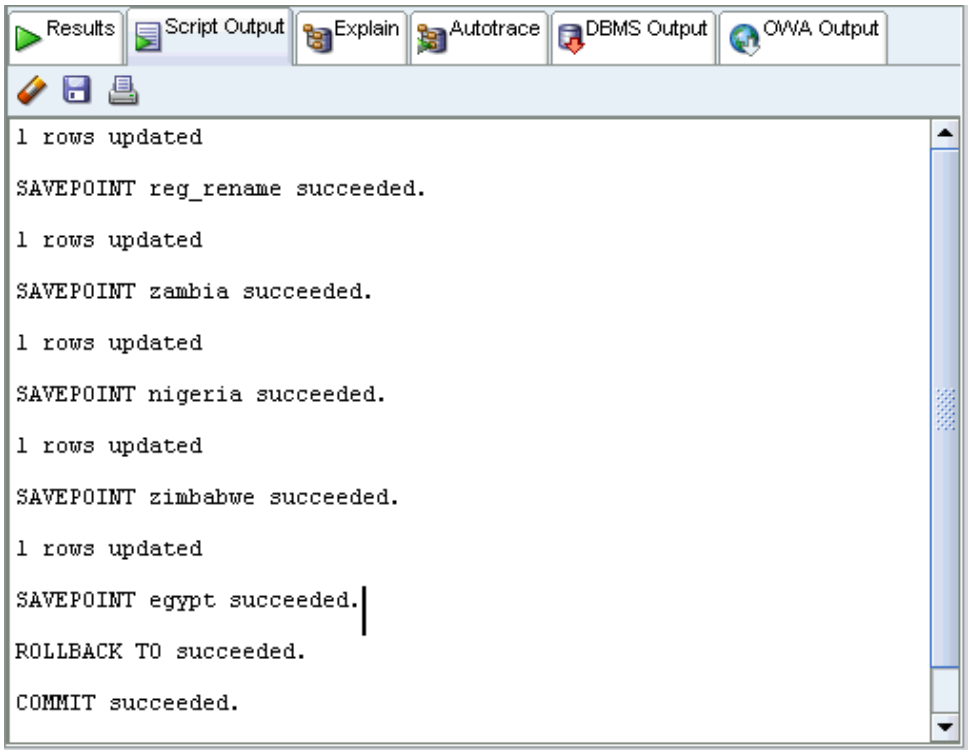
例 2-54 では、regions 表に新規の行を追加した後に、ROLLBACK 文を使用します。

例 2-54 SAVEPOINT 文の使用

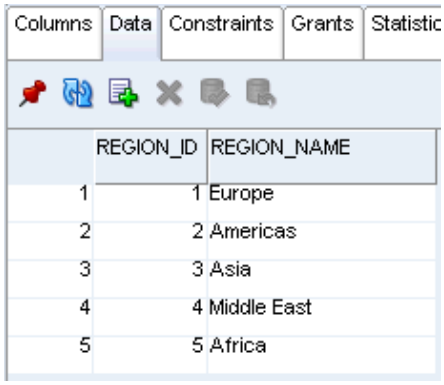
```
UPDATE regions
  SET region_name = 'Middle East'
  WHERE region_name = 'Middle East and Africa';
SAVEPOINT reg_rename;

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;

ROLLBACK TO SAVEPOINT nigeria;

COMMIT;
```

各 UPDATE および SAVEPOINT 文の結果は次のとおりです。

```
1 row updated.
Savepoint created.
```

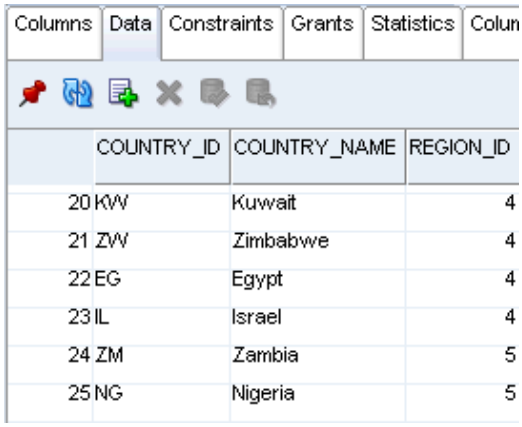


regions 表の内容を手動でチェックします。「**Refresh**」アイコンをクリックする必要がある場合があります。更新された region_name 値を確認します。



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

次に、countries 表の内容を手動でチェックします。「**Refresh**」アイコンをクリックする必要がある場合があります。Zambia および Nigeria の更新された region_name 値を含んでいること、Zimbabwe および Egypt の値は更新されていないことを確認します。



	COUNTRY_ID	COUNTRY_NAME	REGION_ID
20	KW	Kuwait	4
21	ZW	Zimbabwe	4
22	EG	Egypt	4
23	IL	Israel	4
24	ZM	Zambia	5
25	NG	Nigeria	5

データの変更がセーブポイント nigeria に対する ROLLBACK 文によって戻されているかどうか確認できます。

参照：

- 『Oracle Database SQL 言語リファレンス』

3

データベース・オブジェクトの作成および使用

この章では、「データの問合せおよび操作」で説明されているデータベース・オブジェクトのタイプを作成および使用します。

CREATE TABLE、ALTER TABLE、DROP TABLE などの文は、暗黙的なコミットを使用するため、ロールバックできないことに注意してください。

この章の内容は次のとおりです。

データ型の使用

データ型により、データベースでこれらの値を使用できるように一連のプロパティが値と関連付けられます。データ型に応じて、Oracle Database のデータベースにおいて実行できる情報の操作の種類が異なります。たとえば、数値データ型を使用した場合、数値の合計は計算できますが文字は使用できません。

Oracle Database では、最も一般的な VARCHAR2（長さ）、NUMBER（精度、スケール）、DATE、CHAR（長さ）、CLOB、TIMESTAMP およびその他を含む様々なデータ型をサポートしています。表を作成するときに、各列のデータ型を指定する必要があり、（必要に応じて）列に格納できる最も長い値を示します。

ここで使用する一部のデータ型のおよびそのプロパティには、次の機能があります。

- VARCHAR2 は可変長の文字列を格納し、文字データの格納には最も効率のよいオプションです。VARCHAR2 列を表に作成するときには、列に格納できる最大文字数を 1 から 4000 の間で指定します。employees 表の first_name 列には VARCHAR（20）データ型があり、LAST_NAME 列には VARCHAR2（25）データ型があります。
- VARCHAR2 データ型のオプションの NVARCHAR2 では、Unicode の可変長の文字列が格納されます。
- CHAR データ型には、1 から 2000 の間で指定した固定長の文字列が格納され、値の空白埋めが使用されます。

  CHAR2 データ型のオプションの NCHAR では、Unicode の固定長の文字列が格納されます。
- CLOB データ型は、シングルバイト・キャラクタまたはマルチバイト・キャラクタが含まれるキャラクタ・ラージ・オブジェクトのデータ型です。CLOB の最大サイズは（4GB - 1）x（データベース・ブロック・サイズ）です。
- NUMBER データ型はゼロ、整数、および 10 進精度で固定小数点または浮動小数点を使用して、1.0×10^{-130} と 1.0×10^{126} の間の絶対値を持つ正と負の固定数としての実数を格納します。NUMBER データ型は、異なるオペレーティング・システム間で移植可能であり、数値データを格納する必要がある場合にはこのデータ型を使用することをお薦めします。

  精度オプションを使用して数の最大桁数を設定し、スケール・オプションを使用して小数点以下の桁数をいくつにするかを定義します。employees 表の salary 列は NUMBER（8,2）として定義され、基本通貨単位（ドル、ポンド、マルクなど）に対し 6 桁、補助通貨単位（セント、ペニー、ペニッヒなど）として 2 桁が与えられています。
- Oracle Database では、浮動小数点用に基本的な NUMBER データ型の拡張として数値 BINARY_FLOAT および BINARY_DOUBLE データ型が用意されています。BINARY_FLOAT（32 ビット IEEE 754 規格）の範囲は $1.17549 \times e^{-38F}$ と $3.40282 \times e^{38F}$ の間の絶対値で、BINARY_DOUBLE（64 ビット IEEE 754 規格）の範囲は $2.22507485850720 \times e^{-308}$ と $1.79769313486231 \times e^{308}$ の間の絶対数です。両方とも 2 進精度を使用し、これにより高速な算術計算を可能とし、通常は記憶域要件が減少します。
- DATE データ型には、ある時点の値（日付および時間）が格納されます。これには年（世紀を含む）、月、日、時、分および秒が格納されます。有効な日付の範囲は BC 4712 年 1 月 1 日から AD 9999 年 12 月 31 日までです。Oracle Database では日付と時間の値を表示するための様々な形式がサポートされています。employees 表の hire_date 列は DATE として定義されています。
- TIMESTAMP データ型には小数秒までの正確な値が格納されるため、イベントの順序を追跡する必要があるアプリケーションにおいて役立ちます。
- TIMESTAMP WITH TIME ZONE データ型にはタイムゾーン情報が格納されるため、複数の地域にまたがって調整する必要がある日付情報を記録できます。

参照：

- 『Oracle Database SQL 言語リファレンス』

表の作成および使用

表は、Oracle データベースにおけるデータ記憶領域の基本単位であり、ユーザーがアクセスできるすべてのデータを保持します。表は、表のフィールドを表す縦方向の列および表の各レコードの値を表す横方向の行から構成される 2 次元オブジェクトです。

この項では、必要な表およびその他のスキーマ・オブジェクトを作成し、既存の hr スキーマの従業員のパフォーマンス評価プロセスを実装します。

参照 :

- 「表の参照およびデータの表示」(2-4 ページ)

表の作成

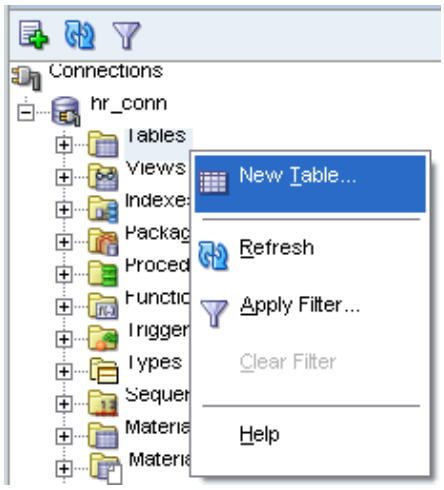
従業員評価プロセスを実装するには、performance_parts、evaluations および scores という 3 つの表を作成する必要があります。

- performance_parts 表にはパフォーマンス測定のカテゴリおよび各項目に関連する重みがリストされます。
- evaluations 表には従業員情報、評価日、役職および評価時のマネージャと部門が含まれます。従業員の役職、マネージャまたは部門の変更はいつ起こるかわからないため、この表の情報は保護する必要があります。
- scores 表には、各評価の各カテゴリに割り当てられたスコアが含まれます。

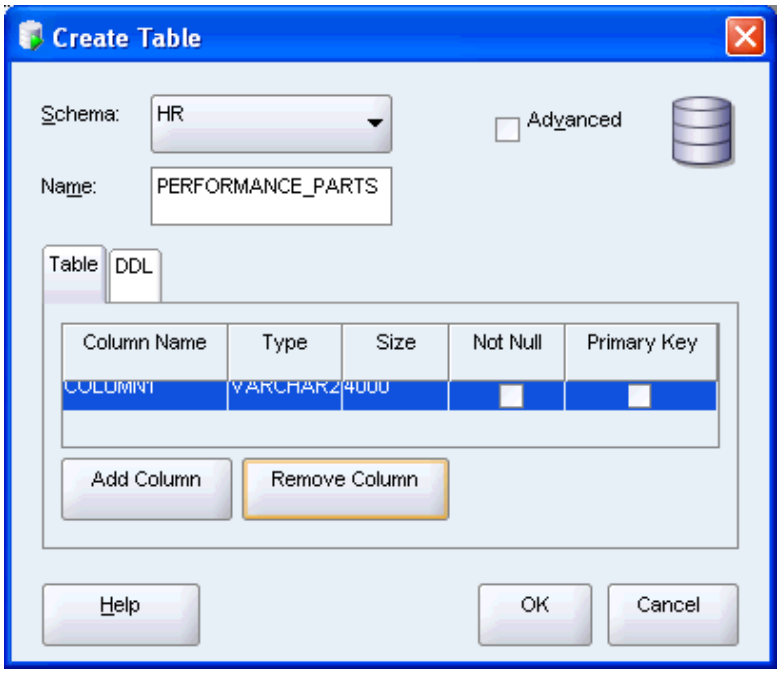
SQL Developer インタフェースを使用して、表を作成するには、次の手順を実行します。

SQL Developer のグラフィカル・インタフェースを使用して、performance_parts 表を作成します。

1. 「Connections」ナビゲーション階層で、hr_conn の横にあるプラス記号（+）をクリックしてスキーマ・オブジェクトのリストを展開します。
2. 「**Tables**」を右クリックします。
3. 「**New Table**」を選択します。

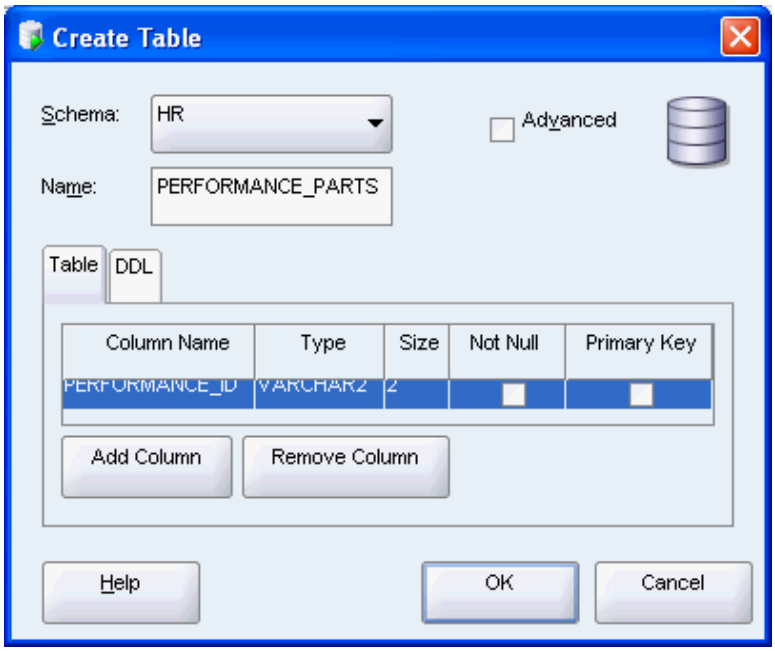


4. 「Create Table」ウィンドウで、次の情報を入力します。
   - 「**Schema**」に HR を選択します。
   - 「**Name**」に PERFORMANCE_PARTS を選択します。



5. 表に作成されているデフォルト列をクリックします。
6. 表の 1 列目の情報として次を入力します。
   - 「**Column Name**」に PERFORMANCE_ID を入力します。
   - 「**Type**」に VARCHAR2 を入力します。
   - 「**Size**」に 2 を入力します。

   「**Not Null**」と「**Primary Key**」プロパティの値を残します。「データ整合性の保証」でここに戻ります。



7. 2 列目の情報として次を入力します。

   「**Add Column**」をクリックします。

   - 「**Column Name**」に NAME を入力します。
   - 「**Type**」に VARCHAR2 を入力します。
   - 「**Size**」に 80 を入力します。

8. 3 列目の情報として次を入力します。

   「**Add Column**」をクリックします。

   - 「**Column Name**」に WEIGHT を入力します。
   - 「**Type**」に NUMBER を入力します。

9. 「**OK**」をクリックします。

   SQL Developer により新しい表 performance_parts が生成されます。

10. 「Connections」ナビゲーション階層で、「**Tables**」の横にあるプラス記号（**+**）をクリックして表のリストを展開します。

    performance_parts は hr スキーマの新しい表であり、locations と regions の間にリストされます。

これで新しい表 performance_parts が作成されました。表をクリックすると、SQL Developer ウィンドウの右側に新しい列を含む表が表示されます。「SQL」タブをクリックすると、この表を作成したスクリプトが表示されます。

例 3-1 では、「SQL Worksheet」ペインで直接情報を入力して、evaluations 表を作成します。

例 3-1 SQL スクリプトでの表の作成

```
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)
)
```

スクリプトの結果が続きます。

```
CREATE TABLE succeeded.
```

これで新しい表 evaluations が作成されました。表をクリックすると、SQL Developer ウィンドウの右側に新しい列を含む表が表示されます。「SQL」タブをクリックすると、この表を作成したスクリプトが表示されます。「Refresh」アイコンをクリックする必要があります。

例 3-2 では、「SQL Worksheet」ペインで情報を入力して、別の表 scores を作成します。

例 3-2 SCORES 表の作成

```
CREATE TABLE scores (
  evaluation_id   NUMBER(8,0),
  performance_id  VARCHAR2(2),
  score           NUMBER(1,0)
);
```

文の結果が続きます。

```
CREATE TABLE succeed.
```

これで新しい表 scores が作成されました。表をクリックすると、SQL Developer ウィンドウの右側に新しい列を含む表が表示されます。「SQL」タブをクリックすると、この表を作成したスクリプトが表示されます。「Refresh」アイコンをクリックする必要があります。

参照：

- 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。

データ整合性の保証

表のデータはアプリケーションでモデル化されたビジネス・ルールを満たす必要があります。これらのルールの多くは、どのタイプのデータ値が各列に対して有効であるかを明示的に宣言する SQL を使用する**整合性制約**で実装されます。

整合性制約を表に適用するとき、表内のすべてのデータが対応するルールに従う必要があります。このため、表にデータを挿入または変更する SQL 文がアプリケーションに含まれているときに、制約が守られていることを Oracle Database が自動的に確認します。制約に違反する行を挿入、更新または削除しようとすると、エラーが生成され、その文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しようとすると、既存の行のいずれかで新しい制約に違反している場合にエラーが生成される場合があります。

Oracle Database ではアプリケーションより迅速に、表内のすべてのデータが整合性制約に従っているかを確認するため、アプリケーション・ロジックに同様のチェック・タイプを含めるよりも、整合性制約により定義することでさらに確実にビジネス・ルールを規定できます。

参照：

- 整合性制約の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

データ整合性制約の種類の理解

整合性制約には 5 つの基本タイプがあります。

- **NOT NULL** 制約は、列に NULL ではないデータが含まれることを保証します。
- **一意**制約は、複数行で同じ列内に同じ値がないことを保証します。この種類の制約は**複合一意**制約のように、列の組合せにおいても使用できます。この制約では NULL 値は無視されます。
- **主キー**制約は 1 つの宣言で NOT NULL と UNIQUE 制約を組み合せます。これにより複数行で同じ列内または列の組合せに同じ値があることを禁止し、NULL 値を禁止します。
- **外部キー**制約では、その制約が定義される列の各値に対して一致する値が指定した他の表内および列内に存在する必要があります。
- **チェック**制約は、値が特定の条件を満たすことを保証します。チェック制約は、比較などのように制約ルールを論理式に基づいて規定する必要があるときに使用します。他の種類の制約を使用して必要なチェックを行うことができる場合は、チェック制約を使用しないことをお薦めします。

整合性制約の追加

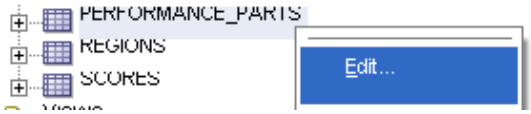
3-3 ページの「表の作成」で作成した表に異なる種類の制約を追加します。

SQL Developer インタフェースを使用して NOT NULL 制約を追加するには、次の手順を実行します。

SQL Developer のグラフィカル・インタフェースを使用して、表に NOT NULL 制約を追加します。

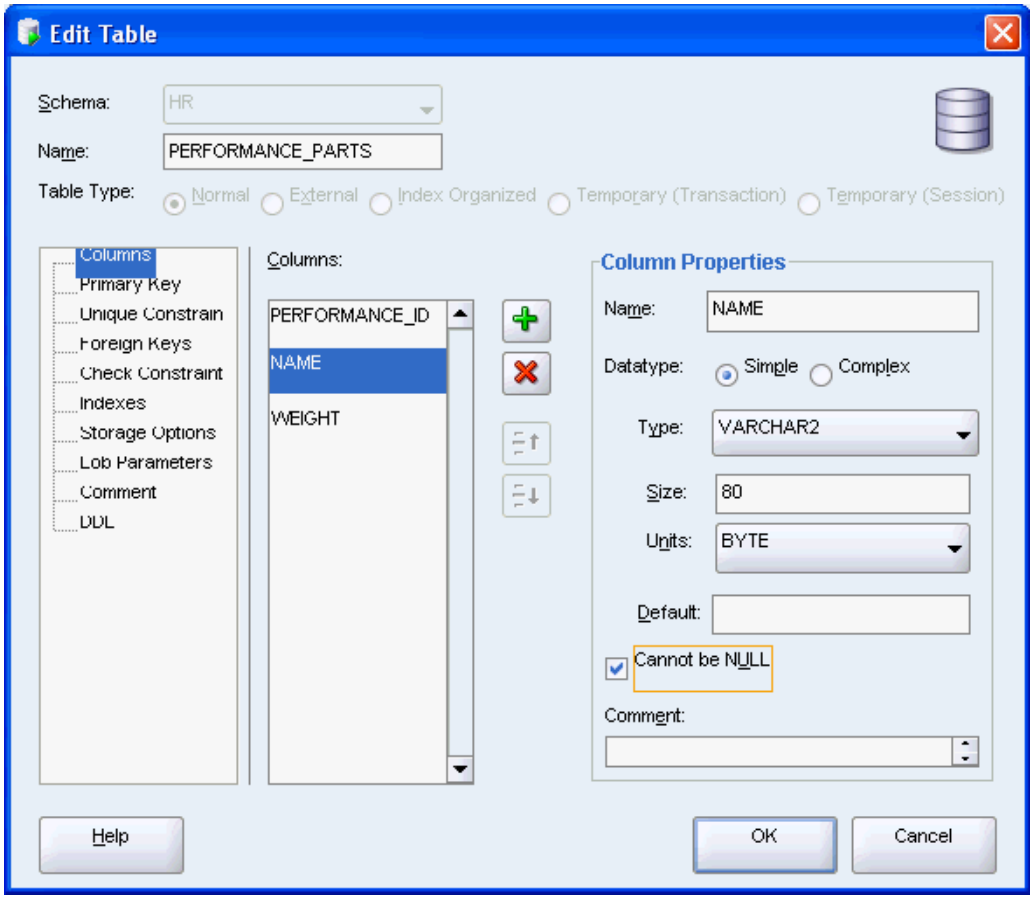
1. 「Connections」ナビゲーション階層で、表の横にあるプラス記号（+）をクリックして表のリストを展開します。
2. performance_parts 表を右クリックします。

3. 「**Edit**」を選択します。



4. 「Edit Table」ウィンドウで、次の手順に従います。
   - 「Edit Table」ウィンドウで、「**Columns**」をクリックします。
   - 列リストで NAME を選択します。
   - 「Column Properties」セクションで、「**Cannot be NULL**」にチェックを入れます。

   「**OK**」をクリックします。



5. 「Confirmation」ウィンドウで「**OK**」をクリックします。

これで performance_parts 表の name 列に対して NOT NULL 制約が作成されました。

performance_parts 表の name 列の定義は次のように変更されました。制約は自動的に有効になります。

```
"NAME" VARCHAR2(80) NOT NULL ENABLE
```

例 3-3 は、「SQL Statement」ウィンドウで必要な情報を直接入力し、performance_parts 表にさらに NOT NULL 制約を追加する方法を示しています。

例 3-3 SQL スクリプトで NOT NULL 制約を追加

```
ALTER TABLE performance_parts
MODIFY weight NOT NULL;
```

スクリプトの結果が続きます。

```
ALTER TABLE performance_parts succeeded.
```

これで performance_parts 表の weight 列に対して NOT NULL 制約が作成されました。「SQL」タブをクリックすると、Weight 列の定義が変更されたことを確認できます。「Refresh」アイコンをクリックする必要があります。

```
"WEIGHT" NUMBER NOT NULL ENABLE
```

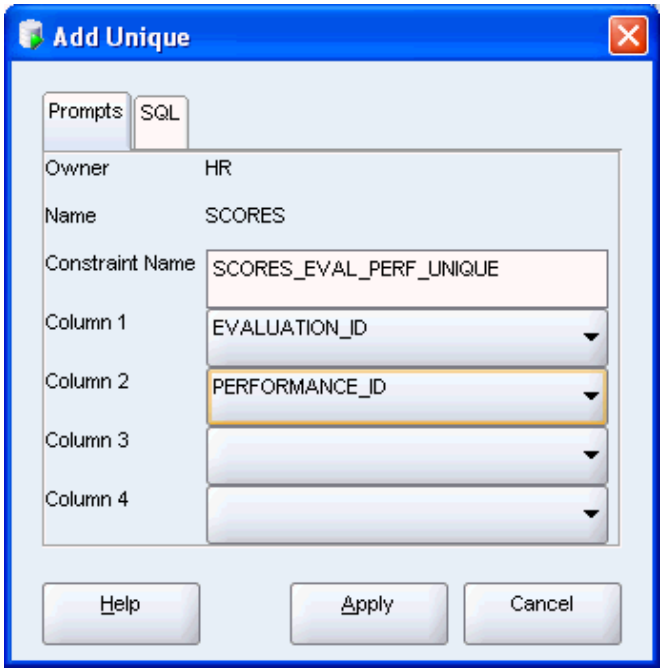
SQL Developer インタフェースを使用して、一意制約を追加するには、次の手順を実行します。

SQL Developer のグラフィカル・インタフェースを使用して、scores 表に一意制約を追加します。この作業を実行するために、NOT NULL 制約の場合と同様に「Edit Table」ウィンドウを使用することもできます。

1. 「Connections」ナビゲーション階層で、表の横にあるプラス記号（+）をクリックして表のリストを展開します。
2. scores 表を右クリックします。
3. **「Constraint」**を選択して**「Add Unique」**を選択します。

4. 「Add Unique」ウィンドウで、次の情報を入力します。
   - 「Constraint Name」を SCORES_EVAL_PERF_UNIQUE に設定します。
   - 「Column 1」を EVALUATION_ID に設定します。
   - 「Column 2」を PERFORMANCE _ID に設定します。

   「Apply」をクリックします。



5. 「Confirmation」ウィンドウで「**OK**」をクリックします。

   これで scores 表に対する一意制約が作成されました。

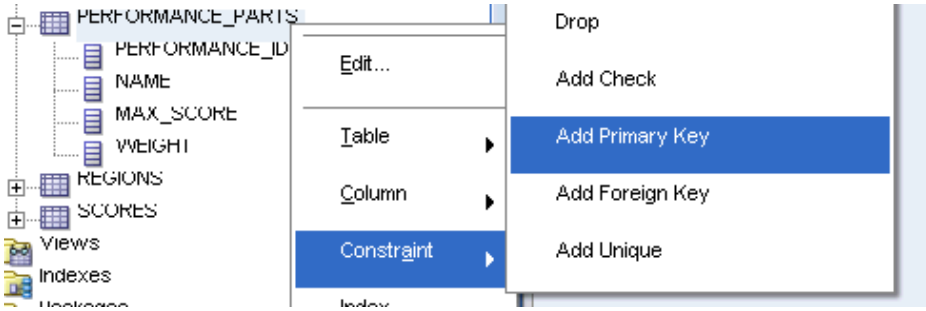
   次の SQL 文が表定義に追加されました。

```
CONSTRAINT "SCORES_EVAL_PERF_UNIQUE" UNIQUE ("EVALUATION_ID", "PERFORMANCE_ID")
```

SQL Developer インタフェースを使用して、主キー制約を追加するには、次の手順を実行します。

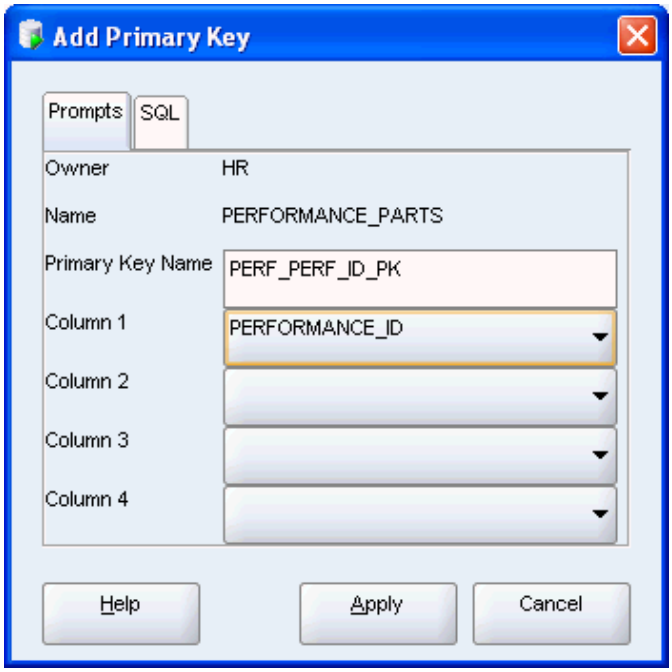
SQL Developer のグラフィカル・インタフェースを使用して、performance_parts 表に主キー制約を追加します。この作業を実行するために、NOT NULL 制約のように「Edit Table」ウィンドウを使用することもできます。

1. 「Connections」ナビゲーション階層で、表の横にあるプラス記号（+）をクリックして表のリストを展開します。
2. performance_parts 表を右クリックします。
3. 「**Constraint**」を選択して、「**Add Primary Key**」を選択します。



4. 「Add Primary Key」ウィンドウで、次の情報を入力します。
   - 「Primary Key Name」を PERF_PERF_ID_PK に設定します。
   - 「Column 1」を PERFORMANCE_ID に設定します。

   「**Apply**」をクリックします。



5. 「Confirmation」ウィンドウで「**OK**」をクリックします。

   これで performance_parts 表に対する主キー制約が作成されました。

次の SQL 文が表定義に追加されました。

```
CONSTRAINT "PERF_PERF_ID_PK" PRIMARY KEY ("PERFORMANCE_ID")
```

例 3-4 では、「SQL Statement」ウィンドウで必要な情報を直接入力して evaluations 表に主キー制約を作成します。

例 3-4 SQL スクリプトで主キー制約を追加

```
ALTER TABLE evaluations
ADD CONSTRAINT eval_eval_id_pk PRIMARY KEY (evaluation_id);
```

スクリプトの結果が続きます。

```
ALTER TABLE evaluations succeeded.
```

evaluations 表に主キー eval_eval_id_pk が作成されました。「SQL」タブをクリックすると、次の SQL 文が表定義に追加されたことを確認できます。「Refresh」アイコンをクリックする必要があります。

```
CONSTRAINT "EVAL_EVAL_ID_PK" PRIMARY KEY ("EVALUATION_ID")
```

SQL Developer インタフェースを使用して、外部キー制約を追加するには、次の手順を実行します。

SQL Developer のグラフィカル・インタフェースを使用して、scores 表に外部キー制約を追加します。この作業を実行するために、NOT NULL 制約のように「Edit Table」ウィンドウを使用することもできます。

1. 「Connections」ナビゲーション階層で、表の横にあるプラス記号（+）をクリックして表のリストを展開します。
2. scores 表を右クリックします。
3. 「**Constraint**」を選択して、「**Add Foreign Key**」を選択します。

4. 「Add Foreign Key」ウィンドウで、次の情報を入力します。
   - 「Constraint Name」を SCORES_EVAL_FK に設定します。
   - 「Column Name」を EVALUATION_ID に設定します。
   - 「References Table Name」を EVALUATIONS に設定します。
   - 「Referencing Column」を EVALUATION_ID に設定します。

   「Apply」をクリックします。

5. 「Confirmation」ウィンドウで**「OK」**をクリックします。

   これで evaluations 表の evalution_id 列に対する外部キー制約が作成されました。

6. 次のパラメータを使用し、手順 2 から 5 を繰り返すことにより、さらに外部キー制約を追加します。
   - 「Constraint Name」を SCORES_PERF_FK に設定します。
   - 「Column Name」を PERFORMANCE_ID に設定します。
   - 「References Table Name」を PERFORMANCE_PARTS に設定します。
   - 「Referencing Column」を PERFORMANCE_ID に設定します。

   「Apply」をクリックします。

次の SQL 文が表定義に追加されました。

```
CONSTRAINT "SCORES_EVAL_FK" FOREIGN KEY ("EVALUATION_ID")
 REFERENCES "HR"."EVALUATIONS" ("EVALUATION_ID") ENABLE
CONSTRAINT "SCORES_PERF_FK" FOREIGN KEY ("PERFORMANCE_ID")
 REFERENCES "HR"."PERFORMANCE_PARTS" ("PERFORMANCE_ID") ENABLE
```

例 3-5 では、「SQL Statement」ウィンドウで必要な情報を直接入力して evaluations 表に外部キー制約を作成します。

例 3-5 SQL スクリプトで外部キー制約を追加

```
ALTER TABLE evaluations
    ADD CONSTRAINT eval_emp_id_fk FOREIGN KEY (employee_id)
    REFERENCES employees(employee_id);
```

スクリプトの結果が続きます。

```
ALTER TABLE evaluations succeeded
```

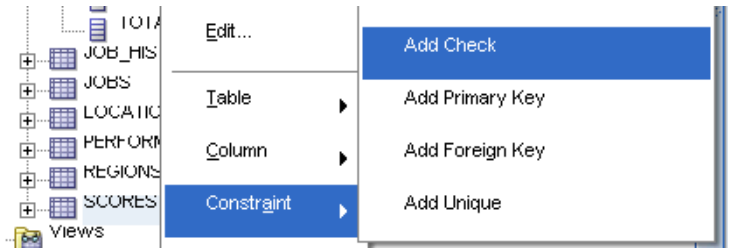
これで employees 表の employee_id 列に外部キー制約が作成されました。「SQL」タブをクリックすると、次の SQL 文が表定義に追加されたことを確認できます。「Refresh」アイコンをクリックする必要があります。

```
CONSTRAINT "EVAL_EMP_ID_FK" FOREIGN KEY ("EMPLOYEE_ID")
 REFERENCES "HR"."EMPLOYEES" ("EMPLOYEE_ID") ENABLE
```

SQL Developer インタフェースを使用して、チェック制約を追加するには、次の手順を実行します。

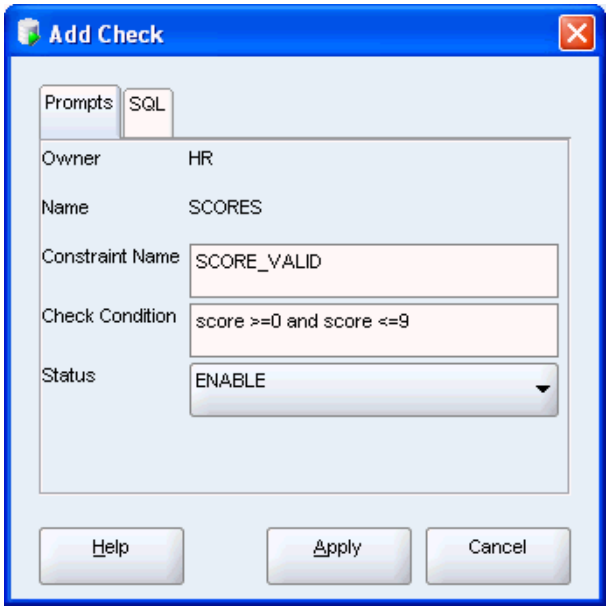
SQL Developer のグラフィカル・インタフェースを使用して、scores 表にチェック制約を追加します。この作業を実行するために、NOT NULL 制約のように「Edit Table」ウィンドウを使用することもできます。

1. 「Connections」ナビゲーション階層で、表の横にあるプラス記号（+）をクリックして表のリストを展開します。
2. scores 表を右クリックします。
3. **「Constraint」**を選択して、**「Add Check」**を選択します。



4. 「Add Check」ウィンドウで、次の情報を入力します。
   - 「Constraint Name」を SCORE_VALID に設定します。
   - 「Check Condition」を score >=0 and score <=9 に設定します。
   - 「Status」を ENABLE に設定します。

   「Apply」をクリックします。



5. 「Confirmation」ウィンドウで「**OK**」をクリックします。

   これで scores 表の score 列に対するチェック制約が作成されました。

次の SQL 文が表定義に追加されました。

```
CONSTRAINT "SCORE_VALID" CHECK (score >=0 and score <=9) ENABLE
```

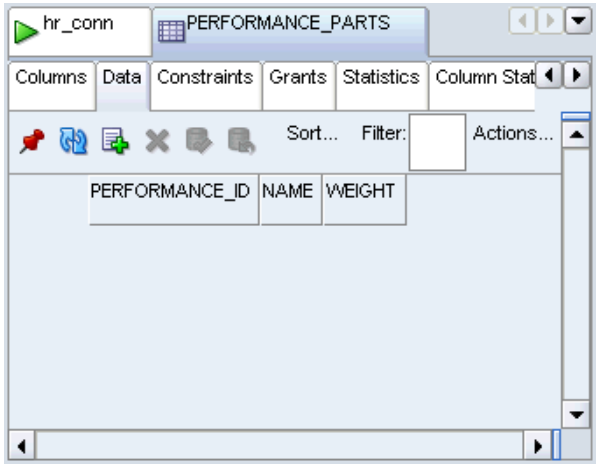
表へのデータの追加、変更および削除

SQL Developer を使用して、表へのデータの入力、編集、および既存のデータの削除ができます。次の作業では、performance_parts 表に対するこれらのプロセスを説明します。

SQL Developer インタフェースを使用して表へデータを追加するには、次の手順を実行します。

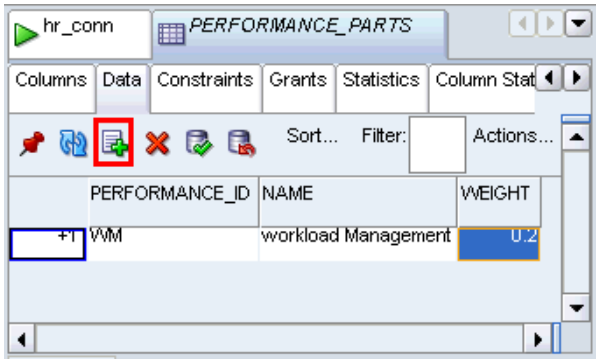
次の手順に従い、データ行を performance_parts 表に追加します。

1. 「Connections」ナビゲーション階層で、performance_parts 表をダブルクリックします。
2. 表示されている performance_parts 表の「Data」タブをクリックします。
3. 「Data」ペインで、「**New Record**」アイコンをクリックします。



4. 新しい行で、列または列間のタブを直接クリックし、次の情報を追加します。
   - 「PERFORMANCE_ID」を WM に設定します。
   - 「NAME」を Workload Management に設定します。
   - 「WEIGHT」を 0.2 に設定します。

   [Enter] キーを押します。



5. 次の情報とともに 2 番目の行を追加します。PERFORMANCE_ID を BR に、NAME を Building Relationships に、および WEIGHT を 0.2 に設定します。

   [Enter] キーを押します。

6. 次の情報とともに 3 番目の行を追加します。PERFORMANCE_ID を CF に、NAME を Customer Focus に、および WEIGHT を 0.2 に設定します。

   [Enter] キーを押します。

7. 次の情報とともに 4 番目の行を追加します。PERFORMANCE_ID を CM に、NAME を Communication に、および WEIGHT を 0.2 に設定します。

   [Enter] キーを押します。

8. 次の情報とともに 5 番目の行を追加します。PERFORMANCE_ID を TW に、NAME を Teamwork に、および WEIGHT を 0.2 に設定します。

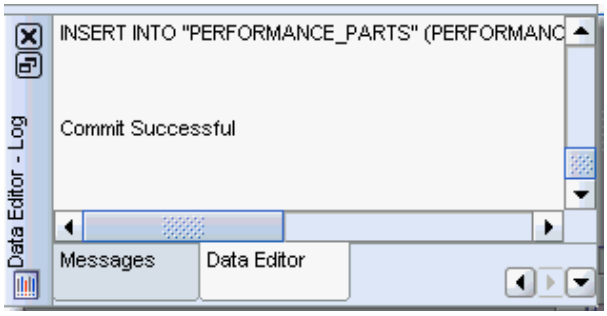
   [Enter] キーを押します。

9. 次の情報とともに 6 番目の行を追加します。PERFORMANCE_ID を RD に、NAME を Results Orientation に、および WEIGHT を 0.2 に設定します。

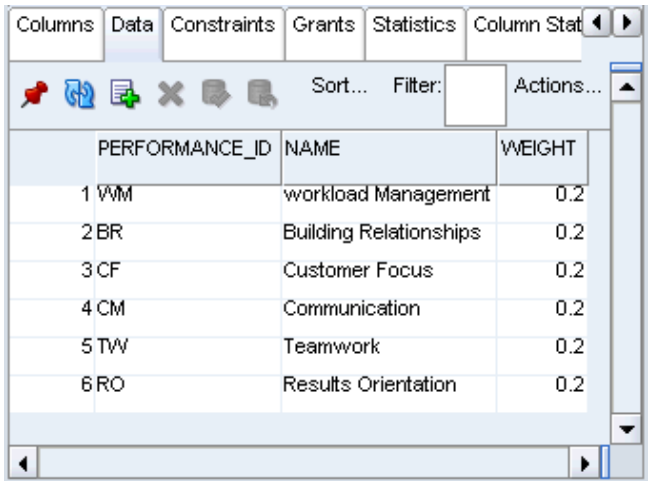
   [Enter] キーを押します。

10. 「**Commit Changes**」アイコンをクリックします。

11. 「**Data Editor Log**」ウィンドウを確認してから閉じます。



12. performance_parts 表の新しいデータを確認します。



これで performance_parts 表には 6 行が追加されました。

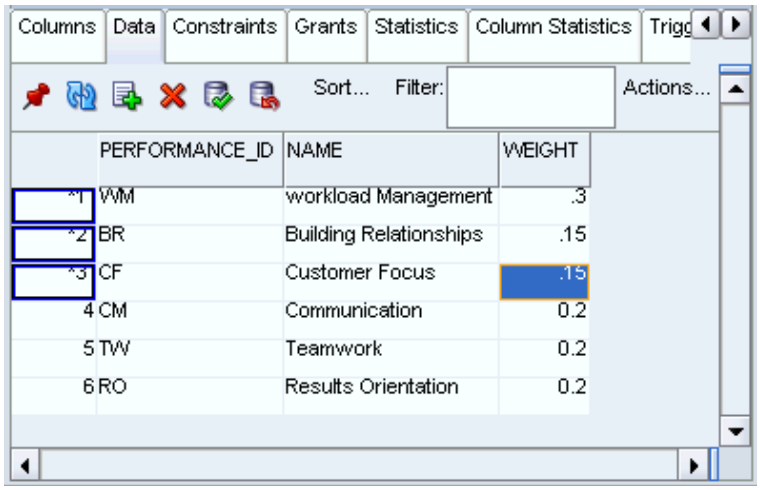
SQL Developer インタフェースを使用して、表データを変更するには、次の手順を実行します。

次の手順に従い、performance_parts 表のデータを変更します。

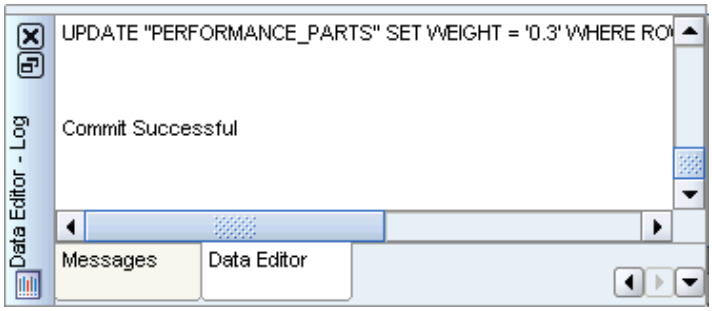
1. 「Connections」ナビゲーション階層で、performance_parts 表をダブルクリックします。
2. 表示されている performance_parts 表の「**Data**」タブをクリックします。
3. 「Data」ペインの Workload Management 行で、weight 値をクリックし、新しい値 0.3 を入力します。

   Building Relationships 行で、weight 値をクリックし、新しい値 0.15 を入力します。

   Customer Focus 行で、weight 値をクリックし、新しい値 0.15 を入力します。



4. [Enter] キーを押します。
5. 「**Commit Changes**」アイコンをクリックします。
6. 「**Data Editor Log**」ウィンドウを確認してから閉じます。

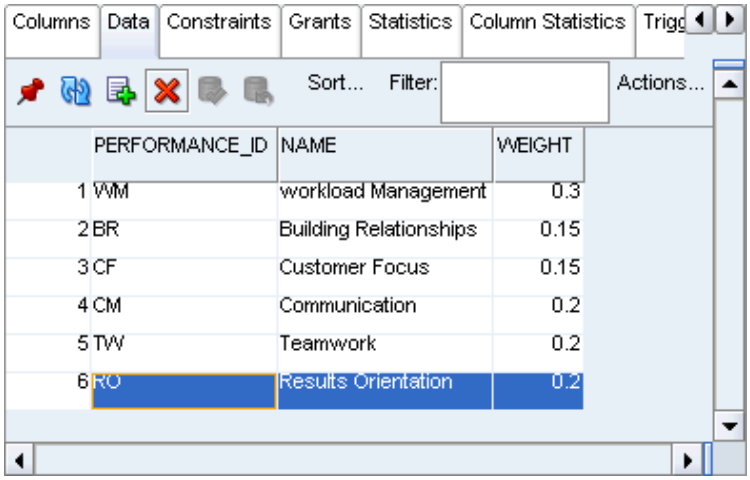


これで performance_parts 表の 3 つの行の値が変更されました。

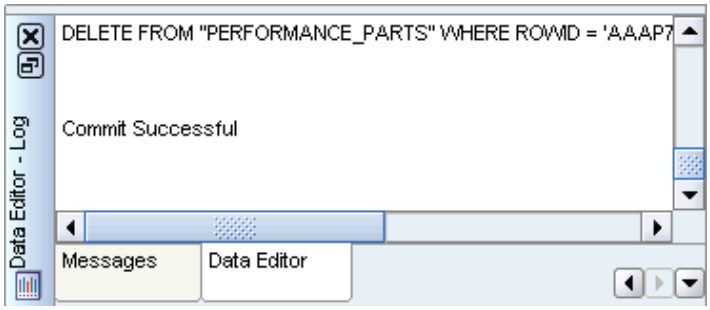
SQL Developer インタフェースを使用して、表データを削除するには、次の手順を実行します。

hr スキーマでモデル化された企業において、workload Management と Results Orientation のカテゴリに非常に多くの重複があったと仮定します。このため、performance_parts 表から Results Orientation 行を削除します。

1. 「Connections」ナビゲーション階層で、performance_parts 表をダブルクリックします。
2. 表示されている performance_parts 表の「Data」タブをクリックします。
3. 「Data」ペインで Results Orientation 行をクリックします。
4. **「Delete Selected Row(s)」**アイコンをクリックします。



5. **「Commit Changes」**アイコンをクリックします。
6. **「Data Editor Log」**ウィンドウを確認してから閉じます。



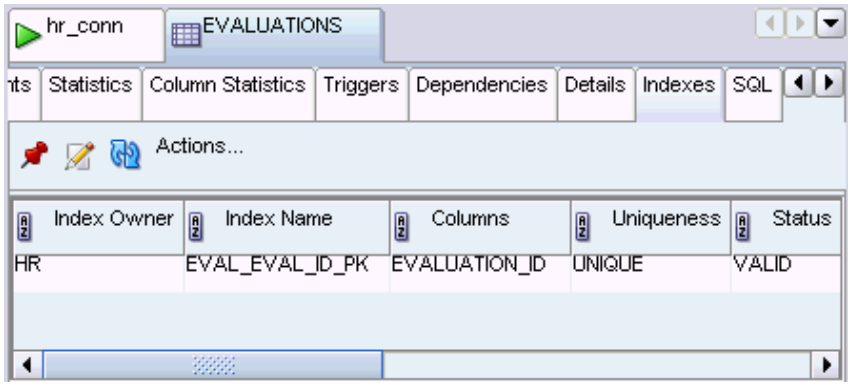
これで performance_parts 表から 1 行が削除されました。

参照：

- 『Oracle Database SQL Developer ユーザーズ・ガイド』

表の索引付け

表で主キーを定義するとき、Oracle Database では主キーが含まれる列に索引が暗黙的に作成されます。たとえば、「**Indexes**」ペインを見ることで `evaluations` 表の主キーに索引が作成されたことを確認できます。



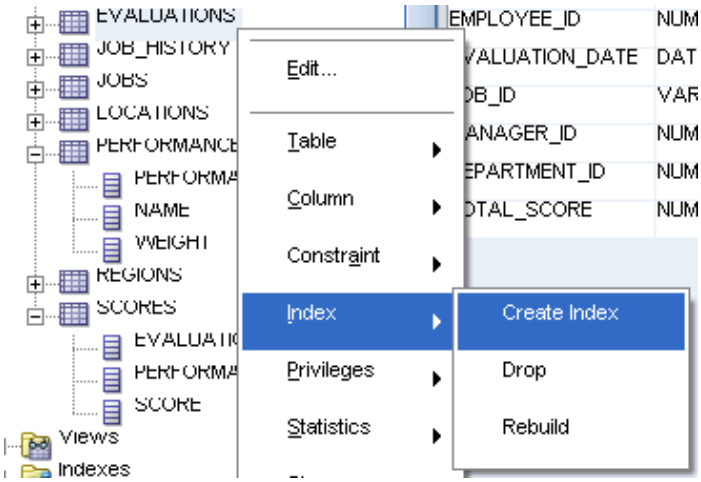
この項では、以前に作成した表に対して様々な種類の索引を追加する方法を説明します。

SQL Developer インタフェースを使用して、索引を作成するには、次の手順を実行します。

次の手順に従い、`evaluations` 表に新しい索引を作成します。

1. 「Connections」ナビゲーション階層で、`evaluations` 表を右クリックします。
2. 「**Index**」を選択して、「**Create Index**」を選択します。

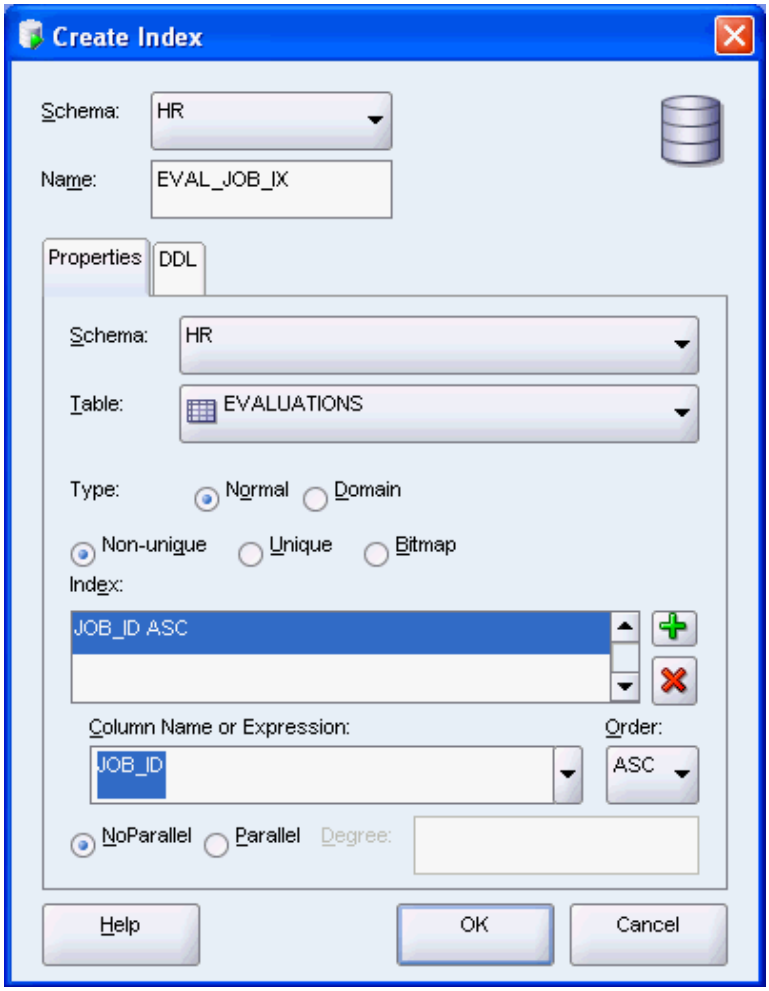
   または、「Connections」ナビゲーション階層で、「**Indexes**」を右クリックし「**New Index**」を選択することもできます。



3. 「Create Index」ウィンドウで、次のパラメータを入力します。
   - 「**Schema**」が HR に設定されていることを確認します。
   - 「**Name**」を EVAL_JOB_IX に設定します。

   プラス記号のような「Add Column Expression」アイコンをクリックします。
   - 「**Column Name or Expression**」を JOB_ID に設定します。
   - 「**Order**」を ASC に設定します。

   「**OK**」をクリックします。

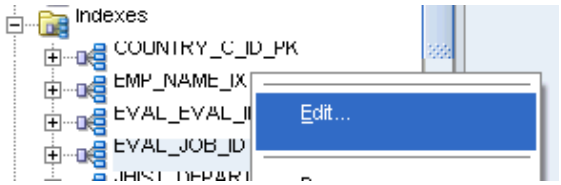


これで evaluations 表の JOB_ID 列に新しい索引 EVAL_JOB_IX が作成されました。この索引は、「Connections」ナビゲーション階層で索引のリストから見つけるか、または evaluations 表を開いて「**Indexes**」タブを参照することにより確認できます。次のスクリプトがこの索引を作成するための SQL 文です。

```
CREATE INDEX eval_job_ix ON evaluations (job_id ASC) NOPARALLEL;
```

SQL Developer インタフェースを使用して、索引を変更するには、次の手順を実行します。
次の手順に従い、EVAL_JOB_IX 索引のソート順序を逆にします。

1. 「Connections」ナビゲーション階層のプラス記号（+）で「Indexes」を拡張します。
2. EVAL_JOB_IX を右クリックして、**「Edit」**を選択します。



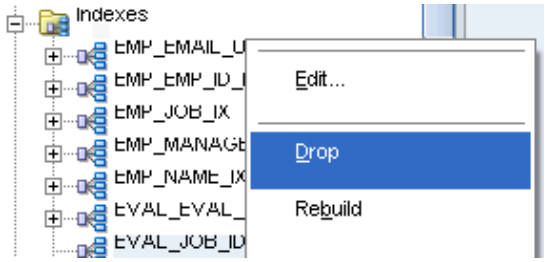
3. 「Edit Index」ウィンドウで、**「Order」**を DESC に変更します。
   「OK」をクリックします。

索引が変更されました。次のスクリプトがこの索引を変更するための SQL 文です。

```
DROP INDEX eval_job_id;
CREATE INDEX eval_job_ix ON evaluations (job_id DESC) NOPARALLEL;
```

SQL Developer インタフェースを使用して、索引を削除するには、次の手順を実行します。
次の手順に従い、EVAL_JOB_IX 索引を削除します。

1. 「Connections」ナビゲーション階層のプラス記号（+）で「Indexes」を拡張します。
2. EVAL_JOB_IX を右クリックして、**「Drop」**を選択します。



3. 「Drop」ウィンドウで、**「Apply」**をクリックします。
4. 「Confirmation」ウィンドウで**「OK」**をクリックします。

これで EVAL_JOB_IX 索引が削除されました。次のスクリプトがこの索引を削除するための SQL 文です。

```
DROP INDEX "HR"."EVAL_JOB_ID";
```

参照：

- 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。
- 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。
- 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。

表の削除

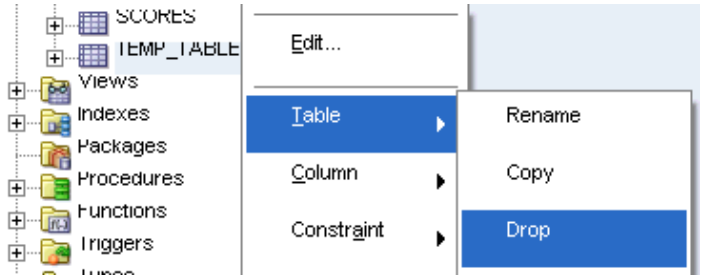
表およびその内容のすべてをスキーマから削除する必要がある場合があります。このためには、SQL 文である DROP TABLE 文を使用してする必要があります。他の概要を学ぶために作成した表を使用し、「SQL Statement」ウィンドウで、削除できる単純な表を次のスクリプトを実行して作成します。

```
CREATE TABLE temp_table(
  id NUMBER(1,0),
  name VARCHAR2(10)
);
```

SQL Developer インタフェースを使用して表を削除するには、次の手順を実行します。

次の手順に従い、hr スキーマから TEMP_TABLE を削除します。

1. 「Connections」ナビゲーション階層で、TEMP_TABLE を右クリックします。
2. **「Table」**を選択して、**「Drop」**を選択します。



3. 「Drop」ウィンドウで、**「Apply」**をクリックします。
4. 「Confirmation」ウィンドウで**「OK」**をクリックします。

これで TEMP_TABLE 表が削除されました。次のスクリプトがこの表を削除するための SQL 文です。

```
DROP TABLE "HR"."TEMP_TABLE";
```

参照：

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

ビューの使用

ビューとは、1 つ以上の表またはビューに基づく論理的な表です。ビューは、異なる複数の表に格納されている情報へ頻繁にアクセスする必要のあるビジネスにおいて特に役立ちます。

ビューの作成

ビューを作成する標準的な構文は次のとおりです。

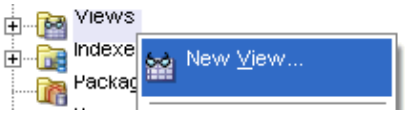
```
CREATE VIEW view_name AS query;
```

SQL Developer インタフェースを使用してビューを作成するには、次の手順を実行します。

次の手順に従い、hr スキーマから新しいビューの作成を削除します。

1. 「Connections」ナビゲーション階層で、**「Views」**を右クリックします。

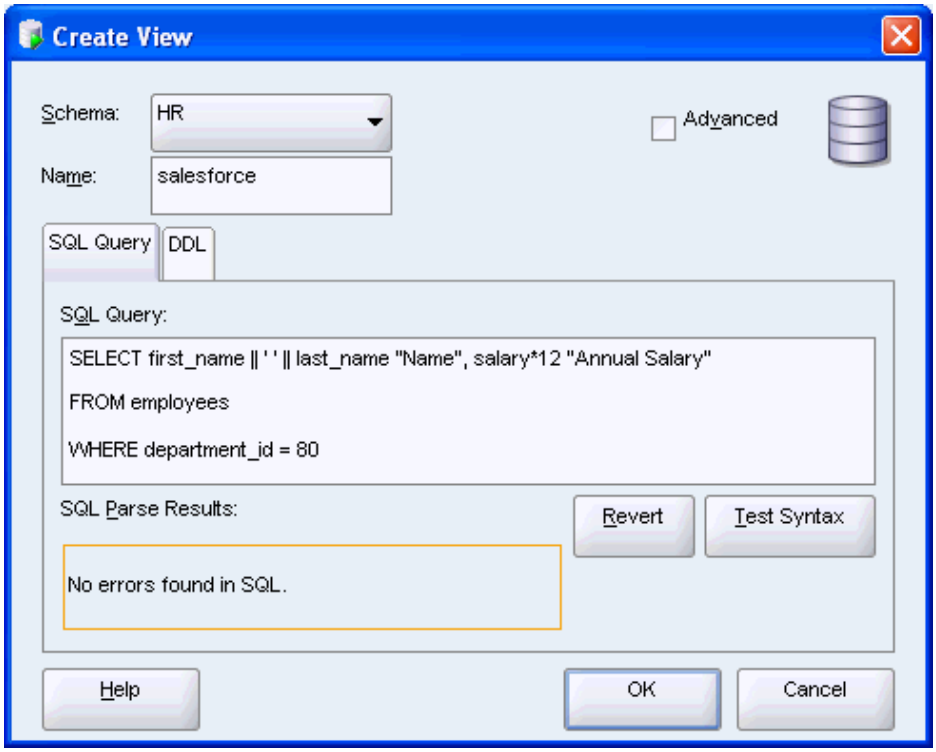
2. 「**New View**」を選択します。



3. 「Create View」ウィンドウで、次のパラメータを入力します。
   - 「Schema」が HR に設定されていることを確認します。
   - 「Name」を SALESFORCE に設定します。
   - 「SQL Query」を次のように設定します。

```
SELECT first_name || ' ' || last_name "Name", salary*12 "Annual Salary"
FROM employees
WHERE department_id = 80
```

4. 「SQL Parse Results」で「**Test Syntax**」をクリックします。



5. 「**OK**」をクリックします。

これで新しいビューが作成されました。このビューを作成するための SQL 文は次のとおりです。

```
CREATE VIEW salesforce AS
SELECT first_name || ' ' || last_name "Name",
 salary*12 "Annual Salary"
FROM employees
WHERE department_id = 80;
```

例 3-6 では、企業および作業場所にいるすべての従業員のビューを、「文字ファンクションの使用」で使用した問合せと同じように作成します。

例 3-6 SQL スクリプトでビューの作成

```
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;
```

スクリプトの結果が続きます。

```
CREATE VIEW succeeded.
```

これで 4 つの別々の表の情報を利用した、つまり 4 方向の**結合**を行った新しいビューが作成されました。「Connections」ナビゲーション階層では、「Views」のとなりのプラス記号をクリックすると、emp_locations を確認できます。

ビューの更新

SQL Developer インタフェースでビューのプロパティを変更するには、次の手順を実行します。

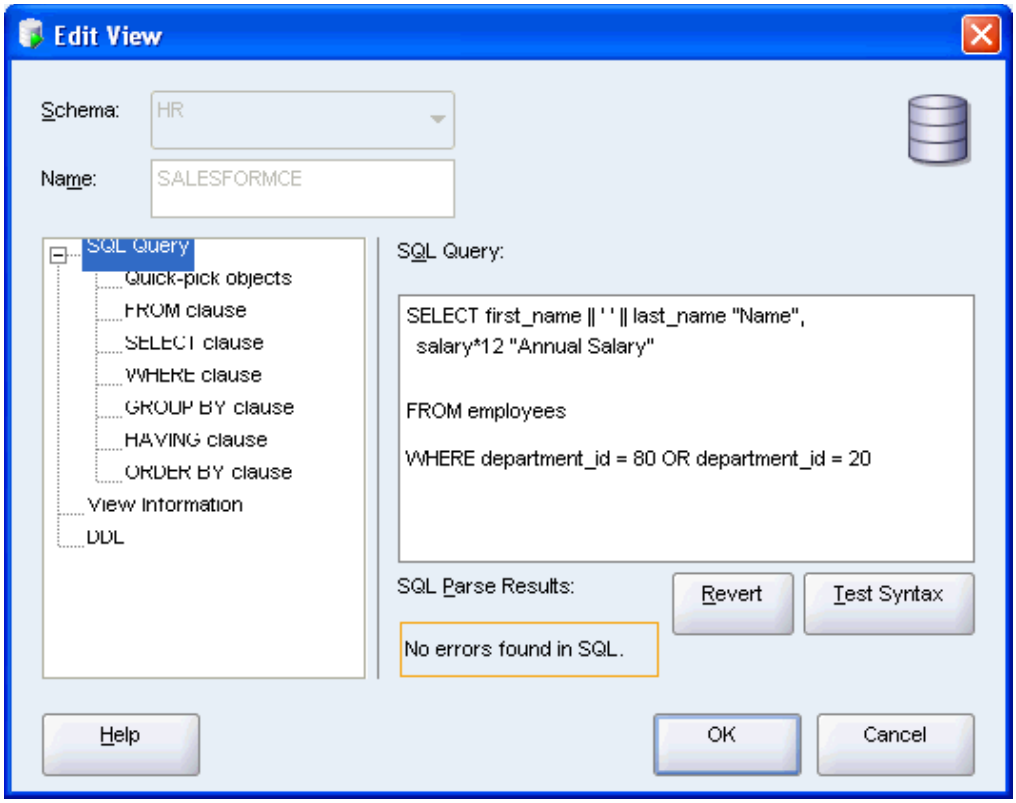
マーケティング部門の従業員を追加して salesforce ビューを変更し、ビュー名を sales_marketing に変更します。

1. 「Connections」ナビゲーション階層で、salesforce ビューを右クリックします。
2. 「**Edit**」を選択します。

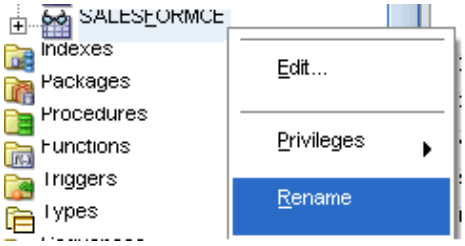
3. 「Edit View」ウィンドウで、最後の行に `OR department_id = 20` を追加して SQL 問合せを変更します。

   「**Test Syntax**」をクリックします。

   「**OK**」をクリックします。



4. ビューの名前を変更するには、`salesforce` を右クリックし、「**Rename**」を選択します。



5. 「Rename」ウィンドウで、「**New View Name**」を SALES_MARKETING に設定します。

   「**Apply**」をクリックします。

6. 「Confirmation」ウィンドウで「**OK**」をクリックします。

これでビューが変更されました。ビュー内容を変更し、ビュー名を変更するための SQL 文は次のとおりです。

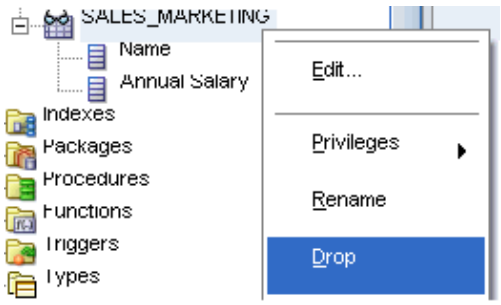
```
CREATE OR REPLACE VIEW salesforce AS query;
RENAME "SALESFORCE" to SALES_MARKETING;
```

ビューの削除

SQL Developer インタフェースを使用して、ビューを削除するには、次の手順を実行します。

DROP VIEW 文を使用して、sales_marketing ビューを削除します。

1. 「Connections」ナビゲーション階層で、sales_marketing ビューを右クリックします。
2. 「**Drop**」を選択します。



3. 「Drop」ウィンドウで、「**Apply**」をクリックします。
4. 「Confirmation」ウィンドウで「**OK**」をクリックします。

これでビューが削除されました。ビューを削除するための SQL 文は次のとおりです。

```
DROP VIEW sales_marketing;
```

参照：

- 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。
- 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。

順序の使用

順序は一意の連続した値を生成するデータベース・オブジェクトであり、一意の主キーが必要なときに非常に役立ちます。hr スキーマにはすでに departments_seq、employees_seq および locations_seq という 3 つの順序があります。

順序は次の擬似列を介して使用します。

- CURRVAL 擬似列は、順序の現在の値を戻します。CURRVAL は、NEXTVAL への最初の呼出しにより順序が開始された後にのみ使用できます。
- NEXTVAL 擬似列は順序を増分し、次の値を戻します。初めて NEXTVAL を使用したときは、順序の初期値が戻されます。NEXTVAL への次の参照では、定義済の増分により順序値が増分され、新しい値が戻されます。

順序は使用規則を除いて他のオブジェクトに接続されません。表の主キーを移入するために順序を使用する場合は、順序を表にリンクするためにネーミング規則を使用することをお薦めします。この説明においては、そのような順序に対するネーミング規則は *table_name*_seq です。

参照：

- 『Oracle Database SQL 言語リファレンス』

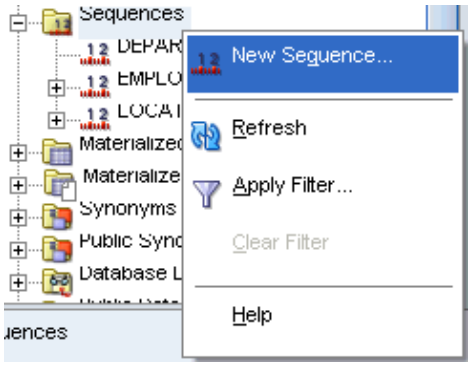
順序の作成

順序は SQL Developer インタフェースまたは「SQL Statement」ウィンドウを使用して作成できます。

SQL Developer インタフェースを使用して、順序を作成するには、次の手順を実行します。

次の手順では、evaluations 表の主キーで使用できる順序、evaluations_seq を作成します。

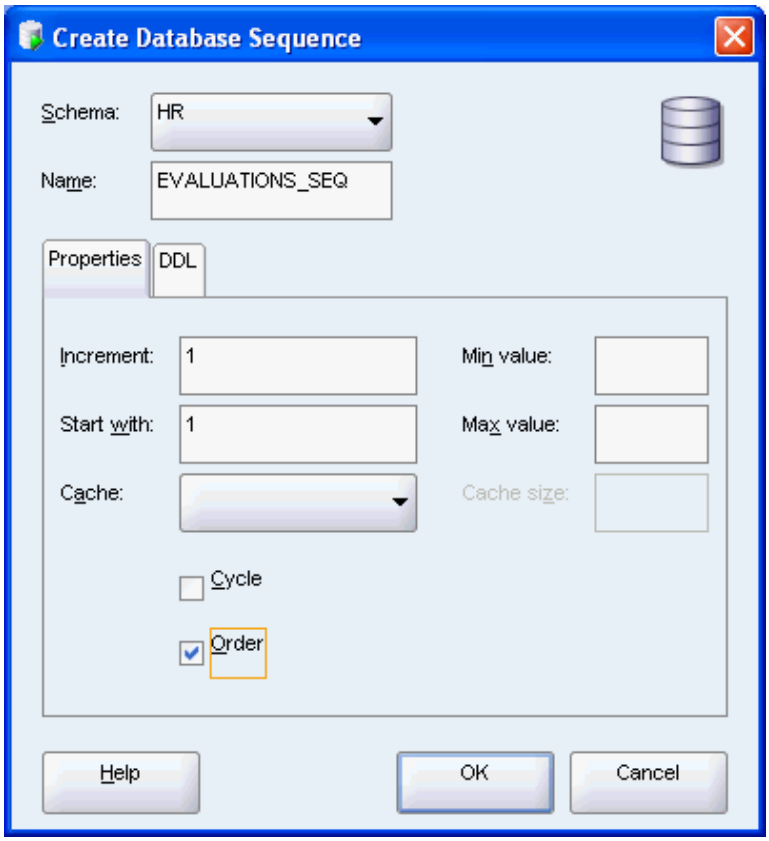
1. 「Connections」ナビゲーション階層で**「Sequences」**を右クリックします。
2. **「New Sequence」**を選択します。



3. 「New Sequence」ウィンドウで、次のパラメータを入力します。
   - 「**Schema**」が HR に設定されていることを確認します。
   - 「**Name**」を EVALUATIONS_SEQ に設定します。

   「Properties」タブで、次を実行します。
   - 「**Increment**」を 1 に設定します。
   - 「**Starts With**」を 1 に設定します。
   - 「**Order**」にチェックを入れます。

   「**OK**」をクリックします。



これで evaluations 表の主キーで使用できる順序が作成されました。プラス記号をクリックして順序ツリーを展開すると、新しい順序を確認できます。順序を作成するための SQL 文は次のとおりです。

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

例 3-7 では、「SQL Statement」ウィンドウで必要な情報を直接入力して、さらに順序を作成します。

例 3-7 SQL スクリプトを使用した順序の作成

```
CREATE SEQUENCE test_seq INCREMENT BY 5 START WITH 5 ORDER;
```

スクリプトの結果が続きます。

```
CREATE SEQUENCE succeeded.
```

参照：

- 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。

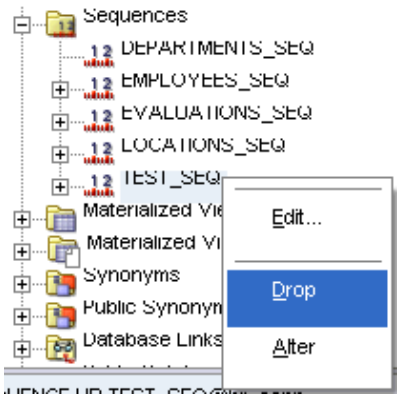
順序の削除

順序を削除するには、SQL 文である DROP SEQUENCE を使用します。SQL Developer でどのように順序が削除されるかを表示するには、以前に作成した test_seq 順序を使用します。新規順序が「Connections」階層ナビゲータに表示されない場合、「Refresh」アイコンをクリックします。

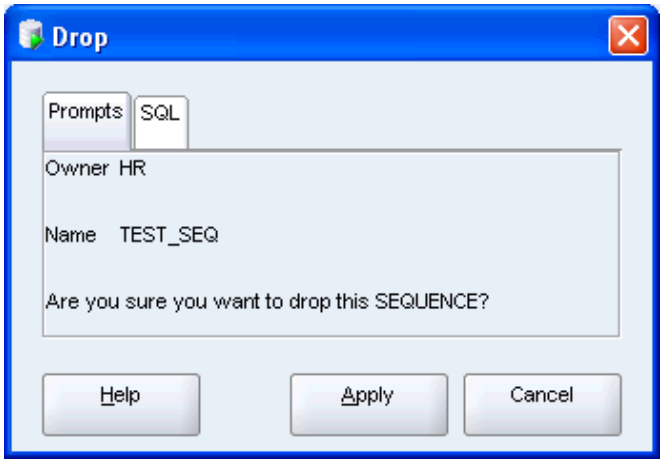
順序を削除するには、次の手順を実行します。

次の手順に従い、順序を削除します。

1. 「Connections」ナビゲータで、test_seq 順序を右クリックします。



2. 「Drop」ウィンドウで、「**Apply**」をクリックします。



3. 「Confirmation」ウィンドウで「**OK**」をクリックします。

これで test_seq 順序が削除されました。順序を削除するための SQL 文は次のとおりです。

```
DROP SEQUENCE "HR"."TEST_SEQ";
```

参照：

- 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。

シノニムの使用

シノニムはスキーマ・オブジェクトの別名であり、SQL 文を簡素化するために、またはセキュリティ強化の目的で実際のデータベース・オブジェクトの名前をわからなくするために使用できます。さらに、データベースの表の名前を（departments から divisions のように）変更する場合、departments のシノニムを作成し、以前と同じアプリケーション・コードを使用し続けることができます。

SQL Developer インタフェースを使用して、シノニムを作成するには、次の手順を実行します。

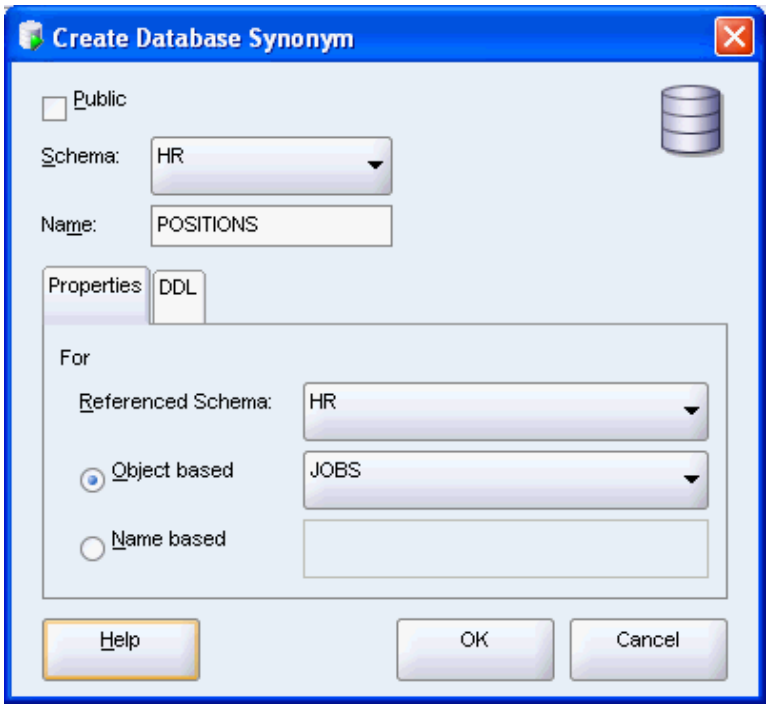
次の手順では、jobs スキーマ・オブジェクトのかわりに使用できるシノニム、positions を作成します。

1. 「Connections」ナビゲーション階層で、**「Synonyms」**を右クリックします。
2. **「New Synonym」**を選択します。

3. 「New Synonym」ウィンドウで、次のパラメータを設定します。
   - **「Schema」**が HR に設定されていることを確認します。
   - **「Name」**を POSITIONS に設定します。

   「Properties」タブで、次を実行します。
   - **「Object Based」**を選択します。これによりシノニムは表、ビュー、順序などの特定のスキーマ・オブジェクトを参照します。
   - **「Object Based」**を JOBS に設定します。

   「OK」をクリックします。



これで jobs 表に対して positions シノニムが作成されました。シノニムを作成する SQL 文は次のとおりです。

```
CREATE SYNONYM positions FOR jobs;
```

例 3-8 では、表名 jobs のかわりに新しい positions シノニムを使用します。

例 3-8 シノニムの使用

```
SELECT first_name || ' ' || last_name "Name", p.job_title "Position"
FROM employees e, positions p
WHERE e.job_id = p.job_id
ORDER BY last_name;
```

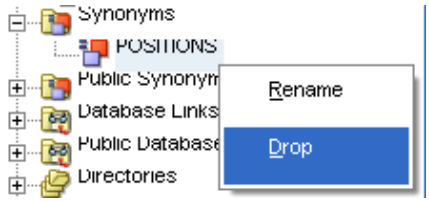
問合せの結果が表示されます。

```
Name                 Position
-------------------- -------------------------
Ellen Abel           Sales Representative
Sundar Ande          Sales Representative
Mozhe Atkinson       Stock Clerk
David Austin         Programmer
...
197 rows selected
```

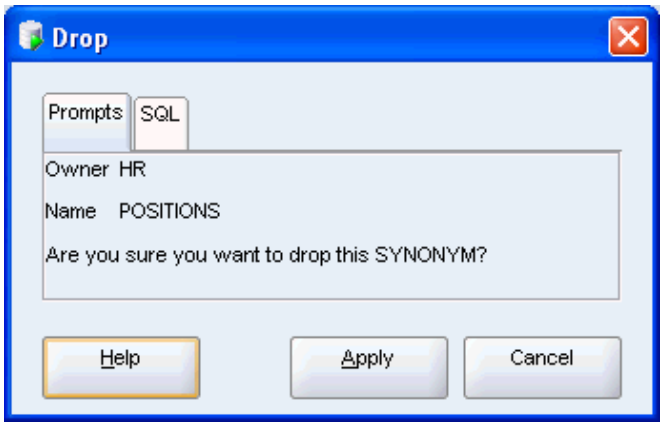
シノニムを削除するには、次の手順を実行します。

次の手順に従い、positions シノニムを削除します。

1. 「Connections」ナビゲータで、positions シノニムを右クリックします。
2. 「**Drop**」を選択します。



3. 「Drop」ウィンドウで、「**Apply**」をクリックします。



4. 「Confirmation」ウィンドウで「**OK**」をクリックします。

これで positions シノニムが削除されました。シノニムを削除するための SQL 文は次のとおりです。

```
DROP SYNONYM positions;
```

参照：

- 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。
- 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。

4

ストアド・プロシージャの開発および使用

この章では、Oracle Database の命令型言語である PL/SQL の使用について説明します。

この章の内容は次のとおりです。

ストアド・プロシージャの概要

SQL を使用してデータベースに問い合せる方法は学びました。ただし、この方法はエンタープライズ・アプリケーションの構築には不十分です。PL/SQL は予測されるプロシージャ構成およびネームスペース構成を持つ第 3 世代の言語で、SQL との密接な統合により複雑で強力なアプリケーションを構築できます。これは、PL/SQL がデータベースで実行されるため、別の接続を確立することなく SQL 文を含めることが可能です。

PL/SQL を使用して作成し、データベースに格納できるプログラム・ユニットの主要なタイプは、スタンドアロン・プロシージャ、スタンドアロン・ファンクションおよびパッケージです。一度データベースに格納されると、これらの PL/SQL コンポーネントは総称して**ストアド・プロシージャ**と呼ばれます。これは、複数の異なるアプリケーションに対するブロックの作成に使用できます。

スタンドアロン・プロシージャおよびファンクションは、一部のプログラム・ロジックのテストに便利ですが、パッケージ内にすべてのコードを配置することをお薦めします。パッケージは、他のシステムへの移植が容易で、さらにプログラム・ユニット名にプログラム名を組み込むことができます。たとえば、旧バージョンの Oracle Database で `continue` というスキーマ・レベル・プロシージャを作成した場合、新しいバージョンの Oracle Database をインストールしてこれを移入すると、コードはコンパイルできません。これは、Oracle が最近、現行のループの反復を終了し、制御を次の反復に転送する文 `CONTINUE` を導入したためです。パッケージ内にプロシージャを作成した場合、プロシージャ *`package_name`*`.continue` はその名前が取得されることを回避できます。

この章の次の項（4-2 ページの「スタンドアロン・プロシージャおよびファンクションの作成および使用」）では、スタンドアロン・プロシージャおよびファンクションの作成および使用方法を説明します。必要がない場合は、次の項をスキップして 4-9 ページの「パッケージの作成および使用」へ移動してください。

スタンドアロン・プロシージャおよびファンクションの作成および使用

Oracle Database では、一度のみ記述し、テストしてから必要とするアプリケーションにアクセスしたコードをデータベース内に格納できます。データベース内に常駐するプログラム単位は、コードが呼び出されたときにデータを一貫して処理することで、アプリケーション開発プロセスを容易にし、一貫性を実現します。

ファンクション（値を戻すもの）およびプロシージャ（値を戻さないもの）などのスキーマ・レベルまたはスタンドアロン・サブプログラムは、Oracle Database でコンパイルされ、格納されます。一度コンパイルされると、そのようなサブプログラムはスキーマ・オブジェクトである**ストアド・プロシージャ**または**ストアド・ファンクション**になり、Oracle Database に接続しているアプリケーションから参照されるかコールされることがあります。起動時に、ストアド・プロシージャおよびストアド・ファンクションはパラメータを受け入れることができます。

プロシージャおよびファンクションは、基本的な PL/SQL ブロック構造に従い、次の要素で構成されます。

- キーワード `DECLARE` で場合により開始する宣言部分は、アプリケーション・ロジックで使用される変数および定数を識別します。この部分はオプションです。
- `BEGIN` で始まり `END` で終了する実行可能部分には、アプリケーション・ロジックが含まれます。この部分は必須です。
- `EXCEPTION` で始まる例外処理部分は、ブロックの実行可能部分で発生する可能性のあるエラー条件を処理します。この部分はオプションです。

PL/SQL ブロックの一般的な書式は次のとおりです。それぞれのストアド・プログラム・ユニットはユニットを名付けるヘッダーを持ち、ファンクション、プロシージャまたはパッケージのいずれかとして識別します。

```
Header AS
[declaration statements
  ...]
BEGIN
  ...
[EXCEPTION
  ...]
END;
```

参照 :

- 宣言するプロシージャの構文の詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

プロシージャおよびファンクションの作成

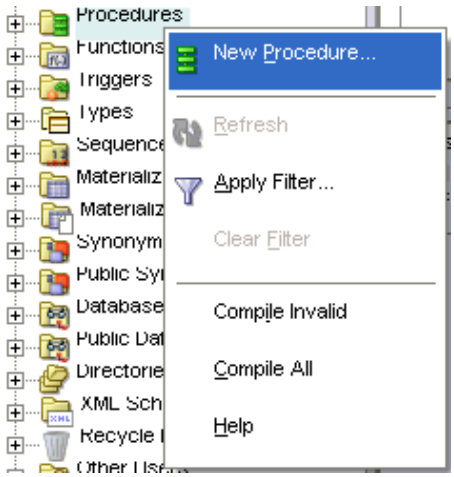
プロシージャおよびファンクションの作成に関する SQL 文は、それぞれ CREATE PROCEDURE および CREATE FUNCTION です。実際には、CREATE OR REPLACE 文を使用するのが最適です。これらの文の一般的な書式は次のとおりです。

```
CREATE OR REPLACE procedure_name(arg1 data_type, ...) AS
BEGIN
  ....
END procedure_name;

CREATE OR REPLACE procedure_name(arg1 data_type, ...) AS
BEGIN
  ....
END procedure_name;
```

プロシージャを作成するには、次の手順を実行します。

evaluation 表に新規の行を作成するプロシージャ add_evaluation を作成します。

1. 「Connections」ナビゲーション階層で、**「Procedures」**を右クリックします。
2. **「New Procedure」**を選択します。

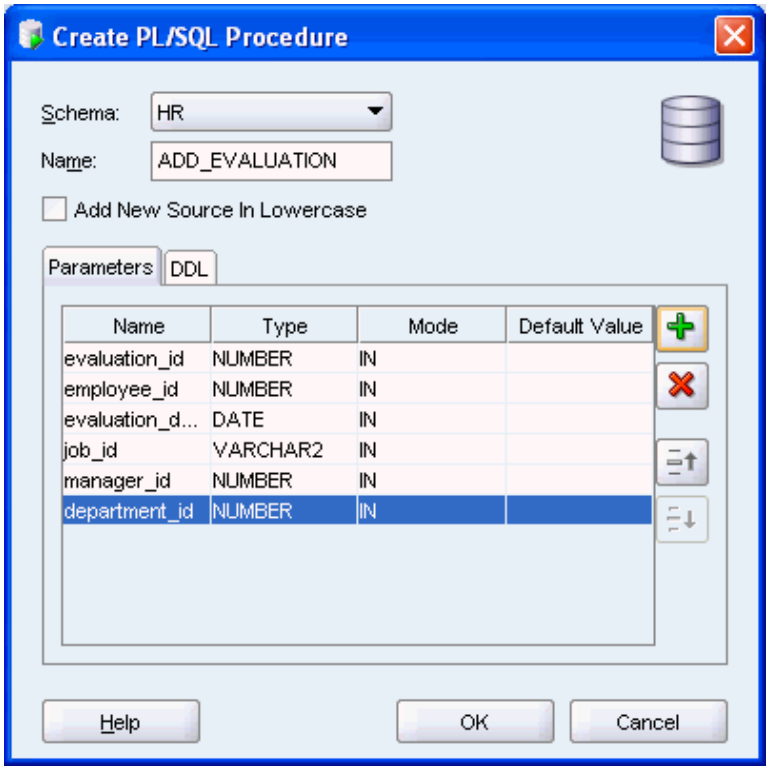


3. 「New Procedure」ウィンドウで、次のパラメータを設定します。
   - 「**Schema**」が HR に設定されていることを確認します。
   - 「**Name**」を ADD_EVALUATION に設定します。

   「Parameters」タブで、「**Add Column**」アイコン（プラス記号）をクリックしてプロシージャの最初のパラメータを指定します。「**Name**」を eval_id に、「**Type**」を NUMBER に、そして「**Mode**」を IN に設定して、「**Default Value**」を空のままにします。

   同様に、同じ順序で次のパラメータを追加します。
   - employee_id: 「**Type**」を NUMBER に、「**Mode**」を IN に設定して、「**Default Value**」を空のままにします。
   - evaluation_date: 「**Type**」を DATE に、「**Mode**」を IN に設定して、「**Default Value**」を空のままにします。
   - job_id: 「**Type**」を VARCHAR2 に、「**Mode**」を IN に設定して、「**Default Value**」を空のままにします。
   - manager_id: 「**Type**」を NUMBER に、「**Mode**」を IN に設定して、「**Default Value**」を空のままにします。
   - department_id: 「**Type**」を NUMBER に、「**Mode**」を IN に設定して、「**Default Value**」を空のままにします。

   「**OK**」をクリックします。



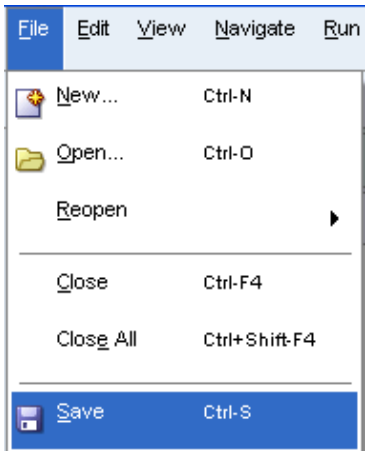
4. ADD_EVALUATION ペインを次のコードを使用してオープンします。

   ペインのタイルがイタリック・フォントになっていることに注意してください。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。

```
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
) AS
BEGIN
  NULL;
END ADD_EVALUATION;
```

5. 「**File**」メニューから「**Save**」を選択し、新規のプロシージャを保存します。または、**[Ctrl] を押しながら [S]** キーを押します。

   Oracle Database により、プロシージャは保存前に自動的にコンパイルされます。

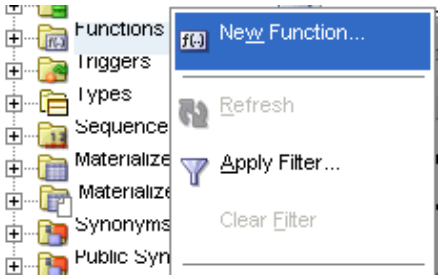


   add_evaluation ペインのタイルがイタリックではなく標準フォントであることに注意してください。プロシージャがデータベースに保存されたことを示します。

ファンクションを作成するには、次の手順を実行します。

特定のカテゴリ内のパフォーマンスに基づいて重み付けされたスコアを計算する新規のファンクション calculate_score を作成します。

1. 「Connections」ナビゲーション階層で、「**Functions**」を右クリックします。
2. 「**New Function**」を選択します。

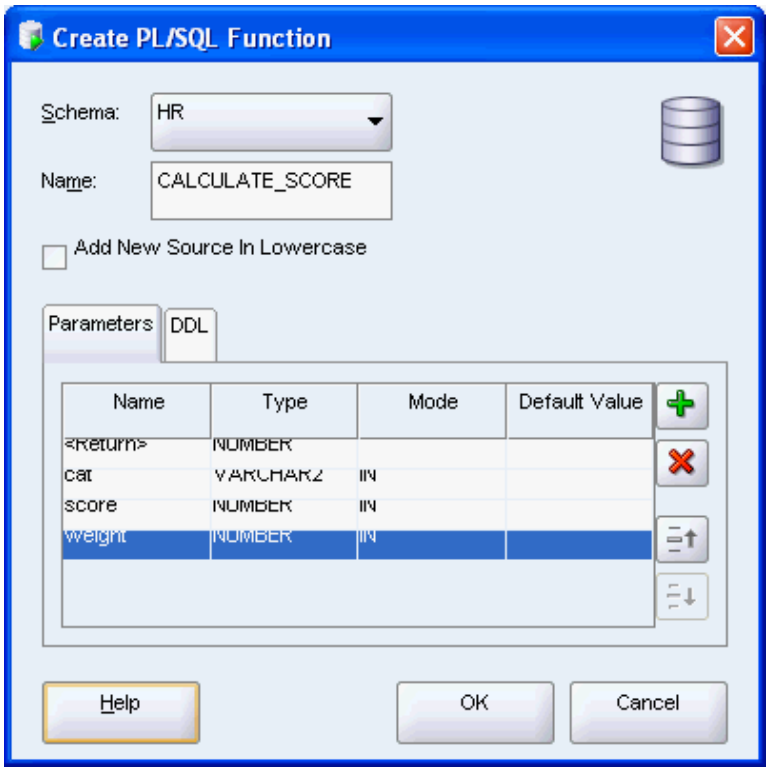


3. 「New Function」ウィンドウで、次のパラメータを設定します。
   - 「**Schema**」が HR に設定されていることを確認します。
   - 「**Name**」を CALCULATE_SCORE に設定します。

   「Parameters」ペインで、「**<return> Type**」を NUMBER に設定します。

   同様に、同じ順序で次のパラメータを追加します。
   - cat: 「**Type**」を VARCHAR2 に、「**Mode**」を IN に設定して、「**Default Value**」を空のままにします。
   - score: 「**Type**」を NUMBER に、「**Mode**」を IN に設定して、「**Default Value**」を空のままにします。
   - weight: 「**Type**」を NUMBER に、「**Mode**」を IN に設定して、「**Default Value**」を空のままにします。

   「**OK**」をクリックします。



4. calculate_score ペインを次のコードでオープンします。

   ペインのタイルがイタリック・フォントになっていることに注意してください。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。

```
CREATE OR REPLACE
FUNCTION calculate_score
( cat IN VARCHAR2
, score IN NUMBER
, weight IN NUMBER
) RETURN NUMBER AS
BEGIN
  RETURN NULL;
END calculate_score;
```

5. 「File」メニューから「Save」を選択して、新規のファンクションを保存します。または、**[Ctrl] を押しながら [S]** キーを押します。

   Oracle Database により、ファンクションは保存前に自動的にコンパイルされます。

   CALCULATE_SCORE ペインのタイルがイタリックではなく標準フォントであることに注意してください。プロシージャがデータベースに保存されたことを示します。

   参照 :

   - CREATE 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。
   - CREATE 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。

プロシージャおよびファンクションの変更

すでに新規のプロシージャおよびファンクションを作成しました。ただし、サブプログラムの署名でしか構成されていないため、この項では、サブプログラムの本体を編集します。

ファンクションを変更するには、次の手順を実行します。

ファンクション calculate_score を編集して、特定のカテゴリに対する評価の重み付けの値を決定します。

1. calculate_score ペインで、次のコードを使用して、ファンクションの本体を置換します。新規のコードは太字フォントで表示されます。

```
BEGIN
  RETURN score * weight;
END calculate_score;
```

2. ファンクションをコンパイルおよび保存します。**[Ctrl] を押しながら [S]** キーを使用します。

   参照 :

   - ALTER 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。
   - ALTER 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。

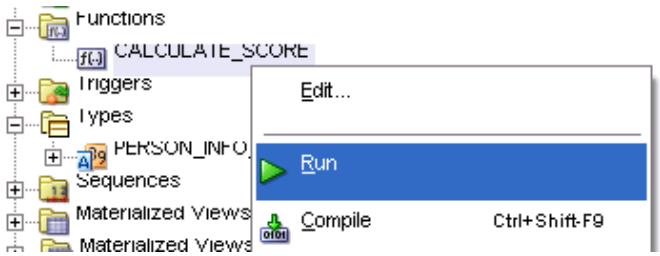
プロシージャおよびファンクションのテスト

次に、変更を加えたファンクションをテストします。

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。

ファンクション calculate_score をテストします。

1. 「Connections」ナビゲータ階層で、calculate_score ファンクションを右クリックします。「Run」を選択します。



2. PL/SQL を実行するウィンドウで、「PL/SQL Block」ペイン内をクリックして、score および weight 変数に対する割当を編集します。新規のコードは太字フォントで表示されます。

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

「**OK**」をクリックします。

3. 「Running - Log」ペインで、次の結果を確認します。

```
Connecting to the database hr_conn.
v_Return = 1.6
Process exited.
Disconnecting from the database hr_conn.
```

参照 :

- プロシージャおよびファンクションの実行を必要とするシステム権限ユーザーの詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。
- 動的 SQL の EXECUTE IMMEDIATE 文の使用方法の詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

プロシージャおよびファンクションの削除

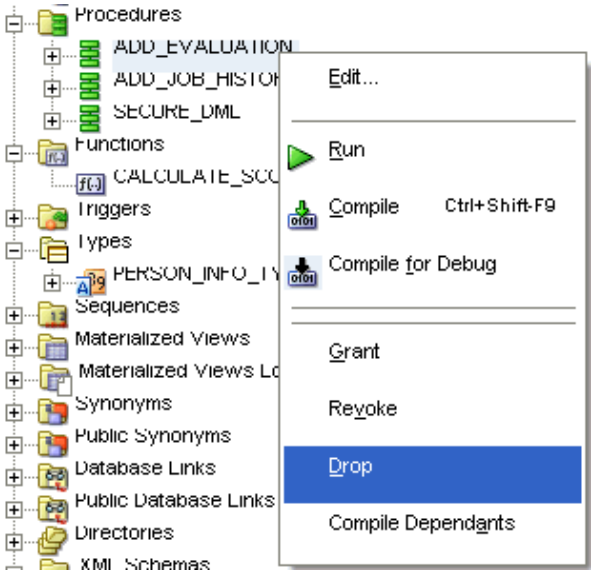
「Connection」ナビゲータまたは SQL 文の DROP を使用して、データベースからプロシージャまたはファンクションを削除できます。

プロシージャを削除するには、次の手順を実行します。

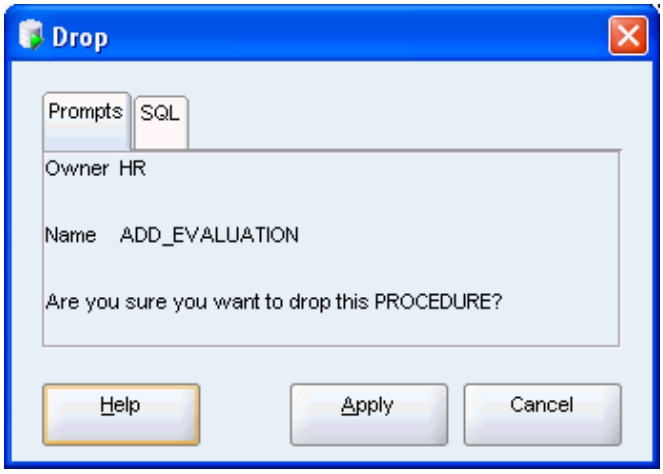
プロシージャ ADD_EVALUATION を削除します。

1. 「Connections」ナビゲータ階層で、ADD_EVALUATION ファンクションを右クリックします。

「**Drop**」を選択します。



2. 「Drop」ウィンドウで、**「Apply」**をクリックします。



3. 「Confirmation」ダイアログ・ボックスで、**「OK」**をクリックします。

データベースから ADD_EVALUATION プロシージャが削除されました。

参照：

- 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。

パッケージの作成および使用

前述の項で、スキーマ・オブジェクトであるプロシージャおよびファンクションを作成し、テストしました。この方法は、アプリケーションのサブセットまたは小さい機能をテストするのに便利です。

エンタープライズ・レベル・アプリケーションは非常に複雑で、インタフェースおよび型の一部は表示されますが、それ以外は他のファンクションおよびプロシージャでのみ使用され、ユーザーからはコールされません。PL/SQL を使用すると、同一の**パッケージ**内に配置することによって、これらのサブプログラム間の関係を形式に従って指定できます。パッケージは、PL/SQL 型、変数、ファンクションおよびプロシージャなどの関連要素を論理的にグループ化して、名前を指定するスキーマ・オブジェクトです。パッケージ内部へのこれらの要素のカプセル化は、4-2 ページの「ストアド・プロシージャの概要」で説明されている名称の重複などの意図しない結果やアプリケーションの存続期間の超過を防ぎます。

パッケージ内で定義されたプロシージャおよびファンクションは、**パッケージ済サブプログラム**になります。他のサブプログラムまたは PL/SQL ブロック内でネストされたプロシージャおよびファンクションは、**ローカル・サブプログラム**と呼ばれます。囲みブロック内にのみ存在し、外部からは参照されません。

4-2 ページの「スタンドアロン・プロシージャおよびファンクションの作成および使用」のようなスタンドアロン・プロシージャおよびファンクションがサイズの大きな規模の開発に限定する理由は、スカラー・パラメータ（NUMBER、VARCHAR2 および DATE）のみが送受信可能であるためです。ただし、パッケージ仕様部で定義されていない場合、コンポジット構造 RECORD を使用することはできません。

パッケージには仕様部および本体の 2 つの部分があります。

パッケージは、パッケージ外部から参照可能な型、変数、定数、例外、カーソル、ファンクションおよびプロシージャを宣言する**パッケージ仕様部**によって定義されます。仕様部はパッケージへのインタフェースです。パッケージ内のサブプログラムをコールするアプリケーションは、パッケージ仕様部の名前およびパラメータを知っておく必要があります。

標準的なパッケージ仕様部は次のような形式です。

```
CREATE OR REPLACE PACKAGE package_name AS
  type definitions for records, index-by tables
  constants
  exceptions
  global variable declarations
  procedure procedure_1(arg1, ...);
  ...
  function function_1(arg1,...) return datat_ype;
  ...
END package_name;
```

パッケージ本体には、これらのサブプログラムを実装するコード、パッケージ内で開始されるすべてのプライベート・サブプログラムに対するコードおよびカーソルに対する問合せが含まれます。コールするアプリケーションを無効にすることなく、パッケージ本体の内部で実装の詳細を変更できます。

パッケージ本体は次のような形式です。

```
CREATE OR REPLACE PACKAGE BODY package_name AS
  PROCEDURE procedure_1(arg1,...) IS
    BEGIN
      ...
    EXCEPTION
      ...
    END procedure_1;
  ...
  FUNCTION function_1(arg1,...)  RETURN data_type IS result_variable data_type
    BEGIN
      ...
      RETURN result_variable;
    EXCEPTION
      ...
  END function_1;
  ...
END package_name;
```

参照:

- パッケージの作成の構文の詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

パッケージのガイドライン

Oracle Database によって供給されるパッケージに精通し、既存の機能と重複する書込みコードを回避する必要があります。

パッケージ本体に実装を書き込む前に、パッケージ仕様部を設計し定義する必要があります。仕様部には、コールするプログラムにパブリックに参照可能である部分のみが含まれ、パッケージ本体内のプライベートな宣言を非表示にします。これにより、実装詳細に関する他のプログラムの安全ではない依存性を回避できます。

パッケージの正常なコンパイルを妨げる、パッケージ本体内の正しく、有効なサブプログラム間の依存性を検出するため、PL/SQL はシングルパス・コンパイラを持ちます。パッケージ本体の上部でこれらの不明なサブプログラムを宣言し、後で指定する必要があります。このため、依存が無効化される可能性を最小限にするためにパッケージ仕様部または本体の最後に新しい要素を追加することをお薦めします。

参照：

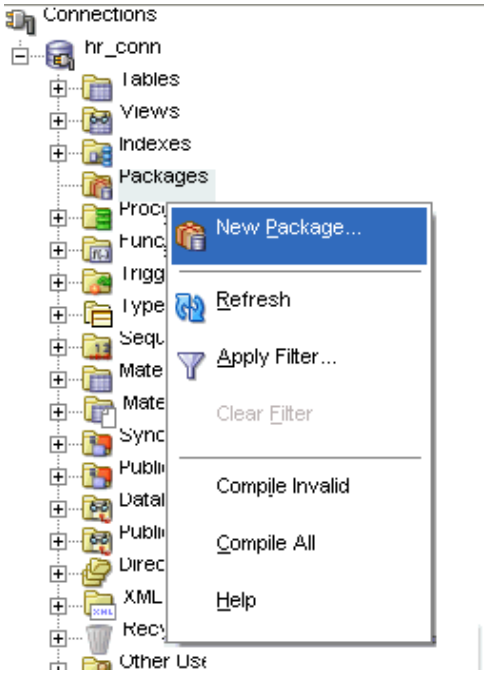
- PL/SQL パッケージの使用方法の詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。
- Oracle Database で使用可能なデフォルトのパッケージの詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

パッケージの作成

従業員評価の実行に必要な機能すべてをカプセル化するパッケージを作成します。パッケージの作成後は、パッケージ本体を作成するためにパッケージの変更方法を説明する 4-13 ページの「パッケージの変更」を参照してください。

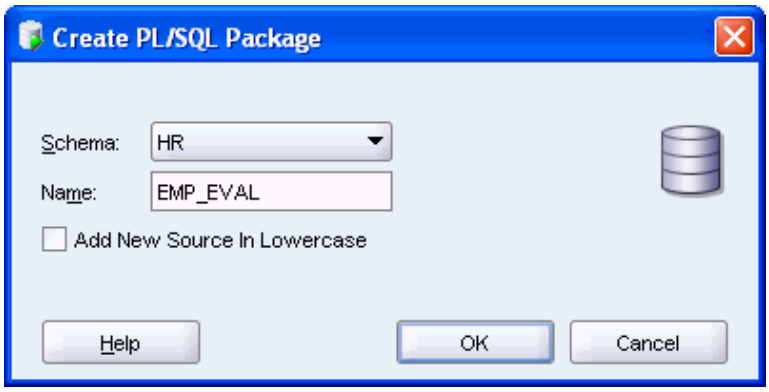
SQL Developer ナビゲーション階層でパッケージを作成するには、次の手順を実行します。

1. 「Connections」ナビゲーション階層で、**「Packages」**を右クリックします。
2. **「New Package」**を選択します。



3. 「Create PL/SQL Package」ダイアログで、次のパラメータを設定します。
   - 「**Schema**」が HR に設定されていることを確認します。
   - 「**Name**」を EMP_EVAL に設定します。

     「**OK**」をクリックします。



4. 次のコードを使用して、emp_eval ペインをオープンします。

```
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ることに注意してください。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。

5. 「**File**」メニューから「**Save**」を選択して、新規のパッケージをコンパイルして保存します。または、**[Ctrl] を押しながら [S]** キーを使用します。

   「Messages - Log」ペインで、パッケージを作成したことを確認します。

```
EMP_EVAL Compiled.
```

   emp_eval ペインのタイトルがイタリックではなく標準フォントであることに注意してください。プロシージャがデータベースに保存されたことを示します。

例 4-1 は、SQL ワークシートにパッケージを直接作成する方法を示しています。

例 4-1 PL/SQL パッケージの作成

```
CREATE OR REPLACE PACKAGE eval AS
  /* package */
END eval;
```

スクリプトの結果が続きます。

```
PACKAGE eval Compiled.
```

参照：

- CREATE PACKAGE 文（パッケージ仕様部）の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

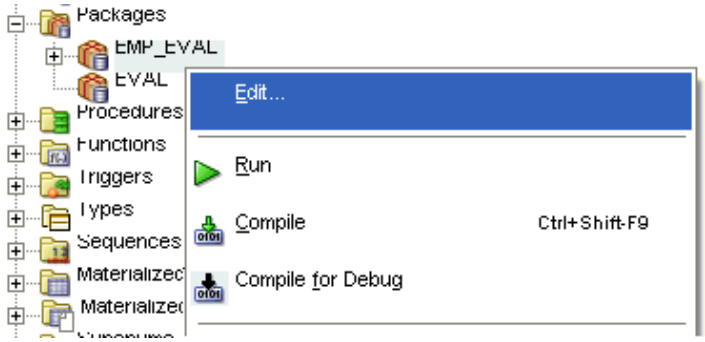
パッケージの変更

この項では、パッケージ emp_eval を変更します。

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。

一部のファンクションおよびプロシージャの指定によって emp_eval のパッケージ仕様部を変更します。

1. 「Connections」ナビゲーション階層で、「Packages」を選択して emp_eval を右クリックします。
2. 「**Edit**」を選択します。



3. EMP_EVAL ペインで、パッケージを編集します。新規のコードが太字フォントで表示されます。

```
create or replace
PACKAGE emp_eval AS
  PROCEDURE eval_department(department_id IN NUMBER);
  FUNCTION calculate_score(evaluation_id IN NUMBER
                         , performance_id IN NUMBER)
                           RETURN NUMBER;
END emp_eval;
```

4. パッケージ仕様部をコンパイルします。

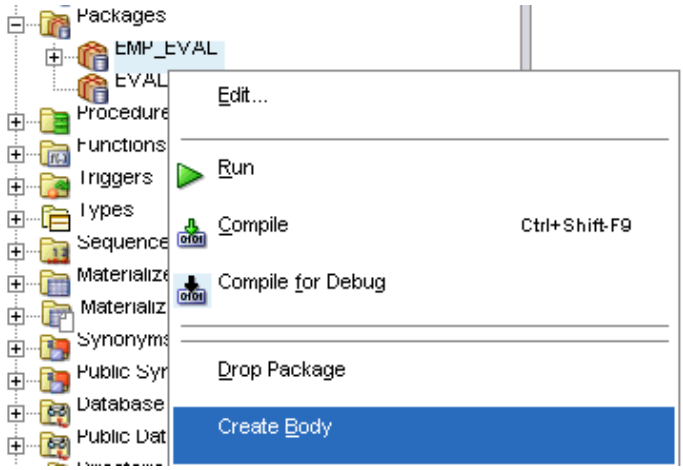
   パッケージが正常にコンパイルしたことを確認する次のメッセージが表示されます。

```
EMP_EVAL Compiled.
```

パッケージ本体を作成するには、次のようにします。

一部のファンクションおよびプロシージャの指定によって emp_eval のパッケージ本体を作成します。

1. 「Connections」ナビゲーション階層で、emp_eval を右クリックします。
2. 「**Create Body**」を選択します。



3. emp_eval 本体ペインに、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(department_id IN NUMBER) AS
  BEGIN
    /* TODO implementation required */
    NULL;
  END eval_department;

  FUNCTION calculate_score(evaluation_id IN NUMBER
                         , performance_id IN NUMBER)
                           RETURN NUMBER AS
  BEGIN
    /* TODO implementation required */
    RETURN NULL;
  END calculate_score;

END emp_eval;
```

4. パッケージ本体をコンパイルし、保存します。

   パッケージ本体が正常にコンパイルしたことを確認する次のメッセージが表示されます。

```
EMP_EVAL Body Compiled.
```

参照：

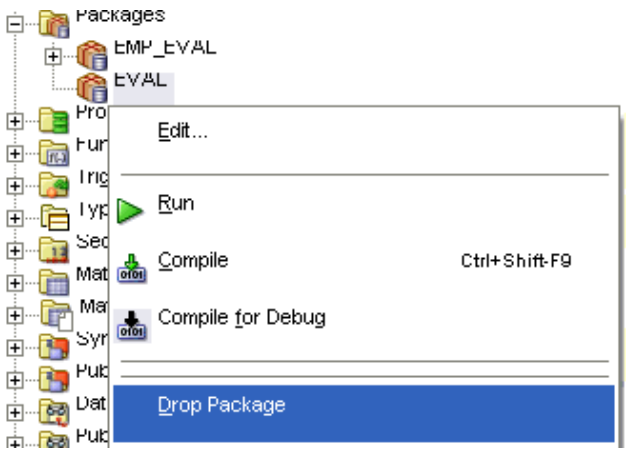
- CREATE PACKAGE BODY 文の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。
- ALTER PACKAGE 文の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

パッケージの削除

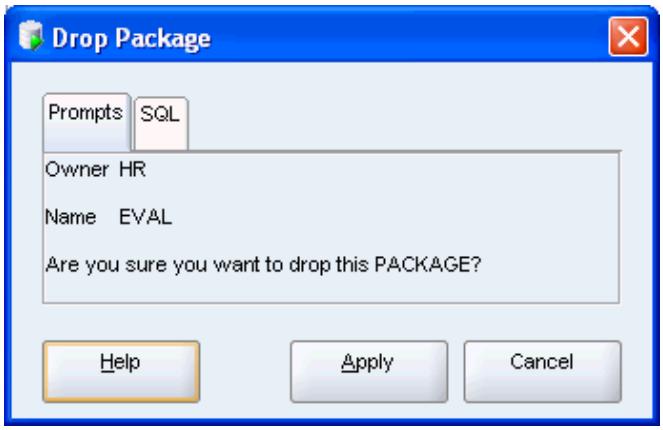
「Connections」ナビゲータ階層か、SQL DROP 文を使用してデータベースからパッケージを削除できます。パッケージを削除する際は、データベースからパッケージ仕様部およびパッケージ本体を削除します。

パッケージを削除するには、次の手順を実行します。

1. 「Connections」ナビゲータ階層で、「Packages」を選択して EVAL を右クリックします。
2. **「Drop Package」**を選択します。



3. パッケージの削除のダイアログで、**「Apply」**をクリックします。



4. 「Confirmation」ダイアログで、**「OK」**をクリックします。

参照：

- DROP PACKAGE 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。

変数および定数の使用

PL/SQL によって SQL が得る重要な利点の 1 つは、プログラミング構成で変数および定数を使用できることです。

変数は、特定のデータ型の指定した値を保持するためにユーザーによって定義されます。この値は変更可能で、実行時に変更できます。

定数は、変更不可の値を保持します。コンパイラによってこの値が不変であることが確認されると、変更可能なコードもコンパイルしません。時間経過によるコード・ベースのメンテナンスをより容易にするため、ダイレクトな値のかわりにコードの定数を使用する必要があります。定数として変更を加えない値すべてを宣言する際、定数はコンパイル・コードを最適化します。

参照 :

- 変数および定数の詳細は、『Oracle Database 概要』を参照してください。

PL/SQL データ型

VARCHAR2、DATE、NUMBER などの SQL データ型に加えて、Oracle Database は PL/SQL を介してのみ使用できるデータ型もサポートします。これらのデータ型には、BOOLEAN、RECORD などのコンポジット・データ型、REF CURSOR、INDEX BY TABLE などの参照型、および数字、文字および日付要素を示す多くの特殊な型が含まれます。数値型 PLS_INTEGER は、整数のバイナリ演算を実行し、パフォーマンスを大幅に向上させるため、特に有効です。これらの PL/SQL 型は、スキーマ・レベルでは使用できず（したがって、表でも使用できず）、パッケージ内で定義される型およびプロセスでのみ使用できることに注意してください。

参照 :

- PL/SQL データ型に関する一般情報は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。
- PLS_INTEGER 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。

変数および定数の使用

変数および定数はあらゆる SQL または PL/SQL データ型を持ち、サブプログラムの宣言ブロックで宣言されます。デフォルトでは、宣言された変数は NULL 値を持ちます。定数を定義する場合、CONSTANT 句を使用し、すぐに値を割り当てる必要があります。

参照 :

- 『Oracle Database PL/SQL 言語リファレンス』

コメントの使用

PL/SQL では、インライン・コメント（--）は二重ハイフンで始まり、ラインの末尾まで拡張されます。複数行にまたがるコメントの場合は、先頭はスラッシュおよびアスタリスク（/*）で、末尾はアスタリスクおよびスラッシュ（*/）である必要があります。

参照 :

- 『Oracle Database PL/SQL 言語リファレンス』

識別子の使用

識別子によって、定数、変数およびサブプログラムなどの PL/SQL プログラム単位の名前が付けられます。すべての識別子は最大 30 文字で、文字で始まり、文字、数字および記号（$'、'_' および '#'）の組合せが続いている必要があります。他の記号は識別子に使用できません。

文字列および文字リテラルの管理時以外は、PL/SQL では大文字、小文字の区別がないため、大文字および小文字を交互に使用できます。つまり、識別子 last_name は LAST_NAME と同等です。2 番目の識別子を宣言すると、エラーが発生します。

変数および定数に対して有効な名前を使用し、適切なネーミング規則を使用する必要があります。たとえば、各定数名を cons_ で開始できます。また、識別子として予約語は使用できません。

参照 :

- 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。
- 識別子に関するデータを収集する方法の詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。
- 識別子名を解決する 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。

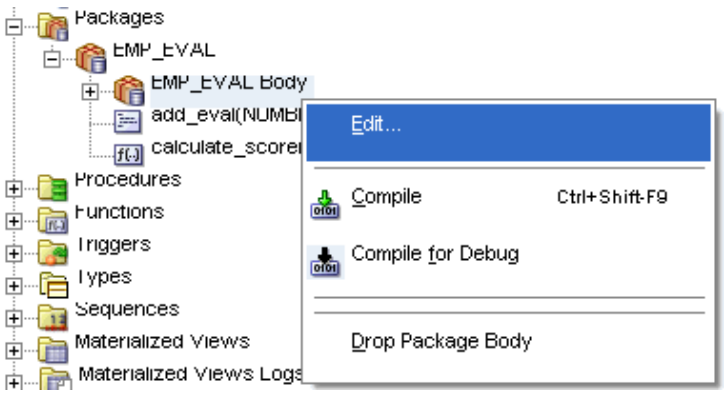
変数および定数の宣言

emp_eval および calculate_score パッケージの新しいファンクションを更新します。これらは、異なるカテゴリで重み付けされたスコアすべてを組み合せて、従業員評価の最終的なスコアを計算します。

変数および定数を宣言するには、次の手順を実行します。

1. 「Connections」ナビゲーション階層で、「Packages」の横にあるプラス記号（+）をクリックしてグループを展開します。
2. emp_eval の横にあるプラス記号をクリックしてパッケージを展開します。
3. **EMP_EVAL Body** を右クリックします。
4. **「Edit」**を選択します。

   emp_eval 本体ペインが表示されます。



5. 次のコードで説明しているように、emp_eval 本体ペインで、変数および定数の追加によってファンクション calculate_score を変更します。新しいコードは太字で表示されます。

```
FUNCTION calculate_score(evaluation_id IN NUMBER
                       , performance_id IN NUMBER)
                         RETURN NUMBER AS
  n_score       NUMBER(1,0);                    -- a variable
  n_weight      NUMBER;                         -- a variable
  max_score     CONSTANT NUMBER(1,0) := 9;      -- a constant limit check
  max_weight    CONSTANT NUMBER(8,8) := 1;      -- a constant limit check
BEGIN
  RETURN NULL;
END calculate_score;
```

6. **[Ctrl]** を押しながら **[S]** キーを使用して、更新されたパッケージ本体を保存します。

   「Messages - Log」ペインに次のメッセージが表示されます。

```
EMP_EVAL Body Compiled
```

参照：

- 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。

データベース列と同一である構造を使用した変数の宣言

「変数および定数の宣言」で、2 つの変数 n_score および n_weight を追加してファンクション calculate_score を変更します。これらの変数はデータベース内表の値である scores 表に格納されている n_score、および performance_parts 表に格納されている n_weight を表します。これらの変数に使用したデータ型は、表の列データ型定義と一致します。

時間が経つにつれて、アプリケーションは進化し、列定義も変化する場合があります。これにより、calculate_score ファンクションが無効になる場合があります。コードのメンテナンスをより容易にするために、適切な列および行の定義に一致するデータ型を含む変数を定義する特別な修飾子を使用する必要があります。たとえば、%TYPE および %ROWTYPE です。

- %TYPE 属性は、表の列または別の変数のデータ型を提供します。これは、表のデータ型が変更になる場合、正しいデータ型割当ておよびランタイム時のファンクションの正しい実装を保証する際に効果的です。
- %ROWTYPE 属性により、RECORD 変数に対する表内の行が定義されます。表の行の列と RECORD 内の対応するフィールドは、同名で同じデータ型を含みます。%ROWTYPE を使用する利点は、%TYPE と同じです。詳細は、4-29 ページの「コンポジット・データ構造（レコード）の使用」を参照してください。

次のタスクは、ファンクションで %TYPE 属性を使用する方法を示しています。calculate_score ファンクションを編集して、ソース表の列に一致するデータ型を n_score および n_weight 変数に割り当てます。max_score および max_weight 定数は表の値との等価性を確認するために使用されるため、これらの定数も表の型と一致する必要があることに注意してください。

%TYPE 属性を使用するには、次の手順を実行します。

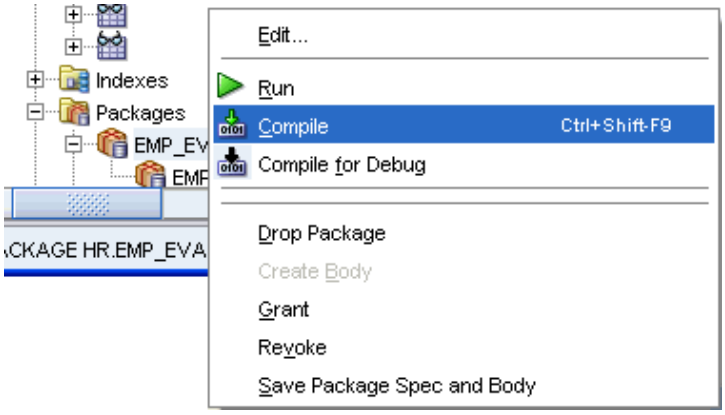
1. 次のコードで説明しているように、emp_eval 本体ペインで、変数の定義を変更してファンクション calculate_score を変更します。新しいコードは太字で表示されます。

```
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;               -- from SCORES
  n_weight      performance_parts.weight%TYPE;   -- from PERFORMANCE_PARTS
  max_score     CONSTANT scores.score%TYPE := 9; -- a constant limit check
  max_weight    CONSTANT performance_parts.weight%TYPE := 1;
                                                 -- a constant limit check
BEGIN
  RETURN NULL;
END calculate_score;
```

2. emp_eval パッケージ仕様部で、関数 calculate_score の宣言を変更します。

```
FUNCTION calculate_score(evaluation_id IN scores.evaluation_id%TYPE
                       , performance_id IN scores.performance_id%TYPE)
                         RETURN NUMBER;
```

3. 「Connections」ナビゲーション階層で、emp_eval パッケージを右クリックし、**「Compile」**を選択します。また、**[Ctrl]、[Shift] および [F9]** を押すとショートカットになります。



「Messages - Log」ペインに次のメッセージが表示されます。

```
EMP_EVAL Body Compiled
```

%ROWTYPE 属性を使用するには、次の手順を実行します。

4-32 ページの「明示的なカーソルの使用」の eval_department プロシージャで使用されるコードを参照してください。

参照 :

- 『Oracle Database PL/SQL 言語リファレンス』

変数への値の割当て

次の 3 つの一般的な方法で変数に値を割り当てられます。3 つの方法は、代入演算子の使用、変数への値の選択および変数のバインドです。この項では、3 つの方法の内の最初の 2 つを説明します。変数のバインドは、Application Express、Java、.NET および PHP の 2 日ガイド・シリーズで説明されています。

参照 :

- 『Oracle Database PL/SQL 言語リファレンス』
- 『Oracle Database 2 日で .Net 開発者ガイド』
- 『Oracle Database 2 日で PHP 開発者ガイド』
- 『Oracle Database 2 日で Java 開発者ガイド』
- 『Oracle Database 2 日で Application Express 開発者ガイド』

代入演算子を使用した値の割当て

サブプログラムの本体および宣言の両方で変数に値を割り当てられます。

次のコードは変数および定数の標準的な宣言を示します。プロシージャおよびファンクションでは、宣言ブロックは DECLARE キーワードを使用しません。かわりに、サブプログラム定義の AS キーワードが続きます。

例 4-2 宣言での変数の値の割当て

emp_eval 本体ペインで、新しい変数 running_total を追加して関数 calculate_score を変更します。running_total の値も、この関数の新しい戻り値です。戻り変数の初期値は 0 に設定します。running_total は、異なる精度と位取りを持つ 2 つの NUMBER の積を保持するため、一般 NUMBER として宣言されていることに注意してください。新しいコードは太字で表示されています。

```
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;               -- from SCORES
  n_weight      performance_parts.weight%TYPE;   -- from PERFORMANCE_PARTS
  running_total NUMBER               := 0;       -- used in calculations
  max_score     CONSTANT scores.score%TYPE := 9; -- a constant limit check
  max_weight    CONSTANT performance_parts.weight%TYPE:= 1;
                                                 -- a constant limit check
BEGIN
  RETURN running_total;
END calculate_score;
```

emp_eval 本体をコンパイルします。

サブプログラムの本体の変数に値を割り当てられます。ファンクションの本体内の running_total 変数の使用によってファンクション calculate_score を編集して、式の値を保持します。

例 4-3 ファンクションの本体での変数の値の割当て

次のコードで説明しているように、emp_eval 本体ペインで、式を変数 running_total に割り当てて、関数 calculate_score を変更します。新しいコードは太字で表示されます。

```
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;               -- from SCORES
  n_weight      performance_parts.weight%TYPE;   -- from PERFORMANCE_PARTS
  running_total NUMBER               :=0;        -- used in calculations
  max_score     CONSTANT scores.score%TYPE := 9; -- a constant limit check
  max_weight    CONSTANT performance_parts.weight%TYPE:= 1;
                                                 -- a constant limit check
BEGIN
  running_total := max_score * max_weight;
  RETURN running_total;
END calculate_score;
```

emp_eval 本体をコンパイルし、保存します。

参照：

- 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。

データベースからの値の割当て

最も簡単な値の割当て方法は、4-19 ページの「代入演算子を使用した値の割当て」で変数 running_total を割り当てたように、代入演算子（:=）を使用する方法です。

ただし、関数 calculate_score の目的は、データベース表に格納される値に基づいて計算することです。プロシージャ、ファンクションまたはパッケージにある既存のデータベースの値を使用するには、SELECT INTO 文の使用によって変数にこれらの値を割り当てる必要があります。後続の計算で変数を使用できます。

例 4-4 変数へのデータベースの値の割当て

次のコードで説明しているように、emp_eval 本体ペインで、変数 n_score および n_weight に表の値を割り当て、関数 calculate_score を変更します。その後、それらの積を running_total 変数へ割り当てます。新しいコードは太字で表示されます。

```
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;               -- from SCORES
  n_weight      performance_parts.weight%TYPE;   -- from PERFORMANCE_PARTS
  running_total NUMBER               := 0;       -- used in calculations
  max_score     CONSTANT scores.score%TYPE := 9; -- a constant limit check
  max_weight    CONSTANT performance_parts.weight%TYPE:= 1;
                                                 -- a constant limit check
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;
```

emp_eval 本体をコンパイルし、保存します。

同様に、employees 表の対応する行の内容に基づいて、evaluations 表へ新しいレコードをインサートするため新しい add_eval プロシージャを追加します。add_eval は順序 evaluations_seq を使用することに注意してください。

例 4-5 他の表からの値を使用した新しい表の行の作成

emp_eval 本体ペインの END emp_eval の上で、evaluations 表に行を挿入するために employees 表の一部の列を使用するプロシージャ add_eval を追加します。emp_eval パッケージの本体でローカル・ファンクション add_eval を作成しますが、パッケージ仕様部では宣言しないことに注意してください。これは、add_eval が他のサブプログラムによって emp_eval パッケージ内でのみ起動されるためです。

```
PROCEDURE add_eval(employee_id IN employees.employee_id%TYPE, today IN DATE) AS
  -- placeholders for variables
  job_id         employees.job_id%TYPE;
  manager_id     employees.manager_id%TYPE;
  department_id  employees.department_id%TYPE;
BEGIN

  -- extracting values from employees for later insertion into evaluations
  SELECT e.job_id INTO job_id FROM employees e
    WHERE employee_id = e.employee_id;
  SELECT e.manager_id INTO manager_id FROM employees e
    WHERE employee_id = e.employee_id;
  SELECT e.department_id INTO department_id FROM employees e
    WHERE employee_id = e.employee_id;

  -- inserting a new row of values into evaluations table
  INSERT INTO evaluations VALUES (
    evaluations_seq.NEXTVAL,   -- evaluation_id
    employee_id,               -- employee_id
    today,                     -- evaluation_date
    job_id,                    -- job_id
    manager_id,                -- manager_id
```

```
    department_id,              -- department_id
    0);                         -- total_score

END add_eval;
```

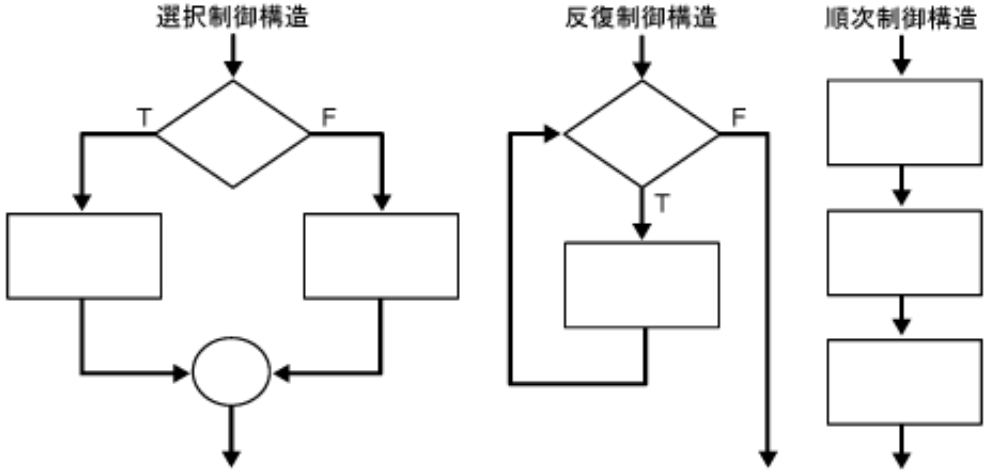
emp_eval 本体をコンパイルし、保存します。

参照：

- 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。

プログラム・フローの制御

制御構造は、SQL に対する PL/SQL 拡張機能の最も強力な機能です。これによって、データを操作し、条件付き選択、反復の制御および連続した文を使用して処理します。条件付き選択は、データ値の異なる型を持ち、異なるプロセスの手順を実行する必要がある状況です。反復の制御は、同一のデータで反復プロセスの手順を実行する必要がある状況です。また、一般的に、プログラムのコードのすべての行は連続で実行されます。代替のラベル付けプログラミングのブランチ（GOTO 文）を実行するために選択を行うのが、順次制御です。



この項では、条件付き選択、および IF...THEN...ELSE、CASE、FOR...LOOP、WHILE...LOOP や LOOP...EXIT WHEN などの反復プログラム・フロー構造について説明します。

参照：

- PL/SQL 制御構造の概要は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

条件付き選択制御の使用

条件付き選択構造は、BOOLEAN 値を TRUE または FALSE と評価する式をテストします。値に応じて、制御構造は割り当てられた一連の文を実行します。一般的な制御構造のメカニズムには、IF...THEN...ELSE およびその変数と、CASE 文の 2 種類があります。

参照：

- IF...THEN...ELSE 選択制御の詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。
- CASE...WHEN 選択制御の詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

IF...THEN...ELSE 選択制御の使用

IF...THEN...ELSE 文は条件付きで一連の文を実行します。テスト条件が TRUE と評価する場合、プログラムは THEN 句で文を実行します。この条件が FALSE と評価する場合、プログラムは ELSE 句で文を実行します。ELSIF キーワードを含めた場合、複数の条件のテストにこの構造を使用できます。IF...THEN...[ELSIF]...ELSE 文の一般的な書式は、次のとおりです。

```
IF condition_1 THEN
  ...;
ELSIF condition_2 THEN  -- optional
  ...;
ELSE                    -- optional
  ...;
END IF;
```

たとえば、サンプルの企業が雇用の最初の 10 年は 1 年に 2 回（12 月 31 日および 6 月 30 日）、ただしその後は 1 年に 1 回（12 月 31 日）、従業員評価を必要とするルールがあるとします。評価が 1 年ごとに何回行われるかを決定する eval_frequency 関数でこのルールを実装できます。これは、hire_date 列の値で IF...THEN...ELSE 句を使用して行われます。

関数 eval_frequency は評価が 1 年に 1 回行われるのか（10 年以上の雇用）、または 1 年に 2 回行われる必要があるかを決定する employees.hire_date を使用します。

emp_eval パッケージの本体で関数 eval_frequency を作成できますが、パッケージ仕様部では宣言しないことに注意してください。これは、eval_frequency が他のサブプログラムによって emp_eval パッケージ内でのみ起動されるためです。

例 4-6 IF...THEN...ELSE 選択制御の仕様

次のコードで説明しているように、emp_eval 本体ペインで、END emp_eval の直前に eval_frequency ファンクションを追加します。制御構造は太字で表示されます。

```
FUNCTION eval_frequency (employee_id IN employees.employee_id%TYPE)
  RETURN PLS_INTEGER AS
  hire_date   employees.hire_date%TYPE;     -- start of employment
  today       employees.hire_date%TYPE;     -- today's date
  eval_freq   PLS_INTEGER;                  -- frequency of evaluations
BEGIN
  SELECT SYSDATE INTO today FROM DUAL;      -- set today's date
  SELECT e.hire_date INTO hire_date         -- determine when employee started
      FROM employees e
      WHERE employee_id = e.employee_id;

   IF((hire_date + (INTERVAL '120' MONTH)) < today) THEN
     eval_freq := 1;
   ELSE
     eval_freq := 2;
   END IF;

   RETURN eval_freq;
 END eval_frequency;
```

emp_eval 本体をコンパイルし、保存します。

参照 :

- IF...THEN...ELSE 選択制御の詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

CASE...WHEN 選択制御の使用

CASE...WHEN 構造は、アクションのコースを決定する変数が多くの可能値を持つ場合、ネストされた IF...THEN 文の代替として便利です。CASE は条件を評価し、各可能値に対して異なるアクションを実行します。可能であれば、可読性および効率性のために IF...THEN のかわりに CASE...WHEN 文を使用します。CASE...WHEN 構造の一般的な書式は、次のとおりです。

```
CASE condition
  WHEN value_1 THEN expression_1;
  WHEN value_2 THEN expression_2;
  ...
  ELSE expression_default;
END CASE;
```

4-23 ページの「IF...THEN...ELSE 選択制御の使用」からの make_evaluation ファンクションで、1 つの役職で長年勤務した従業員に対して給与の値上げを検討する必要がある場合、hr ユーザーに通知すると想定します。employees.job_id の値に応じて、プログラム・ロジックは給与の値上げを推奨するユーザーに通知します。

『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』で説明されている DBMS_OUTPUT.PUT_LINE プロシージャを使用することに注意してください。

例 4-7 CASE...WHEN 条件付き制御の使用

次のコードで説明しているように、emp_eval 本体ペインで、eval_frequency ファンクションを編集して job_id 変数および job_id の値に基づく CASE 文を追加します。新しいコードは太字で表示されます。

```
FUNCTION eval_frequency (employee_id IN employees.employee_id%TYPE)
  RETURN PLS_INTEGER AS
  hire_date   employees.hire_date%TYPE;     -- start of employment
  today       employees.hire_date%TYPE;     -- today's date
  eval_freq   PLS_INTEGER;                  -- frequency of evaluations
  job_id      employees.job_id%TYPE;        -- category of the job

BEGIN
  SELECT SYSDATE INTO today FROM DUAL;    -- set today's date
  SELECT e.hire_date INTO hire_date          -- determine when employee started
      FROM employees e
      WHERE employee_id = e.employee_id;

  IF((hire_date + (INTERVAL '120' MONTH)) < today) THEN
     eval_freq := 1;

     /* Suggesting salary increase based on position */
     SELECT e.job_id INTO job_id FROM employees e
       WHERE employee_id = e.employee_id;
      CASE job_id
        WHEN 'PU_CLERK' THEN DBMS_OUTPUT.PUT_LINE(
          'Consider 8% salary increase for employee number ' || employee_id);
        WHEN 'SH_CLERK' THEN DBMS_OUTPUT.PUT_LINE(
          'Consider 7% salary increase for employee number ' || employee_id);
        WHEN 'ST_CLERK' THEN DBMS_OUTPUT.PUT_LINE(
          'Consider 6% salary increase for employee number ' || employee_id);
        WHEN 'HR_REP' THEN DBMS_OUTPUT.PUT_LINE(
          'Consider 5% salary increase for employee number ' || employee_id);
        WHEN 'PR_REP' THEN DBMS_OUTPUT.PUT_LINE(
          'Consider 5% salary increase for employee number ' || employee_id);
        WHEN 'MK_REP' THEN DBMS_OUTPUT.PUT_LINE(
          'Consider 4% salary increase for employee number ' || employee_id);
        ELSE DBMS_OUTPUT.PUT_LINE(
          'Nothing to do for employee #' || employee_id);
      END CASE;
```

```
  ELSE
    eval_freq := 2;
  END IF;

  RETURN eval_freq;
END eval_frequency;
```

emp_eval 本体をコンパイルし、保存します。

参照 :

- CASE...WHEN 選択制御の詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

反復制御の使用

反復構造またはループは何度も一連の文を実行します。ループの基本的な種類には、FOR...LOOP、WHILE...LOOP および LOOP...EXIT WHEN の 3 種類があります。

参照 :

- LOOP 反復の制御の詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

FOR...LOOP の使用

FOR...LOOP は、一連の手順の定義した回数を繰り返し、ループを実行する定義された整数の範囲である必要があるカウンタ変数を使用します。ループ・カウンタは、FOR...LOOP 文で暗黙的に宣言され、ループが実行されるたびに暗黙的に増加します。ループ・カウンタの値はループの本体内で使用できますが、プログラム上で変更はできないことに注意してください。FOR...LOOP 文は次の書式になります。

```
FOR counter IN integer_1..integer_2 LOOP
  ...
END LOOP;
```

4-24 ページの「CASE...WHEN 選択制御の使用」で説明されている、一部の従業員に対する給与の値上げの推奨に加えて、ファンクション eval_frequency は給与の値上げが継続される場合、従業員の給与が設定した年数でどう変化するかを出力します。

『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』で説明されている DBMS_OUTPUT.PUT プロシージャを使用することに注意してください。

例 4-8 FOR...LOOP 反復制御の使用

emp_eval 本体ペインで、推奨した年数に応じた給与を出力するために、CASE ブロックで割り当てられた推奨した給与の値上げ（sal_raise）を使用するよう eval_frequency ファンクションを編集します。この出力は、現在の給与で開始されます（salary）。新しいコードは太字で表示されます。

```
FUNCTION eval_frequency (employee_id IN employees.employee_id%TYPE)
  RETURN PLS_INTEGER AS
  hire_date   employees.hire_date%TYPE;     -- start of employment
  today       employees.hire_date%TYPE;     -- today's date
  eval_freq   PLS_INTEGER;                  -- frequency of evaluations
  job_id      employees.job_id%TYPE;        -- category of the job
  salary      employees.salary%TYPE;        -- current salary
  sal_raise   NUMBER(3,3) := 0;             -- proposed % salary increase

BEGIN
  SELECT SYSDATE INTO today FROM DUAL;    -- set today's date
  SELECT e.hire_date INTO hire_date          -- determine when employee started
      FROM employees e
      WHERE employee_id = e.employee_id;
```

```
 IF((hire_date + (INTERVAL '120' MONTH)) < today) THEN
    eval_freq := 1;

    /* Suggesting salary increase based on position */
    SELECT e.job_id INTO job_id FROM employees e
      WHERE employee_id = e.employee_id;
    SELECT e.salary INTO salary FROM employees e
      WHERE employee_id = e.employee_id;
     CASE job_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; -- job type does not match ones that should consider increases
     END CASE;

     /* If a salary raise is not zero, print the salary schedule */
     IF (sal_raise != 0) THEN -- start code for salary schedule printout
       BEGIN
         DBMS_OUTPUT.PUT_LINE('If the salary ' || salary || ' increases by ' ||
           ROUND((sal_raise * 100),0) ||
          '% each year over 5 years, it would be ');

         FOR loop_c IN 1..5 LOOP
           salary := salary * (1 + sal_raise);
           DBMS_OUTPUT.PUT (ROUND(salary,2) ||', ');
         END LOOP;

       DBMS_OUTPUT.PUT_LINE('in successive years.');

      END;
    END IF;

   ELSE
     eval_freq := 2;
   END IF;

   RETURN eval_freq;
 END eval_frequency;
```

emp_eval 本体をコンパイルします。

参照:

- LOOP 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。

WHILE...LOOP の使用

WHILE...LOOP は条件が TRUE であるかぎり繰り返します。条件は各ループの上部で評価し、TRUE の場合、ループの本体で文を実行します。条件が FALSE または NULL の場合は、制御がループの後の次の文に渡されます。WHILE...LOOP 制御構造の一般的な書式は次のとおりです。

```
WHILE condition LOOP
  ...
END LOOP;
```

WHILE...LOOP は無限に実行される場合があるため、注意して使用してください。

4-25 ページの「FOR...LOOP の使用」の EVAL_FREQUENCY ファンクションは、FOR...LOOP のかわりに WHILE...LOOP を使用し、推奨した給与が job_id の給与の上限に到達した後に終了します。

例 4-9 WHILE...LOOP 反復制御の使用

emp_eval 本体ペインで、推奨した年数応じた給与を出力するために、CASE ブロックで割り当てられた推奨した給与の値上げ（sal_raise）を使用し、job_id の最大レベルに到達したときに停止するように、eval_frequency ファンクションを編集します。新しいコードは太字で表示されます。

```
FUNCTION eval_frequency (employee_id IN employees.employee_id%TYPE)
  RETURN PLS_INTEGER AS
  hire_date   employees.hire_date%TYPE;     -- start of employment
  today       employees.hire_date%TYPE;     -- today's date
  eval_freq   PLS_INTEGER;                  -- frequency of evaluations
  job_id      employees.job_id%TYPE;        -- category of the job
  salary      employees.salary%TYPE;        -- current salary
  sal_raise   NUMBER(3,3) := 0;             -- proposed % salary increase
  sal_max     jobs.max_salary%TYPE;         -- maximum salary for a job

BEGIN
  SELECT SYSDATE INTO today FROM DUAL;    -- set today's date
  SELECT e.hire_date INTO hire_date          -- determine when employee started
      FROM employees e
      WHERE employee_id = e.employee_id;

  IF((hire_date + (INTERVAL '120' MONTH)) < today) THEN
     eval_freq := 1;

     /* Suggesting salary increase based on position */
     SELECT e.job_id INTO job_id FROM employees e
       WHERE employee_id = e.employee_id;
     SELECT e.salary INTO salary FROM employees e
       WHERE employee_id = e.employee_id;
     SELECT j.max_salary INTO sal_max FROM jobs j
       WHERE job_id = j.job_id;
      CASE job_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 a salary raise is not zero, print the salary schedule */
    IF (sal_raise != 0) THEN -- start code for salary schedule printout
      BEGIN
        DBMS_OUTPUT.PUT_LINE('If the salary ' || salary || ' increases by ' ||
          ROUND((sal_raise * 100),0) ||
          '% each year, it would be ');

        WHILE salary <= sal_max LOOP
          salary := salary * (1 + sal_raise);
          DBMS_OUTPUT.PUT (ROUND(salary,2) ||', ');
        END LOOP;

        DBMS_OUTPUT.PUT_LINE('in successive years.');
      END;
    END IF;
```

```
  ELSE
    eval_freq := 2;
  END IF;

  RETURN eval_freq;
END eval_frequency;
```

参照：

- WHILE... LOOP 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。

LOOP...EXIT WHEN の使用

LOOP...EXIT WHEN 構造を使用すると、移行の処理が必要ない場合にループを終了できます。EXIT WHEN 条件が TRUE と評価する場合、ループは終了し、制御が次の文に渡されます。

4-26 ページの「WHILE...LOOP の使用」の eval_frequency ファンクションは WHILE...LOOP を使用します。最後に計算された値は、最後のループの反復での給与に対する最大の可能値を超過する場合があることに注意してください（通常は超過します）。WHILE...LOOP のかわりに LOOP_EXIT WHEN 構造を使用する場合、ループを終了するための詳細な制御を利用できます。

例 4-10 LOOP...EXIT WHEN 反復制御の使用

emp_eval 本体ペインで、推奨した年数応じた給与を出力するために、CASE ブロックで割り当てられた推奨した給与の値上げ（sal_raise）を使用し、job_id の最大レベルに到達したときに停止するように、eval_frequency ファンクションを編集します。新しいコードは太字で表示されます。

```
FUNCTION eval_frequency (employee_id IN employees.employee_id%TYPE)
  RETURN PLS_INTEGER AS
  hire_date   employees.hire_date%TYPE;     -- start of employment
  today       employees.hire_date%TYPE;     -- today's date
  eval_freq   PLS_INTEGER;                  -- frequency of evaluations
  job_id      employees.job_id%TYPE;        -- category of the job
  salary      employees.salary%TYPE;        -- current salary
  sal_raise   NUMBER(3,3) := 0;             -- proposed % salary increase
  sal_max     jobs.max_salary%TYPE;         -- maximum salary for a job

BEGIN
  SELECT SYSDATE INTO today FROM DUAL;    -- set today's date
  SELECT e.hire_date INTO hire_date          -- determine when employee started
      FROM employees e
      WHERE employee_id = e.employee_id;

  IF((hire_date + (INTERVAL '120' MONTH)) < today) THEN
     eval_freq := 1;

     /* Suggesting salary increase based on position */
     SELECT e.job_id INTO job_id FROM employees e
       WHERE employee_id = e.employee_id;
     SELECT e.salary INTO salary FROM employees e
       WHERE employee_id = e.employee_id;
     SELECT j.max_salary INTO sal_max FROM jobs j
       WHERE job_id = j.job_id;
      CASE job_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 a salary raise is not zero, print the salary schedule */
  IF (sal_raise != 0) THEN -- start code for salary schedule printout
    BEGIN
      DBMS_OUTPUT.PUT_LINE('If the salary ' || salary || ' increases by ' ||
        ROUND((sal_raise * 100),0) ||
        '% each year, it would be ');

      LOOP
        salary := salary * (1 + sal_raise);
        EXIT WHEN salary > sal_max;
        DBMS_OUTPUT.PUT (ROUND(salary,2) ||', ');
      END LOOP;

      DBMS_OUTPUT.PUT_LINE('in successive years.');
    END;
  END IF;

 ELSE
   eval_freq := 2;
 END IF;

 RETURN eval_freq;
END eval_frequency;
```

参照：

- LOOP...EXIT WHEN 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。

コンポジット・データ構造（レコード）の使用

コンポジット・データ構造または**レコード**は、フィールドに格納される関連データ・アイテムのグループで、それぞれに名称およびデータ型が含まれます。レコードを表の行を含む変数、または表の行の一部の列として考えることもできます。このフィールドは表の列に対応しています。レコード構造は、実行時に異なる表から関連フィールドを効率的に使用するために、単一のパラメータとしてサブプログラムに関連アイテムを渡すのに非常に効果的です。

RECORD を型として定義し、点表記法でこのフィールドにアクセスする必要があります。レコードの定義および使用の一般的な書式は、次のとおりです。

```
TYPE record_name IS RECORD(                        -- define record type
 field_1 data_type,                                -- define fields in record
 ...
 field_n data_type);
...
variable_name record_name;                         -- define variable of new type
...
BEGIN
 ...
   ...variable_name.field1...;                     -- use fields of new variable
   ...variable_name.fieldn...;
 ...
END...;
```

4-28 ページの「LOOP...EXIT WHEN の使用」からの eval_frequency ファンクションで、様々な関連パラメータを使用しました。これらのアイテムの一部を組み合せて単一パラメータにする RECORD 構造を使用できます。

ジョブ仕様部の上限および下限を含む型を作成します。

RECORD 型を作成するには、次の手順を実行します。

1. 「Connections」ナビゲーション階層で、「Packages」の横にあるプラス記号（+）をクリックしてグループを展開します。
2. **EMP_EVAL** を右クリックします。
3. **「Edit」**を選択します。

   emp_eval ペインが表示されます。これは、emp_eval パッケージの仕様部を示します。
4. emp_eval パッケージ仕様部で、END emp_eval パッケージ仕様部の終了行の直前に、給与レベルの評価に必要なフィールドを含んだレコード型の定義 sal_info を入力します。

```
TYPE sal_info IS RECORD  -- type for salary, limits, raises, and adjustments
  ( job_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) );
```

5. emp_eval をコンパイルし、保存します。

   「Messages - Log」ペインに次のメッセージが表示されます。

```
EMP_EVAL Compiled
```

一度パッケージ仕様部で新しい RECORD 型を宣言すると、パッケージ本体内でこれを使用して、この型の変数を宣言できます。新しいプロシージャ salary_schedule を作成し、sal_info 型の変数を使用して eval_frequency ファンクションから起動できます。

PL/SQL のコンパイルは単一パス・プロセスで、サブプログラムがクライアント・サブプログラムの後に宣言される場合、PL/SQL コンパイラはエラーを発生させます。この状況を回避するには、パッケージ仕様部でまだ宣言していないサブプログラムすべてをパッケージ本体の上部で宣言します。サブプログラムの定義は、パッケージ本体内のあらゆる場所で可能です。ファンクション eval_frequency およびプロシージャ salary_schedule、add_eval の宣言方法の詳細は、次のタスクの手順 2 を参照してください。

RECORD 型を使用するには、次の手順を実行します。

1. 次のコードで説明しているように、emp_eval 本体ペインで、END emp_eval 文の直前に salary_schedule プロシージャの定義を追加します。このコードは、給与の値上げがゼロではない場合に実行する eval_frequency で BEGIN...END ブロックの内容と同様であることに注意してください。

```
PROCEDURE salary_schedule(emp IN sal_info) AS
  accumulating_sal NUMBER;          -- accumulator
  BEGIN
    DBMS_OUTPUT.PUT_LINE('If the salary of ' || emp.salary ||
         ' increases by ' || ROUND((emp.sal_raise * 100),0) ||
         '% each year, it would be ');
    accumulating_sal := emp.salary;  -- assign value of sal to accumulator
    WHILE  accumulating_sal <= emp.sal_max LOOP
       accumulating_sal :=  accumulating_sal * (1 + emp.sal_raise);
       DBMS_OUTPUT.PUT (ROUND( accumulating_sal,2) ||', ');
    END LOOP;
      DBMS_OUTPUT.PUT_LINE('in successive years.');
  END salary_schedule;
```

2. emp_eval 本体ペインで、emp_eval の本体の定義の上部に eval_frequency および salary_schedule の宣言を入力します。新しいコードは太字で表示されます。

```
create or replace
PACKAGE BODY emp_eval AS

/* local subprogram declarations */
FUNCTION eval_frequency (employee_id employees.employee_id%TYPE) RETURN NUMBER;
```

```
PROCEDURE salary_schedule(emp IN sal_info);
PROCEDURE add_eval(employee_id IN NUMBER, today IN DATE);

/* subprogram definition */
  PROCEDURE eval_department (dept_id IN NUMBER) AS
...
```

3. emp_eval 本体ペインで、eval_frequency ファンクションを編集して、変数 emp_sal として新しい sal_info 型を使用し、このフィールドを移入して、salary_schedule を起動します。給与の値上げがゼロではない場合に実行されたコードは、このファンクションの一部ではなく、salary_schedule プロシージャに組み込まれていることに注意してください。また、ファンクション上部の宣言は変更されたことにも注意してください。新しいコードは太字で表示されます。

```
FUNCTION eval_frequency (employee_id employees.employee_id%TYPE)
  RETURN PLS_INTEGER AS
  hire_date   employees.hire_date%TYPE;     -- start of employment
  today       employees.hire_date%TYPE;     -- today's date
  eval_freq   PLS_INTEGER;                  -- frequency of evaluations
  emp_sal     SAL_INFO;                     -- record for fields associated
                                            -- with salary review
BEGIN
  SELECT SYSDATE INTO today FROM DUAL;    -- set today's date
  SELECT e.hire_date INTO hire_date          -- determine when employee started
      FROM employees e
      WHERE employee_id = e.employee_id;

  IF((hire_date + (INTERVAL '120' MONTH)) < today) THEN
     eval_freq := 1;

     /* populate emp_sal */
     SELECT e.job_id INTO emp_sal.job_id FROM employees e
       WHERE employee_id = e.employee_id;
     SELECT j.min_salary INTO emp_sal.sal_min FROM jobs j
       WHERE emp_sal.job_id = j.job_id;
     SELECT j.max_salary INTO emp_sal.sal_max FROM jobs j
       WHERE emp_sal.job_id = j.job_id;
     SELECT e.salary INTO emp_sal.salary FROM employees e
       WHERE employee_id = e.employee_id;
     emp_sal.sal_raise := 0;  -- default

      CASE emp_sal.job_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 a salary raise is not zero, print the salary schedule */
    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;
```

4. emp_eval 本体をコンパイルし、保存します。

   「Messages - Log」ペインに次のメッセージが表示されます。

   ```
   EMP_EVAL Body Compiled
   ```

   参照：

   - コレクションおよびレコードの詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

カーソルおよびカーソル変数を使用したセットからのデータ取得

カーソルは、データベースへの問合せ、一連のレコードの取得（**結果セット**）、および開発者が一度に 1 行のこれらのレコードへアクセス可能にすることを目的として PL/SQL に組み込まれたポインタの型です。カーソルは、解析された文および関連情報を含むプライベートなメモリー内にある領域のハンドルまたは名称です。Oracle Database は暗黙的にカーソルを管理します。ただし、埋込み SQL 文を効果的に解析するためにプログラム内の名前付きリソースとして、カーソルを明示的に使用できる一部のインタフェースがあります。次は定義された 2 つの主要なカーソルの型です。

- **暗黙カーソル**は、カーソル自体を処理する暗黙コードなしに PL/SQL で使用できます。カーソルによって戻される結果セットはプログラム的に使用できますが、カーソル自体にプログラム的な制御はありません。
- **明示カーソル**を使用すると、カーソルをプログラム的に管理できます。また、結果セットでレコード・アクセスを制御する詳細レベルを使用できます。

各ユーザー・セッションには、初期化パラメータ OPEN_CURSORS によって設定された制限の数まで、多くのオープン・カーソルが含まれることがあります。この初期化パラメータはデフォルトで 50 に設定されています。アプリケーションがシステム・メモリーを保護するためにカーソルをクローズすることを確認する必要があります。OPEN_CURSORS の制限に到達し、カーソルがオープンできない場合は、データベース管理者に連絡をとって OPEN_CURSORS 初期化パラメータを変更します。

参照：

- カーソルの詳細は、『Oracle Database 概要』を参照してください。

明示的なカーソルの使用

FOR...LOOP などで、暗黙カーソルは明示カーソルよりもより効果的です。ただし、明示カーソルはプログラムにより適切で、使用すると名前付きリソースとして特定のメモリー内の領域を管理できます。

明示カーソルには次の表で説明される属性が含まれます。

カーソルの属性	説明
%NOTFOUND	最後のフェッチの結果に基づいて、TRUE または FALSE を戻します。
%FOUND	最後のフェッチの結果、%NOTFOUND 結果の否定に基づいて、TRUE または FALSE を戻します。
%ROWCOUNT	フェッチされた行数を戻します。最初のフェッチ以降の任意の時点にコールされます。また、UPDATE および DELETE 文から影響をうけた行数も戻します。
%ISOPEN	カーソルがまだオープンしている場合、TRUE を戻します。

明示カーソルはフェッチする列として同一タイプの変数として定義される必要があり、レコードのデータ型はカーソル定義から導出されます。また、オープンする必要があり、LOOP...EXIT WHEN 構造内で行を取得することがあります。カーソルを使用する場合の一般的な書式は次のとおりです。

```
DECLARE
  CURSOR cursor_name type IS query_definition;
OPEN cursor_name
  LOOP
    FETCH record;
    EXIT WHEN cursor_name%NOTFOUND;
    ...;              -- process fetched row
  END LOOP;
CLOSE cursor_name;
```

次はカーソルの存続期間内に起こります。

- OPEN 文はカーソルによって識別される問合せを解析し、入力をバインドして、結果セットからのレコードを正常にフェッチできたかを確認します。
- FETCH 文は問合せを実行した後、一致する行を検索し取得します。カーソルによって戻されるデータのバッファとしてローカル変数を定義し、使用する必要があり、その後、特定のレコードを処理します。
- CLOSE 文はカーソルの処理を完了し、カーソルをクローズします。一度カーソルがクローズされると、結果セットから追加のレコードを取得できないことに注意してください。

問合せと一致する各従業員レコードのカーソルを使用して 4-11 ページの「パッケージの作成」で宣言したプロシージャ eval_department を実行できます。

例 4-11 結果セットからの行を取得するカーソルの使用

カーソル emp_cursor は個々の結果セットからの行をフェッチします。各行の eval_frequency ファンクションの値および eval_department プロシージャが実行する年数に応じて、新しい評価レコードは add_eval プロシージャの起動によって従業員に対して作成されます。バッファ変数 emp_record は、%ROWTYPE として定義されます。

emp_eval パッケージ仕様部で、プロシージャ eval_department の宣言を編集します。

```
PROCEDURE eval_department(department_id IN employees.department_id%TYPE);
```

emp_eval 本体ペインで、eval_department プロシージャを編集します。

```
  PROCEDURE eval_department(department_id IN employees.department_id%TYPE) AS
   -- declaring buffer variables for cursor data
   emp_record      employees%ROWTYPE;
   -- declaring variable to monitor if all employees need evaluations
   all_evals       BOOLEAN;
   -- today's date
   today           DATE;
   -- declaring the cursor
   CURSOR emp_cursor IS SELECT * FROM employees e
          WHERE department_id = e.department_id;
BEGIN
-- determine if all evaluations must be done or just for newer employees;
-- this depends on time of the year
  today := SYSDATE;
  IF (EXTRACT(MONTH FROM today) < 6) THEN all_evals := FALSE;
  ELSE  all_evals := TRUE;
  END IF;

  OPEN emp_cursor;
```

```
  -- start creating employee evaluations in a specific department
  DBMS_OUTPUT.PUT_LINE('Determining evaluations necessary in department # ' ||
                       department_id);
  LOOP
    FETCH emp_cursor INTO emp_record;   -- getting specific record
    EXIT WHEN emp_cursor%NOTFOUND;      -- all records are been processed
    IF all_evals THEN
      add_eval(emp_record.employee_id, today); -- create evals for all
    ELSIF (eval_frequency(emp_record.employee_id) = 2) THEN
      add_eval(emp_record.employee_id, today); -- create evals; newer employees
    END IF;
  END LOOP;

  DBMS_OUTPUT.PUT_LINE('Processed ' || emp_cursor%ROWCOUNT || ' records.');

  CLOSE emp_cursor;

END eval_department;
```

emp_eval パッケージ仕様部、emp_eval 本体の順にコンパイルします。

「Messages - Log」ペインに次のメッセージが表示されます。

```
EMP_EVAL Body Compiled
```

参照 :

- カーソルの宣言の詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

カーソル変数の使用 : REF カーソル

カーソルは、カーソルを作成した問合せによって定義されるため静的です。場合によっては、問合せ自体が実行時に作成されます。REF CURSOR と呼ばれるカーソル変数は、特定の問合せから独立しているためにカーソルよりも柔軟性があります。カーソル変数は、問合せに対してオープンされ、結果セットを実行でき、同一の一連の列を戻す問合せに再使用できます。また、サブプログラム間の問合せの結果を渡すために REF CURSOR を適切にします。

REF CURSORS には、結果セットの書式を指定する戻り型を使用して宣言するもの（強いもの）と、あらゆる結果セットを取得するために戻り型を持たずに宣言するもの（弱いもの）があります。エラー発生の可能性が少なくなる戻り型を使用して REF CURSOR を宣言することをお薦めします。これは、正確に公式化された問合せと強力に関連付けられているためです。互換性のある型を関連付けることがあるより柔軟なカーソルを必要とする場合、事前定義の SYS_REFCURSOR を使用します。

REF CURSOR を使用する一般的な書式は次のとおりです。

```
DECLARE
  TYPE cursor_type IS REF CURSOR RETURN return_type;
  cursor_variable cursor_type;
  single_record return_type;
OPEN cursor_variable FOR query_definition;
  LOOP
    FETCH record;
    EXIT WHEN cursor_name%NOTFOUND;
    ...;             -- process fetched row
  END LOOP;
CLOSE cursor_name;
```

これは REF CURSOR およびカーソル変数の存続時間期間に起こります。

- REF CURSOR 型（戻り型）が宣言されます。
- カーソル型を一致するカーソル変数が宣言されます。
- 処理中の結果セットの個々の行の変数が宣言されます。この型は REF CURSOR 型定義の戻り型と同一である必要があります。
- OPEN 文はカーソル変数に対して問合せを解析します。
- ループ内の FETCH 文は問合せを実行し、移行の処理のため REF CURSOR の戻り型として同じ型のローカル変数に一致する行を取得します。
- CLOSE 文はカーソル・プロセスを完了し、REF CURSOR をクローズします。

4-32 ページの「明示的なカーソルの使用」で、プロシージャ eval_department は結果セットの取得、カーソルを使用しての処理、カーソルのクローズ、および終了を実行します。REF CURSOR 型としてカーソルを宣言する場合、カーソルの再使用によってより多くの部門（たとえば、3 つの連続した部門など）を変更します。

ループのフェッチは、入力としてカーソル変数を使用する新しい eval_fetch_control プロシージャの一部であることに注意してください。これは、問合せの定義からの結果セットの処理を区別する追加機能を持ちます。部門ベースではなく、企業内のすべての従業員への評価を開始するプロシージャ（eval_everyone）を記述します。

eval_department はプロシージャ add_eval をコールするレコードの単一フィールドを使用します。このプロシージャは、同一のレコード上で 3 つの異なる問合せを実行します。これは非常に非効率のため、REF CURSOR のレコード・バッファ全体を使用するために add_eval を再度記述します。

REF CURSOR を使用するには、次の手順を実行します。

1. emp_eval 仕様部で、REF CURSOR 型定義、emp_refcursor_type を追加します。この型はすべてのサブプログラムを表示するようにパッケージ・レベルで定義されます。また、プロシージャ eval_everyone の宣言を追加します。新しいコードは太字で表示されます。

```
create or replace
PACKAGE emp_eval AS
  PROCEDURE eval_department (department_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  -- type for salary, limits, raises, and adjustments
      ( job_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;
                           -- the REF CURSOR type for result set fetches
END emp_eval;
```

2. emp_eval 本体ペインで、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。新しいコードは太字で表示されます。

```
CREATE OR REPLACE PACKAGE BODY emp_eval AS
  /* local subprogram declarations */
  FUNCTION eval_frequency (employee_id IN employees.employee_id%TYPE)
    RETURN NUMBER;
  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);
...
```

3. emp_eval 本体ペインで、eval_department プロシージャを編集して部門に基づいた 3 つの異なる結果セットを取得し、eval_loop_control プロシージャをコールします。

```
PROCEDURE eval_department(department_id IN employees.department_id%TYPE) AS
  -- declaring the REF CURSOR
  emp_cursor      emp_refcursor_type;
 department_curr departments.department_id%TYPE;

BEGIN
  department_curr := department_id;   -- starting with the first department
  FOR loop_c IN 1..3 LOOP
    OPEN emp_cursor FOR
      SELECT *
      FROM employees e
      WHERE department_curr = e.department_id;
    -- create employee evaluations is specific departments
    DBMS_OUTPUT.PUT_LINE('Determining necessary evaluations in department #' ||
          department_curr);
    eval_loop_control(emp_cursor);  -- call to process the result set
    DBMS_OUTPUT.PUT_LINE('Processed ' || emp_cursor%ROWCOUNT || ' records.');
    CLOSE emp_cursor;
    department_curr := department_curr + 10;
  END LOOP;
END eval_department;
```

4. emp_eval 本体ペインで、add_eval プロシージャを編集して employee_id のかわりに、取得された employee%ROWTYPE のレコードを使用します。プロシージャの最初の部分ではあらゆる宣言が必要ないことに注意してください。

```
PROCEDURE add_eval(emp_record IN employees%ROWTYPE, today IN DATE) AS
BEGIN
-- inserting a new row of values into evaluations table
  INSERT INTO evaluations VALUES (
    evaluations_seq.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. emp_eval 本体ペインでコードの最後に対して、eval_loop_control プロシージャを追加して結果セットからの個々のレコードをフェッチし、処理します。このコードの多くは、4-32 ページの「明示的なカーソルの使用」の eval_department プロシージャの初期の定義からのものです。新しい構造は太字で表示されています。

```
PROCEDURE eval_loop_control(emp_cursor IN emp_refcursor_type) AS
    -- declaring buffer variable for cursor data
   emp_record      employees%ROWTYPE;
   -- declaring variable to monitor if all employees need evaluations
   all_evals       BOOLEAN;
   -- today's date
   today           DATE;
BEGIN
  -- determine if all evaluations must be done or just for newer employees;
 -- this depends on time of the year
  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;   -- getting specific record
    EXIT WHEN emp_cursor%NOTFOUND;      -- all records are been processed
    IF all_evals THEN
      add_eval(emp_record, today); -- create evaluations for all
    ELSIF (eval_frequency(emp_record.employee_id) = 2) THEN
      add_eval(emp_record, today); -- create evaluations for newer employees
    END IF;
  END LOOP;
END eval_loop_control;
```

6. emp_eval 本体ペインで、企業内のすべての従業員を含む結果セットを取得する eval_everyone プロシージャを追加します。このコードは、手順 3 のプロシージャ eval_department と同様であることに注意してください。

```
PROCEDURE eval_everyone AS
   -- declaring the REF CURSOR type
  emp_cursor emp_refcursor_type;
   BEGIN
    OPEN emp_cursor FOR SELECT * FROM employees;
    -- start creating employee evaluations in a specific department
    DBMS_OUTPUT.PUT_LINE('Determining the number of necessary evaluations');
    eval_loop_control(emp_cursor); -- call to process the result set
  DBMS_OUTPUT.PUT_LINE('Processed ' || emp_cursor%ROWCOUNT || ' records.');
  CLOSE emp_cursor;
 END eval_everyone;
```

7. emp_eval ペインで、emp_eval 仕様部をコンパイルし、保存します。

   「Messages - Log」ペインに次のメッセージが表示されます。

```
EMP_EVAL Compiled
```

8. emp_eval 本体ペインで、emp_eval 本体をコンパイルし、保存します。

   「Messages - Log」ペインに次のメッセージが表示されます。

```
EMP_EVAL Body Compiled
```

参照 :

- カーソル変数の構文の詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。
- カーソル属性の構文の詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

コレクション（索引付き表）の使用

PL/SQL で使用可能な別のグループのユーザー定義のデータ型は、Oracle バージョンの 1 次元配列である**コレクション**です。コレクションは、1 つの変数内に複数の行のデータを保持できるデータ構造です。様々な型のデータを持つ行を 1 行のみ保持するレコードとは対照的に、コレクションのデータはすべて同じ型である必要があります。他の開発言語では、コレクションで表される型の型は**配列**と呼ばれます。

コレクションは、情報のリストを保持するために使用されます。また、コレクションの要素には直接アクセスすることができるため、アプリケーションのパフォーマンスを大幅に向上させることができます。コレクションの構造には、索引付き表、ネストした表および可変配列の 3 つの型があります。

- **索引付き表**は、PL/SQL プログラム内での使用に対して、最も柔軟性に富み、一般的にパフォーマンスが最も優れたコレクションの型です。
- **ネストした表**は、アプリケーションによって部分的に格納および取得される大きなコレクションに有効です。
- **VARRAY** は、アプリケーションによって全体が格納および取得される小さなコレクションに有効です。

ここでは、索引付き表に限定して説明します。

索引付き表は、連想配列またはキーと値のペア・セットとも呼ばれます。各キーは一意で、配列内の対応する値を検出するために使用されます。このキー（または索引）には、整数または文字列のいずれかを使用できます。

連想配列は、任意のサイズのデータ・セットを表します。これによって、配列内の相対位置の確認およびすべての配列要素のループを実行せずに、個々の要素にアクセスできるようになります。

データ検索用の単純な一時記憶域の場合は、連想配列によって、SQL 表に必要なディスク領域およびネットワーク操作を使用せずにデータをメモリーに格納できます。連想配列は、データの永続記憶域ではなく一時記憶域を対象とするため、INSERT、SELECT INTO などの SQL 文は使用できません。ただし、パッケージに型を宣言してパッケージ本体に値を割り当てることで、データベース・セッションが存続している間、連想配列を永続的にすることができます。

初めてキーを使用して値を割り当てる際、そのキーが連想配列に追加されます。同じキーを使用する後続の割当てでは、同じエントリが更新されます。データベース表の主キー、有効な数値ハッシュ関数の結果または一意の文字列値を形成する文字列の連結などの一意のキーを選択します。

索引付き表を宣言する前に、表の型を決定する必要があります。この項の後半では、アプリケーションの一部として索引付き表を使用する方法について説明します。

（PLS_INTEGER および VARCHAR2 で索引付けされた）2 つの型の連想配列を効率的に実装するには、次の手順を実行します。

- カーソルを定義します。
- カーソルの ROWTYPE または TYPE を使用して、索引付き表の構造を定義します。
- BULK COLLECT を使用して、カーソル・データを索引付き表にフェッチします。
- 索引付き表を反復し、特定の要素の索引を使用して値を検索します。

索引付き表へのカーソルの作成

データを索引付き表にフェッチするカーソルを定義し、カーソルの要素型を使用して索引付き表を作成することをお薦めします。例 4-12 は、hr.employees 表からデータをフェッチする employees_jobs_cursor および hr.jobs 表からデータをフェッチする jobs_cursor の 2 つのカーソルの作成方法を示しています。2 つ目のカーソルでは ORDER BY 句が使用されていないことに注意してください。

例 4-12 索引付き表へのカーソルの宣言

```
CURSOR employees_jobs_cursor IS
  SELECT e.first_name, e.last_name, e.job_id
  FROM hr.employees e
  ORDER BY e.job_id, e.last_name, e.first_name;

CURSOR jobs_cursor IS
  SELECT j.job_id, j.job_title
  FROM hr.jobs j;
```

索引付き表の定義

カーソルを宣言した後、例 4-13 に示すように、%ROWTYPE 属性を使用して索引付き PLS_INTEGER 表 employees_jobs および jobs を作成できます。

例 4-13 カーソル構造に基づいた索引付き PLS_INTEGER 表の作成

```
TYPE employees_jobs_type IS TABLE OF employees_jobs_cursor%ROWTYPE
  INDEX BY PLS_INTEGER;
employees_jobs employees_jobs_type;

TYPE jobs_type IS TABLE OF jobs_cursor%ROWTYPE
  INDEX BY PLS_INTEGER;
jobs jobs_type;
```

job_id の job_titles 索引付き表などの VARCHAR2 で索引付けされた表を作成するには、例 4-14 に示すように、元の表 hr.jobs から VARCHAR2 型の定義を使用します。

例 4-14 索引付き VARCHAR2 表の作成

```
TYPE job_titles_type IS TABLE OF hr.jobs.job_title%TYPE
  INDEX BY hr.jobs.job_id%TYPE;
job_titles job_titles_type;
```

索引付き PLS_INTEGER 表 BULK COLLECT への移入

ローカル PL/SQL 変数として大量のデータを参照する必要がある場合は、1 度に 1 行ずつ結果セットをループするより、BULK COLLECT 句を使用する方がはるかに効率的です。一部の列のみを問い合せる場合は、すべての結果を別々のコレクション変数の各列に格納できます。表のすべての列を問い合せる場合は、結果セット全体をレコードのコレクションに格納できます。

例 4-15 に示すように、索引付き PLS_INTEGER 表 employees_jobs および jobs を使用すると、カーソルをオープンし、BULK COLLECT を使用してデータを取得できます。

例 4-15 BULK COLLECT を使用した索引付き PLS_INTEGER 表への移入

```
OPEN employees_jobs_cursor;
FETCH employees_jobs_cursor BULK COLLECT INTO employees_jobs;
CLOSE employees_jobs_cursor;

OPEN jobs_cursor;
FETCH jobs_cursor BULK COLLECT INTO jobs;
CLOSE jobs_cursor;
```

索引付き VARCHAR2 表の作成

jobs 表にデータを含めた後、例 4-16 に示すように、FOR ... LOOP を使用して索引付き VARCHAR2 表 job_titles を作成します。

例 4-16 索引付き VARCHAR2 表の作成

```
FOR i IN 1..jobs.COUNT() LOOP
  job_titles(jobs(i).job_id) := jobs(i).job_title;
END LOOP;
```

索引付き表の反復

構造 employees_jobs は、PLS_INTEGER で索引付けされているため、稠密な索引付き表となっています。例 4-17 に示すように、索引付き表は、表の 1 から COUNT() 値をカウントする FOR ... LOOP 内に操作を配置することで簡単に反復できます。太字の行は、job_titles 表内の値の直接検索を表しています。

例 4-17 索引付き PLS_INTEGER 表の反復

```
FOR i IN 1..employees_jobs.count() LOOP
  DBMS_OUTPUT.PUT_LINE(
    RPAD(employees_jobs(i).employee_id, 10)||
    RPAD(employees_jobs(i).first_name, 15)||
    RPAD(employees_jobs(i).last_name,  15)||
   job_titles(employees(i).job_id));
  END LOOP;
```

構造 job_titles は、VARCHAR2 で索引付けされたスパースな索引付き表です。例 4-18 に示すように、この索引付き表は、最初のキー値と等しい事前定義済カウンタおよび表の NEXT() 値を使用して、WHILE ... END LOOP 内で反復できます。要素は、索引の字句順でソートされます。

例 4-18 索引付き VARCHAR2 表の反復

```
DECLARE i hr.jobs.job_id%TYPE := job_titles.FIRST();
WHILE i IS NOT NULL LOOP
  DBMS_OUTPUT.PUT_LINE(
    RPAD(job_titles(i).job_id, 10)||
    job_titles(i).job_title);
  i := job_titles.NEXT(i);
END LOOP;
```

エラーおよび例外の処理

例外のエラー条件は、PL/SQL コードを使用して簡単に検出および処理します。エラー発生時、通常の処理を停止し、例外処理コードの制御を送信することで、例外が発生します。このコードは PL/SQL ブロックの最後に位置されます。PL/SQL では、固有の例外ハンドラを持つそれぞれの例外を使用し、エラー・ルーチンへのコールおよびチェックが自動的に実行されます。

変数またはデータベース処理を含む特定の共通するエラー条件に対して事前定義済の例外が自動的に発生します。ユーザーのプログラムに関連するエラーである条件に対して、または既存の Oracle メッセージに対するラッパーとして、カスタム例外を宣言することもできます。

参照：

- 例外の詳細は、『Oracle Database 概要』を参照してください。
- 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。
- 標準の Oracle メッセージのリストの詳細は、『Oracle Database エラー・メッセージ』を参照してください。
- エラーおよび例外処理のガイドラインの詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。
- PL/SQL 例外の利点の詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

既存の PL/SQL および SQL 例外

Oracle Database は、PL/SQL プログラムが既知のデータベース・ルールを違反する場合、例外を自動的に発生します。この既知のデータベース・ルールは、SELECT INTO 文が行を戻さない場合の事前定義済の例外 NO_DATA_FOUND などです。次の表は、共通の例外の一部を示したものです。

例外	説明
ACCESS_INTO_NULL	プログラムは初期化されていないオブジェクトの属性に対して値を割り当てようと試みます。
CASE_NOT_FOUND	CASE 文の WHEN 句で何も選択していない場合、ELSE 句はありません。
COLLECTION_IS_NULL	プログラムは初期化されていないネストされた表または VARRAY に対して EXISTS 以外のコレクション・メソッドを適用しようと試みるか、初期化されていないネストされた表または VARRAY の要素に値を割り当てようと試みます。
CURSOR_ALREADY_OPEN	プログラムはすでにオープンになっているカーソルをオープンしようと試みます。カーソルは再オープンされる前にクローズされる必要があります。カーソル FOR ループは参照するカーソルを自動的にオープンするため、プログラムはループ内でこのカーソルをオープンできません。
DUP_VAL_ON_INDEX	プログラムは一意の索引によって制約されている列にある重複した値を格納しようと試みます。
INVALID_CURSOR	プログラムはオープンしていないカーソルのクローズなどの許可されていないカーソル操作を試みます。
INVALID_NUMBER	SQL 文では、文字列には有効な数字が表示されないために文字列の数値への変換は失敗します（プロシージャ文では、VALUE_ERROR が発生します）。また、この例外は長い FETCH 文の LIMIT 句式が正数を評価しない際にも発生します。
LOGIN_DENIED	プログラムは有効ではないユーザー名またはパスワードを使用して Oracle データベースにログインしようと試みます。
NO_DATA_FOUND	SELECT INTO 文は行を戻しません。あるいはユーザーのプログラムがネストされた表で削除された要素、または index-by 表の初期化されていない要素を参照します。 これは、この例外が完了を通知する一部の SQL ファンクションによって内部使用され、問合せの一部としてコールされるファンクション内でこの例外を発生させる場合に伝播されるこの例外に依存しないためです。
NOT_LOGGED_ON	プログラムは Oracle データベースに接続することなくデータベース・コールを発行します。
ROWTYPE_MISMATCH	割当てに含まれたホスト・カーソル変数および PL/SQL カーソル変数には互換性のない戻り型が含まれます。格納されたサブプログラムへ渡されたホスト・カーソル変数をオープンする際、実パラメータおよび仮パラメータが互換性のある必要があります。
SUBSCRIPT_BEYOND_COUNT	プログラムはコレクションの要素数より大きい索引番号を使用してネストした表または VARRAY 要素を参照します。
SUBSCRIPT_OUTSIDE_LIMIT	プログラムは有効範囲外にある索引番号（たとえば、-1 など）を使用してネストした表または VARRAY 要素を参照します。
TOO_MANY_ROWS	SELECT INTO 文は 2 行以上を戻します。
VALUE_ERROR	計算、転換、切捨てまたはサイズ制限エラーが発生します。たとえば、プログラムが文字変数に対して列値を選択した際に値が宣言した変数の長さよりも長い場合、PL/SQL は割当てをキャンセルし、VALUE_ERROR が発生します。これは文字列の数字への変換が失敗する場合にプロシージャ文で VALUE_ERROR が発生します（SQL 文では INVALID_NUMBER が発生します）。
ZERO_DIVIDE	プログラムはゼロで数字を除算しようと試みます。

例 4-19 例外処理

emp_eval 本体ペインで、問合せが結果セットを戻さないケースを処理するために eval_department プロシージャを編集します。新しいコードは太字で表示されます。

```
  PROCEDURE eval_department(department_id IN employees.department_id%TYPE) AS
   -- declaring the REF CURSOR
   emp_cursor      emp_refcursor_type;
   department_curr departments.department_id%TYPE;

  BEGIN
    department_curr := department_id;   -- starting with the first department
    FOR loop_c IN 1..3 LOOP
      OPEN emp_cursor FOR
        SELECT *
          FROM employees e
          WHERE department_curr = e.department_id;
     -- create employee evaluations is specific departments
        DBMS_OUTPUT.PUT_LINE('Determining necessary evaluations in department #' ||
          department_curr);
        eval_loop_control(emp_cursor);  -- call to process the result set
        DBMS_OUTPUT.PUT_LINE('Processed ' || emp_cursor%ROWCOUNT || ' records.');
      CLOSE emp_cursor;
      department_curr := department_curr + 10;
    END LOOP;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE ('The query did not return a result set');
END eval_department;
```

emp_eval 本体をコンパイルし、保存します。

カスタム例外

パッケージにはエラーを処理するカスタム例外が含まれる場合があります。例外はプログラムで宣言され、また、あらゆる宣言リージョンで、サブプログラム、パッケージ本体またはパッケージ仕様部によってどのように使用されるかに応じます。

例外宣言は次の形式を含みます。

```
exception_name EXCEPTION;
```

正しくない値に基づいて、カスタム例外をプログラム的に発生するには、次の形式を使用する必要があります。

```
IF condition THEN
  RAISE exception_name;
```

予期せぬ Oracle エラーをトラップするには、一般的にサブプログラムまたはパッケージの本体内の最後のブロックとしてコード内に指示を処理する例外を含める必要があります。（標準およびカスタムの両方を）処理する例外に特定の名前を付け、予期せぬエラーをトラップする OTHERS ハンドラを使用する必要があります。例外本体は次の形式を含みます。

```
EXCEPTION
  WHEN exception_name_1 THEN
    ...;
    DBMS_OUTPUT.PUT_LINE(message_1);
  ...
  WHEN OTHERS THEN
    ...
    DBMS_OUTPUT.PUT_LINE(message_others);
```

または、例外の発生後に実行が継続するようにプログラムを設計できます。例外ハンドラを使用して BEGIN...END ブロックの例外を生成したと思われるコードを囲む必要があります。たとえば、ループ構造内の例外をトラップするコードはエラーを生成する要素のエラーを処理し、次のループの反復を使用して継続できます。

次の作業では、2 つの考えられる例外 weight_wrong と score_wrong を宣言し、これらの例外を発生させた後、トラップするようにファンクション calculate_score を再設計します。

例 4-20 カスタム例外の処理

emp_eval 本体ペインで、calculate_score ファンクションを編集します。新しいコードは太字で表示されます。

```
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;               -- from SCORES
    n_weight      performance_parts.weight%TYPE;   -- from PERFORMANCE_PARTS
    running_total NUMBER               := 0;       -- used in calculations
    max_score     CONSTANT scores.score%TYPE := 9; -- a constant limit check
    max_weight    CONSTANT performance_parts.weight%TYPE:= 1;
                                                   -- a constant limit check
  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                              -- check that weight is valid
      IF n_weight > max_weight OR n_weight < 0 THEN
        RAISE weight_wrong;
      END IF;
    END;
    BEGIN                               -- check that score is valid
      IF n_score > max_score OR n_score < 0 THEN
        RAISE score_wrong;
      END IF;
    END;
    -- calculate the score
    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;
```

emp_eval 本体をコンパイルし、保存します。

参照：

- 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。

5

トリガーの使用

この章では、データベースの表、ビューまたはイベントと関連付けられたストアド・プロシージャのコードであるデータベース・トリガーについて説明します。

この章の内容は次のとおりです。

トリガーの設計

トリガーはデータベースで指定したイベントが発生した際に、自動的に起動されるストアド・プロシージャのコードです。トリガーは表、ビューまたはイベントと関連付けられます。プロシージャおよびファンクションとは異なり、トリガーは直接的に起動できません。そのかわり、Oracle Database はユーザーまたはアプリケーションに関係なくトリガーされたイベントが発生したときに、トリガーを暗示的に起動します。トリガーを起動したイベントが失敗した場合、オペレーションが正常に処理されないエラーを発生させないかぎり、トリガーが活動していることに気づかないでしょう。

トリガーを正しく使用することで、より効率的なデータベースを使用でき、より堅固でセキュアなアプリケーションを構築およびデプロイできます。これらの利点は、トリガーが次の機能を提供することで実現可能になります。

- データの整合性のチェックおよび実行
- 監査およびロギング
- 複雑なビジネス・ロジックのモデリング
- トランザクションの妥当性のチェックおよび実行
- 派生列の生成
- 表の変更の有効化および制限

データベースに固有で、そのためにすべてのクライアント・アプリケーションに共通の低いレベルのビジネス・ルールを強化するトリガーを使用できます。たとえば、hr スキーマの employees 表にアクセスするいくつかのクライアント・アプリケーションがあると思います。この表に対するトリガーが、表に追加されるすべてのデータが正しい形式であることを確認する場合、このビジネス・ロジックはすべてのクライアント・アプリケーションで再生産およびメンテナンスを行う必要がありません。これは、トリガーがクライアント・アプリケーションによって回避できず、トリガーに格納されたビジネス・ロジックが自動的に使用されるためです。

各トリガーには次の一般的な書式が含まれます。

```
TRIGGER trigger_name
  triggering_statement
   [trigger_restriction]
BEGIN
 triggered_action;
END;
```

トリガーには 4 つの主要な部分があります。

- **トリガー名**は、同一のスキーマの他のトリガーに関して一意である必要があります。トリガー名は他のスキーマ・オブジェクト（表、ビューおよびプロシージャ）に関して一意である必要はありませんが、混乱を防ぐために一貫したネーミング規則を採用することをお薦めします。
- **トリガーを実行する文**は、トリガーの起動を開始するイベントです。これらのイベントには、表およびビューの DML 文（INSERT、UPDATE および DELETE）、スキーマ・オブジェクトの DDL 文（CREATE、ALTER および DROP）、システム・エラー、データベースの起動および終了、また他のシステム・アクションが含まれます。トリガーを実行する文はトリガーの制限の影響を受けます。
- **トリガーの制限**は、トリガーに置かれる制限です。これは、制限が TRUE と評価する場合のみ、データベースはトリガーされたアクションを実行するという意味です。
- **トリガーされたアクション**は、トリガーの本体、または適切な文がトリガーを起動し、制限が TRUE と評価する際に両方に実行される一連の手順です。

参照：

- トリガーの一般情報は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

トリガーのタイプ

Oracle Database のトリガーには 5 つの異なるタイプがあります。

- **文トリガー**は特定の表またはビューの DELETE、INSERT または UPDATE などの DML 文と関連付けられています。

  文トリガーは各 DML 文に対して一度起動します。たとえば、UPDATE 文トリガーは、表の影響を受ける行数に関係なく一度のみ実行します。

  Oracle Database リリース 11*g* R1 から、特定の DML 文と関連付けられたいくつかの異なるトリガーを所持できます。CREATE TRIGGER 文の FOLLOWS および PRECEDES 句を使用して、実行される順序を指定できます。

- **行トリガー**は、表の INSERT、UPDATE または DELETE 文に影響を受ける各行に対して起動します。

  行トリガーは文トリガーとして同一の方法で機能しますが、2 つの追加の指定を使用します。行トリガーはトリガー中の文で FOR EACH ROW 句を使用します。また、行トリガーによって、行の値を参照し、トリガーの本体でイベントをトリガーに設定できます。これは特に、デフォルト値の挿入および無効な値の上書きに便利です。

- 命令文を発行するかわりに、ビューで **INSTEAD OF** トリガーを実行します。INSERT 文がビューで使用されている場合、INSTEAD OF トリガーを使用して、実表または他の表へのデータの挿入、データの挿入が不要な挿入要求のロギングなどの実際に発生する処理の密な制御を実行できます。

  また、Oracle Database はビューに対して発行された挿入の処理ができない場合があります。生成された列の場合、値を正確に決定するトリガーを作成できます。たとえば、ビューが列定義 last_name || ', ' || first_name を使用していた場合、last_name 列中でカンマ文字の前の文字および first_name 列中でカンマ文字の後の文字を更新する、INSTEAD OF トリガーが記述されている可能性があります。

- **ユーザー・イベント・トリガー**は、CREATE、ALTER、DROP などの DDL 文、ユーザー LOGON および LOGOFF、特定の DML アクション（分析および統計、監査、付与および権限の取消しなど）で使用できます。ユーザーがデータベースへ接続した時に起動する LOGON トリガーは、ユーザーに対して環境を設定し、セキュアなアプリケーション・ロールと関連付けられたファンクションを実行するために一般的に使用されます。

- **システム・イベント・トリガー**はデータベースの起動、データベースの終了またはサーバー・エラー・イベントに適用されます。これらのイベントは特定の表、ビューまたは行には関連付けられません。

参照：

- CREATE TRIGGER 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。
- 「文トリガーの作成」（5-5 ページ）
- 「行トリガーの作成」（5-6 ページ）
- 「INSTEAD OF トリガーの作成」（5-8 ページ）
- 「LOGON トリガーおよび LOGOFF トリガーの作成」（5-8 ページ）

トリガーのタイミング

トリガーはトリガー中の文で BEFORE または AFTER 句を使用できます。BEFORE および AFTER で、トリガーを起動するイベントの前または後のどちらにトリガーを実行するかを指定します。文および行トリガーでは、BEFORE トリガーはセキュリティを強化でき、データベースへの変更を行う前にビジネス・ルールを有効にします。一方、AFTER トリガーはロギング・アクションに非常に役立ちます。

INSTEAD OF トリガーは BEFORE または AFTER オプションを使用しません。デフォルトでは、これらのオプションは AFTER の行レベルのトリガーとして同一のセマンティクスを使用します。

システムおよびユーザー・イベント・トリガーでは、明らかな例外を除いて、BEFORE 句および AFTER 句を使用できます。これらの例外は、AFTER が STARTUP、SUSPEND および LOGON に有効な場合、および BEFORE が SHUTDOWN および LOGOFF に有効な場合のみです。

参照:

- 『Oracle Database SQL 言語リファレンス』

トリガー設計のガイドラインおよび制限

アプリケーションのトリガーを計画する際に次のガイドラインおよび制限を検討する必要があります。

- トリガーは、データベースのカスタマイズに役立ちますが、必要な場合のみ使用してください。トリガーを過剰に使用すると相互依存関係が複雑になる可能性があり、大規模なアプリケーションでは管理が困難になります。
- アクションの実行時に関連オブジェクトおよび依存アクションすべてが実行されることを確認します。
- システム・メモリーを急速に使い果してしまうため、再帰的トリガーの使用は避けます。
- 意図しない効果およびパフォーマンスへの影響を実感する場合は、カスケード・トリガーを確認します。
- 既存の Oracle Database の提供を複製するトリガーの使用は避けてください。たとえば、宣言整合性制約を介して削除される無効なデータを拒否するトリガーなどは設計しないでください。
- ビジネス・ルールを効率的に実装するように BEFORE 句および AFTER 句を正確に使用していることを確認します。BEFORE EACH ROW トリガーは :NEW 値を変更できます。
- 32KB を超えないようにトリガーのサイズを制限します。トリガーが多くのコードの行を必要とする場合、トリガーから起動されるストアド・プロシージャへのビジネス・ロジックの移行を検討してください。
- 特定のユーザーまたはクライアント・アプリケーションに関係なく、作成したトリガーをエンタープライズ全体に適切なデータベースおよびビジネス・ロジックへ適用させることを確認します。特別なルールを一部のユーザーおよびクライアント・アプリケーションに対してのみ適用する場合、アプリケーション内のそのビジネス・ロジックをカプセル化します。
- トリガー内では COMMIT、ROLLBACK または SAVEPOINT を使用できません。これは、DDL 文に暗黙的 COMMIT が含まれ、DDL 文もトリガーでは使用できないためです。システム・トリガーの CREATE、ALTER、DROP TABLE、ALTER...COMPILE は、除きます。
- コミット済のシステム・トリガーのみ起動されます。

トリガーの作成および使用

この項では、様々なタイプのトリガーの作成および使用方法を示します。

この項には、次のトピックが含まれます。

- 文トリガーの作成
- 行トリガーの作成
- INSTEAD OF トリガーの作成
- LOGON トリガーおよび LOGOFF トリガーの作成
- トリガーの変更
- トリガーの無効化および有効化
- トリガーのコンパイル
- トリガーの削除

参照 :

- トリガーの作成の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。

文トリガーの作成

文トリガーは、INSERT、UPDATE または DELETE などの特定の文に関連しています。これらは特定の表で実行されるので、これらの操作のロギングに対して文トリガーを使用できます。

例 5-1 は、ログ表の作成方法を示しています。

例 5-1 EVALUATIONS 表のログ表の作成

evaluations_log 表には evaluations 表で各 INSERT、UPDATE または DELETE を持つエントリが格納されます。

```
CREATE TABLE evaluations_log (log_date DATE
                            , action VARCHAR2(50));
```

例 5-2 では、evaluations 表が変更されるたびに evaluations_log に書き込むトリガーを作成します。

例 5-2 文トリガーおよび述語を使用した操作のロギング

トリガー eval_change_trigger は、evaluations 表に対するすべての変更を追跡し、これらの変更が行われた後に新しい行を追加することによって evaluations_log 表で変更を追跡します。この例では、使用可能な 3 つの文からトリガーを起動する文を決定するために、トリガーの本体で**条件述語** INSERTING、UPDATING または DELETING が使用されていることに注意してください。

```
CREATE OR REPLACE TRIGGER eval_modification_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;
```

行トリガーの作成

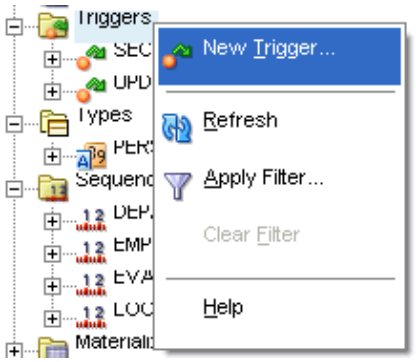
行トリガーは影響を受けるそれぞれの行に対して実行されます。

3-25 ページの「順序の使用」で、evaluations_seq 順序を evaluations 表に対する主キー番号ジェネレータとして作成しました。Oracle Database では、CREATE TABLE 文の一部として、主キーは自動的に作成されません。かわりに、INSERT 文を使用して主キーに対する一意の番号を生成するトリガーを設計する必要があります。

次のタスクでは、SQL Developer Connection ナビゲーション階層を使用して、トリガー new_evaluation を作成します。このトリガーは、同一の従業員の行が同一の期間に存在するかどうかに基づいて、新しい表を evaluations 表に追加する必要があるかどうかをチェックします。

例 5-3 主キー FOR EACH ROW トリガーである BEFORE オプションの生成

1. 「Connections」ナビゲーション階層で、「Triggers」を右クリックします。
2. ドロップダウン・リストから「**New Trigger**」を選択します。

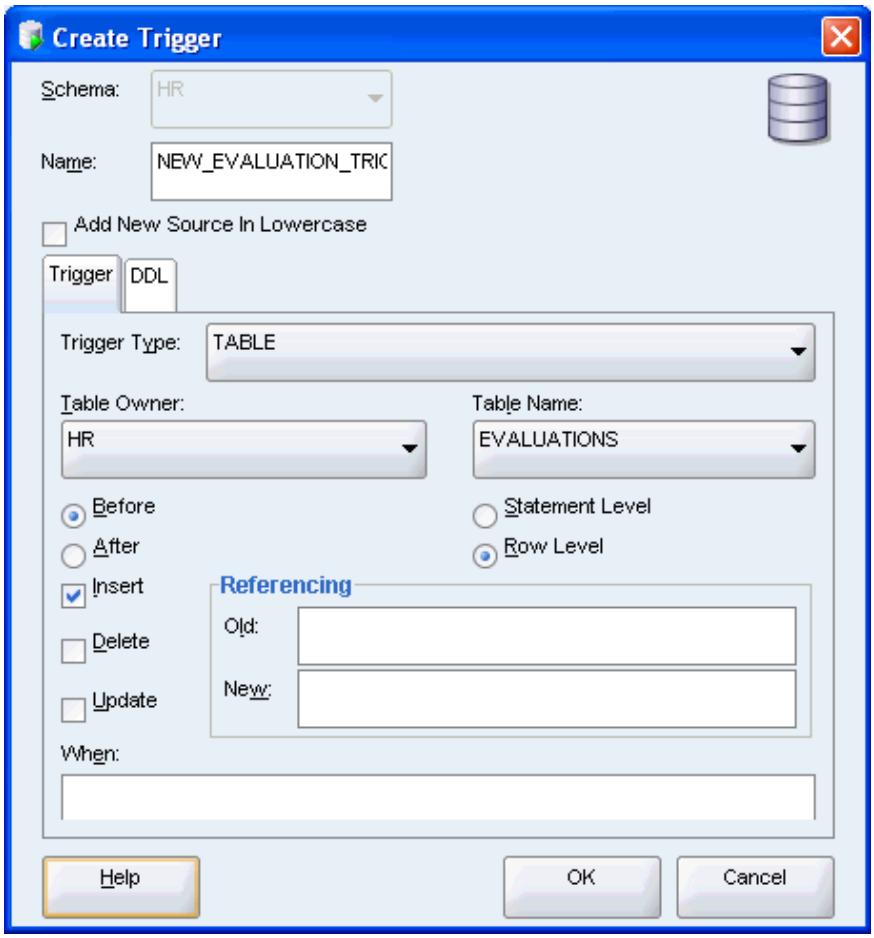


3. 「Create Trigger」ウィンドウで、次のパラメータを設定します。

   - 「**Name**」を `new_evaluation_trigger` に設定します。

   「Trigger」ペインで、「**Trigger Type**」を `TABLE` に、次に「**Table Owner**」を `HR` に、そして「**Table Name**」を `evaluations` に設定し、「**Before**」、「**Insert**」および「**Row Level**」の順で選択します。

   「**OK**」をクリックします。



4. `new_evaluation` ペインを次のコードを使用してオープンします。

   ペインのタイルがイタリック・フォントになっていることに注意してください。トリガー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。

```
CREATE OR REPLACE
TRIGGER new_evaluation
  BEFORE INSERT ON evaluations
  FOR EACH ROW
BEGIN
  NULL;
END;
```

5. 「**File**」メニューから「**Save**」を選択して新規のトリガーを保存します。または **[Ctrl] を押しながら [S]** キーを押します。

   Oracle Database により、トリガーは保存前に自動的にコンパイルされます。

INSTEAD OF トリガーの作成

INSTEAD OF トリガーでは、表示の基礎となる表に対する変更を実装できます。このようなトリガーは「ビューの作成」で作成した emp_locations ビューで使用されます。emp_locations の定義を確認します。

```
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;
```

例 5-4 では、INSTEAD OF トリガー update_name_view_trigger を実装して、従業員の名前を更新します。

例 5-4 INSTEAD OF トリガーを使用したビューからの値の更新

```
CREATE OR REPLACE TRIGGER update_name_view_trigger
INSTEAD OF UPDATE ON emp_locations
BEGIN
-- allow only the following update(s)
  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;
```

LOGON トリガーおよび LOGOFF トリガーの作成

LOGON および LOGOFF トリガーは、ログ表へ書き込むことで、データベースの使用者を監視します。

例 5-5 では、LOGON および LOGOFF イベントの追跡を続行するために hr_users_log 表を作成します。その後、これらのイベントをログ表に書き込むために、トリガー note_hr_logon_trigger（例 5-6）および note_hr_logoff_trigger（例 5-7）を作成します。

例 5-5 アクセス・ログ表、hr_users_log の作成

この表は hr スキーマでのすべてのログオンおよびログイン・イベントのログです。

```
CREATE TABLE hr_users_log (user_name VARCHAR2(30), activity VARCHAR2(20),
                           event_date DATE);
```

例 5-6 LOGON トリガーの作成

hr スキーマに接続すると、このトリガーにより LOGON イベント・レコードが hr_users_log 表に挿入されます。これは AFTER トリガーであることに注意してください。

```
CREATE OR REPLACE TRIGGER note_hr_logon_trigger
  AFTER LOGON
  ON HR.SCHEMA
BEGIN
  INSERT INTO hr_users_log VALUES (USER, 'LOGON', SYSDATE);
END;
```

例 5-7 LOGOFF トリガーの作成

hr スキーマへの接続が切断されると、このトリガーにより LOGOFF イベント・レポートが hr_users_log 表に挿入されます。これは BEFORE トリガーであることに注意してください。

```
CREATE OR REPLACE TRIGGER note_hr_logoff_trigger
  BEFORE LOGOFF
  ON HR.SCHEMA
BEGIN
  INSERT INTO hr_users_log VALUES (USER, 'LOGOFF', SYSDATE);
END;
```

トリガーの変更

new_evaluation_trigger には空の本体があります。

例 5-8 は、evaluations_seq 順序の次の使用可能な値を evaluation_id に割り当てるトリガーの変更方法を示しています。

例 5-8 トリガーの変更

次のコードを使用して、new_evaluation_trigger を置換します。新規コードは太字フォントです。

```
CREATE OR REPLACE TRIGGER new_evaluation_trigger
  BEFORE INSERT ON evaluations FOR EACH ROW
BEGIN
  :NEW.evaluation_id := evaluations_seq.NEXTVAL;
END;
```

トリガーの無効化および有効化

トリガーが参照するオブジェクトが使用不可能である場合またはトリガーが原因の遅延なしに大規模なデータのアップロードを実行する必要がある場合（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。

トリガーを無効にするには、ALTER TRIGGER ... DISABLE 文を使用する必要があります。トリガーを再度有効にするには、ALTER TRIGGER ... ENABLE 文を使用します。

例 5-9 は、トリガーを一時的に無効にする方法を示しています。

例 5-9 トリガーの無効化

```
ALTER TRIGGER eval_change_trigger DISABLE;
```

例 5-10 は、トリガーを再度有効にする方法を示しています。

例 5-10 トリガーの有効化

```
ALTER TRIGGER eval_change_trigger ENABLE;
```

特定の表のすべてのトリガーを無効にする必要がある場合は、ALTER TABLE ... DISABLE ALL TRIGGERS 文を使用する必要があります。表のすべてのトリガーを再度有効にするには、ALTER TABLE ... ENABLE ALL TRIGGERS 文を使用します。

例 5-11 は、特定の表で定義されたすべてのトリガーを一時的に無効にする方法を示しています。

例 5-11 表のすべてのトリガーの無効化

```
ALTER TABLE evaluations DISABLE ALL TRIGGERS;
```

例 5-12 は、特定の表で定義されたすべてのトリガーを再度有効にする方法を示しています。

例 5-12 表のすべてのトリガーの有効化

```
ALTER TABLE evaluations ENABLE ALL TRIGGERS;
```

参照 :

- トリガーの有効化の詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。
- トリガーの無効化の詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

トリガーのコンパイル

CREATE TRIGGER 文が実行されると、トリガーは完全にコンパイルされます。トリガーのコンパイルでエラーが生じた場合、DML 文は失敗します。関連したコンパイル・エラーを表示するには、USER_ERRORS ビューを使用します。

例 5-13 に、データベースに存在するトリガー・エラーを判断する方法を示します。

例 5-13 トリガー・コンパイル・エラーの表示

```
SELECT * FROM USER_ERRORS WHERE TYPE = 'TRIGGER';
```

トリガーがコンパイルされると、基礎となるデータベース・オブジェクトに依存性を作成し、トリガーとオブジェクトが一致しないようにこれらのオブジェクトが削除または変更された場合はトリガーが無効になります。無効なトリガーは次の起動時に再コンパイルされます。

例 5-14 は、データベース内の他のオブジェクトにあるトリガー依存性を確認する方法を示しています。

例 5-14 トリガー依存性の表示

```
SELECT * FROM ALL_DEPENDENCIES WHERE TYPE = 'TRIGGER';
```

手動でトリガーを再コンパイルするには、例 5-15 に示すように ALTER TRIGGER ... COMPILE 文を使用する必要があります。

例 5-15 トリガー・コンパイル・エラーの表示

```
ALTER TRIGGER update_name_view_trigger COMPILE;
```

参照 :

- トリガーのコンパイルの詳細は、『Oracle Database PL/SQL 言語リファレンス』を参照してください。

トリガーの削除

トリガーを削除する必要がある場合は、例 5-16 に示すように DROP TRIGGER 文を使用します。

例 5-16 トリガーの削除

```
DROP TRIGGER eval_change_trigger;
```

トリガーの削除後、アプリケーションに必要のない依存オブジェクトを削除できます。

参照 :

- DROP TRIGGER 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。

6

グローバル環境での作業

この章では、グローバリゼーション・サポート環境でのアプリケーションの開発方法について説明し、SQL および PL/SQL 両方を使用して Unicode プログラミングの使用方法を示します。Unicode プログラミングによって、複数の言語と互換性のある SQL および PL/SQL コードを記述することができます。

この章の内容は次のとおりです。

参照 :

- グローバリゼーション・サポート環境の設定を含む、Oracle Database を使用したグローバリゼーション・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。

グローバリゼーションの概要

Oracle Database のグローバリゼーション・サポートにより、各国語でデータを格納、処理、取得できます。さらにデータベース・ユーティリティ、エラー・メッセージ、ソート順序、日付、時刻、通貨、数字、カレンダ規則が、自動的に固有の言語およびロケールに調整されます。

グローバリゼーション・サポートには各国語サポート（NLS）機能が含まれます。各国語サポートは、各国の言語を選択し、データを特定のキャラクタ・セットで格納する機能です。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アクセスし、実行できる多言語アプリケーションおよびソフトウェア製品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。

参照 :

- グローバリゼーション・サポート環境の設定を含む、Oracle Database を使用したグローバリゼーション・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。

グローバリゼーション・サポート機能

Oracle Database の標準機能には次のものが含まれます。

- **言語サポート :** この機能により、データを各国語で格納、処理、取得できます。Unicode データベースおよびデータ型を使用することにより、Oracle Database では現代言語のほとんどがサポートされています。6-8 ページの「NLS パラメータの設定」を参照してください。
- **地域サポート** : この機能では地域に固有の文化的慣習がサポートされています。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。6-9 ページの「言語および地域パラメータの設定」を参照してください。
- **日付と時刻の書式** : この機能では時間、日、月および年を表示するためのローカル書式がサポートされています。タイム・ゾーンおよび夏時間もサポートされています。6-12 ページの「日付および時刻パラメータの設定」を参照してください。
- **数値と通貨の書式** : この機能では通貨、貸方および借方の記号および数字を表すためのローカル書式がサポートされています。6-18 ページの「通貨パラメータの使用」および 6-17 ページの「数値書式の使用」を参照してください。
- **カレンダ機能 :** この機能ではグレゴリオ暦、日本の元号暦、台湾暦、タイ仏教暦、ペルシャ暦、英国版イスラム暦およびイスラム暦の 7 つの暦法をサポートしています。6-15 ページの「カレンダ定義の設定」を参照してください。
- **言語ソート :** この機能では文化に応じた正確なソートおよび大 / 小文字の変換を行うために、言語の定義がサポートされています。6-21 ページの「言語ソートおよび検索の使用」を参照してください。
- **キャラクタ・セット・サポート :** この機能では多数のシングルバイト、マルチバイトおよび固定幅のコード体系をサポートしています。これらのコード体系は、各国の規格、国際規格およびベンダー固有の規格に基づいています。Oracle Database でサポートしているプラットフォームのキャラクタ・セットのリストは Oracle データベースのインストレーション・ガイドを参照してください。
- **キャラクタ・セマンティクス :** この機能ではキャラクタ・セマンティクスをサポートしています。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記憶要件をバイト数ではなく文字数で定義する場合に役立ちます。6-25 ページの「長さセマンティクスの使用」を参照してください。
- **Unicode サポート** : この機能ではエンコードされたユニバーサル・キャラクタ・セットである Unicode をサポートしています。このセットを使用すると、1 つのキャラクタ・セットを使用して任意の言語の情報を格納できます。SQL および PL/SQL を使用して Unicode データを挿入および取得できます。6-27 ページの「グローバルなアプリケーション開発」を参照してください。

参照：

- グローバリゼーション・サポート環境の設定を含む、Oracle Database を使用したグローバリゼーション・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。

現在の NLS パラメータ値の表示

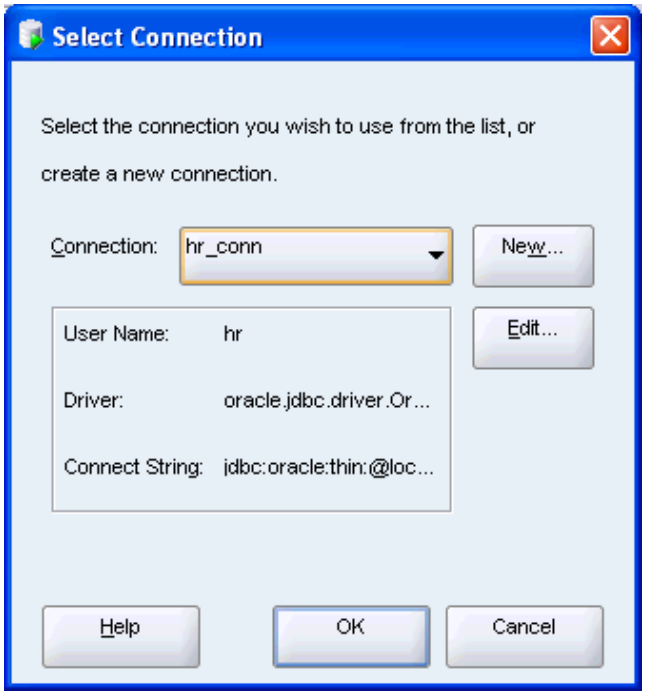
SQL Developer では各国語サポート・パラメータ・レポートによりグローバリゼーション・サポートのパラメータ値がリストされます。

このレポートの NLS パラメータ値は、6-5 ページの「SQL Developer 環境における NLS パラメータ値の使用」で説明しているように、SQL Developer 内のすべてのセッションの最初に使用します。

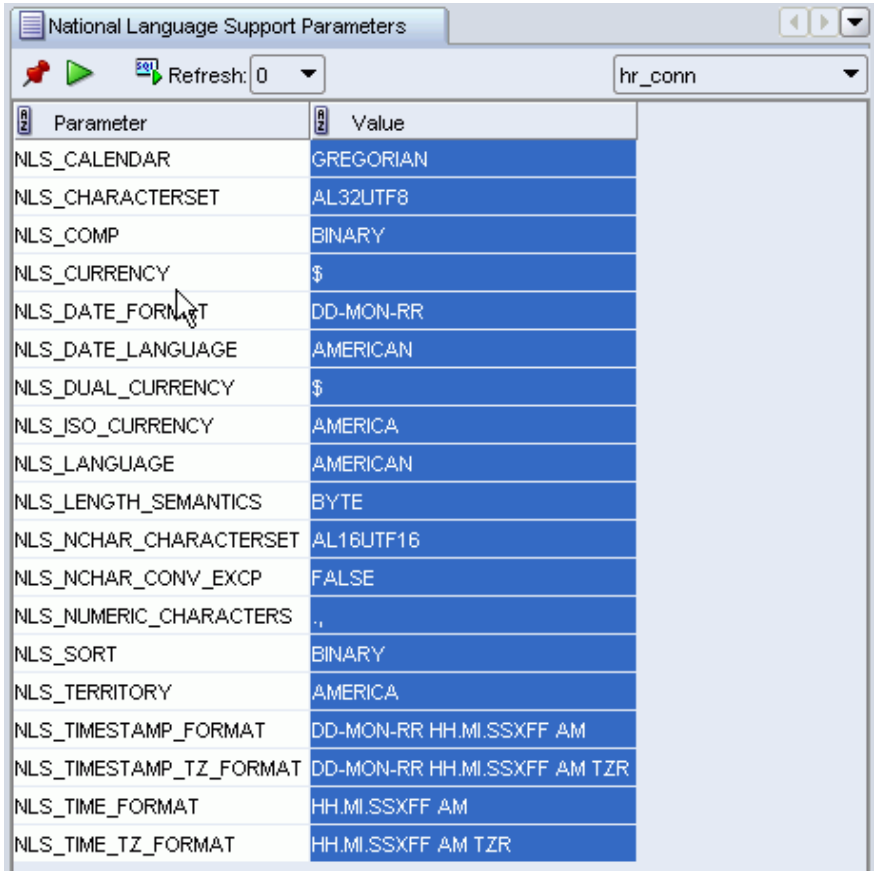
各国語サポート・パラメータ・レポートを表示するには、次の手順を実行します。

1. SQL Developer ウィンドウで、**「Reports」**タブをクリックして「Reports」ナビゲータを表示します。
2. **「Data Dictionary Reports」**ノードの横にあるプラス記号（+）をクリックし、展開します。
3. **「About Your Database」**ノードの横にあるプラス記号（+）をクリックし、展開します。
4. **「National Language Support Parameters」**をクリックします。
5. 「Select Connection」ダイアログ・ボックスで、**「Connection」**を `hr_conn` に設定します。

   「OK」をクリックします。



6. 「National Language Support Parameters」ペインのレポートには、選択した接続に関連付けられたデータベースに対する NLS パラメータの現在の値が表示されます。この値は、NLS_CALENDAR、NLS_CHARSET、NLS_COMP、NLS_CURRENCY、NLS_DATE_FORMAT などです。



参照：

- 「SQL Developer 環境における NLS パラメータ値の使用」(6-5 ページ)
- 現行のデータベース・インスタンスの NLS 設定を表示する V$NLS_PARAMETERS ビューの詳細は、『Oracle Database リファレンス』を参照してください。
- NLS パラメータを含む SQL Developer プリファレンスの詳細は、『Oracle Database SQL Developer ユーザーズ・ガイド』を参照してください。
- SQL Developer レポートの詳細は、『Oracle Database SQL Developer ユーザーズ・ガイド』を参照してください。
- グローバリゼーション・サポート環境の設定を含む、Oracle Database を使用したグローバリゼーション・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。

SQL Developer 環境における NLS パラメータ値の使用

Oracle データベースでは、NLS パラメータ値がデータベース初期化パラメータによって最初に決定されます。DBA は初期化ファイルの値を設定でき、その変更は次のデータベースの設定に影響します。データベース・ユーザーは、次の文を使用することで現在のセッション（データベースへの現在の接続）に対する特定のパラメータ値を変更できます。

```
ALTER SESSION SET parameter-name = "value";
```

SQL Developer を使用するときは、データベース初期化ファイルのパラメータ値は使用されないことに注意してください。かわりに、SQL Developer では次を含むパラメータ値が最初（インストール後）に使用されます。

```
NLS_LANG,"AMERICAN"
NLS_TERR,"AMERICA"
NLS_CHAR,"AL32UTF8"
NLS_SORT,"BINARY"
NLS_CAL,"GREGORIAN"
NLS_DATE_LANG,"AMERICAN"
NLS_DATE_FORM,"DD-MON-RR"
```

SQL Developer（すべての接続における「SQL Worksheet」ウィンドウや各国語サポート・パラメータ・レポートなど）のすべてのセッションで使用されるこれらの NLS パラメータ値およびその他の NLS パラメータ値は、NLS パラメータの「Preferences」ペインで表示できます。

NLS パラメータの値を変更するには、次のオプションが必要です。

- 6-6 ページの「すべてのセッションに対する NLS パラメータ値の変更」で説明されているように、NLS パラメータの「Preferences」ペインを使用して、（現在および今後の）すべての SQL Developer 接続で使用する値を変更します。
- 「SQL Worksheet」ウィンドウで `ALTER SESSION` 文を使用することで、現在の接続のみの値を変更します。

このように、データベース・アプリケーション開発時に、異なる言語設定をテストすることに大きな柔軟性を得られます。たとえば、`ALTER SESSION` を使用して、異なる言語設定を使用する次の PL/SQL 文の出力を表示し、接続を切断し再接続することで SQL Developer のデフォルト設定に戻すことができます。

たとえば、プリファレンスの `NLS_LANGUAGE` 値が `FRENCH` に設定されており、今日が 2007 年 3 月 1 日であると仮定します。「SQL Worksheet」で `SELECT sysdate FROM dual;` と入力し、**「Run Script」**アイコンをクリックすると、次のように出力されます。

```
SYSDATE
-------------------------
01-MARS -07
```

`ALTER SESSION SET NLS_LANGUAGE='AMERICAN';` と入力し、前述の `SELECT` 文を入力すると、次のように出力されます。

```
SYSDATE
-------------------------
01-MAR-07
```

この例を継続し、現在の接続から切断し、再接続すると、セッションの NLS_LANGUAGE 値は（プリファレンスで指定したとおり）FRENCH となり、SELECT 文による出力は次のようになります。

```
SYSDATE
-------------------------
01-MARS -07
```

参照：

- NLS パラメータを含む SQL Developer プリファレンスの詳細は、『Oracle Database SQL Developer ユーザーズ・ガイド』を参照してください。
- グローバリゼーション・サポート環境の設定を含む、Oracle Database を使用したグローバリゼーション・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。

すべてのセッションに対する NLS パラメータ値の変更

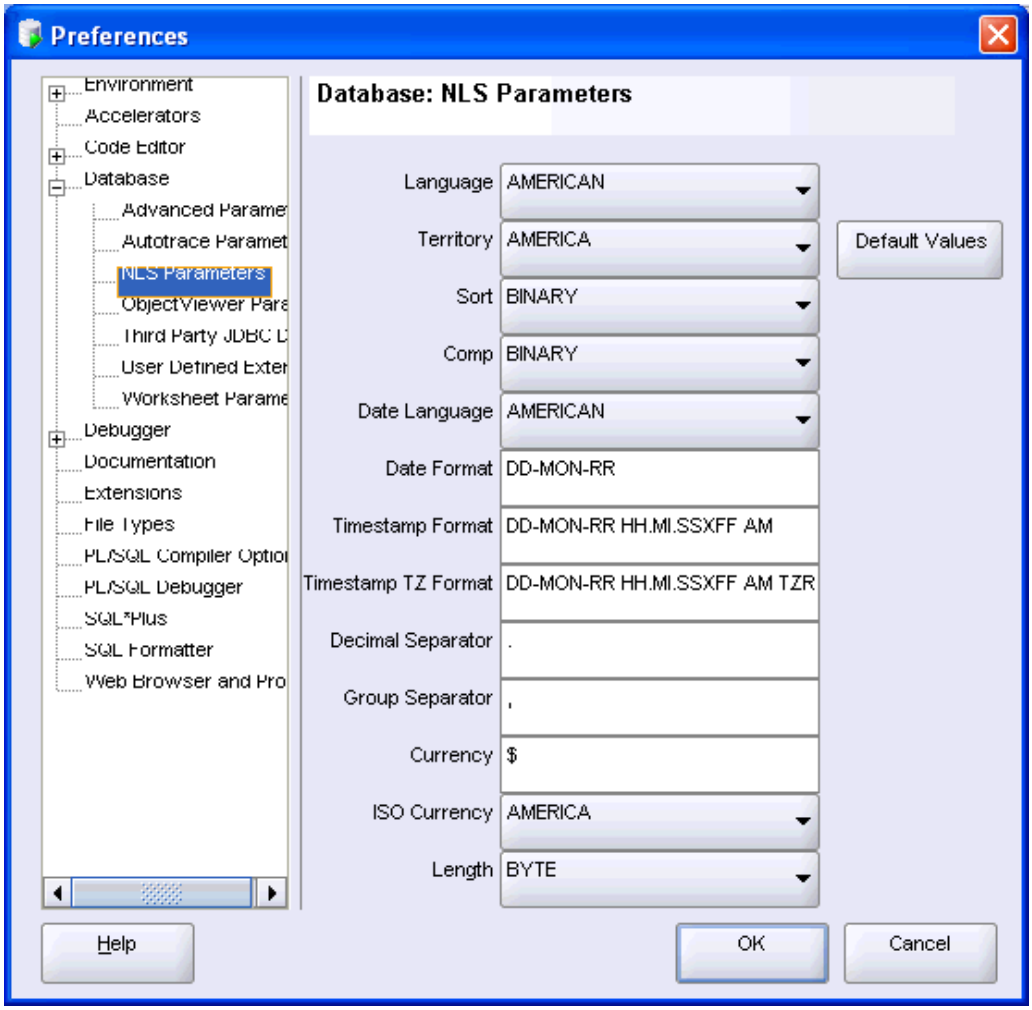
NLS パラメータに対する SQL Developer のユーザー・プリファレンスにより、（現在および今後の）すべての SQL Developer 接続で使用される値が設定されます。データベースの NLS パラメータの「Preferences」ペインでパラメータ値を表示し、変更できます。

6-3 ページの「現在の NLS パラメータ値の表示」で説明しているように、これらのプリファレンスは NLS パラメータ値レポートでも表示されます。

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 ウィンドウで、**「Tools」**をクリックしてから**「Preferences」**をクリックします。

2. 「Preferences」ダイアログ・ボックスで、「**Database**」ノードを展開し、「**NLS Parameters**」を選択します。



各テキスト・ラベルは対応する `NLS_xxx` パラメータを説明する用語です。

3. 目的の変更をパラメータ値に対して行います。

たとえば、スペイン語環境を反映するように `NLS_LANGUAGE` パラメータ値を変更するには、「**Language**」で `SPANISH` を選択します。

4. 「**OK**」をクリックします。

参照 :

- NLS パラメータを含む SQL Developer プリファレンスの詳細は、『Oracle Database SQL Developer ユーザーズ・ガイド』を参照してください。
- グローバリゼーション・サポート環境の設定を含む、Oracle Database を使用したグローバリゼーション・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。

グローバリゼーション・サポート環境の設定

この項では、グローバリゼーション・サポート環境の設定方法を説明します。

参照:

- グローバリゼーション・サポート環境の設定を含む、Oracle Database を使用したグローバリゼーション・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。

NLS_LANG パラメータを使用したロケールの選択

ロケールとはシステムやプログラムを実行する言語的および文化的環境のことです。Oracle ソフトウェアでロケール動作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。このパラメータによって、クライアント・アプリケーションおよびデータベースで使用される言語と地域が設定されます。また、クライアント・プログラムによって入力または表示されるデータのキャラクタ・セットであるクライアントのキャラクタ・セットも設定されます。

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）およびクライアント・アプリケーション（Oracle ツールの表示の書式など）の両方で使用する言語および地域環境が設定されます。

インストール時に定義したデフォルトの NLS_LANG 動作はほとんどの状況において適切ですが、セッション時に動的に NLS 環境を変更することもあります。この場合、ALTER SESSION 文を使用することで、NLS_LANGUAGE、NLS_TERRITORY およびその他の NLS パラメータを変更できます。

ALTER SESSION 文を使用してクライアント・キャラクタ・セットの設定を変更できないことに注意してください。ALTER SESSION 文はセッション環境のみを変更します。クライアントが新しい設定を獲得せず、ローカル環境を変更しない場合、ローカル・クライアント NLS 環境は変更されません。

参照:

- グローバリゼーション・サポート環境の設定を含む、Oracle Database を使用したグローバリゼーション・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。

NLS パラメータの設定

各国語サポート（NLS）パラメータにより、クライアントおよびサーバー両方でのロケール固有の動作が決定します。NLS パラメータは様々な方法で指定できます。SQL ファンクションでユーザー・セッションのパラメータを変更し、パラメータを上書きできます。

NLS パラメータの設定は、次の 2 つの方法で変更できます。

- ALTER SESSION 文で NLS パラメータを設定し、初期化パラメータ・ファイルのセッションに対して設定されたデフォルト値、または環境変数を使用してクライアントで設定されたデフォルト値を上書きします。

  ```
  ALTER SESSION SET NLS_SORT = french;
  ```

  ALTER SESSION 文を使用して行った変更は、現在のユーザー・セッションにのみ適用され、次にログインしたときには変更が適用されません。

- SQL ファンクション内で NLS パラメータを使用して、初期化パラメータ・ファイルのセッションに対して設定されたデフォルト値、環境変数を使用してクライアントで設定されたデフォルト値、または ALTER SESSION 文でセッションに対して設定されたデフォルト値を上書きします。たとえば、次を実行します。

  ```
  TO_CHAR(hiredate,'DD/MON/YYYY','nls_date_language = FRENCH')
  ```

クライアントでの NLS 環境変数の使用を含む NLS パラメータを設定する他の方法は、プラットフォーム依存である場合があります。これは、クライアントに対してロケール依存の動作を指定、または初期化パラメータ・ファイルのセッションのデフォルト値の設定を上書きします。たとえば、Linux システムの場合は次のようになります。

```
% setenv NLS_SORT FRENCH
```

参照 :

- 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』の NLS パラメータの設定に関する項を参照してください。
- ALTER SESSION 文の詳細は、『Oracle Database SQL 言語リファレンス』を参照してください。
- TO_CHAR ファンクションを含む 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。
- 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。
- グローバリゼーション・サポートで使用する初期化パラメータの詳細は、『Oracle Database リファレンス』を参照してください。

言語および地域パラメータの設定

ローカル地域に応じて異なる 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 と定義されている場合であっても、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の円（JPY）を設定できます。

参照 :

- サポートされる言語および地域を含むロケール情報の詳細は、『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_LANG パラメータを使用したロケールの選択」（6-8 ページ）

NLS_LANGUAGE パラメータの使用

NLS_LANGUAGE パラメータは有効な言語名に対して設定できます。デフォルトは NLS_LANG 設定から導出されます。NLS_LANGUAGE により、次のセッション特性に対するデフォルトの規則が指定されます。

- サーバー・メッセージの言語
- 曜日名、月名の言語およびその略称（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 に相当する記号
- ORDER BY 句が指定されている場合の文字データのデフォルトのソート順序（ORDER BY を指定しないかぎり、GROUP BY 句ではバイナリ・ソート順序が使用されます）

NLS_LANGUAGE パラメータを設定するには、次の手順を実行します。

NLS_LANGUAGE パラメータ値を変更し、その影響を問合せからの結果の表示で確認できます。次の例は、最初に NLS_LANGUAGE をイタリア語に設定し、次にドイツ語に設定したことを示しています。

1. SQL Developer で、Oracle Database がインストールされた現行の言語を記録します。

   「Connections」で、まず**「Data Dictionary Reports」**を展開し、次に**「About Your Database」**、そして**「National Language Support Parameters」**を展開します。「Select Connection」ダイアログ・ボックスで、接続のリストから hr_conn を選択します。現行の言語は NLS_LANGUAGE の後にリストされます。

2. 言語をイタリア語に設定します。

```
ALTER SESSION SET NLS_LANGUAGE=ITALIAN;
```

3. SELECT 文を入力して、イタリア語への変更後の出力書式をチェックします。

```
SELECT last_name, hire_date, ROUND(salary/8,2) salary FROM employees
    WHERE employee_id IN (111, 112, 113);
```

この例では次のように出力されます。月名の略称にイタリア語が使用されていることに注意してください。

```
LAST_NAME                 HIRE_DATE     SALARY
------------------------- --------- ----------
Sciarra                   30-SET-97      962.5
Urman                     07-MAR-98        975
Popp                      07-DIC-99      862.5
```

4. 言語をドイツ語に設定します。

```
ALTER SESSION SET NLS_LANGUAGE=GERMAN;
```

5. SELECT 文を入力して、ドイツ語への変更後の出力書式をチェックします。

```
SELECT last_name, hire_date, ROUND(salary/8,2) salary FROM employees
    WHERE employee_id IN (111, 112, 113);
```

この例では次のように出力されます。月名の略称にドイツ語が使用されていることに注意してください。

```
LAST_NAME                 HIRE_DATE     SALARY
------------------------- --------- ----------
Sciarra                   30-SEP-97      962.5
Urman                     07-MRZ-98        975
Popp                      07-DEZ-99      862.5
```

6. 手順 1 に示された元の設定に戻す NLS_LANGUAGE を設定します。次に例を示します。

```
ALTER SESSION SET NLS_LANGUAGE=AMERICAN;
```

参照 :

- NLS_LANGUAGE パラメータの詳細は、『Oracle Database リファレンス』を参照してください。
- サポートされる言語および地域を含むロケール情報の詳細は、『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_TERRITORY パラメータの使用

NLS_TERRITORY パラメータは有効な任意の地域名に対して設定できます。デフォルトは NLS_LANG 設定から導出されます。NLS_TERRITORY により、次の日付および数値書式特性に対するデフォルトの規則が指定されます。

- 日付形式
- 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

NLS_TERRITORY パラメータを変更することにより、導出されたすべての NLS セッション・パラメータが新しい地域のデフォルトの値にリセットされます。

NLS_TERRITORY パラメータを設定するには、次の手順を実行します。

NLS_LANGUAGE パラメータ値を変更し、その影響を問合せからの結果の表示で確認できます。次の例は、NLS_TERRITORY をドイツに設定した結果を示しています。

1. SQL Developer で、初期の SQL Developer デフォルト設定を使用して出力の形式をチェックする SELECT 文を入力します。

```
SELECT TO_CHAR(salary,'L99G999D99') salary FROM employees
   WHERE employee_id IN (100, 101, 102);
```

たとえば、NLS_TERRITORY が AMERICA の場合、この例では次のように出力されます。

```
SALARY
--------------------
          $24,000.00
          $17,000.00
          $17,000.00

3 rows selected
```

2. Oracle Database がインストールされた現行の地域を記録します。

「Connections」で、まず**「Data Dictionary Reports」**、次に**「About Your Database」**、そして**「National Language Support Parameters」**を展開します。「Select Connection」ダイアログ・ボックスで、接続のリストから hr_conn を選択します。現行の地域は NLS_TERRITORY の後にリストされます。

3. NLS_TERRITORY をドイツに設定します。

```
ALTER SESSION SET NLS_TERRITORY=GERMANY;
```

4. SELECT 文を入力して、ドイツへの変更後の出力書式をチェックします。

```
SELECT TO_CHAR(salary,'L99G999D99') salary FROM employees
   WHERE employee_id IN (100, 101, 102);
```

この例では次のように出力されます。3 桁区切りがピリオド（.）に変更され、小数点文字がカンマ（,）に変更されました。通貨記号はドル（$）からユーロ（€）に変更されました。ただし、基礎となるデータは同じであるため、数字は変更されません（つまり、通貨の換算レートは織り込まれません）。

```
SALARY
--------------------
        €24.000,00
        €17.000,00
        €17.000,00

3 rows selected
```

5. 手順 2 に示された元の設定に戻す NLS_TERRITORY を設定します。次に例を示します。

```
ALTER SESSION SET NLS_TERRITORY=AMERICA;
```

参照：

- NLS_TERRITORY パラメータの詳細は、『Oracle Database リファレンス』を参照してください。
- 「NLS_LANGUAGE パラメータの使用」（6-9 ページ）
- サポートされる言語および地域を含むロケール情報の詳細は、『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。

日付および時刻パラメータの設定

日付および時刻の表示を制御し、ローカル書式に基づいて時刻、曜日、月および年のそれぞれの表記規則を許可できます。たとえば、日付を表示する場合、英国では DD/MM/YYYY 書式を使用して表示されますが、中国では一般的に YYYY-MM-DD 書式が使用されます。

参照：

- 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。

日付書式の使用

次の表で説明しているように、日付には複数の異なる書式を使用できます。

国	説明	例
エストニア	dd.mm.yyyy	28.02.2005
ドイツ	dd.mm.rr	28.02.05
中国	yyyy-mm-dd	2005-02-28
英国	dd/mm/yyyy	28/02/2005
米国	mm/dd/yyyy	02/28/2005

NLS_DATE_FORMAT パラメータを使用するには、次の手順を実行します。

NLS_DATE_FORMAT により、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が定義されます。NLS_TERRITORY では、NLS_DATE_FORMAT パラメータのデフォルト値が決定されます。NLS_DATE_FORMAT の値は、有効な任意の日付書式モデルです。次に例を示します。

```
NLS_DATE_FORMAT = "MM/DD/YYYY"
```

文字列リテラルを日付書式に追加するには、その文字列リテラルを二重引用符で囲みます。日付書式に二重引用符を使用する場合は、値全体を一重引用符で囲む必要があることに注意してください。次に例を示します。

```
NLS_DATE_FORMAT = '"Date: "MM/DD/YYYY'
```

Oracle のデフォルトの日付書式が、特定の地域で使用される文化固有の規則に常に対応しているとはかぎりません。'DS' および 'DL' 書式モデルをそれぞれ使用し、SQL で短い日付書式および長い日付書式を使用することにより、ローカライズされた書式で日付を取得できます。次の例は、短い日付書式および長い日付書式の使用例を示しています。

1. SQL Developer で、Oracle Database がインストールされた現行の地域および日付書式を記録します。

   「Connections」で、まず**「Data Dictionary Reports」**、次に**「About Your Database」**、そして**「National Language Support Parameters」**を展開します。「Select Connection」ダイアログ・ボックスで、接続のリストから hr_conn を選択します。現行の日付書式は NLS_DATE_FORMAT の後に、現行の地域は NLS_TERRITORY の後にリストされます。

2. NLS_TERRITORY をアメリカに設定します。

   ```
   ALTER SESSION SET NLS_TERRITORY = America;
   ```

3. 書式モデルを使用して日付を選択します。

```
SELECT hire_date, TO_CHAR(hire_date,'DS') "Short",
  TO_CHAR(hire_date,'DL') "Long" FROM employees
  WHERE employee_id IN (111, 112, 113);
```

この例では次のように出力されます。

```
HIRE_DATE        Short      Long
---------------- ---------- ----------------------------------
30-SEP-97        9/30/1997  Tuesday, September 30, 1997
07-MAR-98        3/7/1998   Saturday, March 07, 1998
07-DEC-99        12/7/1999  Tuesday, December 07, 1999
```

4. 手順 1 に示された元の設定に戻す NLS_TERRITORY および NLS_DATE_FORMAT を設定します。次に例を示します。

```
ALTER SESSION SET NLS_TERRITORY=AMERICA;
ALTER SESSION SET NLS_DATE_FORMAT="MM/DD/YYYY";
```

NLS_DATE_LANGUAGE パラメータを使用するには、次の手順を実行します。

NLS_DATE_LANGUAGE により、TO_CHAR および TO_DATE ファンクションで生成される曜日名および月名の言語が指定されます。NLS_DATE_LANGUAGE では、NLS_LANGUAGE により暗黙的に指定された言語が上書きされます。NLS_DATE_LANGUAGE パラメータには NLS_LANGUAGE パラメータと同じ構文があり、サポートされるすべての言語が有効値です。

NLS_DATE_LANGUAGE パラメータにより、次に使用される言語も決定されます。

- TO_CHAR および TO_DATE 機能によって戻される月名および曜日名の略称
- デフォルトの日付書式（NLS_DATE_FORMAT）に使用される月名および曜日名の略称

デフォルトの日付書式では、NLS_DATE_LANGUAGE パラメータによって決定される月名の略称が使用されます。たとえば、デフォルトの日付書式が DD-MON-YYYY であり、NLS_DATE_LANGUAGE = FRENCH である場合、日付は次のように挿入されます。

```
INSERT INTO table_name VALUES ('12-F 騅 r.-2007');
```

次の例は、NLS_DATE_LANGUAGE をフランス語に設定した結果を示しています。

1. SQL Developer で、Oracle Database がインストールされた現行の曜日名および月名の言語を記録します。

   「Connections」で、まず**「Data Dictionary Reports」**、次に**「About Your Database」**、そして**「National Language Support Parameters」**を展開します。「Select Connection」ダイアログ・ボックスで、接続のリストから hr_conn を選択します。現行の曜日名および月名の言語は NLS_DATE_LANGUAGE の後にリストされます。

2. NLS_DATE_LANGUAGE をフランス語に設定します。

```
ALTER SESSION SET NLS_DATE_LANGUAGE = FRENCH;
```

3. 現在のシステム日付を選択します。

```
SELECT TO_CHAR(SYSDATE, 'Day:Dd Month yyyy') FROM DUAL
```

この例では次のように出力されます。

```
TO_CHAR(SYSDATE,'DAY:DDMONTHYYYY')
----------------------------------
Lundi   :05 Mars      2007
```

4. 手順 1 に示された元の設定に戻す NLS_DATE_LANGUAGE を設定します。次に例を示します。

```
ALTER SESSION SET NLS_DATE_LANGUAGE=AMERICAN;
```

参照：

- NLS_DATE_FORMAT パラメータの詳細は、『Oracle Database リファレンス』を参照してください。
- NLS_DATE_LANGUAGE パラメータの詳細は、『Oracle Database リファレンス』を参照してください。
- 「時刻書式の使用」（6-14 ページ）
- 日付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。
- 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。

時刻書式の使用

この項では、NLS_TIMESTAMP_FORMAT および NLS_TIMESTAMP_TZ_FORMAT パラメータの使用方法を説明します。時刻書式の例のいくつかは次の表のとおりです。

国	説明	例
エストニア	hh24:mi:ss	13:50:23
ドイツ	hh24:mi:ss	13:50:23
中国	hh24:mi:ss	13:50:23
英国	hh24:mi:ss	13:50:23
米国	hh:mi:ssxff am	1:50:23.555 PM

NLS_TIMESTAMP_FORMAT により、TIMESTAMP および TIMESTAMP WITH LOCAL TIME ZONE データ型のデフォルトの日付書式が定義されます。NLS_TERRITORY では、NLS_TIMESTAMP_FORMAT のデフォルト値が決定されます。NLS_TIMESTAMP_FORMAT の値は有効な任意の日付時間書式モデルです。

次の例は NLS_TIMESTAMP_FORMAT の値を示しています。

```
NLS_TIMESTAMP_FORMAT = 'YYYY-MM-DD HH:MI:SS.FF'
```

NLS_TIMESTAMP_TZ_FORMAT パラメータにより、TIMESTAMP および TIMESTAMP WITH LOCAL TIME ZONE データ型のデフォルトの日付書式が定義されます。これには TO_CHAR および TO_TIMESTAMP_TZ ファンクションが使用されます。NLS_TERRITORY パラメータでは、NLS_TIMESTAMP_TZ_FORMAT パラメータのデフォルト値が決定されます。NLS_TIMESTAMP_TZ_FORMAT の値は、有効な任意の日付時間書式モデルです。

書式の値は引用符で囲む必要があります。次に例を示します。

```
NLS_TIMESTAMP_TZ_FORMAT = 'YYYY-MM-DD HH:MI:SS.FF TZH:TZM'
```

NLS_TIMESTAMP_TZ を設定し、使用するには、次の手順を実行します。

次の例では、NLS_TIMESTAMP_TZ_FORMAT 値が設定されます。また、TO_TIMESTAMP_TZ ファンクションを使用し、SELECT 文で明示的に設定した書式を表しています。

1. SQL Developer で、Oracle Database がインストールされた現行の日付書式を記録します。

   「Connections」で、まず**「Data Dictionary Reports」**、次に**「About Your Database」**、そして**「National Language Support Parameters」**を展開します。「Select Connection」ダイアログ・ボックスで、接続のリストから hr_conn を選択します。現行の日付書式は NLS_TIMESTAMP_TZ_FORMAT の後にリストされます。

2. NLS_TIMESTAMP_TZ_FORMAT を設定します。

```
ALTER SESSION SET NLS_TIMESTAMP_TZ_FORMAT = 'YYYY-MM-DD HH:MI:SS.FF TZH:TZM';
```

3. 手順 1 に示された元の設定に戻す NLS_TIMESTAMP_TZ_FORMAT を設定します。次に例を示します。

```
ALTER SESSION SET NLS_TIMESTAMP_TZ_FORMAT='DD-MON-RR HH.MI.SSXFF AM TZR';
```

参照 :

- NLS_TIMESTAMP_TZ_FORMAT パラメータの詳細は、『Oracle Database リファレンス』を参照してください。
- 「日付書式の使用」（6-12 ページ）
- 日付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。
- 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。

カレンダ定義の設定

この項では、カレンダ定義について説明します。

参照 :

- サポートされるカレンダを含むロケール情報の詳細は、『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。

カレンダ書式の概要

次のカレンダ情報は地域別に格納されます。

- **週の最初の曜日** : 一部の文化では、日曜日を最初の曜日とみなしています。また、月曜日を最初の曜日とみなす文化もあります。

  週の最初の曜日は NLS_TERRITORY パラメータにより決定されます。

- **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。Oracle では、この規則をサポートしています。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 パラメータにより決定されます。

- **1 年の日数および月数** : Oracle 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。

参照:

- 「NLS_CALENDAR パラメータの使用」（6-16 ページ）
- サポートされるカレンダを含むロケール情報の詳細は、『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_CALENDAR パラメータの使用

世界中で様々な暦法が使用されています。NLS_CALENDAR によって、Oracle Database で使用する暦法が指定されます。デフォルト値は Gregorian です。値は有効な任意のカレンダ書式名です。

NLS_CALENDAR パラメータには、次の値のいずれかを指定できます。

- Arabic Hijrah（イスラム暦）
- English Hijrah（英語版イスラム暦）
- Gregorian（グレゴリオ暦）
- Japanese Imperial（日本の元号暦）
- Persian（ペルシャ暦）
- ROC Official （台湾暦）
- Thai Buddha（タイ仏教暦）

NLS_CALENDAR パラメータを設定するには、次の手順を実行します。

次の例では、NLS_CALENDAR の値を English Hijrah に設定しています。イスラム暦 1424 年のラマダンの最初の日の値が表示されています。他の NLS パラメータには SQL Developer のデフォルト設定が反映されます。

1. SQL Developer で、Oracle Database がインストールされた現行のカレンダ書式を記録します。

   「Connections」で、まず**「Data Dictionary Reports」**、次に**「About Your Database」**、そして**「National Language Support Parameters」**を展開します。「Select Connection」ダイアログ・ボックスで、接続のリストから hr_conn を選択します。現行の日付書式は NLS_DATE_FORMAT の後に、現行の地域は NLS_CALENDAR の後にリストされます。

2. NLS_CALENDAR を English Hijrah に設定するには、次の手順を実行します。

   ```
   ALTER SESSION SET NLS_CALENDAR='English Hijrah';
   ```

3. イスラム暦 1424 年（グレゴリオ暦では 2007 年）のラマダンの最初の日が表示されます。

   ```
   SELECT TO_DATE('01-Ramadan-1428') FROM DUAL;
   ```

   この例では次のように出力されます。

   ```
   TO_DATE('01-RAMADAN-1428')
   --------------------------
   13 September 2007
   ```

4. 手順 1 に示された元の設定に戻す NLS_CALENDAR を設定します。次に例を示します。

```
ALTER SESSION SET NLS_CALENDAR='GREGORIAN';
```

参照 :

- NLS_CALENDAR パラメータの詳細は、『Oracle Database リファレンス』を参照してください。
- 「カレンダ書式の概要」（6-15 ページ）
- サポートされるカレンダを含むロケール情報の詳細は、『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。

数値書式の使用

データベースでは、数字列を正確に解釈するために、各セッションで使用される数値書式設定の規則を認識している必要があります。たとえば、数値の入力時に小数点文字としてピリオドまたはカンマのどちらを使用するか（234.00 または 234,00）などを認識している必要があります。同じように、アプリケーションでは、数値情報をクライアント側の書式で表示できる必要があります。

次の表に数値書式の例を示します。

国	数値書式
エストニア	1 234 567,89
ドイツ	1.234.567,89
中国	1,234,567.89
英国	1,234,567.89
米国	1,234,567.89

数値書式は NLS_TERRITORY パラメータの設定から導出されますが、NLS_NUMERIC_CHARACTERS パラメータで上書きできます。

参照 :

- 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_NUMERIC_CHARACTERS パラメータの使用

NLS_NUMERIC_CHARACTERS により、グループ・セパレータおよび小数点記号が指定されます。グループ・セパレータとは、千や 100 万などを示すために整数グループを区切る文字です。この文字は G 数値書式モデルで戻されます。小数点記号は、数値の整数部と小数部を区切ります。NLS_NUMERIC_CHARACTERS を設定すると、NLS_TERRITORY の設定から導出されるデフォルト値が上書きされます。値は、グループ・セパレータおよび小数点記号に指定できる 2 つの任意の数字です。

任意の文字を小数点文字またはグループ・セパレータに指定できます。2 つの文字はシングルバイトであり、お互いに異なる文字である必要があります。これらの文字に、数字、プラス記号（+）、マイナス記号（-）、小なり記号（<）、大なり記号（>）を使用することはできません。どちらかの文字を空白にすることができます。

小数点文字としてカンマを設定し、グループ・セパレータとしてピリオドを設定するには、NLS_NUMERIC_CHARACTERS パラメータを次のように指定します。

```
ALTER SESSION SET NLS_NUMERIC_CHARACTERS = ",.";
```

SQL 文には、数値リテラルまたはテキスト・リテラルを表す数値を含むことができます。数値リテラルは引用符で囲みません。数値リテラルは SQL 言語構文の一部で、小数点文字として常にドットを使用し、グループ・セパレータは含まれません。テキスト・リテラルは引用符で囲みます。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。

NLS_NUMERIC_CHARACTERS パラメータを設定するには、次の手順を実行します。

次の例では、ALTER SESSION 文で指定された小数点文字とグループ・セパレータを使用して数値 4000 の書式を設定しています。

1. SQL Developer で、Oracle Database がインストールされた現行の数値書式を記録します。

   「Connections」で、まず**「Data Dictionary Reports」**、次に**「About Your Database」**、そして**「National Language Support Parameters」**を展開します。「Select Connection」ダイアログ・ボックスで、接続のリストから hr_conn を選択します。現行の数値書式は NLS_NUMERIC_CHARACTERS の後にリストされます。

2. NLS_NUMERIC_CHARACTERS を指定したグループ・セパレータおよび小数点文字に設定します。

   ```
   ALTER SESSION SET NLS_NUMERIC_CHARACTERS = ",.";
   ```

   二重引用符を使用します。

3. 書式マスク '9G999D99' を使用して 4000 を表示します。

   ```
   SELECT TO_CHAR(4000, '9G999D99') FROM DUAL;
   ```

   この例では次のように出力されます。グループ・セパレータはピリオド（.）で、小数点文字はカンマ（,）です。

   ```
   TO_CHAR(4000,'9G999D99')
   ------------------------
    4.000,00
   ```

4. 手順 1 に示された元の設定に戻す NLS_NUMERIC_CHARACTERS を設定します。次に例を示します。

   ```
   ALTER SESSION SET NLS_NUMERIC_CHARACTERS=". ";
   ```

参照：

- NLS_NUMERIC_CHARACTERS パラメータの詳細は、『Oracle Database リファレンス』を参照してください。
- 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。

通貨パラメータの使用

基数記号と 3 桁区切りは、ロケールで定義できます。たとえば、小数点は、米国ではピリオド（.）ですが、フランスではカンマ（,）です。金額 $1,234 の持つ意味は国によって異なるため、金額をロケールごとに適切に表示することが重要です。

参照：

- 通貨パラメータを含む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。

通貨書式の概要

世界中で様々な通貨書式が使用されています。標準的な書式の一部を次の表に示します。

国	通貨書式
エストニア	1 234,56 kr
ドイツ	1.234,56€
中国	¥1,234.56
英国	£1,234.56
米国	$1,234.56

参照：

- 「NLS_CURRENCY パラメータの使用」（6-19 ページ）
- 「NLS_ISO_CURRENCY パラメータの使用」（6-20 ページ）
- 「NLS_DUAL_CURRENCY パラメータの使用」（6-21 ページ）
- 通貨パラメータを含む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_CURRENCY パラメータの使用

NLS_CURRENCY パラメータでは、L 数値書式モデルで戻される文字列の各国通貨記号を指定します。NLS_CURRENCY を設定すると、NLS_TERRITORY で暗黙的に定義された設定が上書きされます。値には、有効な任意の通貨記号を指定できます。

変更中の NLS_CURRENCY 値の結果を表示するには、次の手順を実行します。

次の例では、NLS_CURRENCY 値によって置換される L 数値書式モデルを含む書式を使用して給与合計を表示しています。

1. 11000 より大きい値の給与が表示されます。

```
SELECT TO_CHAR(salary, 'L099G999D99') "salary" FROM employees
  WHERE salary > 11000
```

この例では次のように出力されます。この場合、ドル記号（$）が L 数値書式モデルです。

```
salary
---------------------
          $024,000.00
          $017,000.00
          $017,000.00
          $012,000.00
          $014,000.00
          $013,500.00
          $012,000.00
          $011,500.00
          $013,000.00
          $012,000.00

10 rows selected
```

参照：

- NLS_CURRENCY パラメータの詳細は、『Oracle Database リファレンス』を参照してください。
- 「通貨書式の概要」（6-19 ページ）
- 「NLS_ISO_CURRENCY パラメータの使用」（6-20 ページ）
- 「NLS_DUAL_CURRENCY パラメータの使用」（6-21 ページ）
- 通貨パラメータを含む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_ISO_CURRENCY パラメータの使用

NLS_ISO_CURRENCY パラメータでは、C 数値書式モデルで戻される文字列の各国通貨記号を指定します。NLS_ISO_CURRENCY を設定すると、NLS_TERRITORY で暗黙的に定義された値が上書きされます。値には、有効な任意の文字列を指定できます。

各国通貨記号は不明確な場合があります。たとえば、ドル記号（$）は米国ドルを指すこともオーストラリア・ドルを指すこともあります。ISO 仕様には、特定の地域または国に対して固有の通貨記号が定義されています。たとえば、米国ドルに対する ISO 通貨記号は USD です。オーストラリア・ドルに対する ISO 通貨記号は AUD です。

NLS_ISO_CURRENCY パラメータの構文は NLS_TERRITORY パラメータの構文と同じで、サポート対象地域すべてが有効値です。

NLS_ISO_CURRENCY パラメータを設定するには、次の手順を実行します。

次の例では、適切な書式モデルを使用してフランス用の ISO 通貨記号および給与書式を設定しています。

1. SQL Developer で、Oracle Database がインストールされた ISO 通貨を記録します。

   「Connections」で、まず**「Data Dictionary Reports」**、次に**「About Your Database」**、そして**「National Language Support Parameters」**を展開します。「Select Connection」ダイアログ・ボックスで、接続のリストから hr_conn を選択します。現行の ISO 通貨書式は NLS_ISO_CURRENCY の後にリストされます。

2. NLS_ISO_CURRENCY をフランスに設定します。

```
ALTER SESSION SET NLS_ISO_CURRENCY=FRANCE;
```

3. 書式を使用して選択した給与を表示します。

```
SELECT TO_CHAR(salary, 'C099G999D99') "Salary" FROM employees
  WHERE department_id = 60;
```

   この例では次のように出力されます。

```
Salary
------------------
     EUR009,000.00
     EUR006,000.00
     EUR004,800.00
     EUR004,800.00
     EUR004,200.00

5 rows selected
```

4. 手順 1 に示された元の設定に戻す NLS_ISO_CURRENCY を設定します。次に例を示します。

```
ALTER SESSION SET NLS_ISO_CURRENCY=AMERICA;
```

参照：

- NLS_ISO_CURRENCY パラメータの詳細は、『Oracle Database リファレンス』を参照してください。
- 「通貨書式の概要」（6-19 ページ）
- 「NLS_CURRENCY パラメータの使用」（6-19 ページ）
- 「NLS_DUAL_CURRENCY パラメータの使用」（6-21 ページ）
- 通貨パラメータを含む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_DUAL_CURRENCY パラメータの使用

NLS_DUAL_CURRENCY を使用すると、NLS_TERRITORY で暗黙的に定義されたデフォルトの二重通貨記号が上書きされます。値には有効な任意の記号を指定できます。

NLS_DUAL_CURRENCY は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されました。

参照：

- NLS_DUAL_CURRENCY パラメータの詳細は、『Oracle Database リファレンス』を参照してください。
- 「通貨書式の概要」（6-19 ページ）
- 「NLS_CURRENCY パラメータの使用」（6-19 ページ）
- 「NLS_ISO_CURRENCY パラメータの使用」（6-20 ページ）
- 通貨パラメータを含む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。

言語ソートおよび検索の使用

ソート順序は言語によって異なります。ある言語ではアルファベットの文字の並びに応じてソートされ、ある言語では文字の画数でソートされ、またある言語では語の発音によってソートされます。また、文字のアクセントの扱いも言語によって異なります。たとえば、デンマーク語の文字 Æ は、Z の後に来ます。また、Y と Ü は同じ文字の変形とみなされます。

言語ソート・パラメータを使用することで、データのソート方法を定義できます。基本的な言語定義では、文字列を独立した文字の連続として扱います。

参照：

- 言語ソートおよび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。

NLS_SORT パラメータの使用

NLS_SORT パラメータにより、ORDER BY 問合せの照合順序（言語ソート）が指定されます。これにより、NLS_LANGUAGE パラメータから導出されたデフォルトの NLS_SORT 値は上書きされます。NLS_SORT の値は BINARY、または有効な任意の言語ソート名です。

NLS_SORT = BINARY | *sort_name*

値が BINARY である場合、照合順序は基礎となるコード体系の数値コードに基づきます。データ型に応じて、これはデータベース・キャラクタ・セットまたは各国語キャラクタ・セットのバイナリ・ソート順序のいずれかです。値が言語ソートの名前である場合、ソートは定義したソートの順序に基づきます。NLS_LANGUAGE パラメータによってサポートされるほとんどの（すべてではありません）言語は同じ名前の言語ソートをサポートします。

NLS_SORT パラメータを設定するには、次の手順を実行します。

NLS_SORT パラメータ値を変更し、その結果を問合せの結果の表示で確認できます。次の例では NLS_SORT を最初にバイナリに設定して、次にスペイン語（SPANISH_M）に設定した結果を示しています。スペインでは伝統的に ch、ll、ñ を独自の文字として扱い、それぞれ c、l、n の次に来ます。

1. SQL Developer で、Oracle Database がインストールされた現行の照合書式を記録します。

   「Connections」で、まず**「Data Dictionary Reports」**、次に**「About Your Database」**、そして**「National Language Support Parameters」**を展開します。「Select Connection」ダイアログ・ボックスで、接続のリストから hr_conn を選択します。現行の照合書式は NLS_SORT の後にリストされます。

2. NLS_SORT をバイナリに設定します。

   ```
   ALTER SESSION SET NLS_SORT=BINARY;
   ```

3. ORDER BY 句のある SELECT 文を入力し、変更後の出力を確認します。

   ```
   SELECT last_name FROM employees
     WHERE last_name LIKE 'C%'
     ORDER BY last_name;
   ```

   この例では次のように出力されます。

   ```
   LAST_NAME
   -------------------------
   Cabrio
   Cambrault
   Cambrault
   Chen
   Chung
   Colmenares

   6 rows selected
   ```

4. NLS_SORT を SPANISH_M に設定します。

   ```
   ALTER SESSION SET NLS_SORT=spanish_m;
   ```

5. 同様の SELECT 文を入力し、変更後の出力を確認します。

   ```
   SELECT last_name FROM employees
     WHERE last_name LIKE 'C%'
     ORDER BY last_name;
   ```

   この例では次のように出力されます。Colmenares は現在 Chen の前に来ます。

   ```
   LAST_NAME
   -------------------------
   Cabrio
   Cambrault
   Cambrault
   Colmenares
   Chen
   Chung

   6 rows selected
   ```

6. 手順 1 に示された元の設定に戻す NLS_SORT を設定します。次に例を示します。

   ```
   ALTER SESSION SET NLS_SORT=BINARY;
   ```

参照：

- NLS_SORT パラメータの詳細は、『Oracle Database リファレンス』を参照してください。
- 「NLS_COMP パラメータの使用」(6-23 ページ)
- 「大 / 小文字およびアクセントを区別しない検索の使用」(6-24 ページ)
- 言語ソートおよび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。

NLS_COMP パラメータの使用

比較演算子を使用するとき、文字は指定したコード体系のバイナリ・コードにより、比較されます。バイナリ・コードより高い場合、文字は他よりも大きくなります。バイナリでの言語順序は特定の言語の言語順序と一致しない場合があるため、このような比較が言語的に正確ではない可能性があります。

NLS_COMP パラメータの値は SQL 操作の比較動作に影響します。値には BINARY（デフォルト）または LINGUISTIC を指定できます。バイナリ比較ではなく言語比較を実行する場合は、NLS_COMP パラメータを使用することで、SQL 文の NLSSORT ファンクションを使用する面倒な処理を回避できます。NLS_COMP を LINGUISTIC に設定すると、SQL により NLS_SORT パラメータの値に基づいた言語比較が実行されます。

NLS_COMP パラメータを設定するには、次の手順を実行します。

NLS_COMP パラメータ値を変更し、その結果を問合せ結果を表示することで確認できます。次の例では、従業員名に対しバイナリ比較を実行した後にスペイン語の言語比較を実行したことを示しています。

1. SQL Developer で、Oracle Database がインストールされた現行の比較演算子書式を記録します。

   「Connections」で、まず**「Data Dictionary Reports」**、次に**「About Your Database」**、そして**「National Language Support Parameters」**を展開します。「Select Connection」ダイアログ・ボックスで、接続のリストから hr_conn を選択します。現行の比較演算子書式は NLS_COMP の後にリストされます。

2. NLS_SORT をスペイン語に設定し、NLS_COMP を BINARY に設定します。

   ```
   ALTER SESSION SET NLS_SORT=spanish_m NLS_COMP=binary;
   ```

3. 姓が C で始まる従業員を戻す SELECT 文を入力します。

   ```
   SELECT last_name FROM employees
     WHERE last_name LIKE 'C%';
   ```

   この例では次のように出力されます。

   ```
   LAST_NAME
   -------------------------
   Cabrio
   Cambrault
   Cambrault
   Chen
   Chung
   Colmenares

   6 rows selected
   ```

4. NLS_COMP を LINGUISTIC に設定します。

```
ALTER SESSION SET NLS_COMP=linguistic;
```

5. 同様の SELECT 文を入力し、変更後の出力を確認します。

```
SELECT last_name FROM employees
  WHERE last_name LIKE 'C%';
```

この例では次のように出力されます。2 つの行、Chen および Chung は戻されません。これは、スペイン語では ch は c の後に続く別の文字として扱われ、ch は、スペイン語の言語比較が実行されたときに除外されるためです。

```
LAST_NAME
-------------------------
Cabrio
Cambrault
Cambrault
Colmenares

4 rows selected
```

6. 手順 1 に示された元の設定に戻す NLS_COMP を設定します。次に例を示します。

```
ALTER SESSION SET NLS_COMP=BINARY;
```

参照：

- NLS_COMP パラメータの詳細は、『Oracle Database リファレンス』を参照してください。
- 「NLS_SORT パラメータの使用」（6-21 ページ）
- 「大 / 小文字およびアクセントを区別しない検索の使用」（6-24 ページ）
- 言語ソートおよび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。

大 / 小文字およびアクセントを区別しない検索の使用

データベースの操作では、大 / 小文字および文字のアクセントが区別されます。場合によっては、大 / 小文字を区別しない、またはアクセントを区別しない比較を実行する必要があります。大 / 小文字を区別しない、またはアクセントを区別しないソートを指定するには、NLS_SORT セッション・パラメータを使用します。

大 / 小文字を区別しない、またはアクセントを区別しないソートを指定するには、次の手順を実行します。

- 大 / 小文字を区別しないソートを実行する場合は、Oracle ソート名に _CI を付けます。次に例を示します。
  - BINARY_CI: アクセントを区別し、大 / 小文字を区別しないバイナリ・ソート
  - GENERIC_M_CI: アクセントを区別し、大 / 小文字を区別しない GENERIC_M ソート
- アクセントおよび大 / 小文字を区別しないソートを実行する場合は、Oracle 名に _AI を付けます。次に例を示します。
  - BINARY_AI: アクセントおよび大 / 小文字を区別しないバイナリ・ソート
  - FRENCH_M_AI: アクセントおよび大 / 小文字を区別しない FRENCH_M ソート

参照：

- NLS_SORT パラメータの詳細は、『Oracle Database リファレンス』を参照してください。
- 「NLS_SORT パラメータの使用」（6-21 ページ）
- 「NLS_COMP パラメータの使用」（6-23 ページ）
- 言語ソートおよび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。

長さセマンティクスの使用

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記憶要件を定義する場合に役立ちます。たとえば、Unicode データベース（AL32UTF8）で、VARCHAR2 列を英語の 5 文字とともに 5 文字の中国語文字を格納できるように定義する必要があるとします。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。

次の式ではバイト・セマンティクスが使用されています。VARCHAR2 式の BYTE 修飾子および SQL ファンクション名の B 接尾辞に注意してください。

- `VARCHAR2(20 BYTE)`
- `SUBSTRB(`*`string`*`, 1, 20)`

次の式ではキャラクタ・セマンティクスが使用されています。VARCHAR2 式の CHAR 修飾子に注意してください。

- `VARCHAR2(20 CHAR)`
- `SUBSTR(`*`string`*`, 1, 20)`

参照：

- 長さセマンティクスを含むキャラクタ・セットの選択または変更の詳細は、『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_LENGTH_SEMANTICS パラメータの使用

NLS_LENGTH_SEMANTICS パラメータによって、BYTE（デフォルト）または CHAR セマンティクスが指定されます。デフォルトでは、文字データ型 CHAR および VARCHAR2 が、文字単位ではなくバイト単位で指定されます。このため、表定義で CHAR(20) と指定すると、文字データを格納するために 20 バイトが割り当てられます。

NLS_LENGTH_SEMANTICS によって、CHAR、VARCHAR2 および LONG の各列をバイト・セマンティクスまたはキャラクタ・セマンティクスのいずれかを使用して作成できます。NCHAR、NVARCHAR2、CLOB および NCLOB の各列は、常に文字ベースです。既存の列は影響を受けません。

NLS_LENGTH_SEMANTICS パラメータを設定するには、次の手順を実行します。

1. SQL Developer で、Oracle Database がインストールされた現行のセマンティクス書式を記録します。

   「Connections」で、まず**「Data Dictionary Reports」**、次に**「About Your Database」**、そして**「National Language Support Parameters」**を展開します。「Select Connection」ダイアログ・ボックスで、接続のリストから hr_conn を選択します。現行のセマンティクス書式は NLS_LENGTH_SEMANTICS の後にリストされます。

2. NLS_LENGTH_SEMANTICS を BYTE に設定します。

   ```
   ALTER SESSION SET NLS_LENGTH_SEMANTICS=BYTE;
   ```

3. 次のように表を作成します。

   ```
   CREATE TABLE SEMANTICS_BYTE(SOME_DATA VARCHAR2(20));
   ```

4. 表 SEMANTICS_BYTE のデータ型をチェックします。

   「Connections」タブを選択し hr_conn 接続を展開して、「Tables」にすべての表を表示します。SEMANTICS_BYTE 表を選択します。SOME_DATA 列のデータ型は、VARCHAR2(20 BYTE) としてリストされます。

5. NLS_LENGTH_SEMANTICS を CHAR に設定します。

   ```
   ALTER SESSION SET NLS_LENGTH_SEMANTICS=CHAR;
   ```

6. 次のように表を作成します。

   ```
   CREATE TABLE SEMANTICS_CHAR(SOME_DATA VARCHAR2(20));
   ```

7. 表 SEMANTICS_CHAR のデータ型をチェックします。

   「Connections」タブを選択し hr_conn 接続を展開して、「Tables」にすべての表を表示します。SEMANTICS_CHAR 表を選択します。SOME_DATA 列のデータ型は、VARCHAR2(20 CHAR) としてリストされます。

8. SEMANTICS_BYTE および SEMANTICS_CHAR 表を削除します。

   「Tables」ナビゲーション階層で、各表の名称を右クリックして、メニューから**「Table」**を選択してから**「Drop」**を選択します。**「Apply」**をクリックしてから「Confirmation」ダイアログ・ボックスで**「OK」**をクリックします。

9. 手順 1 に示された元の設定に戻す NLS_LENGTH_SEMANTICS を設定します。次に例を示します。

   ```
   ALTER SESSION SET NLS_LENGTH_SEMANTICS=BYTE;
   ```

参照：

- NLS_LENGTH_SEMANTICS パラメータの詳細は、『Oracle Database リファレンス』を参照してください。
- 長さセマンティクスを含むキャラクタ・セットの選択または変更の詳細は、『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。

グローバルなアプリケーション開発

この項では、複数の言語アプリケーション用にデプロイできる SQL および PL/SQL の Unicode 関連の機能を説明します。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。

参照 :

- Unicode を使用した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 の概要

Unicode とはエンコードされたユニバーサル・キャラクタ・セットです。このセットを使用すると、1 つのキャラクタ・セットを使用して任意の言語の情報を格納できます。Unicode には、プラットフォーム、プログラムまたは言語に関係なく、すべての文字に対する一意のコード値が用意されています。

Unicode には次のような利点があります。

- キャラクタ・セット変換機能と言語ソート機能が簡素化されます。
- ネイティブのマルチバイト・キャラクタ・セットに比べてパフォーマンスが改善されます。
- Unicode 規格に基づく Unicode データ型がサポートされます。

Unicode 文字は、Oracle データベースにおいて次の 2 つの方法で格納できます。

- UTF8 エンコードされた文字を SQL `CHAR` データ型として格納できるように、Unicode データベースを作成できます。
- Unicode データ型を使用して、特定の列の多言語データをサポートできます。データベース・キャラクタ・セットの定義方法に関係なく、SQL `NCHAR` データ型の列に Unicode 文字を格納できます。`NCHAR` データ型は Unicode データ型専用です。

参照 :

- 「SQL 文字データ型の使用」(6-27 ページ)
- 「Unicode 文字列リテラルの使用」(6-29 ページ)
- 「NCHAR リテラルの置換」(6-29 ページ)
- Unicode を使用した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。

SQL 文字データ型の使用

SQL `NCHAR` データ型には `NCHAR` および `NVARCHAR2` という 2 つのデータ型があります。

SQL Developer では、各列の**タイプ**に適切な値を選択することにより、表を作成または編集するダイアログ・ボックスでこれらのデータ型を指定できます。また、「SQL Worksheet」を使用し、`CREATE TABLE` 文を入力することで、各列名およびデータ型を指定できます。

参照 :

- 「Unicode の概要」(6-27 ページ)
- 「Unicode 文字列リテラルの使用」(6-29 ページ)
- 「NCHAR リテラルの置換」(6-29 ページ)
- Unicode を使用した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。

NCHAR データ型の使用

NCHAR データ型として表の列または PL/SQL 変数を定義するとき、長さは文字数として指定されます。たとえば、次の文により最大文字長が 30 の列が 1 つ作成されます。

```
CREATE TABLE table1 (column1 NCHAR(30));
```

列の最大バイト数は、最大文字数と各文字の最大バイト数の積です。

たとえば、各国語キャラクタ・セットが UTF8 の場合、最大バイト長は 30 文字に 1 文字当たり 3 バイトを乗算した値、つまり 90 バイトとなります。

すべての NCHAR データ型に使用する各国語キャラクタ・セットは、データベースの作成時に定義します。各国語キャラクタ・セットは UTF8 または AL16UTF16 のいずれかです。デフォルトは AL16UTF16 です。

使用可能な最大列サイズは、各国語キャラクタ・セットが UTF8 の場合は 2000 文字、AL16UTF16 の場合は 1000 文字です。実際のデータが最大バイト制限の 2000 の対象となります。2 つのサイズ制限は同時に満たす必要があります。PL/SQL では、NCHAR データの最大長は 32767 バイトです。NCHAR 変数は 32767 文字まで定義できますが、実際のデータは 32767 バイトを超えることはできません。列の長さより短い値を挿入すると、最大文字長と最大バイト長のいずれか小さい方の値まで空白で埋められます。

参照 :

- 「NVARCHAR2 データ型の使用」(6-28 ページ)
- Unicode を使用した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。

NVARCHAR2 データ型の使用

NVARCHAR2 データ型は、各国語キャラクタ・セットを使用する可変長文字列を指定します。NVARCHAR2 列を使用して表を作成するときは、列の最大文字数を指定します。NVARCHAR2 の長さは、NCHAR の場合と同様に文字単位です。Oracle Database では値が列の最大長を超えないかぎり、ユーザーが指定したとおりに各値が列内に格納されます。文字列の値が最大長まで埋め込まれることはありません。

使用可能な最大列サイズは、各国語キャラクタ・セットが UTF8 の場合は 4000 文字、AL16UTF16 の場合は 2000 文字です。バイト単位では、NVARCHAR2 列の最大長は 4000 バイトです。バイト数の上限と文字数の上限の両方を満たす必要があるため、実際に NVARCHAR2 列に使用できる最大文字数は、4000 バイトに書込み可能な文字数となります。

PL/SQL では、NVARCHAR2 変数の最大長は 32767 バイトです。NVARCHAR2 変数は 32767 文字まで定義できますが、実際のデータは 32767 バイトを超えることはできません。

次の文により、最大文字長が 2000 で最大バイト長が 4000 の NVARCHAR2 列を 1 つ含む表が作成されます。

```
CREATE TABLE table2 (column2 NVARCHAR2(2000));
```

参照 :

- 「NCHAR データ型の使用」(6-28 ページ)
- Unicode を使用した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 文字列リテラルの使用

Unicode 文字列リテラルは、次の方法で SQL および PL/SQL に入力できます。

- 一重引用符で囲まれた文字列リテラルの前に接頭辞 N を付加します。この接頭辞によって、その後に続く文字列リテラルが NCHAR 文字列リテラルであることが明示的に示されます。たとえば、N'résumé' は NCHAR 文字列リテラルです。この方法に関する制限は、6-29 ページの「NCHAR リテラルの置換」を参照してください。
- NCHR(*n*) SQL ファンクションを使用します。NCHR（*n*）SQL ファンクションは、各国語キャラクタ・セット（AL16UTF16 または UTF8）内の文字コードの数を戻します。複数の NCHR*n*）ファンクションを連結した結果は、NVARCHAR2 データになります。このように、クライアントとサーバーのキャラクタ・セット変換を行わずに、NVARCHAR2 文字列を直接作成できます。たとえば、NCHR（32）は空白文字を表します。

  NCHR(*n*) は各国語キャラクタ・セットに関連付けられているため、結果値の移植性はその各国語キャラクタ・セットで実行されるアプリケーションに限定されます。この移植性が重要な場合は、UNISTR ファンクションを使用して移植性の制限を解除します。
- UNISTR('*string*') SQL ファンクションを使用します。UNISTR(*'string'*) ファンクションは、文字列を各国語キャラクタ・セットに変換します。移植性を確保してデータを保持するには、ASCII 文字と Unicode エンコーディングのみを ¥xxxx 形式で含めます。xxxx は、UTF-16 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。たとえば、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 エンコーディングは、各国語キャラクタ・セットに直接変換されます。

最後の 2 つの方法を使用すると、すべての Unicode 文字列リテラルをエンコードできます。

参照：

- 「Unicode の概要」（6-27 ページ）
- 「SQL 文字データ型の使用」（6-27 ページ）
- 「NCHAR リテラルの置換」（6-29 ページ）
- Unicode を使用した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。

NCHAR リテラルの置換

SQL または PL/SQL 文の一部として、接頭辞 N あり、またはなしの任意のリテラルのテキストが文の残りに同じ文字で囲まれます。クライアント側では、文はクライアント・キャラクタ・セットで、NLS_LANG パラメータによって定義されたキャラクタ・セットで決定されます。サーバー側では、文はデータベース・キャラクタ・セットです。

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、キャラクタ・セットはそれに応じて変換されます。データベース・キャラクタ・セットにテキスト・リテラルで使用されるすべての文字が含まれない場合、データはこの変換で消失します。これは CHAR テキスト・リテラルよりも NCHAR 文字列リテラルに大きく影響します。これは N' リテラルがデータベース・キャラクタ・セットから独立するように設計されており、クライアント・キャラクタ・セットで許可されるすべてのデータを含むことができるためです。

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、クライアント環境変数 ORA_NCHAR_LITERAL_REPLACE を TRUE に設定して NCHAR リテラル置換を使用できます。これによりクライアント側の N リテラルを、文の実行時に Unicode にデコードされる内部書式で置換できるようになります。デフォルトでは、下位互換性を維持するために NCHAR リテラル置換は無効です。

参照：

- 「Unicode の概要」（6-27 ページ）
- 「SQL 文字データ型の使用」（6-27 ページ）
- 「Unicode 文字列リテラルの使用」（6-29 ページ）
- Unicode を使用した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。

NLS パラメータを使用したロケール依存の機能の使用

動作をグローバリゼーション・サポートの規則に依存しているすべての SQL ファンクションで、NLS パラメータを指定できます。これらのファンクションは、TO_CHAR、TO_DATE、TO_NUMBER、NLS_UPPER、NLS_LOWER、NLS_INITCAP および NLSSORT です。

これらのファンクションの NLS パラメータを指定すると、セッションの NLS パラメータに依存せずにファンクションを評価できます。この機能は、数字や日付が文字列リテラルとして含まれている SQL 文で重要となる場合があります。

たとえば、日付に対する言語が AMERICAN であることを評価するには、次の 2 つの問合せがあります。

- ALTER SESSION を使用して NLS_DATE_LANGUAGE パラメータおよび NLS_CALENDAR パラメータを設定します。

  ```
  ALTER SESSION SET NLS_DATE_LANGUAGE=American;
  SELECT last_name FROM employees WHERE hire_date > '01-JAN-1999';
  ```

- SQL 文の WHERE 句、TO_DATE ファンクションで NLS_DATE_LANGUAGE パラメータを指定します。

  ```
  SELECT last_name FROM employees
       WHERE hire_date > TO_DATE('01-JAN-1999','DD-MON-YYYY',
       'NLS_DATE_LANGUAGE = AMERICAN');
  ```

  このようにセッション言語に依存しない SQL 文を必要に応じて定義できます。これらの文は、文字列リテラルをビュー内の SQL 文、CHECK 制約、またはトリガーで表示するときに必要となります。

ロケール依存の SQL ファンクションでオプションの NLS パラメータを明示的に指定する必要があるのは、セッションの NLS パラメータ値に依存しないことが必要となる SQL 文のみです。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。

すべての文字ファンクションは、シングルバイトとマルチバイトの両方の文字をサポートします。単位を明示的に示した場合を除いて、文字ファンクションはバイト単位ではなく文字単位で動作します。

SQL ファンクションでビューやトリガーが評価される場合、NLS ファンクションパラメータには現行のセッションからのデフォルト値が使用されます。SQL ファンクションで CHECK 制約が評価される場合は、データベースの作成時に NLS パラメータに指定されたデフォルト値が使用されます。

参照：

- オプションの NLS パラメータによるロケール依存の詳細は、『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 ファンクションの NLS パラメータの指定

NLS パラメータは SQL ファンクションでパラメータ = 値のように指定します。次に例を示します。

```
'NLS_DATE_LANGUAGE = AMERICAN'
```

SQL ファンクションで次の NLS パラメータを指定できます。

```
NLS_DATE_LANGUAGE
NLS_NUMERIC_CHARACTERS
NLS_CURRENCY
NLS_ISO_CURRENCY
NLS_DUAL_CURRENCY
NLS_CALENDAR
NLS_SORT
```

一部の言語では、1 つの小文字が複数の大文字に対応したり、1 つの大文字が複数の小文字に対応します。したがって、NLS_UPPER、NLS_LOWER および NLS_INITCAP ファンクションの出力の長さは入力の長さと異なることがあります。次の表で、特定の SQL ファンクションに対し、どの NLS パラメータが有効であるかを示します。

SQL ファンクション	有効な NLS パラメータ
TO_DATE	NLS_DATE_LANGUAGE, NLS_CALENDAR
TO_NUMBER	NLS_NUMERIC_CHARACTERS, NLS_CURRENCY, NLS_ISO_CURRENCY, NLS_DUAL_CURRENCY,
TO_CHAR	NLS_DATE_LANGUAGE, NLS_NUMERIC_CHARACTERS, NLS_CURRENCY, NLS_ISO_CURRENCY, NLS_DUAL_CURRENCY, NLS_CALENDAR
TO_NCHAR	NLS_DATE_LANGUAGE, NLS_NUMERIC_CHARACTERS, NLS_CURRENCY, NLS_ISO_CURRENCY, NLS_DUAL_CURRENCY, NLS_CALENDAR
NLS_UPPER	NLS_SORT
NLS_LOWER	NLS_SORT
NLS_INITCAP	NLS_SORT
NLSSORT	NLS_SORT

例 6-1 は、SQL ファンクションで NLS パラメータを使用する方法を示す SELECT 文を示しています。これらの SELECT 文を実行した後（SQL ワークショップでグループとして実行可能）、「Script Output」ペインにおける各文の出力を確認します（多くの文の出力は非常に長いです）。

例 6-1 SQL ファンクションでの NLS パラメータの使用

```
SELECT TO_DATE('1-JAN-99', 'DD-MON-YY',
   'NLS_DATE_LANGUAGE = American') "01/01/99" FROM DUAL;

SELECT TO_CHAR(hire_date, 'DD/MON/YYYY',
   'NLS_DATE_LANGUAGE = French') "Hire Date" FROM employees;

SELECT TO_CHAR(SYSDATE, 'DD/MON/YYYY',
   'NLS_DATE_LANGUAGE = ''Traditional Chinese'' ') "System Date" FROM DUAL;

SELECT TO_CHAR(13000, '99G999D99',
  'NLS_NUMERIC_CHARACTERS = '',.''') "13K" FROM DUAL;

SELECT TO_CHAR(salary, '99G999D99L', 'NLS_NUMERIC_CHARACTERS = '',.''
   NLS_CURRENCY = ''EUR''') salary FROM employees;
```

```
SELECT TO_CHAR(salary, '99G999D99C', 'NLS_NUMERIC_CHARACTERS = ''.,''
   NLS_ISO_CURRENCY = Japan') salary FROM employees;

SELECT NLS_UPPER(last_name, 'NLS_SORT = Swiss') "Last Name" FROM employees;

SELECT last_name FROM employees
   ORDER BY NLSSORT(last_name, 'NLS_SORT = German');
```

参照:

- 「SQL ファンクションで受け入れられない NLS パラメータ」(6-32 ページ)
- オプションの NLS パラメータによるロケール依存の詳細は、『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 ファンクションで受け入れられない NLS パラメータ

NLS パラメータ NLS_LANGUAGE、NLS_TERRITORY および NLS_DATE_FORMAT は NLSSORT 以外の SQL ファンクションでは使用できません。

NLS_LANGUAGE は、NLS_DATE_LANGUAGE のセッションの値を妨げます。たとえば TO_CHAR ファンクションで NLS_LANGUAGE を指定すると、そのセッションの NLS_DATE_LANGUAGE パラメータ値と異なる場合は、Oracle Database から無視されます。

NLS_DATE_FORMAT パラメータおよび NLS_TERRITORY_FORMAT パラメータは必要な書式モデルを妨げる可能性があるので、パラメータとしては受け入れられません。TO_CHAR ファンクションまたは TO_DATE ファンクションで NLS パラメータを使用する場合は、日付書式を指定する必要があります。したがって NLS_DATE_FORMAT パラメータおよび NLS_TERRITORY_FORMAT パラメータはこれらの変換ファンクションには有効ではありません。TO_CHAR ファンクションまたは TO_DATE ファンクションで NLS_DATE_FORMAT または NLS_TERRITORY_FORMAT を指定すると、Oracle Database はエラーを返します。

参照:

- 「SQL ファンクションの NLS パラメータの指定」(6-31 ページ)
- オプションの NLS パラメータによるロケール依存の詳細は、『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。

7

データベース・アプリケーションのデプロイ

この章では、アプリケーションをサポートするデータベース・オブジェクトのパッケージを作成し、インストールする方法を説明します。たとえば、このガイドの以前の説明に従った場合、作成したオブジェクトを使用します。他のシステムにデプロイするオブジェクト定義およびデータを収集するプロセスを説明します。実際の環境で、操作はこのガイドで説明されているように単純ではない可能性がありますが、手順および考慮事項は同じです。

すべてのオブジェクト名には一貫した接頭辞を使用することをお薦めします。これによりオブジェクトを簡単に識別できます。SQL Developer 接続ナビゲータ・ディスプレイで、また Oracle データベースのデータ・ディクショナリに対して SQL Developer レポートおよびパフォーマンスの問合せを確認する場合に、一貫した接頭辞によりオブジェクトが分類されます。

この章の内容は次のとおりです。

デプロイメントの概要

アプリケーションをサポートするデータベース・オブジェクトがデプロイされていない場合、アプリケーションのデプロイメントも通常完了しません。参照表のシード・データなどの必要なすべてのデータ、およびデータベース・オブジェクトの両方を作成するスクリプトの作成によって、これらのオブジェクトをデプロイできます。データベース・オブジェクトには、アプリケーション・ロジックを実装するように作成した表、ビュー、ファンクション、パッケージおよびその他が含まれます。

環境のデプロイメント

Oracle Database のアプリケーションをデプロイする際は、次のシステム環境を作成する必要があります。

手順 1: テスト環境の作成

初期のデプロイメント、他の環境にアプリケーションをデプロイする前のアプリケーションの全体のテスト、およびアプリケーション・ユーザーの研修に対して、テスト環境を常に用意する必要があります。

テストでは、アプリケーションの機能および正確にパッケージしたかどうかがチェックされます。アプリケーションと依存関係にあるオブジェクトがない場合、本番環境で実際のユーザーにデプロイされた後ではなく、テスト中にオブジェクトを取り込むことができます。

手順 2: 品質保証（QA）環境の作成

アプリケーションが非常に複雑で、ユーザーがリソースを所持する場合、システムに対する変更を厳格にチェックできる QA 環境を作成します。

手順 3: 教育環境の作成

教育環境では、他の環境へ影響を与えずに内部ユーザーまたは外部ユーザーに研修および実習を提供できます。本番環境の前または後に教育環境を作成でき、また他の環境への更新とは無関係に教育環境を更新できます。

手順 4: 本番環境の作成

本番環境には、企業の通常業務の実際のデータおよびデータベース・オブジェクトが含まれます。本番環境に移る前に、テスト環境ですべてのオブジェクトをテストします。

デプロイする環境の種類にかかわらず、デプロイメント・プロセスは同一です。

デプロイメントの計画

アプリケーションをデプロイする前に、データベース・オブジェクト間の依存性を理解する必要があります。オブジェクトが他のオブジェクトと依存関係にある場合、依存オブジェクトはいずれの場合にも存在するため、適切な順序でオブジェクトを作成する必要があります。依存オブジェクトが存在しない場合、発生するエラーまたは問題は次のとおりです。

- 制約などに関して、`CREATE` 文が失敗します。
- ファンクション、プロシージャおよびパッケージになどに関して、オブジェクトは作成されますが、無効の状態になりました。

データをデプロイするには、各表のデータに対して次のいずれかの方法を使用できます。データの妥当性の度合いに応じて方法を選択します。

- 妥当性の問題が発生する可能性がない場合、データをロードします。

  この方法は、データがあらゆる制約に違反しない場合、主キー列および一意キー列に重複する値が存在しない場合、すべての外部キー参照がすでに存在する場合、およびチェック制約により制御されているすべての列のデータが制約を満たす場合に使用できます。たとえば、開発環境から参照データを単純にロードする場合や適切な順序でデータをロードする場合は、制約には違反しないため、制約を無効化する必要はありません。

- データをロードする前にすべての制約を無効化して、データのロードの完了後に制約を有効化します。

  データの順序付け（ロードする表および依存データが多くある場合など）をせずにデータをロードする場合、または外部のソース（古いアプリケーション、フラット・ファイルまたはスプレッドシートなど）からデータをロードする場合、データをロードする前に制約を無効化する必要があります。

  すべてのデータが制約を満たすことに失敗した場合、制約は有効化されず、データを修正して再試行する必要があります。

異なるタイプのデータベース・オブジェクトに対してインストール・スクリプトを実行する順序の一般的なガイドラインは次のとおりです。

1. パッケージの仕様部
2. 適切な順序の表（制約および索引を持つ表）
3. 順序（トリガーによって最も頻繁に使用されるため）
4. トリガー
5. シノニム
6. ビュー（ファンクション、プロシージャまたはシノニムを参照する場合があるため）
7. パッケージ本体
8. データ（オプションで、データをロードする前にすべての制約を無効化して、その後に有効化）

パッケージの仕様部は、常に有効な状態で、他のオブジェクトに参照される場合があるため、最初にスクリプトを実行します。パッケージ本体は、他のオブジェクト・タイプを参照するため、最後に作成する必要があります。依存性の問題のため、ファンクションおよびプロシージャをパッケージに含めることをお薦めします。

このガイドの他の項の説明に従った場合、サンプルの Oracle HR スキーマにオブジェクトをすでに作成しています。この項では、別の標準の HR スキーマにオブジェクトをデプロイすると想定します。

作成した表で、`scores` には `performance_parts` および `evaluations` 両方に対する外部キーが含まれます。これは、参照表の主キーが作成されるまでこれらの外部キーを作成できないということです。最初に `evaluations` 表および制約を作成し、次に `performance_parts` 表、そして `scores` 表を作成します。表、順序およびトリガーに対してスクリプトが 1 つのみ存在します。このため、この実行に対する手動編集を最小化できます。また、それぞれにスクリプトが 1 つのみ含まれるため、ファンクションおよびパッケージのスクリプトを 1 つ作成できます。最後に、シノニムおよびビューのスクリプトを作成します。

これは非常に単純なデプロイメントの一例です。実際のアプリケーションでは、データベース設計者と話し合い、オブジェクトの作成順序を計画する必要があります。Entity Relationship Diagram などの設計のダイアグラムがあると、このフェーズで非常に役立ちます。

データベース・オブジェクトのエクスポート

この項では、データベース・オブジェクトのエクスポート方法を説明します。

SQL Developer を使用したデータベース・オブジェクトのエクスポート

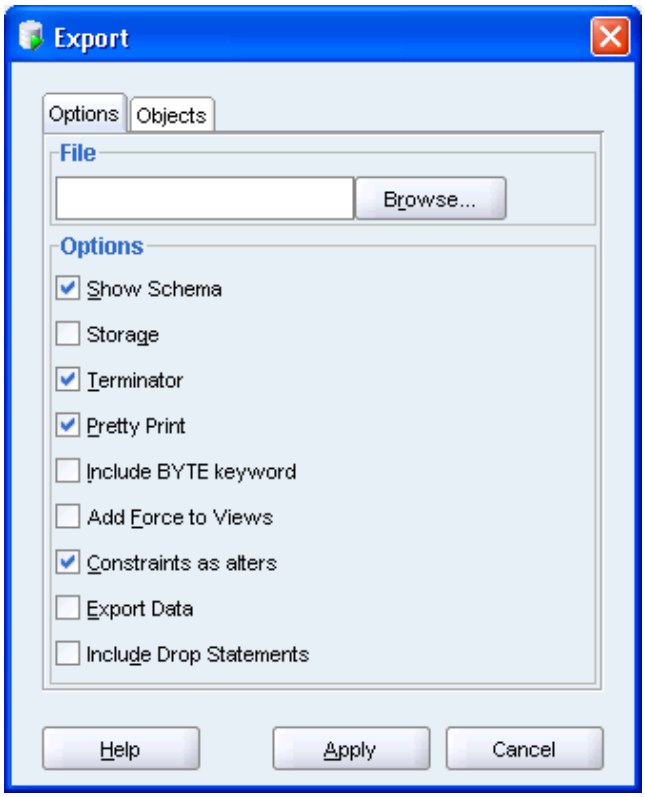
データベース・オブジェクトを作成するためにスクリプトを保持している場合、それらのスクリプトを使用できます。スクリプトを保持していない場合、データベースの各オブジェクトの定義に基づいたそれぞれのオブジェクトのデータ定義言語（DDL）を生成する必要があります。データベース・オブジェクトに DDL を生成するには、Oracle SQL Developer、特に DDL（およびデータ）のエクスポート機能を使用できます。この機能では、指定したオブジェクトおよびオブジェクトのタイプを作成する DDL 文が生成されます。また、作成した新しい表へエクスポートされたデータを挿入する `INSERT` 文を生成できます。

DDL 文および表データをエクスポートするには、次の手順を実行します。

1. DDL 文および表データをエクスポートするディレクトリを作成します。

   Oracle インストール・ディレクトリとは別にこのディレクトリを作成します。たとえば、`C:¥my_exports` です。

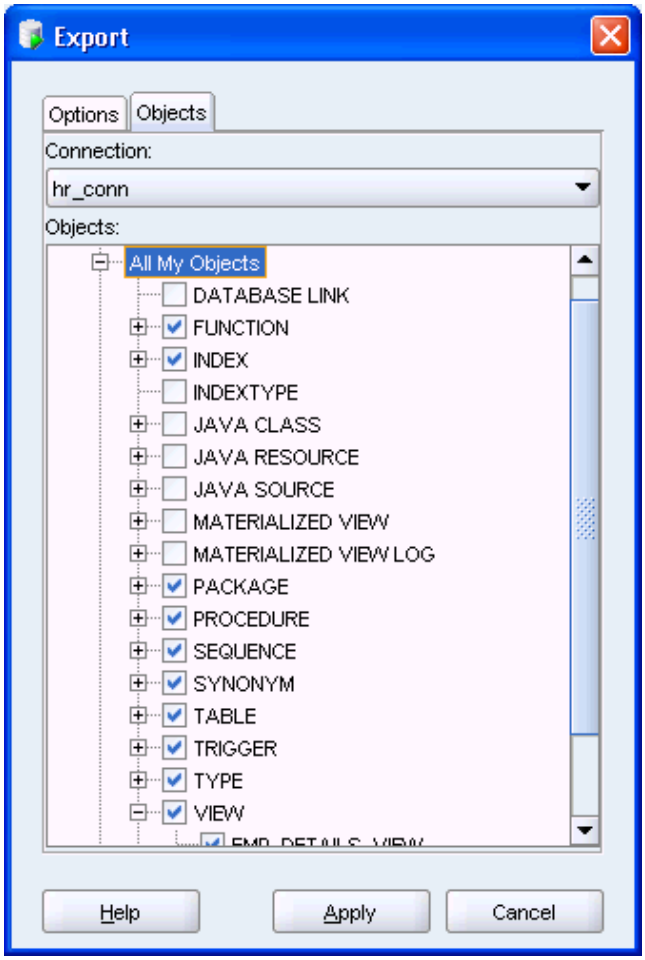
2. SQL Developer のメイン・メニューから、「**Tools**」を選択し「**Export DDL (and Data)**」を選択します。



3. **「Options」**タブをクリックします（デフォルトで選択する必要があります）。
4. 「File」フィールドで、オブジェクトを作成し、データを挿入する SQL 文を含むように作成されたエクスポート・ファイルの名称および位置を指定します。たとえば、`C:¥my_exports¥hr_export.sql` です。
5. 「Options」で、次からオプションを選択し、オブジェクト・タイプ内のオブジェクトを指定するか、生成した SQL 文に対するオプションを指定します。
   - **Show Schema:** このオプションを選択した場合、スキーマ名には `CREATE` 文が含まれます。このオプションを選択しない場合、スキーマ名には `CREATE` 文は含まれません。このオプションはエクスポートされたオブジェクトを異なる名称を持つスキーマで再作成する場合に便利です。
   - **Storage:** このオプションを選択した場合、データベース・オブジェクトの定義の `STORAGE` 句すべてがエクスポートされた DDL 文で保存されます。現行の記憶域定義を使用しない場合（異なるシステム環境でオブジェクトを再作成する場合など）、このオプションの選択を解除します。
   - **Terminator:** このオプションを選択した場合、行の終了記号文字は各行の最後に挿入されます。
   - **Pretty Print:** このオプションを選択した場合、文は出力ファイルで見やすく整形され、ファイルのサイズは元のサイズよりも大きくなります。
   - **Include BYTE Keyword:** このオプションを選択した場合、列長の指定はバイトを参照します。このオプションを選択しない場合、列長の指定は文字を参照します。
   - **Add Force to Views:** このオプションを選択した場合、`FORCE` オプションがすべての `CREATE VIEW` 文に追加され、各ビューにエラーが含まれる場合でもビューが作成されます。
   - **Constraints as Alters:** このオプションを選択した場合、各表の制約は `CREATE TABLE` 文で定義されるかわりに、個別の `ALTER TABLE` 文で定義されます。
   - **Export Data:** このオプションを選択した場合、文はエクスポートされた表またはビューに対してデータを挿入するために含まれます。このオプションを選択しない場合、文はエクスポートされた表またはビューに対してデータを挿入するために含まれることはありません。そのため、DDL 文のみが含まれます。
   - **Include Drop Statements:** このオプションを選択した場合、`CREATE` 文の前に `DROP` 文が含まれ、同じ名前を使用してすべての既存のオブジェクトを削除します。
6. **「Objects」**タブをクリックします。

7. 「**Objects**」タブで、次の手順を実行します。
   - 「**Connection**」リストから hr_conn を選択します。
   - 「Objects」で、「All」を選択し、「All My Objects」を選択して hr_conn 接続で使用可能なオブジェクトを表示します。

     エクスポートするオブジェクトのタイプ（制約、データベース・リンク、ファンクションなど）が選択されていることを確認します。INSERT 文を使用して表データを挿入する場合、「Data」が選択されていることを確認します。特定のオブジェクト・タイプまたは表データをエクスポートしない場合、該当するオプションの選択を解除します。



8. 「**Apply**」をクリックしてスクリプトを生成します。

順序およびトリガーのエクスポートに関する特殊な考慮事項

順序およびトリガーをエクスポートする場合、特殊な考慮事項が要求されます。順序に関して、生成された DDL は現行の値に関連する順序を開始します。表をロードする主キーおよびデータの移入に使用されている順序がある場合、その順序を保持します。ただし、データをロードしない場合、スクリプトの作成後にスクリプトを編集して、START WITH 値をリセットできます。

トリガーに関して、前に挿入するトリガーが表に存在していて、データのロードを計画する場合、トリガーを確認し、指定したアクションをトリガーで発生させるどうかを決定する必要があります。たとえば、主キー値は高い頻度でトリガーによって移入されます。INSERT 文から主キーを保護するには、主キー値が NULL のときにのみ、トリガーによって主キーが移入されるようにします。次に例を示します。

```
IF :new.evaluation_id IS NULL
   THEN SELECT evaluations_seq.Nextval
          INTO :new.evaluation_id
          FROM dual;
END IF;
```

ただし、トリガーを前述の例のように指定しない場合、データをロードする前にトリガーを指定しなおすか、トリガーを無効にして、データのロードの完了後にトリガーを有効にする必要があります。また、順序の現行の値が、主キー列の最高値よりも大きいことを確認します。

監査列（CREATED_ON または CREATED_BY など）がトリガーによって移入された場合、現行の値が NULL である場合のみに使用するトリガーを新しい値に設定することでソース表からデータを保護する必要があります。次に例を示します。

```
if :new.created_on is null
   then :new.created_on := sysdate;
end if;
```

このガイドの説明を使用して作成した表ではどのトリガーも使用されない場合、スクリプトの生成に関する関連項目の説明に従って生成されたコードを編集する必要はありません。

順序および表の作成に関するスクリプトの生成

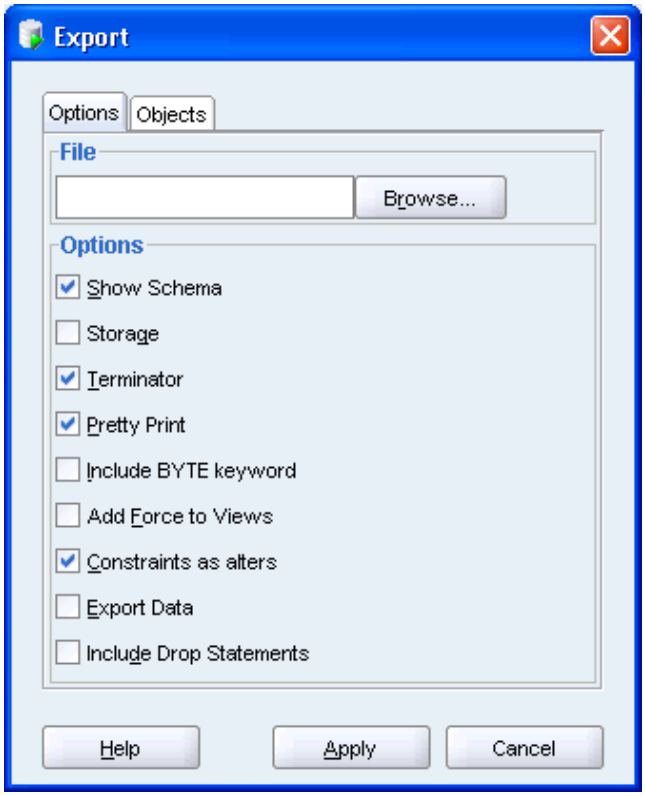
このガイドの前の項の操作に従った場合には作成済の順序および表を作成するスクリプトを生成します。このスクリプトによって、必要な制約、索引およびトリガーも作成されます。

順序および表の作成に関するスクリプトを生成するには、次の手順を実行します。

1. DDL 文および表データをエクスポートするディレクトリを作成します。

   Oracle インストール・ディレクトリとは別にこのディレクトリを作成します。たとえば、C:¥my_exports です。

2. SQL Developer のメイン・メニューから、「**Tools**」を選択し「**Export DDL (and Data)**」を選択します。



3. 「Export」ダイアログ・ボックスで、オブジェクトを作成し、データを挿入する SQL 文を含むように作成されたエクスポート・ファイルの名前および位置を指定します。たとえば、`C:¥my_exports¥2day_tables.sql` です。
4. この例の場合の表の作成に関して、「Options」で、次のオプションを指定します。
   - **Show Schema:** 選択を解除します。この例では、テスト環境または本番環境のスキーマ名が開発環境と異なる可能性があるため、このオプションの選択が不適切である場合があります。
   - **Storage:** 選択を解除します。保持する表に対して特定の記憶域句を指定した場合、このオプションを確認します。ただし、記憶域句を指定しなかった場合、または（本番環境とは異なる）開発環境またはテスト環境に固有の記憶域句を指定した場合は、このオプションの選択を解除します。
   - **Terminator:** 選択します。これは結果のスクリプトを実行するために必要です。
   - **Pretty Print:** 選択します。このオプションを選択した場合、出力がより読みやすくなります。
   - **Include BYTE Keyword:** 選択を解除します。マルチバイト・キャラクタ・セットを使用して作業をしていない場合は、このオプションを選択する必要があります。
   - **Add Force to Views:** この例では別のスクリプトへビューが追加されるため、選択を解除します。
   - **Constraints as Alters:** 選択します。各 `CREATE TABLE` の後の別の SQL 文に制約を追加します。

- **Export Data:** 選択を解除します。この例ではデータを個別にエクスポートするため選択を解除します。
- **Include Drop Statements:** 選択を解除します。同じ名前を持つ既存のオブジェクトをこれらのオブジェクトに置換する場合に、DROP 文を含めることができます。ただし、より実践的な方法として、表を作成する前に、オブジェクトの古いバージョンを削除できる別の削除スクリプトを含めます。これによって削除する予定のないオブジェクトを誤って削除してしまうことを回避します。

5. **「Objects」**タブをクリックします。
6. 「Objects」タブで、次の手順を実行します。
   - 使用するデータベース接続で `hr_conn` を選択します。
   - エクスポートするオブジェクトのタイプでは、**すべてのオブジェクト**を展開します。次に、**「Sequences」**を展開して **EVALUATIONS_SEQ** を選択します。そして、**「Tables」**を展開して **EVALUATIONS**、**PERFORMANCE_PARTS** および **SCORES** を選択します。他のオブジェクト・タイプの選択を解除します。
7. **「Apply」**をクリックしてスクリプトを生成します。

オブジェクト定義は、オブジェクト・タイプによってアルファベット順でファイルに追加されます。順序は作成される最初のオブジェクトです。順序は `evaluations` のトリガーによって参照されるため、便利です。表の依存性には、最初に `evaluations` を作成し、次に `performance_parts`、そして `scores` を作成することが必要です。これらは、アルファベット順で発生するため、変更を加える必要はありません。ただし、変更が必要な場合、テキスト・エディタを使用するか、SQL Developer 内で生成したスクリプトをオープンして、生成したスクリプトを編集することは可能です。

PL/SQL オブジェクトの作成に関するスクリプトの生成

このガイドのこのガイドの前の項の操作に従った場合には作成済のパッケージ（ファンクションを含む）を作成するスクリプトを生成します。

PL/SQL オブジェクトの作成に関するスクリプトを生成するには、次の手順を実行します。

1. SQL Developer のメイン・メニューから、**「Tools」**を選択し**「Export DDL (and Data)」**を選択します。
2. 「Export」ダイアログ・ボックスで、オブジェクトを作成し、データを挿入する SQL 文を含むように作成されたエクスポート・ファイルの名前および位置を指定します。たとえば、`C:¥my_exports¥2day_plsql.sql` です。
3. この例の場合の表の作成に関して、「Options」で、次のオプションを指定します。
   - **Show Schema:** 選択を解除します。この例では、テスト環境または本番環境のスキーマ名が開発環境と異なる可能性があるため、このオプションの選択が不適切である場合があります。
   - **Storage:** 選択を解除します。ここでは適用されないため、選択を解除します。
   - **Terminator:** 選択します。これは結果のスクリプトを実行するために必要です。
   - **Pretty Print:** 選択します。このオプションを選択した場合、出力がより読みやすくなります。
   - **Include BYTE Keyword:** 選択を解除します。マルチバイト・キャラクタ・セットを使用して作業をしていない場合は、このオプションを選択する必要があります。
   - **Add Force to Views:** 選択を解除します。ここでは適用されないため、選択を解除します。
   - **Constraints as Alters:** 選択を解除します。ここでは適用されないため、選択を解除します。

- **Export Data:** 選択を解除します。この例ではデータを個別にエクスポートするため選択を解除します。
- **Include Drop Statements:** 選択を解除します。同じ名前を持つ既存のオブジェクトをこれらのオブジェクトに置換する場合に、DROP 文を含めることができます。ただし、より実践的な方法として、表を作成する前に、オブジェクトの古いバージョンを削除できる別の削除スクリプトを含めます。これによって削除する予定のないオブジェクトを誤って削除してしまうことを回避します。

4. **「Objects」**タブをクリックします。
5. 「Objects」タブで、次の手順を実行します。
   - 使用するデータベース接続で `hr_conn` を選択します。
   - エクスポートされるオブジェクトのタイプでは、**All My Objects** を展開します。次に、**「Functions」**を展開して **CALCULATE_SCORE** を選択します。**「Packages」**を展開して **EMP_EVAL** を選択します。他のオブジェクト・タイプの選択を解除します。
6. **「Apply」**をクリックしてスクリプトを生成します。

シノニムおよびビューの作成に関するスクリプトの生成

このガイドのこのガイドの前の項の操作に従った場合には作成済シノニムおよびビューを作成するスクリプトを生成します。

シノニムおよびビューの作成に関するスクリプトを生成するには、次の手順を実行します。

1. SQL Developer のメイン・メニューから、**「Tools」**を選択し**「Export DDL (and Data)」**を選択します。
2. 「Export」ダイアログ・ボックスで、オブジェクトを作成し、データを挿入する SQL 文を含むように作成されたエクスポート・ファイルの名前および位置を指定します。たとえば、`C:¥my_exports¥2day_other.sql` です。
3. この例の場合の表の作成に関して、「Options」で、次のオプションを指定します。
   - **Show Schema:** 選択を解除します。この例では、テスト環境または本番環境のスキーマ名が開発環境と異なる可能性があるため、このオプションの選択が不適切である場合があります。
   - **Storage:** 選択を解除します。ここでは適用されないため、選択を解除します。
   - **Terminator:** 選択します。これは結果のスクリプトを実行するために必要です。
   - **Pretty Print:** 選択します。このオプションを選択した場合、出力がより読みやすくなります。
   - **Include BYTE Keyword:** 選択を解除します。マルチバイト・キャラクタ・セットを使用して作業をしていない場合は、このオプションを選択する必要があります。
   - **Add Force to Views:** 選択します。これによって無効の場合でも、ビューが作成されます。すべてのビューが無効の場合、後で問題を修正し、これらのビューをコンパイルできます。
   - **Constraints as Alters:** 選択を解除します。ここでは適用されないため、選択を解除します。
   - **Export Data:** 選択を解除します。この例ではデータを個別にエクスポートするため選択を解除します。
   - **Include Drop Statements:** 選択を解除します。同じ名前を持つ既存のオブジェクトをこれらのオブジェクトに置換する場合に、DROP 文を含めることができます。ただし、より実践的な方法として、表を作成する前に、オブジェクトの古いバージョンを削除できる別の削除スクリプトを含めます。これによって削除する予定のないオブジェクトを誤って削除してしまうことを回避します。

4. **「Objects」**タブをクリックします。
5. 「Objects」タブで、次の手順を実行します。
   - 使用するデータベース接続で hr_conn を選択します。
   - エクスポートされるオブジェクトのタイプでは、**All My Objects** を展開します。次に、**「Synonyms」**を展開して **POSITIONS** を選択します。**「Views」**を展開して **EMP_LOCATION** を選択します。他のオブジェクト・タイプの選択を解除します。
6. **「Apply」**をクリックしてスクリプトを生成します。

表、PL/SQL オブジェクト、シノニムおよびビューを作成するスクリプトを生成した後に、ターゲット・データベースに移行するデータすべてを取得するスクリプトを生成できます。

データのエクスポート

データをエクスポートするには、デプロイされた表に挿入するために既存の表データを取得する必要があります。7-2 ページの「デプロイメントの計画」で説明されているように、すべての依存データが存在し、妥当性の問題がないことを確信している場合は、ターゲット・スキーマにデータを挿入できます。または制約を無効にしてデータをロードし、ロードが完了した後、制約をもう一度有効にすることも可能です。

無効化を選択し、その後に制約を有効化した場合、次のオプションがあります。

- SQL Developer を使用し、表および制約を確認して、表および制約を同時に無効化および有効化します。
- 各制約を無効化および有効化する SQL 文を含むように、ファイルのコピーを作成し、各制約の名称を検索して、2day_tables.sql ファイルを編集します。
- 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。

  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、「SQL Worksheet」ウィンドウ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');
```

このガイドの説明に従った場合、データを追加した表は performance_parts のみです。DDL（およびデータ）のエクスポート機能を使用してデータをエクスポートしますが、この方法はデータとともに DDL を出力します。また、「Connections」ナビゲータの「Connections」にあるツリー内から表を選択し、右クリックして「Export Data」を選択した後、INSERT を選択します。このオプションを使用すると、選択した列へのエクスポートを制約でき、またどのデータをエクスポートするか制御する WHERE 句を含めることができます。これが使用する方法です。

データの INSERT 文を作成するには、次の手順を実行します。

1. SQL Developer の「Connections」ナビゲータで、データベース・オブジェクトのスクリプト（hr_conn）を生成するために使用したデータベース接続を展開します。
2. hr_conn 接続で**「Tables」**を展開します。
3. **PERFORMANCE_PARTS** 表の名称を右クリックし**「Export Data」**を選択して、**INSERT** を選択します。

4. 「Export」ダイアログ・ボックスの**「File」**フィールドで、C:¥my_exports¥2day_data.sql を入力してエクスポート・ファイル名を指定します。
5. **「Apply」**をクリックします。

他の表に INSERT 文を作成する必要がある場合、ファイルを区別する他の表のデータをエクスポートすることもできますが、クリップボードへデータをエクスポートして最初のファイルに貼り付けることもできます。これを行うには、出力の**「Clipboard」**を指定します。これによって文がクリップボードに置かれ、ファイルに追加できます。

インストールの実行

この段階で、別のスキーマでオブジェクトを作成するために必要なスクリプト・ファイルをすべて保持しています。これらのスクリプトは次の順序（この演習でスクリプトを作成した順序）に従って実行します。データをロードする前に表が存在することを確認します。

1. 2day_tables.sql
2. 2day_plsql.sql
3. 2day_other.sql
4. 2day_data.sql

必要に応じて、適切な順序でこれらのスクリプトを実行するマスター・スクリプトを作成して、ファイルに結果を記録できます。この種類のマスター・スクリプトは、SQL*Plus を使用して一般的に実行されます。この演習のマスター・スクリプトは次の例と同様です。

```
spool my_dir/create_log.txt
@my_dir/2day_tables.sql
@my_dir/2day_plsql.sql
@my_dir/2day_other.sql
@my_dir/2day_data.sql
commit;
spool off
```

作成したスクリプトを実行する SQL Developer も使用できます。マスター・スクリプトがファイル・パスおよびファイル名を完全に指定する場合（たとえば、C:¥my_dir¥2day_tables.sql）、マスター・スクリプトをオープンおよび実行できます。また、各スクリプトを個々にオープンおよび実行できます。

SQL Developer で、インストール・スクリプトを実行するには、次の手順を実行します。

1. 「SQL Worksheet」ウィンドウで右クリックして**「Open File」**を選択します。
2. **C:¥my_exports** ディレクトリに位置する 2day_tables.sql ファイルを検索およびオープンします。

   表の作成のための DDL 文を表示します。次の手順では、各文の結果が成功か失敗かを監視するためにスクリプトとしてこれらの文を実行します。
3. **「Run Script」**アイコンをクリックするか、キーボード上の **[F5]** を押します。

   各文の結果は、「Script Output」ペインに表示されます。各文が正常に実行された結果が表示されます。
4. **「Clear」**アイコンをクリックして SQL ワークシートの内容を消去します。
5. 次に示す追加の各スクリプト・ファイル対して前述の 3 つの手順（ファイルの検索およびオープン、スクリプトとして表の内容の実行、SQL ワークシート・エントリ領域のクリア）を実行します。
   - **2day_plsql.sql**
   - **2day_other.sql**

6. **2day_data.sql** ファイルを検索およびオープンします。

   表データの挿入のための DDL 文を表示します。次の手順では、各文が成功したか、または失敗したかを監視するためにスクリプトとしてこれらの文を実行します。

7. **「Run Script」**アイコンをクリックします（または [F5] を押します）。

   Oracle では、DML 文は自動的にコミットされません。これは `INSERT`、`UPDATE` または `DELETE` 文をロールバックできるという意味です。データベースにデータを格納するには、次の手順で実行するトランザクションのコミットが必要になります。

8. **「Commit」**アイコンをクリックしてデータベースにデータをコミットします。

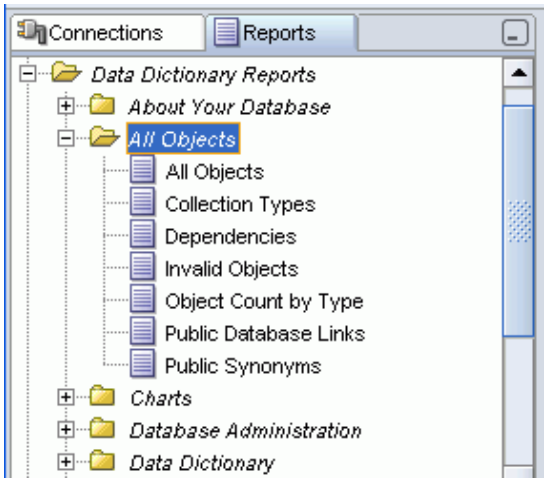
インストールの検証

アプリケーションをサポートするすべてのデータベース・オブジェクトを作成した後、SQL Developer を使用して新規オブジェクトの定義にアクセスできます。また、SQL Developer レポートを使用してインストールが有効かどうかの判断に役立つ情報を確認できます。これらのレポートには次の項目が含まれます。

- **All Objects:** 各オブジェクトでは、所有者、名称、タイプ（表、ビュー、索引など）、ステータス（有効か無効か）、作成日、データ定義言語（DDL）オペレーションが最後に実行された日がリストされます。DDL の最新の日付は、特定の時間またはその後にオブジェクト定義に変更が加えられたかどうかを検索するのに便利です。
- **Invalid Objects:** 無効のステータスを持つすべてのオブジェクトがリストされます。
- **Object Counts by Type:** 特定の所有者と関連付けされた各オブジェクトのタイプでは、オブジェクトの数がリストされます。このレポートは 2 つのスキーマが一致するかチェックするのに便利です。また、特に多くの数のオブジェクトおよび特定のタイプのオブジェクトを作成したユーザーを識別するのにも便利です。

インストールの有効性をチェックするレポートを表示するには、次の手順を実行します。

1. SQL Developer の「Reports」ナビゲータで、**「Data Dictionary Reports」**を展開した後に**「All Objects」**を展開します。



2. 「All Objects」、「Invalid Objects」および件数別オブジェクト・タイプなどを表示する各レポートをクリックします。

   指定した各レポートで、バインド変数を要求される際に（表示を制限しない場合は）、使用するデータベース接続を選択して**「Apply」**をクリックします。

インストール・スクリプトのアーカイブ

データベース・アプリケーションのデプロイおよびインストールの検証後、アプリケーションのインストール・スクリプトを作成した場合、作成時および各スクリプトの目的を記したコメントを使用して、ソース・コード制御システムでインストール・スクリプトのアーカイブを検討します。このアーカイブは、別の環境へデプロイする必要がある際に便利です。そのスクリプトを使用してクリーンなインストールを再作成できます。データをアーカイブするには、Oracle Data Pump を使用できます。

参照:

- 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。

索引

記号

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

Z

あ

い

え

お

か

き

く

け

こ

さ

し

す

せ

そ

て

と

な

ね

は

ひ

ふ

へ

ほ

ま

み

め

ゆ

ら

り

れ

ろ