Oracle® Database

2 日で Java 開発者ガイド

11*g* リリース 1（11.1）

部品番号 : E05692-02

2007 年 11 月

ORACLE®

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

部品番号 : E05692-02

原本名 : Oracle Database 2 Day + Java Developer's Guide, 11*g* Release 1 (11.1)

原本部品番号 : B28765-02

原本著者 : Deepa Aswani、Rosslynne Hefferan、Maitreyee Chaliha

原本協力者 : Kathleen Heap、Simon Law、Kuassi Mensah、Chris Schalk、Christian Bauwens、Mark Townsend、Paul Lo、Venkatasubramaniam Iyer

目次

3 Oracle Database への接続

4 データの問合せおよび表示

5 データの更新

6 アプリケーションの拡張 : 拡張 JDBC 機能

索引

例一覧

図一覧

表一覧

はじめに

ここでは、このマニュアルの対象読者およびこのマニュアルで使用される表記規則について説明します。また、より詳しい情報を参照できる関連ドキュメントのリストも記載しています。

対象読者

このマニュアルは、Java を使用して Oracle Database のデータにアクセスし、変更するアプリケーション開発者を対象にしています。このマニュアルでは、単純な Java Database Connectivity（JDBC）アプリケーションを使用してこれらの作業を実行する方法について説明します。また、このマニュアルでは、アプリケーションの作成に Oracle JDeveloper 統合開発環境（IDE）を使用しています。このマニュアルは、Java プログラミングに関心のあるすべての読者を対象としていますが、少なくとも次の項目について事前に理解していることを前提としています。

- Java
- Oracle PL/SQL
- Oracle Database

ドキュメントのアクセシビリティについて

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

ドキュメント内のサンプル・コードのアクセシビリティについて

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

外部 Web サイトのドキュメントのアクセシビリティについて

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

Oracle サポート・サービスへの TTY アクセス

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

関連ドキュメント

詳細は、次の Oracle ドキュメントを参照してください。

- 『Oracle JDeveloper インストレーション・ガイド』および次の URL の OTN（Oracle Technology Network）にある JDeveloper のオンライン・ドキュメント

  ```
  http://www.oracle.com/technology/documentation/jdev.html
  ```

- 『Oracle Database JDBC 開発者ガイドおよびリファレンス』
- 『Oracle Database Java 開発者ガイド』

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、操作に関連付けられている Graphical User Interface あるいは本文中または用語集で定義されている用語を示します。
イタリック体	イタリック体は、特定の値を指定する必要があるプレースホルダや変数を示します。
固定幅フォント	固定幅フォントは、段落内のコマンド、URL、コード例、画面上に表示されるテキストまたはユーザーが入力するテキストを示します。

サポートおよびサービス

次の各項に、各サービスに接続するための URL を記載します。

Oracle サポート・サービス

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

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

製品マニュアル

製品のマニュアルは、次の URL にあります。

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

研修およびトレーニング

研修に関する情報とスケジュールは、次の URL で入手できます。

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

その他の情報

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

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

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

1

Oracle Database での Java の使用

Oracle Database は、データの格納、使用および変更に使用できるリレーショナル・データベースです。Java アプリケーションでは、リレーショナル・データベースのデータのアクセスおよび操作に、Java Database Connectivity（JDBC）標準が使用されます。

JDBC は、Sun 社が開発した業界標準の Application Program Interface（API）です。JDBC を使用すると、Java コードに SQL 文を埋め込むことができます。JDBC は、X/Open SQL Call Level Interface（CLI）に基づいており、SQL92 Entry Level 規格に準拠しています。Oracle などの各ベンダーは、標準 `java.sql` パッケージのインタフェースを実装することによって、JDBC 実装を作成しています。

関連項目：

- `http://java.sun.com/javase/technologies/database/index.jsp`

このマニュアルでは、簡単な Java アプリケーションを使用し、Oracle Database に接続してデータベースのデータにアクセスおよび変更する方法について説明します。また、従業員データを表示するためのマスター・ディテール・アプリケーションの開発には、Oracle Application Development Framework（ADF）が使用されます。

この章では、このマニュアルで作成される Java アプリケーションの概要、および Java アプリケーションの開発に使用できるツールについて説明します。この章の内容は、次のとおりです。

Java を使用した Oracle Database への接続

JDBC はデータベース・アクセス・プロトコルです。JDBC を使用してデータベースに接続し、データベースに対して SQL 文を実行して問合せを行うことができます。コアの Java クラス・ライブラリには、JDBC API、`java.sql` および `javax.sql` が用意されています。ただし、JDBC は、ベンダーが特定のデータベースに対して必要な特別機能を持つドライバを提供できるように設計されています。

注意： Oracle Database 11*g* リリース 1 では、JDK 5 以上がサポートされています。今回のリリースの JDBC サポートには、`ojdbc5.jar` ファイルおよび `ojdbc6.jar` ファイルが含まれています。`ojdbc6.jar` ファイルによって、JDBC 4.0 との整合性が提供されます。このファイルを使用するには、JDK 6 が必要です。

Oracle Database では、JDBC Thin ドライバ、Oracle Call Interface（OCI）ドライバおよび `oracle.sql` パッケージと `oracle.jdbc` パッケージを使用したクライアント側アプリケーションの開発がサポートされています。これらのパッケージのクラスおよびインタフェースによって、JDBC 標準を拡張します。これにより、Oracle のデータ型にアクセスして変更したり、JDBC 用の Oracle パフォーマンス拡張を Java アプリケーションで非常に柔軟に使用できるようになります。

次の項では、JDBC 標準に対する Oracle サポートについて説明します。

- Oracle JDBC Thin ドライバ
- Oracle JDBC OCI ドライバ
- Oracle JDBC パッケージ

関連項目：

- 『Oracle Database JDBC 開発者ガイドおよびリファレンス』
- 『Oracle Database Java 開発者ガイド』

Oracle JDBC Thin ドライバ

ほとんどの場合、JDBC Thin ドライバを使用することをお薦めします。JDBC-OCI は、OCI 固有の機能の場合にのみ必要となります。

JDBC Thin ドライバは、Pure Java の Type IV ドライバです。JDBC Thin ドライバでは、Java™ 2 Platform Standard Edition 5.0（Java Development Kit（JDK）5 とも呼ばれる）がサポートされています。また、JDK 6 に対するサポートも含まれています。JDBC Thin ドライバはプラットフォームに依存していないため、クライアント側アプリケーションの開発にその他の Oracle ソフトウェアを必要としません。JDBC Thin ドライバは、SQL*Net を使用してサーバーと通信を行い、Oracle Database にアクセスします。

JDBC Thin ドライバを使用すると、Oracle ネットワーク・プロトコルの Pure Java 実装（Two-Task Common（TTC プロトコル）および SQL*Net）を提供することによって、データベースへの直接接続が可能になります。ドライバでは、TCP/IP プロトコルがサポートされており、データベース・サーバーの TCP/IP ソケットに Transparent Network Substrate（TNS）リスナーが必要です。Thin ドライバは、適切な Java 仮想マシン（Java Virtual Machine: JVM）があるマシンであれば、どのマシン上でも機能します。

Oracle 固有の JDBC 機能および標準機能にアクセスするには、`oracle.jdbc` パッケージを使用します。

Oracle JDBC OCI ドライバ

JDBC OCI ドライバは、Java アプリケーションで使用される Type II ドライバです。JDBC OCI ドライバを使用するには、Oracle クライアント・インストールが必要です。JDBC OCI ドライバでは、プロセス間通信（IPC）、Named Pipes、TCP/IP、InternetworkPacket Exchange/Sequenced Packet Exchange（IPX/SPX）など、インストールされているすべての Oracle Net アダプタがサポートされています。

OCI は API の 1 つであり、OCI を使用すると、ネイティブ・プロシージャまたはファンクション・コールを使用するアプリケーションを作成できます。JDBC OCI ドライバは Java および C を組み合せて記述されており、JDBC コールを OCI へのコールに変換します。変換は、C エントリ・ポイントをコールするネイティブ・メソッドを使用して行われます。これらのコールは、SQL*Net を使用してデータベースとの通信を行います。

Oracle JDBC パッケージ

JDBC API に対する Oracle サポートは、`oracle.jdbc` パッケージおよび `oracle.sql` パッケージによって提供されています。これらのパッケージでは、Java Development Kit（JDK）リリース 1.5 から 1.6 がすべてサポートされています。

oracle.sql

`oracle.sql` パッケージにより、SQL 形式のデータへの直接アクセスがサポートされます。このパッケージは、主に、SQL データへの Java マッピングを提供するクラスとそれらのサポート・クラスによって構成されます。基本的に、クラスは、SQL データの Java ラッパーとして機能します。キャラクタは、Java `chars` に変換されてから、UCS-2 キャラクタ・セットのバイトに変換されます。それぞれの `oracle.sql.*` データ型クラスによって、`oracle.sql.Datum`（すべてのデータ型に共通したファンクションと機能を含むスーパークラス）が拡張されます。一部のクラスは、JDBC 2.0 準拠データ型用です。データ型クラス以外に、`oracle.sql` パッケージでは、オブジェクトおよびコレクションで使用するクラスとインタフェースがサポートされています。

oracle.jdbc

`oracle.jdbc` パッケージのインタフェースでは、`java.sql` パッケージのインタフェースに対する Oracle の拡張機能が定義されます。これらの拡張機能によって、Oracle SQL 形式のデータにアクセスできるようになります。また、その他の Oracle 固有の機能（Oracle のパフォーマンス拡張など）にアクセスできるようにもなります。

このパッケージの主要なクラスとインタフェースでは、標準の JDBC 機能をサポートし、次のような処理を実行するメソッドがあります。

- Oracle の Statement オブジェクトを返します。
- すべての文の Oracle パフォーマンス拡張を設定します。
- `oracle.sql.*` 型を準備済のコール可能文にバインドします。
- `oracle.sql` 形式でデータを取得します。
- データベースおよび結果セットに関するメタ情報を取得します。
- SQL の型の識別に使用される整定数を定義します。

関連項目：『Oracle Database JDBC 開発者ガイドおよびリファレンス』

JDeveloper を使用した JDBC アプリケーションの作成

このマニュアルの Java アプリケーション・チュートリアルでは、Oracle JDeveloper 10*g* リリース 10.1.3 を統合開発環境（IDE）として使用し、Java アプリケーションの開発、およびユーザーがデータを表示して変更するための Web ページの作成を行います。

Oracle JDeveloper は、Java アプリケーションおよび Web サービスのモデリング、開発、デバッグ、最適化およびデプロイをサポートする IDE です。

JDeveloper には、Java プログラムに埋め込まれた SQL 文を使用してデータベースにアクセスする Java プログラムを作成し、テストするための機能が備えられています。JDeveloper には、データベースに関して次の処理を行うファンクションおよび機能が備えられています。

- データベース接続の作成
- データベース・オブジェクトの表示
- データベース・オブジェクトの作成、編集または削除
- PL/SQL ファンクション、プロシージャおよびパッケージの作成および編集

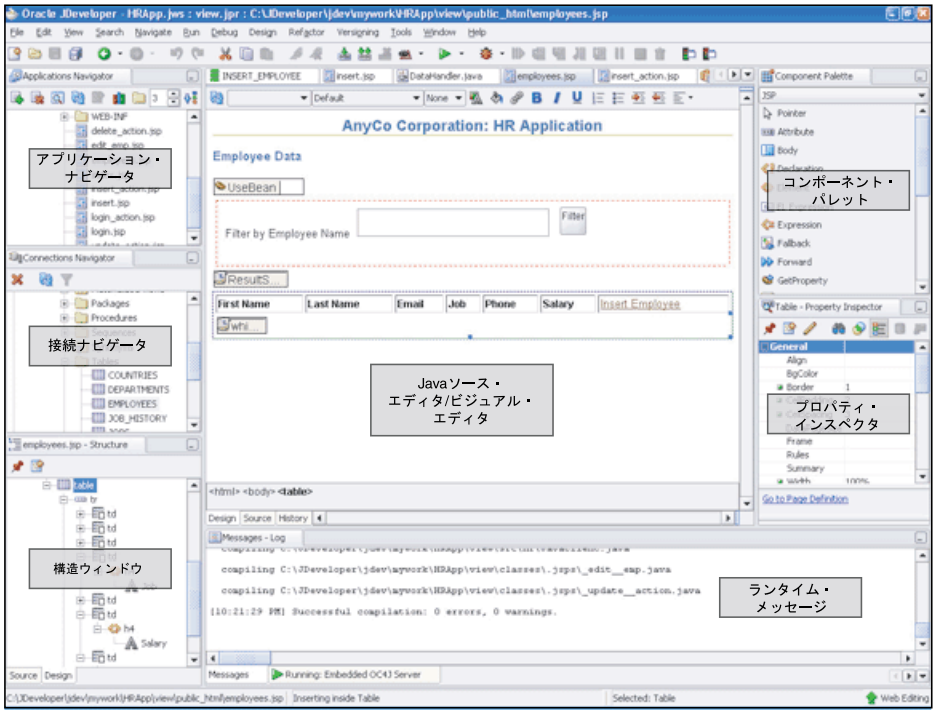
JDeveloper ユーザー・インタフェース

Oracle JDeveloper は、様々なアプリケーション開発ツール用のウィンドウを使用する IDE です。ウィンドウの表示 / 非表示を切り替えたり、ウィンドウをドッキング / ドッキング解除して、作業方法に合ったデスクトップを作成できます。

これらのツール以外に、JDeveloper には、プロジェクトの内容を分類し表示できる多くのナビゲータが備えられています。アプリケーション・ナビゲータおよびシステム・ナビゲータには、プロジェクトのファイルが表示され、構造ウィンドウには、個別の項目の構造が表示されます。

ウィンドウは、任意に配置し、「表示」メニューから開いたり終了することができます。図 1-1 に、JDeveloper ユーザー・インタフェース（GUI）で使用可能なナビゲータ、パレットおよび作業領域のデフォルトのレイアウトを示します。

図 1-1 JDeveloper ユーザー・インタフェース



関連項目： JDeveloper オンライン・ヘルプの IDE でのウィンドウの処理に関する項を参照してください。

JDeveloper ツール

JDeveloper では、次のツールを使用して Java アプリケーションを簡単に作成できます。

- **構造ウィンドウ**: ツリー・ビューに、現在編集中（Java、XML または JSP/HTML）のアプリケーションのすべての要素が表示されます。
- **Java ビジュアル・エディタ**: ユーザー・インタフェースの要素を迅速かつ簡単に組み合わせて配置できます。
- **JSP/HTML ビジュアル・エディタ**: HTML および JSP ページを視覚的に編集する場合に使用できます。
- **Java ソース・エディタ**: Java コードを作成する場合に役立つ豊富な機能があります。このような機能には、構文およびセマンティック・エラーをわかりやすくするハイライト、インポート文を追加およびソートする場合の支援、Java コード・インサイト機能、コード・テンプレートがあります。

注意： Java コード・インサイト機能は、Java ソース・エディタでコードを作成する場合にコンテキスト固有のインテリジェント入力を提供する機能です。このマニュアルでは、Java コード・インサイトを使用したコードの挿入方法が数多く記載されています。

- **コンポーネント・パレット**: ページ上に表示するボタンやテキスト領域など、ユーザー・インタフェース・コンポーネントを選択します。
- **プロパティ・インスペクタ**: ユーザー・インタフェース・コンポーネントなどの項目のプロパティを簡単に設定できます。

JDeveloper UI 内でのこれらのツールへのアクセス方法についてより理解するには、図 1-1 を参照してください。

サンプル Java アプリケーションの概要

このマニュアルでは、Java、JDBC および Oracle ADF を使用してアプリケーションを作成する方法について説明します。このアプリケーションには、次のファンクションおよび機能を組み込みます。

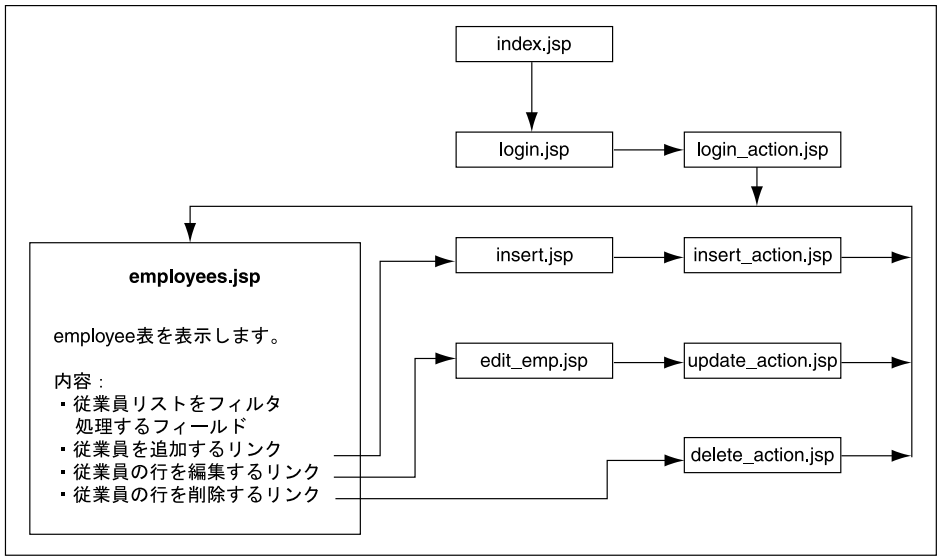
1. ユーザーがログインしたり、ユーザー名およびパスワードを検証できるようにします。
2. データベース接続を確立します。
3. データベースでデータを問い合せ、JavaBean を使用してデータを取得します。
4. JavaServer Pages（JSP）テクノロジを使用して、データを表示します。
5. ユーザーがレコードを挿入、更新または削除できるようにします。
6. マスター・ディテール・アプリケーションの情報にアクセスし、変更します。
7. 例外を処理します。

注意： アプリケーションは、Oracle Database に付属の `HR` スキーマに接続します。Oracle Database のクライアント・インストールには Thin ドライバと OCI ドライバの両方が付属していますが、サンプル・アプリケーションでは、JDBC Thin ドライバのみが使用されます。

アプリケーション Web ページ（JSP ページ）の概要

図 1-2 に、このアプリケーションに対して開発されるページ間の関係を示します。

図 1-2 サンプル・アプリケーションの Web ページ



サンプル・アプリケーションの Web ページの概要は、次のとおりです。

- `index.jsp`

  アプリケーションの開始ページです。アプリケーションのログイン・ページ `login.jsp` が自動的に表示されます。

- `login.jsp`

  このページで、ユーザーはアプリケーションにログインできます。ユーザー名、パスワードおよびホスト情報は、検証され、データベースにログインするための接続記述子の作成に使用されます。

- `login_action.jsp`

  `login.jsp` でユーザーが入力したログイン情報の認証を処理する非表示のページです。認証が成功すると、ユーザーに `employees.jsp` が表示されます。それ以外の場合は、メッセージが含まれる `login.jsp` ページが表示されます。

- `employees.jsp`

  アプリケーションのメイン・ページです。AnyCo Corporation の HR スキーマのすべての従業員のリストが表示されます。ユーザーは、任意の文字列を使用して、従業員リストをフィルタできます。また、このページには、ユーザー・データを追加、編集および削除するリンクが含まれています。ただし、これらのアクションは、それぞれのアクション専用に作成されたその他の JSP ページで処理されます。

- `insert.jsp`

  `employees.jsp` ページ上にある従業員データを挿入するリンクを選択すると、このページが表示されます。このページには、新しい従業員レコードのすべての詳細が入力されるフォームが含まれています。このフォームで入力された詳細は、`insert_action.jsp` ページによって処理されます。

- `insert_action.jsp`

  `insert.jsp` ページで入力された新しい従業員のデータの挿入を処理する非表示のページです。

- edit.jsp

  employees.jsp ページ上にある従業員データを編集するリンクを選択すると、このページが表示されます。このフォームのテキスト・フィールドには、1 人の従業員の現在のデータが表示され、ユーザーはこの情報を編集することができます。

- update_action.jsp

  edit.jsp ページでの送信アクションによって、データがこの非表示のページに送信されます。このページによって、編集済データがデータベースに挿入されます。

- delete_action.jsp

  employees.jsp ページ上にある従業員レコードを削除するリンクは、この非表示のページによって処理されます。このページでは、従業員データが削除され、再度 employees.jsp ページが表示されます。

クラス

サンプル・アプリケーションには、次のクラスが含まれています。

- DataHandler.java

  このクラスには、サンプル・アプリケーションの重要なファンクションの実装に使用されるメソッドがすべて含まれています。このクラスには、ユーザー資格証明の検証、データベースへの接続、従業員データの取得（フィルタあり / なし）、データの挿入、データの更新、例外の処理などを行うメソッドが含まれています。

- Employees.java

  このクラスは、単一の従業員レコードを保持する JavaBean です。このクラスには、各レコード・フィールドの値を取得して設定するアクセッサ・メソッドが含まれています。また、従業員レコードを取得して変更するアクセッサ・メソッドも含まれています。

- JavaClient.java

  このクラスは、DataHandler クラスをテストする場合にのみ使用されます。

注意： このアプリケーションは、このマニュアル全体をとおしてチュートリアルの形式で開発していきます。そのため、このマニュアルは、章の順番に読むことをお薦めします。

開発者フレームワークを使用した高度なアプリケーション開発

Web、ワイヤレス、デスクトップまたは Web サービス・インタフェースを使用してデータを検索、表示、作成、変更および検証するエンタープライズ・ソリューションを開発する場合、作業を簡素化するには開発者フレームワークを使用する必要があります。

フレームワークを使用すると、開発者は、明確に定義されたインタフェースに基づいてコードを作成できます。これにより、大幅に時間を節約できます。また、Java EE フレームワークにはエンタープライズ・アプリケーションに必要なインフラストラクチャが備えられているため、Java EE 環境という点においても有益です。つまり、Java EE フレームワークによって、Java EE デザイン・パターンに表現される概念をより具体化します。

Oracle Application Development Framework（Oracle ADF）は、Java EE 標準およびオープンソース・テクノロジに基づくエンドツーエンド・アプリケーション・フレームワークであり、サービス指向アプリケーションの実装を簡素化および促進します。

機能の豊富な環境を使用することにより、複雑なアプリケーションをどのように簡単に作成できるようになるかについては、第 7 章のマスター・ディテール・アプリケーションを参照してください。

2

アプリケーション開発の開始

Oracle Database に接続する Java アプリケーションを開発するには、必要に応じていくつかのコンポーネントをインストールしておく必要があります。この章では、次の項目について説明します。

インストールする必要があるもの

サンプル・アプリケーションを開発するには、次の製品およびコンポーネントをインストールする必要があります。

- Oracle Database サーバー
- Oracle Database クライアント
- J2SE または JDK
- 統合開発環境
- Web サーバー

次の項目では、これらの要件について詳細に説明します。

Oracle Database サーバー

この Java アプリケーションを開発するには、Oracle Database サーバーおよび HR スキーマ（データベースに付属）による作業環境が必要です。クライアントをインストールする場合は、Oracle Database サーバーをインストールしてから Oracle Database クライアントをインストールします。インストールによって Oracle Database のインスタンスが作成されます。また、このデータベースを管理するための追加ツールを利用できるようになります。サーバーのインストールは、プラットフォーム固有です。詳細は、次の Oracle Database インストレーション・ガイドおよびリリース・ノートを参照してください。

- 『Oracle Database インストレーション・ガイド 11*g* リリース 1（11.1）for Linux』
- 『Oracle Database インストレーション・ガイド 11*g* リリース 1（11.1）for Microsoft Windows』

JDBC アプリケーションのための HR スキーマの変更

HR ユーザー・アカウント（このマニュアルの Java アプリケーションで使用するサンプル HR スキーマの所有者）は、最初はロックされています。HR としてログインするには、まず管理権限を持つユーザー（SYS）としてログインし、アカウントのロックを解除する必要があります。

データベースがローカルにインストールされている場合は、コマンド・プロンプトまたはコンソール・ウィンドウを使用して、次のようにアカウントのロックを解除します。

1. DBA 権限を持つユーザーとして SQL*Plus にログインします。次に例を示します。

```
> SQLPLUS SYS/ AS SYSDBA
Enter password: password
```

2. 次のコマンドを実行します。

```
> PASSWORD HR
Changing password for HR
New password: password
Retype new password: password
```

3. 次のように接続をテストします。

```
> CONNECT HR
Enter password: password
```

データベースに接続したことを示すメッセージが表示されます。

注意： Oracle Database でのセキュアなパスワードの作成および使用の詳細は、『Oracle Database セキュリティ・ガイド』を参照してください。

また、HR スキーマにある制約およびトリガーの一部は、このマニュアルで作成する Java アプリケーションの目的に合っていません。次の SQL 文を使用して、これらの制約およびトリガーを削除する必要があります。

```
DROP TRIGGER HR.UPDATE_JOB_HISTORY;
DROP TRIGGER HR.ADD_JOB_HISTORY;
DROP TRIGGER HR.SECURE_EMPLOYEES;
ALTER TABLE EMPLOYEES  DROP CONSTRAINT JHIST_EMP_FK;
DELETE FROM JOB_HISTORY;
```

Oracle Database クライアント

Oracle Database クライアントのインストールは任意ですが、インストールすることをお薦めします。Oracle Database クライアントをコンピュータにインストールすると、そのシステムから Oracle Database に簡単にアクセスできます。インストールには、次の開発ツールも含まれます。

- Oracle JDBC ドライバ
- Oracle Open Database Connectivity（ODBC）ドライバ
- Oracle Provider for OLE DB
- Oracle Data Provider for .NET（ODP.NET）
- Oracle Services for Microsoft Transaction Server

クライアントのインストールは、プラットフォーム固有です。クライアントのインストールの詳細は、次の Oracle Database クライアント・インストレーション・ガイドを参照してください。

- 『Oracle Database Client インストレーション・ガイド 11*g* リリース 1（11.1）for Linux』
- 『Oracle Database Client インストレーション・ガイド 11*g* リリース 1（11.1）for Microsoft Windows』

J2SE または JDK

Java アプリケーションを作成およびコンパイルするには、Java 2 Platform, Standard Edition, Software Development Kit（J2SE SDK）（以前の Java Development Kit（JDK））がすべて必要です。データベースにアクセスするアプリケーションを作成およびコンパイルするには、J2SE に付属の JDBC API がすべて必要です。このダウンロードには、Java Runtime Environment（JRE）も含まれています。

注意：

- Oracle Database では、JDK 1.2、JDK 1.3、JDK 1.4 およびすべての `classes12*.*` ファイルをサポートしていません。`ojdbc5.jar` および `ojbc6.jar` ファイルを、それぞれ JDK 5.*n* および JDK 6.*n* とともに使用する必要があります。
- `oracle.jdbc.driver.*` クラス、`ojdbc4.jar` ファイルおよび `OracleConnectionCacheImpl` クラスは、サポートされなくなり、使用できません。
- JDK のバージョンの表記規則が、JDK バージョン 1.*n* から JDK *n* に変更されました。詳細は、次の Sun 社の Java のサイトを参照してください。

  `http://java.sun.com/j2se/1.5.0/docs/relnotes/version-5.0.html`

関連項目：

- Java のインストールの詳細は、`http://java.sun.com/javase/index.jsp` を参照してください。
- JDBC API の詳細は、`http://java.sun.com/javase/technologies/database.jsp` を参照してください。

統合開発環境

アプリケーションの開発を簡単にするため、統合開発環境（IDE）でアプリケーションを開発することができます。このマニュアルでは、Oracle JDeveloper を使用して、このアプリケーションのファイルを作成します。JDeveloper のインストールの詳細は、「Oracle JDeveloper のインストール」を参照してください。

Web サーバー

このマニュアルで開発するサンプル・アプリケーションは、JavaServer Pages（JSP）テクノロジを使用して情報を表示し、ユーザーからの入力を受け入れます。これらのページをデプロイするには、サーブレットおよび JSP コンテナを使用する Web サーバー（Apache Tomcat アプリケーション・サーバーなど）が必要です。

このマニュアルでは、JSP ページのデプロイに、JDeveloper の埋込みサーバーを使用します。これは、Oracle Application Server Containers for J2EE サーバー、または略して OC4J サーバーと呼ばれます。Oracle JDeveloper をインストールしない場合でも、任意の Web サーバーを使用して JSP ページをデプロイできます。

JDeveloper では、次の本番アプリケーション・サーバーへの直接デプロイメントがサポートされています。

- Oracle Application Server
- BEA WebLogic
- Apache Tomcat
- IBM WebSphere
- JBoss

これらのサーバーの詳細は、ベンダー固有のドキュメントを参照してください。

Oracle Database クライアントのインストールの検証

Oracle Database クライアントのインストールは、プラットフォーム固有です。サンプル・アプリケーションの作成に進む前に、クライアントのインストールが成功したことを検証する必要があります。この項では、Oracle Database クライアントのインストールを検証する手順について説明します。

クライアントのインストールの検証には、次の作業があります。

- インストールされたディレクトリおよびファイルのチェック
- 環境変数のチェック
- JDBC ドライバ・バージョンの判別

インストールされたディレクトリおよびファイルのチェック

Oracle Java 製品をインストールすると、次のディレクトリが作成されます。

- *ORACLE_HOME*/jdbc
- *ORACLE_HOME*/jlib

表 2-1 に示すディレクトリが *ORACLE_HOME* ディレクトリ内に作成されているどうかを確認します。

表 2-1 ORACLE_HOME ディレクトリ内のディレクトリおよびファイル

ディレクトリ	説明
/jdbc/lib	lib ディレクトリには、必須 Java クラスの ojdbc5.jar および ojdbc6.jar が含まれています。これらには、JDK 5 および JDK 6 で使用する JDBC ドライバ・クラスが含まれています。
/jdbc/Readme.txt	このファイルには、ドライバに関する最新情報およびリリース固有の情報が含まれています。これらの情報は、製品の他のドキュメントには含まれていない場合があります。
/jlib	このディレクトリには、orai18n.jar ファイルが含まれています。このファイルには、グローバリゼーションおよびマルチバイト・キャラクタ・セットをサポートするためのクラスが含まれています。

注意： これらのファイルは、Sun 社の Web サイトからも入手できます。ただし、Oracle から提供されるファイルを使用することをお薦めします。これらのファイルは、Oracle のドライバでテストされています。

環境変数のチェック

この項では、JDBC Thin ドライバに対して設定する必要がある環境変数について説明します。インストールされている JDBC Thin ドライバをクラスパスに設定する必要があります。JDK 5 の場合、次の値を CLASSPATH 変数に設定する必要があります。

ORACLE_HOME/jdbc/lib/ojdbc5.jar
ORACLE_HOME/jlib/orai18n.jar

CLASSPATH 変数に設定した JDBC クラス・ファイル（ojdbc6.jar など）とグローバリゼーション・クラス・ファイル（orai18n.jar）は 1 つのみであることを確認します。

JDBC ドライバ・バージョンの判別

Oracle Database 11*g* リリース 1 を起動し、次のように、データベースでの JDBC サポートの詳細を確認します。

```
> java -jar ojdbc6.jar
  Oracle 11.1.0.0. JDBC 4.0 compiled with JDK6
```

また、OracleDatabaseMetaData クラスの getDriverVersion メソッドをコールすることによって、インストールした JDBC ドライバのバージョンを確認できます。

注意： JDBC Thin ドライバでは、データベースがインストールされているコンピュータで TCP/IP リスナーが実行されていることが必要です。

例 2-1 に、ドライバ・バージョンを確認する方法を示します。

例 2-1 JDBC ドライバ・バージョンの確認

```
import java.sql.*;
import oracle.jdbc.*;
import oracle.jdbc.pool.OracleDataSource;

class JDBCVersion
{
  public static void main (String args[]) throws SQLException
  {
    OracleDataSource ods = new OracleDataSource();
    ods.setURL("jdbc:oracle:thin:hr/hr@localhost:1521/XE");
    Connection conn = ods.getConnection();

    // Create Oracle DatabaseMetaData object
    DatabaseMetaData meta = conn.getMetaData();

    // gets driver info:
    System.out.println("JDBC driver version is " + meta.getDriverVersion());
  }
}
```

Oracle JDeveloper のインストール

このマニュアルで、JDBC を使用するサンプル Java アプリケーションの作成に使用する統合開発環境（IDE）は、Oracle JDeveloper リリース 10.1.3 です。このリリースの JDeveloper は、Microsoft Windows XP、Windows 2000、Windows NT、Linux、Solaris、Mac OS X および HP-UX オペレーティング・システムでサポートされています。JDeveloper のインストールについては、『Oracle JDeveloper インストレーション・ガイド Studio Edition』で詳細に説明されています。これは、Oracle Technology Network の次の場所で、オンラインで入手できます。

`http://www.oracle.com/technology/documentation/jdev/1013install/install.html`

このマニュアルは、JDeveloper のシステム要件を詳細に説明し、サポートされるプラットフォームでの JDeveloper のインストールについて詳細に説明しています。また、『Oracle JDeveloper リリース・ノート』も参照する必要があります。これは、次の Oracle Technology Network で、オンラインで入手できます。

`http://www.oracle.com/technology/products/jdev/htdocs/10.1.3.0.3/readme.html`

JDeveloper Studio Edition: 基本インストールと完全インストール

JDeveloper 10.1.3 には、3 つのエディションがあります。また、エディションごとに、基本インストールと完全インストールがあります。Studio Edition には Oracle ADF が含まれています。これは、このマニュアルで作成するマスター・ディテール・アプリケーションの開発に必要です。

JDeveloper Studio Edition の基本インストールまたは完全インストールをインストールします。完全インストールには、JDeveloper 以外に、必要なバージョンの Java、専用の Oracle Java Virtual Machine for JDeveloper（OJVM）およびオンライン・ドキュメントが含まれているため、ダウンロードされるファイル・サイズが大きくなります。迅速にダウンロードするには、JDeveloper の基本インストールをインストールします。

JDeveloper のインストール手順

JDeveloper は ZIP ファイルで提供されるため、インストーラは必要ありません。インストール・プロセスの概要は、次のとおりです。

1. 基本バージョンをインストールする場合、マシン上に J2SE バージョン 1.5.0_05 が必要です。完全バージョンをインストールする場合、この J2SE は含まれています。
2. Oracle Technology Network の次の場所から、JDeveloper バージョン 10.1.3 Studio Edition をダウンロードします。

   `http://www.oracle.com/technology/software/products/jdev/htdocs/soft1013.html`

   基本インストール（`jdevstudiobase1013.zip`）または完全インストール（`jdevstudio1013.zip`）をダウンロードします。
3. ダウンロードしたファイルをターゲット・インストール・ディレクトリに解凍します。

   注意： JDeveloper を既存の `ORACLE_HOME` にインストールしないでください。Oracle Universal Uninstaller を使用してアンインストールできなくなります。

   UNIX または Linux システムに `jdevstudio1013.zip` をインストールする場合は、SDK を指定するように `jdev.conf` を変更する必要があります。ファイル *<jdev_install>*`/jdev/bin/jdev.conf` の変数 `SetJavaHome` を、Java のインスール先に設定します。

   たとえば、UNIX 環境で、Sun J2SE SDK の場所が `/usr/local/java` というディレクトリである場合、`jdev.conf` のエントリは次のようになります。

   ```
   SetJavaHome /usr/local/java
   ```

   この他に実行する必要がある作業は、すべての JDeveloper ファイルの読取り権限を設定することと、すべてのユーザーに、JDeveloper ディレクトリにあるファイルに対する書込み権限と実行権限を付与することです。
4. 基本インストールを使用する場合は、さらにいくつかの設定作業があります。JDeveloper 構成ファイルに Java のインストール先を設定する、オプションで OJVM をインストールする、オンライン・ドキュメントをローカルで使用できるようにダウンロードする、などです。

   関連項目： JDeveloper のインストレーション・ガイドは、`http://www.oracle.com/technology/documentation/jdev/1013install/install.html` を参照してください。

JDeveloper の起動

Windows で JDeveloper を起動するには、*`jdev_install`*`¥jdev¥bin¥jdevw.exe` ファイルを実行します。*`jdev_install`* は、JDeveloper ファイルを解凍した場所のパスです。内部診断情報を表示するためにコンソール・ウィンドウを使用するには、`jdevw.exe` ではなく、同じディレクトリ内の `jdev.exe` ファイルを実行します。

その他のプラットフォームで JDeveloper を起動するには、*`jdev_install`*`/jdev/bin/jdev` ファイルを実行します。

3

Oracle Database への接続

この章は、この後に続く 5 つの章の最初の章です。これらの章では、Oracle Database にアクセスし、そのデータを表示、変更、削除および更新する Java アプリケーションの各部分を作成する方法について説明します。Java アプリケーションからデータベースにアクセスするには、`java.sql.Connection` オブジェクトを使用してデータベースに接続する必要があります。

この章は次の項で構成されています。

JDeveloper からの Oracle Database への接続

JDeveloper でデータベース接続を設定および管理して、アプリケーションを Oracle Database やオフライン・データベース・オブジェクトなどの外部データソースと通信可能にすることができます。これは、接続ナビゲータを使用して行います。アプリケーション・サーバーへの接続など、アプリケーションが必要とする他の接続の管理にも同じナビゲータを使用します。次の項目では、接続ナビゲータを使用してデータベースとそのオブジェクトを表示し、データベースへの接続を作成する方法について説明します。

JDeveloper 接続ナビゲータ

接続ナビゲータには、現在定義されている接続がすべて表示されます。接続ナビゲータを表示するには、JDeveloper の画面左上にあるナビゲータ・パネルで**「接続」**タブ（表示されている場合）を選択するか、「表示」メニューを使用します。JDeveloper IDE のデフォルトのレイアウトは、図 1-1 を参照してください。

接続ナビゲータを使用して、表示されている接続を参照できます。特に、データベース・スキーマについては、データベース・オブジェクト、表、ビューおよびそれらの内容を表示することもできます。

データベース接続は、「データベース」ノードの下に表示されます。データベース内のオブジェクトを表示するには、接続を開きます。スキーマを開くと、そのスキーマ内のオブジェクト・タイプのノードが表示されます。オブジェクト・タイプのノードを開くと、個々のオブジェクトが表示されます。表のノードを開くと、表の構造および表内のデータを表示できます。

データベース接続の作成

接続の詳細がわかっている任意のデータベースに接続できます。データベース接続を作成するときは、ユーザー名とパスワードを指定する必要があります。デフォルトでは、接続で指定したユーザーのスキーマのみを表示できます。

接続を作成するには、次の手順を実行します。

1. JDeveloper を起動します。
2. **「表示」**メニューで、**「接続ナビゲータ」**を選択します。接続ナビゲータが表示され、選択可能な接続のリストが表示されます。
3. **「データベース」**を右クリックし、ショートカット・メニューで**「データベース接続の作成」**を選択します。「データベース接続の作成」ウィザードが表示されます。「ようこそ」画面で**「次へ」**をクリックします。ウィザードの「タイプ」画面が表示されます。
4. 「タイプ」画面では、接続名およびタイプのデフォルト値（`DBConnection1` および `Oracle (JDBC)`）を変更しないでください。**「次へ」**をクリックします。ウィザードの「認証」画面が表示されます。
5. 「認証」画面で、**「ユーザー名」**フィールドと**「パスワード」**フィールドの両方に `HR` と入力します。「ロール」の値は入力せず、**「パスワードを配布」**を選択します。**「次へ」**をクリックします。ウィザードの「接続」画面が表示されます。

6. 「接続」画面では、データベースが配置されているコンピュータに関する情報を入力する必要があります。この情報は、データベース管理者から入手します。

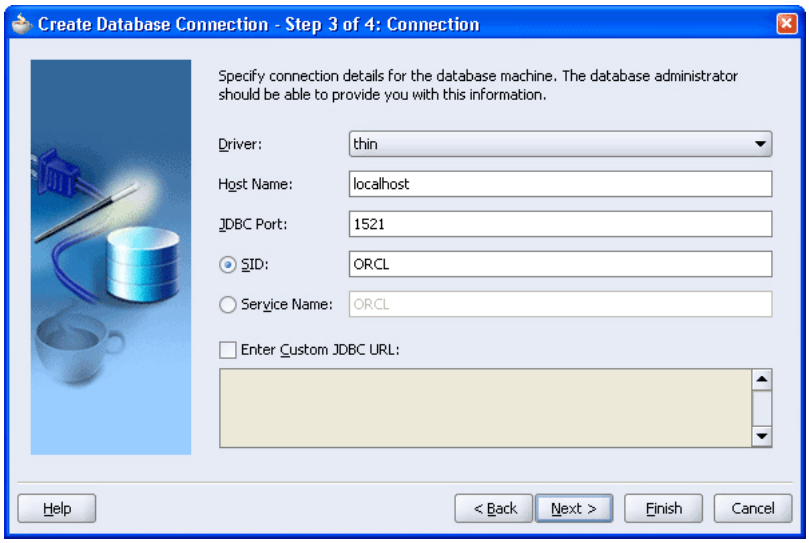
   次の情報を入力します。

   - **ドライバ**: `thin`
   - **ホスト名**: *`Oracle Database`* がインストールされているコンピュータのホスト名

     データベースが同じコンピュータ上にある場合は、**「ホスト名」**パラメータには `localhost` と入力します。
   - **JDBC ポート**: `1521`
   - **SID**: `ORCL`

     図 3-1 に、これらの詳細を入力する「接続」画面を示します。

図 3-1 接続の詳細の指定



7. 「テスト」画面で、データベースに正常に接続できるかどうかをテストできます。**「接続のテスト」**をクリックします。接続が成功すると、「ステータス」フィールドに「成功」と表示されます。
8. **「終了」**をクリックして接続を作成し、ウィザードを閉じます。

JDeveloper での Oracle Database からの切断と再接続

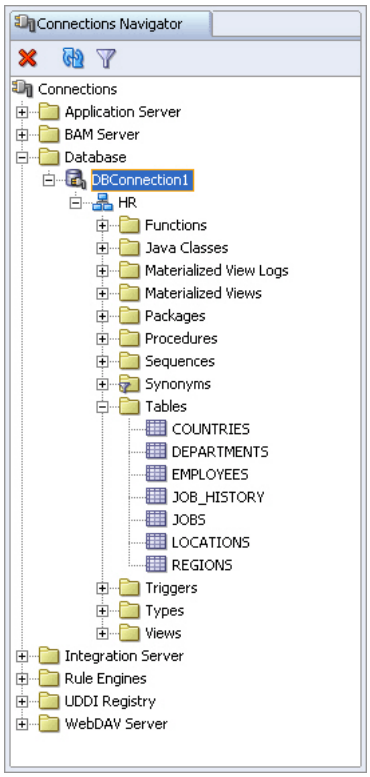
JDeveloper でデータベースから切断するには、接続ナビゲータで、接続名を右クリックして**「切断」**を選択します。接続ナビゲータの表示には接続名のみが表示され、ノードを開くためのプラス（+）記号は表示されなくなります。データベースに再接続するには、接続名を右クリックして**「接続」**を選択します。

接続ナビゲータを使用したデータの表示

データベースへの接続を正常に確立した後で、その内容を接続ナビゲータで表示できます。接続ナビゲータには、データベース、そのオブジェクト、それらのインスタンスおよびそれぞれの内容について、ナビゲーション可能な階層ツリー構造が表示されます。作成したデータベース接続の階層の各レベルで内容を表示するには、次の手順を実行します。

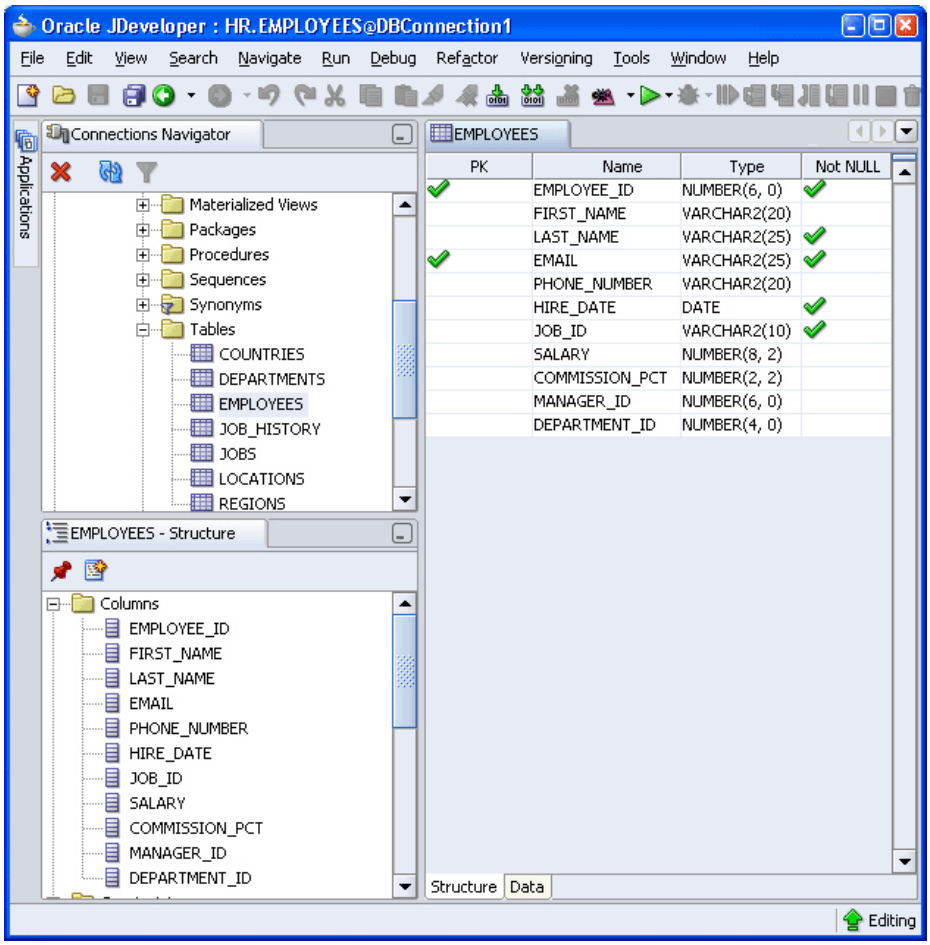
1. 接続ナビゲータの「データベース」ノードには、接続名が付いたノードが表示されています。接続名の左にあるプラス記号（+）をクリックして、ナビゲーション・ツリーを開きます。接続したスキーマの名前（この場合は HR）が表示されます。
2. HR スキーマ内のすべてのオブジェクトのリストを表示するには、**「HR」**ナビゲーション・ツリーを開きます。あるオブジェクト・タイプ（たとえば表）についてインスタンスのリストを表示するには、**「表」**ナビゲーション・ツリーを開きます。

図 3-2 接続ナビゲータでのデータベース・オブジェクトへのアクセス



3. ナビゲータの下の構造ウィンドウには、ナビゲータで選択したオブジェクトの詳細な構造が表示されます。ナビゲータで表（**Employees** など）を選択して、構造ウィンドウにその表の列を表示します。

図 3-3 表の構造およびデータの表示



4. ナビゲータで表をダブルクリックすると、その表の構造がウィンドウのメインの編集領域に表示されます。すべての列（名前、タイプ、サイズなど）の詳細が含まれているため、表の定義を調べることができます。

   表のデータを表示するには、表の構造の下にある**「データ」**タブを選択します。表のデータを表示して参照できます。

5. 接続ナビゲータでオブジェクトを編集することもできます。表を編集するには、表を右クリックし、ショートカット・メニューで**「編集」**を選択します。選択した表をダイアログ・ボックスで変更できます。

JDeveloper でのアプリケーションおよびプロジェクトの設定

JDeveloper では、アプリケーション内で作業を行い、アプリケーション内で、多数のプロジェクトに編成できます。JDeveloper には多数のアプリケーション・テンプレートがあり、標準的なタイプのアプリケーションに合うプロジェクト構造を、比較的迅速かつ簡単に作成するために役立ちます。JDeveloper でアプリケーションを作成する際は、構築するアプリケーションのタイプに合ったアプリケーション・テンプレートを選択します。

選択したアプリケーション・テンプレートによって、初期のプロジェクト構造（アプリケーション内で指定したプロジェクト・フォルダ）および含まれるアプリケーション・テクノロジが決まります。その後に、個々のアプリケーションに必要な特別なライブラリやテクノロジを追加できます。必要に応じて、追加のプロジェクトを作成することもできます。

JDeveloper アプリケーション・ナビゲータの使用

アプリケーション・ナビゲータには、アプリケーションとプロジェクトがすべて表示されます。JDeveloper を初めて起動すると、JDeveloper IDE の左側にデフォルトでアプリケーション・ナビゲータが表示されます。

アプリケーション・ナビゲータを表示するには（表示されていない場合）、JDeveloper 画面の左上にあるナビゲータ・パネルで**「アプリケーション」**タブをクリックするか、「表示」メニューで**「アプリケーション・ナビゲータ」**を選択します。

アプリケーション・ナビゲータには、プロジェクト内の項目の論理的なグループが表示されます。個々の項目の構造を表示するには、項目を選択して、その構造を構造ウィンドウに表示できます。

アプリケーション・ナビゲータから、適切なデフォルトのエディタに項目を表示できます。たとえば、Java ファイルをダブルクリックすると、ファイルは Java ソース・エディタに開きます。JavaServer Pages（JSP）ファイルをダブルクリックすると、JSP/HTML ビジュアル・エディタに開きます。

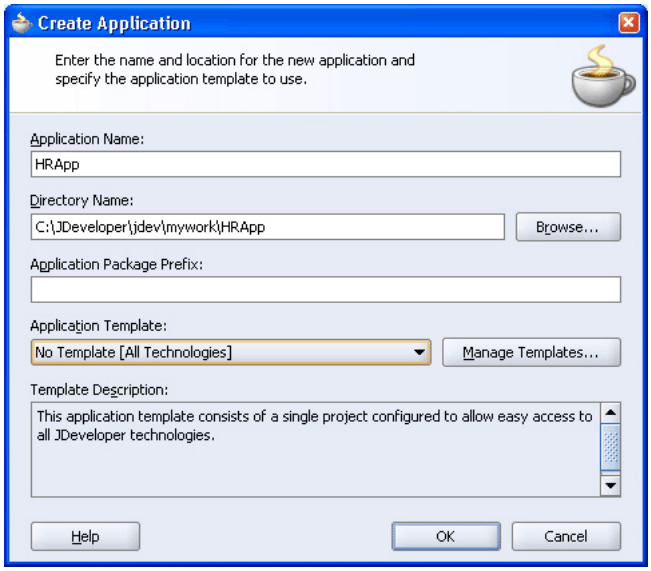
アプリケーションおよびプロジェクトの作成

JDeveloper の使用を開始するには、アプリケーションを作成する必要があります。また、作業を格納するためのプロジェクトを少なくとも 1 つ作成する必要があります。手順は次のとおりです。

1. アプリケーション・ナビゲータで、**「アプリケーション」**を右クリックし、ショートカット・メニューで**「新規アプリケーション」**を選択します。図 3-4 に示す「アプリケーション・ワークスペースの作成」ダイアログ・ボックスが表示されます。

2. 「アプリケーション名」フィールドに HRApp と入力し、「アプリケーション・テンプレート」リストで**「テンプレートなし [すべてのテクノロジ]」**を選択します。**「OK」**をクリックします。「プロジェクトの作成」ダイアログ・ボックスが表示されます。

図 3-4 アプリケーションの作成



3. 「プロジェクトの作成」ダイアログ・ボックスで、プロジェクトの名前として view と入力します。
4. 新しい HRApp アプリケーションがアプリケーション・ナビゲータに表示されます。
5. アプリケーションを保存します。このことを行うには、**「ファイル」**メニューで、**「すべて保存」**を選択します。

プロジェクトの範囲で利用可能な Javadoc およびソース・コードの表示

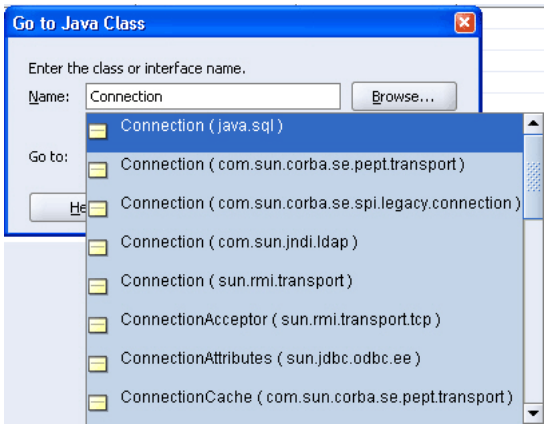
プロジェクトのテクノロジ範囲内で利用可能なクラスの Javadoc またはコードを JDeveloper で表示できます。また、それらのクラスに対して利用可能なすべてのメソッドの詳細を表示できます。

たとえば、Connection クラスのコードまたは Javadoc を表示するには、次の手順を実行します。

1. アプリケーション・ナビゲータでプロジェクトが選択されている状態で、**「ナビゲート」**メニューで**「Java クラスに移動」**を選択します。この操作は、プロジェクト内の特定のファイルに対して行うこともできます。
2. 「Java クラスに移動」ダイアログ・ボックスで、**「ソース」**または**「Javadoc」**を選択します。

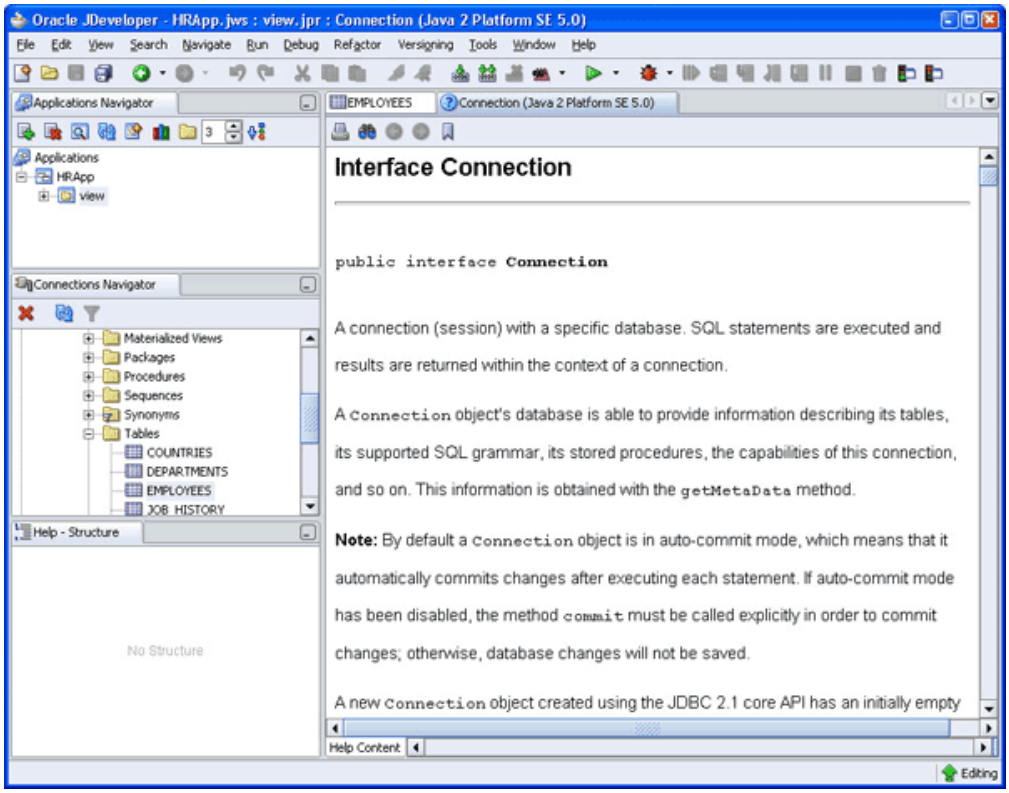
3. 表示するクラスの名前を「**名前**」フィールドに入力するか、「**参照**」をクリックしてクラスを検索します。`Connection` クラスの場合、まず `Connection` と入力し、表示されるリストで「**Connection (java.sql)**」を選択します。

図 3-5 JDeveloper で Javadoc を表示するクラスの選択



4. 「**OK**」をクリックします。

図 3-6 JDeveloper での Javadoc の表示



Java アプリケーションからの Oracle Database への接続

ここまでは、JDeveloper からデータベースに接続する方法を見てきました。Java アプリケーションから接続を起動するには、JDBC Application Program Interface（API）の Connection オブジェクトを使用します。

この項では、Java アプリケーションからのデータベースへの接続について、次の項目で説明します。

- Oracle Database への接続の概要
- データベース URL の指定
- JDeveloper での Java クラスの作成
- Java ライブラリ
- JDBC および JSP ライブラリの追加
- JDBC パッケージのインポート
- 接続関連の変数の宣言
- 接続メソッドの作成

Oracle Database への接続の概要

Java では、DataSource オブジェクトのインスタンスを使用してデータベースへの接続を取得します。DataSource インタフェースは、以前の JDBC DriverManager クラスに完全に置き換わるものです。oracle.jdbc.pool パッケージの OracleDataSource クラスにより、javax.sql.DataSource インタフェースが実装されました。オーバーロードされた getConnection メソッドによって、データベースへの物理接続が返されます。

注意： データベースへの接続を確立するための DriverManager クラスの使用は、廃止される予定です。

その DataSource オブジェクトに適した setxxx メソッドを使用してプロパティを設定することも、これらのプロパティを入力パラメータとして受け入れる getConnection メソッドを使用することもできます。

表 3-1 に、DataSource の主要なプロパティを示します。

表 3-1 標準的なデータソース・プロパティ

名前	型	説明
databaseName	String	サーバー上の特定のデータベースの名前。Oracle の用語では、サービス名（または SID）とも呼ばれます。Oracle Database のデフォルトは、ORCL です。
dataSourceName	String	基礎となるデータソース・クラスの名前。
description	String	データソースの説明。
networkProtocol	String	サーバーとの通信のためのネットワーク・プロトコル。Oracle では、JDBC Oracle Call Interface（OCI）ドライバにのみ該当し、デフォルトは tcp です。
password	String	接続するユーザーのパスワード。
portNumber	int	サーバーがリクエストをリスニングするポートの番号。
serverName	String	データベース・サーバーの名前。
user	String	ログインに使用されるユーザー名

表 3-1 標準的なデータソース・プロパティ（続き）

名前	型	説明
driverType	String	Oracle JDBC ドライバのタイプ。oci または thin です。 これは、Oracle 固有のプロパティです。
url	String	データベース接続文字列の URL を指定します。標準の portNumber、networkProtocol、serverName および databaseName プロパティのかわりに、このプロパティを使用できます。 これは、Oracle 固有のプロパティです。

DataSource オブジェクトの url プロパティを必要なすべてのパラメータとともに設定すると、他のプロパティを設定したり、getDBConnection メソッドに追加パラメータを指定しなくても、データベースに接続できます。データベース URL の設定の詳細は、「データベース URL の指定」の項を参照してください。

注意： getConnection メソッドで指定したパラメータは、それまでにアプリケーションで指定されたすべてのプロパティおよび url パラメータの設定よりも優先されます。

関連項目：『Oracle Database JDBC 開発者ガイドおよびリファレンス』

データベース URL の指定

データベース URL は、DataSource オブジェクトの url プロパティの値に指定する文字列です。完全な URL の構文は次のとおりです。

jdbc:oracle:*driver_type*:[*username*/*password*]@*database_specifier*

URL の最初の部分には、使用する JDBC ドライバを指定します。クライアント側アプリケーションでサポートされている *driver_type* の値は、thin および oci です。大カッコは、ユーザー名とパスワードの組合せがオプションであることを示します。*database_specifier* の値は、アプリケーションが接続されるデータベースを示します。

次に示すのは、Thin ドライバによってサポートされる Thin スタイルのサービス名の構文です。

jdbc:oracle:*driver_type*:[*username*/*password*]@//*host_name*:*port_number*:*SID*

例 3-1 に、このマニュアルで作成するサンプル・アプリケーションで、ユーザー名とパスワードを指定し、さらにデータベースがローカルに格納されている場合のデータベース接続 URL を示します。

例 3-1 DataSource オブジェクトの url プロパティの指定

```
jdbc:oracle:thin:hr/hr@localhost:1521:UORCL
```

Oracle Database クライアントのデフォルト・サービス機能の使用

Oracle Database に、新しい接続機能があります。Oracle Database クライアントをインストールすると、接続 URL のデータベース識別子部分に詳細を指定する必要はありません。一定の条件では、Oracle Database 接続アダプタに必要な指定は、データベースがインストールされているコンピュータのホスト名のみです。

Oracle Database に導入されたこの機能により、JDBC 接続 URL の構文の一部はオプションになります。

jdbc:oracle:*driver_type*:[*username*/*password*]@[//]*host_name*[:*port*][:ORCL]

この URL では、次のようになります。

- // はオプションです。
- *:port* はオプションです。

  デフォルトの Oracle Net リスナー・ポート（1521）が使用されない場合にのみ、ポートを指定します。

- :ORCL（サービス名）はオプションです。

  Oracle Database クライアントの接続アダプタは、ホスト上のデフォルトのサービスに接続します。これは、ホストの listener.ora ファイルで、ORCL に設定されています。

例 3-2 に、listener.ora ファイルの基本構成を示します。ここでは、デフォルトのサービスが定義されています。

例 3-2 listener.ora のデフォルトのサービス構成

```
MYLISTENER = (ADDRESS_LIST=
 (ADDRESS=(PROTOCOL=tcp)(HOST=test555)(PORT=1521))
)
DEFAULT_SERVICE_MYLISTENER=dbjf.regress.rdbms.dev.testserver.com

SID_LIST_MYLISTENER = (SID_LIST=
(SID_DESC=(SID_NAME=dbjf)(GLOBAL_DBNAME=dbjf.regress.rdbms.dev.testserver.com)(ORACLE_
HOME=/test/oracle))
)
```

listener.ora ファイルを変更した後で、次のコマンドを使用してリスナーを再起動する必要があります。

```
> lsnrctl start mylistener
```

この構成では、次の URL になります。

```
jdbc:oracle:thin:@//test555.testserver.com
jdbc:oracle:thin:@//test555.testserver.com:1521
jdbc:oracle:thin:@test555.testserver.com
jdbc:oracle:thin:@test555.testserver.com:1521
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=test555.testserver.com)(POR
T=1521)))
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=test555.testserver.com)))
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=test555.testserver.com)(POR
T=1521))(CONNECT_DATA=(SERVICE_NAME=)))
```

注意： デフォルト・サービスは、Oracle Database 11*g* リリース 1 の新機能です。他のバージョンの Oracle Database クライアントを使用してデータベースに接続する場合は、SID およびポート番号を指定する必要があります。

JDeveloper での Java クラスの作成

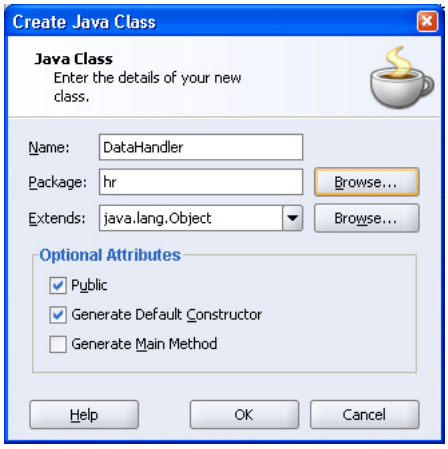
Java アプリケーションの作成では、最初に Java クラスを作成します。次の手順では、DataHandler という名前のクラスを作成する方法について説明します。このクラスには、データベースを問い合せてそのデータを変更するメソッドが含まれます。

1. アプリケーション・ナビゲータで、**View** プロジェクトを右クリックし、ショートカット・メニューで**「新規」**を選択します。

2. **「新規ギャラリ」**で、**「一般」**カテゴリを選択します。**「項目」**リストで、**「Java クラス」**を選択し、**「OK」**をクリックします。「Java クラスの作成」ダイアログ・ボックスが表示されます。

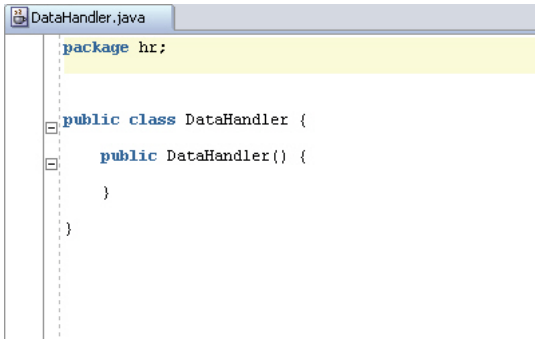
3. 「Java クラスの作成」ダイアログ・ボックスで、クラスの**「名前」**として `DataHandler` と入力し、**「パッケージ」**として `hr` と入力します。「オプション属性」のデフォルト値は変更せず、**「OK」**をクリックします。図 3-7 に、適切な値が指定された「Java クラスの作成」ダイアログ・ボックスを示します。

図 3-7 Java クラスの作成



4. `DataHandler` スケルトン・クラスが作成され、Java ソース・エディタに表示されます。パッケージ宣言、クラス宣言およびデフォルトのコンストラクタがデフォルトで作成されます。図 3-8 に、Java ソース・エディタに表示されたクラスを示します。独自の Java コードを追加できます。

図 3-8 Java ソース・エディタ



Java ライブラリ

Oracle JDeveloper には、Java アプリケーション・プログラミングに役立つ標準ライブラリがあります。これらのライブラリには、Application Development Framework（ADF）、JDBC 用の Oracle ライブラリ、JSP などの API サポートが含まれています。

プロジェクトで JDBC を使用するには、Oracle JDBC ライブラリをプロジェクトにインポートします。同様に、JSP テクノロジを使用するには、JSP ランタイム・ライブラリをインポートします。

Oracle JDBC ライブラリの概要

Oracle JDBC ライブラリの主要なパッケージは、次のとおりです。

- `oracle.jdbc`: `oracle.jdbc` パッケージのインタフェースは、`java.sql` パッケージのインタフェースに対する Oracle の拡張機能を定義します。これらの拡張機能によって、Oracle SQL 形式のデータおよびその他の Oracle 固有の機能（Oracle のパフォーマンス拡張など）にアクセスできます。
- `oracle.sql`: `oracle.sql` パッケージは、SQL 形式のデータへの直接アクセスをサポートします。このパッケージは、主に、SQL データへの Java マッピングを提供するクラスとそれらのサポート・クラスによって構成されます。
- `oracle.jdbc.pool`: このパッケージには、データベースへの接続を取得するために使用される `OracleDataSource` クラスが含まれています。オーバーロードされた `getConnection` メソッドによって、データベースへの物理接続が返されます。

JSP ランタイム・ライブラリの概要

このライブラリには、JDeveloper に付属の OC4J サーバーで JSP ファイルを解析して実行するために必要なクラスおよびタグ・ライブラリが含まれています。

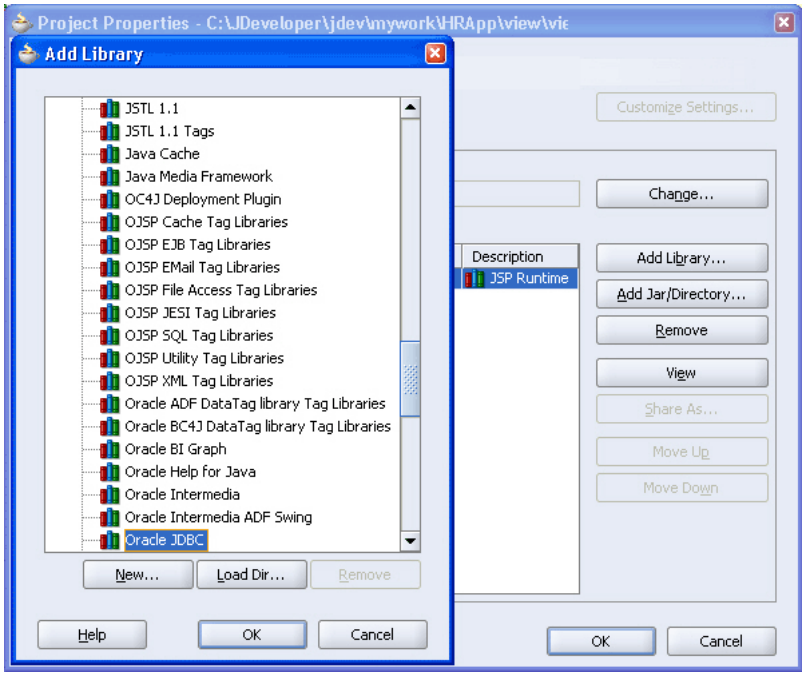
JDBC および JSP ライブラリの追加

プロジェクトにライブラリを含めるには、次の手順を実行します。

1. アプリケーション・ナビゲータで **View** プロジェクトをダブルクリックして、「プロジェクト・プロパティ」ダイアログ・ボックスを表示します。
2. **「ライブラリ」**をクリックし、**「ライブラリの追加」**をクリックします。「ライブラリの追加」ダイアログ・ボックスが表示され、Java 2 Platform, Standard Edition（J2SE）バージョンの選択可能なライブラリのリストが表示されます。

3. 「ライブラリの追加」ダイアログ・ボックスで、Extension フォルダ内のライブラリのリストをスクロールします。**「JSP Runtime」**ライブラリを選択し、「OK」をクリックして、プロジェクトの選択されたライブラリのリストに追加します。同様に、Oracle JDBC ライブラリを追加します。図 3-9 に、`view` プロジェクトに追加された Oracle JDBC ライブラリを示します。

図 3-9 ライブラリのインポート



4. 「**OK**」をクリックします。

JDBC パッケージのインポート

Java アプリケーションで JDBC を使用するには、次の JDBC パッケージをインポートします。

1. `DataHandler.java` クラスが Java ソース・エディタで開かれていない場合は、アプリケーション・ナビゲータで、**View** プロジェクト、**「アプリケーション・ソース」**および自分のパッケージ（**「hr」**）を開き、**「DataHandler.java」**をダブルクリックします。
2. 生成されたパッケージ宣言の最後で、新しい行に、例 3-3 に示すとおり `import` 文を入力します。

例 3-3 Java アプリケーションでのパッケージのインポート

```
package hr;
import java.sql.Connection;
import oracle.jdbc.pool.OracleDataSource;
```

接続関連の変数の宣言

接続情報は、接続変数（接続 URL、ユーザー名および対応するパスワード）を使用して接続メソッドに渡されます。

JDeveloper の Java ソース・エディタを使用して、DataHandler.java クラスを次のように編集します。

1. DataHandler コンストラクタの後に行を追加し、3 つの接続変数を次のように宣言します。

```
String jdbcUrl = null;
String userid = null;
String password = null;
```

これらの変数は、ユーザーがログイン時に入力した値をアプリケーションで保持し、ユーザーを認証してデータベースへの接続を作成するために使用されます。jdbcUrl 変数は、接続するデータベースの URL を保持するために使用されます。userid および password 変数は、ユーザーを認証し、セッションで使用されるスキーマを識別するために使用されます。

注意： ログイン変数は、アプリケーションを保護するために null に設定されています。この時点では、アプリケーション・ログイン機能はまだアプリケーションに組み込まれていません。そのため、ログイン機能が組み込まれるまでの間にアプリケーションをテストするため、ログイン変数の値を次のように設定できます。

データベースの接続文字列に、jdbcUrl 変数を設定します。

```
String jdbcUrl = "jdbc:oracle:thin:@localhost:1521:ORCL";
```

変数 userid および password を次のように hr に設定します。

```
String userid = "hr";
String password = "hr";
```

テストが終了したらすぐに、これらを null にリセットします。

セキュリティ機能およびプラクティスの詳細は、『Oracle Database セキュリティ・ガイド』および使用する開発環境についてベンダが提供するドキュメントを参照してください。

---

2. 新しい行で、次のように接続インスタンスを宣言します。

```
Connection conn;
```

これで、Java クラスには例 3-4 のコードが含まれます。

例 3-4 接続変数および接続オブジェクトの宣言

```
package hr;
import java.sql.Connection;
import oracle.jdbc.pool.OracleDataSource;

public class DataHandler {
    public DataHandler() {
    }
    String jdbcUrl = null;
    String userid = null;
    String password = null;
    Connection conn;
}
```

接続メソッドの作成

データベースに接続するには、次のようにメソッドを作成します。

1. 接続宣言の後に次のメソッド宣言を追加します。

   ```
   public void getDBConnection() throws SQLException
   ```

   Java コード・インサイト機能によって、SQLException エラー処理パッケージのインポートを促すメッセージが表示されます。[Alt] キーを押しながら [Enter] キーを押してインポートします。import java.sql.SQLException 文がインポート・パッケージのリストに追加されます。

2. 同じ行の最後で、開きカッコ（{）を追加し、[Enter] キーを押します。JDeveloper によって自動的に閉じカッコが作成され、カッコの間の新しい空の行にカーソルが置かれます。

3. 新しい行で、次のように OracleDataSource インスタンスを宣言します。

   ```
   OracleDataSource ds;
   ```

4. 次のように入力して、新しい OracleDataSource オブジェクトを追加します。

   ```
   ds = new OracleDataSource();
   ```

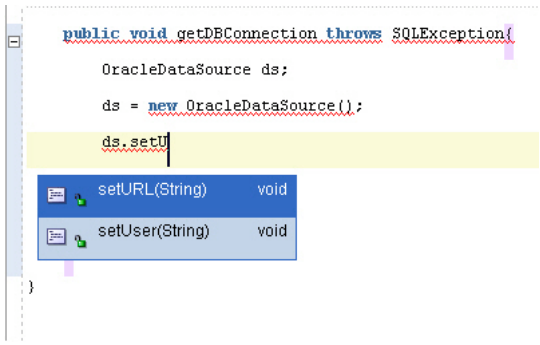
5. 次のように入力を開始して、DataSource オブジェクトの URL を設定します。

   ```
   ds.setURL(jdbcUrl);
   ```

   Java コード・インサイトによって、選択可能な OracleDataSource メソッドのリストが表示されます。リストをスクロールして setURL(String) メソッドを選択し、[Enter] キーを押してコードに挿入します。このファンクションのカッコ内に、jdbcUrl と入力します。

   図 3-10 に、JDeveloper の Java コード・インサイト機能がコードの挿入にどのように役立つかを示します。

図 3-10 Java コード・インサイト



6. 次の行で、次のように入力します。

   ```
   conn = ds.getConnection(userid,password);
   ```

   通常、Java コード・インサイトによって ds のメソッドのリストが表示されます。今回は、getConnection(String,String) を選択します。カッコ内に、userid,password と入力します。行の最後にセミコロン（;）を付けます。

   コードは、例 3-5 のコードのようになります。

例 3-5 データベースに接続するメソッドの追加

```
package hr;
import java.sql.Connection;
import java.sql.SQLException;

import oracle.jdbc.pool.OracleDataSource;

public class DataHandler {
    public DataHandler() {
    }
    String jdbcUrl = null;
    String userid = null;
    String password = null;
    Connection conn;
    public void getDBConnection() throws SQLException{
        OracleDataSource ds;
        ds = new OracleDataSource();
        ds.setURL(jdbcUrl);
        conn=ds.getConnection(userid,password);

    }
}
```

7. クラスをコンパイルして、構文エラーがないことを確認します。このことを行うには、Java ソース・エディタで右クリックし、ショートカット・メニューで**「メイク」**を選択します。Java ソース・エディタ・ウィンドウの下のログ・ウィンドウに、コンパイルが成功したことを示すメッセージが表示されます。

4

データの問合せおよび表示

この章では、データベースを問い合せる機能およびコードを DataHandler.java ファイルに追加します。この章は次の項で構成されています。

Oracle Database 内のデータの問合せの概要

Java クラスから Oracle Database を問い合せてデータを取得する手順の概要は、次のとおりです。

1. `OracleDataSource.getConnection` メソッドを使用して、接続を作成します。これについては、第 3 章「Oracle Database への接続」で説明されています。
2. 接続オブジェクトに対して使用可能なメソッドを使用して、SQL 文を定義します。SQL 問合せ文の定義には、`createStatement` メソッドを使用します。
3. 文に対して使用可能なメソッドを使用して、問合せを実行します。`executeQuery` メソッドを使用してデータベースの問合せを実行し、問合せ条件に一致する行のセットを生成します。これらの結果は、`ResultSet` オブジェクトに含まれます。
4. `ResultSet` オブジェクトを使用して、データをアプリケーション・ページに表示します。

次の項では、Java アプリケーションからのデータベースの問合せに関する主要な Java Database Connectivity（JDBC）の概念について説明します。

- SQL 文
- Statement オブジェクトの問合せメソッド
- 結果セット

関連項目：『Oracle Database JDBC 開発者ガイドおよびリファレンス』

SQL 文

データベースに接続し、そのプロセスで `Connection` オブジェクトを作成した後、次に、`Statement` オブジェクトを作成します。JDBC `Connection` オブジェクトの `createStatement` メソッドによって、JDBC `Statement` タイプのオブジェクトが戻されます。例 4-1 に、`Statement` オブジェクトの作成方法を示します。

例 4-1 Statement オブジェクトの作成

```
Statement stmt = conn.createStatement();
```

`Statement` オブジェクトは、アプリケーション内にコーディングできる静的な SQL 問合せを実行するために使用します。

また、更新値を変更しながら、数多くの類似の問合せをデータベースに対して実行する必要がある場合は、`Statement` オブジェクトを拡張する `OraclePreparedStatement` オブジェクトを使用します。Oracle Database のストアド・プロシージャにアクセスするには、`OracleCallableStatement` オブジェクトを使用します。

関連項目：

- 「OraclePreparedStatement の使用」
- 「OracleCallableStatement の使用」
- 『Oracle Database JDBC 開発者ガイドおよびリファレンス』

Statement オブジェクトの問合せメソッド

Statement オブジェクトに埋め込まれた問合せを実行するには、execute メソッドのバリアントを使用します。このメソッドの主要なバリアントを表 4-1 に示します。

表 4-1 java.sql.Statement の主要な問合せ実行メソッド

メソッド名	戻り型	説明
execute(String sql)	Boolean	指定された SQL 文を実行します。ブール・レスポンス（問合せが正常に実行された場合は true、それ以外は false）が戻されます。
addBatch()	void	コマンドの PreparedStatement オブジェクト・バッチにパラメータのセットを追加します。
executeBatch()	int[]	コマンドのバッチをデータベースに送信して実行し、すべてのコマンドが正常に実行された場合は、更新件数の配列を戻します。
executeQuery(String sql)	ResultSet	指定された SQL 文を実行します。単一の ResultSet オブジェクトが戻されます。
executeUpdate(String sql)	int	指定された SQL 文を実行します。INSERT、UPDATE または DELETE 文あるいは何も戻さない SQL 文（SQL DDL 文など）の場合があります。

関連項目：

- http://java.sun.com/j2se/1.5.0/docs/api/index.html

結果セット

ResultSet オブジェクトには、データベースの結果セットを表すデータの表が含まれます。この表は、データベースを問い合せる文を実行することによって生成されます。

カーソルが、ResultSet オブジェクト内のデータの現在の行を示します。最初、カーソルは最初の行の前に置かれています。ResultSet オブジェクトの next メソッドを使用して、カーソルを結果セット内の次の行に移動します。ResultSet オブジェクト内に行がなくなると、false が戻されます。通常、ResultSet オブジェクトの内容は、ループ内で false が戻されるまで next メソッドを使用して読み取られます。

ResultSet インタフェースには、現在の行から列の値を取得するためのアクセッサ・メソッド（getBoolean、getLong、getInt など）があります。値は、列の索引番号または列の名前を使用して取得できます。

デフォルトでは、同時に開くことができるのは、Statement オブジェクトごとに 1 つの ResultSet オブジェクトのみです。そのため、複数の ResultSet オブジェクトからデータを読み取るには、複数の Statement オブジェクトを使用する必要があります。ResultSet オブジェクトを生成した Statement オブジェクトが閉じられた場合、再実行された場合または複数の一連の結果から次の結果を取得するために使用された場合、その ResultSet オブジェクトは自動的に閉じられます。

関連項目：

- SQL タイプと Java タイプのマッピングの詳細は、http://java.sun.com/j2se/1.3/docs/guide/jdbc/getstart/mapping.html を参照してください。
- 結果セットとその機能の詳細は、『Oracle Database JDBC 開発者ガイドおよびリファレンス』を参照してください。

ResultSet オブジェクトの機能

スクロール可能性とは、結果セット内を前後に移動できることを言います。また、**相対位置指定**や**絶対位置指定**によって、結果セット内の任意の位置に移動することもできます。相対位置指定では、現在の行から、指定した行数を前後に移動できます。絶対位置指定では、結果セットの最初または最後から数えて、指定した行番号に移動できます。

スクロール可能または位置指定可能な結果セットを作成するときに、**更新検出**も指定する必要があります。この機能は、結果セットの外部から基礎となるデータベースに対して行われた変更を検出して示す結果セットの機能のことです。更新検出結果セットでは、結果セットが開かれているときにデータベースに対して行われた変更を知ることができ、基礎となるデータの動的ビューを利用できます。結果セット内の行の基礎となる列の値に対する変更を把握できます。**更新可能性**とは、結果セット内のデータを更新し、変更をデータベースにコピーできることを言います。これには、結果セットへの新しい行の挿入や既存の行の削除も含まれます。結果セットは更新可能、または読取り専用です。

結果セットのオブジェクト・タイプのまとめ

スクロール可能性と更新検出は、更新可能性とは無関係であり、3 つの結果セット・タイプと 2 つの同時実行性タイプの組合せで、次の 6 つの結果セット・カテゴリができます。

- 転送専用 / 読取り専用
- 転送専用 / 更新可能
- スクロール - 更新検出 / 読取り専用
- スクロール - 更新検出 / 更新可能
- スクロール - 非更新検出 / 読取り専用
- スクロール - 非更新検出 / 更新可能

例 4-2 に、スクロール - 更新検出および読取り専用の `ResultSet` オブジェクトを宣言する方法を示します。

例 4-2 スクロール - 更新検出、読取り専用の ResultSet オブジェクトの宣言

```
stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_READ_
ONLY);
```

> **注意：** 転送専用の更新可能結果セットでは、`ResultSet` オブジェクト内の特定の行に位置指定することを想定していません。`next` メソッドの使用による行の反復中にのみ、それらの行を更新できます。

Java アプリケーションからのデータの問合せ

この項では、JDeveloper を使用して Oracle Database 内のデータを問い合せる Java クラスを作成する方法について、次の項で説明します。

- JDeveloper でのデータを問い合せるメソッドの作成
- 接続メソッドおよび問合せメソッドのテスト

JDeveloper でのデータを問い合せるメソッドの作成

次の手順では、簡単な問合せメソッドを DataHandler.java クラスに追加する方法について説明します。JDeveloper 統合開発環境（IDE）で DataHandler.java が開かれていない場合は、アプリケーション・ナビゲータでダブルクリックして、Java ソース・エディタに表示します。

1. DataHandler クラスで、Statement および ResultSet JDBC クラスを使用するために、次の import 文を既存の import 文の後に追加します。

   ```
   import java.sql.Statement;
   import java.sql.ResultSet;
   ```

2. connection 宣言の後、Statement、ResultSet および String オブジェクトの変数を次のように宣言します。

   ```
   Statement stmt;
   ResultSet rset;
   String query;
   String sqlString;
   ```

3. getAllEmployees という名前のメソッドを作成します。このメソッドは、従業員情報をデータベースから取得するために使用します。メソッドのシグネチャを入力します。

   ```
   public ResultSet getAllEmployees() throws SQLException{
   ```

4. [Enter] を押して、このメソッドの閉じカッコ、およびメソッド・コードの入力を開始する新しい行を挿入します。

5. 以前に作成した getDBConnection メソッドをコールします。

   ```
   getDBConnection();
   ```

6. Connection インスタンスの createStatement メソッドを使用して、SQL 文を実行するコンテキストを指定し、ResultSet タイプを定義します。読取り専用、スクロール - 更新検出 ResultSet タイプを指定します。

   ```
   stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_READ_
   ONLY);
   ```

   Java コード・インサイト機能によって、文の構文が正しいことを確認できます。

7. 問合せを定義し、トレース・メッセージを印刷します。次のコードでは、簡単な問合せが使用されています。Employees 表のすべての行と列が戻され、データは従業員 ID 順に並べられます。

   ```
   query = "SELECT * FROM Employees ORDER BY employee_id";
   System.out.println("\nExecuting query: " + query);
   ```

8. 次のように、問合せを実行して結果を ResultSet インスタンスに取得します。

   ```
   rset = stmt.executeQuery(query);
   ```

9. ResultSet オブジェクトを戻します。

   ```
   return rset;
   ```

10. 作業内容を保存します。「**ファイル**」メニューで、「**すべて保存**」を選択します。

getAllEmployees メソッドのコードは、例 4-3 に示すようになります。

例 4-3 Connection、Statement、Query および ResultSet オブジェクトの使用

```
public ResultSet getAllEmployees() throws SQLException{
    getDBConnection();
    stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
           ResultSet.CONCUR_READ_ONLY);
    query = "SELECT * FROM Employees ORDER BY employee_id";
    System.out.println("\nExecuting query: " + query);
    rset = stmt.executeQuery(query);
    return rset;
}
```

接続メソッドおよび問合せメソッドのテスト

次の手順では、簡単な Java クラスを作成し、DataHandler.java クラスのメソッドをテストします。この段階でアプリケーションをテストするため、一時的に jdbcUrl 変数の値をデータベースの接続文字列に設定し、userid および password 変数の値を HR スキーマへのアクセスに必要な値（いずれの場合も hr）に設定します。

1. アプリケーション・ナビゲータから、Java ビジュアル・エディタで DataHandler.java クラスを開きます。
2. 次のように jdbcUrl、userid および password 変数を変更して、HR スキーマに必要な値が含まれるようにします。

   ```
   String jdbcUrl = "connect-string"
   String userid = "hr";
   String password = "hr";
   ```

   connect-string の例は、次のとおりです。

   ```
   jdbc:oracle:thin:@dbhost.companyname.com:1521:ORCL
   ```

   関連項目： 第 3 章の「接続関連の変数の宣言」を参照してください。

3. hr パッケージ内に新しい Java クラスを作成します。この Java クラスに JavaClient という名前を付け、public クラスにし、デフォルトのコンストラクタおよび main メソッドを生成します。スケルトン JavaClient.java クラスが作成され、Java ソース・エディタに表示されます。

   関連項目： Java クラス・ファイルの作成の詳細は、第 3 章を参照してください。

4. ResultSet パッケージをインポートします。

   ```
   import java.sql.ResultSet;
   ```

5. main メソッド宣言で、次のように例外処理を追加します。

   ```
   public static void main(String[] args) throws Exception{
   ```

6. デフォルトで作成された JavaClient オブジェクトを DataHandler オブジェクトで置き換えます。次の行を見つけます。

   ```
   JavaClient javaClient = new JavaClient();
   ```

   これを次のように置き換えます。

   ```
   DataHandler datahandler = new DataHandler();
   ```

7. `getAllEmployees` 問合せの結果を保持するように `ResultSet` オブジェクトを定義し、結果セットの行を反復して最初の 4 つの列 `Employee Id`、`First Name`、`Last Name` および `Email` を表示します。このことを行うには、次のコードを `main` メソッドに追加します。

```
ResultSet rset = datahandler.getAllEmployees();

while (rset.next()) {
System.out.println(rset.getInt(1) + " " +
  rset.getString(2) + " " +
  rset.getString(3) + " " +
  rset.getString(4));
}
```

8. `JavaClient.java` ファイルをコンパイルし、コンパイル・エラーをチェックします。このことを行うには、Java ソース・エディタで右クリックし、ショートカット・メニューで**「メイク」**を選択します。

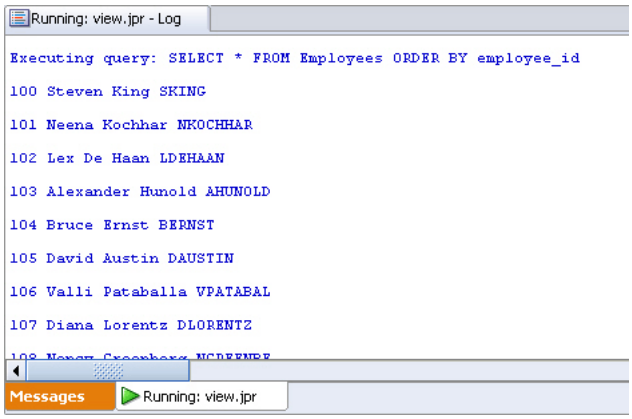
   コンパイルでエラーがない場合は、次のメッセージがログ・ウィンドウに表示されます。

   コンパイルが成功しました： 0 件のエラー、0 件の警告

9. `JavaClient.java` ファイルを実行します。このことを行うには、Java ソース・エディタのウィンドウで右クリックし、ショートカット・メニューで**「実行」**を選択します。

10. ログ・ウィンドウで出力を確認します。図 4-1 に示すようにトレース・メッセージが表示され、その後、`Employees` 表からの 4 つの列が表示されます。

図 4-1 ログ・ウィンドウでの問合せメソッドのテスト出力



11. アプリケーションのテストが終了したら、`DataHandler.java` の `jdbcUrl`、`userid` および `password` 変数を `null` に戻します。

関連項目：「接続関連の変数の宣言」

JSP ページの作成

HRApp アプリケーションでは、JavaServer Pages（JSP）テクノロジを使用してデータを表示します。JSP テクノロジによって、サーバーおよびプラットフォームに依存しない動的な Web コンテンツを簡単かつ迅速に作成できます。JSP ページには、`.jsp` 拡張子が付きます。この拡張子によって、ページを JSP コンテナで処理する必要があることが Web サーバーに指定されます。JSP コンテナによって、JSP タグおよびスクリプトレットが解析され、必要なコンテンツが生成され、結果が HTML または XML ページとしてクライアントに戻されます。

JSP ページを開発するには、次の項目の一部またはすべてを使用します。

- HTML タグ。動的に生成される Web ページを設計および書式設定します。
- 標準 JSP タグまたは Java ベースのスクリプトレット。ページ上に動的なコンテンツを生成する他のコンポーネントを呼び出します。
- カスタム・タグ・ライブラリの JSP タグ。ページ上に動的なコンテンツを生成します。

> **関連項目：** JSP に関する Sun 社のドキュメントを参照してください。
>
> `http://java.sun.com/products/jsp/`

この項では、このマニュアルのアプリケーションの JSP ページを作成する方法について説明します。

- ページ表示の概要
- 簡単な JSP ページの作成
- JSP ページへの静的コンテンツの追加
- JSP ページへのスタイルシートの追加

ページ表示の概要

このマニュアルで作成するアプリケーションでは、次の処理を行うために JSP ページが使用されます。

- データを表示します。
- 従業員の追加および従業員データの編集を行うユーザーが入力した入力データを保持します。
- ユーザー資格証明の検証およびデータベース内の従業員レコードの追加、更新および削除のアクションを処理するために必要なコードを保持します。

JSP ページは HTML または XML としてユーザーに表示されるので、静的な HTML および XML ページの場合と同じようにデータの表示を制御できます。標準的な HTML タグを使用してページを書式設定できます。たとえば、ヘッダーで `title` タグを使用して、ページに表示するタイトルを指定します。

ページ上の見出し、表、リストなどの項目に HTML タグを使用します。スタイルシートを使用して項目の表示を定義することもできます。JDeveloper を使用してアプリケーションを開発する場合、リストからスタイルを選択できます。

次の項では、サンプル・アプリケーションの JSP ページで使用される主な要素について説明します。

- JSP タグ
- スクリプトレット
- HTML タグ
- HTML フォーム

JSP タグ

このマニュアルのサンプル・アプリケーションでは、JSP タグが使用されています。単一の従業員レコードを保持するために使用されるアプリケーション・メソッドと JavaBean を保持する Java クラスを初期化するため、およびアプリケーション内の同じページまたは別のページにユーザーを転送するためです。

ページでは、`jsp:useBean` タグを使用して、アプリケーションに必要なすべてのメソッドを含むクラスを初期化し、`jsp:forward` タグを使用して、指定されたページにユーザーを転送します。JSP タグのコンポーネント・パレットから必要なタグをドラッグし、表示された対応するダイアログ・ボックスでタグのプロパティを入力できます。

関連項目： JavaBeans の詳細は、
`http://java.sun.com/products/javabeans/` を参照してください。

スクリプトレット

スクリプトレットは、データベース上で動作する Java メソッドを実行し、JSP ページでその他の処理を実行するために使用されます。コンポーネント・パレットからスクリプトレット・タグ・コンポーネントをドラッグし、ページにドロップできます。これにより、スクリプトレット・コードを入力することができるようになります。JDeveloper では、スクリプトレットのコードはスクリプトレット・ソース・エディタ・ダイアログ・ボックスに入力します。

このアプリケーションでは、様々な処理にスクリプトレットを使用します。たとえば、あるスクリプトレットは、`Employees` 表内のすべての従業員を含む `ResultSet` オブジェクトを戻す `DataHandler` メソッドをコールします。このオブジェクトを使用して、そのデータを JSP ページに表示できます。別の例では、あるスクリプトレットは、同じ `ResultSet` オブジェクトを反復して表の行の各項目を表示するために使用されます。

HTML タグ

HTML タグは一般的に、見出しや表など、ユーザー・インタフェースの動的ではない部分のレイアウトおよび表示に使用されます。JDeveloper では、「表」コンポーネントをコンポーネント・パレットからページにドラッグ・アンド・ドロップできます。表の行および列の数を指定する必要があります。すべての表タグが自動的に作成されます。

HTML フォーム

HTML フォームは、Web ページ上で、ユーザーとの相互作用またはユーザーからの情報の収集に使用されます。`FORM` 要素は、ページ上のコントロールのコンテナとして機能し、フォーム入力の処理に使用されるメソッドを指定します。

表示する従業員を選択するフィルタ・コントロールの場合、`employees.jsp` ページ自体がフォームを処理します。ログイン、挿入、編集および削除処理の場合、これらのフォームを処理するために追加の JSP ページが作成されます。このアプリケーションの JSP ページの相互関係については、図 1-2 を参照してください。

JSP ページでフォームを追加するには、HTML タグのコンポーネント・パレットからフォームを選択します。ページのフォーム・コンポーネントの外側またはフォームが含まれていないページでコントロールを追加しようとすると、JDeveloper によって、そのコントロールを格納するフォーム・コンポーネントを追加するように求められます。

簡単な JSP ページの作成

次の手順では、簡単な JSP ページを作成する方法について説明します。

1. アプリケーション・ナビゲータで、**View** プロジェクトを右クリックし、ショートカット・メニューで**「新規」**を選択します。
2. 「新規ギャラリ」で、**「フィルタ方法」**リストから**「すべてのテクノロジ」**を選択します。
3. **「Web 層」**ノードを開き、**「JSP」**を選択します。
4. **「項目」**リストで、**「JSP」**を選択し、**「OK」**をクリックします。JSP の作成ウィザードが表示されます。
5. 「JSP ファイル」画面で、JSP ページの名前を入力し、**「JSP ページ」**を選択します。
6. 「エラー・ページ・オプション」画面で、**「このファイルの捕捉されない例外の処理にエラー・ページを使用しない」**を選択します。

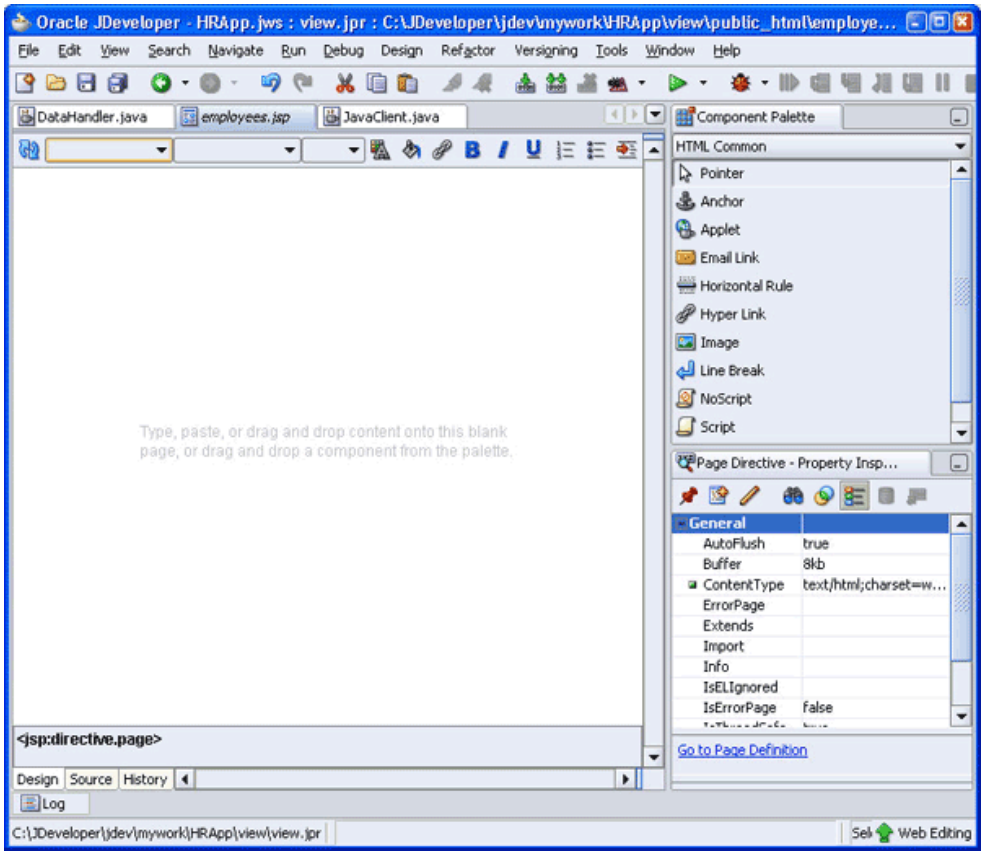
注意： アプリケーションによってスローされた Java 例外をユーザーにわかりやすくするために、エラー・ページを作成してください。エラー・ページを作成する際は、発生する可能性があるアプリケーション・エラーについての説明をユーザーにわかりやすく記述します。ただし、これについては、アプリケーションの必須機能ではないため、このマニュアルでは説明していません。

7. この段階では、JSP タグ・ライブラリを追加しないでください。「タグ・ライブラリ」画面で**「次へ」**をクリックします。
8. **「HTML オプション」**画面でデフォルト設定のまま、**「終了」**をクリックします。
JSP/HTML ビジュアル・エディタに新しいページが開き、テキストおよびコンポーネントを Web ページに追加できるようになります。

JSP ページへの静的コンテンツの追加

JDeveloper には、JSP/HTML ビジュアル・エディタの右側に、コンポーネント・パレットとプロパティ・インスペクタがあります。ページの一番下にある「設計」タブの隣の「ソース」タブをクリックして、JSP ソース・エディタを使用することもできます。コンポーネント・パレットではコンポーネントをページに追加でき、プロパティ・インスペクタではコンポーネントのプロパティを設定できます。図 4-2 に、ビジュアル・エディタの空白ページを示します。

図 4-2 JDeveloper のビジュアル・ソース・エディタでの JSP ページへのコンテンツの追加



次の手順では、employees.jsp ページにテキストを追加する方法について説明します。ビジュアル・エディタを使用して JSP を変更します。ビジュアル・エディタは WYSIWYG エディタのようなものであり、これを使用してコンテンツを変更できます。

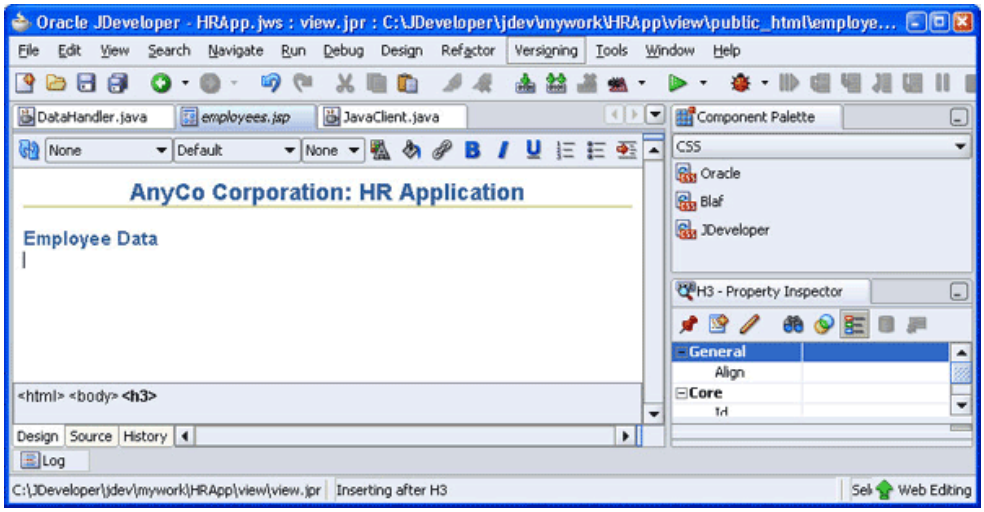
1. employees.jsp がビジュアル・エディタで開かれている状態で、ページの一番上の行に、**AnyCo Corporation: HR Application** と入力します。ページの一番上にある左側のスタイルのリストから、**Heading 2** を選択します。
2. 追加した見出しにカーソルを置いたまま、**「設計」**メニューから**「位置」**を選択し、次に、**「中央揃え」**を選択します。
3. 同様に、新しい行で **Employee Data** と入力し、**Heading 3** スタイルで書式設定します。ページの左側に配置します。

JSP ページへのスタイルシートの追加

見出し、テキストなどの要素が Web ページで使用されるフォントや色などの表示特性と調和して書式設定されるように、スタイルシート参照をページに追加できます。次のように、スタイルシートをページに追加できます。

1. employees.jsp がビジュアル・エディタで開かれている状態で、コンポーネント・パレットの右上にあるリスト矢印をクリックして、**「CSS」**を選択します。
2. **「CSS」**リストから、**「JDeveloper」**をページにドラッグします。スタイルシートを選択するとすぐにページに追加され、ページは JDeveloper スタイルで書式設定されます。図 4-3 に、前の項のコンテンツが追加され、JDeveloper スタイルシートが適用された JSP ページを示します。

図 4-3 JSP ページへの静的コンテンツの追加



注意： JDeveloper バージョン 10.1.3 では、JSP の作成ウィザードでの JSP ページの作成中に、スタイルシートを JSP ページに関連付けることができます。ただし、ページにドラッグ・アンド・ドロップするのではなく、JSP ページに適用するスタイルシートを参照して見つける必要がある点のみ異なります。

JSP ページへの動的コンテンツの追加 : データベースの問合せ結果

この項では、次の項目について説明します。

- DataHandler クラスを初期化する JSP useBean タグの追加
- 結果セットの作成
- 結果セットを表示する表の JSP ページへの追加

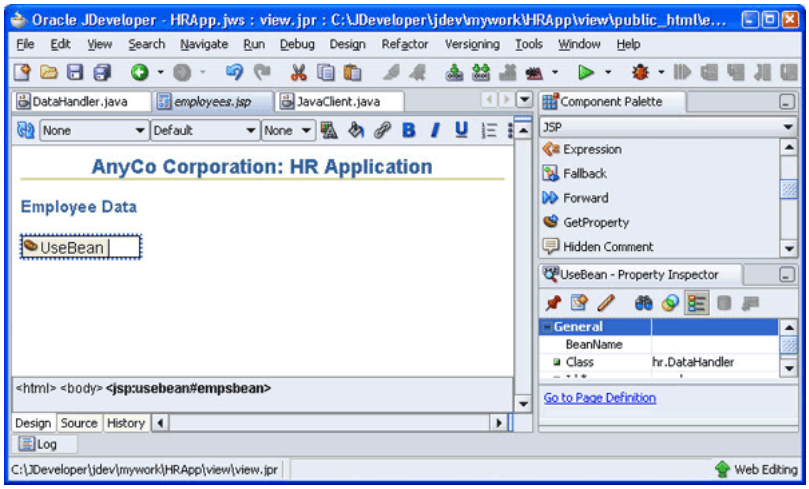
DataHandler クラスを初期化する JSP useBean タグの追加

jsp:useBean タグによって、ページで実行されるメソッドを保持するクラスが識別および初期化されます。jsp:useBean タグを追加するには、次の手順を実行します。

1. ビジュアル・エディタで employees.jsp を開きます。
2. コンポーネント・パレットで、コンポーネントの「**JSP**」セットを選択します。リストをスクロールして、「**UseBean**」を選択します。次に、それを JSP ページの見出しの下にドラッグ・アンド・ドロップします。
3. 「UseBean の挿入」ダイアログ・ボックスで、empsbean を「**ID**」として入力し、「**クラス**」には hr.DataHandler クラスを参照して選択します。「**有効範囲**」を「セッション」に設定し、「タイプ」および「Bean 名」フィールドは空白のままにします。
4. 「**OK**」をクリックして、ページにタグを作成します。

図 4-4 に、employees.jsp ページでの useBean タグの表示を示します。

図 4-4 employees.jsp ファイルでの useBean の表示



結果セットの作成

次の手順では、スクリプト要素をページに追加して getAllEmployees メソッドをコールし、戻された結果セット・データを保持する方法について説明します。この問合せは、DataHandler クラスで定義され、jsp:useBean タグを使用してページで初期化されます。

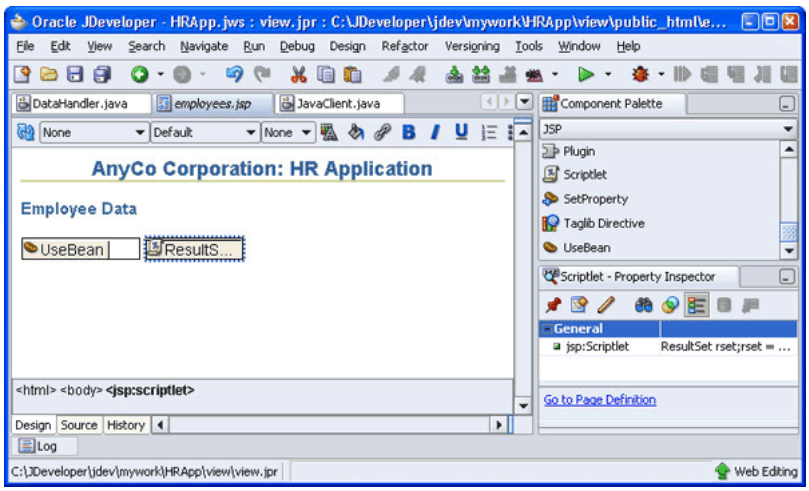
1. ビジュアル・エディタで employees.jsp ページを開きます。コンポーネント・パレットの「JSP」部分で、「**スクリプトレット**」を選択して、JSP ページの UseBean の表示の横にドラッグ・アンド・ドロップします。

2. 「スクリプトレットの挿入」ダイアログ・ボックスで、次のコード行を入力します。このコード行は getAllEmployees メソッドをコールし、ResultSet オブジェクトを生成します。

```
ResultSet rset;
rset = empsbean.getAllEmployees();
```

「OK」をクリックします。スクリプトレットの表示は、図 4-5 に示すようにページに表示されます。

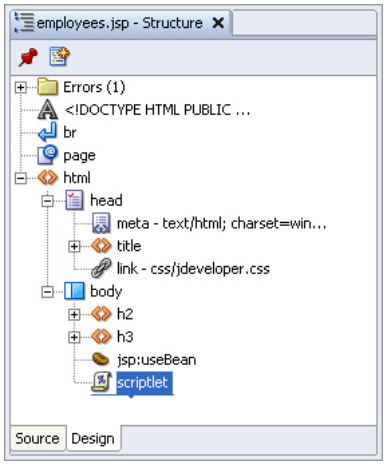
図 4-5 JSP ページでのスクリプトレットの表示



3. ビジュアル・エディタの一番下にある「ソース」タブを選択して、ページに対してこれまでに作成されたコードを表示します。ResultSet の下の波線は、コードにエラーがあることを示しています。

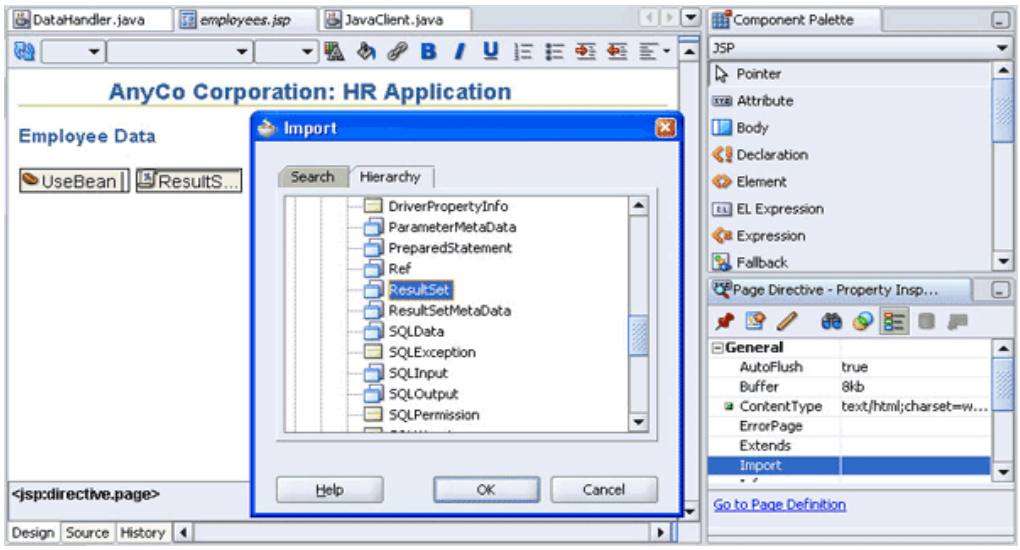
4. 左側の構造ウィンドウでも、ページにエラーがあることを示しています。ウィンドウの一番上にスクロールし、「JSP エラー」ノードを開きます。図 4-6 に、コードのエラーが構造ウィンドウにどのように表示されるかを示します。

図 4-6 構造ウィンドウでのエラーの表示



5. `ResultSet` パッケージをインポートする必要があります。このことを行うには、構造ウィンドウで「**page**」ノードをクリックして、プロパティ・インスペクタでページのプロパティを表示します。
6. 「**インポート**」プロパティの右の空のボックスをクリックします。省略記号（**...**）をクリックします。図 4-7 に示すように、「インポート」ダイアログ・ボックスが表示されます。

図 4-7 JDeveloper でのパッケージのインポート



7. インポート・リストで、「**階層**」タブを選択し、「**java**」ノード、「**sql**」ノードの順に開き、**ResultSet** を選択します。「**OK**」をクリックします。
8. 「ソース」タブで、`import` 文がページのコードに追加されているかどうかを確認します。構造ウィンドウのリストからエラーが消えています。次の項に進む前に、「**設計**」タブを選択してこのページの設計ビューに戻ります。

結果セットを表示する表の JSP ページへの追加

次の手順では、JSP ページに表を追加して、`getAllEmployees` 問合せの結果を表示する方法について説明します。

1. `employees.jsp` ページがビジュアル・エディタで開かれていない場合は、アプリケーション・ナビゲータでダブルクリックして開き、「設計」タブで作業します。`employees.jsp` ファイルがビジュアル・エディタで開かれている状態で、カーソルをスクリプトレットの後に置き、コンポーネント・パレットの「HTML Common」ページから、「**表**」コンポーネントを選択します。
2. 「表の挿入」ダイアログ・ボックスで、1 行および 6 列を指定します。すべてのレイアウト・プロパティをデフォルトのままにします。「**OK**」をクリックします。
3. ページに表示された表の行に、各列の見出しのテキストを次のように入力します。**First Name**、**Last Name**、**Email**、**Job**、**Phone**、**Salary**。**Heading 4** を使用して、列名を書式設定します。
4. 今度は表の各列に対して戻された値を表示するために、出力用のスクリプト要素を追加します。このことを行うには、次のように表を選択します。表の上部の枠線にカーソルを置き、カーソルのイメージが表のイメージに変わったらクリックします。JSP のコンポーネント・パレットから、「**スクリプトレット**」を選択します。（スクリプトレットを表にドラッグする必要はありません。自動的に挿入されます。）

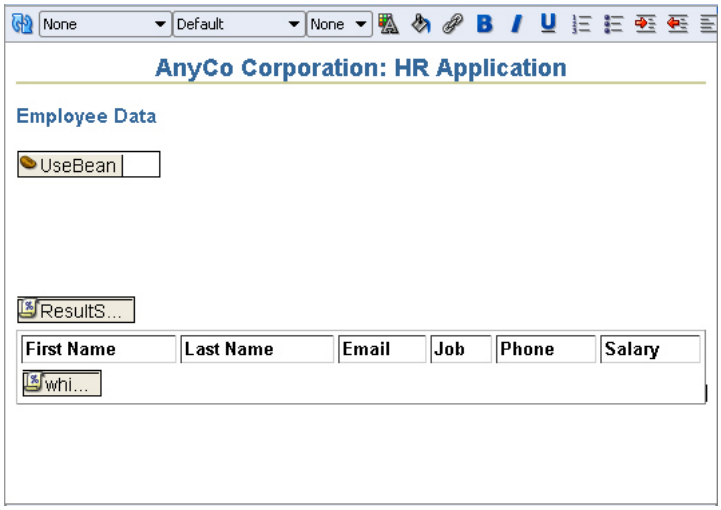
5. 「スクリプトレットの挿入」ダイアログ・ボックスで、次のコード行を入力します。

```
while (rset.next ())
{
out.println("<tr>");
out.println("<td>" +
  rset.getString("first_name") + "</td><td> " +
  rset.getString("last_name") + "</td><td> " +
  rset.getString("email") + "</td><td> " +
  rset.getString("job_id") + "</td><td>" +
  rset.getString("phone_number") + "</td><td>" +
  rset.getDouble("salary") + "</td>");
out.println("</tr>");
}
```

6. 「OK」をクリックします。

図 4-8 に、作成された JSP ページを示します。

図 4-8 JSP ページでの表



問合せの結果セットのフィルタ

特定のパラメータまたは条件で問合せの結果をフィルタできます。アプリケーションのユーザーがデータ・フィルタをカスタマイズできるようにすることもできます。このマニュアルで作成するサンプル・アプリケーションでは、次の手順で問合せ結果をフィルタします。

1. 必要なフィルタ・セットの判別

   問合せフィールドにフィルタ基準を入力することによって、ユーザーは表示する従業員レコードのセットを指定できます。この場合は、検索する名前の一部です。`employees.jsp` ページでは、この入力はフォーム・コントロールによって受け入れられ、処理されます。

2. 問合せの `ResultSet` を戻すメソッドの作成

   SQL 問合せ文の作成に、ユーザー入力文字列を使用します。この文によって、ユーザーが入力した文字列が名前に含まれるすべての従業員が選択されます。問合せでは、姓と名の両方でこの文字列が検索されます。

3. 問合せの結果の表示

   このことは、`employees.jsp` ページにコードを追加して、フィルタされる問合せを実行するメソッドを使用することによって実行されます。

この項では、問合せデータのフィルタについて、次の項で説明します。

- 結果をフィルタする Java メソッドの作成
- 問合せフィルタ・メソッドのテスト
- JSP ページへのフィルタ・コントロールの追加
- JSP ページでのフィルタされたデータの表示

結果をフィルタする Java メソッドの作成

次の手順では、getEmployeesByName メソッドを作成する方法について説明します。このメソッドによって、ユーザーは従業員を姓または名でフィルタできます。

1. アプリケーション・ナビゲータから、Java ビジュアル・エディタで DataHandler.java クラスを開きます。
2. getAllEmployees メソッドの後に、次のように getEmployeesByName メソッドを宣言します。

```
public ResultSet getEmployeesByName(String name) throws SQLException {

}
```

3. メソッド本体で次のコードを追加し、検索ヒット数が増えるように、名前を大文字に変換します。

```
name = name.toUpperCase();
```

4. データベースに接続するメソッドをコールします。

```
getDBConnection();
```

5. ResultSet タイプを指定し、問合せを作成します。

```
stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                            ResultSet.CONCUR_READ_ONLY);
query =
"SELECT * FROM Employees WHERE UPPER(first_name) LIKE \'%" + name + "%\'" +
" OR UPPER(last_name) LIKE \'%" + name + "%\' ORDER BY employee_id";
```

6. トレース・メッセージを出力します。

```
System.out.println("\nExecuting query: " + query);
```

7. 問合せを実行し、以前と同様に結果セットを戻します。

```
rset = stmt.executeQuery(query);
return rset;
```

8. ファイルを保存し、コンパイルしてコンパイル・エラーがないことを確認します。

問合せフィルタ・メソッドのテスト

「接続メソッドおよび問合せメソッドのテスト」で作成した JavaClient.java クラスを使用して、getEmployeesByName メソッドをテストできます。次の手順で説明するように、getEmployeesByName メソッドを追加して問合せ結果を表示する必要があります。

1. Java ソース・エディタで JavaClient.java クラスを開きます。

2. 結果セットに getAllEmployees 問合せからの結果が表示された後、次のように条件問合せの結果セットを定義します。

```
rset = datahandler.getEmployeesByName("King");

System.out.println("\nResults from query: ");

while (rset.next()) {
  System.out.println(rset.getInt(1) + " " +
  rset.getString(2) + " " +
  rset.getString(3) + " " +
  rset.getString(4));
}
```

3. この段階でアプリケーションをテストするため、一時的に、DataHandler クラスの jdbcUrl、userid および password 変数の値を調整して、HR スキーマに必要な値を指定します。ファイルを保存し、コンパイルして構文エラーをチェックします。

> **注意：** テスト後、userid、password および jdbcUrl の値を null に戻してください。詳細は、「接続関連の変数の宣言」を参照してください。

4. コードをテスト実行するには、Java ソース・エディタで右クリックし、ショートカット・メニューで**「実行」**を選択します。ログ・ウィンドウには、最初に getAllEmployees メソッドの結果が表示され、次に、getEmployeesByName("xxx") 問合せの結果が表示されます。ここでは、xxx を「King」に設定して、フィルタ機能をテストしています。実際の操作では、このパラメータはアプリケーションのユーザーが指定した値に設定され、検索がフィルタされます。

JSP ページへのフィルタ・コントロールの追加

フィルタ基準を受け入れてフィルタ結果を表示するには、employees.jsp ページを変更する必要があります。次の手順では、ユーザーからの入力を受け入れる employees.jsp ページにフォーム要素およびコントロールを追加して、従業員を名前でフィルタします。

1. employees.jsp ページがビジュアル・エディタに表示されている状態で、カーソルを useBean タグとスクリプトレットの間に置きます。
2. コンポーネント・パレットの「HTML Forms」ページで、**「フォーム」**を選択します。
3. 「フォームの挿入」ダイアログ・ボックスで、「アクション」フィールドの下矢印を使用して、**employees.jsp** を選択します。他のフィールドは空のままにし、**OK」**をクリックします。

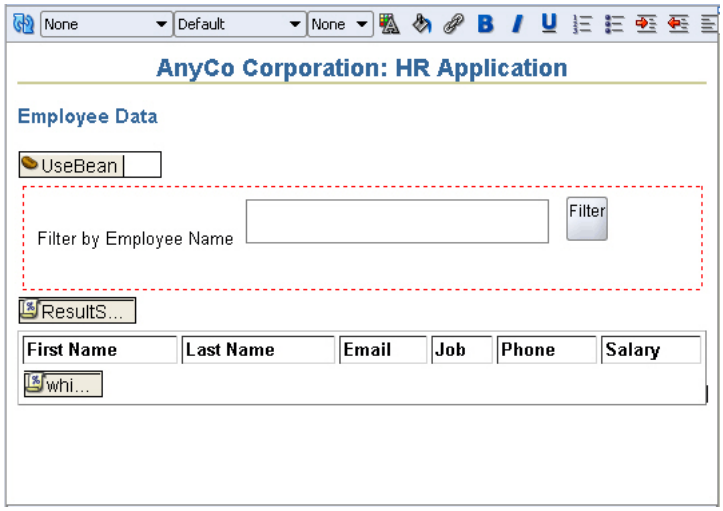
   フォームがビジュアル・エディタのページに表示され、点線の四角形で表されます。
4. コンポーネント・パレットの「HTML Forms」ページで、**「テキスト・フィールド」**にスクロールします。それを選択し、フォーム・コンポーネント内にドラッグ・アンド・ドロップします。「テキスト・フィールドの挿入」ダイアログ・ボックスで、**「名前」**フィールドの値として query と入力し、**「OK」**をクリックします。フォーム内にテキスト・フィールド・ボックスが表示されます。ユーザーはこのフィールドにフィルタ基準を入力できます。
5. テキスト・フィールドの左にカーソルを置き、次のテキストを追加します。

   Filter by Employee name:
6. コンポーネント・パレットの「HTML Forms」ページで、**「Submit Button」**にスクロールします。それを選択し、**フォーム**・コンポーネント内のテキスト・フィールドの右にドラッグします。

7. 「送信ボタンの挿入」ダイアログ・ボックスで、**「名前」**フィールドは空のままで**「値」**フィールドの値として `Filter` と入力し、**「OK」**をクリックします。

   図 4-9 に、`employees.jsp` ファイル内のこれらの HTML フォーム・コンポーネントを示します。

図 4-9 JSP ページ内の HTML フォーム・コンポーネント



JSP ページでのフィルタされたデータの表示

前の項で、ユーザー入力を受け入れるテキスト・フィールド・コンポーネントを JSP ページに作成しました。このテキスト・フィールドで、ユーザーは従業員の名前をフィルタするために使用する文字列を指定できます。また、送信ボタンも追加しました。

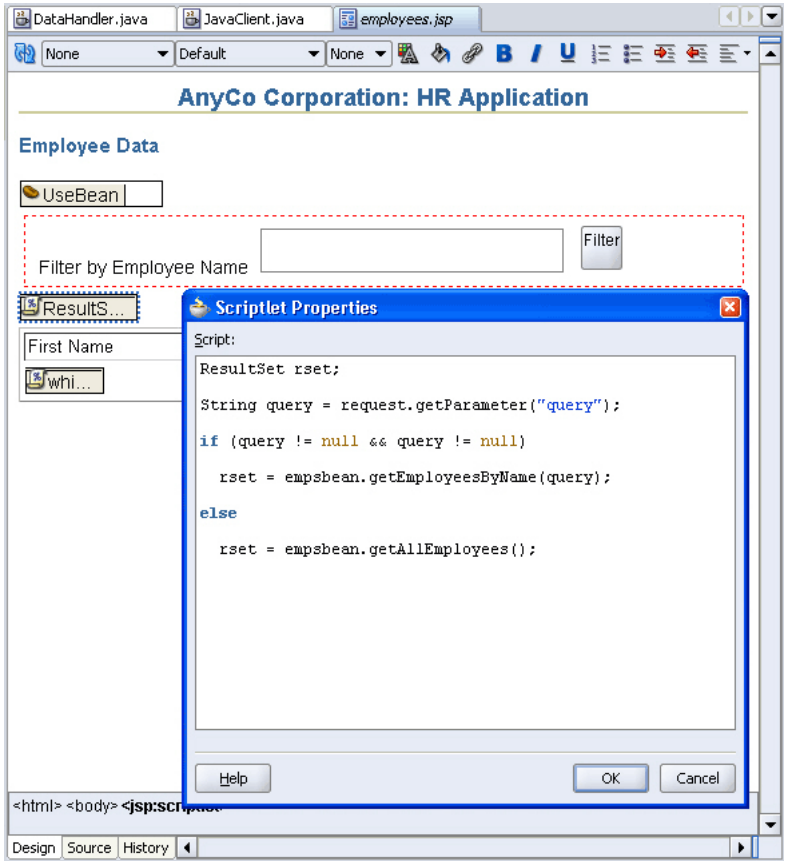
次の手順では、`employees.java` ファイル内のスクリプトレットにコードを追加して、`getEmployeesByName` メソッドを使用できるようにします。このメソッドは、結果をフィルタするための値をユーザーが送信した場合にのみ使用されます。このフィルタ基準を指定しない場合は、`getAllEmployees` メソッドが使用されます。

1. ビジュアル・エディタで `employees.jsp` ファイルを開きます。
2. ページの**スクリプトレット**・タグ（表内ではありません）をダブルクリックして、「プロパティ」ダイアログ・ボックスを開きます。次のようにコードを変更します。

```
ResultSet rset;
String query = request.getParameter("query");
if (query != null && query != null)
  rset = empsbean.getEmployeesByName(query);
else
  rset = empsbean.getAllEmployees();
```

   図 4-10 に、「スクリプトレットのプロパティ」ダイアログ・ボックスを使用してコードを変更する方法を示します。

図 4-10「スクリプトレットのプロパティ」ダイアログ・ボックスの使用



3. 「OK」をクリックします。
4. ファイルを保存します。

アプリケーションへのログイン機能の追加

サンプル・アプリケーションで使用されるログイン機能は、アプリケーション管理セキュリティの簡単な例です。完全な Java EE セキュリティ実装ではありませんが、サンプル・アプリケーションでの例として使用されます。

この簡単なログイン機能を実装するには、次の処理を実行する必要があります。

- ユーザーを認証するメソッドの作成
- ログイン・ページの作成
- 失敗したログインのエラー・レポートの準備
- ログイン・インタフェースの作成
- ログイン操作を処理する JSP ページの作成

ユーザーを認証するメソッドの作成

次の手順では、DataHandler.java クラスにメソッドを作成します。このメソッドでは、userid および password に対して入力された値と、データベース・スキーマが必要とする値が一致するかどうかをチェックすることによってユーザーを認証します。

1. ソース・エディタで DataHandler.java クラスを開きます。
2. ユーザーが入力した userid、password および host の値が有効かどうかをチェックする authenticateUser という名前のメソッドを作成します。

```
public boolean authenticateUser(String jdbcUrl, String userid, String password,
  HttpSession session) throws SQLException {

}
```

3. 波線の下線と HttpSession のクラスをインポートする必要があるというメッセージが表示されます。[Alt] キーを押しながら [Enter] キーを押して、javax.servlet.http.HttpSession クラスをインポートします。
4. メソッド本体で、次のようにコールの jdbcUrl、userid および password 値を現在のオブジェクトの属性に割り当てます。

```
this.jdbcUrl= jdbcUrl;
this.userid = userid;
this.password = password;
```

5. 指定された値でデータベースへの接続を試行します。成功した場合は、値 true を戻します。この値を、次のように try ブロックで囲みます。

```
try {
  OracleDataSource ds;
  ds = new OracleDataSource();
  ds.setURL(jdbcUrl);
  conn = ds.getConnection(userid, password);
  return true;
}
```

> **関連項目：** try および catch ブロックの使用の詳細は、第 5 章の「例外処理」を参照してください。

6. ログイン資格証明が一致しない場合を処理するには、try ブロックの後に catch ブロックを追加します。このブロックのコードでは、ログ・メッセージを出力し、エラー・メッセージを設定します。このエラー・メッセージは、ログイン試行が失敗した場合にユーザーに表示されます。jdbcUrl、userid および password 変数は null に戻され、メソッドは値 false を戻します。このことを行うには、次のコードを入力します。

```
catch ( SQLException ex )  {
  System.out.println("Invalid user credentials");
  session.setAttribute("loginerrormsg", "Invalid Login. Try Again...");
  this.jdbcUrl = null;
  this.userid = null;
  this.password = null;
  return false;
}
```

例 4-4 に、完成したコードを示します。

例 4-4 ユーザー検証の実装

```
public boolean authenticateUser(String userid, String password,
  HttpSession session) throws SQLException {

  this.jdbcUrl = jdbcUrl;
  this.userid = userid;
```

```
  this.password = password;
  try {
    OracleDataSource ds;
    ds = new OracleDataSource();
    ds.setURL(jdbcUrl);
    conn = ds.getConnection(userid, password);
    return true;
  } catch ( SQLException ex )  {
  System.out.println("Invalid user credentials");
  session.setAttribute("loginerrormsg", "Invalid Login. Try Again...");
  this.jdbcUrl = null;
  this.userid = null;
  this.password = null;
  return false;
  }
}
```

ログイン・ページの作成

次の手順では、login.jsp ページを作成します。このページで、ユーザーは作業を行うスキーマのログイン詳細を入力します。

1. View プロジェクトで、新しい JSP ページを作成します。名前を login.jsp に変更し、その他はすべてデフォルトをそのまま使用します。JSP/HTML ビジュアル・エディタで新しいページが開き、テキストおよびコンポーネントを Web ページに追加できるようになります。
2. JDeveloper スタイルシートをページに適用します。
3. このページに、以前に設定したものと同じ見出し **AnyCo Corporation: HR Application** を付け、**Heading 2** スタイルを割り当て、ページの中央に配置します。
4. 次の行で、**Application Login** と入力し、**Heading 3** スタイルを適用します。この見出しをページの左側に配置します。

失敗したログインのエラー・レポートの準備

次の手順では、ユーザー・ログインが失敗したときにエラー・メッセージを表示する機能を login.jsp ページに追加します。login.jsp ページで使用されるスクリプトレットおよび式によって、エラー・メッセージを保持する変数が設定されます。ユーザー・ログインが失敗した場合、接続メソッドによってセッションのメッセージが設定されます。このページではそのようなメッセージがあるかどうかをチェックし、存在する場合にはメッセージを表示します。

1. login.jsp ページがビジュアル・エディタで開かれている状態で、カーソルをこのページのテキストの後に置きます。次に、コンポーネント・パレットの「JSP」ページから、**「スクリプトレット」**要素をパレットからページにドラッグ・アンド・ドロップします。
2. 「スクリプトレットの挿入」ダイアログ・ボックスで、次のコードを入力します。

```
String loginerrormsg = null;
loginerrormsg = (String) session.getAttribute("loginerrormsg");
if (loginerrormsg != null) {
```

3. 別のスクリプトレットを同じ方法で追加し、今度は 1 つの閉じカッコ（}）のみを「スクリプトレットの挿入」ダイアログ・ボックスで入力します。
4. 2 つのスクリプトレットの間にカーソルを置き、[Enter] を押して新しい行を作成します。新しい行に **Heading 4** スタイルを適用します。
5. カーソルを新しい行に置いたまま、コンポーネント・パレットの「JSP」ページで、**「式」**をクリックします。
6. 「式の挿入」ダイアログ・ボックスで、loginerrormsg と入力します。

7. login.jsp ページに追加されたコードを表示するには、ビジュアル・エディタの下の**「ソース」**タブを選択します。コードは次のように表示されます。

```
<%
    String loginerrormsg = null;
    loginerrormsg = (String) session.getAttribute("loginerrormsg");
    if (loginerrormsg != null) {
%>
<h4>
  <%=  loginerrormsg %>
</h4>
<%
}
%>
```

次の項に進む前に、**「設計」**タブを選択してこのページの設計ビューに戻ります。

ログイン・インタフェースの作成

これらの手順では、ユーザーがログイン詳細を入力するフィールドを login.jsp ページに追加します。

1. login.jsp ページがビジュアル・エディタで開かれていない場合は、アプリケーション・ナビゲータでダブルクリックして開き、「設計」タブが選択されていることを確認します。
2. カーソルを 2 番目のスクリプトレットの後に置き、コンポーネント・パレットの「HTML Forms」ページで、**「フォーム」**を選択します。フォームがビジュアル・エディタのページに表示され、点線の四角形で表されます。
3. コンポーネント・パレットの「HTML Forms」ページで、**「フォーム」**を選択します。「フォームの挿入」ダイアログ・ボックスで、**「アクション」**フィールドの値として login_action.jsp と入力します。このファイルは、**login.jsp** ファイルでのユーザー入力を処理するために使用されます。（このページはまだ作成されていないため、リストから選択できません。）他のフィールドは空のままにし、**OK」**をクリックします。

   フォームがビジュアル・エディタのページに表示され、点線の四角形で表されます。
4. **表**をページに追加します。フォーム内に配置します。3 行および 2 列のレイアウトを指定し、その他のレイアウトのデフォルトはそのまま使用します。
5. 3 行の最初の列に、ユーザーに表示するテキストとして次のように入力します。

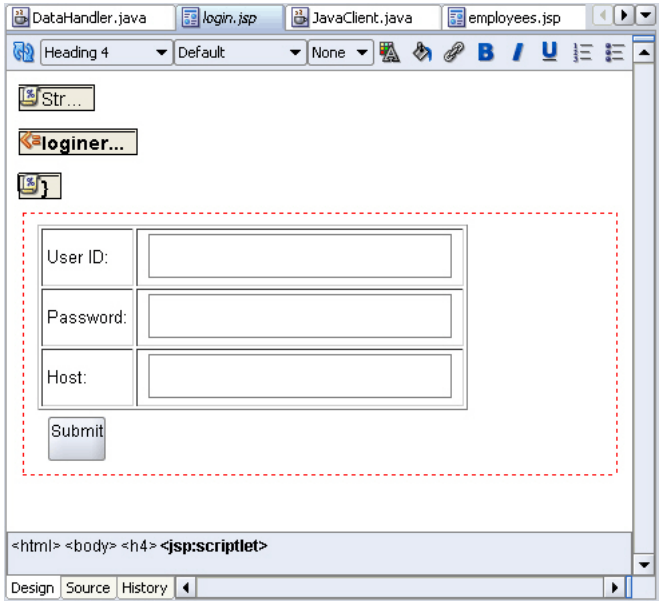
   User ID:

   Password:

   Host:
6. コンポーネント・パレットの「HTML」ページから、**「テキスト・フィールド」**を User ID: セルの右のセルにドラッグします。「テキスト・フィールドの挿入」ダイアログ・ボックスで、**Name** プロパティの値として userid と入力します。他のフィールドは空のままにし、**「OK」**をクリックします。
7. 同じ方法で、**「テキスト・フィールド」**を Password: セルの右のセルに追加し、**「名前」**プロパティの値として password と入力します。同様に、**「テキスト・フィールド」**を Host: セルの右のセルに追加し、**「名前」**プロパティの値として host と入力します。
8. **「発行」**ボタンを表の下のフォームにドラッグします。ボタンの**「値」**プロパティに Submit と入力します。

図 4-11 に示すように login.jsp ページが表示されます。

図 4-11 ログイン・ページ



ログイン操作を処理する JSP ページの作成

次の手順では、login_action.jsp ページを作成します。これは、ログイン操作を処理する非表示のページです。

1. JSP ページを作成し、login_action.jsp という名前を付けます。JSP ページのすべてのデフォルト設定をそのまま使用します。
2. login_action.jsp がビジュアル・エディタで開かれている状態で、コンポーネント・パレットの「JSP」ページから、Page Directive コンポーネントをページにドラッグします。「Page Directive の挿入」ダイアログ・ボックスの**「インポート」**フィールドで、**java.sql.ResultSet** をインポートするために参照します。**「OK」**をクリックします。
3. **jsp:usebean** タグをページにドラッグします。empsbean を **ID** として入力し、**hr.DataHandler** を**クラス**として参照して選択します。**「有効範囲」**を「セッション」に設定し、**「OK」**をクリックします。
4. カーソルを useBean タグの後に置き、**スクリプトレット**をページに追加します。次のコードを「スクリプトレットの挿入」ダイアログ・ボックスに入力し、**「OK」**をクリックします。

```
boolean userIsValid = false;
String host = request.getParameter("host");
String userid = request.getParameter("userid");
String password = request.getParameter("password");
String jdbcUrl = "jdbc:oracle:thin:@" + host;
userIsValid = empsbean.authenticateUser(jdbcUrl, userid, password, session);
```

5. 別の**スクリプトレット**を追加し、次のコードを追加します。

```
if (userIsValid){
```

6. コンポーネント・パレットの「JSP」ページで、**Forward** を見つけてページにドラッグし、**jsp:forward** タグをページに追加します。「Forward の挿入」ダイアログ・ボックスで、**employees.jsp** と入力します。

7. 別のスクリプトレットを追加し、次のコードを入力します。

```
} else {
```

8. 別の **jsp:forward** タグを追加し、今度は `login.jsp` に移動します。
9. 最後の**スクリプトレット**を追加し、閉じカッコ（}）を入力します。
10. 作業内容を保存します。

`login_action.jsp` に追加されたコードを表示するには、**「ソース」**タブを選択します。次のようなコードが表示されます。

```
<body>
<%@ page import="java.sql.ResultSet"%><jsp:useBean id="empsbean"
                                                    class="hr.DataHandler"
                                                    scope="session"/>
<%boolean userIsValid = false;
String host = request.getParameter("host");
String userid = request.getParameter("userid");
String password = request.getParameter("password");
String jdbcUrl = "jdbc:oracle:thin:@" + host + ":1521:ORCL";
userIsValid = empsbean.authenticateUser(jdbcUrl, userid, password, session);%><%if
(userIsValid){%><jsp:forward page="employees.jsp"/><%if (userIsValid){%><jsp:forward
page="login.jsp"/><%}%>
</body>
```

JSP ページのテスト

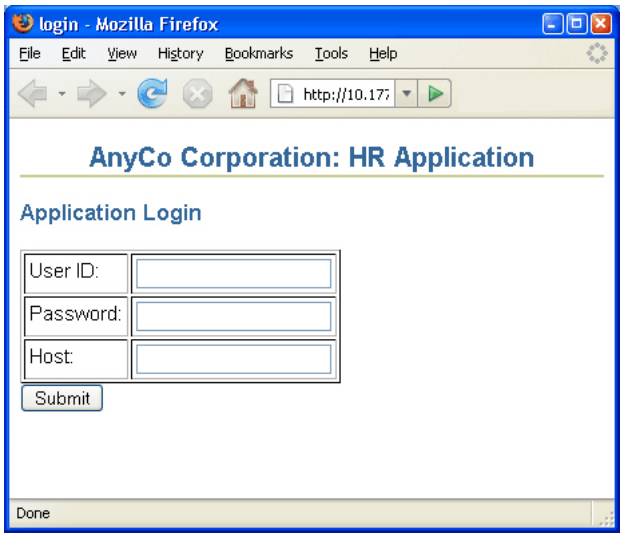
ログイン・ページと従業員のフィルタをテストするには、次の手順を実行します。

1. アプリケーション・ナビゲータで、**View** プロジェクトを右クリックし、**「実行」**を選択します。

   プロジェクトのデフォルトの実行ターゲットの指定を求められる場合があります。ここでは、実行ターゲットに `login.jsp` を設定します。後でプロジェクトのプロパティを変更して、デフォルトの実行ターゲット・ページを任意のページにすることができます。

   図 4-12 に示すように、ログイン・ページがブラウザに表示されます。

図 4-12 ブラウザでのサンプル・アプリケーションのログイン・ページ



2. データベースについて次のログイン詳細を入力し、「**発行**」をクリックします。

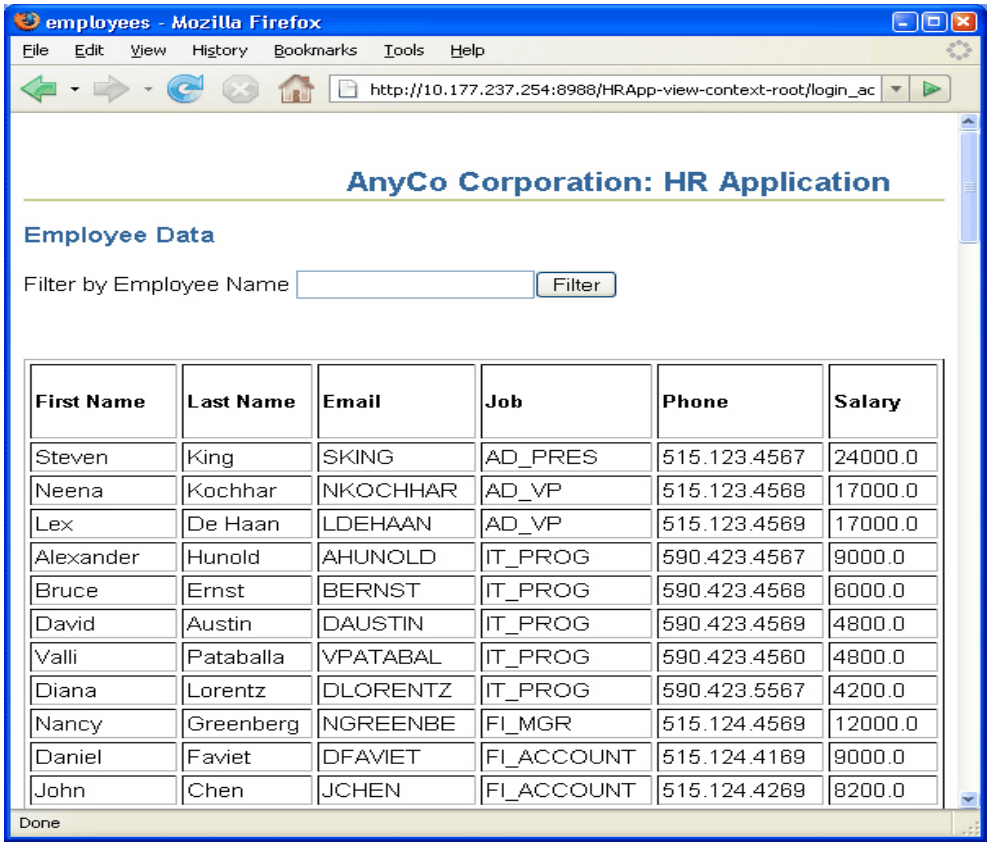
   User ID: `hr`

   Password: `hr`

   Host: Oracle Database が存在するマシンのホスト名

   図 4-13 に示すように、`Employee.java` ファイルがブラウザに表示されます。

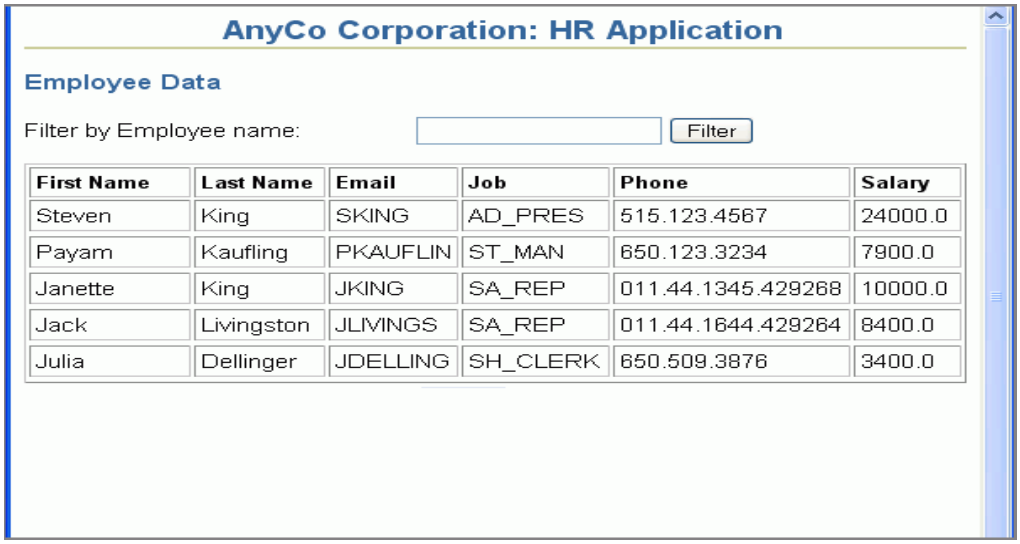
図 4-13 employee.jsp でのフィルタされていない従業員データ



First Name	Last Name	Email	Job	Phone	Salary
Steven	King	SKING	AD_PRES	515.123.4567	24000.0
Neena	Kochhar	NKOCHHAR	AD_VP	515.123.4568	17000.0
Lex	De Haan	LDEHAAN	AD_VP	515.123.4569	17000.0
Alexander	Hunold	AHUNOLD	IT_PROG	590.423.4567	9000.0
Bruce	Ernst	BERNST	IT_PROG	590.423.4568	6000.0
David	Austin	DAUSTIN	IT_PROG	590.423.4569	4800.0
Valli	Pataballa	VPATABAL	IT_PROG	590.423.4560	4800.0
Diana	Lorentz	DLORENTZ	IT_PROG	590.423.5567	4200.0
Nancy	Greenberg	NGREENBE	FI_MGR	515.124.4569	12000.0
Daniel	Faviet	DFAVIET	FI_ACCOUNT	515.124.4169	9000.0
John	Chen	JCHEN	FI_ACCOUNT	515.124.4269	8200.0

3. 従業員データをフィルタする文字列を入力します。たとえば、「**Filter by Employee Name**」フィールドに `ing` と入力し、「**Filter**」をクリックします。次に示すように、フィルタされたリストが表示されます。

図 4-14 employee.jsp でのフィルタされた従業員データ



5

データの更新

この章では、サンプル・アプリケーションの変更方法および Oracle データベース内のデータをユーザーが編集、更新および削除できるようにする機能の追加方法について説明します。この章は次の項で構成されています。

JavaBean の作成

簡単に言うと、Bean は、プロパティ、イベントおよびメソッドを持つ Java クラスです。Bean は、プロパティごとにアクセッサ（get および set メソッド）も持っています。特定の基本ルールに従っているオブジェクトは、すべて Bean といえます。Bean を作成するために拡張する必要がある特別なクラスはありません。

この章のサンプル・アプリケーションの作成手順では、単一の従業員レコードを保持するために JavaBean を使用します。既存レコードを編集したり、新規レコードを追加するときに、表の単一行の変更された値または新しい値を保持するコンテナとして JavaBean が使用され、これによりデータベースの更新に使用する行が準備されます。

Bean は従業員レコードのフィールドごとにプロパティを持ち、JDeveloper はこれらのプロパティごとにアクセッサ（get および set メソッド）を作成します。次の項では、サンプル・アプリケーションの JavaBean の作成方法について説明します。

JDeveloper での JavaBean の作成

Employee.java はサンプル・アプリケーションで使用する JavaBean で、単一の従業員レコードを保持してその内容を変更します。JavaBean を作成するには、次の手順を実行します。

1. **View** プロジェクトを右クリックして、ショートカット・メニューから**「新規」**を選択します。
2. 「新規ギャラリ」ダイアログ・ボックスの「フィルタ方法」フィールドで、**「すべてのテクノロジ」**を選択します。
3. 「一般」カテゴリを開いて、**「一般」**カテゴリの**「JavaBeans」**を選択します。**「項目」**リストから、**「Bean」**を選択します。**「OK」**をクリックします。
4. 「Bean 作成」ダイアログ・ボックスで、名前に Employee、パッケージに hr を入力し、**「拡張」**フィールドが java.lang.Object に設定されていることを確認します。**「OK」**をクリックして、Bean を作成します。
5. ファイルを保存します。これで、Employee.java ファイルに次のコードが含まれます。

```
package hr;

public class Employee {
  public Employee(){
  }
}
```

JavaBean のプロパティおよびメソッドの定義

JavaBean では、Employees 表の列ごとに 1 つのフィールドを作成し、フィールドごとにアクセッサ・メソッド（get および set メソッド）を作成する必要があります。

1. java.sql.Date（いずれかのフィールドのフィールド型）の import 文を追加します。

```
import java.sql.Date;
```

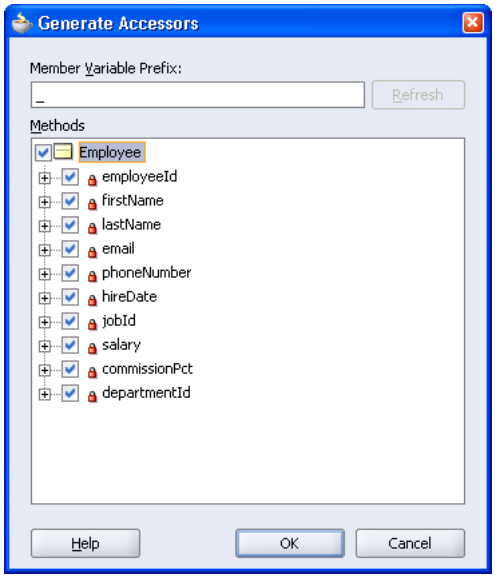
2. Employees 表の列ごとに、Employee クラスにフィールドを追加します。各フィールドは private で、フィールド型は次のとおりです。

```
  private Integer employeeId;
  private String firstName;
  private String lastName;
  private String email;
  private String phoneNumber;
  private Date hireDate;
```

```
    private String jobId;
    private Double salary;
    private Double commissionPct;
    private Integer departmentId;
```

3. ソース・エディタのページで右クリックし、ショートカット・メニューで**「アクセッサの生成」**を選択します。「アクセッサの生成」ダイアログ・ボックスで、トップレベルの **Employee** ノードを選択します。Employee ノードとそのすべてのフィールドに、チェック・マークが表示されます。**「OK」**をクリックします。図 5-1 に、すべてのフィールドが選択されている「アクセッサの生成」ダイアログ・ボックスを示します。

図 5-1「アクセッサの生成」ダイアログ・ボックス



4. ファイルを保存します。これで、Employee.java ファイルに次のコードが含まれます。

例 5-1 基本的な Java Bean とアクセッサ・メソッドのスケルトン・コード

```
package hr;
import java.sql.Date;

public class Employee {
    public Employee() {
    }
    private Integer employeeId;
    private String firstName;
    private String lastName;
    private String email;
    private String phoneNumber;
    private Date hireDate;
    private String jobId;
    private Double salary;
    private Double commissionPct;
    private Integer departmentId;

    public void setEmployeeId(Integer employeeId) {
        this.employeeId = employeeId;
    }

    public Integer getEmployeeId() {
        return employeeId;
    }
```

```
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getFirstName() {
        return firstName;
    }
...
...
...
...
// This list has been shortened and is not comprehensive. The actual code contains //
accessor methods for all the fields declared in the bean.

    public void setDepartmentId(Integer departmentId) {
        this.departmentId = departmentId;
    }

    public Integer getDepartmentId() {
        return departmentId;
    }
}
```

Java クラスからのデータの更新

Java アプリケーションからデータベースの表に含まれる行を更新するには、次のタスクを実行する必要があります。

1. 特定の従業員行を検索するメソッドを作成します。このメソッドを使用して、特定の従業員の値を編集ページに表示します。
2. Bean から更新済の従業員データを取り出してデータベースを更新するメソッドを作成します。
3. アプリケーションのメイン・ページにある従業員データの各行に、その従業員データの編集を可能にするリンクを含めます。このリンクを使用すると、`edit.jsp` ファイルが開き、その従業員のデータが編集可能な状態で表示されます。
4. `edit.jsp` という名前の JSP ページを作成します。このページには、単一の従業員の全データを表示するフォームと表が含まれ、ユーザーはこのページを使用して値を変更できます。
5. `edit.jsp` ページのフォームを処理し、更新された値を `Employee.java` Bean に書き込んで、`updateEmployee` メソッドをコールする JSP ページを作成します。

これらの方法については、次の項を参照してください。

- 従業員レコードを識別するメソッドの作成
- 従業員データを更新するメソッドの作成
- 更新ページにナビゲートするリンクの追加
- 従業員データを編集する JSP ページの作成
- 更新アクションを処理する JSP ページの作成

従業員レコードを識別するメソッドの作成

次の手順で作成するメソッドは、特定の従業員レコードの検索に使用します。このメソッドは、特定の従業員レコードを編集または削除する場合や、Employee.java ページでその従業員のリンクを選択した場合に使用されます。

1. DataHandler クラスが Java ソース・エディタでまだ開いていない場合は、アプリケーション・ナビゲータでこのクラスをダブルクリックして開きます。
2. DataHandler クラスで、更新する従業員レコードを識別するための新規メソッドを宣言します。

```
public Employee findEmployeeById(int id) throws SQLException {

}
```

3. このメソッドのボディで、selectedEmp という名前の、Employee Bean の新規インスタンスを作成します。

```
Employee selectedEmp = new Employee();
```

4. データベースに接続します。

```
getDBConnection();
```

5. Statement オブジェクトを作成し、ResultSet 型を定義して問合せを作成します。デバッグに役立てるために、トレース・メッセージを追加します。

```
stmt =
  conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                       ResultSet.CONCUR_READ_ONLY);
query = "SELECT * FROM Employees WHERE employee_id = " + id;
System.out.println("\nExecuting: " + query);
```

6. 問合せを実行し、ResultSet オブジェクトを使用して結果を格納します。

```
rset = stmt.executeQuery(query);
```

7. rset で返された結果セットを使用し、従業員 Bean の set メソッドを使用して Bean のフィールドに移入します。

```
while (rset.next()) {
  selectedEmp.setEmployeeId(new Integer(rset.getInt("employee_id")));
  selectedEmp.setFirstName(rset.getString("first_name"));
  selectedEmp.setLastName(rset.getString("last_name"));
  selectedEmp.setEmail(rset.getString("email"));
  selectedEmp.setPhoneNumber(rset.getString("phone_number"));
  selectedEmp.setHireDate(rset.getDate("hire_date"));
  selectedEmp.setSalary(new Double(rset.getDouble("salary")));
  selectedEmp.setJobId(rset.getString("job_id"));
}
```

8. 移入されたオブジェクトを返します。

```
return selectedEmp;
```

従業員データを更新するメソッドの作成

次の手順では、データベース内の従業員データを更新するメソッドの作成方法について説明します。

1. DataHandler クラスを開きます。
2. 次のように、updateEmployee メソッドを宣言します。

```
public String updateEmployee(int employee_id, String first_name,
                             String last_name, String email,
                             String phone_number, String salary,
                             String job_id) throws SQLException {

}
```

3. このメソッドのボディで、選択した従業員の詳細を格納する、Employee Bean のインスタンスを作成します。

```
Employee oldEmployee = findEmployeeById(employee_id);
```

4. データベースに接続します。

```
getDBConnection();
```

5. Statement オブジェクトを作成し、前述と同様に ResultSet 型を指定します。

```
stmt =
  conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                       ResultSet.CONCUR_READ_ONLY);
```

6. 作成する必要がある SQL UPDATE 文の詳細を集積する StringBuffer を作成します。

```
StringBuffer columns = new StringBuffer( 255 );
```

7. 従業員レコードのフィールドごとに、ユーザーが値を変更したかどうかをチェックします。値が変更されている場合には、関連するコードを StringBuffer に追加します。追加する 2 つ目以降の項目については、カンマを使用してそれぞれの項目を区切ります。次のコードは、first_name 変数が変更されているかどうかをチェックし、変更されている場合は、データベース更新に使用される StringBuffer の SQL に詳細を追加します。

```
if ( first_name != null &&
   !first_name.equals(oldEmployee.getFirstName() ) )
{
  columns.append( "first_name = '" + first_name + "'" );
}
```

last_name については、新しい名字を追加する前に、StringBuffer 内にすでになんらかの変更があるかどうかをチェックし、変更がある場合は、カンマを追加して新しい変更と以前の変更を分けます。次のコードを使用します。

```
if ( last_name != null &&
    !last_name.equals(oldEmployee.getLastName() ) ) {
  if ( columns.length() > 0 ) {
    columns.append( ", " );
  }
  columns.append( "last_name = '" + last_name + "'" );
}
```

同じコード・ロジックを使用して、email および phone_number の変更をチェックします。

注意： この手順にはコードの重要な部分のみが含まれています。例 5-2 には、このメソッドのコードをすべて記載しています。

---

salary フィールドについては、String 値を取得して、次のように StringBuffer に追加します。

```
if ( salary != null &&
    !salary.equals( oldEmployee.getSalary().toString() ) ) {
  if ( columns.length() > 0 ) {
    columns.append( ", " );
  }
  columns.append( "salary = '" + salary + "'" );
```

8. すべての変更セットが収集されたら、実際に変更があるかどうかをチェックします。つまり、StringBuffer に何か含まれているかどうかをチェックします。含まれている場合は、StringBuffer の情報を使用して SQL UPDATE 文を構築し、この SQL 文を実行します。StringBuffer に変更が含まれていない場合には、そのことを知らせるメッセージを出力します。

```
if ( columns.length() > 0 )
{
  sqlString = "update Employees SET " + columns.toString() +
          " WHERE employee_id = " + employee_id;
  System.out.println("\nExecuting: " + sqlString);
  stmt.execute(sqlString);
}
else
{
  System.out.println( "Nothing to do to update Employee Id: " +
                      employee_id);
}
```

9. 「success」というメッセージを返します。

```
return "success";
```

10. 作業内容を保存してファイルを作成し、構文エラーがないかどうかをチェックします。

例 5-2 に、このメソッドのすべてのコードを示します。

例 5-2 データベース・レコードの更新のメソッド

```
public String updateEmployee(int employee_id, String first_name,
                             String last_name, String email,
                             String phone_number, String salary,
                             String job_id) throws SQLException {

  Employee oldEmployee = findEmployeeById(employee_id);
  getDBConnection();
  stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                              ResultSet.CONCUR_READ_ONLY);

  StringBuffer columns = new StringBuffer( 255 );
  if ( first_name != null &&
     !first_name.equals( oldEmployee.getFirstName() ) )
  {
    columns.append( "first_name = '" + first_name + "'" );
  }
  if ( last_name != null &&
      !last_name.equals( oldEmployee.getLastName() ) ) {
    if ( columns.length() > 0 ) {
      columns.append( ", " );
    }
    columns.append( "last_name = '" + last_name + "'" );
  }
  if ( email != null &&
      !email.equals( oldEmployee.getEmail() ) ) {
```

```
    if ( columns.length() > 0 ) {
      columns.append( ", " );
    }
    columns.append( "email = '" + email + "'" );
  }
  if ( phone_number != null &&
      !phone_number.equals( oldEmployee.getPhoneNumber() ) ) {
    if ( columns.length() > 0 ) {
      columns.append( ", " );
    }
    columns.append( "phone_number = '" + phone_number + "'" );
  }
  if ( salary != null &&
      !salary.equals( oldEmployee.getSalary().toString() ) ) {
    if ( columns.length() > 0 ) {
      columns.append( ", " );
    }
    columns.append( "salary = '" + salary + "'" );
  }
  if ( job_id != null &&
      !job_id.equals( oldEmployee.getJobId() ) ) {
    if ( columns.length() > 0 ) {
      columns.append( ", " );
    }
    columns.append( "job_id = '" + job_id + "'" );
  }

  if ( columns.length() > 0 )
  {
  sqlString =
    "UPDATE Employees SET " + columns.toString() +
      " WHERE employee_id = " + employee_id;
    System.out.println("\nExecuting: " + sqlString);
    stmt.execute(sqlString);
  }
  else
  {
    System.out.println( "Nothing to do to update Employee Id: " +
                        employee_id);
  }
  return "success";
}
```

更新ページにナビゲートするリンクの追加

次の手順では、employees.jsp ページの従業員表の各行にリンクを追加します。ユーザーはリンクをクリックしてその行を編集します。

1. ビジュアル・エディタで employees.jsp を開きます。
2. 従業員の詳細を表示する表に、新規に列を追加します。列を追加するには、表の最後の列にカーソルを置いて右クリックし、ショートカットメニューで**「表」**を選択してから、**「行または列の挿入」**を選択します。「行または列の挿入」ダイアログ・ボックスで**「列」**および**「選択した列の後」**を選択して、**「OK」**をクリックします。
3. この追加された列には、「Edit」と表示されるリンクが行ごとに含まれます。これらのリンクは、選択した従業員レコードを編集できる個々のページにナビゲートします。これには、Employees 表内のスクリプトレットをダブルクリックして「スクリプトレットのプロパティ」ダイアログ・ボックスを表示します。

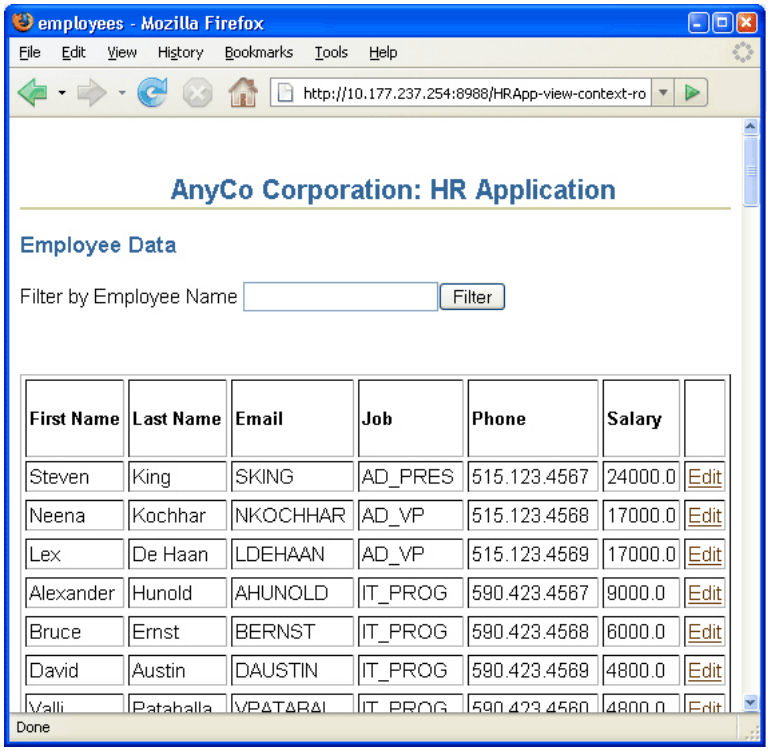
4. edit.jsp ページへのリンクが含まれるように、スクリプトレットを変更します。変更されたスクリプトレットには次のコードが含まれている必要があります。

```
while (rset.next ())
{
out.println("<tr>");
  out.println("<td>" +
  rset.getString("first_name") + "</td><td> " +
  rset.getString("last_name") + "</td><td> " +
  rset.getString("email") + "</td><td> " +
  rset.getString("job_id") + "</td><td>" +
  rset.getString("phone_number") + "</td><td>" +
  rset.getDouble("salary") +
  "</td><td> <a href=\"edit.jsp?empid=" + rset.getInt(1) +
  "\">Edit</a></td>");
out.println("<tr>");
}
```

従業員の編集リンクがクリックされると、このコードは、従業員レコードの更新を処理する edit.jsp ページに、従業員 ID を渡します。edit.jsp ページはこの ID を使用して、その従業員のレコードをデータベース内で検索します。

5. employees.jsp を保存します。図 5-2 に、実行されブラウザに表示された employees.jsp を示します。図に示されたリンクをクリックすると、従業員データを編集できます。

図 5-2 employees.jsp の従業員の編集へのリンク



従業員データを編集する JSP ページの作成

この項では、従業員レコードの更新に使用する edit.jsp ファイルを作成します。

1. 新しい JSP ページを作成して、edit.jsp という名前を付けます。その他はすべてデフォルトを使用します。
2. このページに、以前に設定したものと同じ見出し **AnyCo Corporation: HR Application** を付け、**Heading 2** スタイルを割り当て、ページの中央に配置します。
3. 次の行に、**Edit Employee Record** と入力し、**Heading 3** スタイルを適用します。この見出しをページの左側に配置します。
4. **JDeveloper** スタイルシートをページに追加します。
5. jsp:usebean タグを追加します。empsbean を **ID** として入力し、hr.DataHandler を**クラス**として入力します。**「有効範囲」**を「セッション」に設定し、**「OK」**をクリックします。
6. カーソルを useBean タグの後に置いて、jsp:usebean タグをもう 1 つ追加します。今度は、employee を**「ID」**として入力し、**hr.Employee** を「クラス」として参照して選択し、「有効範囲」を「page」のままにします。**「OK」**をクリックします。
7. **スクリプトレット**をページに追加します。スクリプトレット・コードは、従業員 ID を findEmployeeById メソッドに渡して、Employee Bean 内のデータを取り出します。次のコードを「スクリプトレットの挿入」ダイアログ・ボックスに入力します。

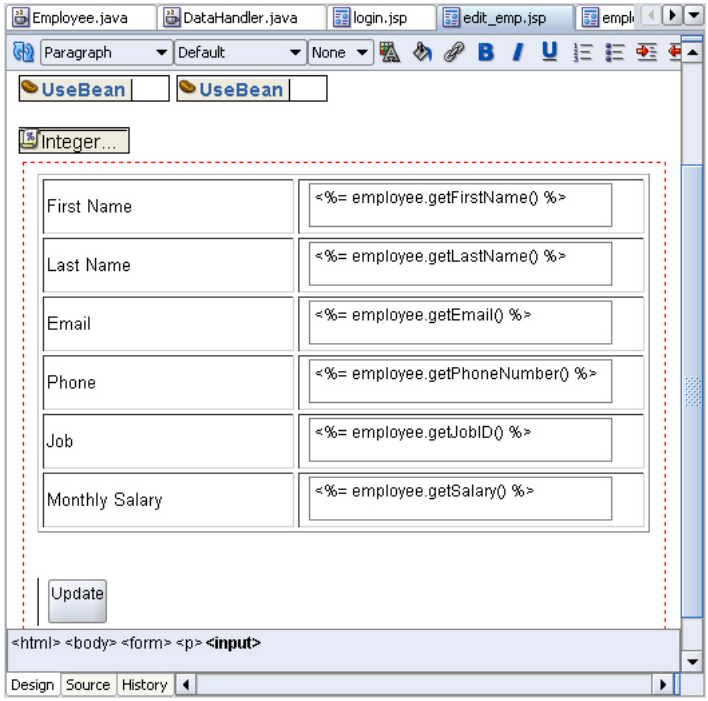
   ```
   Integer employee_id = new Integer(request.getParameter("empid"));
   employee = empsbean.findEmployeeById(employee_id.intValue());
   ```

8. **フォーム**をページに追加します。「フォームの挿入」ダイアログで、**「アクション」**フィールドに update_action.jsp と入力します。まだ作成していないため、このページはドロップダウン・リストからは選択できません。
9. **表**をページに追加します。フォーム内に配置します。6 行 2 列のレイアウトを指定し、その他のレイアウトはデフォルトを使用します。
10. 各行の最初の列に、次の見出しをそれぞれ入力します。First Name、Last Name、Email、Phone, Job、Monthly Salary
11. コンポーネント・パレットの「HTML Forms」ページから、**「Hidden Field」**コンポーネントをドラッグします。2 列目の、First Name 見出しの横にドロップします。「Hidden Field の挿入」ダイアログで、employee_id を**「名前」**プロパティに入力し、<%= employee.getEmployeeId() %> を**「値」**プロパティに入力します。
12. この列の、First Name 見出しの横に、**「テキスト・フィールド」**コンポーネントをドラッグします。「テキスト・フィールドの挿入」ダイアログで、first_name を**「名前」**フィールドに入力し、<%= employee.getFirstName() %> を**「値」**フィールドに入力します。**「OK」**をクリックします。
13. この列の、Last Name 見出しの横に、2 つ目の**「テキスト・フィールド」**コンポーネントをドラッグします。「テキスト・フィールドの挿入」ダイアログで、last_name を**「名前」**フィールドに入力し、<%= employee.getLastName() %> を「値」フィールドに入力します。**「OK」**をクリックします。
14. 同様の方法で、残りの各列見出しの横にテキスト・フィールドを追加します。email、phone_number、job_id および salary をフィールド名として使用し、各フィールドに対応する getter メソッドを指定します。これらは次の表に示されています。
15. **「発行」**ボタンを、フォームの表の下に追加します。Update を**「値」**として入力します。

16. アプリケーションを保存します。

    結果として作成される edit.jsp ページは、図 5-3 のようなページになります。

図 5-3 従業員の詳細を編集する JSP ページの作成



更新アクションを処理する JSP ページの作成

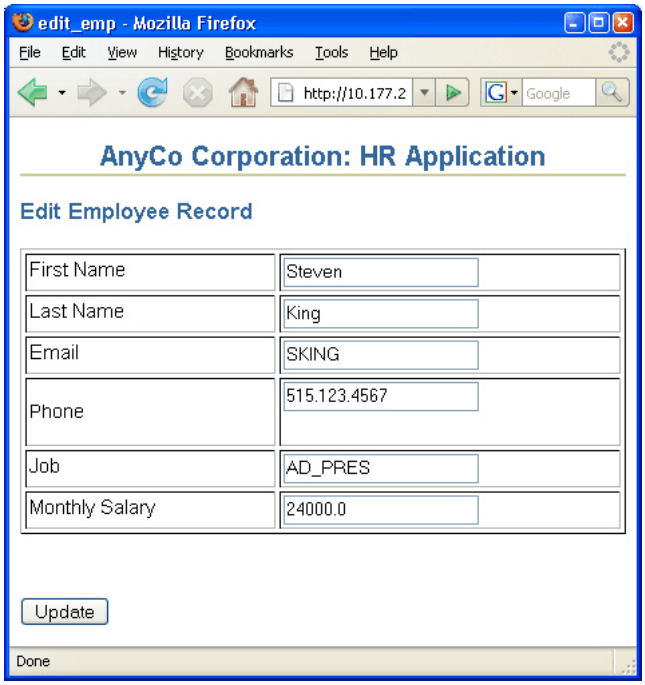
この項では、update_action.jsp ファイルの作成方法について説明します。このページは、従業員レコードの更新に使用する edit.jsp ページのフォームを処理します。このページには表示要素はなく、edit.jsp フォームを処理するためにのみ使用され、employees.jsp ファイルに制御を返します。

1. 新しい JSP ページを作成して、update_action.jsp という名前を付けます。その他はすべて、JSP 作成ウィザードのページのデフォルトを使用します。
2. コンポーネント・パレットの「JSP」ページから、**「Page Directive」**コンポーネントをページにドラッグします。「Page Directive の挿入」ダイアログ・ボックスで、**java.sql.ResultSet** を参照してインポートします。**「OK」**をクリックします。
3. **jsp:usebean** タグを追加します。empsbean を **ID** として入力し、hr.DataHandler を**クラス**として入力します。**「有効範囲」**を「セッション」に設定し、**「OK」**をクリックします。
4. **スクリプトレット**をページに追加します。次のコードを「スクリプトレットの挿入」ダイアログ・ボックスに入力します。

```
Integer employee_id = new Integer(request.getParameter("employee_id"));
String first_name = request.getParameter("first_name");
String last_name = request.getParameter("last_name");
String email = request.getParameter("email");
String phone_number = request.getParameter("phone_number");
String salary = request.getParameter("salary");
String job_id = request.getParameter("job_id");
empsbean.updateEmployee(employee_id.intValue(), first_name, last_name, email,
phone_number, salary, job_id );
```

5. **jsp:forward** タグをページにドラッグします。「Forward の挿入」ダイアログ・ボックスで、**「ページ」**プロパティに `employees.jsp` と入力します。
6. 作業内容を保存します。
7. プロジェクトを実行して、従業員レコードを編集できるかどうかをテストします。`employees.jsp` ページで、いずれかの従業員の **Edit** をクリックすると、図 5-4 に示されているページに移動します。従業員の詳細を変更し、`employees.jsp` ページで変更が反映されているかどうかをチェックします。

図 5-4 従業員データの編集



従業員レコードの挿入

Employees 表に新しい従業員レコードを挿入する手順は、従業員レコードの更新処理と似ています。

1. `Employees` 表に新しい従業員行を挿入するメソッドを作成します。
2. アプリケーションのメイン・ページにリンクを追加します。ユーザーは、このリンクをクリックすることで、新しい従業員を挿入できます。リンクをクリックすると、`insert.jsp` が開き、新しい行の詳細を入力できる空のフォームが表示されます。
3. `insert.jsp` ページのフォームを処理する JSP ページを作成します。
4. 新しい従業員の値を入力するためのフォームを制御する JSP ページを作成します。

ここでは、新しい従業員データを挿入する Java アプリケーション・コードの作成方法について次の各項で説明します。

- データを挿入するメソッドの作成
- 挿入ページにナビゲートするリンクの追加
- 挿入アクションを処理する JSP ページの作成
- 新規データを入力する JSP ページの作成

データを挿入するメソッドの作成

次の手順では、新しい従業員レコードを挿入するメソッドを作成します。

1. Java ソース・エディタで DataHandler.java を開きます。
2. 新しい従業員レコードを追加するメソッドを宣言します。

```
public String addEmployee(String first_name,
  String last_name, String email,
  String phone_number, String job_id, int salary) throws SQLException {

}
```

3. データベースに接続するための行を追加します。

```
getDBConnection();
```

4. Statement オブジェクトを作成し、ResultSet 型を以前と同様に定義して SQL 文を作成します。

```
stmt =
  conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                       ResultSet.CONCUR_READ_ONLY);
sqlString =
  "INSERT INTO Employees VALUES (EMPLOYEES_SEQ.nextval, '" +
  first_name + "','" +
  last_name + "','" +
  email + "','" +
  phone_number + "'," +
  "SYSDATE, '" +
  job_id + "', " +
  salary + ",.30,100,80)";
```

> **注意：** 最後の 3 つの列（Commission、ManagerId および DepartmentId）には、サンプル・アプリケーション用のハードコードされた値が格納されます。

5. トレース・メッセージを追加して、SQL 文を実行します。
6. 挿入が成功した場合は、「success」というメッセージを返します。
7. 構文エラーをチェックするためのファイルを作成します。

例 5-3 に、addEmployee() メソッドのコードを示します。

例 5-3 新規従業員レコードの追加のメソッド

```
public String addEmployee(String first_name,
  String last_name, String email,
  String phone_number, String job_id, int salary) throws SQLException {
  getDBConnection();
  stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                              ResultSet.CONCUR_READ_ONLY);
  sqlString =
    "INSERT INTO Employees VALUES (EMPLOYEES_SEQ.nextval, '" +
     first_name + "','" +
    last_name + "','" +
    email + "','" +
    phone_number + "'," +
    "SYSDATE, '" +
    job_id + "', " +
```

```
    salary + ",.30,100,80)";

  System.out.println("\nInserting: " + sqlString);
  stmt.execute(sqlString);
  return "success";
}
```

挿入ページにナビゲートするリンクの追加

次の手順では、従業員表のヘッダー行にリンクを追加します。ユーザーはリンクをクリックすることで、新しい従業員を追加できます。

1. ビジュアル・エディタで `employees.jsp` を開きます。
2. コンポーネント・パレットの「HTML Common」ページから、**「Hyper Link」**コンポーネントを、ヘッダー行の最後にある空の列ヘッダーのセルにドラッグします。「ハイパー・リンクの挿入」ダイアログ・ボックスで、**「ハイパーリンク」**フィールドに `insert.jsp` を入力し、**「テキスト」**フィールドに `Insert Employee` を入力します。まだ作成していないため、`insert.jsp` を参照して検索することはできません。**「OK」**をクリックします。
3. `employees.jsp` を保存します。

新規データを入力する JSP ページの作成

次の手順では、`insert.jsp` ページを作成します。このページを使用することで、ユーザーは新しい従業員レコードの詳細を入力できます。

1. 新しい JSP ページを作成して、`insert.jsp` という名前を付けます。
2. このページに、以前に設定したものと同じ見出し **AnyCo Corporation: HR Application** を付け、**Heading 2** としてフォーマットして、中央に配置します。
3. 次の行に、**Insert Employee Record** と入力し、**Heading 3** フォーマットを適用します。この見出しをページの左側に配置します。
4. **JDeveloper** スタイルシートをページに追加します。
5. **フォーム**を追加します。「フォームの挿入」ダイアログ・ボックスで、**「アクション」**プロパティに `insert_action.jsp` と入力して、**「OK」**をクリックします。
6. **表**を**フォーム**内に追加します。6 行 2 列を指定し、その他のレイアウトはデフォルトを使用します。
7. 各行の最初の列に、次の見出しをそれぞれ入力します。**First Name**、**Last Name**、**Email**、**Phone**, **Job**、**Monthly Salary**
8. **First Name** ヘッダーの右の列に、**「テキスト・フィールド」**をドラッグ・アンド・ドロップします。「フィールドの挿入」ダイアログ・ボックスで、**「名前」**プロパティに `first_name` と入力します。

9. **Last Name**、**Email**、**Phone** および **Monthly Salary** ヘッダーのそれぞれの横に、**「テキスト・フィールド」**をドラッグします。これらのテキスト・フィールドごとに、「フィールドの挿入」ダイアログ・ボックスで**「名前」**プロパティに値を指定します。次の表に、値を示します。

テキスト・フィールドの対象	「名前」プロパティに設定する値
Last Name	`last_name`
Email	`email`
Phone	`phone_number`
Monthly Salary	`salary`

Job 行は手順が異なります。

10. コンポーネント・パレットの「HTML Forms」ページから、**コンボ・ボックス**コンポーネントを、**Job** 見出しの横の列にドラッグします。

11. 「選択の挿入」ダイアログ・ボックスで、名前に `job_id` を入力し、サイズに `1` を入力します。**「追加」**をクリックします。**「値」**フィールドをクリックして `SA_REP` と入力し、**「キャプション」**フィールドをクリックして `Sales Representative` と入力します。**「追加」**をクリックして次のジョブ・タイトルをそれぞれ追加し、**「OK」**をクリックします。

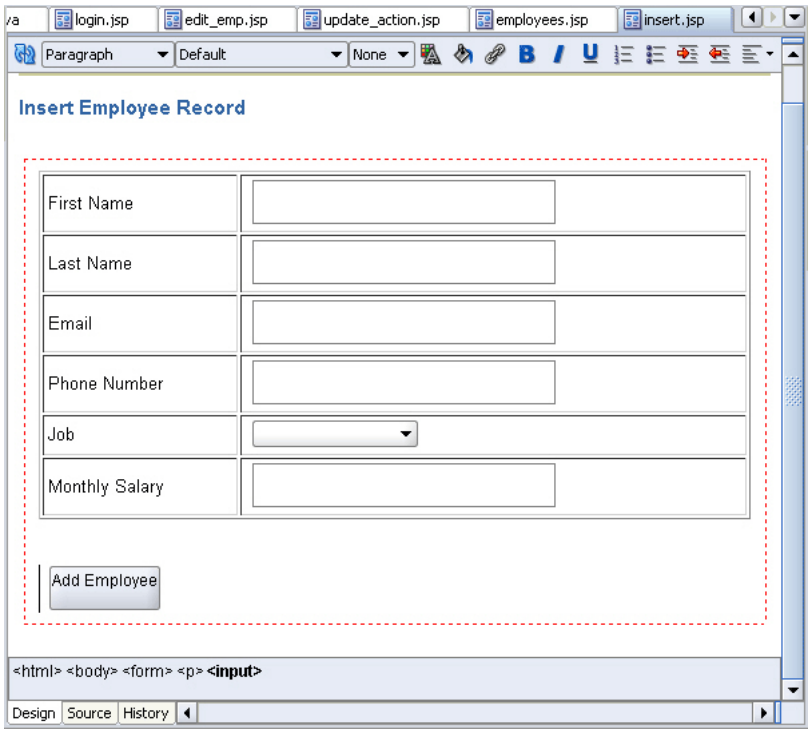
値	キャプション
`HR_REP`	HR Representative
`PR_REP`	PR Representative
`MK_MAN`	Marketing Manager
`SA_MAN`	Sales Manager
`FI_MAN`	Finance Manager
`IT_PROG`	Software Developer
`AD_VIP`	Vice President

12. **「発行」**ボタンを表の下のフォームにドラッグします。「送信ボタンの挿入」ダイアログ・ボックスで、**「値」**プロパティに `Add Employee` と入力します。

13. 作業内容を保存します。

図 5-5 に、ビジュアル・エディタでの insert.jsp ページを示します。

図 5-5 従業員データを挿入するフォーム



挿入アクションを処理する JSP ページの作成

次の手順では、insert_action.jsp ページを作成します。このページは、新しい従業員レコードを入力する insert.jsp ページからのフォーム入力を処理します。このページには表示要素はなく、insert.jsp フォームを処理するためにのみ使用され、employees.jsp ファイルに制御を返します。

1. 前述と同様に、JSP ページを作成します。このページに insert_action.jsp という名前を付けます。
2. **jsp:usebean** タグを追加します。以前と同様に、**ID** に empsbean を入力し、**クラス**に hr.DataHandler を入力します。**「有効範囲」**を「セッション」に設定し、**「OK」**をクリックします。
3. カーソルを useBean タグの後に置き、**スクリプトレット**をページに追加します。次のコードを「スクリプトレットの挿入」ダイアログ・ボックスに入力します。

```
String first_name = request.getParameter("first_name");
String last_name = request.getParameter("last_name");
String email = request.getParameter("email");
String phone_number = request.getParameter("phone_number");
String job_id = request.getParameter("job_id");
Integer salary = new Integer(request.getParameter("salary"));

empsbean.addEmployee(first_name, last_name, email, phone_number, job_id,
salary.intValue());
```

4. **jsp:forward** タグをページにドラッグします。「Forward の挿入」ダイアログ・ボックスで、employees.jsp と入力します。
5. 作業内容を保存します。

6. **View** プロジェクトを実行して、新しい従業員レコードを挿入できるかどうかをテストします。

従業員を挿入するには、図 5-6 に示した `employees.jsp` ページで、`Insert Employee` をクリックします。

図 5-6 新規従業員データの挿入

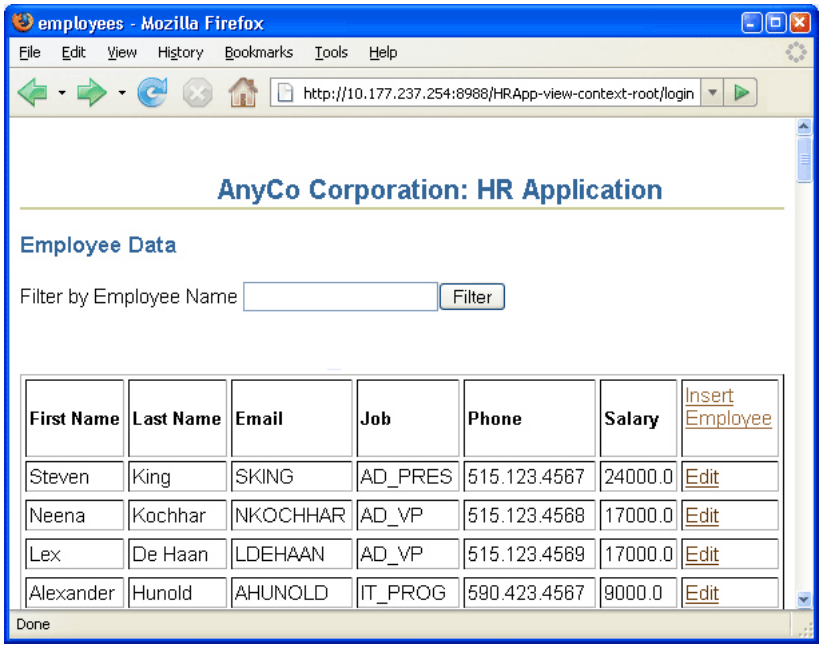
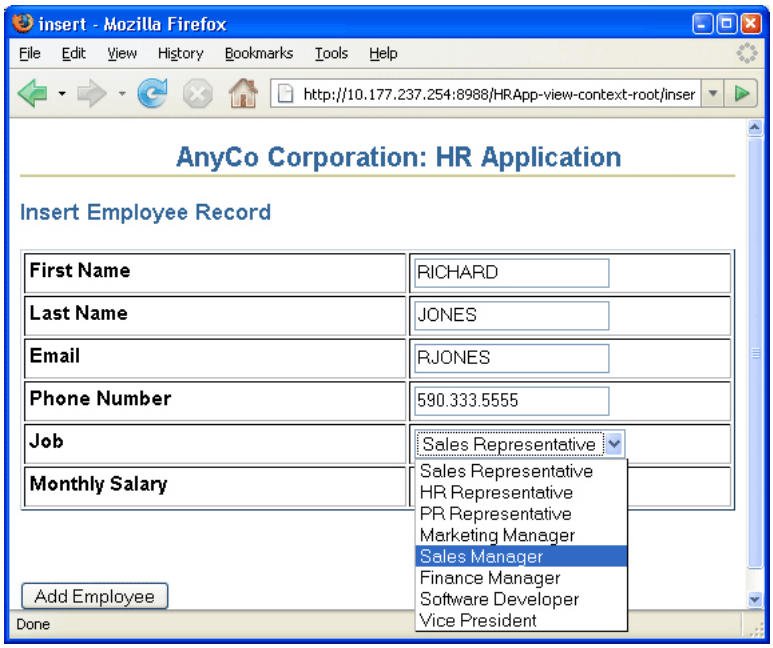


図 5-7 に、入力済のデータと、ジョブの選択に使用されるジョブ・リストが表示されている、従業員データを新規に挿入するページを示します。

図 5-7 従業員データの挿入



従業員レコードの削除

レコードを削除する手順は、レコードを編集および挿入する手順と似ています。

1. 「従業員レコードを識別するメソッドの作成」で作成したメソッドを使用して、特定の従業員行を識別します。このメソッドは、削除する行を識別するために使用されます。
2. 従業員レコードをデータベースから削除するメソッドを作成します。
3. アプリケーションのメイン・ページの各行にリンクを追加します。ユーザーは、このリンクをクリックすることで、その行の従業員を削除できます。リンクをクリックすると、ユーザーは削除する従業員レコードの ID による `delete_action.jsp` ページにナビゲートされます。
4. データベースから従業員を削除するには、手順 2 で作成した削除メソッドをコールする JSP ページを作成します。

この項では、従業員データの削除に関連する次のタスクについて説明します。

- データを削除するメソッドの作成
- 従業員を削除するリンクの追加
- 削除アクションを処理する JSP ページの作成

データを削除するメソッドの作成

次の手順で作成するメソッドは、ID を使用した従業員レコードの削除に使用します。

1. Java ソース・エディタで `DataHandler.java` を開きます。
2. 削除する従業員レコードを識別するための新規メソッドを宣言します。

```
public String deleteEmployeeById(int id) throws SQLException {

}
```

3. 前述と同様に、データベースに接続します。

```
getDBConnection();
```

4. `Statement` オブジェクトを作成し、`ResultSet` 型を以前と同様に定義して SQL 文を作成します。デバッグに役立てるために、トレース・メッセージを追加します。

```
stmt =
  conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                       ResultSet.CONCUR_READ_ONLY);
sqlString = "DELETE FROM Employees WHERE employee_id = " + id;
System.out.println("\nExecuting: " + sqlString);
```

5. SQL 文を実行します。

```
stmt.execute(sqlString);
```

6. SQL 文の実行が正常に完了した場合は、`Success` というメッセージを返します。

```
return "success";
```

例 5-4 に、`deleteEmployeeById()` メソッドのコードを示します。

例 5-4 従業員レコードを削除するメソッド

```
public String deleteEmployeeById(int id) throws SQLException {
getDBConnection();
stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                            ResultSet.CONCUR_READ_ONLY);
    sqlString = "DELETE FROM Employees WHERE employee_id = " + id;
```

```
    System.out.println("\nExecuting: " + sqlString);
    stmt.execute(sqlString);
    return "success";
}
```

従業員を削除するリンクの追加

次の手順では、employees.jsp ページの従業員表の各行にリンクを追加します。リンクをクリックすると、その行の従業員データがすべて削除されます。

1. ビジュアル・エディタで employees.jsp を開きます。
2. Edit リンクを配置するために作成した列に、行を削除するための別のリンクを追加します。このためには、Employees 表内のスクリプトレットをダブルクリックしてスクリプトレットのプロパティ」ダイアログ・ボックスを表示します。
3. delete_action.jsp ページへのリンクが含まれるように、スクリプトレットを変更します。変更されたスクリプトレットには次のコードが含まれます。

```
while (rset.next ())
{
out.println("<tr>");
  out.println("<td>" +
  rset.getString("first_name") + "</td><td> " +
  rset.getString("last_name") + "</td><td> " +
  rset.getString("email") + "</td><td> " +
  rset.getString("job_id") + "</td><td>" +
  rset.getString("phone_number") + "</td><td>" +
  rset.getDouble("salary") +
  "</td><td> <a href=\"edit.jsp?empid=" + rset.getInt(1) +
  "\">Edit</a>   <a href=\"delete_action.jsp?empid=" +
  rset.getInt(1) + "\">Delete</a></td>");
out.println("<tr>");
}
```

4. employees.jsp を保存します。

削除アクションを処理する JSP ページの作成

次の手順では、delete_action.jsp ページを作成します。このページは削除操作のみを処理します。このページには、表示要素はありません。

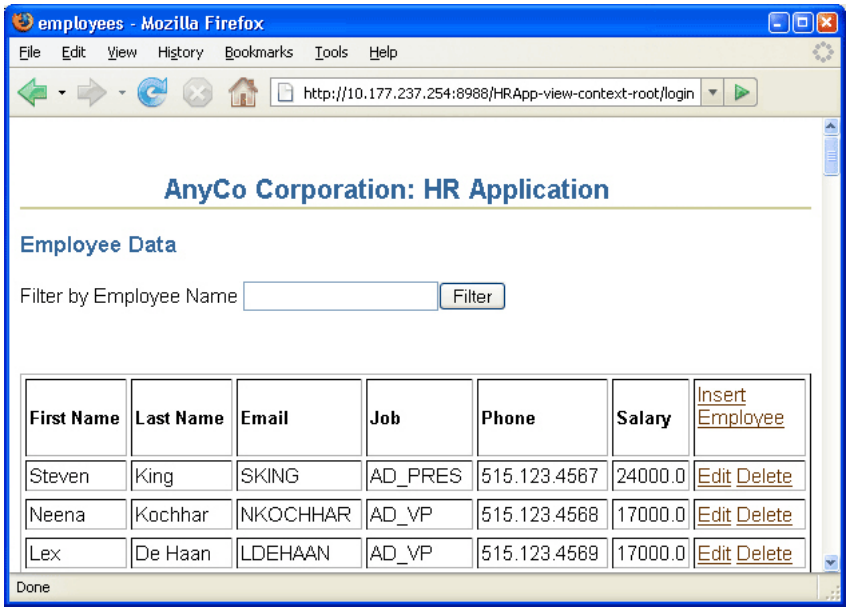
1. 新しい JSP ページを作成して、delete_action.jsp という名前を付けます。
2. **jsp:usebean** タグを追加します。以前と同様に、**ID** に empsbean を入力し、**クラス**に hr.DataHandler を入力します。**「有効範囲」**を「セッション」に設定し、**「OK」**をクリックします。
3. **スクリプトレット**をページに追加します。次のコードを「スクリプトレットの挿入」ダイアログ・ボックスに入力します。

```
Integer employee_id =
 new Integer(request.getParameter("empid"));
empsbean.deleteEmployeeById(employee_id.intValue());
```

4. コンポーネント・パレットから Forward をドラッグして、**jsp:forward** タグをページに追加します。「Forward の挿入」ダイアログ・ボックスで、**employees.jsp** と入力します。
5. 作業内容を保存します。

6. プロジェクトを実行して従業員を削除してみます。図 5-8 に、employees.jsp から従業員レコードを削除するリンクを示します。

図 5-8 employees.jsp から従業員を削除するためのリンク



いずれかの従業員レコードの **Delete** をクリックすると、その従業員レコードが削除されます。

例外処理

SQLException オブジェクトのインスタンスを使用すると、データベース・アクセス・エラーやその他のエラーに関する情報を取得できます。それぞれの SQLException インスタンスには、エラーを記述する文字列を含む様々な種類の情報が用意されています。この文字列は Java Exception メッセージとして使用され、getMessage メソッドで取得できます。

サンプル・アプリケーションでは、try ブロックと catch ブロックを使用します。これらは、例外を処理する Java メカニズムです。Java では、メソッドが例外をスローした場合に、それを処理するメカニズムが必要です。一般的には、catch ブロックで例外を捕捉して、例外発生時の一連のアクションを指定します。これは、単なるメッセージの表示のみの場合もあります。

データベース・アクセス・エラーが発生すると、各 JDBC メソッドは SQLException をスローします。そのため、このようなメソッドを実行するアプリケーションのメソッドにはすべて例外処理が必要です。

サンプル・アプリケーションのメソッドにはすべて例外処理のコードが含まれています。たとえば、データベースへの接続を取得する getDBConnection は、SQLException をスローします。これは、次のような getAllEmployees メソッドと同様です。

```
public ResultSet getAllEmployees() throws SQLException {
}
```

SQLExceptions を捕捉して処理するコードのサンプルについては、DataHandler.java クラスの authenticateUser メソッドのコードを参照してください。この例では、try ブロックに、ユーザーを認証するために実行される作業のコードが含まれており、catch ブロックに認証に失敗した場合の処理が含まれています。

次の項では、SQLExceptions を捕捉して処理するコードをサンプル・アプリケーションに追加する方法について説明します。

Java メソッドへの例外処理の追加

サンプル・アプリケーションのメソッドで SQL 例外を処理するには、次の手順を実行します。

1. メソッドが SQLException をスローしていることを確認します。次にメソッドの例を示します。

```
public ResultSet getAllEmployees() throws SQLException
```

2. try ブロックと catch ブロックを使用して、SQLExceptions を捕捉します。たとえば、getAllEmployees メソッドでは、既存コードを try ブロックにまとめ、catch ブロックを次のように追加しています。

```
public ResultSet getAllEmployees() throws SQLException {
  try {
    getDBConnection();
    stmt =
      conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                           ResultSet.CONCUR_READ_ONLY);
    sqlString = "SELECT * FROM Employees order by employee_id";
    System.out.println("\nExecuting: " + sqlString);
    rset = stmt.executeQuery(sqlString);
  }
  catch (SQLException e) {
    e.printStackTrace();
  }
  return rset;
}
```

3. また別の例として、try ブロックと catch ブロックを使用するように書き直した deleteEmployee メソッドでは、メソッドが成功した場合にのみ「success」を返します。つまり、return 文を try ブロックに記述しています。コードは次のとおりです。

```
  public String deleteEmployeeById(int id) throws SQLException {

    try {
      getDBConnection();
      stmt =
        conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                             ResultSet.CONCUR_READ_ONLY);
      sqlString = "delete FROM Employees where employee_id = " + id;
      System.out.println("\nExecuting: " + sqlString);

      stmt.execute(sqlString);
      return "success";
    }
    catch (SQLException e) {
      e.printStackTrace();
    }
  }
```

SQLException を処理するメソッドの作成

サンプル・アプリケーションのコードの改良点として、SQLException をスローするメソッドで使用する例外処理のメソッドを作成します。例として、次のメソッドは、サンプル・アプリケーションのどのメソッドの catch ブロックからでもコールできます。このメソッドは累積されたすべての例外を処理し、それぞれについてスタック・トレースを出力します。

例 5-5 アプリケーションの SQLException を処理するメソッドの追加

```
public void logException( SQLException ex )
{
  while ( ex != null ) {
    ex.printStackTrace();
    ex = ex.getNextException();
  }
}
```

また、catch ブロックでは、メソッドが失敗した理由を説明するテキストを返せます。つまり、メソッドの catch ブロックを次のように記述できます。

```
catch ( SQLException ex ) {
  logException( ex );
  return "failure";
}
```

この機能をアプリケーションに追加するには、次の手順を実行します。

1. DataHandler.java で、logException メソッドを追加します。
2. try ブロックと catch ブロックを含むように各メソッドを編集します。
3. 各メソッドの catch ブロックで、logException メソッドを実行します。
4. String の戻り値を持つメソッドの場合は、次のように return 文を記述して、メソッドが失敗したことを示すメッセージを返します。

   ```
   return "failure";
   ```

サンプル・アプリケーションのナビゲーション

web.xml ファイルは、Web アプリケーションのデプロイメント・ディスクリプタです。web.xml ファイルの 1 つのセクションを、アプリケーションの開始ページの定義に使用できます。次に例を示します。

```
<web-app>
...
  <welcome-file>
    myWelcomeFile.jsp
  </welcome-file>
...
</web-app>
```

web.xml ファイルにようこそページを定義しなかった場合、一般的には、index という名前で拡張子に .html、.htm または .jsp を持つファイルが開始ページとして使用されます（存在する場合）。Jdeveloper で、アプリケーションのデフォルトの実行ターゲットにするページ、つまり、最初に表示されるアプリケーションのページを定義できます。ページは、プロジェクトのプロパティで定義します。

アプリケーションが起動して開始ページが表示された後、アプリケーション内のナビゲーションは次の方法を使用して実現されます。

- リンク : HTML アンカー・タグの形式で、リンクのターゲット（通常はナビゲート先の別の JSP ページを特定する）およびリンクのテキストを定義します。
- HTML 送信ボタン : このボタンを使用してページのフォーム（新規データまたは変更済のデータを入力するフォームなど）を送信します。
- jsp:forward タグ : 問合せおよびフォームを処理する JSP ページで実行され、同じ JSP ページに再度フォワードするかまたは別の JSP ページにフォワードします。

アプリケーションの開始ページの作成

次の手順では、index.jsp ページを作成します。このページはアプリケーションのデフォルトの開始ページになります。ページには表示要素はいっさい含まれません。単に、ユーザーをアプリケーションのログイン・ページ（login.jsp）にフォワードします。これを行うには、jsp:forward タグを使用します。jsp:forward タグは、問合せおよびフォームを処理する JSP ページで実行され、同じ JSP ページに再度フォワードするかまたは別の JSP ページにフォワードします。

1. 新しい JSP ページを作成して、index.jsp という名前を付けます。
2. サンプル・アプリケーションでは、このページにテキストを追加しません。コンポーネント・パレットの「JSP」ページから、**Forward** をドラッグして、jsp:forward タグをページに含めます。
3. フォワードタグの「Forward の挿入」ダイアログ・ボックスで、login.jsp を**「ページ」**として入力します。

これで、この新しいページをアプリケーションのデフォルト・ターゲットに指定できます。次の手順を実行します。

1. アプリケーション・ナビゲータで、View プロジェクトを右クリックして、「プロジェクト・プロパティ」を選択します。
2. 表示されたツリーで、**「実行 / デバッグ」**を選択します。「実行 / デバッグ」領域で「プロジェクト設定を使用」が選択されていることを確認し、「実行構成」領域で「デフォルト構成」が選択されていることを確認します。**「編集」**をクリックします。
3. 「起動設定の編集」ダイアログ・ボックスで、**「起動設定」**を選択します。右側の「起動設定」領域で、「デフォルトの実行ターゲット」フィールドの横にある**「参照」**をクリックして、今作成した新しい index.jsp ページを検索し、**「OK」**をクリックします。再度**「OK」**をクリックして、ダイアログ・ボックスを閉じます。

これで、**View** プロジェクトを右クリックし、ショートカット・メニューで**「実行」**を選択すると、アプリケーションを実行できます。アプリケーションを実行すると、アプリケーションのデフォルトの起動ターゲットとして設定されている index.jsp が実行されます。index.jsp によって直接ログイン・ページ（login.jsp）にフォワードされ、このページがブラウザに表示されます。

6

アプリケーションの拡張：拡張 JDBC 機能

この章では、Java アプリケーションで使用できる追加機能について説明します。一部の機能でサンプル・アプリケーションに実装されていないものもありますが、パフォーマンスを向上させるためにコード内で使用できる機能拡張もあります。

この章は次の項で構成されています。

動的 SQL の使用

本番環境では、動的 SQL、つまり即時に SQL 文を生成することが常に必要です。特に、データベースに対して行われる更新の場合など、最終的な問合せが実行時までわからないことがよくあります。

異なる更新値を持つ多数の類似問合せをデータベースに対して実行する必要がある場合は、`Statement` オブジェクトを拡張する `OraclePreparedStatement` オブジェクトを使用できます。この場合、リテラルの更新値をバインド変数に置き換えます。また、`OracleCallableStatement` オブジェクトを介してストアド・プロシージャをコールすることによって、データベースで PL/SQL ストアド・ファンクションを使用することもできます。

この項では、次の項目について説明します。

- OraclePreparedStatement の使用
- OracleCallableStatement の使用
- バインド変数の使用

OraclePreparedStatement の使用

データベースに対して静的 SQL 問合せを実行する場合は、`Statement` オブジェクトを使用します。ただし、複数の類似問合せを実行したり、データベースの多数の列に影響する複数の更新を実行する場合は、それぞれの問合せをアプリケーションにハード・コードすることはできません。

同じ SQL 文を複数回実行する場合は、`OraclePreparedStatement` を使用できます。次のような問合せについて考えてみます。

```
SELECT * FROM Employees WHERE ID=xyz;
```

この問合せの `xyz` の値が変更されるたびに、SQL 文を再度コンパイルする必要があります。

`OraclePreparedStatement` 機能を使用すると、実行する SQL 文はプリコンパイルされて `PreparedStatement` オブジェクトに格納されるため、実行するたびにコンパイルしなくても、何度でも実行できます。文中のデータが変化する場合は、そのデータのプレースホルダとしてバインド変数を使用し、実行時にリテラル値を指定することができます。

`OraclePreparedStatement` の使用について、次に例を示します。

例 6-1 PreparedStatement の作成

```
OraclePreparedStatement pstmt = conn.prepareStatement("UPDATE Employees
                                SET salary = ? WHERE ID = ?");
   pstmt.setBigDecimal(1, 153833.00)
   pstmt.setInt(2, 110592)
```

`OraclePreparedStatement` インタフェースを使用するメリットは、次のとおりです。

- 同じ `PreparedStatement` オブジェクトを使用することによって、更新をバッチ化できます。
- 何度も実行される SQL 文を 1 回目の実行時にのみコンパイルするため、パフォーマンスを改善できます。
- バインド変数を使用して、コードを単純かつ再利用可能にすることができます。

OracleCallableStatement の使用

OracleCallableStatement インタフェースを使用すると、データベース上のストアド・プロシージャにアクセスできます。このインタフェースは、OraclePreparedStatement インタフェースを拡張します。OracleCallableStatement インタフェースは、ストアド・プロシージャをコールするために、標準的な JDBC エスケープ構文で構成されています。このインタフェースは、結果パラメータの有無にかかわらず使用できます。ただし、結果パラメータを使用する場合は、それが OUT パラメータとして登録されている必要があります。このインタフェースでは、その他のパラメータとして、IN または OUT、あるいはその両方を使用できます。

これらのパラメータは、OraclePreparedStatement インタフェースから継承されたアクセッサ・メソッドを使用することによって設定されます。IN パラメータは、set*XXX* メソッドを使用して設定され、OUT パラメータは get*XXX* メソッドを使用して取得されます。*XXX* は、Java データ型のパラメータです。

また、CallableStatement は、複数の ResultSet オブジェクトを返すこともあります。

例として、次のように OracleCallableStatement を作成して、foo ストアド・プロシージャをコールすることができます。

例 6-2 CallableStatement の作成

```
OracleCallableStatement cs = (OracleCallableStatement)
conn.prepareCall("{call foo(?)}");
```

次のいずれかの方法で、文字列 bar をこのプロシージャに渡すことができます。

```
cs.setString(1,"bar"); // JDBC standard
// or...
cs.setString("myparameter","bar"); // Oracle extension
```

バインド変数の使用

バインド変数は、SQL 文中のリテラルのかわりとなる変数です。バインド変数を OraclePreparedStatement および OracleCallableStatement とともに使用して、SQL 文の構築に使用されるパラメータ値を指定します。バインド変数を使用すると、本番環境でのパフォーマンスが大幅に向上します。

PL/SQL ブロックまたはストアド・プロシージャ・コールには、入力変数と出力変数を区別するために、IN、OUT および IN OUT 修飾子を使用できます。入力変数値は set*XXX* メソッドを使用して設定され、OUT 変数値は get*XXX* メソッドを使用して取得されます。*XXX* は、Java データ型の値です。この値は、アクセス先のデータベース内の列の SQL データ型によって異なります。

ストアド・プロシージャのコール

Oracle Java Database Connectivity（JDBC）ドライバでは、PL/SQL ストアド・プロシージャおよび無名ブロックの処理がサポートされています。Oracle JDBC ドライバでは、Oracle PL/SQL ブロック構文およびほとんどの SQL92 エスケープ構文がサポートされています。次の PL/SQL コールは、すべての Oracle JDBC ドライバで動作します。

例 6-3 ストアド・プロシージャのコール

```
// SQL92 syntax
CallableStatement cs1 = conn.prepareCall
                        ( "{call proc (?,?)}" ) ; // stored proc
CallableStatement cs2 = conn.prepareCall
                        ( "{? = call func (?,?)}" ) ; // stored func
```

```
// Oracle PL/SQL block syntax
CallableStatement cs3 = conn.prepareCall
                        ( "begin proc (?,?); end;" ) ; // stored proc
CallableStatement cs4 = conn.prepareCall
                        ( "begin ? := func(?,?); end;" ) ; // stored func
```

Oracle 構文の例として、ストアド・ファンクションを作成する PL/SQL コードの一部を示します。この PL/SQL ファンクションでは文字列を取得して接尾辞を連結します。

例 6-4 ストアド・ファンクションの作成

```
create or replace function foo (val1 char)
return char as
begin
return val1 || 'suffix';
end;
```

次のように、Java プログラムでこのストアド・ファンクションをコールできます。

例 6-5 Java でのストアド・ファンクションのコール

```
OracleDataSource ods = new OracleDataSource();
ods.setURL("jdbc:oracle:thin:@<hoststring>");
ods.setUser("hr");
ods.setPassword("hr");
Connection conn = ods.getConnection();
CallableStatement cs = conn.prepareCall ("begin ? := foo(?); end;");
cs.registerOutParameter(1,Types.CHAR);
cs.setString(2, "aa");
cs.executeUpdate();
String result = cs.getString(1);
```

次の項では、このマニュアルのサンプル・アプリケーションでストアド・プロシージャを使用する方法について説明します。

- JDeveloper での PL/SQL ストアド・プロシージャの作成
- ストアド・プロシージャを使用するためのメソッドの作成
- ユーザーによるストアド・プロシージャの選択の許可
- アプリケーションからのストアド・プロシージャのコール

JDeveloper での PL/SQL ストアド・プロシージャの作成

JDeveloper を使用すると、接続ナビゲータを使用して、ストアド・プロシージャをデータベースに作成できます。次の手順では、サンプル・アプリケーションで従業員レコードを挿入するための代替方法として使用できるストアド・プロシージャを作成します。

1. **「接続」**タブを選択して、接続ナビゲータを表示します。
2. データベース接続ノード（デフォルトでは、DBConnection1）および **HR** ノードを開き、HR データベース内のオブジェクトを表示します。
3. **「プロシージャ」**を右クリックし、**「新規 PL/SQL プロシージャ」**を選択します。
4. 「PL/SQL プロシージャ作成」ダイアログ・ボックスで、オブジェクト名として `insert_employee` と入力します。**「OK」**をクリックします。

   プロシージャのスケルトン・コードがソース・エディタに表示されます。

5. プロシージャ名の後に、次のコード行を入力します。

```
PROCEDURE    "INSERT_EMPLOYEE" (p_first_name  employees.first_name%type,
  p_last_name    employees.last_name%type,
  p_email        employees.email%type,
  p_phone_number employees.phone_number%type,
  p_job_id       employees.job_id%type,
  p_salary       employees.salary%type
)
```

6. BEGIN 文の後の NULL を示す行を次の内容で置き換えます。

```
  INSERT INTO Employees VALUES (EMPLOYEES_SEQ.nextval, p_first_name ,
    p_last_name , p_email , p_phone_number, SYSDATE, p_job_id,
    p_salary,.30,100,80);
```

   この文では、DataHandler.java クラスの addEmployee メソッドの最後の 3 行に使用される、ハードコードされた同じ値が使用されていることがわかります。

7. END 文にプロシージャ名を追加します。

```
END insert_employee;
```

8. ファイルを保存し、コンパイル・エラーがないかを確認します。

例 6-6 に、ストアド・プロシージャの完成したコードを示します。

例 6-6 従業員データを挿入するための PL/SQL ストアド・プロシージャの作成

```
PROCEDURE    "INSERT_EMPLOYEE" (p_first_name  employees.first_name%type,
  p_last_name    employees.last_name%type,
  p_email        employees.email%type,
  p_phone_number employees.phone_number%type,
  p_job_id       employees.job_id%type,
  p_salary       employees.salary%type
)
AS
BEGIN
  INSERT INTO Employees VALUES (EMPLOYEES_SEQ.nextval, p_first_name ,
    p_last_name , p_email , p_phone_number, SYSDATE, p_job_id,
    p_salary,.30,100,80);
END insert_employee;
```

ストアド・プロシージャを使用するためのメソッドの作成

次の手順では、addEmployee メソッドのかわりに使用できるメソッドを DataHandler.java クラスに追加します。ここで追加する新しいメソッドでは、insert_employee ストアド・プロシージャを使用します。

1. **「アプリケーション」**タブを選択して、アプリケーション・ナビゲータを表示します。
2. DataHandler.java ファイルが Java ソース・エディタでまだ開いていない場合は、ダブルクリックして開きます。
3. 次のように CallableStatement をインポートします。

```
import java.sql.CallableStatement;
```

4. addEmployee メソッドの後に、addEmployeeSP メソッドの宣言を追加します。

```
public String addEmployeeSP(String first_name, String last_name,
  String email, String phone_number, String job_id,
  int salary) throws SQLException {
}
```

このメソッドのシグネチャは addEmployee のものと同じです。

5. このメソッド内に try ブロックを追加し、このブロック内でデータベースに接続します。

```
try {
  getDBConnection();
}
```

6. さらに、try ブロック内で次の SQL 文字列を作成します。

```
sqlString = "begin hr.insert_employee(?,?,?,?,?,?); end;";
```

文中の疑問符（?）はバインド変数で、ストアド・プロシージャによって使用される、first_name、last_name などの値のプレースホルダの役割を果たします。

7. CallableStatement を作成します。

```
CallableStatement callstmt = conn.prepareCall(sqlString);
```

8. IN パラメータを設定します。

```
  callstmt.setString(1, first_name);
  callstmt.setString(2, last_name);
  callstmt.setString(3, email);
  callstmt.setString(4, phone_number);
  callstmt.setString(5, job_id);
  callstmt.setInt(6, salary);
```

9. トレース・メッセージを追加し、コール可能文を実行します。

```
  System.out.println("\nInserting with stored procedure: " +
                     sqlString);
  callstmt.execute();
```

10. 戻りメッセージを追加します。

```
  return "success";
```

11. try ブロックの後に catch ブロックを追加して、すべてのエラーを捕捉します。例 5-5 で作成した logException をコールします。

```
catch ( SQLException ex ) {
  System.out.println("Possible source of error: Make sure you have created the
stored procedure");
  logException( ex );
  return "failure";
}
```

12. DataHandler.java を保存します。

例 6-7 に、完成したメソッドを示します。

注意： logException() メソッド（例 5-5 を参照）を追加していない場合は、logException(ex) の下に表示される赤い波線によってエラーが示されます。ファイルのコンパイルに進む前に、このメソッドが DataHandler.java クラスに存在する必要があります。

例 6-7 Java での PL/SQL ストアド・プロシージャの使用

```
public String addEmployeeSP(String first_name, String last_name,
  String email, String phone_number, String job_id,
  int salary) throws SQLException {

  try {
    getDBConnection();
    sqlString = "begin hr.insert_employee(?,?,?,?,?,?); end;";
    CallableStatement callstmt = conn.prepareCall(sqlString);
    callstmt.setString(1, first_name);
    callstmt.setString(2, last_name);
    callstmt.setString(3, email);
    callstmt.setString(4, phone_number);
    callstmt.setString(5, job_id);
    callstmt.setInt(6, salary);
    System.out.println("\nInserting with stored procedure: " +
                       sqlString);

    callstmt.execute();
    return "success";
  }
  catch ( SQLException ex ) {
    System.out.println("Possible source of error: Make sure you have created the stored
procedure");
    logException( ex );
    return "failure";
  }
}
```

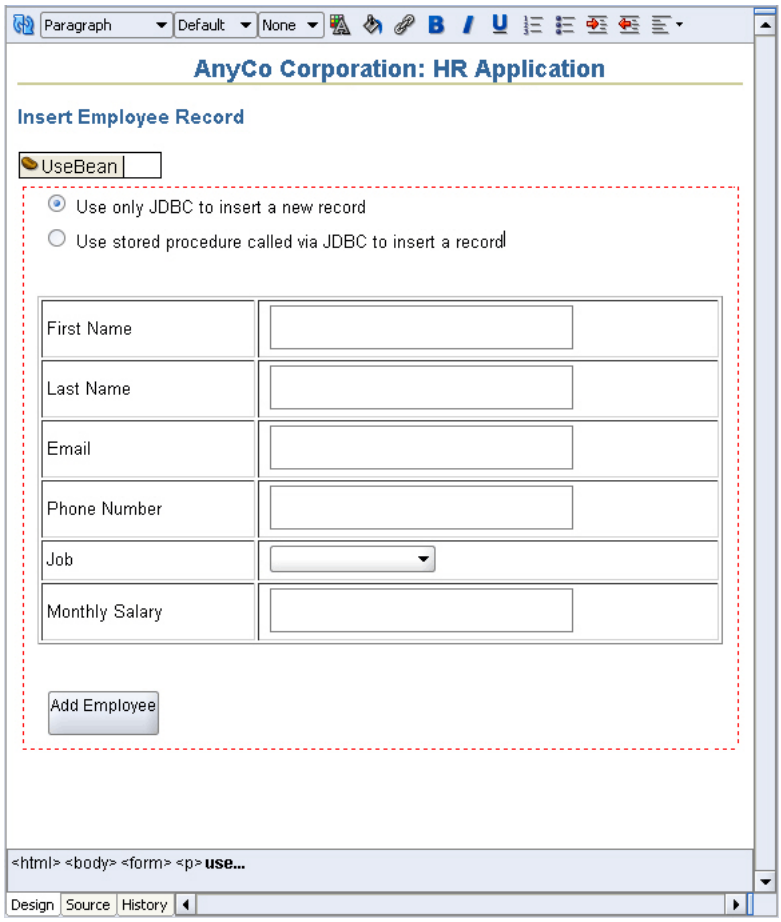
ユーザーによるストアド・プロシージャの選択の許可

この項の手順では、insert.jsp ページにラジオ・ボタンを追加します。従業員を挿入する場合に、ストアド・プロシージャを使用するか、または Java コードで SQL 問合せを使用するかを、このラジオ・ボタンによって、ユーザーが選択できるようにします。

1. insert.jsp を開いていない場合は、ビジュアル・エディタで開きます。
2. 「Insert Employee Record」という見出しの後に、新しい行を作成します。この新しい行にカーソルを置き、**「UseBean」**をコンポーネント・パレットの「JSP」ページからドラッグして jsp:useBean タグをこのページに追加します。ID として empsbean を入力し、**「クラス」**として hr.DataHandler を参照して選択し、**「有効範囲」**を「セッション」に設定します。ページ上で UseBean が選択されている状態で、この行のスタイルを Heading 3 ではなく None に設定します。
3. **「Radio Button」**コンポーネントをコンポーネント・パレットの「HTML Forms」ページから表の上にあるフォーム内のページにドラッグします。「ラジオ・ボタンの挿入」ダイアログ・ボックスで、**「名前」**に useSP を入力し、**「値」**に false を入力して、**「チェック」**を選択します。**「OK」**をクリックします。
4. ビジュアル・エディタでボタンの右側にカーソルを置き、「新しいレコードの追加には JDBC のみを使用する」など、ボタンの目的を説明するテキストを入力します。
5. 現在の行の末尾で [Enter] を押して、新しい行を作成します。
6. 2 つ目の**「Radio Button」**を最初のラジオ・ボタンの下にドラッグします。「ラジオ・ボタンの挿入」ダイアログ・ボックスで、**「名前」**に useSP を入力し、**「値」**に true を入力して、「チェック」チェック・ボックスが選択されていないことを確認します。
7. ビジュアル・エディタでボタンの右側にカーソルを置き、「レコードの追加には JDBC からコールするストアド・プロシージャのみを使用する」など、ボタンの目的を説明するテキストを入力します。
8. ページを保存します。

図 6-1 に、ストアド・プロシージャを使用するときのオプションとなるラジオ・ボタンのある insert.jsp を示します。

図 6-1 ストアド・プロシージャ・オプションのためのリンクの追加



アプリケーションからのストアド・プロシージャのコール

この項の手順では、insert.jsp ページ上でフォームを処理する insert_action.jsp ファイルを変更し、ラジオ・ボタンの選択によって、新しい従業員レコードを挿入するための適切なメソッドを選択します。

1. insert_action.jsp を開いていない場合は、ビジュアル・エディタで開きます。
2. スクリプトレットをダブルクリックして「スクリプトレットのプロパティ」ダイアログ・ボックスを表示し、次のように salary 変数の後に新しい変数を追加します。

```
String useSPFlag = request.getParameter("useSP");
```

3. 「Scriptlet のプロパティ」ダイアログ・ボックスのまま、その下で、既存の empsbean.addEmployee 行を次のコード行で置き換えます。このコード行では、addEmployeeSP メソッドまたは pure JDBC addEmployee メソッドを選択してレコードを挿入します。

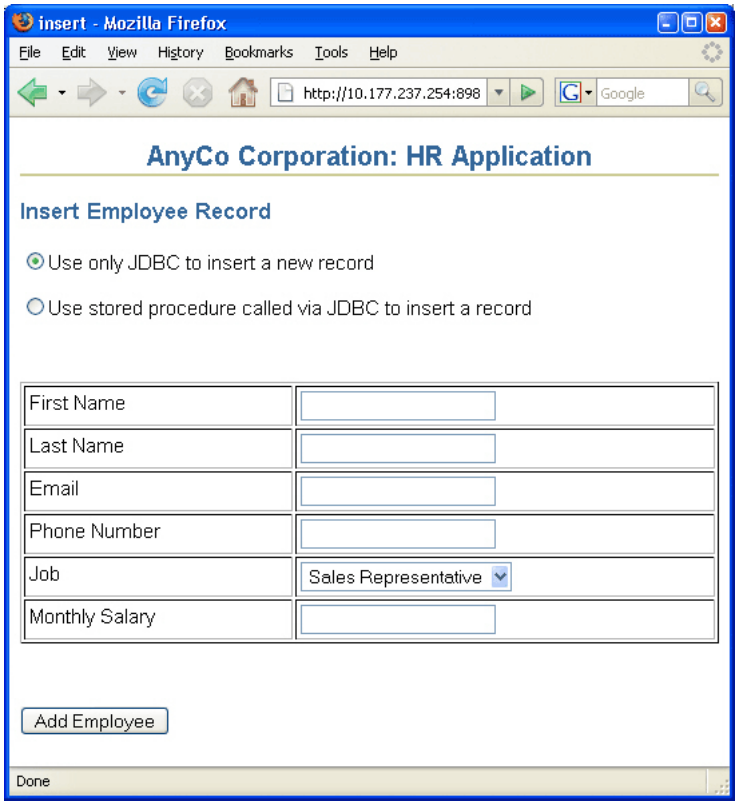
```
if ( useSPFlag.equalsIgnoreCase("true"))
  empsbean.addEmployeeSP(first_name, last_name, email,
  phone_number, job_id, salary.intValue());
// otherwise use pure JDBC insert
```

```
else
  empsbean.addEmployee(first_name, last_name, email,
  phone_number, job_id, salary.intValue());
```

4. insert_action.jsp を保存します。

これで、アプリケーションを実行すると、挿入ページのラジオ・ボタンを使用して新しい従業員レコードを挿入する方法を選択できます。ブラウザでは、ページは図 6-2 のように表示されます。

図 6-2 ストアド・プロシージャを使用したレコードの入力



カーソル変数の使用

Oracle JDBC ドライバは、REF CURSOR 型のカーソル変数をサポートします。これは、JDBC 標準には含まれません。REF CURSOR 型は、JDBC 結果セットとしてサポートされます。

カーソル変数は、問合せ作業領域の内容ではなく、メモリーの場所を保持します。カーソル変数を宣言すると、ポインタが作成されます。SQL では、ポインタのデータ型は REF *x* です。REF は REFERENCE の短縮であり、*x* は参照される実体を示します。したがって、REF CURSOR はカーソル変数への参照を示します。多数の作業領域を指すために多数のカーソル変数が存在する場合があるため、REF CURSOR は、多数の異なる型のカーソル変数を識別するカテゴリまたはデータ型指定子と考えることができます。本来、REF CURSOR は問合せ結果をカプセル化します。

Oracle は結果セットを返しません。問合せによって返されたデータにアクセスするには、CURSOR および REF CURSOR を使用します。CURSOR には、問合せ結果およびメタデータが含まれます。REF CURSOR（CURSOR 変数）データ型には、カーソルへの参照が含まれます。RDBMS とクライアント、またはデータベースで PL/SQL と Java の間での受け渡しが可能です。問合せまたはストアド・プロシージャから返される場合もあります。

注意： REF CURSOR インスタンスはスクロールできません。

この項では、次の項目について説明します。

- Oracle の REF CURSOR 型のカテゴリ
- REF CURSOR データへのアクセス
- サンプル・アプリケーションでの REF CURSOR の使用

Oracle の REF CURSOR 型のカテゴリ

カーソル変数を作成するには、REF CURSOR カテゴリに属する型を識別することから開始します。次に例を示します。

```
dept_cv DeptCursorTyp
...
```

次に、DeptCursorTyp 型であることを宣言して、カーソル変数を作成します。

例 6-8 REF CURSOR 型の宣言

```
DECLARE TYPE DeptCursorTyp IS REF CURSOR
```

REF CURSOR は、特定のデータ型というよりも、データ型のカテゴリです。ストアド・プロシージャは、REF CURSOR カテゴリのカーソル変数を返すことがあります。この出力は、データベース・カーソルまたは JDBC 結果セットと同等です。

REF CURSOR データへのアクセス

Java では、REF CURSOR は ResultSet オブジェクトとしてマテリアライズされ、次のようにアクセスできます。

例 6-9 Java での REF CURSOR データへのアクセス

```
import oracle.jdbc.*;
...
CallableStatement cstmt;
ResultSet cursor;

// Use a PL/SQL block to open the cursor
cstmt = conn.prepareCall
         ("begin open ? for select ename from emp; end;");

cstmt.registerOutParameter(1, OracleTypes.CURSOR);
cstmt.execute();
cursor = ((OracleCallableStatement)cstmt).getCursor(1);

// Use the cursor like a normal ResultSet
while (cursor.next ())
    {System.out.println (cursor.getString(1));}
```

前述の例では、次の手順が実行されます。

1. 接続クラスの prepareCall メソッドを使用して、CallableStatement オブジェクトが作成されます。
2. コール可能文によって、REF CURSOR を返す PL/SQL プロシージャが実装されます。
3. 通常どおり、コール可能文の出力パラメータは、その型を定義するために登録されている必要があります。REF CURSOR には、型コード OracleTypes.CURSOR を使用します。

4. コール可能文が実行され、REF CURSOR が返されます。
5. getCursor メソッドを使用するために、CallableStatement オブジェクトが OracleCallableStatement にキャストされます。このメソッドは、標準 JDBC Application Program Interface（API）に対する Oracle の拡張機能であり、REF CURSOR を ResultSet オブジェクトに返します。

サンプル・アプリケーションでの REF CURSOR の使用

次の項では、サンプル・アプリケーションを拡張し、新しい従業員レコードを挿入するときに、動的に生成された職務 ID と職務名のリストを「ジョブ」フィールドに表示します。

- データベースでのパッケージの作成
- データベース・ファンクションの作成
- メソッドからの REF CURSOR のコール
- 動的に生成されたリストの表示

これを行うには、REF CURSOR を使用して、職務の結果セットを Jobs 表から取得するデータベース・ファンクション GET_JOBS を作成します。新しい Java メソッド getJobs は、このデータベース・ファンクションをコールして結果セットを取得します。

データベースでのパッケージの作成

次の手順では、データベースで新しいパッケージを作成して、REF CURSOR 宣言を保持します。

1. **「接続」**タブを選択して、ナビゲータで表示します。
2. **Database** ノード、**DBConnection1** ノードおよび **HR** ノードを開き、データベース・オブジェクトのリストを表示します。「パッケージ」にスクロールします。**「パッケージ」**を右クリックし、**「新規 PL/SQL パッケージ」**を選択します。
3. 「PL/SQL パッケージ作成」ダイアログ・ボックスで、名前として JOBSPKG と入力します。**「OK」**をクリックします。パッケージの定義がソース・エディタに表示されます。
4. 最初の行の末尾にカーソルを置き、[Enter] を押して新しい行を作成します。新しい行で、次のように REF CURSOR を宣言します。

```
TYPE ref_cursor IS REF CURSOR;
```

5. パッケージを保存します。

例 6-10 に、パッケージのコードを示します。

例 6-10 データベースでのパッケージの作成

```
PACKAGE "JOBSPKG" AS
    TYPE ref_cursor IS REF CURSOR;
END;
```

データベース・ファンクションの作成

次の手順では、REF CURSOR を使用して、職務の結果セットを Jobs 表から取得するデータベース・ファンクション GET_JOBS を作成します。

1. 接続ナビゲータで、必要なノードを再度開いて、HR データベースのオブジェクトを表示します。**「ファンクション」**を右クリックし、ショートカット・メニューで**「新規 PL/SQL ファンクション」**を選択します。
2. 「PL/SQL ファンクション作成」ダイアログ・ボックスで、名前として GET_JOBS と入力します。**「OK」**をクリックします。GET_JOBS ファンクションの定義がソース・エディタに表示されます。

3. ファンクション定義の最初の行で、VARCHAR2 のかわりに、戻り値として JobsPkg.ref_cursor を指定します。
4. AS キーワードの後に、次のように入力します。

```
jobs_cursor JobsPkg.ref_cursor;
```

5. BEGIN ブロックで、次のコードを入力して現在の内容を置き換えます。

```
OPEN jobs_cursor FOR
SELECT job_id, job_title FROM jobs;
RETURN jobs_cursor;
```

6. ファンクションを保存します。

例 6-11 に、ファンクションのコードを示します。

例 6-11 ストアド・ファンクションの作成

```
FUNCTION    "GET_JOBS"
RETURN JobsPkg.ref_cursor
AS jobs_cursor JobsPkg.ref_cursor;
BEGIN
  OPEN jobs_cursor FOR
  SELECT job_id, job_title FROM jobs;
  RETURN jobs_cursor;
END;
```

メソッドからの REF CURSOR のコール

次の手順では、GET_JOBS ファンクションをコールして結果セットを取得する Java メソッド getJobs を DataHandler クラスに作成します。

1. DataHandler.java がまだ開いていない場合は、ダブルクリックしてソース・エディタで開きます。
2. メソッド宣言を入力します。

```
public ResultSet getJobs() throws SQLException {

}
```

3. メソッド本体で、データベースに接続します。

```
getDBConnection();
```

4. 接続に続いて、新しい変数 jobquery を宣言します。

```
String jobquery = "begin ? := get_jobs; end;";
```

5. prepareCall メソッドを使用して、CallableStatement を作成します。

```
CallableStatement callStmt = conn.prepareCall(jobquery);
```

6. Oracle 固有の型を使用して、OUT パラメータの型を登録します。

```
callStmt.registerOutParameter(1, OracleTypes.CURSOR);
```

7. Oracle 固有の型を使用することを指定すると、[Alt]+[Enter] キーを押して oracle.jdbc.OracleTypes をインポートすることを求めるメッセージが表示されます。[Alt]+[Enter] キーを押した後、表示されるリストで **OracleTypes**（oracle.jdbc）を選択します。
8. 文を実行して結果セットを返します。

```
callStmt.execute();
rset = (ResultSet)callStmt.getObject(1);
```

9. ここまでに入力したコードを try ブロックで囲みます。
10. 例外を捕捉するための catch ブロックを追加し、logException メソッドもコールします。

```
catch ( SQLException ex ) {
  logException( ex );
}
```

11. catch ブロックを閉じた後で、結果セットを返します。

```
return rset;
```

12. 構文をチェックするためのファイルを作成します。

getJobs メソッドのコードを次に示します。

```
public ResultSet getJobs() throws SQLException {
try {
  getDBConnection();
  String jobquery = "begin ? := get_jobs; end;";
  CallableStatement callStmt = conn.prepareCall(jobquery);
  callStmt.registerOutParameter(1, OracleTypes.CURSOR);
  callStmt.execute();
  rset = (ResultSet)callStmt.getObject(1);
} catch ( SQLException ex ) {
logException( ex );
}
return rset;
}
```

動的に生成されたリストの表示

職務 ID と職務名のリストを表示するドロップダウン・リストを挿入ページに作成するために、職務 ID と職務名をハードコードしました。次の手順では、これを動的に生成されたリストに置き換えます。このリストは、前の項で REF CURSOR によって作成したリストです。

1. insert.jsp が開いていない場合は、アプリケーション・ナビゲータのビジュアル・エディタでダブルクリックして開きます。
2. **「Page Directive」**を useBean タグの右側のページにドラッグします。「Page Directive の挿入」ダイアログ・ボックスで、**「言語」**に Java を入力し、**「インポート」**フィールドで **java.sql.ResultSet** を参照して選択します。**「OK」**をクリックします。
3. 「Page Directive」の隣のページにスクリプトレットをドラッグします。「スクリプトレットの挿入」ダイアログ・ボックスで、次のコードを追加し、getJobs メソッドを実行して、職務のリストを含む結果セットを返します。

```
ResultSet rset = empsbean.getJobs();
```

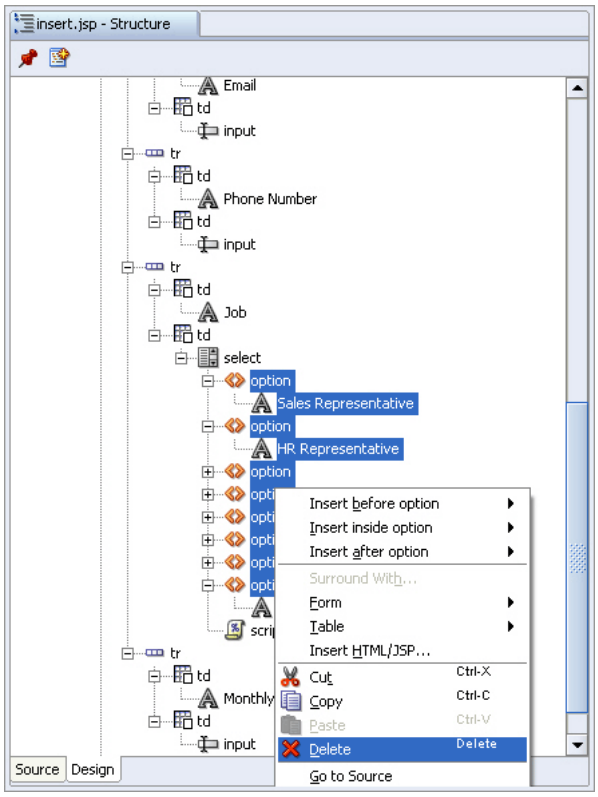
4. ページで **ListBox** コンポーネントを選択し、JSP のコンポーネント・パレットの**「スクリプトレット」**をクリックします。（この場合、スクリプトレットをページにドラッグ・アンド・ドロップする必要はありません。）「スクリプトレットの挿入」ダイアログ・ボックスが表示されます。
5. 次のコードを「スクリプトレットの挿入」ダイアログ・ボックスに入力します。**「OK」**をクリックします。

```
while (rset.next ())
 {
  out.println("<option value=" + rset.getString("job_id") + ">" +
  rset.getString("job_title") + "</option> "  );
 }
```

6. 次のように、ハードコードされた値を削除します。

   ListBox コンポーネントが選択された状態のまま、「構造」ウィンドウで**「ジョブ」**フィールドにスクロールします。select キーワードの下のハードコードされたオプションのリストを確認します。各オプションを削除しますが、スクリプトレットは保持されることを確認します。

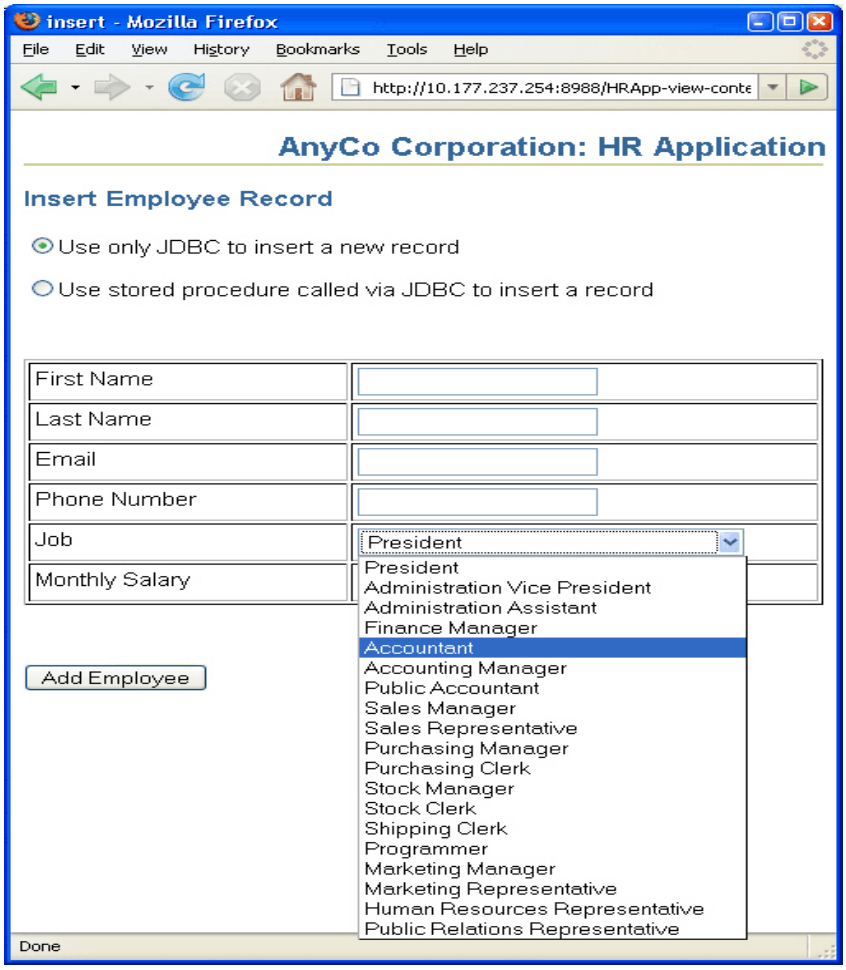
図 6-3 ドロップダウン・リストボックス・オプションの「構造」ビュー



7. ページを保存します。

ここで、アプリケーションを実行し、クリックして新しい従業員を挿入し、リストを使用して選択可能な職務のリストを表示します。図 6-4 に、ブラウザでの動的な職務のリストを示します。

図 6-4 ブラウザでの動的に生成されたリスト



7

Oracle ADF を使用したマスター・ディテール・アプリケーションの作成

この章では、Oracle Application Developer Framework（Oracle ADF）を使用したマスター・ディテール・アプリケーションの作成方法について説明します。内容は次のとおりです。

マスター・ディテール・アプリケーションの概要

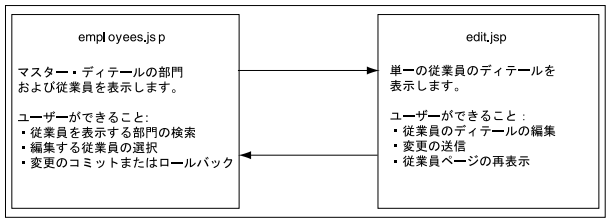
マスター・ディテール・アプリケーションを使用すると、関連する表のデータを同時に表示できます。マスター表のレコードをディテール表の関連するレコードとともに表示できます。マスター・ディテール・データを編集するためのプロビジョニングがアプリケーションに組み込まれている場合は、共通のインタフェースを使用して両方の表のデータを編集することもできます。この章で作成するマスター・ディテール・アプリケーションは、次のもので構成されます。

- Java/XML オブジェクトの ADF 中間層セット。HR スキーマの表内のデータへのアクセスおよび更新を可能にします。これは、`model` という名前の 1 つのプロジェクトにあります。
- ユーザー・インタフェース（UI）、つまりビュー。アプリケーションの UI として機能する JSP のセットで構成されます。これは、`view` という名前のプロジェクトにあります。

`model` および `view` プロジェクトは、Java EE Model-View-Controller（MVC）デザイン・パターンに基づいています。このデザイン・パターンは、Oracle ADF を使用して簡単に実装できます。

図 7-1 に、このアプリケーション用に開発された項目の関係を示します。

図 7-1 マスター・ディテール・アプリケーションのページ



このアプリケーションは、Oracle Database の HR スキーマにアクセスします。マスター表として部門表を使用して、従業員表からディテール・データを表示します。この章では、Oracle ADF と JDeveloper を使用してこのアプリケーションを作成する方法について説明します。

Oracle ADF の使用

Oracle ADF は、Java EE 標準およびオープンソース・テクノロジに基づくエンドツーエンド・アプリケーション・フレームワークであり、サービス指向アプリケーションの作成を簡略化および促進します。Oracle ADF を使用すると、Web、ワイヤレス、デスクトップまたは Web サービス・インタフェースを使用してデータを検索、表示、作成、変更および検証するエンタープライズ・ソリューションを開発できます。Oracle JDeveloper 10*g* と Oracle ADF を連携して使用することで、ドラッグ・アンド・ドロップによるデータ・バインディング、ビジュアルな UI 設計およびチーム開発機能が組み込まれた、設計からデプロイまでの全開発のライフサイクルを網羅する環境が提供されます。

次の項では、マスター・ディテール・アプリケーションの作成に使用するいくつかの Oracle ADF 機能について説明します。

- Oracle ADF ビジネス・コンポーネント
- Oracle ADF Faces
- ADF データ・コントロール

関連項目：

- Oracle ADF アーキテクチャの詳細は、`http://www.oracle.com/technology/products/jdev/collateral/papers/1013/adf_10.1.3_overview.pdf` を参照してください。
- Oracle ADF でのリソースのコンパイルの詳細は、`http://www.oracle.com/technology/products/jdev/tips/muench/requiredreading/index.html` を参照してください。

Oracle ADF ビジネス・コンポーネント

Oracle ADF ビジネス・コンポーネントは、データベース中心の Enterprise Java EE アプリケーション用のビジネス・サービスを開発するための Java EE 準拠のテクノロジです。Oracle ADF ビジネス・コンポーネントにより、Oracle Forms などの 4GL ツールに慣れた開発者にとって、ビジネス・サービスの構築が簡単になります。

Oracle ADF ビジネス・コンポーネントのテクノロジには、次のような特徴があります。

- オブジェクト・リレーショナル・マッピングおよび独自のライブラリ・クラスのインスタンスの永続性を自動的に処理します。
- SQL を使用するデータ取得の複雑なリクエストを可能にします。
- トランザクション管理を自動的に処理します。
- 複雑なビジネス・ロジックを実装するためのフレームワークを提供します。
- 多数の Java EE デザイン・パターンを自動的に実装します。
- アプリケーションのパフォーマンスとスケーラビリティを向上させる強力なキャッシングおよびデータ受動化システムを備えています。

これらの機能はすべて完全にカスタマイズ可能です。このマニュアルで作成するアプリケーションの ADF 中間層を作成するには、データベース表から ADF ビジネス・コンポーネントを作成します。

Oracle ADF Faces

Oracle ADF Faces は、JavaServer Faces（JSF）JSR 127 仕様に基づいています。Oracle ADF Faces コンポーネントは、アプリケーションのユーザー・インタフェースで使用されます。これらのコンポーネントは、JSF をサポートする任意の IDE で使用できます。

Oracle ADF Faces を使用すると、アプリケーションの一貫したルック・アンド・フィールを設定できます。これにより、ルック・アンド・フィールに準拠することより、ユーザー・インタフェースの相互作用に重点を置くことができます。ADF Faces コンポーネントは、多言語および翻訳の実装やアクセシビリティ機能もサポートしています。

JDeveloper には、複数の設計ツール、ウィザード、特殊なダイアログ・ボックスおよびプロパティ・エディタがあり、これらは ADF Faces コンポーネントをページに挿入して使用するのに役立ちます。たとえば、ビジュアル・エディタでは、コンポーネント・パレットからコンポーネントをドラッグ・アンド・ドロップしてユーザー・インタフェースを設計できます。XML または JSP/HTML のコーディングに慣れている場合は、ページ・ファイルのソースを編集して、ADF Faces コンポーネントのタグを挿入することもできます。

Oracle ADF Faces でのファセットの使用方法

ファセットは、名前付きの子コンポーネントに似ています。親コンポーネント内でファセットを使用して、ページでの子コンポーネントの表示方法を制御します。ファセットは、特定のタイプの UI コンポーネントのプレースホルダです。

サンプル・アプリケーションで使用される ADF Faces af:panelPage コンポーネントでは、ページ全体をレイアウトできます。ページレベルおよびアプリケーションレベルのテキスト、イメージ、アクションおよびボタンをページの特定の領域にレイアウトするためのファセットがサポートされています。

ADF Faces コンポーネントをページにドロップすると、JSP/HTML ビジュアル・エディタには、点線の四角形としてファセットが表示されます。

ADF データ・コントロール

Oracle ADF データ・コントロールによって、アプリケーション・クライアントは、モデル・オブジェクト・レイヤーによって定義されたビジネス・サービスにアクセスできます。ビジネス・サービスは、モデル・プロジェクトによって定義された任意のコレクション、値またはアクションです。実行時に、データ・バインドされた UI コンポーネントは、ビジネス・サービスによって定義されたビジネス・サービスにアクセスできます。

Oracle ADF ビジネス・コンポーネントをビジネス・サービス・テクノロジとして使用すると、データ・モデル・コンポーネントが Oracle ADF データ・コントロールとしてモデル・レイヤーに公開されます。サンプル・アプリケーションでは、データ・コントロール・インタフェースはすでに実装されているため、作成する Oracle ADF ビジネス・コンポーネントのデータ・コントロールを作成する必要はありません。

アプリケーションおよびプロジェクトの作成

マスター・ディテール・アプリケーションの開発に進む前に、アプリケーションとデータベースの間の接続を確立する Connection オブジェクトを作成する必要があります。Connection オブジェクトの作成手順は、第 3 章を参照してください。

1. 「**ファイル**」メニューから「**新規**」を選択して新規ギャラリを表示します。「**一般**」カテゴリから、「**アプリケーション**」を選択します。
2. 「アプリケーションの作成」ダイアログ・ボックスで、アプリケーションの「**名前**」として AnyCo_ADF_MD と入力し、「**アプリケーション・パッケージの接頭辞**」は空白のままにして、「**アプリケーション・テンプレート**」で「**テンプレートなし [すべてのテクノロジ]**」を選択します。「**OK**」をクリックします。
3. 「プロジェクトの作成」ダイアログ・ボックスで、「**プロジェクト名**」として model と入力し、「**OK**」をクリックします。

model というプロジェクトを含む AnyCo_ADF_MD という名前のアプリケーションができました。

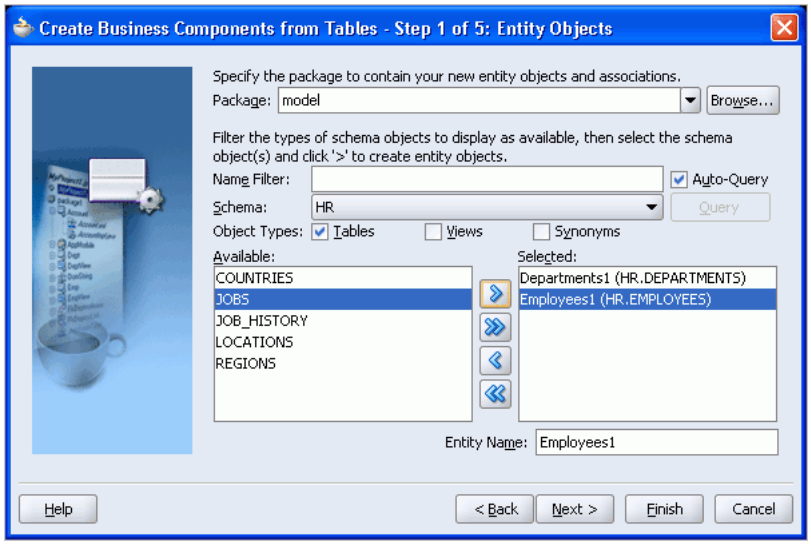
model プロジェクトでのビジネス・コンポーネントの作成

model プロジェクトで、hr.Departments および hr.Employees 表をアプリケーションで表示および編集できるようにする ADF ビジネス・コンポーネントを作成します。

1. JDeveloper のアプリケーション・ナビゲータで、**model** プロジェクトを選択します。
2. 「**ファイル**」メニューから「**新規**」を選択して新規ギャラリを表示します。**ビジネス層**カテゴリを開き、「**ADF Business Components**」を選択します。「**項目**」リストで、「**表からのビジネス・コンポーネント**」を選択します。
3. 「ビジネス・コンポーネント・プロジェクトの初期化」画面で、**hr** 接続が選択されていることを確認し、「**OK**」をクリックします。
4. 「表からのビジネス・コンポーネントの作成 - ようこそ」画面で、「**次へ**」をクリックします。

5. 「エンティティ・オブジェクト」画面で、選択可能な表のリストを表示するには、**「自動問合せ」**チェックボックスを選択します。図 7-2 に示すように、表の**「選択可能」**リストから、**DEPARTMENTS** および **EMPLOYEES** 表を**「選択済」**リストに移動します。**「次へ」**をクリックします。

図 7-2 エンティティ・オブジェクトを作成するスキーマ・オブジェクトの選択

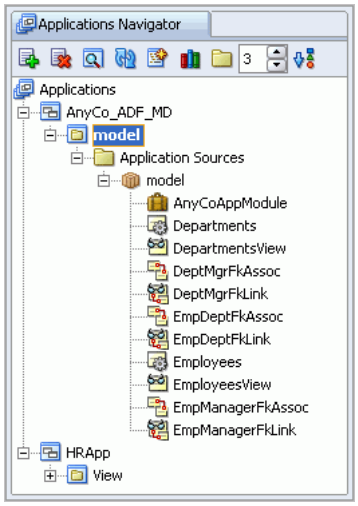


6. 「更新可能なビュー・オブジェクト」画面で、**Departments (HR.DEPARTMENTS)** および **Employees (HR.EMPLOYEES)** を**「選択済」**リストに移動します。**「次へ」**をクリックします。
7. 「読取り専用ビュー・オブジェクト」画面で、**「次へ」**をクリックします。
8. 「アプリケーション・モジュール」画面で、ADF アプリケーション・モジュールに名前を付けることができます。**「名前」**として `AnyCoAppModule` を入力します。**「次へ」**をクリックします。
9. 「ダイアグラム」で、ビジネス・コンポーネント・ダイアグラムをリクエストできます。ただし、このアプリケーションでは重要ではないため、**「次へ」**をクリックします。
10. 「終了」ダイアログ・ボックスで、詳細を確認し、正しければ**「終了」**をクリックします。定義したビジネス・コンポーネントが `model` プロジェクトに作成されます。
11. 作業内容をすべて保存します。

ADF 中間層を生成しました。これに対して、ユーザー・インタフェースを作成できます。この時点で、データベースからの外部キー関係を処理するその他のビジネス・コンポーネント・オブジェクトも生成されます。`DeptMgrFkAssoc`、`DeptMgrFkLink` などです。

図 7-3 に、アプリケーション・ナビゲータを示します。model プロジェクトの内容を開いて、アプリケーションに作成される項目を表示できます。

図 7-3 ADF モデルのナビゲート



マスター・ディテール・データの表示

アプリケーション・ユーザー・インタフェースは、一連の JSP ページで構成されます。このアプリケーションでは、ビューと呼ばれるユーザー・インタフェース（UI）は別のプロジェクトで定義されます。

アプリケーション UI のプロジェクトの作成

アプリケーション UI を作成するには、次のように view という名前のプロジェクトを定義します。

1. アプリケーション・ナビゲータで **AnyCo_ADF_MD** アプリケーションを選択し、**「ファイル」**メニューで**「新規」**を選択して、新規ギャラリを表示します。**「一般」**カテゴリから、**「空のプロジェクト」**を選択します。
2. 「プロジェクトの作成」ダイアログ・ボックスで、新しいプロジェクトの**「名前」**として view と入力し、**「OK」**をクリックします。

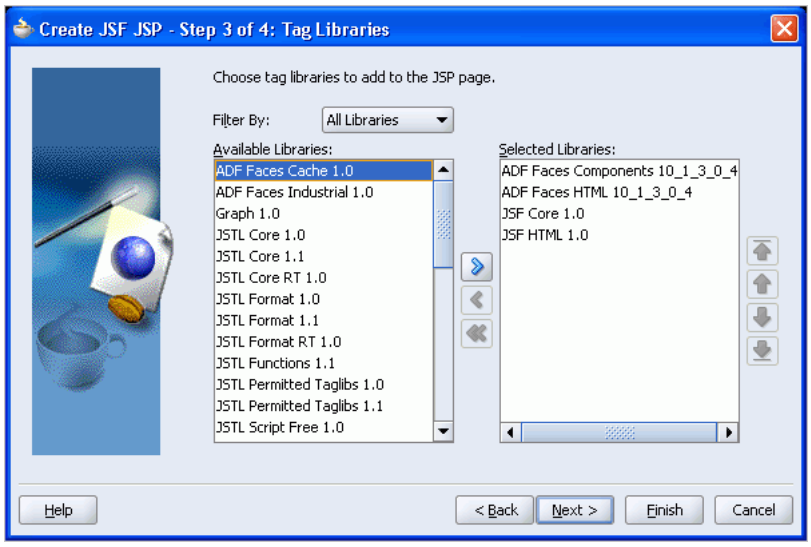
従業員の詳細を表示する JSP の作成

次の手順を使用して、employees.jsp という名前のページを作成します。このページを使用すると、部門および従業員のマスター・ディテール・ページの調整されたセットを参照できるようになります。

1. アプリケーション・ナビゲータで、**view** プロジェクトを選択します。プロジェクトを右クリックし、**「新規」**を選択して新規ギャラリを表示します。
2. 新規ギャラリで、**「Web 層」**カテゴリを開き、**「JSF」**を選択します。**「項目」**リストで**「JSF JSP」**を選択し、**「OK」**をクリックします。
3. 「JSF JSP の作成 - ようこそ」画面で、**「次へ」**をクリックします。

4. 「Web アプリケーション」画面で、デフォルトを受け入れて**「次へ」**をクリックします。
5. 「JSP ファイル」画面で、**「ファイル名」**として `employees.jsp` と入力し、ページ・タイプに**「JSP ページ (*.jsp)」**が選択されていることを確認します。**「次へ」**をクリックします。
6. 「コンポーネント・バインディング」画面で、**「マネージド Bean で UI コンポーネントを自動公開しない」**が選択されていることを確認します。**「次へ」**をクリックします。
7. 「タグ・ライブラリ」画面で、**「ADF Faces Components」**および**「ADF Faces HTML」**が含まれていない場合は、右側の**「選択済のライブラリ」**に追加します。**「JSF Core」**および**「JSF HTML」**タグ・ライブラリも、「選択済のライブラリ」として右側にリストされている必要があります。これを図 7-4 に示します。

図 7-4 JSF JSP のライブラリの選択



8. **「終了」**をクリックします。

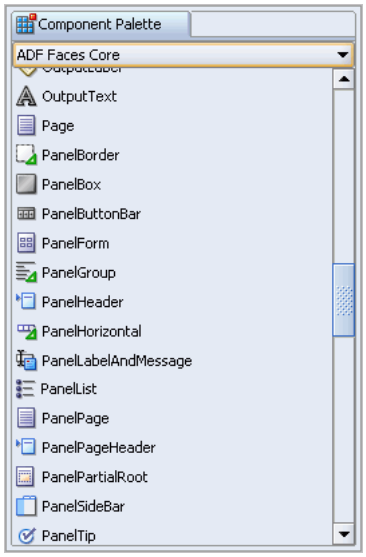
JSP/HTML ビジュアル・エディタに新しい空の `employees.jsp` が表示され、ページの設計を開始する準備が整いました。

ページ・レイアウトおよび見出しの定義

次の手順では、ページに項目を追加してページ・レイアウトを定義します。これまでの章と同様に、コンポーネント・パレットを使用して、タグをページにドロップします。このページでは、ADF Faces PanelPage コンポーネントを使用します。

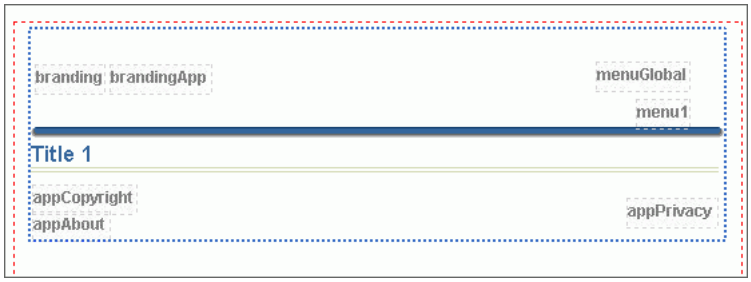
1. コンポーネント・パレットが表示されていない場合は、**「表示」**メニューを使用して表示します。コンポーネントの**「ADF Faces Core」**タブを選択します。**PanelPage** コンポーネントを空のページにドラッグ・アンド・ドロップします。コンポーネント・パレットの「ADF Faces Core」ページを図 7-5 に示します。

図 7-5 コンポーネント・パレットの「ADF Faces Core」



ドロップすると、図 7-6 に示すように、Title 1 というテキストとともに濃い青色の線が表示されます。

図 7-6 ADF Faces PanelPage コンポーネント

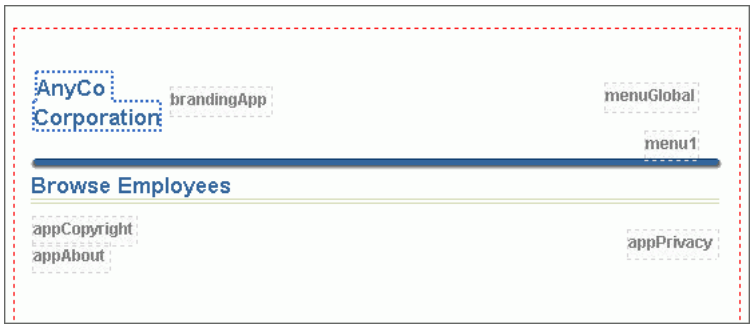


2. ページで、PanelPage コンポーネントをクリックし、プロパティ・インスペクタで**「タイトル」**フィールドの値を `Title 1` から `Browse Employees` に変更して、[Enter] を押します。タイトルの変更がビジュアル・エディタのページに反映されます。
3. この手順で、ページ・タイトルとして使用する別の **ADF Faces** コンポーネントをページに追加します。**PanelPage** の branding セクションにコンポーネントをドラッグします。コンポーネント・パレットの「ADF Faces Core」ページから、**OutputText** コンポーネントをドラッグし、branding ファセット（ページの左上にある branding というタイトルが付いた点線の四角形）にドロップします。
4. 新しい OutputText が outputText1 というデフォルト値で表示されます。プロパティ・インスペクタで OutputText の値を変更して、これを **AnyCo Corporation** に変更します。

5. OutputText の **StyleClass** プロパティを値 **AFHeaderLevelOne** に変更して、テキストのルック・アンド・フィールを変更します。これは、ADF Faces によって定義されているスタイルです。

   変更すると、テキストはビジュアル・エディタで大きく青色で表示されます。図 7-7 に、これらの手順の後のビジュアル・エディタでのページを示します。

図 7-7 テキストが追加された PanelPage コンポーネント



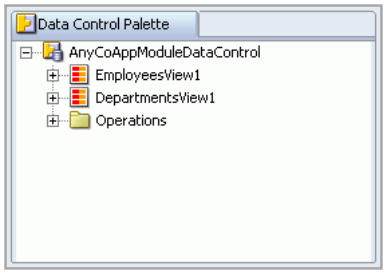
JSP ページでのマスター・データの表示

次の手順では、ページに項目を追加して、ユーザーが部門およびその従業員を表示できるようにします。これらの手順では、データ・コントロール・パレットを使用します。これは、デフォルトでは JDeveloper の表示の右、コンポーネント・パレットの横にあります。データ・コントロール・パレットによって、ADF データ・コントロールという名前のデータ・オブジェクトをページにドロップできます。

最初の段階は、部門、および別の部門とその関連従業員へのナビゲーション・ボタンを表示する読取り専用フォームを追加することです。このことを行うには、model プロジェクトで作成したデータ・コントロールの 1 つをページにドラッグします。

1. **データ・コントロール・パレット**を選択し、**AnyCoAppModule** ノードを開きます。図 7-8 に、AnyCoAppModule が開かれたデータ・コントロール・パレットを示します。

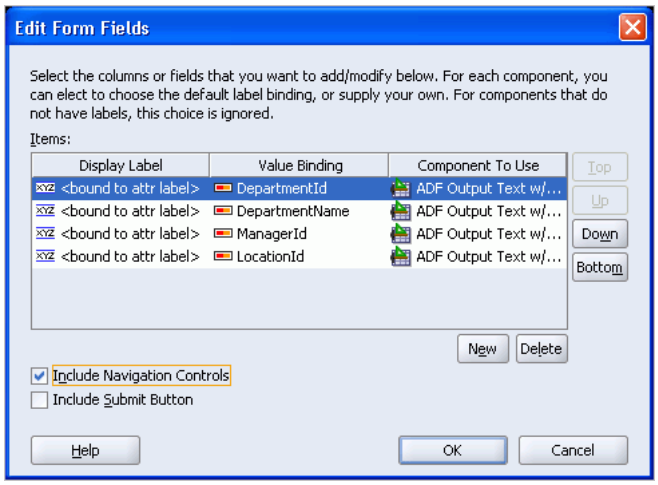
図 7-8 データ・コントロール・パレット



2. データ・コントロール・パレットで、**DepartmentsView1** ノードを選択し、ページの中央の Browse Employees というテキストのすぐ下にドラッグします。
3. ドロップすると、ポップアップ・メニューが表示されます。**「作成」**、**「フォーム」**、**「ADF 読取り専用フォーム」**の順に選択します。

4. 「フォーム・フィールドの編集」ダイアログ・ボックスで、**「ナビゲーション・コントロールを含める」**を選択し、**「OK」**をクリックします。これを図 7-9 に示します。

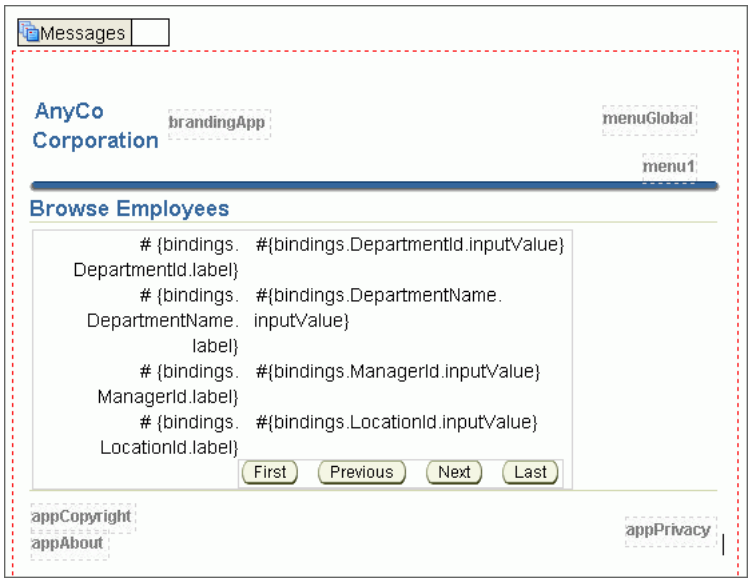
図 7-9 フォーム・フィールドの追加



ナビゲーション・ボタンがページに表示された読取り専用フォームがビジュアル・エディタに表示されます。フォームが表示されない場合は、リフレッシュ・ボタンをクリックします。フォーム・フィールドの値は、#{...} などの式言語を使用して表示されます。これは、ページにレンダリングされるアプリケーション・データを JSF が識別する方法です。

図 7-10 に、これらの手順の後でページがビジュアル・エディタにどのように表示されるかを示します。

図 7-10 ビジュアル・エディタでのフォーム・フィールド



マスター・レコードのディテール・データの表示

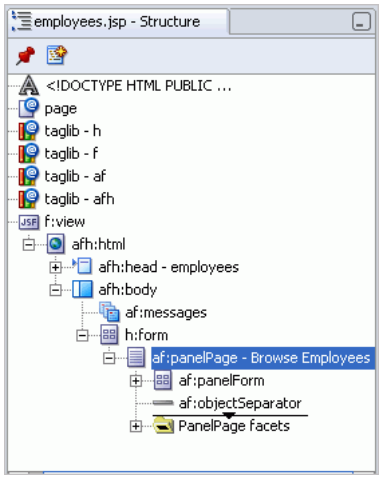
次の手順では、最初に水平のセパレータをページに追加して、部門データと従業員のディテール・データを分割します。次に、各マスター部門の従業員のディテール・データを表示するために、model プロジェクトで作成したビジネス・コンポーネントから作成された別のデータ・コントロールを使用します。従業員データを特定のマスター部門にリンクするデータ・コントロールを選択します。従業員データは ObjectSeparator の下に表示します。

1. ページにセパレータを追加するには、コンポーネント・パレットの「ADF Faces Core」ページから **ObjectSeparator** コンポーネントをドラッグし、ビジュアル・エディタのフォームの下に配置します。水平の点線がフォームの下に表示されます。
2. データ・コントロール・パレットで、**DepartmentsView1** ノードを開きます。

   部門表のフィールド以外に、子の EmployeesView3 ノードもあります。このノードは、部門表への外部キーによって制限された詳細な従業員または従業員セットを表します。
3. **EmployeesView3** ノードをページの水平の点線（ObjectSeparator）の下にドラッグ・アンド・ドロップします。

   ドロップする前にマウスを移動すると、どこにドロップされるかがビジュアル・エディタおよび左下の「構造」ウィンドウに表示されます。図 7-11 に示すように、`af:objectSeparator` の後になるように、ドロップされる位置を「構造」ウィンドウで調整します。

図 7-11 ドロップされる位置を示す「構造」ウィンドウ

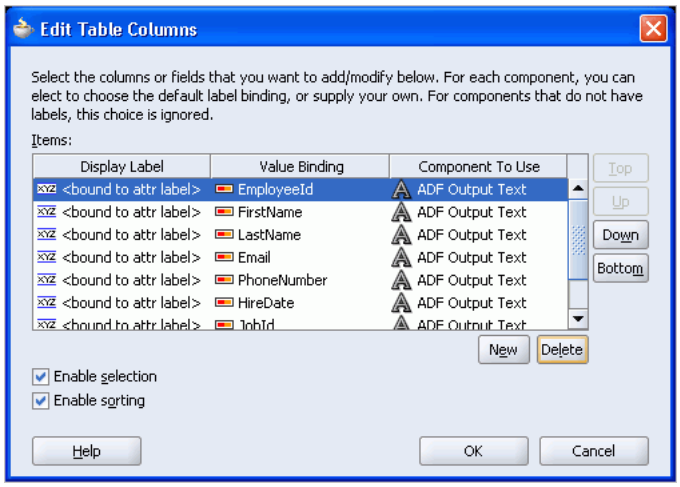


4. ドロップすると、ショートカット・メニューが表示されます。**「作成」**、**「表」**、**「ADF 読取り専用表」**の順に選択します。
5. 「表の列の編集」ダイアログ・ボックスで、**「選択の有効化」**および**「ソートの有効化」**を選択します。

   「OK」をクリックする前に、3 つの列 **DepartmentId**、**ManagerId** および **CommissionPct** を削除します。これらの各行で順番に行を選択し、**「削除」**をクリックします。

   図 7-12 に、「表の列の編集」ダイアログ・ボックスを示します。

図 7-12 表の列の編集

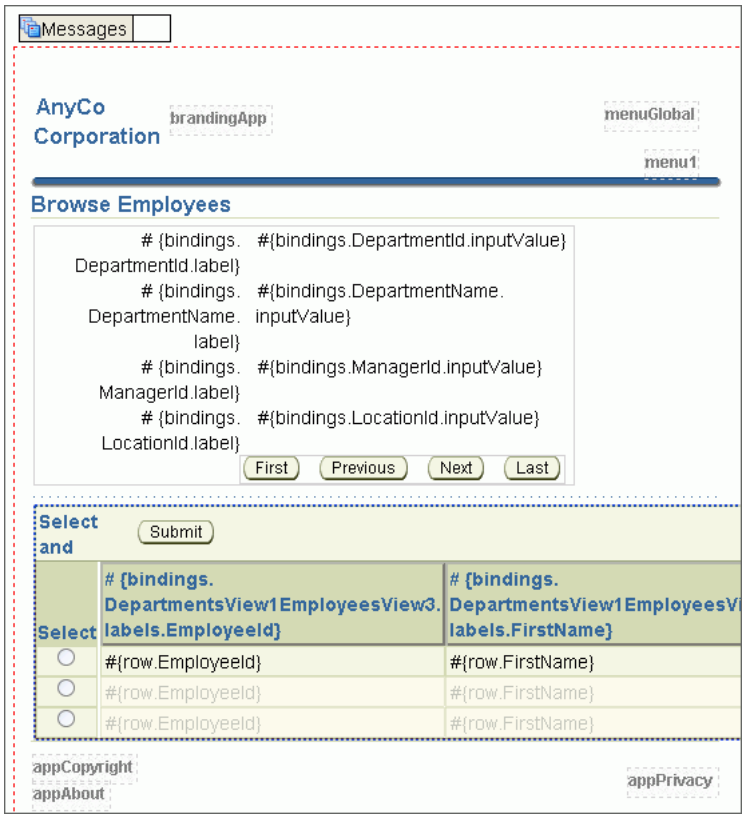


「**OK**」をクリックして表を生成します。

6. 作業内容を保存します。

図 7-13 に示すように、マスター・フォームとディテール表の両方がビジュアル・エディタに表示されます。

図 7-13 ビジュアル・エディタでのマスター・ディテール表示



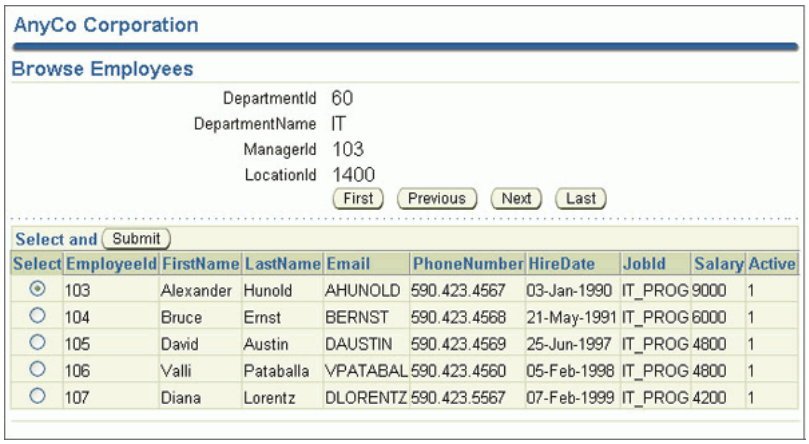
アプリケーションのテスト

部門と従業員をページに追加した後は、ここまでのアプリケーションをテストします。アプリケーションをテストするには、次の手順を実行します。

1. アプリケーション・ナビゲータで、ページ **employees.jsp** を右クリックし、**「実行」**を選択します。JDeveloper に埋め込まれている OC4J サーバーを使用してアプリケーションがローカルで起動されます。アプリケーションが起動すると、従業員のマスター・ディテール・ページがブラウザに表示されます。
2. マスターとディテールの連携をテストするには、部門フォームのナビゲーション・ボタン（**「次へ」**、**「前へ」**、**「先頭へ」**、**「最後へ」**）をクリックします。ナビゲーション・ボタンをクリックすると、関連する従業員が下の表に表示されます。

   ブラウザに表示されるページは、図 7-14 のようになります。

図 7-14 ブラウザに表示された従業員データ



3. テストを終了するには、サーバーを停止します。これを行うには、JDeveloper で、「実行中 : 埋込み OC4J サーバー - ログ」というログ・ウィンドウを右クリックし、**「終了」**を選択します。

アプリケーション・ページ間のナビゲーション : JSF ナビゲーション・ダイアグラム

JSF ナビゲーション・ダイアグラムを使用して、アプリケーションを計画できます。このダイアグラムには、アプリケーション・ページおよびそれらの間のナビゲーション・ケースが表示されます。

ダイアグラムから新しいページを直接作成し、それらの間のナビゲーションを定義できます。コンポーネント・パレットからドロップした要素を使用します。

その後、ビジュアル・エディタなどのツールを使用して JSF ページを編集したり、ダイアグラムからナビゲーション・ケースを直接編集することができます。

JSF ナビゲーション・ダイアグラムを使用したページの作成

ユーザーが従業員データを編集できるようにするために、新しいページ `edit.jsp` を作成します。このページに、データ・コントロール・パレットから ADF 入力フォームをドロップします。

次の手順では、JSF JSP ウィザードを使用して新しいページを直接作成するかわりに、JSF ナビゲーション・ダイアグラムから新しいページを作成します。後の手順では、ページ間のナビゲーションもダイアグラムでビジュアルに定義します。

1. アプリケーションの JSF ナビゲーション・ダイアグラムを開くには、アプリケーション・ナビゲータで、**view** プロジェクトを右クリックし、**「JSF ナビゲーションを開く」**を選択します。

   空のナビゲーション・ダイアグラムが表示されます。

2. ナビゲーション・ルールの定義を開始するには、既存の **employees.jsp** ページをアプリケーション・ナビゲータから空のダイアグラムにドラッグします。

   `employees.jsp` ページを表すページ・アイコンがナビゲーション・ダイアグラムに表示されます。

3. 新しいページを作成するには、JSF ナビゲーション・ダイアグラムを編集ウィンドウに表示したまま、コンポーネント・パレットの「JSF ナビゲーション・ダイアグラム」ページから、**「JSF ページ」**をダイアグラムにドラッグします。

   新しいページ・アイコンがダイアグラムに表示されたら、テキスト `edit.jsp` を入力して名前 `/untitled1.jsp` を置き換え、**[Enter]** を押します。（名前の前にスラッシュを追加する必要はありません。自動的に追加されます。）

   この時点で新しいページがダイアグラムに追加されましたが、ページ・ファイル自体は存在しません。このことを示すために、ページ・アイコンには黄色の注意記号が付いています。

4. `edit.jsp` ページを作成するには、ダイアグラムでページ `edit.jsp` のページ・アイコンをダブルクリックします。

5. 「JSF JSP の作成」ウィザードで、「ようこそ」ページが表示されたら、**「次へ」**をクリックします。ページの名前はすでに入力されており、残りの手順は以前に作成した `employees.jsp` ページと同じなので、**「終了」**をクリックします。

   ビジュアル・エディタに新しいページが表示され、ページを設計する準備が整いました。

ページ間のナビゲート

ユーザーによる JSF アプリケーションのナビゲーションは、ユーザーがリンクをクリックしたときに次に表示されるページを決定するナビゲーション・ルールを使用して定義されます。ページ上の様々なリンクなど、様々なケースがナビゲーション・ケースとして定義されます。

ダイアグラムを使用して、ページ間のフローを表すアプリケーションのページ間のナビゲーションを描くことができます。

JSF ナビゲーション・ダイアグラムでナビゲーションを描くと、2 つのことが起こります。必要な構成ファイルにナビゲーション・ケースが自動的に追加されます。また、ページをつなぐ矢印として、ナビゲーション・ケースがダイアグラムに表示されます。矢印の向きは、ユーザーのナビゲート元およびナビゲート先のページを示します。

1 つのページからの異なるナビゲーション・ケースを区別するために、各ナビゲーション・ケースにラベルが関連付けられます。

ページ間のナビゲーションの定義

次の手順では、アプリケーションの 2 つのページ間のナビゲーションを可能にするために、アプリケーションにナビゲーションの詳細を追加します。

1. JSF ナビゲーション・ダイアグラムに戻り、2 つのページ間のナビゲーションを定義します。これには、編集ウィンドウで開かれている項目の中から、**faces-config.xml** のタブを選択します。ダイアグラムが開かれていない場合は、前に行ったように開きます。

2. ダイアグラムで **employees.jsp** ページ・アイコンをクリックして、このページにフォーカスします。

3. コンポーネント・パレットの「JSF ナビゲーション・ダイアグラム」ページで、**「JSF ナビゲーション・ケース」**をクリックします（ドラッグではありません）。

4. ダイアグラムで、**employee.jsp** アイコンを再度クリックします。ページをクリックした後、マウスをページから移動し始めると、**employees.jsp** ページにつながる線が表示されます。**edit.jsp** ページをクリックして、この線を新しいページにつなぎます。

   2 つのページが線でつながれました。これは、`employees.jsp` ページから `edit.jsp` ページにナビゲートするナビゲーション・ケースを表します。「成功」というデフォルト値のラベルが付いています。

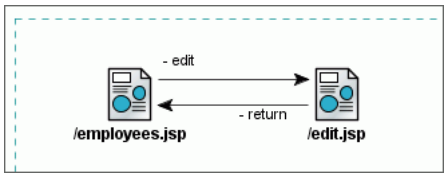
5. 値を変更するには、**「成功」**をクリックし、**edit** と入力し、**[Enter]** キーを押します。edit というテキストが、線のラベルとしてダイアグラムに表示されます。

6. **edit.jsp** ページから **employees.jsp** ページにナビゲートする別のナビゲーション・ケースを追加し、**return** という名前を付けます。このナビゲーション・ケースは、`edit.jsp` ページから `employees.jsp` ページへ戻るナビゲーションを表します。

7. 作業内容を保存します。

ナビゲーション・ダイアグラムには、両方のナビゲーション・ケースが表示されます。1 つは、ユーザーが従業員の詳細を編集できるように `employees.jsp` ページから `edit.jsp` ページへナビゲートするためのナビゲーション・ケースで、もう 1 つは、ユーザーが `employees.jsp` ページに戻るためのナビゲーション・ケースです。図 7-15 に、`employees.jsp` ページおよび `edit.jsp` ページのナビゲーション・ダイアグラムを示します。

図 7-15 ナビゲーション・ダイアグラム



データの編集

ユーザーが従業員データを編集できる編集ページを作成するには、ADF Faces コンポーネントを使用して、従業員ページとまったく同じ方法でページをレイアウトします。

従業員データには、データ・コントロールが使用されています。特定の部門の特定の従業員の正しい従業員データを表示するために、従業員ページで使用した部門データ・コントロールの子データ・コントロールとして従業員データを提供できるデータ・コントロールがあります。このデータ・コントロールによって、従業員データの正しいセットが提供されます。

ユーザーがデータを編集できるように、表ではなく ADF フォームにデータを表示します。

編集フォームの作成

次の手順では、`edit.jsp` ページを作成します。

1. ビジュアル・エディタで `edit.jsp` ページを開きます。

2. `edit.jsp` ページで、コンポーネント・パレットの「ADF Faces Core」ページから、**PanelPage** コンポーネントをページに追加します。PanelPage のタイトルを **Edit Employee** に変更します。

3. また、**OutputText** コンポーネントをページ上部の branding ファセットに追加します。**「値」**プロパティを **AnyCo Corporation** に、**「styleClass」**プロパティを **AFHeaderLevelOne** に設定します。

4. データ・コントロール・パレットで、**DepartmentsView1** ノードを開き（閉じられている場合）、その下にある同じ **EmployeesView3** ノードを確認します。これは、次に示すように、DepartmentsView1 の子です。

5. **EmployeesView3** ノードをパネルの中央の Edit Employee というタイトルの下にドラッグします。表示されるダイアログ・ボックスで、**「作成」**、**「フォーム」**および「ADF フォーム」を選択します。

注意： データ・コントロール・パレットから、親 DepartmentsView1 の下にある子 EmployeesView3 ノードを選択することが重要です。子 EmployeesView3 には、外部キー関係で指定されたように、特定の親 Deparment の従業員のディテール・レコードが表示されます。

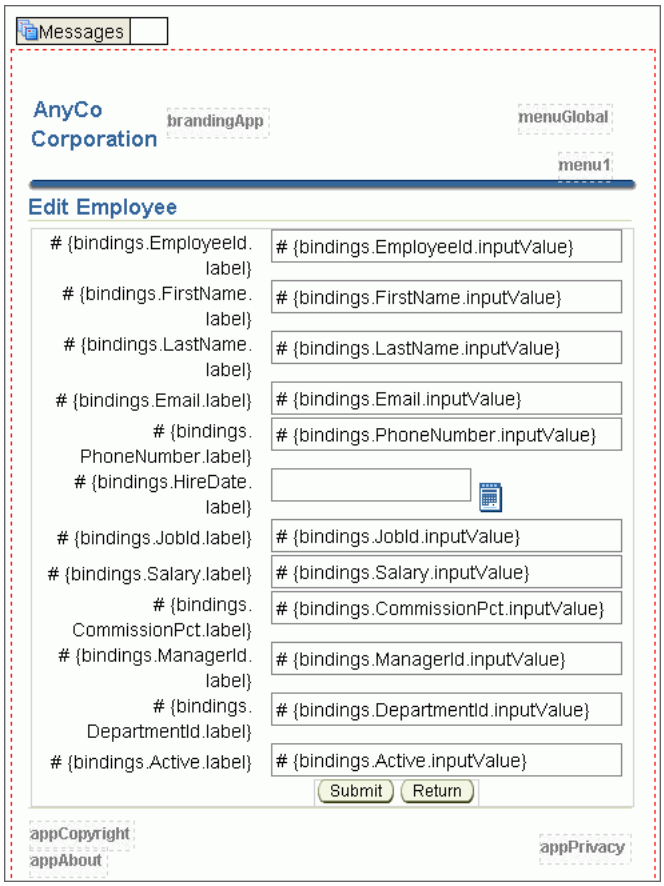
6. 「フォーム・フィールドの編集」ダイアログ・ボックスで、**「送信ボタンを含める」**を選択し、**「OK」**をクリックします。

編集フォームはほぼ完成です。最後に、ユーザーが employees.jsp ページに戻るためのボタンを追加します。これを行うには、次の手順を実行します。

1. コンポーネント・パレットの「ADF Faces Core」ページから、**CommandButton** をページにドラッグし、ページ下部にある**「発行」**ボタンの横に配置します。
2. プロパティ・インスペクタを使用して、ボタンの**「テキスト」**プロパティを **commandButton 1** から **Return** に変更します。
3. 再度プロパティ・インスペクタを使用して、CommandButton の**「アクション」**プロパティを **Return** に変更します。「アクション」フィールドの矢印を使用して、値 return を選択できます。この値は、edit.jsp ページから employees.jsp ページにユーザーが戻るナビゲーション・ケースで入力した値です。
4. ページを保存します。

編集ページが完成しました。図 7-16 に示すようなページになります。

図 7-16 ビジュアル・エディタでの従業員の詳細フォームの編集



編集ページへのナビゲート

残りの処理は、従業員ページを表示したユーザーが編集ページにナビゲートして従業員の詳細を編集できるようにすることです。これを行うには、ナビゲーション・ダイアグラムで定義した edit という名前のナビゲーション・ケースを使用します。従業員ページの従業員表にはすでにボタンがあり、ユーザーが編集ページにナビゲートするために使用できます。

1. ビジュアル・エディタで `employees.jsp` ページを開きます。
2. **「発行」**ボタンを選択してフォーカスします。
3. プロパティ・インスペクタで、**「テキスト」**プロパティを **Edit** に変更します。「アクション」プロパティを edit に設定します。**「アクション」**フィールドの矢印を使用して、値 **edit** を選択します。この値は、`employees.jsp` ページから `edit.jsp` ページにユーザーがナビゲートするナビゲーション・ケースとしてナビゲーション・ダイアグラムで指定した値です。

これで、ユーザーがこのボタンをクリックすると、選択したレコードが表示された編集ページにナビゲートされます。

COMMIT および ROLLBACK の有効化

ユーザーが従業員の詳細に対して行った編集をコミットできるように、またはロールバックしてそれらを破棄できるように、コミットおよびロールバック機能を有効にするには、`employees.jsp` ページの下部に「コミット」ボタンおよび「ロールバック」ボタンを追加します。

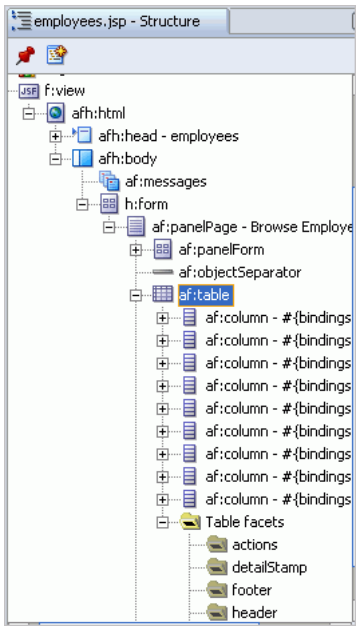
この手順でもファセットを使用します。この場合、フッター・ファセットを表に追加し、「コミット」ボタンおよび「ロールバック」ボタンをこのフッター・ファセットに追加します。

次の手順でフッター・ファセットを追加するには、「構造」ウィンドウを使用します。

1. ビジュアル・エディタに `employees.jsp` が表示されている状態で、「構造」ウィンドウで表コンポーネント（**af:table**）を開きます。構造内には、Table Facets フォルダがあります。
2. **Table Facets** フォルダを開きます。ADF Faces 表コンポーネントでサポートされる様々なファセット用の複数のサブフォルダがあります。

   図 7-17 に示すように、フッター・ファセットは現在空のため灰色です。

図 7-17 空のフッター・ファセット

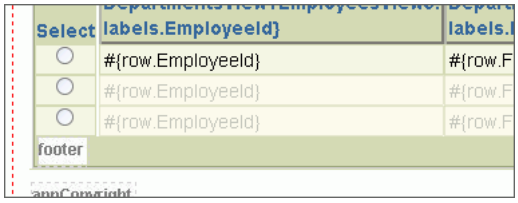


「コミット」ボタンおよび「ロールバック」ボタンを保持できるように、フッター・ファセットをこのプレースホルダに挿入します。

3. 「構造」ウィンドウで、フッター・ファセット・ノードを右クリックし、**「ファセット - 表」**、**「フッター」**の順に選択します。

   図 7-18 に示すように、フッター・ファセットがページの表の一番下に追加されます。

図 7-18 フッター・ファセットの挿入



これで、JSF コンポーネントをフッター・ファセットに配置できます。

必要な「コミット」ボタンおよび「ロールバック」ボタンは、データ・コントロール・パレットの Operations ノード内にあります。このノードは、親 AnyCoAppModuleDataControl の直下の子です。

ページにボタンを追加するには、次の手順を実行します。

1. データ・コントロール・パレットで、**AnyCoAppModuleDataControl** フォルダ、**Operations** フォルダの順に開きます。Operations フォルダ内に、「コミット」および「ロールバック」ノードがあります。

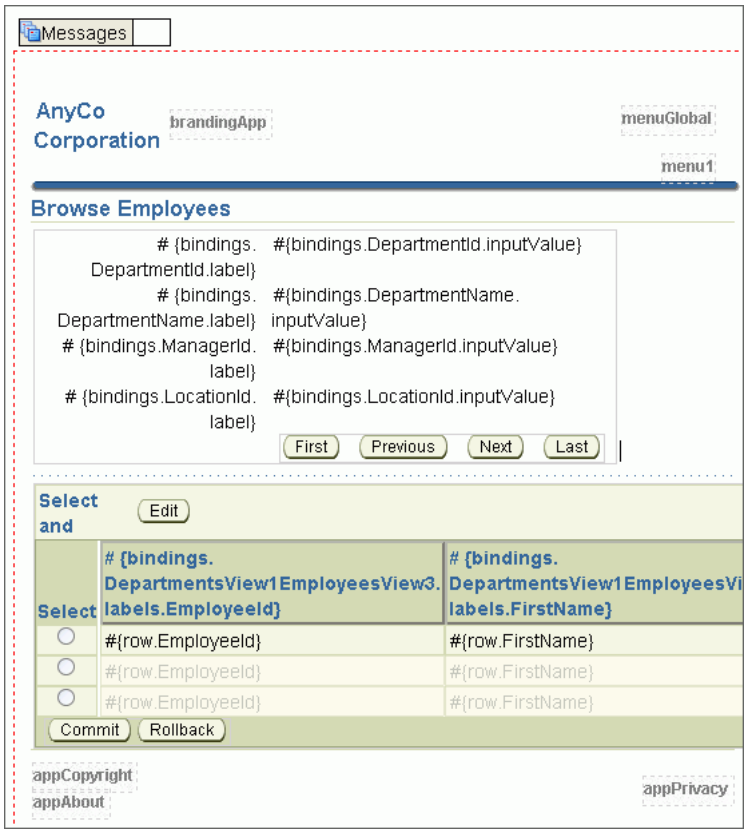
2. **「コミット」**および**「ロールバック」**ノードそれぞれを、表の一番下のフッター・ファセットにドラッグします。

   それぞれの処理をデータ・コントロール・パレットからフッター・ファセットにドロップすると、ダイアログ・ボックスが表示されます。**「ADF コマンド・ボタンの作成」**を選択します。

3. 作業内容をすべて保存します。

これでアプリケーションが完成しました。最終的な employees.jsp ページは、図 7-19 に示すようになります。

図 7-19 ビジュアル・エディタでの最終的なマスター・ディテール・アプリケーション



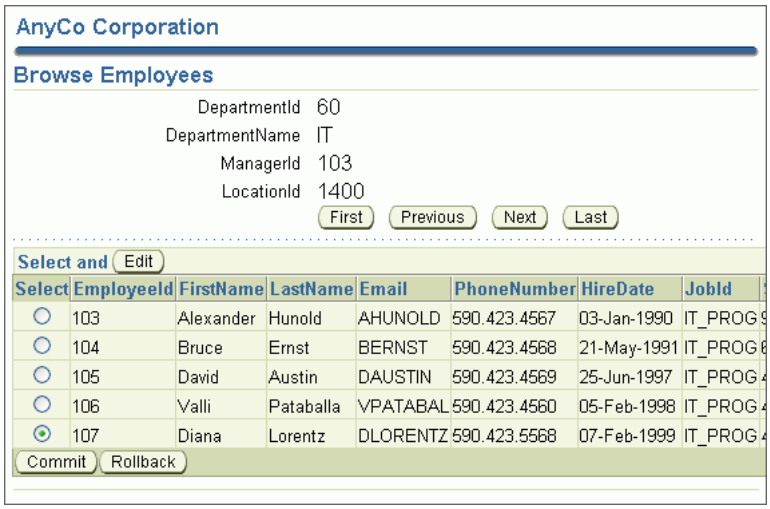
アプリケーションの実行

次のようにアプリケーションを実行できます。

1. アプリケーション・ナビゲータで、employees.jsp を右クリックし、ショートカット・メニューで「**実行**」を選択します。
2. アプリケーションを実行すると、異なる部門にナビゲートし、個々の従業員を選択して編集できるようになります。従業員の給与または入社日を更新してみます。

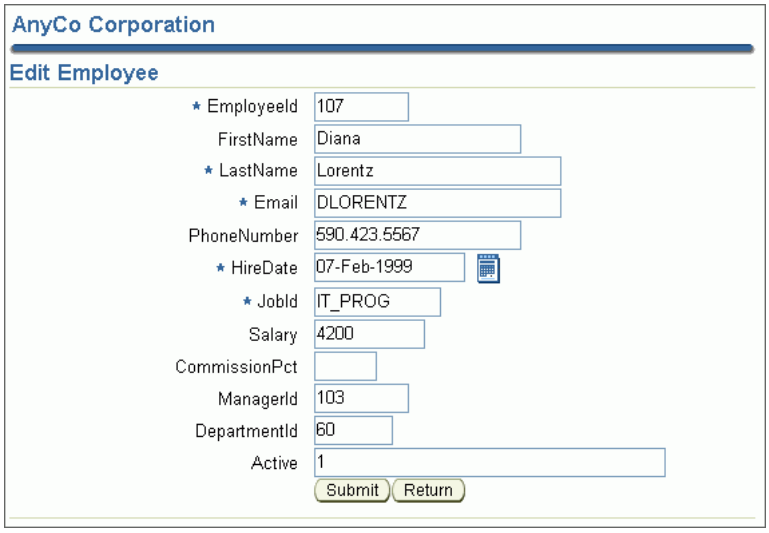
ブラウザに表示される従業員ページを図 7-20 に示します。

図 7-20 ブラウザに表示されたマスター・ディテール・アプリケーション



ブラウザに表示される編集ページを図 7-21 に示します。

図 7-21 マスター・ディテール・アプリケーションの内容の編集



8

Oracle Database からの切断

JDeveloper でのデータベースからの切断は簡単な操作ですが、Java アプリケーションでは単独のプロセスではありません。アプリケーションでは、ResultSet、Statement および Connection オブジェクトを使用した後、それらをすべて明示的にクローズする必要があります。Connection オブジェクトをクローズすると、データベースから切断されます。close メソッドによって、メモリーがクリーンアップされ、データベース・カーソルが解放されます。そのため、ResultSet および Statement オブジェクトを明示的にクローズしないと、重大なメモリー・リークが発生する場合があり、データベース内のカーソルを使い果たすことがあります。その後、接続をクローズする必要があります。

この章は次の項で構成されています。

オープンしたすべてのオブジェクトをクローズするメソッドの作成

次の手順では、closeAll メソッドを DataHandler クラスに追加します。

1. アプリケーション・ナビゲータで DataHandler.java をダブルクリックして、Java ソース・エディタで開きます。
2. DataHandler クラスの最後で、次のように closeAll メソッドを宣言します。

```
public void closeAll() {

}
```

3. メソッド本体で、次のように ResultSet オブジェクトがオープンしているかどうかを確認します。

```
if ( rset != null ) {
```

4. オープンしている場合は、次のようにクローズして例外を処理します。

```
  try { rset.close(); } catch ( Exception ex ) {}
  rset = null;
}
```

5. Statement オブジェクトに同じ操作を繰り返します。

```
if ( stmt != null ) {
  try { stmt.close(); } catch ( Exception ex ) {}
  stmt = null;
}
```

6. 最後に、Connection オブジェクトをクローズします。

```
if ( conn != null ) {
  try { conn.close(); } catch ( Exception ex ) {}
  conn = null;
}
```

アプリケーションでのオープンしたオブジェクトのクローズ

ResultSet、Statement および Connection オブジェクトは、使い終わった後クローズする必要があります。DataHandler クラスでは、挿入、更新および削除メソッドでは、戻る前にこれらのオブジェクトをクローズする必要があります。問合せメソッドでは、employees.jsp ページが問合せによって返された行の処理を終了するまで、これらのオブジェクトをクローズできないことに注意してください。

次の手順では、DataHandler.java ファイルで closeAll メソッドへの適切なコールを追加します。

1. Java ソース・エディタで DataHandler.java を開きます。
2. addEmployee メソッドの最後で、catch ブロックの閉じカッコの後に、finally ブロックでの closeAll メソッドへのコールを追加します。

```
finally {
  closeAll();
}
```

3. 同じコールを addEmployeeSP、deleteEmployeeById、findEmployeeById、updateEmployee および authenticateUser メソッドに追加します。

4. ビジュアル・エディタで employees.jsp ファイルを開きます。Employees 表内のスクリプトレットを検索し、ダブルクリックして「スクリプトレットの挿入」ダイアログ・ボックスを開きます。
5. 次の文を while ループの後に追加します。

```
empsbean.closeAll();
```

6. 作業内容を保存し、アプリケーションをコンパイルおよび実行して、すべて正しく動作することを確認します。

9

グローバル・アプリケーションの構築

様々なロケールをサポートするグローバル・インターネット・アプリケーションを構築するには、適切な開発の実施が必要です。ロケールとは、各国の言語およびその言語が話される地域のことです。アプリケーション自体で、ユーザーのロケール設定を認識し、ユーザーが期待する文化的な慣習に従って、内容を表示する必要があります。正しい日付書式や数値書式を使用するなど、適切なロケール特性でデータを表示することが重要です。Oracle Database は完全に国際化されており、グローバル・アプリケーションを開発およびデプロイするためのグローバル・プラットフォームを提供します。

この章では、Java および Oracle Database 環境でのグローバル・アプリケーション開発について説明します。グローバル・インターネット・アプリケーションの開発およびデプロイに関連する基本タスク、すなわちロケール認識の開発、ユーザー設定言語での HTML コンテンツの構築、ユーザー・ロケールの文化的な慣習に従ったデータの表示などについて説明します。

この章では、次の項目について説明します。

ロケール認識の開発

グローバル・インターネット・アプリケーションでは、ユーザー・ロケールを認識する必要があります。日付、時間、通貨の書式設定など、ロケールに依存する機能は Java や SQL などのプログラミング環境に組み込まれています。アプリケーションではロケールに依存する機能を使用して、ユーザー・ロケールの文化的な慣習に従って、HTML ページを書式設定できます。

プログラミング環境によって、ロケールを表す方法は異なります。たとえば、フランス語（カナダ）のロケールは、次のように表されます。

環境	表記	ロケール	説明
Java	Java ロケール・オブジェクト	`fr_CA`	Java では、ISO の言語コードおよび国コードを使用します。 `fr` は ISO 639 規格で定義された言語コードです。`CA` は ISO 3166 規格で定義された国コードです。
SQL および PL/SQL	`NLS_LANGUAGE` および `NLS_TERRITORY` パラメータ	`NLS_LANGUAGE ="CANADIAN FRENCH"` `NLS_TERRITORY ="CANADA"`	**関連項目：**『Oracle Database 2 日で開発者ガイド』の第 6 章「グローバル環境での作業」

表 9-1 に、一般的に使用されるロケールの Java および Oracle 環境での定義を示します。

表 9-1 Java、SQL および PL/SQL プログラミング環境でのロケール表記

ロケール	Java	NLS_LANGUAGE、NLS_TERRITORY
中国語（中国）	`zh_CN`	`SIMPLIFIED CHINESE、CHINA`
中国語（台湾）	`zh_TW`	`TRADITIONAL CHINESE、TAIWAN`
英語（アメリカ）	`en_US`	`AMERICAN、AMERICA`
英語（イギリス）	`en_GB`	`ENGLISH、UNITED KINGDOM`
フランス語（カナダ）	`fr_CA`	`CANADIAN FRENCH、CANADA`
フランス語（フランス）	`fr_FR`	`FRENCH、FRANCE`
ドイツ語（ドイツ）	`de_DE`	`GERMAN、GERMANY`
イタリア語（イタリア）	`it_IT`	`ITALIAN、ITALY`
日本語（日本）	`ja_JP`	`JAPANESE、JAPAN`
韓国語（韓国）	`ko_KR`	`KOREAN、KOREA`
ポルトガル語（ブラジル）	`pt_BR`	`BRAZILIAN PORTUGUESE、BRAZIL`
ポルトガル語（ポルトガル）	`pt_PT`	`PORTUGUESE、PORTUGAL`
スペイン語（スペイン）	`es_ES`	`SPANISH、SPAIN`

異なるプログラミング環境をまたいでグローバル・アプリケーションを記述する場合、ユーザーのロケール設定を環境間で同期化する必要があります。たとえば、PL/SQL プロシージャをコールする Java アプリケーションでは、Java ロケールを対応する `NLS_LANGUAGE` および `NLS_TERRITORY` 値にマッピングし、ユーザー・ロケールが一致するようにパラメータ値を変更してから、PL/SQL プロシージャをコールする必要があります。

Oracle と Java ロケール間のマッピング

Oracle Globalization Development Kit（GDK）では、LocaleMapper クラスが提供されています。Java、IANA、ISO、Oracle 間で、等価のロケールおよびキャラクタ・セットがマッピングされます。Java アプリケーションが、Oracle ロケール名で指定されたロケール情報をクライアントから受け取る場合があります。Java アプリケーションで情報を正しく処理するには、等価の Java ロケールにマッピングできる必要があります。

例 9-1 に、LocaleMapper クラスの使用方法を示します。

例 9-1 Java ロケールから Oracle の言語および地域へのマッピング

```
Locale locale = new Locale("fr", "CA");
String oraLang = LocaleMapper.getOraLanguage(locale);
String oraTerr = LocaleMapper.getOraTerritory(locale);
```

GDK は、Oracle アプリケーション開発者にグローバル・インターネット・アプリケーションの開発フレームワークを提供する、一連の Java Application Program Interface（API）です。GDK によって、Java の既存のグローバリゼーション機能が補完されます。中間層 Java アプリケーションと Oracle Database サーバー間のロケール動作が同期化されます。

> **関連項目：** GDK の詳細は、次の URL を参照してください。
>
> http://www.oracle.com/technology/tech/globalization/gdk/index.html

ユーザー・ロケールの判別

グローバル環境では、ロケール設定が異なるユーザーをアプリケーションで受け入れる必要がある場合があります。ユーザーの優先ロケールを判別します。優先ロケールがわかったら、アプリケーションはそのロケールの言語を使用して HTML コンテンツを構築し、そのロケールの文化的な慣習に従う必要があります。

ユーザー・ロケールを判別する最も一般的な方法の 1 つは、ユーザーのブラウザのデフォルトの ISO ロケール設定に基づきます。通常、ブラウザはロケール設定を Accept-Language HTTP ヘッダーを使用して HTTP サーバーに送信します。このヘッダーが NULL に設定されている場合、使用できるロケール設定情報がないため、アプリケーションは、理論上は事前定義されたアプリケーションのデフォルト・ロケールにフォールバックする必要があります。

JSP ページと Java サーブレットの両方で、例 9-2 に示すように、サーブレット API へのコールを使用して Accept-Language HTTP ヘッダーを取得できます。

例 9-2 Java での Accept-Language ヘッダーを使用したユーザー・ロケールの判別

```
String lang = request.getHeader("Accept-Language")
StringTokenizer st = new StringTokenizer(lang, ",")
if (st.hasMoreTokens()) userLocale = st.nextToken();
```

このコードは Accept-Language ヘッダーを HTTP リクエストから取得し、最初の ISO ロケールを抽出した後、その ISO ロケールをユーザー設定ロケールとして使用します。

Java アプリケーションでのロケール認識

Java ロケール・オブジェクトは、対応するユーザーのロケールを Java で表します。ロケールに使用される Java エンコーディングは、Java 文字列とバイト・データ間で正しく変換する必要があります。Java コードでユーザー・ロケールを認識する場合は、ロケールの Java エンコーディングを考慮する必要があります。Java メソッドで Java ロケールおよびエンコーディングを認識させるには、次の 2 つの方法があります。

- メソッドのデフォルトの Java ロケールおよびデフォルトの Java エンコーディングを使用する
- メソッドの Java ロケールおよび Java エンコーディングを明示的に指定する

グローバル・アプリケーションを開発する場合は、2 番目の方法を採用し、現在のユーザー・ロケールに対応する Java ロケールおよび Java エンコーディングを明示的に指定することをお薦めします。例 9-3 に示すように、getDateTimeInstance メソッドで、ユーザー・ロケール（user_locale）に対応する Java ロケール・オブジェクトを指定できます。

例 9-3 Java でのユーザー・ロケールの明示的な指定

```
DateFormat df = DateFormat.getDateTimeInstance(DateFormat.FULL, DateFormat.FULL, user_
locale);
dateString = df.format(date); /* Format a date */
```

HTML ページのエンコーディング

HTML ページのエンコーディングは、ブラウザおよびインターネット・アプリケーションにとって重要な情報です。ページ・エンコーディングは、インターネット・アプリケーションがサービスを提供するロケールに使用されるキャラクタ・セットです。ブラウザでは、正しいフォントおよびキャラクタ・セット・マッピング表を使用して HTML ページを表示できるように、ページ・エンコーディングを認識する必要があります。インターネット・アプリケーションでは、HTML フォームからの入力を処理できるように、HTML ページ・エンコーディングを認識する必要があります。

異なるロケールに異なるネイティブ・エンコーディングを使用するかわりに、すべてのページ・エンコーディングに UTF-8（Unicode エンコーディング）を使用することをお薦めします。UTF-8 エンコーディングを使用すると、グローバル・アプリケーションのコーディングが簡単になるだけでなく、単一ページで多言語のコンテンツを配置できるようになります。

この項では、次の項目について説明します。

- HTML ページのページ・エンコーディングの指定
- Java サーブレットおよび JSP ページでのページ・エンコーディングの指定

HTML ページのページ・エンコーディングの指定

HTML ページのエンコーディングを指定するには、2 つの方法があります。1 つは HTTP ヘッダーで指定する方法、もう 1 つは HTML ページ・ヘッダーで指定する方法です。

HTTP ヘッダーでのエンコーディングの指定

Content-Type HTTP ヘッダーを HTTP 指定に含めます。例 9-4 に示すように、コンテンツ・タイプおよびキャラクタ・セットを指定します。

例 9-4 HTTP 指定でのページ・エンコーディングの指定

```
Content-Type: text/html; charset=utf-8
```

charset パラメータでは、HTML ページのエンコーディングを指定します。charset パラメータに指定できる値は、ブラウザがサポートする文字エンコーディングの IANA 名です。

HTML ページ・ヘッダーでのエンコーディングの指定

この方法は、主に静的な HTML ページに対して使用します。例 9-5 に示すように、HTML ヘッダーで文字エンコーディングを指定します。

例 9-5 HTML ページでのページ・エンコーディングの指定

```
<meta http-equiv="Content-Type" content="text/html;charset=utf-8">
```

charset パラメータでは、HTML ページのエンコーディングを指定します。Content-Type HTTP ヘッダーと同様に、charset パラメータに指定できる値は、ブラウザがサポートする文字エンコーディングの IANA 名です。

Java サーブレットおよび JSP ページでのページ・エンコーディングの指定

contentType ページ・ディレクティブを使用して、JavaServer Pages（JSP）ファイルの Content-Type HTTP ヘッダーで HTML ページのエンコーディングを指定できます。次に例を示します。

```
<%@ page contentType="text/html; charset=utf-8" %>
```

このエンコーディングは、クライアントへの応答用に JSP ファイルが使用する MIME タイプおよび文字エンコーディングです。JSP コンテナに有効な任意の MIME タイプまたは IANA キャラクタ・セット名を使用できます。デフォルトの MIME タイプは text/html であり、デフォルトのキャラクタ・セットは ISO-8859-1 です。前述の例では、キャラクタ・セットは UTF-8 に設定されています。contentType ページ・ディレクティブのキャラクタ・セットによって、JSP エンジンに対し、動的な HTML ページをエンコードし、指定されたキャラクタ・セットで HTTP Content-Type ヘッダーを設定するように指示されます。

Java サーブレットの場合、Servlet API の setContentType メソッドをコールして、HTTP ヘッダーでページ・エンコーディングを指定できます。例 9-6 の doGet 関数は、このメソッドのコール方法を示しています。

例 9-6 setContentType を使用したサーブレットでのページ・エンコーディングの指定

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{

// generate the MIME type and character set header
response.setContentType("text/html; charset=utf-8");

...

// generate the HTML page
Printwriter out = response.getWriter();
out.println("<HTML>");

...

out.println("</HTML>");
}
```

getWriter メソッドの前に setContentType メソッドをコールする必要があります。これは、getWriter メソッドによって、setContentType メソッド・コールで指定されたキャラクタ・セットを使用する出力ストリーム・ライターが初期化されるためです。ライターに書き込まれ、最終的にはブラウザに書き込まれる HTML コンテンツは、setContentType コールによって指定されたエンコーディングでエンコードされます。

HTML ページのコンテンツを翻訳するための構成

ユーザー・インタフェースをユーザーのローカル言語で表示できるようにすることは、アプリケーションのグローバル化に関する基本タスクの 1 つです。HTML ページのコンテンツの翻訳可能なソースは、次のカテゴリに分類されます。

- アプリケーション・コード内にハードコードされたテキスト文字列
- 静的な HTML ファイル、イメージ・ファイル、テンプレート・ファイル（CSS など）
- データベースに格納された動的データ

この項では、翻訳可能なコンテンツの外部化について説明します。内容は次のとおりです。

- Java サーブレットおよび JSP ページの文字列
- 静的ファイル
- データベースからのデータ

Java サーブレットおよび JSP ページの文字列

Java サーブレットおよび JSP ページの翻訳可能な文字列を Java リソース・バンドルに外部化する必要があります。これにより、これらのリソース・バンドルを Java コードとは別に翻訳できます。翻訳後、リソース・バンドルは英語バンドルと同じベース・クラス名を持ちますが、接尾辞として Java ロケール名が付きます。Java リソース・バンドルの参照メカニズムが正しく機能するように、バンドルは英語リソース・バンドルと同じディレクトリ内に配置する必要があります。

> **関連項目：** Java リソース・バンドルに関する Sun 社のドキュメントを参照してください。
>
> `http://java.sun.com/j2se/1.4.2/docs/api/java/util/ResourceBundle.html`

多言語アプリケーションではユーザー・ロケールは固定ではないため、ユーザー・ロケールに対応する Java ロケール・オブジェクトを明示的に指定し、`getBundle` メソッドをコールする必要があります。次の例では、Java ロケール・オブジェクトは `user_locale` です。

```
ResourceBundle rb = ResourceBundle.getBundle("resource", user_locale);
String helloStr = rb.getString("hello");
```

前述のコードは、ユーザー設定ロケールに対応するリソース・バンドルから、テキスト文字列 `hello` のローカライズされたバージョンを取得します。

> **関連項目：** Java でのリソース・バンドルの作成の詳細は、9-9 ページの「JDeveloper での JSP ページのテキストのローカライズ」を参照してください。

静的ファイル

HTML や GIF などの静的ファイルは、すぐに翻訳できます。これらのファイルを翻訳するときは、UTF-8 をファイル・エンコーディングとして、対応する言語に翻訳する必要があります。翻訳されたファイルの言語を区別する場合、異なる言語の静的ファイルは異なるディレクトリに格納するか、または異なるファイル名で格納することができます。

データベースからのデータ

JSP ページを使用するか Java サーブレットを使用するかに関係なく、製品名や製品の説明などの動的な情報は、多くの場合データベースに格納されています。様々な翻訳を区別するには、この情報を保持するデータベース・スキーマに情報の言語を示す列を含める必要があります。翻訳された情報を選択するには、問合せに `WHERE` 句を含めて、問合せの希望する言語で情報を選択する必要があります。

ユーザー・ロケールの表記規則によるデータの表示

アプリケーションのデータは、ユーザーが期待するとおりに表示する必要があります。そうしないと、データの意味が間違って解釈される場合があります。たとえば、「12/11/05」はアメリカ合衆国では「2005 年 12 月 11 日」を意味しますが、イギリスでは「2005 年 11 月 12 日」を意味します。数値および通貨の書式にも同様の混乱があります。たとえば、ピリオド（.）はアメリカ合衆国では小数点の記号ですが、ドイツでは千単位のセパレータとして使用されます。

言語にはそれぞれ独自のソート・ルールがあります。アルファベット順に照合される言語、文字の画数に従う言語、単語の発音で順序付けられる言語などがあります。ユーザーが慣れている言語順序でソートしないでデータを表示すると、情報の検索が難しくなり、時間がかかる場合があります。

アプリケーションのロジックおよびデータベースから取得されるデータ量によっては、アプリケーション・レベルではなくデータベース・レベルでデータを書式設定する方が適切な場合があります。Oracle Database には、ユーザーのロケール設定がわかっているときにデータの表示を調整するのに役立つ多くの機能があります。次の項では、SQL でのロケールに依存する操作の例について説明します。

- Oracle の日付書式
- Oracle の数値書式
- Oracle の言語ソート
- Oracle のエラー・メッセージ

Oracle の日付書式

Oracle Database には、3 つの異なる日付表示書式があります。標準の日付、短い日付および長い日付です。例 9-7 に、アメリカ合衆国とドイツにおける短い日付書式と長い日付書式の違いを示します。

例 9-7 ロケールによる日付書式の違い（アメリカ合衆国とドイツ）

```
SQL> ALTER SESSION SET NLS_TERRITORY=america NLS_LANGUAGE=american;

Session altered.

SQL> SELECT employee_id EmpID,
  2  SUBSTR(first_name,1,1)||'.'||last_name "EmpName",
  3  TO_CHAR(hire_date,'DS') "Hiredate",
  4  TO_CHAR(hire_date,'DL') "Long HireDate"
  5  FROM employees
  6* WHERE employee_id <105;

     EMPID EmpName                   Hiredate   Long HireDate
---------- ------------------------- ---------- ----------------------------
       100 S.King                    06/17/1987 Wednesday, June 17, 1987
       101 N.Kochhar                 09/21/1989 Thursday, September 21, 1989
       102 L.De Haan                 01/13/1993 Wednesday, January 13, 1993
       103 A.Hunold                  01/03/1990 Wednesday, January 3, 1990
       104 B.Ernst                   05/21/1991 Tuesday, May 21, 1991


SQL> ALTER SESSION SET SET NLS_TERRITORY=germany NLS_LANGUAGE=german;

Session altered.

SQL> SELECT employee_id EmpID,
  2  SUBSTR(first_name,1,1)||'.'||last_name "EmpName",
  3  TO_CHAR(hire_date,'DS') "Hiredate",
  4  TO_CHAR(hire_date,'DL') "Long HireDate"
  5  FROM employees
  6* WHERE employee_id <105;


     EMPID EmpName                   Hiredate Long HireDate
---------- ------------------------- -------- ------------------------------
       100 S.King                    17.06.87 Mittwoch, 17. Juni 1987
       101 N.Kochhar                 21.09.89 Donnerstag, 21. September 1989
       102 L.De Haan                 13.01.93 Mittwoch, 13. Januar 1993
       103 A.Hunold                  03.01.90 Mittwoch, 3. Januar 1990
       104 B.Ernst                   21.05.91 Dienstag, 21. Mai 1991
```

Oracle の数値書式

例 9-8 に、アメリカ合衆国とドイツでの小数点文字とグループのセパレータの違いを示します。

例 9-8 ロケールによる数値書式の違い（アメリカ合衆国とドイツ）

```
SQL> ALTER SESSION SET SET NLS_TERRITORY=america;

Session altered.

SQL> SELECT employee_id EmpID,
  2  SUBSTR(first_name,1,1)||'.'||last_name "EmpName",
  3  TO_CHAR(salary, '99G999D99') "Salary"
  4  FROM employees
  5* WHERE employee_id <105

     EMPID EmpName                   Salary
---------- ------------------------- ----------
       100 S.King                     24,000.00
       101 N.Kochhar                  17,000.00
       102 L.De Haan                  17,000.00
       103 A.Hunold                    9,000.00
       104 B.Ernst                     6,000.00

SQL> ALTER SESSION SET SET NLS_TERRITORY=germany;

Session altered.

SQL> SELECT employee_id EmpID,
  2  SUBSTR(first_name,1,1)||'.'||last_name "EmpName",
  3  TO_CHAR(salary, '99G999D99') "Salary"
  4  FROM employees
  5* WHERE employee_id <105

     EMPID EmpName                   Salary
---------- ------------------------- ----------
       100 S.King                     24.000,00
       101 N.Kochhar                  17.000,00
       102 L.De Haan                  17.000,00
       103 A.Hunold                    9.000,00
       104 B.Ernst                     6.000,00
```

Oracle の言語ソート

スペインでは、従来から「ch」、「ll」および「ñ」を独自の文字として扱っており、順序としてはそれぞれ c、l および n の後になります。例 9-9 に、従業員名 Chen および Chung に対してスペイン語のソートを使用した場合の結果を示します。

例 9-9 言語ソートの違い（バイナリとスペイン語）

```
SQL> ALTER SESSION SET NLS_SORT=binary;

Session altered.

SQL> SELECT employee_id EmpID,
  2         last_name "Last Name"
  3  FROM employees
  4  WHERE last_name LIKE 'C%'
  5* ORDER BY last_name

     EMPID Last Name
---------- -------------------------
       187 Cabrio
```

```
       148 Cambrault
       154 Cambrault
       110 Chen
       188 Chung
       119 Colmenares

6 rows selected.

SQL> ALTER SESSION SET NLS_SORT=spanish_m;

Session altered.

SQL> SELECT employee_id EmpID,
  2         last_name "Last Name"
  3  FROM employees
  4  WHERE last_name LIKE 'C%'
  5* ORDER BY last_name

     EMPID Last Name
---------- -------------------------
       187 Cabrio
       148 Cambrault
       154 Cambrault
       119 Colmenares
       110 Chen
       188 Chung

6 rows selected.
```

Oracle のエラー・メッセージ

NLS_LANGUAGE パラメータによって、データベースから返されるデータベース・エラー・メッセージの言語も制御されます。SQL 文を送信する前にこのパラメータを設定すると、ローカル言語固有のデータベース・エラー・メッセージがアプリケーションに返されます。

次のサーバー・メッセージについて考えてみます。

```
ORA-00942: table or view does not exist
```

NLS_LANGUAGE パラメータをフランス語に設定すると、サーバー・メッセージは次のように表示されます。

```
ORA-00942: table ou vue inexistante
```

> **関連項目：** Oracle Database でのグローバリゼーション・サポート機能の詳細は、『Oracle Database 2 日で開発者ガイド』の第 6 章「グローバル環境での作業」を参照してください。

JDeveloper での JSP ページのテキストのローカライズ

Java アプリケーションでは、リソース・バンドルを使用して、JSP ページで使用される、ローカライズされた様々なテキストが提供されています。

リソース・バンドルには、ロケール固有のオブジェクトが含まれています。ロケール固有のリソース（ページに表示するテキストなど）がプログラムで必要な場合は、現在のユーザー・ロケールに適したリソース・バンドルからロードできます。この方法では、実際のテキストをリソース・バンドルに分離することによって、大部分がユーザー・ロケールから独立したプログラム・コードを記述できます。

リソース・バンドル・テクノロジには、次のような特徴があります。

- リソース・バンドルは、メンバーで共通の基準名を共有するファミリに属しますが、名前にはロケールを示す構成要素も追加されます。たとえば、リソース・バンドルのファミリの基準名が MyResources であるとします。ドイツ語のロケール固有のバージョンは、MyResources_de などとなります。
- ファミリ内の各リソース・バンドルには同じ項目が含まれていますが、項目はそのリソース・バンドルによって表されるロケール用に翻訳されています。たとえば、あるボタンで使用される String は、MyResources では Cancel と定義されていますが、MyResources_de では Abbrechen と定義されている場合があります。
- 異なる国の異なるリソースを特殊化できます。たとえば、スイス（CH）のドイツ語（de）などです。

アプリケーションでリソース・バンドルを使用するには、次の手順を実行します。

1. リソース・バンドルを作成します。
2. ビジュアル・コンポーネントがあるページで、そのページで使用するリソース・バンドルを識別します。
3. ページに表示するテキストの各項目について、ハードコードされたテキストを使用するかわりに、リソース・バンドルからテキストを取得します。

> **関連項目：** リソース・バンドルに関する Sun 社のドキュメントを参照してください。
>
> http://java.sun.com/j2se/1.4.2/docs/api/java/util/ResourceBundle.html

サンプル・アプリケーションでは、次の場所でリソース・バンドルを使用できます。

- JSP ページの見出しおよびラベル。この場合、ページでテキストを直接入力するのではなく、スクリプトレットを使用してテキストを検索できます。
- ボタンなどのコントロールの値。この場合、ボタンの value プロパティを、リソース・バンドルからテキストを取得する式に設定します。

この項では、次のタスクについて説明します。

- リソース・バンドルの作成
- JSP ページでのリソース・バンドル・テキストの使用

リソース・バンドルの作成

デフォルトのリソース・バンドルを作成するには、次の手順を実行します。

1. クラス java.util.ListResourceBundle を拡張する新しい Java クラス MyResources.java を作成します。
2. リソース・バンドル・クラスおよび内容を返すメソッドを次のように定義します。

```
public class MyResources extends ListResourceBundle
{
  public Object[][] getContents() {
    return contents;
  }
  static final Object[][] contents = {
  };
}
```

3. ページに必要なテキストの各項目のエントリを追加します。キーとそのキーのテキストを指定します。たとえば、次の例では、コメントは他の言語に翻訳する必要がある文字列を示します。

```
static final Object[][] contents = {
  // LOCALIZE THIS
  {"CompanyName", "AnyCo Corporation"},
  {"SiteName", "HR Application"},
  {"FilterButton", "Filter"},
  {"UpdateButton", "Update"},
  // END OF MATERIAL TO LOCALIZE
};
```

完成したリソース・バンドル・クラスは、例 9-10 のようになります。

例 9-10 リソース・バンドル・クラスの作成

```
public class MyResources extends ListResourceBundle
{
  public Object[][] getContents() {
    return contents;
  }
  static final Object[][] contents = {
    // LOCALIZE THIS
    {"CompanyName", "AnyCo Corporation"},
    {"SiteName", "HR Application"},
    {"FilterButton", "Filter"},
    {"UpdateButton", "Update"},
    // END OF MATERIAL TO LOCALIZE
  };
}
```

アプリケーションをグローバル化するには、サポートする様々なロケールについて、各言語での項目のテキストを含む、ロケール固有のリソース・バンドルを作成する必要があります。

JSP ページでのリソース・バンドル・テキストの使用

リソース・バンドルで定義されたテキストを JSP ページで使用するには、次の手順を実行します。

1. 使用する JSP ページ（edit.jsp など）をビジュアル・エディタで開きます。
2. ページの一番上で最初の見出しの前に新しい行を作成し、行の**「スタイル」**を None に設定します。**jsp:usebean** タグを新しい行に追加します。**ID** として myResources を入力し、**クラス**として hr.MyResources を入力します。**「有効範囲」**を「セッション」に設定し、**「OK」**をクリックします。
3. **jsp:scriptlet** をページにドラッグします。ページの見出しなど、リソース・バンドル・テキストを表示する場所にドラッグします。

   「スクリプトレットの挿入」ダイアログ・ボックスで、次のように、リソース・バンドルからテキストを取得するスクリプトを入力します。

```
out.println(myResources.getString("CompanyName") + ": " +
myResources.getString("SiteName"));
```

4. 見出しにすでにテキストが表示されていた場合は、ここで削除できます。

5. ビジュアル・エディタの下の**「ソース」**タブを選択すると、次のようなページのコードが表示されます。

```
<h2 align="center">
  <% = myResources.getString("CompanyName") + ": " +
               myResources.getString("SiteName"));
  %>
</h2>
```

6. リソース・バンドル・テキストをボタンのラベルとして使用するには、ビジュアル・エディタでボタンをダブルクリックします。ボタンのプロパティ・ダイアログ・ボックスで、ボタンの**「値」**パラメータに、次のようなスクリプトを入力します。

```
<% out.println(myResources.getString("UpdateButton"));%>
```

7. ページのソース・コードを表示すると、次のようなコードが表示されます。

```
<input type="submit"
  value=<% out.println(myResources.getString("UpdateButton"));%> />
```

ここでアプリケーションを実行すると、リソース・バンドルで定義したテキストがページに表示されます。

索引

L

M

N

O

P

R

S

T

U

W

X

あ

い

お

か

く

け

こ

さ

す

せ

て

と

は

ひ

ふ

ま

ゆ

ら

れ