8
Java Stored Procedures Application Example

This chapter describes how to build a Java application with stored procedures.

This chapter contains the followings steps, from the design phase to the actual implementation, to develop a sample application:

- About Planning the Database Schema
- Creating the Database Tables
- Writing the Java Classes
- Loading the Java Classes
- Publishing the Java Classes
- Calling the Java Stored Procedures

8.1 About Planning the Database Schema

The objective of this example is to develop a simple system for managing customer purchase orders. To do this, you must devise a database schema plan. First, identify the business entities involved and their relationships. In this example, the basic entities are customers, purchase orders, line items, and stock items. So, you can have the following tables in the schema:

- `Customers`
- `Orders`
- `LineItems`
- `StockItems`

The `Customers` table has a one-to-many relationship with the `Orders` table because a customer can place one or many orders, but a given purchase order can be placed by only one customer. The relationship is optional because zero customers may place a given order. For example, an order may be placed by someone previously not defined as a customer.

The `Orders` table has a many-to-many relationship with the `StockItems` table because a purchase order can refer to many stock items, and a stock item can be referred to by many purchase orders. However, you do not know which purchase orders refer to which stock items. As a result, you introduce the notion of a line item. The `Orders` table has a one-to-many relationship with the `LineItems` table because a purchase order can list many line items, but a given line item can be listed by only one purchase order.

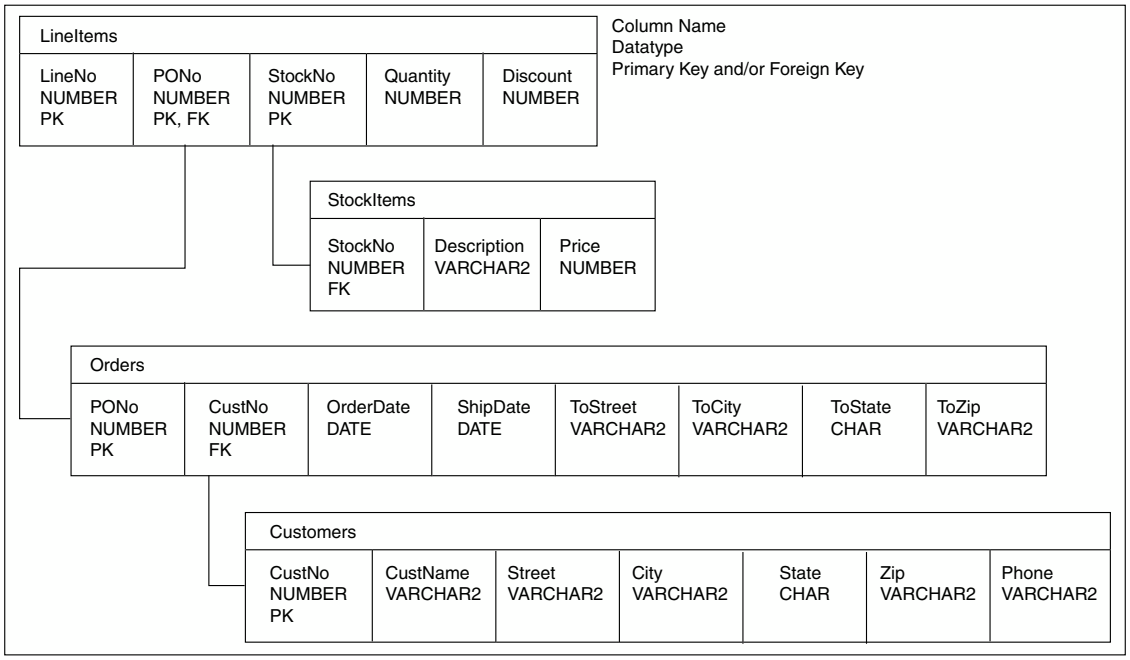
The `LineItems` table has a many-to-one relationship with the `StockItems` table because a line item can refer to only one stock item, but a given stock item can be referred to by many line items. The relationship is optional because zero line items may refer to a given stock item.

Figure 8-1 depicts the relationships between tables. In the schema plan, you establish these relationships using primary and foreign keys.

A primary key is a column or combination of columns whose values uniquely identify each row in a table. A foreign key is a column or combination of columns whose values match the

primary key in some other table. For example, the `PONo` column in the `LineItems` table is a foreign key matching the primary key in the `Orders` table. Every purchase order number in the `LineItems.PONo` column must also appear in the `Orders.PONo` column.

Figure 8-1 Schema Plan for Purchase Order Application



8.2 Creating the Database Tables

After planning the database schema, create the database tables for the schema plan. Define the `Customers` table as follows:

```
CREATE TABLE Customers (
CustNo NUMBER(3) NOT NULL,
CustName VARCHAR2(30) NOT NULL,
Street VARCHAR2(20) NOT NULL,
City VARCHAR2(20) NOT NULL,
State CHAR(2) NOT NULL,
Zip VARCHAR2(10) NOT NULL,
Phone VARCHAR2(12),
PRIMARY KEY (CustNo)
);
```

The `Customers` table stores information about customers. Essential information is defined as `NOT NULL`. For example, every customer must have a shipping address. However, the `Customers` table does not manage the relationship between customers and their purchase orders. As a result, this relationship must be managed by the `Orders` table, which you can define as follows:

```
CREATE TABLE Orders (
PONo NUMBER(5),
Custno NUMBER(3) REFERENCES Customers,
OrderDate DATE,
ShipDate DATE,
ToStreet VARCHAR2(20),
```

```
ToCity VARCHAR2(20),
ToState CHAR(2),
ToZip VARCHAR2(10),
PRIMARY KEY (PONo)
);
```

The line items have a relationship with purchase orders and stock items. The `LineItems` table manages these relationships using foreign keys. For example, the `StockNo` foreign key column in the `LineItems` table references the `StockNo` primary key column in the `StockItems` table, which you can define as follows:

```
CREATE TABLE StockItems (
StockNo NUMBER(4) PRIMARY KEY,
Description VARCHAR2(20),
Price NUMBER(6,2))
);
```

The `Orders` table manages the relationship between a customer and purchase order using the `CustNo` foreign key column, which references the `CustNo` primary key column in the `Customers` table. However, the `Orders` table does not manage the relationship between a purchase order and its line items. As a result, this relationship must be managed by the `LineItems` table, which you can define as follows:

```
CREATE TABLE LineItems (
LineNo NUMBER(2),
PONo NUMBER(5) REFERENCES Orders,
StockNo NUMBER(4) REFERENCES StockItems,
Quantity NUMBER(2),
Discount NUMBER(4,2),
PRIMARY KEY (LineNo, PONo)
);
```

8.3 Writing the Java Classes

After creating the database tables, you consider the operations required in a purchase order system and write the appropriate Java methods. In a simple system based on the tables defined in the preceding examples, you need methods for registering customers, stocking parts, entering orders, and so on. You can implement these methods in a Java class, `POManager`, as follows:

```
import java.sql.*;
import java.io.*;
import oracle.jdbc.*;

public class POManager
{
  public static void addCustomer (int custNo, String custName, String street,
   String city, String state, String zipCode, String phoneNo) throws SQLException
  {
    String sql = "INSERT INTO Customers VALUES (?,?,?,?,?,?,?)";
    try
    {
      Connection conn = DriverManager.getConnection("jdbc:default:connection:");
      PreparedStatement pstmt = conn.prepareStatement(sql);
      pstmt.setInt(1, custNo);
      pstmt.setString(2, custName);
      pstmt.setString(3, street);
      pstmt.setString(4, city);
      pstmt.setString(5, state);
      pstmt.setString(6, zipCode);
```

```
    pstmt.setString(7, phoneNo);
    pstmt.executeUpdate();
    pstmt.close();
  }
  catch (SQLException e)
  {
    System.err.println(e.getMessage());
  }
}

public static void addStockItem (int stockNo, String description, float price)
                                                                  throws SQLException
{
  String sql = "INSERT INTO StockItems VALUES (?,?,?)";
  try
  {
    Connection conn = DriverManager.getConnection("jdbc:default:connection:");
    PreparedStatement pstmt = conn.prepareStatement(sql);
    pstmt.setInt(1, stockNo);
    pstmt.setString(2, description);
    pstmt.setFloat(3, price);
    pstmt.executeUpdate();
    pstmt.close();
  }
  catch (SQLException e)
  {
    System.err.println(e.getMessage());
  }
}

public static void enterOrder (int orderNo, int custNo, String orderDate,
 String shipDate, String toStreet, String toCity, String toState,
  String toZipCode) throws SQLException
{
  String sql = "INSERT INTO Orders VALUES (?,?,?,?,?,?,?,?)";
  try
  {
    Connection conn = DriverManager.getConnection("jdbc:default:connection:");
    PreparedStatement pstmt = conn.prepareStatement(sql);
    pstmt.setInt(1, orderNo);
    pstmt.setInt(2, custNo);
    pstmt.setString(3, orderDate);
    pstmt.setString(4, shipDate);
    pstmt.setString(5, toStreet);
    pstmt.setString(6, toCity);
    pstmt.setString(7, toState);
    pstmt.setString(8, toZipCode);
    pstmt.executeUpdate();
    pstmt.close();
  }
  catch (SQLException e)
  {
    System.err.println(e.getMessage());
  }
}

public static void addLineItem (int lineNo, int orderNo, int stockNo,
 int quantity, float discount) throws SQLException
{
  String sql = "INSERT INTO LineItems VALUES (?,?,?,?,?)";
  try
  {
```

```
    Connection conn = DriverManager.getConnection("jdbc:default:connection:");
    PreparedStatement pstmt = conn.prepareStatement(sql);
    pstmt.setInt(1, lineNo);
    pstmt.setInt(2, orderNo);
    pstmt.setInt(3, stockNo);
    pstmt.setInt(4, quantity);
    pstmt.setFloat(5, discount);
    pstmt.executeUpdate();
    pstmt.close();
  }
  catch (SQLException e)
  {
    System.err.println(e.getMessage());
  }
}

public static void totalOrders () throws SQLException
{
  String sql = "SELECT O.PONo, ROUND(SUM(S.Price * L.Quantity)) AS TOTAL " +
   "FROM Orders O, LineItems L, StockItems S " +
   "WHERE O.PONo = L.PONo AND L.StockNo = S.StockNo " +
   "GROUP BY O.PONo";
  try
  {
    Connection conn = DriverManager.getConnection("jdbc:default:connection:");
    PreparedStatement pstmt = conn.prepareStatement(sql);
    ResultSet rset = pstmt.executeQuery();
    printResults(rset);
    rset.close();
    pstmt.close();
  }
  catch (SQLException e)
  {
    System.err.println(e.getMessage());
  }
}

static void printResults (ResultSet rset) throws SQLException
{
  String buffer = "";
  try
  {
    ResultSetMetaData meta = rset.getMetaData();
    int cols = meta.getColumnCount(), rows = 0;
    for (int i = 1; i <= cols; i++)
    {
      int size = meta.getPrecision(i);
      String label = meta.getColumnLabel(i);
      if (label.length() > size)
        size = label.length();
      while (label.length() < size)
        label += " ";
      buffer = buffer + label + " ";
    }
    buffer = buffer + "\n";
    while (rset.next())
    {
      rows++;
      for (int i = 1; i <= cols; i++)
      {
        int size = meta.getPrecision(i);
        String label = meta.getColumnLabel(i);
```

```
        String value = rset.getString(i);
        if (label.length() > size)
          size = label.length();
        while (value.length() < size)
          value += " ";
        buffer = buffer + value + " ";
      }
      buffer = buffer + "\n";
    }
    if (rows == 0)
      buffer = "No data found!\n";
    System.out.println(buffer);
  }
  catch (SQLException e)
  {
    System.err.println(e.getMessage());
  }
}

public static void checkStockItem (int stockNo) throws SQLException
{
  String sql = "SELECT O.PONo, O.CustNo, L.StockNo, " +
   "L.LineNo, L.Quantity, L.Discount " +
   "FROM Orders O, LineItems L " +
   "WHERE O.PONo = L.PONo AND L.StockNo = ?";
  try
  {
    Connection conn = DriverManager.getConnection("jdbc:default:connection:");
    PreparedStatement pstmt = conn.prepareStatement(sql);
    pstmt.setInt(1, stockNo);
    ResultSet rset = pstmt.executeQuery();
    printResults(rset);
    rset.close();
    pstmt.close();
  }
  catch (SQLException e)
  {
    System.err.println(e.getMessage());
  }
}

public static void changeQuantity (int newQty, int orderNo, int stockNo)
                                                          throws SQLException
{
  String sql = "UPDATE LineItems SET Quantity = ? " +
   "WHERE PONo = ? AND StockNo = ?";
  try
  {
    Connection conn = DriverManager.getConnection("jdbc:default:connection:");
    PreparedStatement pstmt = conn.prepareStatement(sql);
    pstmt.setInt(1, newQty);
    pstmt.setInt(2, orderNo);
    pstmt.setInt(3, stockNo);
    pstmt.executeUpdate();
    pstmt.close();
  }
  catch (SQLException e)
  {
    System.err.println(e.getMessage());
  }
}
```

```
  public static void deleteOrder (int orderNo) throws SQLException
  {
    String sql = "DELETE FROM LineItems WHERE PONo = ?";
    try
    {
      Connection conn = DriverManager.getConnection("jdbc:default:connection:");
      PreparedStatement pstmt = conn.prepareStatement(sql);
      pstmt.setInt(1, orderNo);
      pstmt.executeUpdate();
      sql = "DELETE FROM Orders WHERE PONo = ?";
      pstmt = conn.prepareStatement(sql);
      pstmt.setInt(1, orderNo);
      pstmt.executeUpdate();
      pstmt.close();
    }
    catch (SQLException e)
    {
      System.err.println(e.getMessage());
    }
  }
}
```

8.4 Loading the Java Classes

After writing the Java classes, use the `loadjava` tool to upload your Java stored procedures into Oracle Database, as follows:

```
> loadjava -u HR@myPC:1521:orcl -v -r -t POManager.java
Password: password
initialization complete
loading : POManager
creating : POManager
resolver : resolver ( ("*" HR) ("*" public) )
resolving: POManager
```

The `-v` option enables the verbose mode, the `-r` option compiles uploaded Java source files and resolves external references in the classes, and the `-t` option tells the `loadjava` tool to connect to the database using the client-side JDBC Thin driver.

8.5 Publishing the Java Classes

After loading the Java classes, publish your Java stored procedures in the Oracle data dictionary. To do this, you must write call specifications that map Java method names, parameter types, and return types to their SQL counterparts.

The methods in the `POManager` Java class are logically related. You can group their call specifications in a PL/SQL package. To do this, first, create the package specification, as follows:

```
CREATE OR REPLACE PACKAGE po_mgr AS
PROCEDURE add_customer (cust_no NUMBER, cust_name VARCHAR2,
street VARCHAR2, city VARCHAR2, state CHAR, zip_code VARCHAR2,
phone_no VARCHAR2);
PROCEDURE add_stock_item (stock_no NUMBER, description VARCHAR2,
price NUMBER);
PROCEDURE enter_order (order_no NUMBER, cust_no NUMBER,
order_date VARCHAR2, ship_date VARCHAR2, to_street VARCHAR2,
to_city VARCHAR2, to_state CHAR, to_zip_code VARCHAR2);
PROCEDURE add_line_item (line_no NUMBER, order_no NUMBER,
```

```
stock_no NUMBER, quantity NUMBER, discount NUMBER);
PROCEDURE total_orders;
PROCEDURE check_stock_item (stock_no NUMBER);
PROCEDURE change_quantity (new_qty NUMBER, order_no NUMBER,
stock_no NUMBER);
PROCEDURE delete_order (order_no NUMBER);
END po_mgr;
```

Then, create the package body by writing call specifications for the Java methods, as follows:

```
CREATE OR REPLACE PACKAGE BODY po_mgr AS
PROCEDURE add_customer (cust_no NUMBER, cust_name VARCHAR2,
street VARCHAR2, city VARCHAR2, state CHAR, zip_code VARCHAR2,
phone_no VARCHAR2) AS LANGUAGE JAVA
NAME 'POManager.addCustomer(int, java.lang.String,
java.lang.String, java.lang.String, java.lang.String,
java.lang.String, java.lang.String)';

PROCEDURE add_stock_item (stock_no NUMBER, description VARCHAR2,
price NUMBER) AS LANGUAGE JAVA
NAME 'POManager.addStockItem(int, java.lang.String, float)';

PROCEDURE enter_order (order_no NUMBER, cust_no NUMBER,
order_date VARCHAR2, ship_date VARCHAR2, to_street VARCHAR2,
to_city VARCHAR2, to_state CHAR, to_zip_code VARCHAR2)
AS LANGUAGE JAVA
NAME 'POManager.enterOrder(int, int, java.lang.String,
java.lang.String, java.lang.String, java.lang.String,
java.lang.String, java.lang.String)';

PROCEDURE add_line_item (line_no NUMBER, order_no NUMBER,
stock_no NUMBER, quantity NUMBER, discount NUMBER)
AS LANGUAGE JAVA
NAME 'POManager.addLineItem(int, int, int, int, float)';

PROCEDURE total_orders
AS LANGUAGE JAVA
NAME 'POManager.totalOrders()';

PROCEDURE check_stock_item (stock_no NUMBER)
AS LANGUAGE JAVA
NAME 'POManager.checkStockItem(int)';

PROCEDURE change_quantity (new_qty NUMBER, order_no NUMBER,
stock_no NUMBER) AS LANGUAGE JAVA
NAME 'POManager.changeQuantity(int, int, int)';

PROCEDURE delete_order (order_no NUMBER)
AS LANGUAGE JAVA
NAME 'POManager.deleteOrder(int)';
END po_mgr;
```

8.6 Calling the Java Stored Procedures

After publishing the Java classes, call your Java stored procedures from the top level and from database triggers, SQL data manipulation language (DML) statements, and PL/SQL blocks. Use the dot notation to reference these stored procedures in the `po_mgr` package.

From an anonymous PL/SQL block, you may start the new purchase order system by stocking parts, as follows:

```
BEGIN
  po_mgr.add_stock_item(2010, 'camshaft', 245.00);
  po_mgr.add_stock_item(2011, 'connecting rod', 122.50);
  po_mgr.add_stock_item(2012, 'crankshaft', 388.25);
  po_mgr.add_stock_item(2013, 'cylinder head', 201.75);
  po_mgr.add_stock_item(2014, 'cylinder sleeve', 73.50);
  po_mgr.add_stock_item(2015, 'engine bearning', 43.85);
  po_mgr.add_stock_item(2016, 'flywheel', 155.00);
  po_mgr.add_stock_item(2017, 'freeze plug', 17.95);
  po_mgr.add_stock_item(2018, 'head gasket', 36.75);
  po_mgr.add_stock_item(2019, 'lifter', 96.25);
  po_mgr.add_stock_item(2020, 'oil pump', 207.95);
  po_mgr.add_stock_item(2021, 'piston', 137.75);
  po_mgr.add_stock_item(2022, 'piston ring', 21.35);
  po_mgr.add_stock_item(2023, 'pushrod', 110.00);
  po_mgr.add_stock_item(2024, 'rocker arm', 186.50);
  po_mgr.add_stock_item(2025, 'valve', 68.50);
  po_mgr.add_stock_item(2026, 'valve spring', 13.25);
  po_mgr.add_stock_item(2027, 'water pump', 144.50);
  COMMIT;
END;
```

Register your customers, as follows:

```
BEGIN
  po_mgr.add_customer(101, 'A-1 Automotive', '4490 Stevens Blvd',
    'San Jose', 'CA', '95129', '408-555-1212');
  po_mgr.add_customer(102, 'AutoQuest', '2032 America Ave',
    'Hayward', 'CA', '94545', '510-555-1212');
  po_mgr.add_customer(103, 'Bell Auto Supply', '305 Cheyenne Ave',
    'Richardson', 'TX', '75080', '972-555-1212');
  po_mgr.add_customer(104, 'CarTech Auto Parts', '910 LBJ Freeway',
    'Dallas', 'TX', '75234', '214-555-1212');
  COMMIT;
END;
```

Enter the purchase orders placed by various customers, as follows:

```
BEGIN
  po_mgr.enter_order(30501, 103, '14-SEP-1998', '21-SEP-1998',
    '305 Cheyenne Ave', 'Richardson', 'TX', '75080');
  po_mgr.add_line_item(01, 30501, 2011, 5, 0.02);
  po_mgr.add_line_item(02, 30501, 2018, 25, 0.10);
  po_mgr.add_line_item(03, 30501, 2026, 10, 0.05);

  po_mgr.enter_order(30502, 102, '15-SEP-1998', '22-SEP-1998',
    '2032 America Ave', 'Hayward', 'CA', '94545');
  po_mgr.add_line_item(01, 30502, 2013, 1, 0.00);
  po_mgr.add_line_item(02, 30502, 2014, 1, 0.00);

  po_mgr.enter_order(30503, 104, '15-SEP-1998', '23-SEP-1998',
    '910 LBJ Freeway', 'Dallas', 'TX', '75234');
  po_mgr.add_line_item(01, 30503, 2020, 5, 0.02);
  po_mgr.add_line_item(02, 30503, 2027, 5, 0.02);
  po_mgr.add_line_item(03, 30503, 2021, 15, 0.05);
  po_mgr.add_line_item(04, 30503, 2022, 15, 0.05);

  po_mgr.enter_order(30504, 101, '16-SEP-1998', '23-SEP-1998',
    '4490 Stevens Blvd', 'San Jose', 'CA', '95129');
  po_mgr.add_line_item(01, 30504, 2025, 20, 0.10);
  po_mgr.add_line_item(02, 30504, 2026, 20, 0.10);
```

```
  COMMIT;
END;
```

In SQL*Plus, after redirecting output to the SQL*Plus text buffer, you can call the `totalOrders()` method, as follows:

```
SQL> SET SERVEROUTPUT ON
SQL> CALL dbms_java.set_output(2000);
...
SQL> CALL po_mgr.total_orders();
PONO   TOTAL
30501  1664
30502  275
30503  4149
30504  1635

Call completed.
```