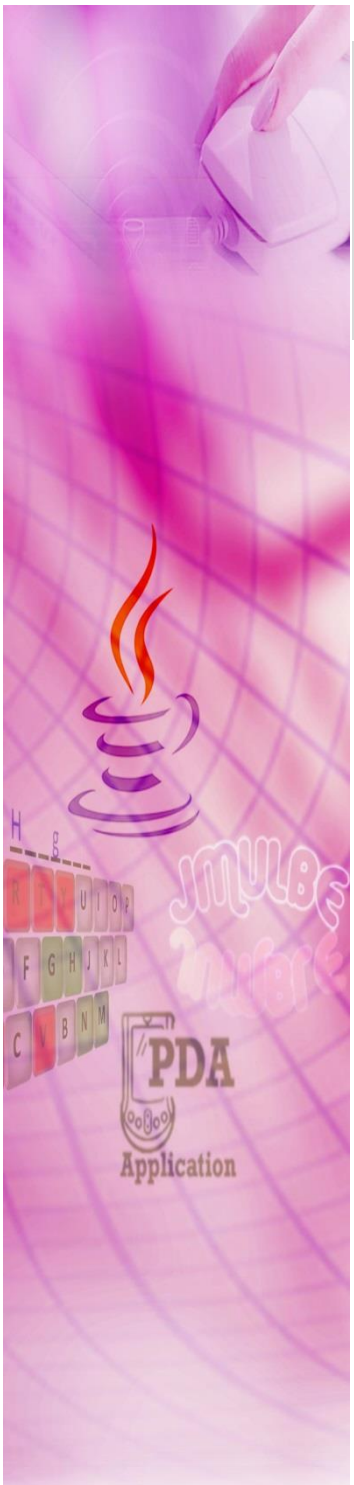

Instructor Inputs

Session 13

Session Overview

This session covers the Creating and Executing JDBC Statements and Handling SQL Exceptions topics of the Using JDBC API section and the Accessing Result Sets section of Chapter 9. In addition, it covers the Create Applications Using the PreparedStatement Object section of Chapter 10 of the book, Programming in Java – SG.

This session will help the students to become familiar with creating and executing JDBC statements. In addition, they will learn to handle SQL exceptions. Further, the students will learn how to execute the SQL statements with arguments using JDBC.

Using JDBC API

Handling Tips

Start the session by telling the students that SQL queries are used to retrieve and store data in a database. Therefore, to create a Java application to retrieve and store data in a database, they need to create and execute the equivalent JDBC statements. Next, discuss how to send and retrieve data from the database by using the `Statement` object. In addition, tell them that the `Statement` object can be created by using the `createStatement()` method of the `Connection` interface. Further, discuss the methods of the `Statement` interface to send the static SQL statement to a database. In addition, while discussing this topic, you can explain the additional methods given under the Creating and Executing JDBC Statements subtopic in the Additional Inputs topic.

Thereafter, explain to the students that while connecting to a database or retrieving data from a database, exceptions can occur. For example, if the table name specified in a query does not exist in the database, an exception will be thrown. Therefore, to prevent the abnormal interruption of the program execution, the exceptions must be caught in the program. Thereafter, explain the classes and methods used to handle the SQL exceptions. In addition, while explaining this topic, you can discuss the information of subclasses given under the Handling SQL Exception subtopic in the Additional Inputs topic.

Additional Inputs

Creating and Executing JDBC Statements

The following table lists some additional methods of the `Statement` interface to send the static SQL statement to a database.

Method	***Description***
`boolean execute(String sql, int[] columnIndexes)`	*It executes the specified SQL statement that may return multiple results, and signals the driver that the auto-generated keys indicated in the given array should be made available for retrieval. The array contains the indexes of the columns in the target table that contain the auto-generated keys that should be made available.*

Method	*Description*
`boolean execute(String sql, String[] columnNames)`	*It executes the specified SQL statement that may return multiple results, and signals the driver that the auto-generated keys indicated in the given array should be made available for retrieval. The array contains the names of the columns in the target table that contain the auto-generated keys that should be made available.*
`boolean executeUpdate(String sql, int[] columnIndexes)`	*It executes the specified SQL statement and signals the driver that the auto-generated keys indicated in the given array should be made available for retrieval. The array contains the indexes of the columns in the target table that contain the auto-generated keys that should be made available.*
`boolean executeUpdate(String sql, String[] columnNames)`	*It executes the specified SQL and signals the driver that the auto-generated keys indicated in the given array should be made available for retrieval. The array contains the names of the columns in the target table that contain the auto-generated keys that should be made available.*

The Methods of the Statement Interface

Handling SQL Exception

The following table lists some commonly-used subclasses of the `SQLException` class.

Class	*Description*
`BatchUpdateException`	*This class handles the exception that is thrown when an error occurs during a batch update operation.*
`RowSetWarning`	*This class provides the information about database warnings set on the `RowSet` objects.*
`SQLWarning`	*This class provides the information on database access warnings that may be retrieved from the `Connection`, `Statement`, and `ResultSet` objects.*

The Subclasses of the SQLException Class

Accessing Result Sets

Explain to the students that when they retrieve data from a database, there must be a component that can store the retrieved data temporarily. Thereafter, inform the students that the result sets are used to store the data retrieved from the database. Further, tell them that the result sets contain cursors that allow them to traverse through the retrieved data. Thereafter, discuss the various types of result sets and discuss the overloaded versions of the `createStatement()` method that are used to create the different types of result set objects. In addition, while discussing this topic, you can explain the additional fields of the `ResultSet` interface given under the Types of Result Sets subtopic in the Additional Inputs topic.

Further, explain that the `ResultSet` interface provides the various methods to access data stored in a `ResultSet` object. Next, explain the methods of the `ResultSet` interface. In addition, while discussing this topic, you can explain the additional methods of the `ResultSet` interface given under the Methods of the ResultSet Interface subtopic in the Additional Inputs topic.

Additional Inputs

Types of Result Sets

The following table lists the additional fields of the `ResultSet` interface.

ResultSet Field	***Description***
`FETCH_FORWARD`	*Specifies that the cursor of the `ResultSet` object will be processed in the forward direction, first to last.*
`FETCH_REVERSE`	*Specifies that the cursor of the `ResultSet` object will be processed in the reverse direction, last to first.*
`FETCH_UNKNOWN`	*Specifies that the cursor of the `ResultSet` object will be processed in an unknown direction.*

The Additional Fields of the ResultSet Interface

Methods of the ResultSet Interface

The following table lists some additional methods of the `ResultSet` interface.

Method	***Description***
`void cancelRowUpdates()`	*Cancels the updates made to the current row in the `ResultSet` object.*
`boolean isClosed()`	*Determines whether the `ResultSet` object has been closed.*

Method	*Description*
`boolean next()`	*Moves the cursor forward one row from its current position.*
`void refreshRow()`	*Refreshes the current row with its most recent value in the database.*

The Additional Methods of the ResultSet Interface

Activity 9.1: Creating a JDBC Application to Query a Database

Handling Tips

Discuss the problem statement with the students.

The solution file, **AuthorsInfo.java**, for this activity is provided at the following location in the TIRM CD:

- **Datafiles For Faculty\Activities\Chapter 09\Activity 9.1\Solution**

Creating Applications Using the PreparedStatement Object

Handling Tips

Explain to the students that while performing the database operations by using a Java application, data may be accepted at runtime. For example, consider a Java application that is used to insert the customer records in the `Customers` table. In this application, the user enters the customer information in a form and clicks a button to save the record. Inform the students that to achieve such functionalities in Java applications, they need to use the `PreparedStatement` interface and its methods to accept data from the users at runtime. In addition, while discussing this topic, you can explain the additional methods of the `PreparedStatement` interface given under the Methods of the PreparedStatement Interface subtopic in the Additional Inputs topic.

Explain how an SQL statement is prepared by using the `prepareStatement()` method. Explain how data can be retrieved, inserted, and deleted in a database by using the `PreparedStatement` object. In addition, while discussing these topics, you can explain the additional codes given under the Retrieving Rows, Inserting Rows, and Updating and Deleting Rows subtopics in the Additional Examples topic.

Additional Inputs

Methods of the PreparedStatement Interface

The following table lists some additional methods of the `PreparedStatement` interface.

Method	***Description***
`void setDate(int index, Date value)`	*Sets the specified `Date` object for the parameter corresponding to the specified index.*
`void setNull(int index, int value)`	*Sets the `SQL NULL` value for the parameter corresponding to the specified index.*
`void setObject(int index, Object value)`	*Sets the specified `Object` object for the parameter corresponding to the specified index.*
`void setTime(int index, Time value)`	*Sets the specified `Time` object for the parameter corresponding to the specified index.*

The Additional Methods of the PreparedStatement Interface

Retrieving Rows

You can use the following code snippet to retrieve the details of the student from the `Student` table by using the `PreparedStatement` object:

```
String str = "SELECT * FROM Student WHERE st_age = ? and st_teacher = ?";
PreparedStatement ps = con.prepareStatement(str);
ps.setInt(1, 13);
ps.setString(2, "Mark");

ResultSet rs=ps.executeQuery();
while(rs.next())
{
     System.out.println(rs.getString(1) + " " + rs.getString(2));
}
```

In the preceding code snippet, the `str` variable stores the `SELECT` statement that contains two input parameters. The `setInt()` and `setString()` methods are used to set the values for the `st_age` and `st_teacher` attributes of the `Student` table, respectively. The `SELECT` statement is executed by using the `executeQuery()` method, which returns a `ResultSet` object.

Inserting Rows

You can use the following code snippet to create a `PreparedStatement` object that inserts a row into the `Student` table by passing the student's data at runtime:

```
String str = "INSERT INTO Student (st_name, st_age) VALUES (?,?)";
PreparedStatement ps = con.prepareStatement(str);
ps.setString(1, "David wills");
ps.setInt(2, 14);
int rt=ps.executeUpdate();
```

In the preceding code snippet, the `str` variable stores the `INSERT` statement that contains two input parameters. The `setInt()` and `setString()` methods are used to set the values for the `st_name` and `st_age` columns of the `Student` table, respectively. The `INSERT` statement is executed by using the `executeUpdate()` method, which returns an integer value that specifies the number of rows inserted into the table.

Updating and Deleting Rows

You can use the following code snippet to modify the teacher name to `Peter Donald` where the class is `10` in the `Student` table by using the `PreparedStatement` object:

```
String str = "UPDATE Student SET st_teacher= ? WHERE st_class= ? ";
PreparedStatement ps = con.prepareStatement(str);
ps.setString(1, "Peter Donald");
ps.setInt(2, 10);
int rt=ps.executeUpdate();
```

In the preceding code snippet, two input parameters, `st_teacher` and `st_class`, contain the values for the teacher name and class attributes of the `Student` table, respectively.

You can use the following code snippet to delete a row from the `Student` table, where `st_teacher` is `John White` or `st_class` is `8` by using the `PreparedStatement` object:

```
String str = "DELETE FROM Student WHERE st_teacher= ? OR st_class = ?";
PreparedStatement ps = con.prepareStatement(str);
ps.setString(1, "Abraham White");
ps.setInt(2, 8);
int rt=ps.executeUpdate();
```

In the preceding code snippet, two input parameters, `st_teacher` and `st_class`, are used to delete the rows according to the specified values for the teacher name and class attributes of the `Student` table, respectively.

FAQs

- *Can we create the `PreparedStatement` object by passing the query without any parameter?*
  Ans: Yes, we can create the `PreparedStatement` object by passing the query without any parameter.

- *How can we retrieve Character Large Object (CLOB) and Binary Large Object (BLOB) stored in the `ResultSet` object?*

  Ans: The `getClob()` and `getBlob()` methods of the `ResultSet` interface are used to retrieve CLOB and BLOB stored in the `ResultSet` object. The `getClob()` method retrieves the value of the selected column in the current row of the `ResultSet` object as a CLOB object. The `getBlob()` method retrieves the value of the selected column in the current row of the `ResultSet` object as a BLOB object.

- *Can we use the `execute()` method of the `PreparedStatement` object to insert the values in the table?*

  Ans: Yes, we can use the `execute()` method of the `PreparedStatement` object to insert the values in the table, as shown in the following code snippet:

```
PreparedStatement ps = con.prepareStatement("INSERT INTO publishers (pub_id,
pub_name) VALUES (?, ?)");
   ps.setString(1, "P019");
   ps.setString(2, "Darwin Press");
   ps.execute();
```