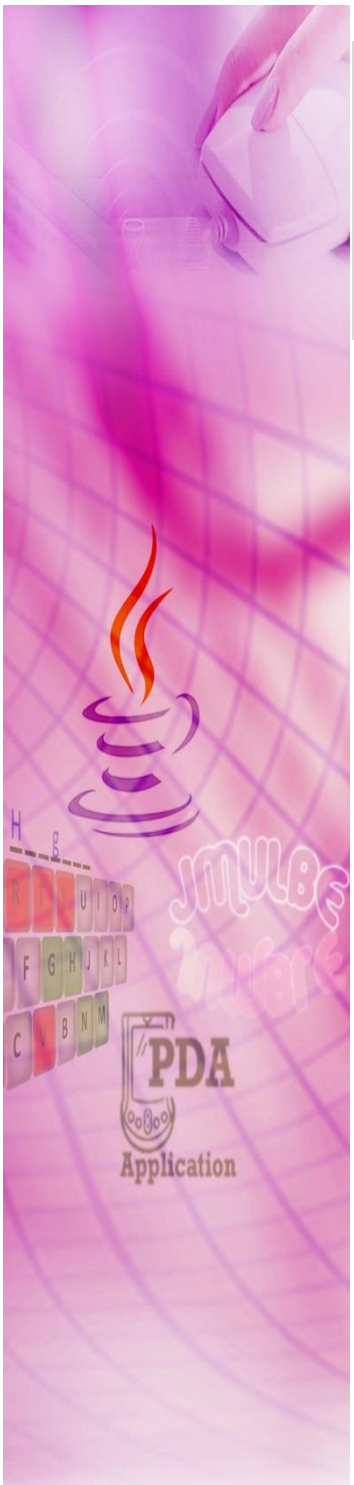

Instructor Inputs

Session 14

Session Overview

This session covers the Managing Database Transactions, Implementing Batch Updates in JDBC, Creating and Calling Stored Procedures in JDBC, and Using Metadata in JDBC sections of Chapter 10 of the book, Programming in Java – SG.

This session will help the students to understand the management of transactions and how to perform batch updates. In addition, the students will learn how to create and execute stored procedures.

Activity 10.1: Creating an Application that Uses the PreparedStatement Object

Handling Tips

Discuss the problem statement with the students.

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

- **Datafiles For Faculty\Activities\Chapter 10\Activity 10.1\Solution**

Managing Database Transactions

Handling Tips

Explain to the students that while performing data manipulation, multiple SQL statements may be executed to accomplish a task. For example, if an amount is transferred from the bank account, A, to another bank account, B, multiple SQL statements are executed. Few statements are used to connect to the account, A, and deduct the amount transferred and few statements are used to connect to the account, B, and add the amount transferred.

Inform the students that although multiple SQL statements are used in the preceding example, only one task is performed, that is, transferring of amount. Therefore, this task is considered as a single transaction. The transaction in the preceding example will be complete only when the amount is successfully deducted from the account, A, and added to the account, B. However, if any of the SQL statements fails and the amount is not deducted or added in the account, the transaction is considered to be failed. Explain to the students that a transaction is committed only when all the constituent SQL statements are executed successfully.

In addition, you can explain the various properties of a transaction by using the information provided in the Additional Inputs topic.

Additional Inputs

For a transaction to commit successfully within a database, it should possess the Atomicity, Consistency, Isolation, and Durability (ACID) properties.

The ACID properties that a transaction must possess are:

- **Atomicity**: This property states that either all the modifications are performed or none of the modifications are performed. For example, if some amount is transferred from the account, A, to the account, B, and the transaction succeeds, the amount is deducted from the account, A, and added to the

account, B. However, if the transaction fails, the amount is neither deducted from the account, A, nor added to the account, B.

- **Consistency**: This property states that the data is in a consistent state after a transaction is completed successfully. For example, the account, A, contains $500 and the account, B, contains $300 before the execution of a transaction. If $200 is transferred from the account, A, to the account, B, the account, A, should contain $300 and the account, B, should contain $500 after the successful execution of the transaction. However, if the transaction fails due to any reason, the account, A, should contain $500 and the account, B, should contain $300 to maintain the integrity of the database.
- **Isolation**: This property states that any data modification made by one transaction must be isolated from the modifications made by the other transactions. In simpler words, a transaction either accesses data in the state in which it was before a concurrent transaction modified it or accesses data after the second transaction has been completed. There is no scope for the transaction to see an intermediate state.
- **Durability**: This property states that any change in data by a completed transaction remains permanently in effect in the database. Therefore, any change in data due to a completed transaction persists even in the event of a system failure. This is ensured by taking backups of data and restoring transaction logs.

Implementing Batch Updates in JDBC

Handling Tips

Explain to the students that instead of executing each SQL statement separately, they can execute the statements as a batch. Inform that executing the SQL statements as a batch increases the efficiency of the application. Explain the methods given in the SG that are used to perform batch operations in a Java application. Further, explain the classes and methods used to handle exceptions that occur during batch operations. In addition, while discussing the topic, you can demonstrate the code given in the Additional Examples topic.

Additional Examples

Consider the following code that generates `BatchUpdateException`:

```
import java.sql.*;

public class BookInfo1
{
    public static void main(String args[])
    {
        try {
            /*Initialize and load Type 4 JDBC driver*/
            Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
            /*Connect to a data source using Library DSN*/
            try (Connection con =
DriverManager.getConnection("jdbc:sqlserver://localhost;databaseName=Library;
user=sa;password=password@123");
                    /*Create a Statement object*/
                    Statement stmt = con.createStatement();)
            {
                con.setAutoCommit(false);
                /*Add the INSERT statements to a batch*/
                stmt.addBatch("exec Authors_info");
```

```
                    /*Execute a batch using executeBatch() method*/
                    int[] results = stmt.executeBatch();
                    System.out.println("");
                    System.out.println("Using Statement object");
                    System.out.println("-------------------------");
                    for (int i = 0; i < results.length; i++)
                    {
                        System.out.println("Rows affected by " + (i + 1) + "
UPDATE statement: " + results[i]);
                    }
                    con.commit();
                }
            }
            catch (BatchUpdateException bue)
            {
                System.out.println("Error : " + bue);
            }
            catch (SQLException sqle)
            {
                System.out.println("Error : " + sqle);
            }
            catch (Exception e)
            {
                System.out.println("Error : " + e);
            }
        }
    }
```

In the preceding code, the `executeBatch()` method generate `BatchUpdateException` because the `executeBatch()` method of the `Statement` interface is used, which cannot execute the procedure.

Creating and Calling Stored Procedures in JDBC

Handling Tips

Explain to the students that stored procedures are a group of SQL statements stored in a database with unique names. Explain that stored procedures can be created and executed in the JDBC application by using the various methods of the `CallableStatement` interface. In addition, you can explain the benefits of a stored procedure by using the information provided in the Additional Inputs topic.

Inform the students that they can create and execute stored procedures with and without parameters. Discuss the methods used to create and call stored procedures without parameters. Explain the creation and invocation of stored procedures with parameters.

Additional Inputs

Use of stored procedures in an application provides the following benefits:

- **Precompiled execution**: The stored procedures are compiled only once when they are executed the first time. For further execution of the stored procedures, SQL Server uses the precompiled version. Therefore, the efficiency of the application improves in the situations where the stored procedures are called repeatedly.

- **Reduced client/server traffic**: As multiple SQL statements are combined together in a stored procedure, the client application does not need to connect to the database server for the execution of each SQL statement. Therefore, the traffic between the client application and the database server reduces if stored procedures are used.

Using Metadata in JDBC

Handling Tips

Explain to the students that while manipulating data stored in a database by using Java applications, they may need to retrieve information about the database components. In addition, they may need to be aware of the information about the `ResultSet` objects used in a Java application. Inform the students that they can retrieve information about the database components by using the objects of the `DatabaseMetaData` interface. Further, discuss the `DatabaseMetaData` interface and its commonly-used methods given in the SG. Thereafter, demonstrate the code given in the SG. In addition, while discussing this topic, you can discuss some additional methods of the `DatabaseMetaData` interface given in the Additional Inputs topic.

Next, explain the `ResultSetMetaData` interface that provides methods to retrieve information about the `ResultSet` objects. Thereafter, discuss the various methods of the `ResultSetMetaData` interface given in the SG. In addition, while discussing this topic, you can discuss the additional example given in the Additional Examples topic.

Additional Inputs

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

Methods	***Description***
`Connection getConnection()`	*Returns the connection that produced the metadata object.*
`String getDriverName()`	*Returns the name of the JDBC driver.*

The Methods of the DatabaseMetaData interface

Additional Examples

You can use the following code to retrieve and display the names and data types of the columns of the `Authors` table by using the `ResultSetMetaData` interface:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;

public class JDBCDemo
{
     public static void main(String[] args)
```

```
    {
        try (Connection con =
DriverManager.getConnection("jdbc:sqlserver://localhost;databaseName=Library;
user=sa;password=password@123");)
        {
            Statement stmt = con.createStatement();
            ResultSet res = stmt.executeQuery("Select * from Authors");
            if (res.next())
            {
                ResultSetMetaData rsmd = res.getMetaData();
                int columnCount = rsmd.getColumnCount();
                System.out.println("Column Name     Column Size     Is
Nullable");
                System.out.println("-------------------------------------");
                for (int i = 1; i <= columnCount; i++)
                {
                    System.out.println(rsmd.getColumnLabel(i) + " " +
rsmd.getColumnDisplaySize(i) + " " + rsmd.isNullable(i));
                }

            }
        }
        catch (SQLException ex)
        {
            System.out.println("Display Error Code");
            System.out.println(ex);
        }
    }
}
```

In the preceding code, a connection is established with the `Library` data source. The object of the `ResultSetMetaData` interface is declared using the `getMetaData()` method of the `ResultSet` interface. The `ResultSetMetaData` interface is used to retrieve the column name and column size, and whether a column can contain a `null` value by using the `getColumnLabel()`, `getColumnDisplaySize()`, and `isNullable()` methods.

Activity 10.2: Creating an Application to Determine the Structure of a Table

Handling Tips

Discuss the problem statement with the students.

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

- **Datafiles For Faculty\Activities\Chapter 10\Activity 10.2\Solution**

FAQs

- *Can the parameters of a `PreparedStatement` object be changed or cleared once they are set by using the setXXX methods?*

  Ans: Yes, the parameters of a `PreparedStatement` object can be changed or cleared. When the parameters of a `PreparedStatement` object are set by using the setXXX methods, this object keeps the values until they are reset. You can clear the parameter values by using the `clearParameters()` method of the `PreparedStatement` interface.

- *How can you determine the current status of the auto-commit mode?*

  Ans: The current status of the auto-commit mode can be determined by using the `getAutoCommit()` method of the `Connection` interface. For example, you can use the following code snippet to determine the auto-commit mode of the `Connection` object, `con`:

  ```
  boolean mode=con.getAutoCommit();
  ```

- *Is it possible to roll back a transaction till a savepoint in a JDBC application? If yes, how?*

  Ans: Yes, a transaction can be rolled back to a savepoint in a JDBC application. It can be achieved by using the `rollback(Savepoint savepoint)` method of the `Connection` interface.

- *What is the difference between a batch and a stored procedure?*

  Ans: A batch is a set of one or more SQL statements sent from a client to an instance of the database server for execution. It represents a unit of work submitted to the database engine by the users. However, a stored procedure is a set of SQL statements that is compiled and stored in the database server with a unique name.

  All the SQL statements are compiled and executed by the database server when a batch is executed. However, a stored procedure is a precompiled set of statements. Therefore, when a stored procedure is executed, the database server directly executes the constituent statements.