Lesson: 1.2

JDBC ResultSet

Objectives:

In this lesson, you will exercise to:

- ✓ Understand about JDBC ResultSet.
- ✓ Create different types of ResultSet.
- ✓ Navigate the ResultSet.
- ✓ View the ResultSet.
- ✓ Update the ResultSet.

JDBC ResultSet

- A *ResultSet* is a Java object that contains the results of executing an SQL query.
- In other words, it contains the rows that satisfy the conditions of the query.
- The general form of a result set is a table with column headings and the corresponding values returned by a query.
- A result set is also a table with rows and columns, but it contains only the column values from a database table that satisfy the conditions of a query.
- The SQL statements that read data from a database query return the data in a result set.
- The SELECT statement is the standard way to select rows from a database and view them in a result set.
- The **java.sql.ResultSet** interface represents the result set of a database query.
- A *ResultSet* object maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row.
- A user can access the data in a result set using a *cursor* one row at a time from top to bottom. A *cursor* can be thought of as a pointer to the rows of the result set that has the ability to keep track of which row is currently being accessed.
- A default *ResultSet* object is not updatable and has a cursor that moves forward only.

✍ Task 5 : Explore the API

5.1. Identify the constant indicating that the rows in a result set will be processed in a forward direction; first-to-last.
5.2. Identify the method which returns ResultSet.TYPE_FORWARD_ONLY, ResultSet.TYPE_SCROLL_INSENSITIVE, or ResultSet.TYPE_SCROLL_SENSITIVE
5.3. Identify the constant value of the constant indicating the concurrency mode for a ResultSet object that may be updated.
5.4. Identify the method that moves the cursor a relative number of rows, either positive or negative.
5.5. Identify the method which true if a row was deleted and deletions are detected; false otherwise.

Creating JDBC ResultSet

- The ResultSet interface provides methods for retrieving and manipulating the results of executed queries, and ResultSet objects can have different functionality and characteristics.
- These characteristics are result set type, result set concurrency, and cursor holdability.
- A result set is created by executing a *query*, and the type of result set depends on the arguments that are supplied to the *Connection* methods; *createStatement (or prepareStatement or prepareCall).*

- JDBC provides following connection methods to create statements with desired ResultSet:
    1. **createStatement(int RSType, int RSConcurrency);**
    2. **prepareStatement(String SQL, int RSType, int RSConcurrency);**
    3. **prepareCall(String sql, int RSType, int RSConcurrency);**
- The first argument indicate the type of a ResultSet object and the second argument is one of two ResultSet constants for specifying whether a result set is read-only or updatable.
- The type of a ResultSet object determines the level of its functionality in two areas: the ways in which the cursor can be manipulated, and how concurrent changes made to the underlying data source are reflected by the ResultSet object.
- The *sensitivity* of a ResultSet object is determined by one of three different ResultSet types:
    - TYPE_FORWARD_ONLY,
    - TYPE_SCROLL_INSENSITIVE and
    - TYPE_SCROLL_SENSITIVE.
- The concurrency of a ResultSet object determines what level of update functionality is supported.
- There are two concurrency levels:
    - CONCUR_READ_ONLY.
    - CONCUR_UPDATABLE.

Sample code Fragment

Uses only JDBC 1.0 API, supplies no arguments to the method *createStatement* and thus creates a default *ResultSet* object, one that is forward-only and uses read-only concurrency.

```
Connection con = DriverManager.getConnection (“jdbc: my_subprotocol:my_subname");
Statement stmt = con.createStatement ();
ResultSet rs = stmt.executeQuery (“SELECT Purchase_ID, Amount FROM Purchase”);
```

✍ Task 6 : Explore the API

6.1 Identify the constant indicating the concurrency mode for a *ResultSet* object that may NOT be updated.

6.2 Identify the constant value of the constant indicating the concurrency mode for a ResultSet object that may be updated.

6.3 Retrieves the value of the designated column in the current row of this *ResultSet* object as an *int*. Identify the method which returns the column value and if the value is SQL NULL, the value returned is 0.

6.4 Retrieves the value of the designated column in the current row of this *ResultSet* object as a *String*. Identify the method which returns the column value and if the value is SQL NULL, the value returned is also null.

6.5 Identify the method which retrieves the value of the designated column in the current row of this *ResultSet* object as a byte array in the Java programming language.

Core Warning
The first column in a *ResultSet* row has index 1, not 0.

✍ Fill in Queries:

1. Complete the method declaration: **public <*return type*> getMetaData() throws <*Exception*>**

2. Method which moves the cursor to the given row number in this ResultSet object.

3. Statement which creates a scrollable result set that is sensitive to updates and that is updatable.

4. Complete the method declaration: **public <*return type*> getBigDecimal (*Parameter1*, *Parameter2*) throws <*Exception*>**

5. Method which returns the *Statement* object that produced this *ResultSet* object or null if the result set was produced some other way

Navigating the JDBC ResultSet

- A *ResultSet* object maintains a *cursor*, which points to its current row of data. The cursor moves down one row each time the method next is called.
- When a ResultSet object is first created, the cursor is positioned before the first row, so the first call to the *next* method puts the cursor on the first row, making it the current row.
- ResultSet rows can be retrieved in sequence from top to bottom as the cursor moves down one row with each successive call to the method next.
- When a cursor is positioned on a row in a ResultSet object (not before the first row or after the last row), that row becomes the current row.

- A scrollable result set's cursor can move both forward and backward as well as to a particular row.
- To move the cursor, can use the following methods of ResultSet object:
  - previous, first, last, absolute, relative, afterLast, and beforeFirst.

Core Note

A cursor remains valid until the *ResultSet* object or its parent *Statement* object is closed.

✍ Task 7 : Explore the API

7.1 Method which returns true if the cursor is after the last row; false if the cursor is at any other position or the *ResultSet* contains no rows.
7.2 Method which moves the cursor to the front of this *ResultSet* object, just before the first row. This method has no effect if the *ResultSet* contains no rows.
7.3 Method which returns true if the cursor is on the result set; false otherwise.
7.4 Method which returns int and retrieves the current row number.
7.5 Method which moves the cursor to a special row in the result set that can be used to insert a new row into the database.

Task 5 Workspace

Task 6 Workspace

Task 7 Workspace

Core Note

ResultSet and *ResultSetMetaData* do not directly provide a method to return the number of rows returned from a query. However, in JDBC 2.0, you can position the cursor at the last row in the ResultSet by calling last, and then obtain the current row number by calling *getRow*.

Viewing JDBC ResultSet

- The ResultSet interface provides *getter* methods (getBoolean, getLong, and so on) for retrieving column values from the current row.
- Values can be retrieved using either the index number of the column or the name of the column. In general, using the column index will be more efficient.
- For the *getter* methods, a JDBC driver attempts to convert the underlying data to the Java type specified in the getter method and returns a suitable Java value.
- Column names used as input to *getter* methods are case insensitive.
- The ResultSet interface contains dozens of methods for getting the data of the current row.
- There is a get method for each of the possible data types, and each get method has two versions:
    1. One that takes in a column name.
    2. One that takes in a column index.
- The *ResultSet.getXXX* methods provide the means for retrieving column values from the current row.

Sample code Fragment

Here is an example that prints the values of the first two columns and the Reservation_id and User_id, for all rows of a ResultSet.

```
while (resultSet.next ())
        {
                System.out.println(resultSet.getInt(1) + " " +
                resultSet.getInt(User_id) + " " +
                resultSet.getDouble("Credit_Card_number") + " "
        }
```

- Similarly there are get methods in the ResultSet interface for each of the eight Java primitive types, as well as common types such as java.lang.String, java.lang.Object, and java.net.URL
- When a *getXXX* method is invoked, the driver attempts to convert the underlying data to the type XXX in the Java programming language and then returns a suitable value.

✍ Fill in Queries:

1. Complete the method declaration: **public <*return type*> getInt (*Parameter1*) throws <*Exception*>**

2. Complete the Statement object for creating sensitive and read only ResultSet

   Statement stmt =

3. Create a scrollable, insensitive, read-only *ResultSet* with a cursor that is not holdable.

4. Complete the method declaration: **public <*return type*> getDouble (*Parameter1*) throws <*Exception*>**

5. The method **DatabaseMetaData . <*method name*>(*Parameter1*)** will return true if newly inserted rows can be seen in result sets of the specified type.

Updating JDBC ResultSet

- Updating a row in a ResultSet object is a two-phase process. First, the new value for each column being updated is set, and then the change is applied to the row.
- The row in the underlying data source is not updated until the second phase is completed.
- A ResultSet object may be updated (have its rows modified, inserted, or deleted) programmatically if its concurrency type is *CONCUR_UPDATABLE*.
- The new *updateXXX* methods make it possible to update values in a result set without using SQL commands.
- The *updateXXX* methods take two parameters, the first to indicate which column is to be updated, and the second to give the value to assign to the specified column.
- If an update method is called on a ResultSet whose concurrency level is ResultSet.CONCUR_READ_ONLY, then a *SQLException* must be thrown.
- It is very important that the *updateRow ()* be called while the cursor is still on the current row (the row to be updated).

Core Note

The updateXXX methods update a value in the current row of the result set, but they do not update the value in the underlying database table. It is the method *updateRow* that updates the database.

✍ Task 8 : Explore the API

8.1 Identify the method which returns an array of update counts containing one element for each command in the batch.

8.2 Identify the method which an application can call to explicitly cancel the updates to a row.

8.3 Identify the method which throws *BatchUpdateException* if one of the commands sent to the database fails to execute properly or attempts to return a result set.

8.4 Identify the method which updates the underlying database with the new contents of the current row of this ResultSet object.

8.5 Identify the method which adds the insert row to both the result set and database, may throw a *SQLException* if the number of columns in the insert row does not match the number of columns in the database table.

✍ Fill in Queries:

1. Method which retrieves the vendor-specific exception code for this SQLException object.

2. Complete the method declaration: **public *<return type>* updateString(*Parameter1* , *Parameter2*) throws <*Exception*>**

3. Method which gives a hint as about the direction in which the rows in this *ResultSet* object will be processed.

4. Complete the method declaration: **public *<return type>* updateBoolean (*Parameter1* , *Parameter2*) throws <*Exception*>**

5. Complete the method declaration: **public *<return type>* updateBytes(*Parameter1* , *Parameter2*) throws <*Exception*>**

Task 8 Workspace

✍ Code 1 : Test the Coding Skills

Code the Program

```
//SimpleResultSetTest.java
package jdbcdemo;

/* Following is the example which makes use of few getXXX methods described in
   ResultSet.*/
//import all the required classes and interfaces;



public class SimpleResultSetTest {
  // Oracle driver name and database URL
     static final String ORACLE_DRIVER = "_____";

     static final String DB_URL = "_____";

  //Database credentials
  static final String USER = "username";
  static final String PASS = "password";

public static void main(_______ . _______ .String[] args) {
  Connection conn = null;
  Statement stmt = null;
  try{
    //Register JDBC driver
    Class.forName("_____");

    //Open a connection
    System.out.println("Connecting to database...");
    conn = DriverManager._____(_____,_____,_____);

    //Execute a query to create ResultSet which is sensitive and updatable.
    System.out.println("Creating statement...");
    stmt = _____._____( _____,_____);

    String sql;
    sql = "_____"; //Query to get Reservation details

    ResultSet rs = stmt._____(_____);

    // Move cursor to the last row.
    System.out.println("Moving cursor to the last...");

    _____ . _____();
```

Code the Program…Continued

```
//Extract data from result set
    System.out.println("Displaying record...");
While( _____ . _________________){
    //Retrieve by column name
    int id          = _____ . ___________________("Reservation_id");
    int age         = _____ . ___________________("Flight_id");
    String userid   = _____ . ___________________("User_id");    }

//Clean-up environment.
_____;

_____;

 }
catch(_____ se) {
    //Handle errors for JDBC
    se._____(); }
  }
} // End of Class.
```

✍ Code 2 : Test the Coding Skills

Code the Program

```
//Inserting a row into a Passenger database table using an updatable ResultSet.
try {
   // Create an updatable result set
   Statement stmt = connection.createStatement(_____);
   ResultSet resultSet = _____ . _____("SQL Select Query       ");

   // Move cursor to the insert row.
   resultSet._____();

   // Set values for the new row.
   resultSet._____("_____", "_____");

   // Insert the row
   resultSet._____();
} catch (SQLException e) { _____;
}
```