JDBC

Executing Query & Processing Results

WIPRO
Applying Thought

Executing Query & Processing Results

Agenda
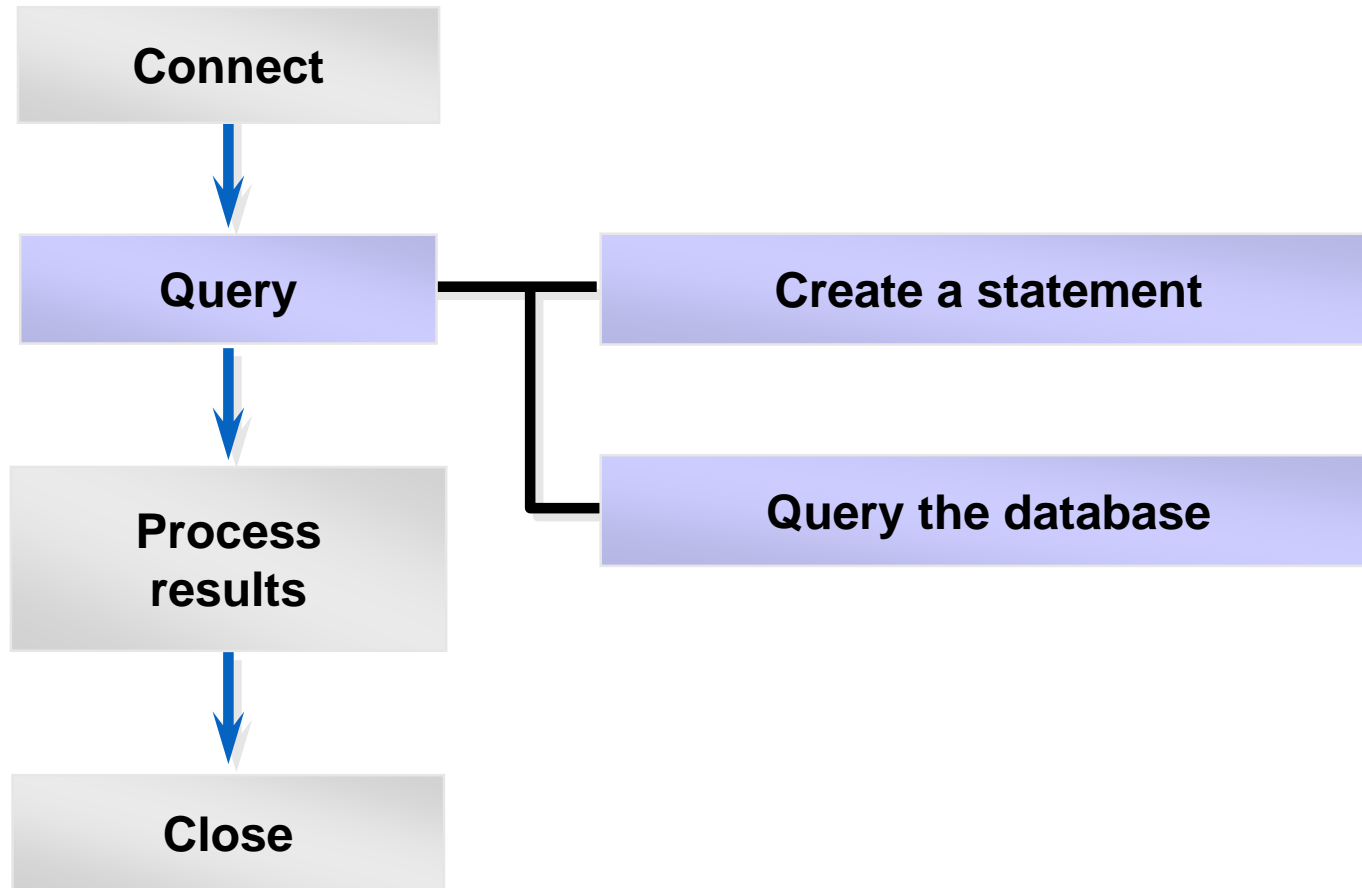
1   **Executing Query**

2   **Process Result**

Objectives

At the end of this module, you will be able to:

- Create and execute a query using JDBC API

- Process the data returned by the database

Stage 2: Query

```
┌─────────────────┐
│     Connect     │
└─────────────────┘
         │
         ▼
┌─────────────────┐        ┌──────────────────────────┐
│      Query      │────────│    Create a statement    │
└─────────────────┘        └──────────────────────────┘
         │                  ┌──────────────────────────┐
         ▼                  │    Query the database    │
┌─────────────────┐         └──────────────────────────┘
│     Process     │
│     results     │
└─────────────────┘
         │
         ▼
┌─────────────────┐
│      Close      │
└─────────────────┘
```

Stage 2: Query

- Once a connection is established, it is used to pass SQL statements to its underlying database.

- A Statement object is used to send SQL statements to a database. The Statement interface provides basic methods for executing statements and retrieving results.

- The JDBC API does not put any restrictions on the kinds of SQL statements that can be sent; this provides a great deal of flexibility, allowing the use of database-specific statements or even non-SQL statements. It requires, however, that the user be responsible for making sure that the underlying database can process the SQL statements being sent and suffer the consequences if it cannot. For example, an application that tries to send a stored procedure call to a DBMS that does not support stored procedures will be unsuccessful and will generate an exception.

Query: The Statement Object

- To execute SQL statements use Statement Object.

- You need an active connection to create a JDBC   statement

- Statement object has three methods to execute a   SQL statements:

  - executeQuery() for SELECT statements
  - executeUpdate()for   INSERT,   UPDATE,   DELETE,   or   DDL statements
  - execute() for either type of statement

How to Query the Database?

1. To execute SQL statement , we should first create Statement object, as:

    > Statement *stmt* = *conn*.createStatement();

2. To execute the query on the database

    > ResultSet *rset* = *stmt*.executeQuery(*statement*);
    >
    > int *count* = *stmt*.executeUpdate(*statement*);
    >
    > boolean *isquery* = *stmt*.execute(*statement*);

How to Query the Database?

- Once a connection to a particular database is established, that connection can be used to send SQL statements. A Statement object is created with the Connection method createStatement, as in the following code fragment:

- Statement stmt = conn.createStatement();

- The SQL statement that will be sent to the database is supplied as the argument to one of the execute methods on a Statement object.

- This is demonstrated in the following example, which uses the method executeQuery:

- ResultSet rset = stmt.executeQuery("SELECT a, b, c FROM Table2");

- The variable rset references a result set discussed in the following sections.

Querying the Database: Examples

- Following Statements are used to execute Select statement:

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

ResultSet rset = stmt.executeQuery

  ("select NAME, VERTICAL from STUDENT");
```

- Following Statements are used to execute Select statement:

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

int rowcount = stmt.executeUpdate

  ("delete from STUDENT  where ID = 1000");
```

Querying the Database

- As mentioned earlier, the Statement interface provides three different methods for executing SQL statements: executeQuery, executeUpdate, and execute. The correct method to use is determined by what the SQL statement produces.

- The method executeQuery() method is used for statements that produce a result set, such as SELECT statements.

- The method executeUpdate() method is used to execute DML statements and also SQL DDL (Data Definition Language) statements. The return value of executeUpdate() method is an integer (referred to as the update count) that indicates the number of rows that were affected. For statements such as CREATE TABLE or DROP TABLE, which do not operate on rows, the executeUpdate() method is returns always zero.

- The method execute is used to execute statements that return more than one result set, more than one update count, or a combination of the two.

Query: Other Methods

- JDBC provides two other statement objects:
- **PreparedStatement**, for precompiled SQL statements, is covered later.
- **CallableStatement**, for statements that execute stored procedures, is also covered later.

- **Objects and Interfaces**
- **java.sql.Statement** is an interface, not an object. When you declare a Statement object and initialize it using the **createStatement()** method, you are creating the implementation of the Statement interface supplied by the Oracle driver or any other driver that you are using.

Process the Results

Now that we have obtained the results of querying the database in the ResultSet object, we need to iterate through this object and retrieve its contents for further processing in the Java program.

Connect

Query

Process results

Close

Step through the results

Assign results to Java variables
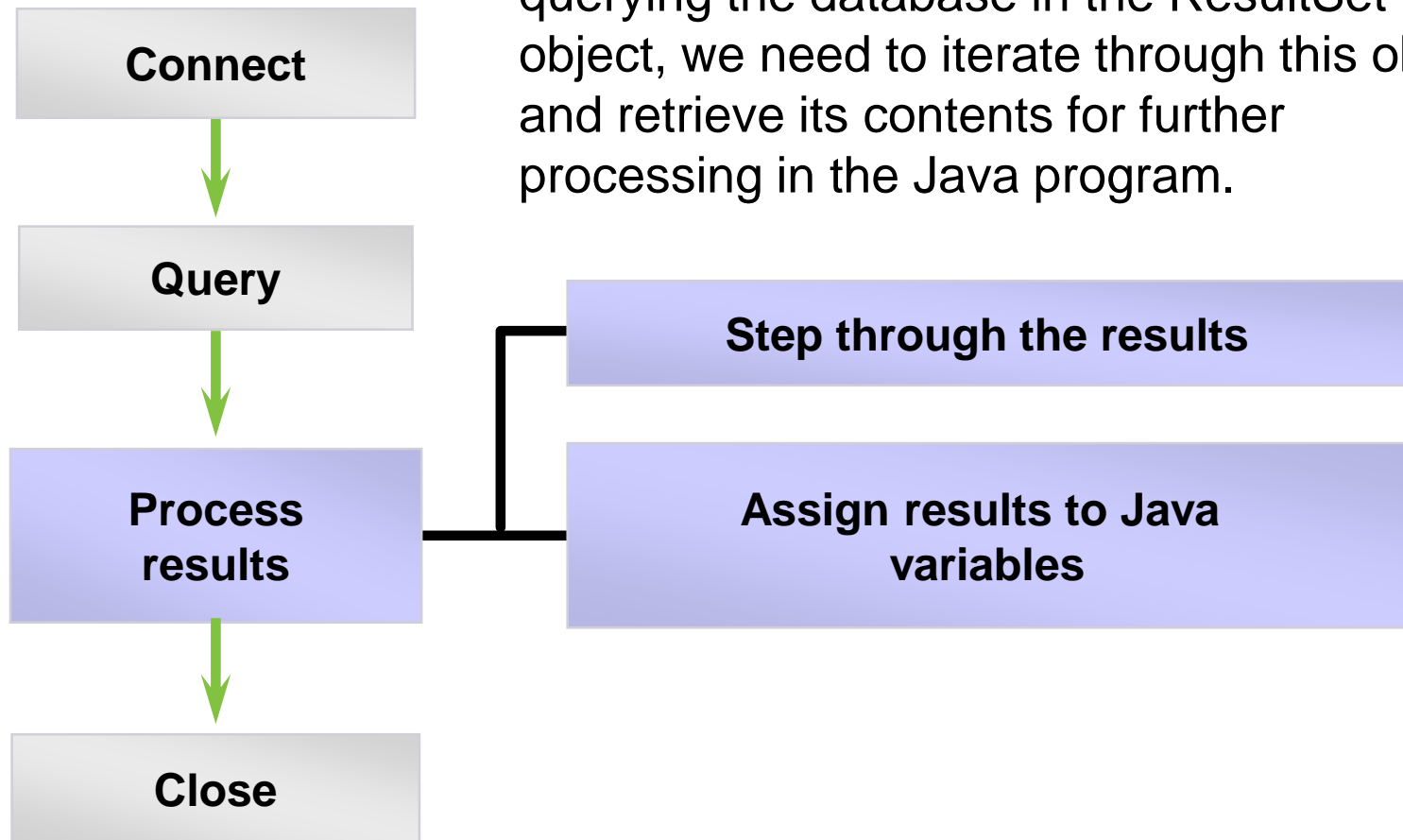
Process the Results: The ResultSet Object

ResultSet is an object that contains the results of executing a SQL statement

A ResultSet maintains a cursor pointing to its current row of data

Use next() to step through the result set row by row

To retrieve the data from the columns, we can use getXXX() method

Process the Results: The ResultSet Object

The results are available in ResultSet object. In other words, it contains the rows that satisfy the conditions of the query.

The data stored in a ResultSet object is retrieved through a set of get methods that allows access to the various columns of the current row.

The ResultSet.next() method moves the cursor to the next row in the ResultSet object.

The general form of a result set is a table with column headings and the corresponding values returned by a query.

Process the Results: The ResultSet Object

For example, if your query is SELECT column_a, column_b, column_c FROM Table1, your result set will have the following form:

Column_a	column_b	column_c
---------	-----------	----------
12345	Cupertino	2459723.495
83472	Redmond	1.0
83492	Boston	35069473.43

We can retrieve the results using the methods of ResultSet interface. Each of these get*XXX*() methods attempts to convert the column value to the specified Java type and returns a suitable Java value. For example, getInt() method gets the column value as an int value, getString() method gets the column value as a String value, and geLong() method returns the column value as a long value.

How to Process the Result?

```
while (rset.next()) { … }
```

```
String val =
rset.getString(colname);
```

```
String val =
rset.getString(colIndex);
```

```
while (rset.next()) {
  String name = rset.getString("NAME");
  String supervisor = rset.getString("SUPERVISOR");
  … // Process or display the data
}
```

How to Process the Result?

- 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. This means that any methods called while the cursor is positioned on that row will operate on values in that row (methods such as getXXX).

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

Setting up classpath

Before we can execute any JDBC program, we have to set the classpath for the type IV driver. We need to know, what is the name of the jar file, which contains type IV driver. The jar file, which we will be using is *classes12.jar.* This jar file is found within jlib folder under oracle folder hierarchy.

For e.g., In my system, classes12.jar is within the following path :
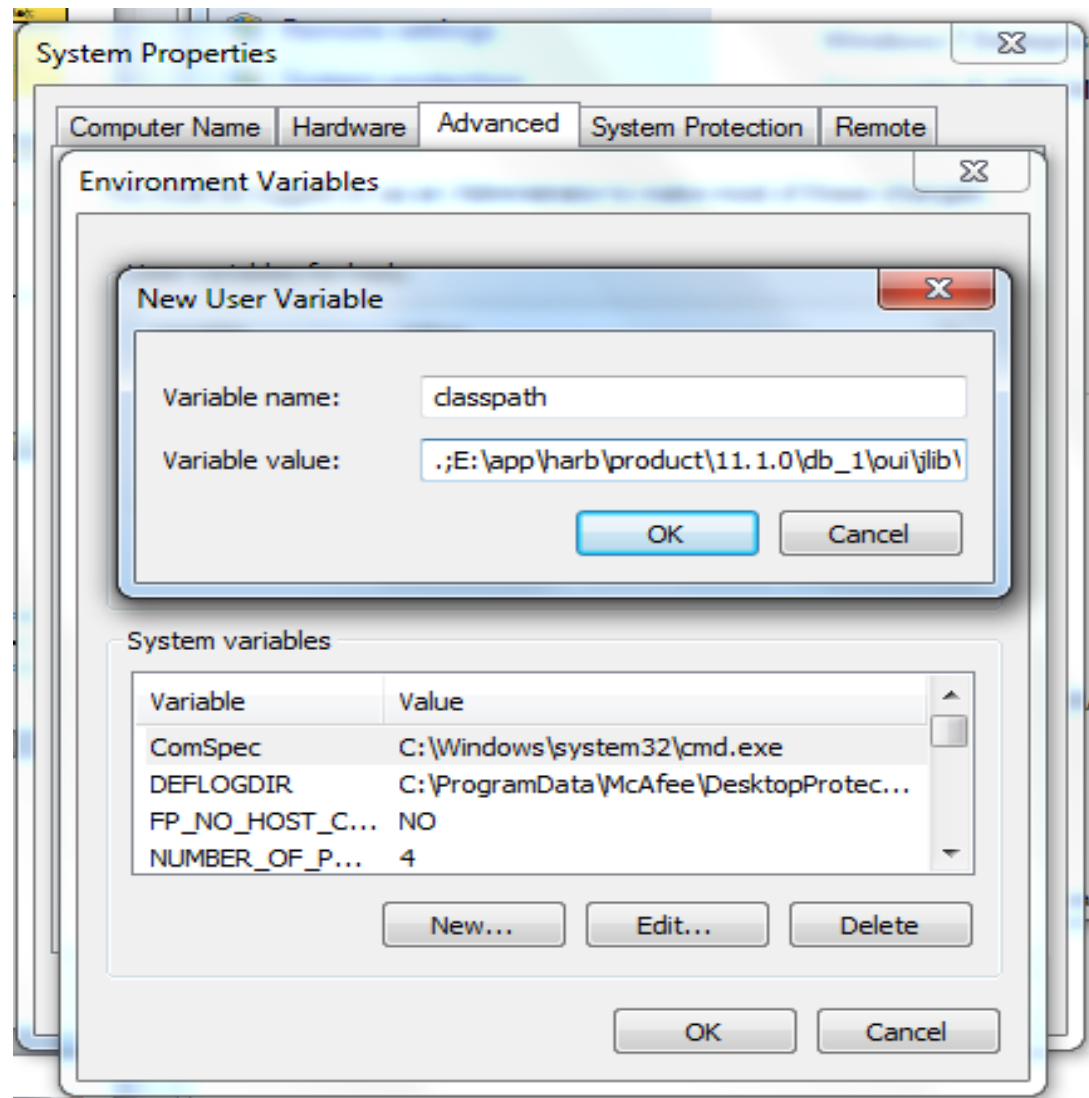
E:\app\harb\product\11.1.0\db_1\oui\jlib\classes.jar

Add the above path(including classes12.jar) to the classpath that you have already set. If you don't have classpath set in your system, create a new enviroment variable and add the following as its value :

Variable name : **classpath**
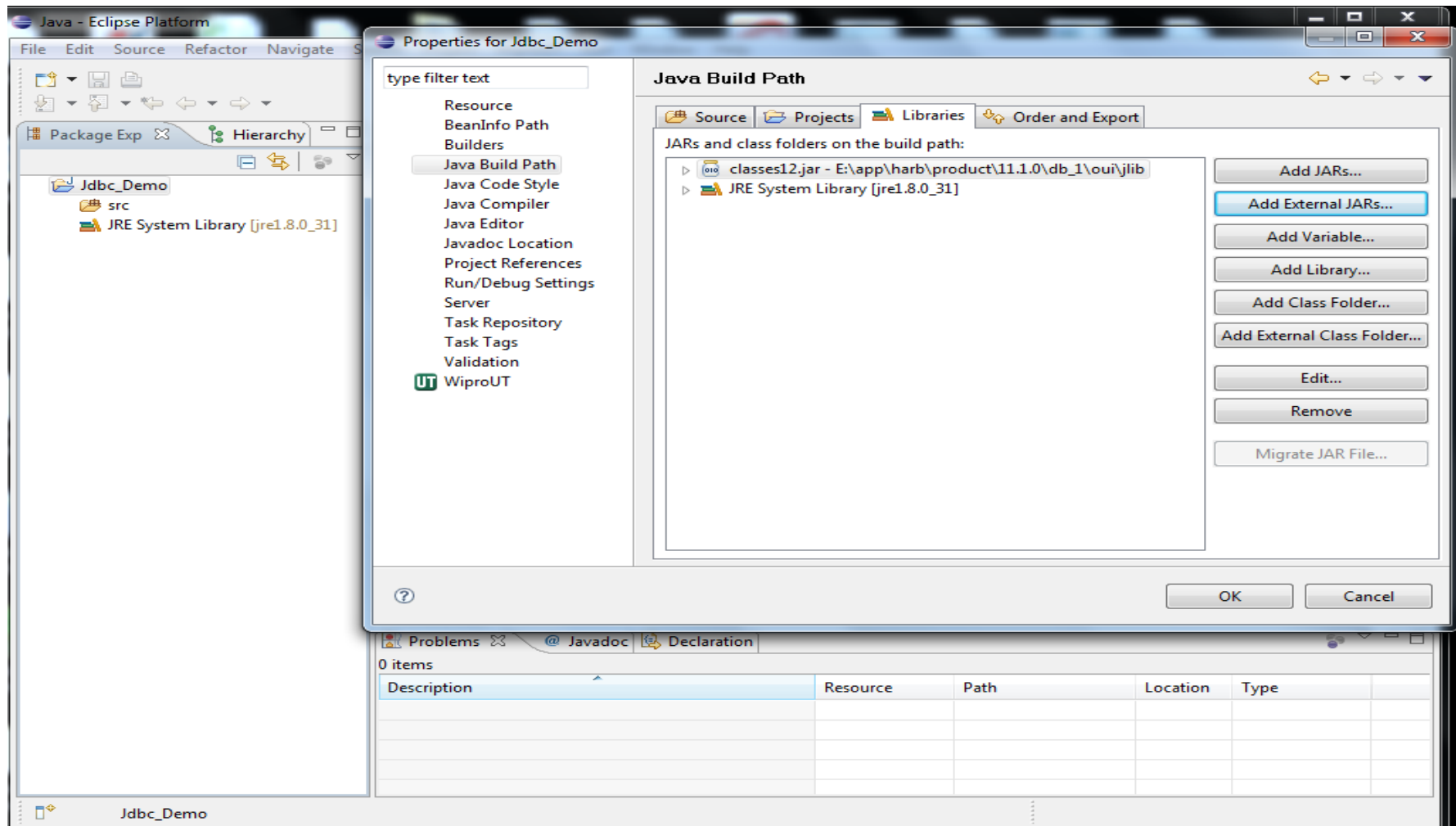
Variable value : .;
   E:\app\harb\product\11.1.0\db_1\oui\jlib\classes12.jar;

Setting up classpath (Contd.).

Setting up classpath (Contd.).

If you are using eclipse IDE, then it will not recognize the environment variable, classpath. You will have to add classes12.jar, as an external jar file to the build path.

Example

```
import java.sql.*;
class MakeConnection1{
  Connection con;
   Statement stmt;
   ResultSet rs;
   MakeConnection1(){
   try{
        Class.forName("oracle.jdbc.driver.OracleDriver");
      con=DriverManager.getConnection
("Jdbc:Oracle:thin:@localhost:1521:ORCL","scott","tiger");
      stmt=con.createStatement();
      rs=stmt.executeQuery("Select ename, sal from emp");
      while(rs.next())
        System.out.println(rs.getString(1)+" "+rs.getInt(2));
```
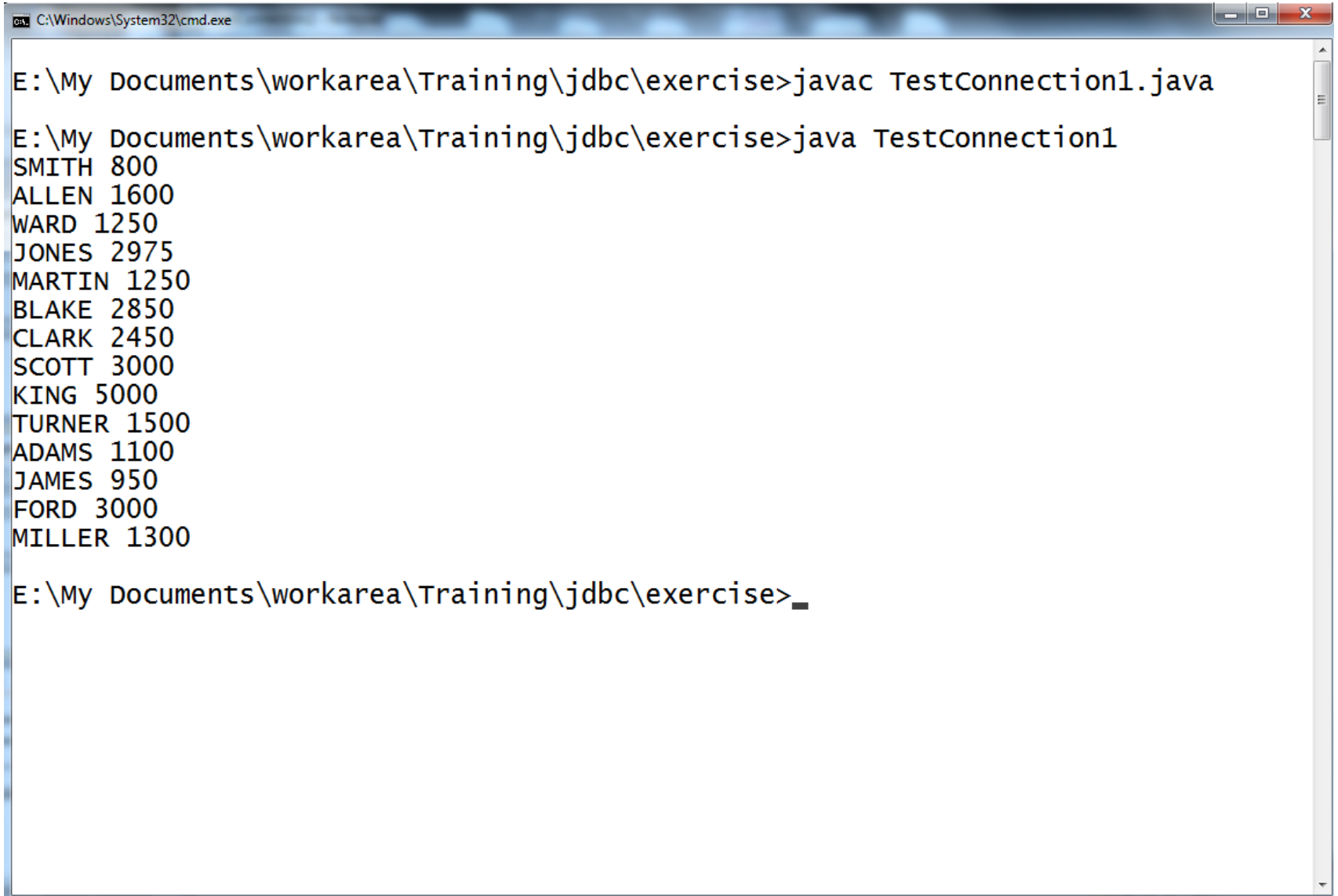
contd..

```
   conn.close();
  } // end of try block
  catch(SQLException e){
    System.out.println(e);
  }
  catch(ClassNotFoundException e){
    System.out.println(e);
  }

 }
}
public class TestConnection1{
  public static void main(String args[]){
      new MakeConnection1();
  }
}
```

Example (Contd.).


```
C:\Windows\System32\cmd.exe

E:\My Documents\workarea\Training\jdbc\exercise>javac TestConnection1.java

E:\My Documents\workarea\Training\jdbc\exercise>java TestConnection1
SMITH 800
ALLEN 1600
WARD 1250
JONES 2975
MARTIN 1250
BLAKE 2850
CLARK 2450
SCOTT 3000
KING 5000
TURNER 1500
ADAMS 1100
JAMES 950
FORD 3000
MILLER 1300

E:\My Documents\workarea\Training\jdbc\exercise>_
```

24

Quiz

1. **What does the next() method invoked on the result set object return**

   a) String

   b) int

   c) boolean

   d) float

2. **Which one of the following method can be invoked on the connection object to create an empty Statement object**

   a) createStatement();

   b) getStatement();

   c) prepareStatement();

   d) executeStatement();

   .

Answers :

1 : c

2 : a

How to handle SQL Null values?

Java primitive types cannot have null values

Do not use a primitive type when your query might return a SQL null

Use ResultSet.wasNull() to determine whether a column has a null value

```
while (rset.next()) {
    String year = rset.getString("YEAR");
    if (rset.wasNull() {
        … // Handle null value}
…}
```

How to handle SQL Null values?

- To determine if a given result value is JDBC NULL, one must first read the column and then use the method ResultSet.wasNull() method. This is true because a JDBC NULL retrieved by one of the ResultSet.getXXX methods may be converted to either null, 0, or false, depending on the type of the value.

- The following getXXX() methods will retrieve values are:
  - null - for those getXXX methods that return objects in the Java programming language (getString, getBigDecimal, getBytes, getDate, getTime, getTimestamp, getAsciiStream, getObject, getCharacterStream, getUnicodeStream, getBinaryStream, getArray, getBlob, getClob, and getRef)

# How to handle SQL Null values?

- 0 (zero) - for getByte, getShort, getInt, getLong, getFloat, and getDouble

- false - for getBoolean

- For ex., if the method getInt returns 0 from a column that allows null values, an application cannot know for sure whether the value in the database was 0 or NULL until it calls the method wasNull, as shown in the following code fragment, where rs is a ResultSet object.

  int n = rs.getInt(3); boolean b = rs.wasNull();

- If b is true, the value stored in the third column of the current row of rs is JDBC NULL. The method wasNull checks only the last value retrieved, so to determine whether n was NULL, wasNull had to be called before another getXXX method was invoked.

Close Connection
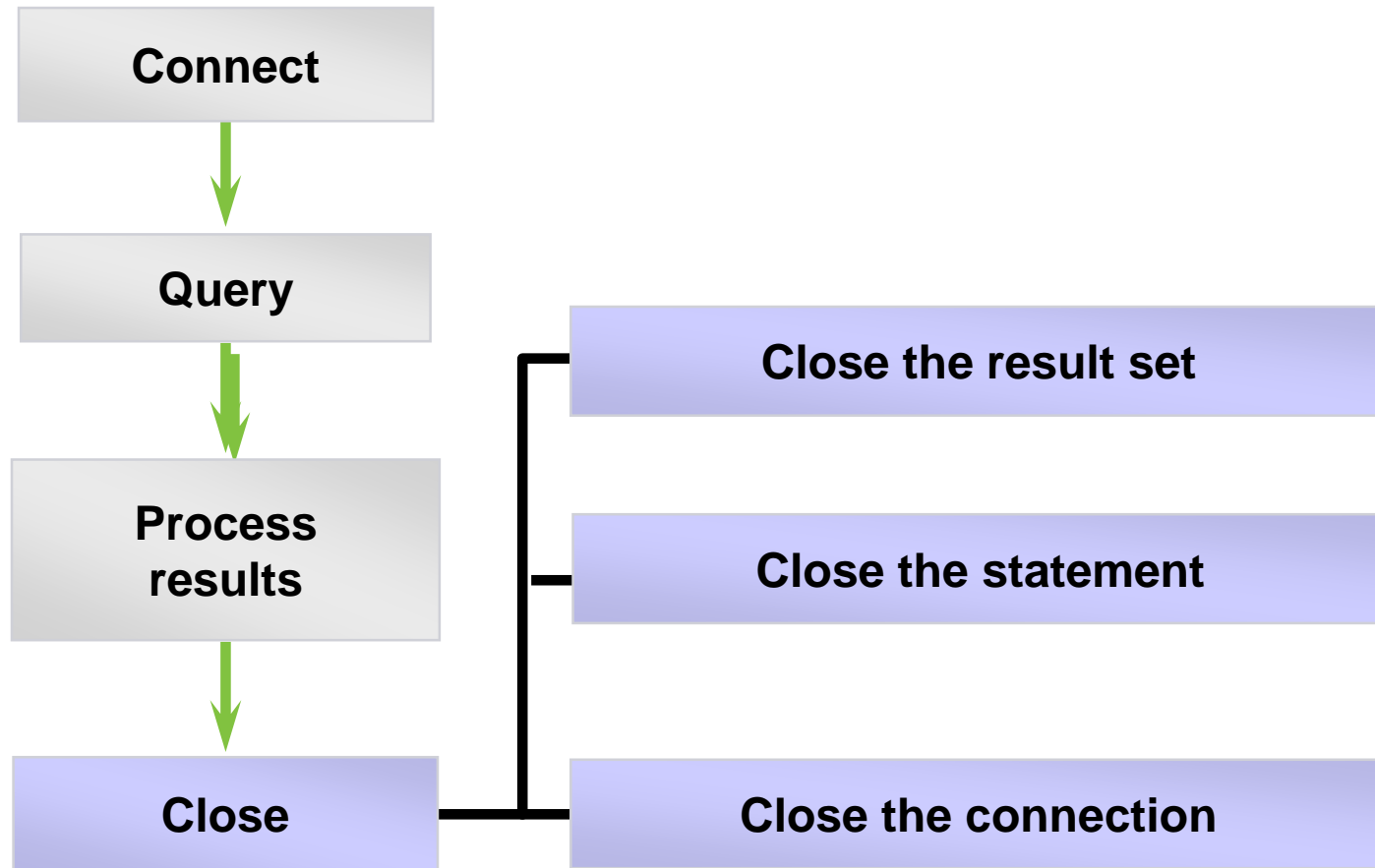
Close Connection

- Normally, nothing needs to be done to close a ResultSet object; it is automatically closed by the Statement object that generated it when that Statement object is closed, is re-executed, or is used to retrieve the next result from a sequence of multiple results. The method close is provided so that a ResultSet object can be closed explicitly, thereby immediately releasing the resources held by the ResultSet object. This could be necessary when several statements are being used and the automatic close does not occur soon enough to prevent database resource conflicts.

How to Close the Connection?

1. **Close the ResultSet object**

   ```
   rset.close();
   ```

2. **Close the `Statement` object**

   ```
   stmt.close();
   ```

3. **Close the connection**

   ```
   conn.close();
   ```

How to Close the Connection?

- When a connection is in auto-commit mode, the statements being executed within it are committed or rolled back when they are completed. A statement is considered complete when it has been executed and all its results have been returned.

- For the method **executeQuery**, which returns one result set, the statement is completed when all the rows of the **ResultSet** object have been retrieved.

- For the method **executeUpdate**, a statement is completed when it is executed. In the rare cases where the method execute is called, however, a statement is not complete until all of the result sets or update counts it generated have been retrieved.

How to Close the Connection?

- **Statement** objects will be closed automatically by the Java garbage collector. Nevertheless, it is recommended as good programming practice that they be closed explicitly when they are no longer needed. This frees DBMS resources immediately and helps avoid potential memory problems.

- The same connection object can be used to execute multiple statements and retrieve many result sets. However, once all the work with the database is over, it is a good programming practice to close the connection explicitly. If this is not closed, after a particular timeout period defined at the database, the connection is automatically closed. Nevertheless, this would mean that till the timeout, this connection is not available to any other users of the database. Hence, explicit closing of the connection is recommended.

Quiz

1. Which of the following methods is not used for querying database?
   a. executeQuery()
   b. executeStatement()
   c. executeUpdate()
   d. execute()

2. Which of the method is used to close a connection?
   a. connection.close();
   b. connection.terminate();
   c. connection.exit();
   d. None of the above

Answers:
1. B
2. A

Summary

In this module, you were able to:

- – Create and execute a query using JDBC API

- – Process the data returned by the database

Thank you