DAY 3

Close the connections

So finally we have sent the data to the specified location and now we are on the verge of completing our task.

By closing the connection, objects of Statement and ResultSet will be closed automatically. The close() method of the Connection interface is used to close the connection.

```
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 Demoj {

     public static void main(String[] args)  {
          // TODO Auto-generated method stub
          String url = "jdbc:mysql://localhost:3306/employee";
          String un = "root";
          String pwd = "root";
          Statement stmt = null;
          ResultSet res = null;
          Connection con = null;

          try {
               Class.forName("com.mysql.cj.jdbc.Driver");
               System.out.println("Driver successfully
loaded");
               con=DriverManager.getConnection(url, un, pwd);
               System.out.println("Connection established");

stmt=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,0);
               String query="select * from emp";
               res = stmt.executeQuery(query);

               ResultSetMetaData metaData = res.getMetaData();
```

```
                // to get the number of column in the table
                System.out.println(metaData.getColumnCount());
                //to get all column names of the table
                for(int i=1;i<metaData.getColumnCount();i++) {

System.out.println(metaData.getColumnName(i)+"
"+metaData.getColumnTypeName(i));
                }

           } catch (ClassNotFoundException  e) {
                e.printStackTrace();
           }
           catch(SQLException e) {
                e.printStackTrace();
           }


           try {
                res.close();
                stmt.close();
                con.close();
           } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
           }

     }

}
```

Insert Data into the database:

String query="insert into emp(`id`,`name`,`desig`) values (5,'jo','sme')";
System.out.println(stmt.execute(query));
stmt.execute(query) -You will get a result set when you give a select query not when insert or update, you will always get false in case of this type of query.

```
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 Demoj {

	public static void main(String[] args)  {
		// TODO Auto-generated method stub
		String url = "jdbc:mysql://localhost:3306/employee";
		String un = "root";
		String pwd = "root";
		Statement stmt = null;
		ResultSet res = null;
		Connection con = null;

		try {
			Class.forName("com.mysql.cj.jdbc.Driver");
			System.out.println("Driver successfully loaded");

			con=DriverManager.getConnection(url, un, pwd);
			System.out.println("Connection established");


stmt=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,0);

			String query="insert into emp(`id`,`name`,`desig`)
values (5,'jo','sme')";

			System.out.println(stmt.execute(query));

			System.out.println("Query executed successfully");



		} catch (ClassNotFoundException  e) {
			e.printStackTrace();
		}
```

```
            catch(SQLException e) {
                  e.printStackTrace();
            }


            try {
                  stmt.close();
                  con.close();
            } catch (SQLException e) {
                  e.printStackTrace();
            }

      }

}
```

Batch Processing in JDBC

What if we want to insert 10000 data into the database that time it is not efficient with the above process. Instead of executing a single query, we can execute a batch (group) of queries. It makes the performance fast.

Using Batch File U can insert all queries at a time.

```
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 Demoj {

      public static void main(String[] args)  {
            // TODO Auto-generated method stub
            String url="jdbc:mysql://localhost:3306/employee";
            String un="root";
            String pwd="root";
            Statement stmt=null;
            ResultSet res=null;
            Connection con=null;
```

```
            try {
                  Class.forName("com.mysql.cj.jdbc.Driver");
                  System.out.println("Driver sucessfully loaded");

                  con=DriverManager.getConnection(url, un, pwd);
                  System.out.println("Connection established");

                  stmt=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,0);

                  String query="insert into emp(`id`,`name`,`desig`) values
(7,'rob','sme')";
                  String query1="insert into emp(`id`,`name`,`desig`) values
(8,'rob','sme')";
                  String query2="insert into emp(`id`,`name`,`desig`) values
(9,'rob','sme')";

                  stmt.addBatch(query);
                  stmt.addBatch(query1);
                  stmt.addBatch(query2);

                  stmt.executeBatch();

                  System.out.println("Query executed sucessfully");



            } catch (ClassNotFoundException  e) {
                  e.printStackTrace();
            }
            catch(SQLException e) {
                  e.printStackTrace();
            }


            try {
                  stmt.close();
                  con.close();
            } catch (SQLException e) {
                  e.printStackTrace();
            }

      }

}
```

Prepared Statement:

In real-time, the data that is entered in the front-end should be extracted by a java program and java should insert the query into the database.

In database management systems (DBMS), a **prepared statement** or **parameterized statement** is a feature used to execute the same or similar database statements repeatedly with high efficiency. Typically used with SQL statements such as queries or updates, the prepared statement takes the form of a template into which certain constant values are substituted during each execution.

The typical workflow of using a prepared statement is as follows:

1. **Prepare**: At first, the application creates the statement template and sends it to the DBMS. Certain values are left unspecified, called *parameters*, *placeholders,* or *bind variables* (labeled "?" below):
   `INSERT INTO products (name, price) VALUES (?, ?);`
2. Then, the DBMS compiles (parses, optimizes, and translates) the statement template, and stores the result without executing it.
3. **Execute**: At a later time, the application supplies (or *binds*) values for the parameters of the statement template, and the DBMS executes the statement (possibly returning a result).

Now we don't have front end interaction using scanner we will take input and we will set the values as shown below:

pstmt = con.prepareStatement(query);

```
Scanner s =new Scanner(System.in);
int n = s.nextInt();
String s1 = s.next();
String s2 = s.next();
pstmt.setInt(1, n);
pstmt.setString(2, s1);
pstmt.setString(3, s2);
pstmt.execute();
```

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;
```

```
public class Demoj {

	public static void main(String[] args)  {
		// TODO Auto-generated method stub
		String url="jdbc:mysql://localhost:3306/employee";
		String un="root";
		String pwd="root";
		PreparedStatement pstmt=null;
		ResultSet res=null;
		Connection con=null;

		try {
			Class.forName("com.mysql.cj.jdbc.Driver");
			System.out.println("Driver successfully loaded");

			con=DriverManager.getConnection(url, un, pwd);
			System.out.println("Connection established");
			String query ="insert into emp(`id`,`name`,`desig`) values
(?,?,?)";

			pstmt = con.prepareStatement(query);

			Scanner s =new Scanner(System.in);
			int n = s.nextInt();
			String s1 = s.next();
			String s2 = s.next();
			pstmt.setInt(1, n);
			pstmt.setString(2, s1);
			pstmt.setString(3, s2);
			pstmt.execute();
			System.out.println("Query executed");


		} catch (ClassNotFoundException  e) {
			e.printStackTrace();
		}
		catch(SQLException e) {
			e.printStackTrace();
		}


		try {
			pstmt.close();
			con.close();
		} catch (SQLException e) {
```

```
                    e.printStackTrace();
            }

      }

}
```