**JDBC TUTORIALS**

JDBC stands for Java Database Connection. It provides portable access to different databases. It enables us to build our own custom queries.

Statement.executeQuery() – only for select statement

Statement.executeUpdate() – for Crud Operations

* **boolean execute (String SQL)**:
  + Returns a boolean value of true if a ResultSet object can be retrieved; otherwise, it returns false. Use this method to execute SQL DDL statements or when you need to use truly dynamic SQL.
* **int executeUpdate (String SQL)**
  + Returns the number of rows affected by the execution of the SQL statement. Use this method to execute SQL statements for which you expect to get a number of rows affected - for example, an INSERT, UPDATE, or DELETE statement.
* **ResultSet executeQuery (String SQL)**
  + Returns a ResultSet object. Use this method when you expect to get a result set, as you would with a SELECT statement.

**Prepared Statements:**

A prepared statement is a precompiled SQL statement. It provides the following benefits:

* Makes it easier to set parameter values
* Prevents against SQL dependency injection attacks
* May improve system performance.

Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)");

stmt.setInt(1,101);//1 specifies the first parameter in the query

stmt.setString(2,"Ratan");

**int** i=stmt.executeUpdate();

System.out.println(i+" records inserted");

**Stored Procedures :**

A group of SQL statements that can perform a particular task. It is usually created by DBA.It is created in SQL supported by native database. Can have any combination of input, output, input/output parameters.

// Basically Calling procedure.

Stored Procedure :

CREATE OR REPLACE PROCEDURE getEmpName

(EMP\_ID IN NUMBER, EMP\_FIRST OUT VARCHAR) AS

BEGIN

SELECT first INTO EMP\_FIRST

FROM Employees

WHERE ID = EMP\_ID;

END;

To call stored procedure from Java, JDBC api provides callable statements.

String SQL = "{call getEmpName (?, ?)}";

cstmt = conn.prepareCall (SQL);

**Batch Processing :**

Batch Processing allows you to group related SQL statements into a batch and submit them with one call to the database.

When you send several SQL statements to the database at once, you reduce the amount of communication overhead, thereby improving performance.

* JDBC drivers are not required to support this feature. You should use the *DatabaseMetaData.supportsBatchUpdates()* method to determine if the target database supports batch update processing. The method returns true if your JDBC driver supports this feature.
* The **addBatch()** method of *Statement, PreparedStatement,* and *CallableStatement* is used to add individual statements to the batch. The **executeBatch()** is used to start the execution of all the statements grouped together.
* The **executeBatch()** returns an array of integers, and each element of the array represents the update count for the respective update statement.
* Just as you can add statements to a batch for processing, you can remove them with the **clearBatch()** method. This method removes all the statements you added with the addBatch() method. However, you cannot selectively choose which statement to remove.

// Create statement object

Statement stmt = conn.createStatement();

// Set auto-commit to false

conn.setAutoCommit(false);

// Create SQL statement

String SQL = "INSERT INTO Employees (id, first, last, age) " +

"VALUES(200,'Zia', 'Ali', 30)";

// Add above SQL statement in the batch.

stmt.addBatch(SQL);

// Create one more SQL statement

String SQL = "INSERT INTO Employees (id, first, last, age) " +

"VALUES(201,'Raj', 'Kumar', 35)";

// Add above SQL statement in the batch.

stmt.addBatch(SQL);

// Create one more SQL statement

String SQL = "UPDATE Employees SET age = 35 " +

"WHERE id = 100";

// Add above SQL statement in the batch.

stmt.addBatch(SQL);

// Create an int[] to hold returned values

int[] count = stmt.executeBatch();

//Explicitly commit statements to apply changes

conn.commit();

**NamedQuery : HQL**

@org.hibernate.annotations.NamedQuery(name = "DeptEmployee\_findByEmployeeNumber", query = "from DeptEmployee where employeeNumber = :employeeNo")

**Named Native Query :**

@org.hibernate.annotations.NamedNativeQueries( @org.hibernate.annotations.NamedNativeQuery(name = "DeptEmployee\_GetEmployeeByName", query = "select \* from deptemployee emp where name=:name", resultClass = DeptEmployee.class) )