Batch Processing Using JDBC

- 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.
- addBatch() method of Statement and PreparedStatement is used to add set of sql queries.
- executeBatch() returns an array of integers, and each element of the array represents the "No. of rows affected"count for the respective "other than SELECT" type of SQL statements.
- 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.

Steps to use Batch processing using Statement

1. Create a Statement object using createStatement()
2. Add as many as SQL statements you like into batch using addBatch() method on created Statement object.
3. Execute all the SQL statements using executeBatch() method on created statement object.
4. then remove the batch by invoking clearBatch().

Example:

```
public class BatchDemo {
	public static void main(String[] args)
	{
		Connection con = null;
		Statement stmt = null;
		try {
			Class.forName("com.mysql.jdbc.Driver");

			String dburl =
"jdbc:mysql://localhost:3306/hejm15_db?user=root&password=root";
			 con = DriverManager.getConnection(dburl);

			 String query1 = "insert into student_info
values(1,'sunil','kumar','c')";
			String query2 = "insert into guardian_info
values(1,'chandra','father')";
			String query3= "insert into student_otherinfo
values(1,'sunil.chandra267@gmail.com','qwerty')";
```

```
                        stmt = con.createStatement();
                        //1. create a batch
                        stmt.addBatch(query1);
                        stmt.addBatch(query2);
                        stmt.addBatch(query3);


                        //2.execute batch
                        int[] countArr=stmt.executeBatch();

                        for (int count : countArr)
                        {
                               System.out.println("no of rows affected is "+count);

                        }//end of for

                       //3. clear the batch
                        stmt.clearBatch();
                }//end of try
                catch (ClassNotFoundException | SQLException e)
                {
                       e.printStackTrace();
                }//end of catch
                finally {
                       if(stmt!=null)
                              try {
                                     stmt.close();
                              } catch (SQLException e) {
                                     e.printStackTrace();
                              }
                       if(con!=null)
                              try {
                                     con.close();
                              } catch (SQLException e) {
                                     e.printStackTrace();
                              }
                }//end of finally
        }//end of main
}//end of class
```

JDBC Transaction

- A transaction is a "Group of SQL Queries" that are executed as a unit. So either all of the SQL Queries get executed successfully or none of them get executed.
- Transactions helps us to achieve "Data Consistency"
- Following steps are followed to make use of Transactions in JDBC
    a. Begin the transaction by disabling AutoCommit Mode [**con.setAutoCommit(false)]**
    b. Issue One/More SQL Queries[Generally "more than one" Insert/Update/Delete SQL Queries].
    c. If No Exception, then "Commit" the transaction[**con.commit()**]
    d. If Exception occurs, then "Rollback" the transaction [**con.rollback()**]

Methods Syntax :-

1. void Connection.setAutoCommit(boolean disable) throws SQLException
2. void Connection.commit() throws SQLException
3. void Connection.rollback() throws SQLException

Note:

1. Transactions can also be used with one or more Select SQL Queries but it's of no use.
2. Transactions can also be used with ONLY ONE Insert/Update/Delete SQL Query but it's of no use .
3. Whenever there is a scenario to execute "More than One Insert/Update/Delete" SQL Queries then "we must make use of Transactions".
4. Transactions Steps 1 to 3 we write inside "try block" & Transaction Step 4 (Rollback the Txn) we write inside "catch block".
5. 5.If there is more than one "catch block", then we have to write the Rollback statement in all the catch blocks.
6. We can also have rollback in the "finally block" but this impacts the performance. Since Rollback/Commit is an expensive operation,we should make use of Rollback/Commit whenever there is a need. i.e. "At any instant of time we should issue either Commit/Rollback but not both".

Example:

```
package edu.jspiders.batchproject;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

public class BatchDemo {
	public static void main(String[] args)
	{
		Connection con = null;
		Statement stmt = null;
		try {
			Class.forName("com.mysql.jdbc.Driver");

			String dburl =
"jdbc:mysql://localhost:3306/hejm15_db?user=root&password=root";
			 con = DriverManager.getConnection(dburl);

			 //disable autocommit mode
			 con.setAutoCommit(false);

		 String query1 = "insert into student_info values(1,'sunil','kumar','c')";
		String query2 = "insert into guardian_info values(1,'chandra','father')";
	       String query3= "insert into student_otherinfo
values(1,'sunil.chandra267@gmail.com','qwerty')";

			 stmt = con.createStatement();
			 //1. create a batch
			 stmt.addBatch(query1);
			 stmt.addBatch(query2);
			 stmt.addBatch(query3);


			 //2.execute batch
			 int[] countArr=stmt.executeBatch();

			 for (int count : countArr)
			 {
				System.out.println("no of rows affected is "+count);

			 }//end of for

			//3. clear the batch
			 stmt.clearBatch();

			 // after successfull execution of query commit
			 con.commit();
		}//end of try
		catch (ClassNotFoundException | SQLException e)
		{
			if(con!=null)
			{
				try {
					//if execution is failure rollback the executed
						//queries
					con.rollback();
```

```
                        } catch (SQLException e1) {
                        e1.printStackTrace();
                        }
                  }
                  e.printStackTrace();
            }//end of catch
            finally {
                  if(stmt!=null)
                        try {
                              stmt.close();
                        } catch (SQLException e) {
                              e.printStackTrace();
                        }
                  if(con!=null)
                        try {
                              con.close();
                        } catch (SQLException e) {
                              e.printStackTrace();
                        }
            }//end of finally
      }//end of main
}//end of class
```