JDBC - 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.

Batching with Statement Object

Here is a typical sequence of steps to use Batch Processing with Statement Object −

- Create a Statement object using either *createStatement* methods.
- Set auto-commit to false using *setAutoCommit*.
- Add as many as SQL statements you like into batch using *addBatch* method on created statement object.
- Execute all the SQL statements using *executeBatch* method on created statement object.
- Finally, commit all the changes using *commit* method.

Example

The following code snippet provides an example of a batch update using Statement object −

```
// 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();
```

For a better understanding, let us study the Batching - Example Code.

Batching with PrepareStatement Object

Here is a typical sequence of steps to use Batch Processing with PrepareStatement Object −

1. Create SQL statements with placeholders.
2. Create PrepareStatement object using either *prepareStatement* methods.
3. Set auto-commit to false using *setAutoCommit*.
4. Add as many as SQL statements you like into batch using *addBatch* method on created statement object.
5. Execute all the SQL statements using *executeBatch* method on created statement object.
6. Finally, commit all the changes using *commit* method.

The following code snippet provides an example of a batch update using PrepareStatement object −

```
// Create SQL statement
String SQL = "INSERT INTO Employees (id, first, last, age) " +
             "VALUES(?, ?, ?, ?)";

// Create PrepareStatement object
PreparedStatemen pstmt = conn.prepareStatement(SQL);

//Set auto-commit to false
conn.setAutoCommit(false);

// Set the variables
pstmt.setInt( 1, 400 );
pstmt.setString( 2, "Pappu" );
pstmt.setString( 3, "Singh" );
pstmt.setInt( 4, 33 );
// Add it to the batch
pstmt.addBatch();

// Set the variables
pstmt.setInt( 1, 401 );
pstmt.setString( 2, "Pawan" );
pstmt.setString( 3, "Singh" );
pstmt.setInt( 4, 31 );
// Add it to the batch
pstmt.addBatch();

//add more batches
.
.
.
.
//Create an int[] to hold returned values
int[] count = stmt.executeBatch();

//Explicitly commit statements to apply changes
conn.commit();
```

For a better understanding, let us study the Batching - Example Code.

Loading [MathJax]/jax/output/HTML-CSS/fonts/TeX/fontdata.js