

Lesson 01 Demo 05

Executing Batch Processing

Objective: To utilize Batch Processing and perform multiple SQL statements in a batch for efficient execution and management of large datasets. This approach minimizes database round-trips and enhances performance.

Tool required: Eclipse IDE

Prerequisites: None

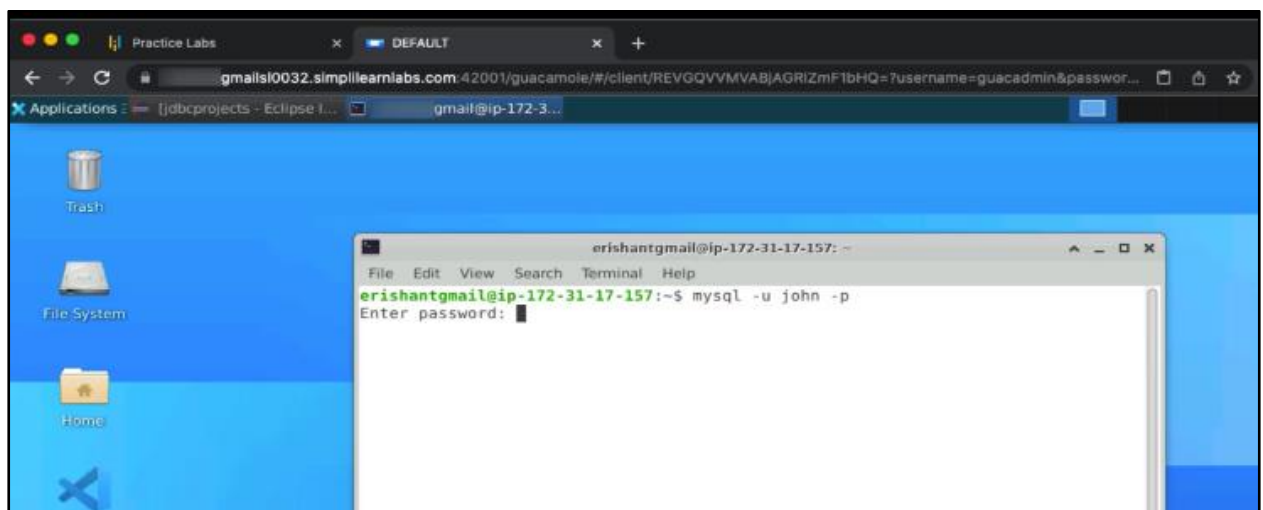
Steps to be followed:

1. Create a table for employee details
2. Create a method for batch execution
3. Use a prepared statement
4. Use mixed SQL statements in a batch

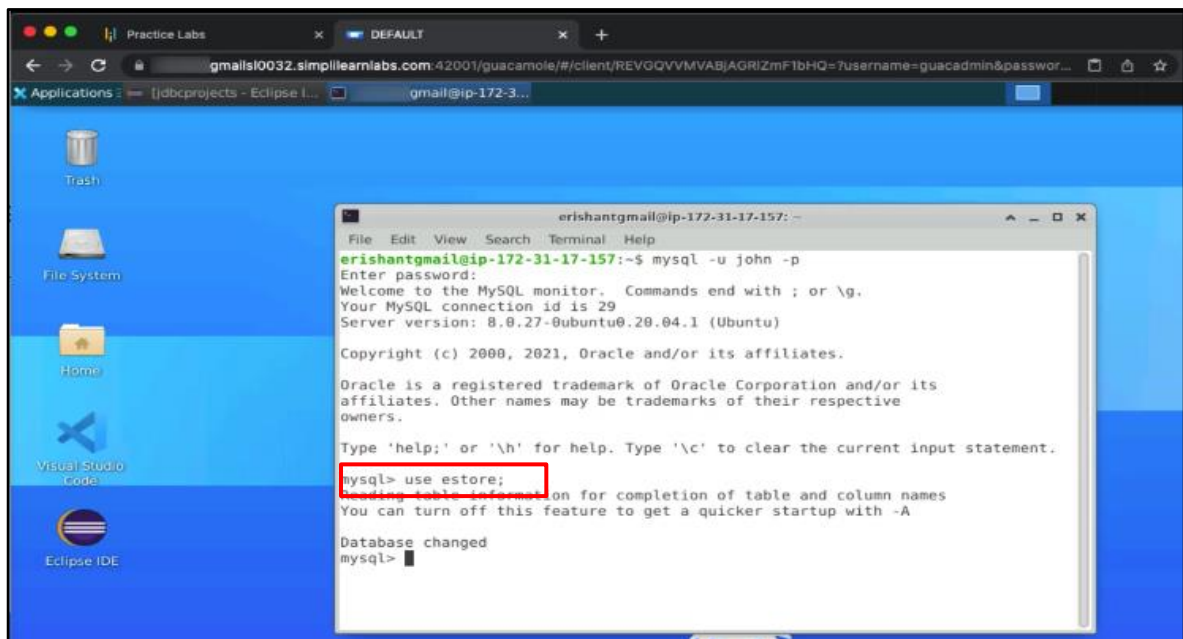
Step 1: Create a table for employee details

1.1 Open the terminal window and log in to MySQL using the following command:

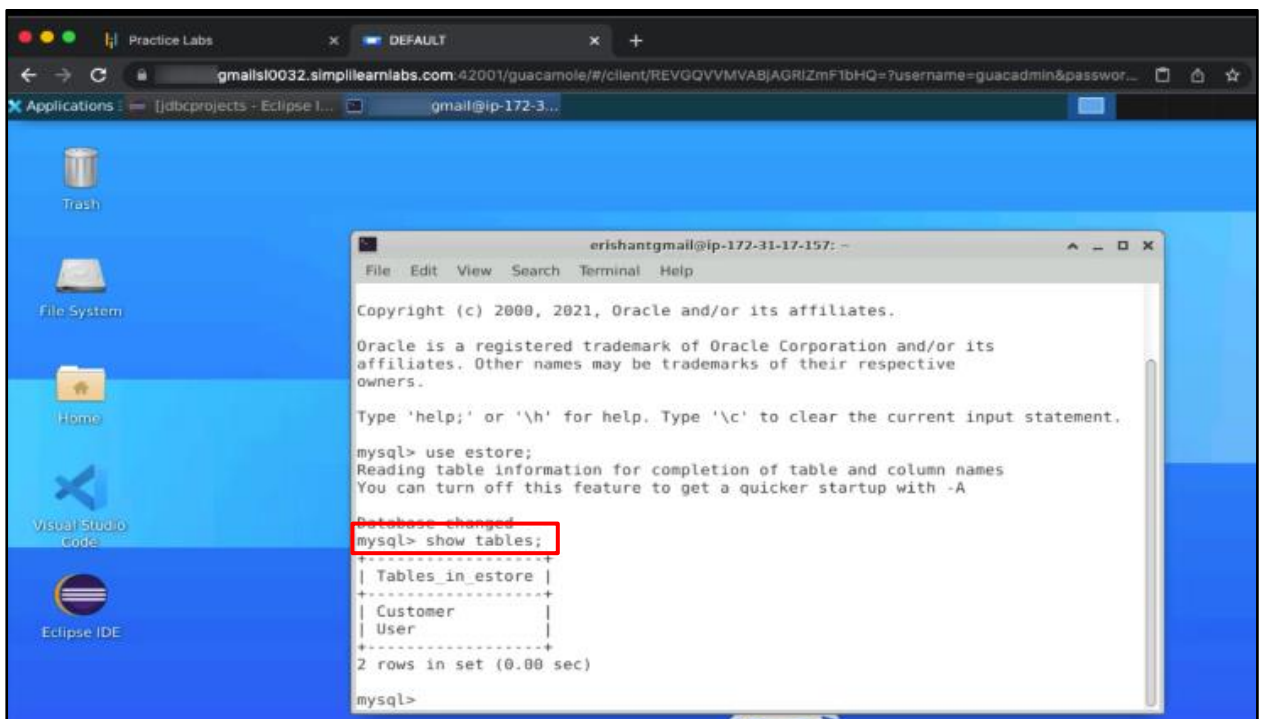
```
mysql -u john -p
```



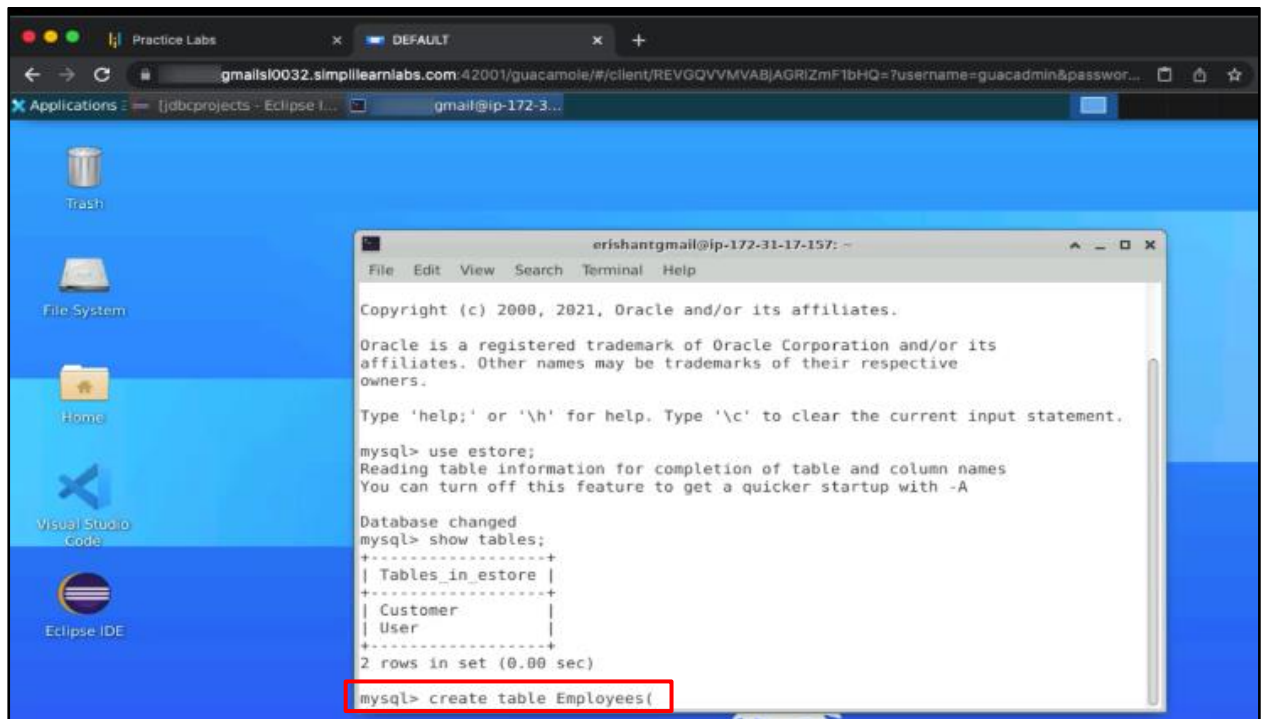
1.2 Enter the **use estore;** command to change the database



1.3 Run **show tables;** command to list tables in the **estore** database



1.4 Run **create table Employees()** command to create a new table in the **estore** database



```

erishantgmail@ip-172-31-17-157: ~
File Edit View Search Terminal Help

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

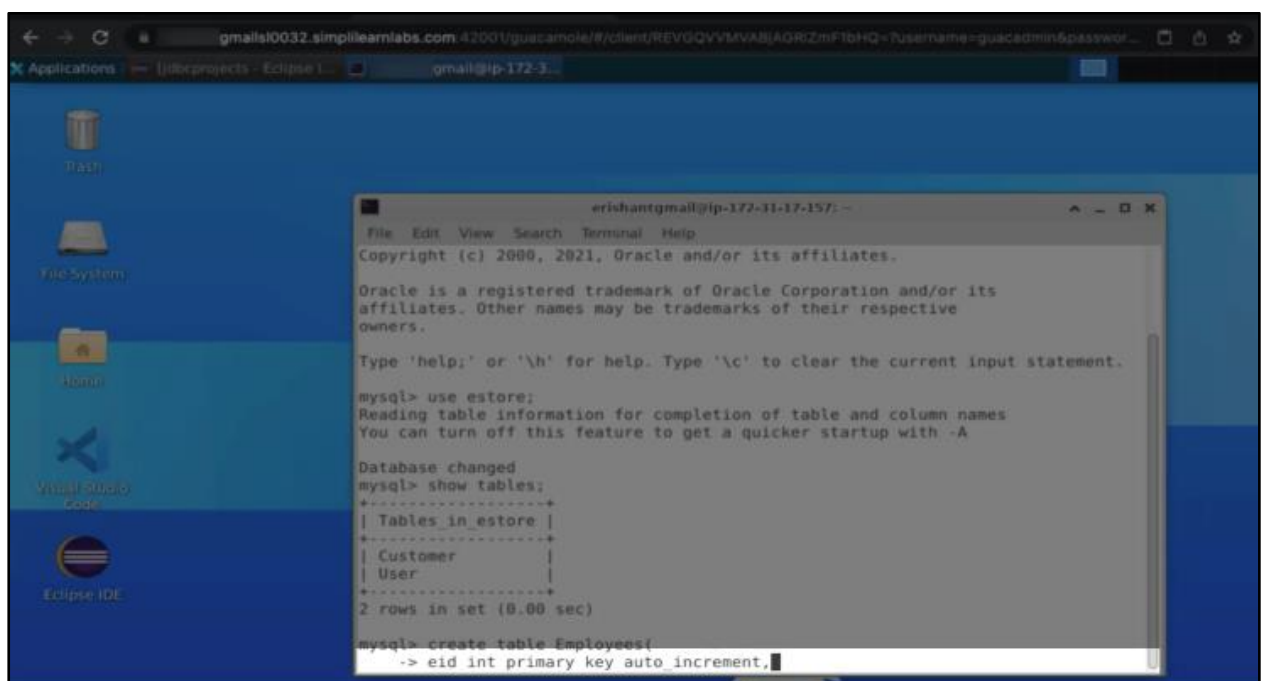
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use estore;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----+
| Tables_in_estore |
+-----+
| Customer         |
| User             |
+-----+
2 rows in set (0.00 sec)

mysql> create table Employees(
  
```

1.5 Enter the attribute **eid int primary key auto_increment**, to create a column for employee id



```

erishantgmail@ip-172-31-17-157: ~
File Edit View Search Terminal Help

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use estore;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----+
| Tables_in_estore |
+-----+
| Customer         |
| User             |
+-----+
2 rows in set (0.00 sec)

mysql> create table Employees(
-> eid int primary key auto_increment,
  
```

1.6 Add other attributes **name varchar (256)**, **email varchar (256)**, and **salary int** to the table

```

mysql> use estore;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----+
| Tables_in_estore |
+-----+
| Customer         |
| User             |
+-----+
2 rows in set (0.00 sec)

mysql> create table Employees(
-> eid int primary key auto_increment,
-> name varchar(256),
-> email varchar(256),
-> salary int
-> );
Query OK, 0 rows affected (0.04 sec)

mysql>
  
```

1.7 Enter **show tables;** command to check the table is created

```

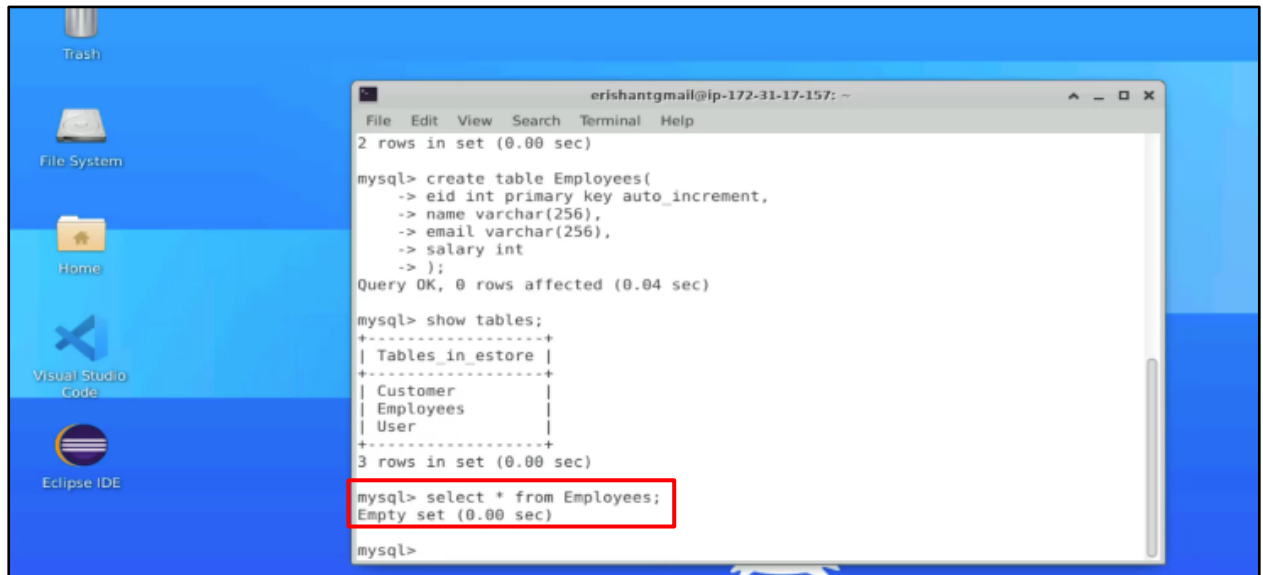
mysql> create table Employees(
-> eid int primary key auto_increment,
-> name varchar(256),
-> email varchar(256),
-> salary int
-> );
Query OK, 0 rows affected (0.04 sec)

mysql> show tables;
+-----+
| Tables_in_estore |
+-----+
| Customer         |
| Employees        |
| User             |
+-----+
3 rows in set (0.00 sec)

mysql>
  
```

1.8 Run the **select** command to check the data in the table created. For now, the table is empty.

select * from Employees;



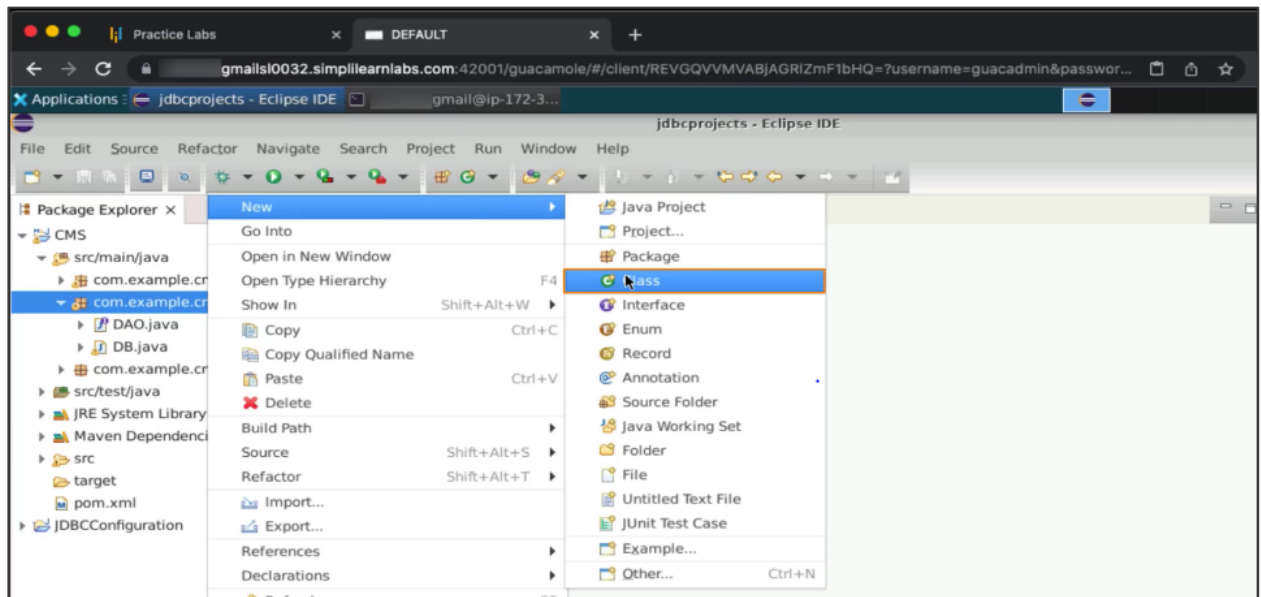
The screenshot shows a Linux desktop with a blue background. On the left sidebar, there are icons for Trash, File System, Home, Visual Studio Code, and Eclipse IDE. A terminal window is open in the center, displaying the following MySQL commands and their outputs:

```
erishantgmail@ip-172-31-17-157: ~  
File Edit View Search Terminal Help  
2 rows in set (0.00 sec)  
  
mysql> create table Employees(  
-> eid int primary key auto_increment,  
-> name varchar(256),  
-> email varchar(256),  
-> salary int  
-> );  
Query OK, 0 rows affected (0.04 sec)  
  
mysql> show tables;  
+-----+  
| Tables_in_estore |  
+-----+  
| Customer         |  
| Employees        |  
| User             |  
+-----+  
3 rows in set (0.00 sec)  
  
mysql> select * from Employees;  
Empty set (0.00 sec)  
  
mysql>
```

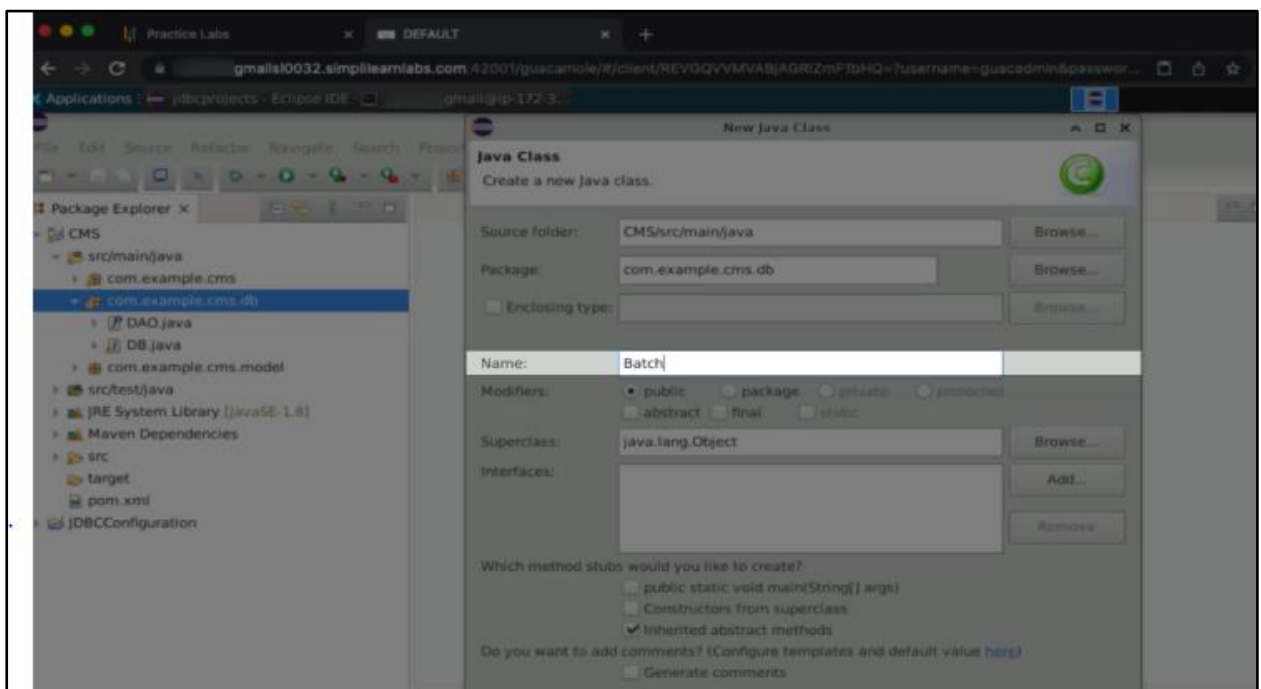
The output for the `select * from Employees;` command is highlighted with a red rectangle, showing "Empty set (0.00 sec)".

Step 2: Create a method for batch execution

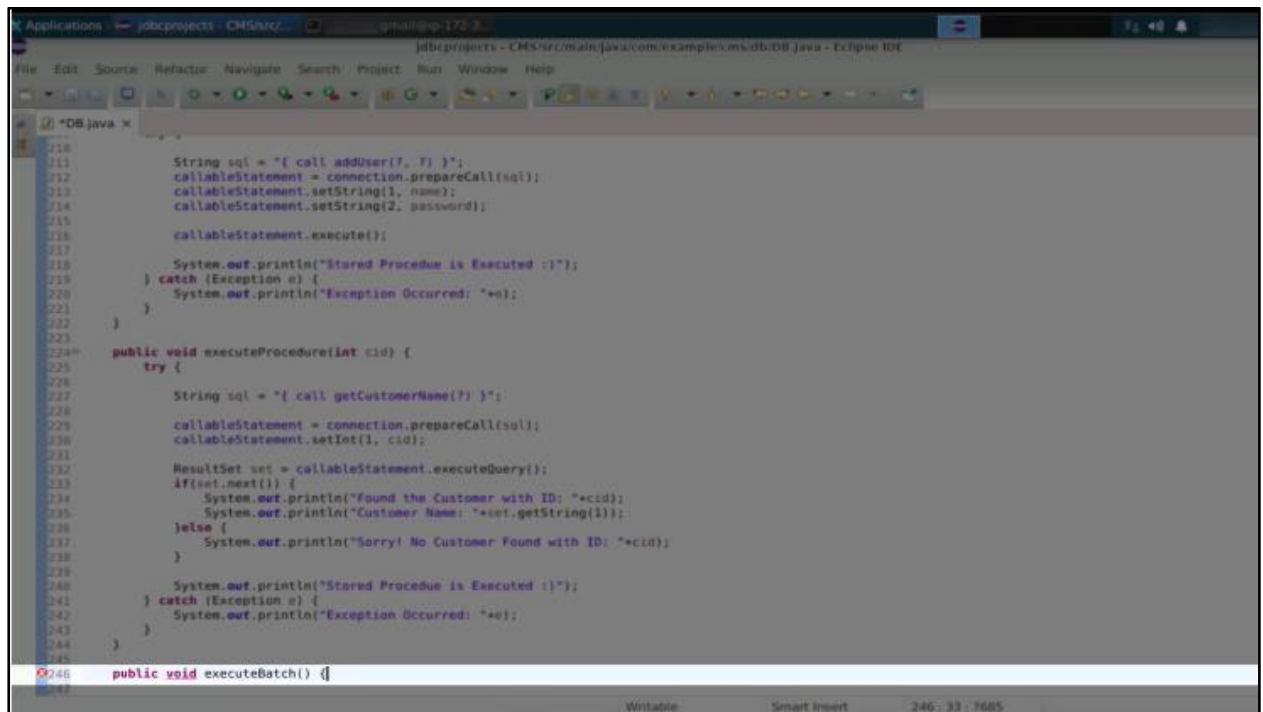
- 2.1 Create a new class in the package DB, open **Eclipse IDE**, go to **CMS** in the folder section
 > **src** > **com.example.cms**, then right-click, select **New**, and click on **Class**



- 2.2 Name the class **Batch**



2.3 Create a new method **executeBatch()** for the batch execution process



```

210 String sql = "{ call addUser(?, ?) }";
211 callableStatement = connection.prepareCall(sql);
212 callableStatement.setString(1, name);
213 callableStatement.setString(2, password);
214
215 callableStatement.execute();
216
217 System.out.println("Stored Procedure is Executed :)");
218 } catch (Exception e) {
219     System.out.println("Exception Occurred: "+e);
220 }
221 }
222
223
224 public void executeProcedure(int cid) {
225     try {
226
227         String sql = "{ call getCustomerName(?) }";
228         callableStatement = connection.prepareCall(sql);
229         callableStatement.setInt(1, cid);
230
231         ResultSet set = callableStatement.executeQuery();
232         if(set.next()) {
233             System.out.println("Found the Customer with ID: "+cid);
234             System.out.println("Customer Name: "+set.getString(1));
235         } else {
236             System.out.println("Sorry! No Customer Found with ID: "+cid);
237         }
238
239         System.out.println("Stored Procedure is Executed :)");
240     } catch (Exception e) {
241         System.out.println("Exception Occurred: "+e);
242     }
243 }
244
245
246 public void executeBatch() {}
247

```


2.4 Create a **try-catch** block with multiple SQL statements

```

220         System.out.println("Exception Occurred: "+e);
221     }
222 }
223
224 public void executeProcedure(int cid) {
225     try {
226         String sql = "{ call getCustomerName(?) }";
227         callableStatement = connection.prepareCall(sql);
228         callableStatement.setInt(1, cid);
229
230         ResultSet set = callableStatement.executeQuery();
231         if(set.next()) {
232             System.out.println("Found the Customer with ID: "+cid);
233             System.out.println("Customer Name: "+set.getString(1));
234         } else {
235             System.out.println("Sorry! No Customer Found with ID: "+cid);
236         }
237         System.out.println("Stored Procedure is Executed :)");
238     } catch (Exception e) {
239         System.out.println("Exception Occurred: "+e);
240     }
241 }
242
243
244
245
246 public void executeBatch() {
247     try {
248         String sql1 = "insert into Employee values(null, 'John', 'john@example.com', 30000)";
249
250         } catch (Exception e) {
251             System.out.println("Exception Occurred: "+e);
252         }
253     }
254 }
255
256 }
257

```

```

        ResultSet set = callableStatement.executeQuery();
        if(set.next()) {
            System.out.println("Found the Customer with ID: "+cid);
            System.out.println("Customer Name: "+set.getString(1));
        } else {
            System.out.println("Sorry! No Customer Found with ID: "+cid);
        }

        System.out.println("Stored Procedure is Executed :)");
    } catch (Exception e) {
        System.out.println("Exception Occurred: "+e);
    }
}

public void executeBatch() {
    try {

        String sql1 = "insert into Employee values(null, 'John', 'john@example.com', 30000)";
        String sql2 = "insert into Employee values(null, 'Fionna', 'fionna@example.com', 34000)";
        String sql3 = "insert into Employee values(null, 'Leo', 'leo@example.com', 40000)";
        String sql4 = "insert into Employee values(null, 'Mike', 'mike@example.com', 50000)";
        String sql5 = "insert into Employee values(null, 'Kim', 'kim@example.com', 65000)";

        statement.addBatch(sql1);

    } catch (Exception e) {
        System.out.println("Exception Occurred: "+e);
    }
}

```


2.5 Execute a statement called **addbatch()** to create multiple SQL commands easily

```
String sql = "{ call getCustomerName(?) }";

callableStatement = connection.prepareCall(sql);
callableStatement.setInt(1, cid);

ResultSet set = callableStatement.executeQuery();
if(set.next()) {
    System.out.println("Found the Customer with ID: "+cid);
    System.out.println("Customer Name: "+set.getString(1));
}else {
    System.out.println("Sorry! No Customer Found with ID: "+cid);
}

System.out.println("Stored Procedure is Executed :)");
} catch (Exception e) {
    System.out.println("Exception Occurred: "+e);
}

}

public void executeBatch() {
    try {

        String sql1 = "insert into Employee values(null, 'John', 'john@example.com', 30000)";
        String sql2 = "insert into Employee values(null, 'Fionna', 'fionna@example.com', 34000)";
        String sql3 = "insert into Employee values(null, 'Leo', 'leo@example.com', 40000)";
        String sql4 = "insert into Employee values(null, 'Mike', 'mike@example.com', 50000)";
        String sql5 = "insert into Employee values(null, 'Kim', 'kim@example.com', 65000)";

        statement.addBatch(sql1);
        statement.addBatch(sql2);
        statement.addBatch(sql3);
        statement.addBatch(sql4);
        statement.addBatch(sql5);

    } catch (Exception e) {
        System.out.println("Exception Occurred: "+e);
    }
}
```

2.6 Enter **executeBatch()**; statement, which will submit a batch of commands to the database for execution

```
System.out.println("Stored Procedure is Executed :)");
} catch (Exception e) {
    System.out.println("Exception Occurred: "+e);
}

}

public void executeBatch() {
    try {

        String sql1 = "insert into Employee values(null, 'John', 'john@example.com', 30000)";
        String sql2 = "insert into Employee values(null, 'Fionna', 'fionna@example.com', 34000)";
        String sql3 = "insert into Employee values(null, 'Leo', 'leo@example.com', 40000)";
        String sql4 = "insert into Employee values(null, 'Mike', 'mike@example.com', 50000)";
        String sql5 = "insert into Employee values(null, 'Kim', 'kim@example.com', 65000)";

        statement.addBatch(sql1);
        statement.addBatch(sql2);
        statement.addBatch(sql3);
        statement.addBatch(sql4);
        statement.addBatch(sql5);

        int[] results = statement.executeBatch();

    } catch (Exception e) {
        System.out.println("Exception Occurred: "+e);
    }
}
```

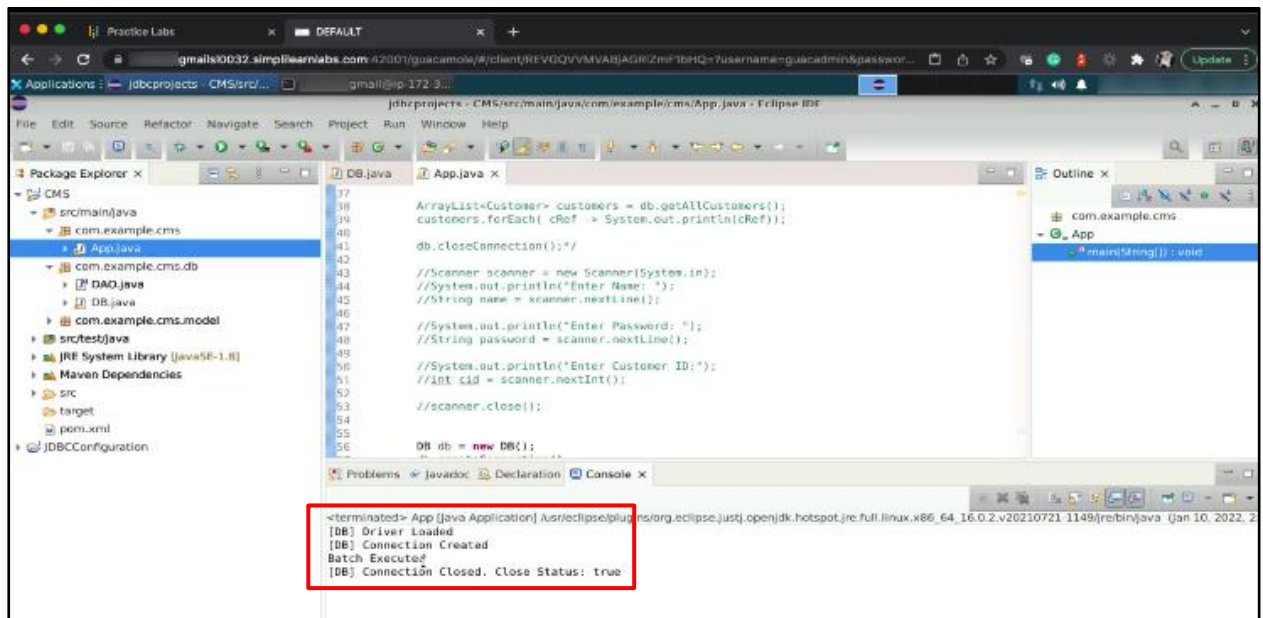
2.7 Go to **App.java** file comment statements on lines 43, 50, 51, 53, and 59, then add **db.executeBatch()** for SQL batch execution

The screenshot shows the Eclipse IDE with the **App.java** file open. The Package Explorer on the left shows the project structure. The main editor displays the code for **App.java**. Lines 43, 50, 51, 53, and 59 are commented out. A new line, 50, has been added with the code `db.executeBatch();`. The console at the bottom shows the output of the application, including "Enter Customer ID:", "[DB] Driver Loaded", "[DB] Connection Created", and "Sorry! No Customer Found with ID: 1".

2.8 Change the method name to **executeSQLStatmentsInBatch()** in both files to avoid confusion with other methods. Also, change **Employee** to **Employees** as the database name in the **DB.java** file

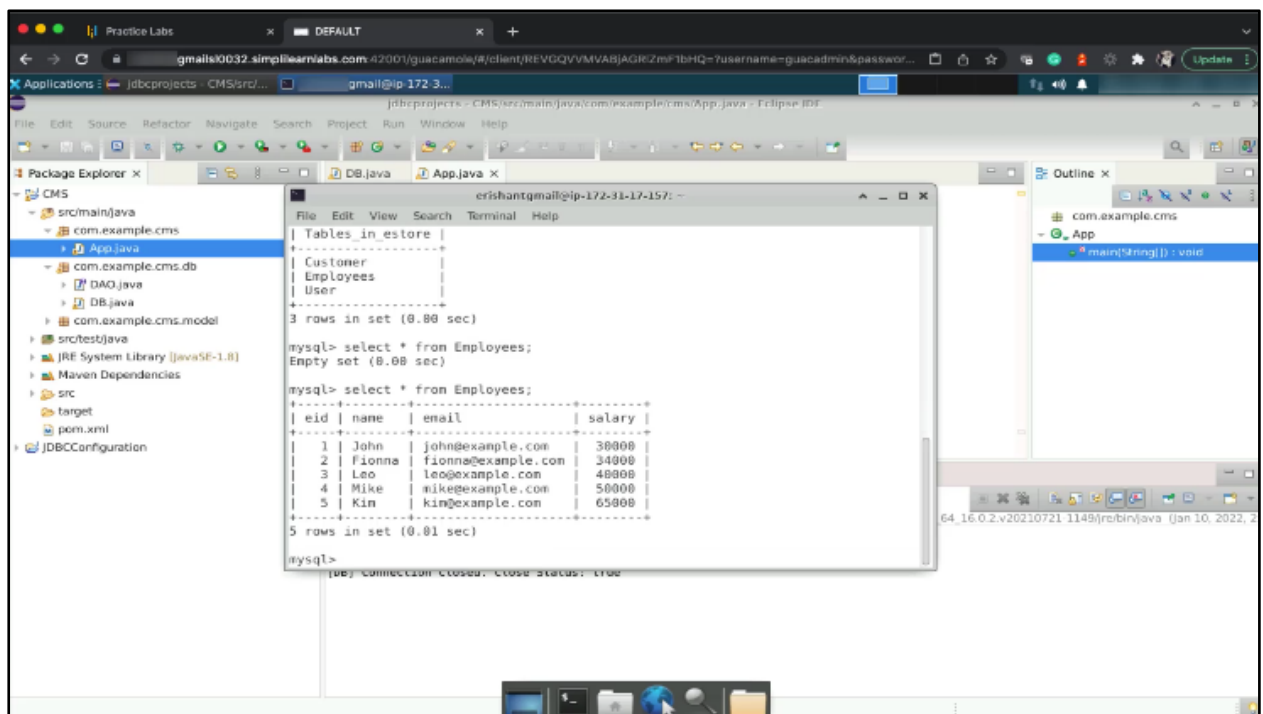
The screenshot shows the Eclipse IDE with the **DB.java** file open. The Package Explorer on the left shows the project structure. The main editor displays the code for **DB.java**. A new method, `public void executeSQLStatmentsInBatch() {`, has been added to the class. The method contains five SQL insert statements for the **Employees** table. The console at the bottom shows the output of the application, including "Enter Customer ID:", "[DB] Driver Loaded", "[DB] Connection Created", and "Exception Occurred: java.sql.BatchUpdateException: Table 'store.Employee' doesn't exist".

2.9 Save and run the code. You can see **Batch Executed** as output in the console.



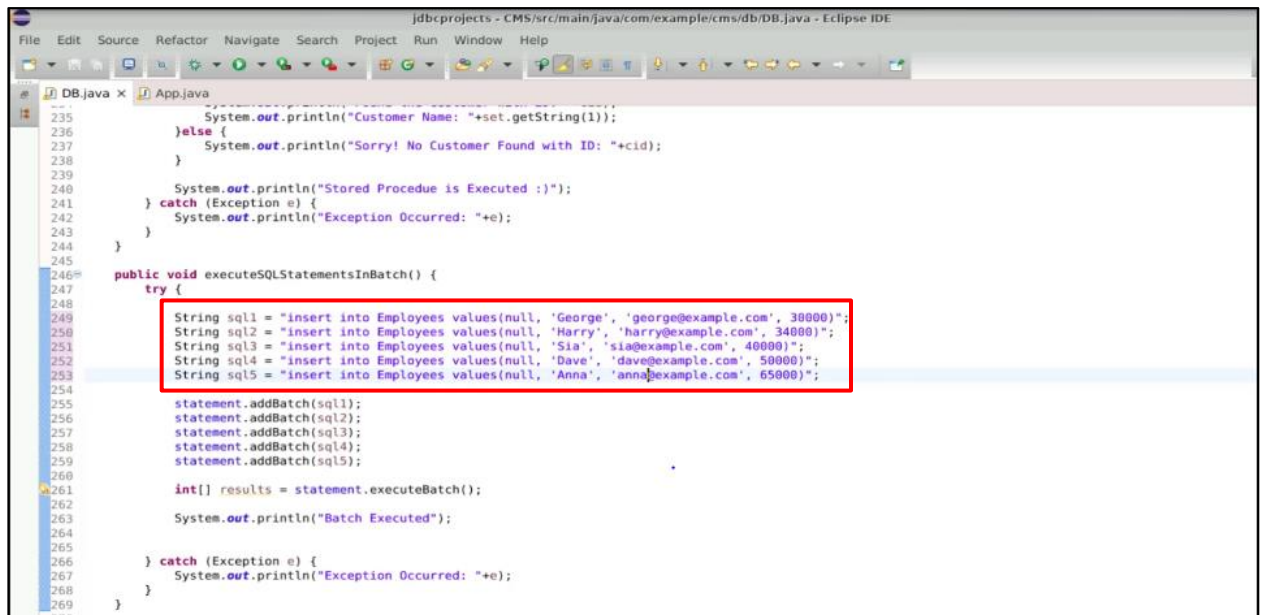
2.10 Rerun the select command in the terminal. You will get the list of employees inserted.

select * from Employees



Step 3: Use a prepared statement

3.1 Change the details of the employees with new employee details in the **DB.java** file



The screenshot shows the Eclipse IDE with the file `DB.java` open. The code is part of a project named `jdbcpjprojects - CMS/src/main/java/com/example/cms/db/DB.java`. The code includes a `catch` block for exceptions and a `public void executeSQLStatementsInBatch()` method. The `executeSQLStatementsInBatch` method contains a `try` block where five SQL `INSERT` statements are defined and added to a batch. These statements are highlighted with a red box in the original image. The statements are as follows:

```
String sql1 = "insert into Employees values(null, 'George', 'george@example.com', 38000)";
String sql2 = "insert into Employees values(null, 'Harry', 'harry@example.com', 34000)";
String sql3 = "insert into Employees values(null, 'Sia', 'sia@example.com', 40000)";
String sql4 = "insert into Employees values(null, 'Dave', 'dave@example.com', 50000)";
String sql5 = "insert into Employees values(null, 'Anna', 'anna@example.com', 65000)";
```

After the SQL statements, the code calls `statement.addBatch(sql1);` through `statement.addBatch(sql5);`, then `int[] results = statement.executeBatch();` and `System.out.println("Batch Executed");`. A final `catch` block handles any exceptions that occur during the batch execution.

3.2 Comment all the `addbatch()` statements and create a preparedStatement API: `preparedStatement = connection.prepareStatement();`

```

235      System.out.println("Customer Name: "+set.getString(1));
236    }else {
237      System.out.println("Sorry! No Customer Found with ID: "+cid);
238    }
239
240    System.out.println("Stored Procedure is Executed :");
241  } catch (Exception e) {
242    System.out.println("Exception Occurred: "+e);
243  }
244  }
245
246  public void executeSQLStatementsInBatch() {
247    try {
248
249      String sql1 = "insert into Employees values(null, 'George', 'george@example.com', 30000)";
250      String sql2 = "insert into Employees values(null, 'Harry', 'harry@example.com', 34000)";
251      String sql3 = "insert into Employees values(null, 'Sia', 'sia@example.com', 40000)";
252      String sql4 = "insert into Employees values(null, 'Dave', 'dave@example.com', 50000)";
253      String sql5 = "insert into Employees values(null, 'Anna', 'anna@example.com', 65000)";
254
255      /*statement.addBatch(sql1);
256      statement.addBatch(sql2);
257      statement.addBatch(sql3);
258      statement.addBatch(sql4);
259      statement.addBatch(sql5);
260
261      int[] results = statement.executeBatch();*/
262
263      preparedStatement = connection.prepareStatement();
264
265      System.out.println("Batch Executed");
266
267    } catch (Exception e) {
268      System.out.println("Exception Occurred: "+e);
269    }
270  }
271  }
272

```

3.3 Add a SQL statement for preparedStatement API to insert the data in the database

```

235      System.out.println("Customer Name: "+set.getString(1));
236    }else {
237      System.out.println("Sorry! No Customer Found with ID: "+cid);
238    }
239
240    System.out.println("Stored Procedure is Executed :");
241  } catch (Exception e) {
242    System.out.println("Exception Occurred: "+e);
243  }
244  }
245
246  public void executeSQLStatementsInBatch() {
247    try {
248
249      String sql1 = "insert into Employees values(null, 'George', 'george@example.com', 30000)";
250      String sql2 = "insert into Employees values(null, 'Harry', 'harry@example.com', 34000)";
251      String sql3 = "insert into Employees values(null, 'Sia', 'sia@example.com', 40000)";
252      String sql4 = "insert into Employees values(null, 'Dave', 'dave@example.com', 50000)";
253      String sql5 = "insert into Employees values(null, 'Anna', 'anna@example.com', 65000)";
254
255      /*statement.addBatch(sql1);
256      statement.addBatch(sql2);
257      statement.addBatch(sql3);
258      statement.addBatch(sql4);
259      statement.addBatch(sql5);
260
261      int[] results = statement.executeBatch();*/
262
263      String sql = "insert into Employees values(null, 'George', 'george@example.com', 30000)";
264      preparedStatement = connection.prepareStatement();
265
266      System.out.println("Batch Executed");
267
268    } catch (Exception e) {
269      System.out.println("Exception Occurred: "+e);
270    }
271  }
272  }
273

```


3.4 Use the **addBatch()** function to add a batch for the preparedStatement to enter the details of the employee's name **George**

```

244 }
245
246 public void executeSQLStatementsInBatch() {
247     try {
248
249         String sql1 = "insert into Employees values(null, 'George', 'george@example.com', 30000)";
250         String sql2 = "insert into Employees values(null, 'Harry', 'harry@example.com', 34000)";
251         String sql3 = "insert into Employees values(null, 'Sia', 'sia@example.com', 40000)";
252         String sql4 = "insert into Employees values(null, 'Dave', 'dave@example.com', 50000)";
253         String sql5 = "insert into Employees values(null, 'Anna', 'anna@example.com', 65000)";
254
255         /*statement.addBatch(sql1);
256         statement.addBatch(sql2);
257         statement.addBatch(sql3);
258         statement.addBatch(sql4);
259         statement.addBatch(sql5);
260
261         int[] results = statement.executeBatch();*/
262
263         String sql = "insert into Employees values(null, ?, ?, ?)";
264         PreparedStatement = connection.prepareStatement(sql);
265
266         PreparedStatement.setString(1, "George");
267         PreparedStatement.setString(2, "george@example.com");
268         PreparedStatement.setInt(3, 30000);
269         PreparedStatement.addBatch();
270
271
272
273         System.out.println("Batch Executed");
274
275     } catch (Exception e) {
276

```

3.5 Repeat the above step for other records, and at the end, add the **preparedStatement** to execute the batch as: **int[] = results = preparedStatement.executeBatch()**

```

250 String sql2 = "insert into Employees values(null, 'Harry', 'harry@example.com', 34000)";
251 String sql3 = "insert into Employees values(null, 'Sia', 'sia@example.com', 40000)";
252 String sql4 = "insert into Employees values(null, 'Dave', 'dave@example.com', 50000)";
253 String sql5 = "insert into Employees values(null, 'Anna', 'anna@example.com', 65000)";
254
255 /*statement.addBatch(sql1);
256 statement.addBatch(sql2);
257 statement.addBatch(sql3);
258 statement.addBatch(sql4);
259 statement.addBatch(sql5);
260
261 int[] results = statement.executeBatch();*/
262
263 String sql = "insert into Employees values(null, ?, ?, ?)";
264 PreparedStatement = connection.prepareStatement(sql);
265
266 PreparedStatement.setString(1, "George");
267 PreparedStatement.setString(2, "george@example.com");
268 PreparedStatement.setInt(3, 30000);
269 PreparedStatement.addBatch();
270
271 PreparedStatement.setString(1, "Harry");
272 PreparedStatement.setString(2, "harry@example.com");
273 PreparedStatement.setInt(3, 34000);
274 PreparedStatement.addBatch();
275
276 PreparedStatement.setString(1, "Sia");
277 PreparedStatement.setString(2, "sia@example.com");
278 PreparedStatement.setInt(3, 40000);
279 PreparedStatement.addBatch();
280
281 int[] results = PreparedStatement.executeBatch();
282
283 System.out.println("Batch Executed");
284
285 } catch (Exception e) {
286     System.out.println("Exception Occurred: "+e);
287

```

3.6 Go back to the **App.java** file and execute the code. You will see **Batch Executed** as the output without any error.

The screenshot shows the Eclipse IDE with the `App.java` file open. The code includes database operations and a batch execution. The Console window on the right shows the following output:

```
<terminated> App [Java Application] /usr/eclipse/plugins/org.eclipse.justi.openjdk.hotspot
[[DB] Driver Loaded
[DB] Connection Created
Batch Executed
[DB] Connection Closed. Close Status: true
```

3.7 Go to the terminal and run the select command. The three new employee records get inserted

The screenshot shows the Eclipse IDE with two Java files, `DB.java` and `App.java`, and a terminal window. The terminal window shows the output of a SQL query:

```
mysql> select * from Employees;
+----+-----+-----+-----+
| id | name | email | salary |
+----+-----+-----+-----+
| 1 | John | john@example.com | 38000 |
| 2 | Fionna | fionna@example.com | 34000 |
| 3 | Leo | leo@example.com | 48000 |
| 4 | Mike | mike@example.com | 50000 |
| 5 | Kin | kin@example.com | 65000 |
+----+-----+-----+-----+
5 rows in set (0.01 sec)
```

The `App.java` file shows the following code:

```
//db.createCustomer(customer);
//db.updateCustomer(customer);
//db.deleteCustomer(3);
//System.out.println();

ArrayList<Customer> customers = db.getAllCustomers();
customers.forEach( cRef -> System.out.println(cRef));

db.closeConnection();*/

//Scanner scanner = new Scanner(System.in);
//System.out.println("Enter Name: ");
//String name = scanner.nextLine();

//System.out.println("Enter Password: ");
//String password = scanner.nextLine();

//System.out.println("Enter Customer ID:");
//int cid = scanner.nextInt();

//scanner.close();

DB db = new DB();
db.createConnection();
//db.executeProcedure(name, password);
//db.executeProcedure(cid);

db.executeSQLStatementsInBatch();
```


Step 4: Using mixed SQL statements in a batch

4.1 Mark the `preparedStatement` part as a comment

```
App.java
String sql2 = "insert into Employees values(null, 'Harry', 'harry@example.com', 34000)";
String sql3 = "insert into Employees values(null, 'Sia', 'sia@example.com', 40000)";
String sql4 = "insert into Employees values(null, 'Dave', 'dave@example.com', 50000)";
String sql5 = "insert into Employees values(null, 'Anna', 'anna@example.com', 65000)";

/*statement.addBatch(sql1);
statement.addBatch(sql2);
statement.addBatch(sql3);
statement.addBatch(sql4);
statement.addBatch(sql5);

int[] results = statement.executeBatch();*/

/*String sql = "insert into Employees values(null, ?, ?, ?)";
preparedStatement = connection.prepareStatement(sql);

preparedStatement.setString(1, "George");
preparedStatement.setString(2, "george@example.com");
preparedStatement.setInt(3, 30000);
preparedStatement.addBatch();

preparedStatement.setString(1, "Harry");
preparedStatement.setString(2, "harry@example.com");
preparedStatement.setInt(3, 34000);
preparedStatement.addBatch();

preparedStatement.setString(1, "Sia");
preparedStatement.setString(2, "sia@example.com");
preparedStatement.setInt(3, 40000);
preparedStatement.addBatch();

int[] results = preparedStatement.executeBatch();*/

System.out.println("Batch Executed");
```

4.2 Add one insertion SQL statement to insert the data

```
public void executeSQLStatementsInBatch() {
    try {
        /*String sql1 = "insert into Employees values(null, 'George', 'george@example.com', 30000)";
        String sql2 = "insert into Employees values(null, 'Harry', 'harry@example.com', 34000)";
        String sql3 = "insert into Employees values(null, 'Sia', 'sia@example.com', 40000)";
        String sql4 = "insert into Employees values(null, 'Dave', 'dave@example.com', 50000)";
        String sql5 = "insert into Employees values(null, 'Anna', 'anna@example.com', 65000)";*/

        String sql4 = "insert into Employees values(null, 'Dave', 'dave@example.com', 50000)";

        statement.addBatch(sql1);
        statement.addBatch(sql2);
        statement.addBatch(sql3);
        statement.addBatch(sql4);
        statement.addBatch(sql5);

        int[] results = statement.executeBatch();

        /*String sql = "insert into Employees values(null, ?, ?, ?)";
        preparedStatement = connection.prepareStatement(sql);

        preparedStatement.setString(1, "George");
        preparedStatement.setString(2, "george@example.com");
        preparedStatement.setInt(3, 30000);
        preparedStatement.addBatch();

        preparedStatement.setString(1, "Harry");
        preparedStatement.setString(2, "harry@example.com");
        preparedStatement.setInt(3, 34000);
        preparedStatement.addBatch();

        preparedStatement.setString(1, "Sia");
```

4.3 Add an SQL statement to update the **employee** table. Set the name as **Ron** for employee ID 2

```
public void executeSQLStatementsInBatch() {
    try {
        /*String sql1 = "insert into Employees values(null, 'George', 'george@example.com', 30000)";
        String sql2 = "insert into Employees values(null, 'Harry', 'harry@example.com', 34000)";
        String sql3 = "insert into Employees values(null, 'Sia', 'sia@example.com', 40000)";
        String sql4 = "insert into Employees values(null, 'Dave', 'dave@example.com', 50000)";
        String sql5 = "insert into Employees values(null, 'Anna', 'anna@example.com', 65000)";*/

        String sql1 = "insert into Employees values(null, 'Dave', 'dave@example.com', 50000)";
        String sql2 = "update Employees set name = 'Ron' where eid = 2";

        statement.addBatch(sql1);
        statement.addBatch(sql2);
        statement.addBatch(sql3);
        statement.addBatch(sql4);
        statement.addBatch(sql5);

        int[] results = statement.executeBatch();

        /*String sql = "insert into Employees values(null, ?, ?, ?)";
        preparedStatement = connection.prepareStatement(sql);

        preparedStatement.setString(1, "George");
        preparedStatement.setString(2, "george@example.com");
        preparedStatement.setInt(3, 30000);
        preparedStatement.addBatch();

        preparedStatement.setString(1, "Harry");
        preparedStatement.setString(2, "harry@example.com");
        preparedStatement.setInt(3, 34000);
```


4.6 Go to the terminal and run the **select** command. You can see all the SQL statements in the batch get executed successfully.

```

App.java x
customer.setBirthDate(
customer.setAge(35);
customer.setInDateTim
customer.setOutDateTim
customer.setTemperature

System.out.println("Con
DB db = new DB();
db.createConnection();

//db.createCustomer(cus
//db.updateCustomer(cus

//db.deleteCustomer(3);

//System.out.println();

ArrayList<Customer> cus
customers.forEach( cRef

db.closeConnection();*/

//Scanner scanner = new
//System.out.println("E
//String name = scanner

//System.out.println("E
//String password = sca

//System.out.println("Enter Customer ID:");
//int cid = scanner.nextInt();

//scanner.close();

erishantgmail@ip-172-31-17-157: ~
File Edit View Search Terminal Help
4 | Mike | mike@example.com | 50000 |
5 | Kim | kim@example.com | 65000 |
6 | George | george@example.com | 30000 |
7 | Harry | harry@example.com | 34000 |
8 | Sia | sia@example.com | 40000 |
+-----+
8 rows in set (0.00 sec)

mysql> select * from Employees;
+-----+
| eid | name | email | salary |
+-----+
1 | John | john@example.com | 30000 |
2 | Rog | fionna@example.com | 34000 |
3 | Leo | leo@example.com | 40000 |
4 | Mike | mike@example.com | 50000 |
6 | George | george@example.com | 30000 |
7 | Harry | harry@example.com | 34000 |
8 | Sia | sia@example.com | 40000 |
9 | Dave | dave@example.com | 50000 |
+-----+
8 rows in set (0.00 sec)

mysql>

```

By following these steps, you have successfully executed Batch Processing and performed multiple SQL statements in a batch for efficient execution and management of large datasets. This approach minimizes database round-trips and enhances performance.