

Day 14- Facilitation Guide

Index

- I. Executing insert,update and delete queries from JDBC
- II. Handling Errors in MySql

For (1.5 hrs) ILT

During the previous session about the JDBC interface,Connect Java with MySQL using JDBC and and execute query using Statement interface, we covered a variety of important concepts:

JDBC (Java Database Connectivity) is a Java API that facilitates interaction between Java applications and relational databases like MySQL, Oracle, and PostgreSQL. It offers a standardized approach for connecting to databases, executing SQL queries, and handling results in Java applications.

To use JDBC with MySQL:

Import the JDBC Library to your project's classpath.Load the MySQL driver with Class.forName().Establish a connection using DriverManager.getConnection().Choose between Statement or PreparedStatement for SQL queries.For SELECT queries, use executeQuery(); for modifications, use executeUpdate().Process SELECT query results with ResultSet.Close resources (Connection, Statement, ResultSet) to prevent leaks.

Statement Interface:

A Statement in JDBC is an interface that represents an SQL statement to be executed against a database.

ResultSet:

A ResultSet in JDBC is an interface that represents the result set of a SELECT query executed against a database.

In this session, we'll investigate insert, update, and delete queries, along with the implementation of an error handling mechanism.

Trainer will ask questions to students to increase the curiosity:

What's the primary purpose of inserting new data into a database, and how does it benefit businesses and applications?

I. Executing insert,update and delete queries from JDBC

Executing Insert Query

Here's a step-by-step process on executing an INSERT query using JDBC in a Java application to add data to a MySQL database. This assumes you have already set up your MySQL database and have the necessary JDBC driver in your project's classpath.

Step 1: Import Necessary Packages:

Import the required JDBC packages at the beginning of your Java class:

```
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.PreparedStatement;  
import java.sql.SQLException;
```

Step 2: Establish Database Connection:

Create a method to establish a connection to your MySQL database. Replace yourDatabaseURL, yourUsername, and yourPassword with your database credentials:

```
public Connection dbConnect() {  
    Connection connection = null;  
    try {  
        String url = "jdbc:mysql://yourDatabaseURL:3306/yourDatabaseName";  
        String username = "yourUsername";  
        String password = "yourPassword";  
        connection = DriverManager.getConnection(url, username, password);  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
    return connection;  
}
```

Step 3: Create INSERT Query:

Define your SQL INSERT query. In this example, we'll insert a new record into a hypothetical "Student" table:

```
String insertQuery = "INSERT INTO Student  
(StudentID,FirstName,LastName,DateOfBirth,Gender, Email,Phone,marks)  
VALUES\r\n"  
+ " ('S106','lilly', 'Doe','2001-10-10','M', 'lilly@example.com','9878457876',78)  
";
```

Step 4:Prepare and Execute the INSERT Statement:

Use a Statement to execute the INSERT query.

```
stmt.executeUpdate(sql);
```

Step 5:Close the Connection:

Always close the database resources (connection, statement, etc.) to release connections and resources:

```
connection.close();
```

Handle Exceptions: Be sure to handle any potential SQLExceptions by catching and handling them appropriately in your application.

Let's explore complete Example:

```
package jdbcConnectivity;  
import java.sql.Connection;  
import java.sql.Statement;  
public class InsertQuery {  
    public static void main(String[] args) {  
        // Creating Connection and Statement object in the try with resource block  
        try (Connection con = ConnectDB.dbConnect(); Statement stmt =  
con.createStatement();) {  
            // create query  
            String sql ="INSERT INTO Student  
(StudentID,FirstName,LastName,DateOfBirth,Gender, Email,Phone,marks)  
VALUES\r\n"  
+ " ('S106','lilly', 'Doe','2001-10-10','M', 'lilly@example.com','9878457876',78)";  
;  
            // execute the query  
            int row = stmt.executeUpdate(sql);
```

```

        System.out.println(row + " row inserted successfully!!");
    } catch (Exception e) {
        System.out.println(e);
    }
}
}

```

Output

```

activity/InsertQuery.java - Eclipse IDE
Project Run Window Help
History Synchronize Git Staging Git Reflog Properties JUnit Console x SonarLint Report
<terminated> InsertQuery [Java Application] C:\Users\Asus\AppData\Local\Temp\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.7.v20230425-1502\jre\bin\javaw.exe (15-Sep-2023 10:00:00)
1 row inserted successfully!!

```

```
mysql> select * from student;
```

StudentID	FirstName	LastName	DateOfBirth	Gender	Email	Phone	marks
S101	John	Doe	2000-10-10 00:00:00	M	john@example.com	9878457945	40
S102	Jane	Smith	2013-08-08 00:00:00	M	jane@example.com	9977457745	78
S103	Alice	Johnson	2011-09-08 00:00:00	F	alice@example.com	9876457845	40
S104	Jim	Doe	2011-07-08 00:00:00	F	jim.doe@india.com	9876457845	40
S105	Peter	Parker	2011-06-05 00:00:00	F	p_parker@example.com	9876457845	40
S106	lilly	Doe	2001-10-10 00:00:00	M	lilly@example.com	9878457876	78

6 rows in set (0.00 sec)

Here you can see the record has been inserted successfully.

Executing Update Query

Executing an UPDATE query through JDBC (Java Database Connectivity) allows you to modify data in a relational database. To do this, you'll need to establish a database connection, create a SQL UPDATE statement, and execute it.

Here's a step-by-step tutorial on how to execute an UPDATE query through JDBC:

Step 1: Set Up Your Project

Before you begin, make sure you have the JDBC driver for your database system. You can usually find these drivers on the database vendor's website. Add the JDBC driver JAR file to your project's classpath.

Step 2: Import Required Libraries

In your Java code, you'll need to import the necessary libraries:

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

```

```
import java.sql.SQLException;
```

Step 3: Establish a Database Connection

You need to create a connection to your database. Replace the placeholders with your database URL, username, and password.

```
String jdbcUrl = "jdbc:mysql://localhost:3306/your_database_name";
String username = "your_username";
String password = "your_password";

try {
    Connection connection = DriverManager.getConnection(jdbcUrl, username,
password);
} catch (SQLException e) {
    e.printStackTrace();
}
```

Step 4: Create the SQL UPDATE Statement

Define your SQL UPDATE statement. Replace your_table_name with the name of the table you want to update and set the new values as needed.

```
String updateQuery = "UPDATE your_table_name SET column1 = ?, column2 = ?
WHERE some_condition";
```

Step 5: Prepare and Execute the UPDATE Query

Prepare a Statement with the SQL query and set the new values using placeholders. Execute the update query with executeUpdate().

```
try {
    Statement st = con.createStatement();

    String sql = "update student set FirstName='Lilly steave',Gender='F' where
StudentId='S106'";

    int rowsUpdated = st.executeUpdate(sql);
    System.out.println(rowsUpdated + " rows updated.");
} catch (SQLException e) {
    e.printStackTrace();
}
```

Replace newValue1, newValue2, and some_condition with your specific values and conditions for the update.

Step 6: Close the Database Connection

Always close the database connection when you're done with it to release resources.

```
try {
    if (connection != null) {
        connection.close();
    }
} catch (SQLException e) {
    e.printStackTrace();
}
```

Note: You can use a try-with-resources block to automatically close the database connection without the need for a finally block, reducing boilerplate code. Here's the updated code:

```
try(Connection con=ConnectDB.dbConnect();
Statement st=con.createStatement();) {
```

In this version, both the Connection and Statement are created within the try-with-resources blocks, and they will be automatically closed when the try block exits, whether it exits normally or due to an exception. This eliminates the need for a separate finally block to handle resource cleanup.

Let's explore complete Example:

```
package jdbcConnectivity;
```

```
import java.sql.Connection;
import java.sql.Statement;
```

```
public class updateRecord {
```

```
    public static void main(String[] args) {
        // creating Connection and Statement under Try Block
```

```

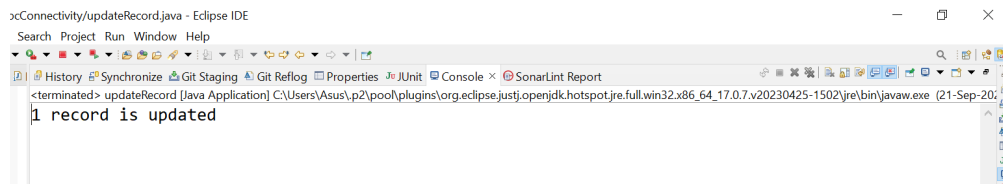
        try (Connection con = ConnectDB.dbConnect(); Statement st =
con.createStatement();) {

            // Creating update query
            String sql = "update student set FirstName='Lilly steave',Gender='F'
where StudentId='S106'";
            // executing query
            int data = st.executeUpdate(sql);
            System.out.println(data + " record is updated");

        } catch (Exception ex) {
            System.out.println(ex);
        }
    }
}

```

Output



```
mysql> select * from student;
```

StudentID	FirstName	LastName	DateOfBirth	Gender	Email	Phone	marks
S101	John	Doe	2000-10-10 00:00:00	M	john@example.com	9878457945	40
S102	Jane	Smith	2013-08-08 00:00:00	M	jane@example.com	9977457745	78
S103	Alice	Johnson	2011-09-08 00:00:00	F	alice@example.com	9876457845	40
S104	Jim	Doe	2011-07-08 00:00:00	F	jim.doe@india.com	9876457845	40
S105	Peter	Parker	2011-06-05 00:00:00	F	p_parker@example.com	9876457845	40
S106	Lilly steave	Doe	2001-10-10 00:00:00	F	lilly@example.com	9878457876	78

6 rows in set (0.00 sec)

Here you can see the record has been updated successfully.

Executing Delete Query

To execute a DELETE query in MySQL using JDBC (Java Database Connectivity), you'll need to establish a database connection, create a SQL DELETE statement, and then execute it.

Here's a step-by-step guide on how to do this:

Step 1: Set Up Your Project

Before you start, make sure you have set up your Java project with the JDBC driver for MySQL. You should have the JDBC driver JAR file in your project's classpath. You can download the MySQL JDBC driver from the official MySQL website.

Step 2: Import Required Libraries

In your Java code, import the necessary JDBC libraries:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
```

Step 3: Establish a Database Connection

Create a connection to your MySQL database by specifying the database URL, username, and password:

```
String jdbcUrl = "jdbc:mysql://localhost:3306/your_database_name";
String username = "your_username";
String password = "your_password";

try {
    Connection connection = DriverManager.getConnection(jdbcUrl, username,
password);
} catch (SQLException e) {
    e.printStackTrace();
}
```

Replace `your_database_name`, `your_username`, and `your_password` with your actual database information.

Step 4: Create the SQL DELETE Statement

Define your SQL DELETE statement, specifying the table and the condition for deletion. For example, to delete rows where a certain condition is met:

```
String deleteQuery = "DELETE FROM your_table_name WHERE some_condition";
```

Replace `your_table_name` with the name of the table from which you want to delete rows and `some_condition` with the condition that identifies the rows to be deleted.

Step 5: Prepare and Execute the DELETE Query

Prepare a Statement with the DELETE query and execute it:

```
try {
    Statement st = con.createStatement();
    String sql = "delete from Student where StudentId='S106'";

    // Execute the DELETE query
    int rowsDeleted = st.executeUpdate(sql);
    System.out.println(rowsDeleted + " rows deleted.");
} catch (SQLException e) {
    e.printStackTrace();
}
```

The executeUpdate method returns the number of rows affected by the DELETE query.

Step 6: Close the Database Connection

Always close the database connection when you're done with it to release resources:

```
try {
    if (connection != null) {
        connection.close();
    }
} catch (SQLException e) {
    e.printStackTrace();
}
```

Let's explore complete Example:

```
package jdbcConnectivity;
import java.sql.Connection;
import java.sql.Statement;
public class DeleteRecord {
    public static void main(String[] args) {
        // creating Connection and Statement under Try Block
        try (Connection con = ConnectDB.dbConnect(); Statement st =
con.createStatement();) {
            // Creating update query
            String sql = "delete from employee where eid='1'";
            // executing query
            int data = st.executeUpdate(sql);
```

```

        System.out.println(data + " record is deleted");
    } catch (Exception ex) {
        System.out.println(ex);
    }
}
}
}

```

Output:

The screenshot shows the Eclipse IDE interface. The console window displays the output of the application: "1 record is deleted". The title bar of the console window indicates the application is "DeleteRecord [Java Application]".

```
mysql> select * from student;
```

StudentID	FirstName	LastName	DateOfBirth	Gender	Email	Phone	marks
S101	John	Doe	2000-10-10 00:00:00	M	john@example.com	9878457945	40
S102	Jane	Smith	2013-08-08 00:00:00	M	jane@example.com	9977457745	78
S103	Alice	Johnson	2011-09-08 00:00:00	F	alice@example.com	9876457845	40
S104	Jim	Doe	2011-07-08 00:00:00	F	jim.doe@india.com	9876457845	40
S105	Peter	Parker	2011-06-05 00:00:00	F	p_parker@example.com	9876457845	40

5 rows in set (0.01 sec)

```
mysql>
```

Here you can see, the last Student record got deleted whose StudentId is S106.

Knowledge check...

Now that we've covered the basic understanding of executing insert, update and delete query, it's time to test your understanding. The Trainer will conduct a short poll quiz to assess your knowledge on these topics.

1. Which SQL command is used to modify existing data in a database?

- A) DELETE
- B) INSERT
- C) UPDATE
- D) ALTER

2.Which keyword is used to add new rows of data to a table in an INSERT statement?

- A) INTO
- B) VALUES
- C) INSERT INTO
- D) ADD

3.In an SQL UPDATE statement, what clause is used to specify the rows to be updated?

- A) SET
- B) WHERE
- C) FROM
- D) CHANGE

At the end of the quiz,the Trainer will provide feedback and explanations to the participants, enhancing the learning experience and understanding of the content.

Food for thought..

The trainer can ask the students the following question to engage them in a discussion:

Encourage students to think about the consequences of not handling errors and how it can impact data integrity and application reliability.

II. Handling Errors in MySql

Handling errors in MySQL can be achieved using SQL constructs and MySQL's error handling capabilities. While MySQL does not have built-in support for advanced procedural error handling like PL/SQL, you can still manage and report errors using SQL statements.

1. Error Messages in MySQL:

MySQL provides error messages that contain an error code and a description of the error. These error messages can be useful for identifying and troubleshooting issues.

Some Examples:

- **If I input an incorrect table name:**

```
mysql> select * from employe;
```

```
ERROR 1146 (42S02): Table 'studentmanagementsystem.employe' doesn't exist
```

- **If I input incorrect column name:**

```
mysql> select uid from employee where uid=102;  
ERROR 1054 (42S22): Unknown column 'uid' in 'field list'
```

Question:What types of errors can occur in SQL, and how are they classified?

2. Primary key and Unique constraints error

In MySQL, primary key and unique constraints ensure that values in specified columns are unique, preventing duplicate entries. When an error related to primary key or unique constraints occurs, it typically means that an attempt was made to insert or update data in a way that violates these uniqueness rules.

Let's explore this with examples:

- **Primary Key Constraint Error:**

A primary key uniquely identifies each row in a table. If an attempt is made to insert a duplicate primary key value, MySQL will generate an error.

Suppose you have a table called students with a primary key constraint on the student_id column:

```
CREATE TABLE students (  
    student_id INT PRIMARY KEY,  
    student_name VARCHAR(50),  
    Email varchar(255),  
    Marks int  
);
```

If you try to insert two rows with the same student_id, you will encounter a primary key constraint error:

-- This will result in an error because student_id 101 already exists.

```
INSERT INTO student (student_id, student_name,Email,marks) VALUES (1,  
'John','john@gmail.com',78);
```

```
mysql> INSERT INTO student (student_id, student_name,Email,marks) VALUES (1, 'John','john@gmail.com',78);  
ERROR 1054 (42S22): Unknown column 'student_id' in 'field list'  
mysql>
```

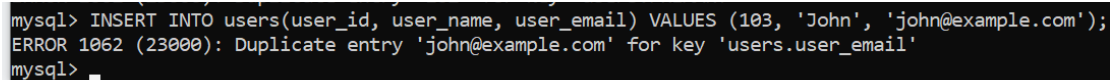
- **Unique Constraint Error:**

A unique constraint ensures that values in the specified column(s) are unique but allows for multiple null values. If a non-null value is duplicated, MySQL will generate an error. Suppose you have a table called employees with a unique constraint on the employee_email column:

```
CREATE TABLE users(  
    user_id INT PRIMARY KEY,  
    user_name VARCHAR(50),  
    user_email VARCHAR(100) UNIQUE  
);
```

If you attempt to insert or update a row with a duplicate email, you will encounter a unique constraint error:

```
-- This will result in an error because the email john@example.com' already exists.  
INSERT INTO users(user_id, user_name, user_email) VALUES (102, 'John',  
john@example.com);
```

A screenshot of a MySQL command prompt window. The prompt shows the command 'INSERT INTO users(user_id, user_name, user_email) VALUES (103, 'John', 'john@example.com');' followed by the error message 'ERROR 1062 (23000): Duplicate entry 'john@example.com' for key 'users.user_email''. The prompt ends with 'mysql>'.

```
mysql> INSERT INTO users(user_id, user_name, user_email) VALUES (103, 'John', 'john@example.com');  
ERROR 1062 (23000): Duplicate entry 'john@example.com' for key 'users.user_email'  
mysql>
```

Handling Primary Key and Unique Constraint Errors:

To handle these errors, you can use error handling techniques. For example, you can use DECLARE CONTINUE HANDLER to catch and handle specific SQL error codes like 1062 (duplicate key) or 23000 (unique constraint violation) and take appropriate actions such as rolling back a transaction, logging the error, or providing a user-friendly error message.

Remember that the exact error code may vary depending on the MySQL server version and configuration, so it's essential to check the specific error code associated with your MySQL environment.

Using DECLARE CONTINUE HANDLER for Specific Errors:

To handle specific errors, you can declare handlers for specific SQL error codes. For instance, to handle a duplicate key error, you can do the following:

```
DECLARE CONTINUE HANDLER FOR 1062 -- Duplicate key error  
BEGIN
```

```
SELECT 'Duplicate key error occurred.';
-- You can add additional handling logic here.
END;
```

```
-- Perform an INSERT operation
INSERT INTO your_table_name (column1, column2) VALUES (value1, value2);
```

In this case, the handler is declared for error code 1062, which corresponds to a duplicate key error. When such an error occurs during the INSERT operation, the handler code is executed.

Note: In our upcoming sessions, we will delve into the concepts of Begin, Declare, and End.

Exercise:

Use ChatGPT to explore Error Handling in database:

Put the below problem statement in the message box and see what ChatGPT says.

How do different database systems handle errors differently, and why?