Day 17- Facilitation Guide

Index

For (1.5 hrs) ILT

During the previous session, we explored Pl/SQL,Stored Procedures, and Triggers.
We covered a variety of important concepts:

PL/SQL: PL/SQL is a powerful extension of SQL used for writing procedural code within a database. It combines SQL for data manipulation with control structures like loops and conditionals.

Stored Procedures: A stored procedure is a precompiled collection of one or more SQL statements stored in the database. It can be executed as a single unit.

Triggers: A trigger is a database object that automatically executes in response to specific database events, such as data modifications (INSERT, UPDATE, DELETE).

In summary, PL/SQL is a procedural language used for writing code within a database, stored procedures are precompiled units of SQL statements for encapsulating logic, and triggers are database objects that automatically respond to events to enforce rules and automate actions. All three play important roles in database development and management.

In this session, we will delve into more advanced SJDBC topics, the concept of prepared Statement, callable, and Using JDBC with other Databases.

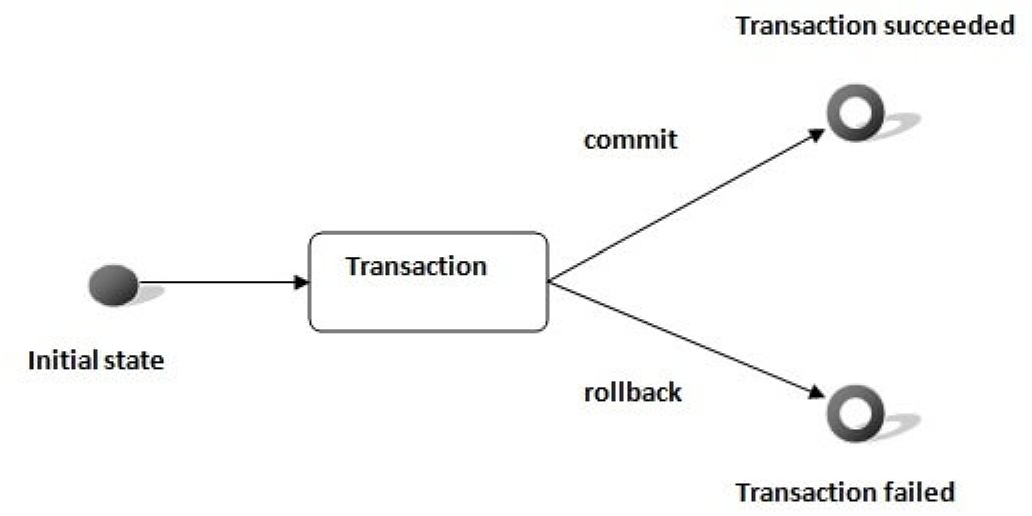
I. Advanced JDBC Topics

Advanced JDBC topics build upon the fundamental concepts of Java Database Connectivity (JDBC) and provide a deeper understanding of database interactions in Java applications.

1. Transaction Management in JDBC

Transaction represents a single unit of work.The ACID properties describe the transaction management well. ACID stands for Atomicity, Consistency, isolation and durability.Atomicity means either all successful or none.Consistency ensures bringing the database from one consistent state to another consistent state.Isolation ensures that a transaction is isolated from other transactions.
Durability means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc.

Advantage of Transaction Management
fast performance It makes the performance fast because the database is hit at the time of commit.



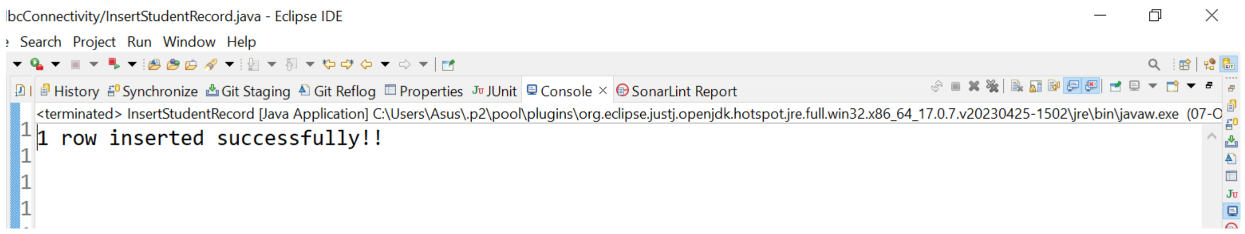
In JDBC, the Connection interface provides methods to manage transactions.

Method	Description
void setAutoCommit(boolean status)	It is true that by default means each transaction is committed by default.
void commit()	commits the transaction.
void rollback()	cancels the transaction.

Let's explore a comprehensive example using JDBC:

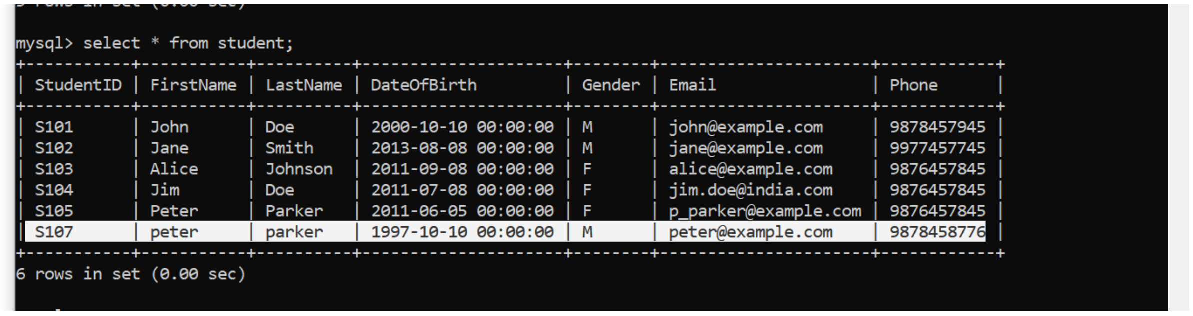
```
package jdbcConnectivity;

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

public class InsertStudentRecord {

	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();
					)

			{
			con.setAutoCommit(false);
			//create query
			String sql="INSERT INTO Student
(StudentID,FirstName,LastName,DateOfBirth,Gender, Email,Phone) VALUES\r\n"
					+ "    ('S107','peter', 'parker','1997-10-10','M',
'peter@example.com','9878458776')";


			//execute the query
			int row=stmt.executeUpdate(sql);
			con.commit();

			System.out.println(row+" row inserted successfully!!");

			}catch (Exception e) {
				System.out.println(e);
			}
}
	}
```

Output:


bcConnectivity/InsertStudentRecord.java - Eclipse IDE
Search Project Run Window Help
History
Synchronize
Git Staging
Git Reflog
Properties
JUnit
Console
SonarLint Report

```
<terminated> InsertStudentRecord [Java Application] C:\Users\Asus\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.7.v20230425-1502\jre\bin\javaw.exe (07-O
1 row inserted successfully!!
```

If you see the table Student Table , you will see that 1 record has been added.



```
mysql> select * from student;
+-----------+-----------+----------+---------------------+--------+----------------------+------------+
| StudentID | FirstName | LastName | DateOfBirth         | Gender | Email                | Phone      |
+-----------+-----------+----------+---------------------+--------+----------------------+------------+
| S101      | John      | Doe      | 2000-10-10 00:00:00 | M      | john@example.com     | 9878457945 |
| S102      | Jane      | Smith    | 2013-08-08 00:00:00 | M      | jane@example.com     | 9977457745 |
| S103      | Alice     | Johnson  | 2011-09-08 00:00:00 | F      | alice@example.com    | 9876457845 |
| S104      | Jim       | Doe      | 2011-07-08 00:00:00 | F      | jim.doe@india.com    | 9876457845 |
| S105      | Peter     | Parker   | 2011-06-05 00:00:00 | F      | p_parker@example.com | 9876457845 |
| S107      | peter     | parker   | 1997-10-10 00:00:00 | M      | peter@example.com    | 9878458776 |
+-----------+-----------+----------+---------------------+--------+----------------------+------------+
6 rows in set (0.00 sec)
```

2. Batch Processing in JDBC

Instead of executing a single query, we can execute a batch (group) of queries. It makes the performance fast. It is because when one sends multiple statements of SQL at once to the database, the communication overhead is reduced significantly, as one is not communicating with the database frequently, which in turn results to fast performance.

The java.sql.Statement and java.sql.PreparedStatement interfaces provide methods for batch processing.

Advantage of Batch Processing

Fast Performance

Methods of Statement interface

The required methods for batch processing are given below:

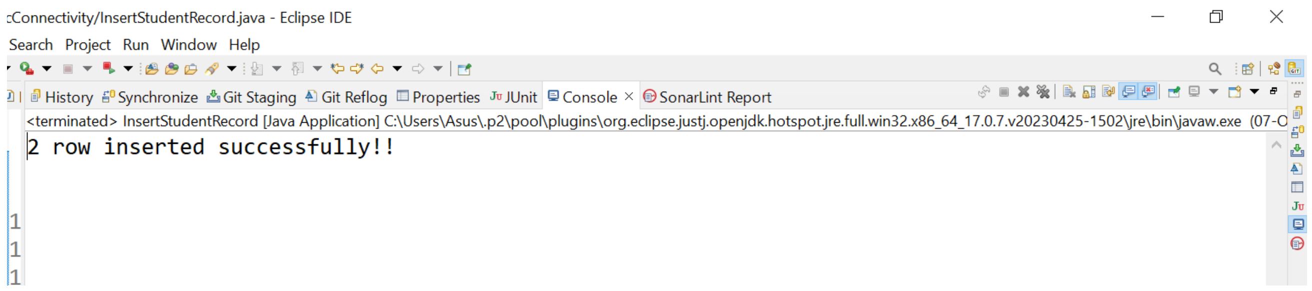
Method	Description
void addBatch(String query)	The addBatch(String query) method of the CallableStatement, PreparedStatement, and Statement is used to single statements to a batch.
int[] executeBatch()	The executeBatch() method begins the execution of all the grouped together statements. The method returns an integer array, and each of the element of the array represents the updated count for respective update statement.
boolean DatabaseMetaData.supportsBatchUpdates() throws SQLException	If the target database facilitates the batch update processing, then the method returns true.
void clearBatch()	The method removes all the statements that one has added using the addBatch() method.

Let's explore a comprehensive example using JDBC:

```
package jdbcConnectivity;

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

public class InsertStudentRecord {

        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();
                                )
```

```
                        {
                        con.setAutoCommit(false);
                        //create query
                        stmt.addBatch("INSERT INTO Student
(StudentID,FirstName,LastName,DateOfBirth,Gender, Email,Phone) VALUES
('S108','Elizabeth', 'parker','1990-10-10','F', 'elizabeth@example.com','9878888776')");
                        stmt.addBatch("INSERT INTO Student
(StudentID,FirstName,LastName,DateOfBirth,Gender, Email,Phone) VALUES
('S109','Grace', 'joe','1987-04-10','F', 'Grace@example.com','9870988776')");

                        stmt.executeBatch();//executing the batch

                        con.commit();


                        System.out.println("2 row inserted successfully!!");

                        }catch (Exception e) {
                                System.out.println(e);
                        }
}
        }
```

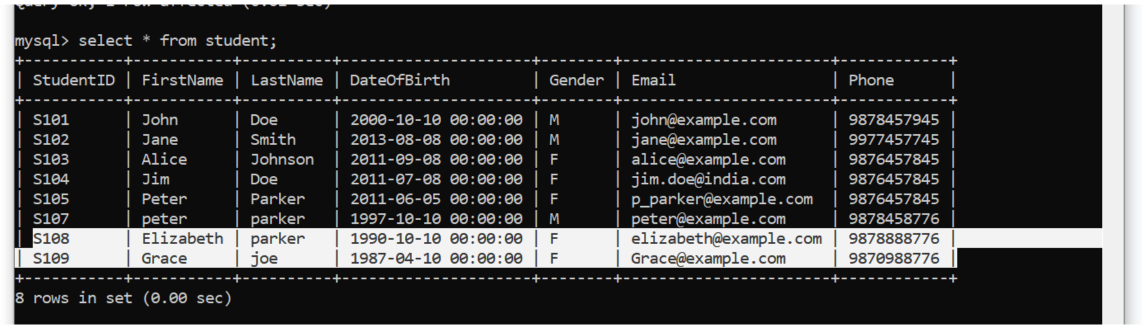
Output:



:Connectivity/InsertStudentRecord.java - Eclipse IDE
Search Project Run Window Help
History Synchronize Git Staging Git Reflog Properties JUnit Console SonarLint Report

```
<terminated> InsertStudentRecord [Java Application] C:\Users\Asus\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.7.v20230425-1502\jre\bin\javaw.exe (07-O
2 row inserted successfully!!
```

If you see the table Student, two records have been added.



```
mysql> select * from student;
+-----------+-----------+----------+---------------------+--------+----------------------+------------+
| StudentID | FirstName | LastName | DateOfBirth         | Gender | Email                | Phone      |
+-----------+-----------+----------+---------------------+--------+----------------------+------------+
| S101      | John      | Doe      | 2000-10-10 00:00:00 | M      | john@example.com     | 9878457945 |
| S102      | Jane      | Smith    | 2013-08-08 00:00:00 | M      | jane@example.com     | 9977457745 |
| S103      | Alice     | Johnson  | 2011-09-08 00:00:00 | F      | alice@example.com    | 9876457845 |
| S104      | Jim       | Doe      | 2011-07-08 00:00:00 | F      | jim.doe@india.com    | 9876457845 |
| S105      | Peter     | Parker   | 2011-06-05 00:00:00 | F      | p_parker@example.com | 9876457845 |
| S107      | peter     | parker   | 1997-10-10 00:00:00 | M      | peter@example.com    | 9878458776 |
| S108      | Elizabeth | parker   | 1990-10-10 00:00:00 | F      | elizabeth@example.com | 9878888776 |
| S109      | Grace     | joe      | 1987-04-10 00:00:00 | F      | Grace@example.com    | 9870988776 |
+-----------+-----------+----------+---------------------+--------+----------------------+------------+
8 rows in set (0.00 sec)
```

II. Prepared Statement

The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized queries.

<u>Let's see the example of parameterized query:</u>

String sql="insert into emp values(?,?,?)";

As you can see, we are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.

<u>Why use PreparedStatement?</u>

Improves performance: The performance of the application will be faster if you use the PreparedStatement interface because the query is compiled only once.

<u>How to get the instance of PreparedStatement?</u>

The prepareStatement() method of Connection interface is used to return the object of PreparedStatement.

Syntax:
public PreparedStatement prepareStatement(String query)throws SQLException{}

<u>Methods of PreparedStatement interface</u>

The important methods of PreparedStatement interface are given below:

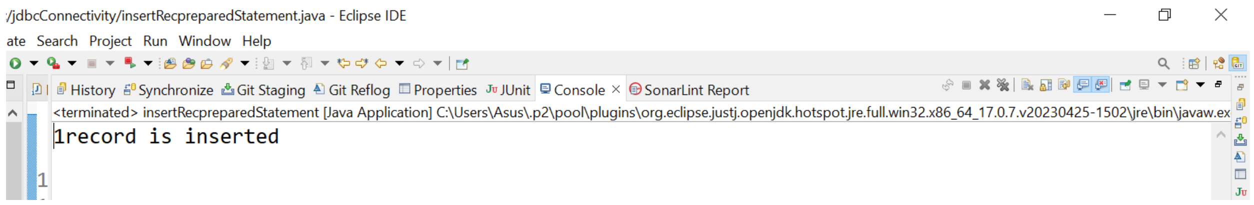
Method	Description
public void setInt(int paramIndex, int value)	sets the integer value to the given parameter index.
public void setString(int paramIndex, String value)	sets the String value to the given parameter index.
public void setFloat(int paramIndex, float value)	sets the float value to the given parameter index.
public void setDouble(int paramIndex, double value)	sets the double value to the given parameter index.
public int executeUpdate()	executes the query. It is used for create, drop, insert, update, delete etc.
public ResultSet executeQuery()	executes the select query. It returns an instance of ResultSet.

Example of PreparedStatement interface that inserts the record

```
package jdbcConnectivity;

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

public class insertRecpreparedStatement {

        public static void main(String[] args) {
                try {
                        Connection con=ConnectDB.dbConnect();
                        String sql="insert into student values(?,?,?,?,?,?,?)";
                        PreparedStatement ps=con.prepareStatement(sql);
```

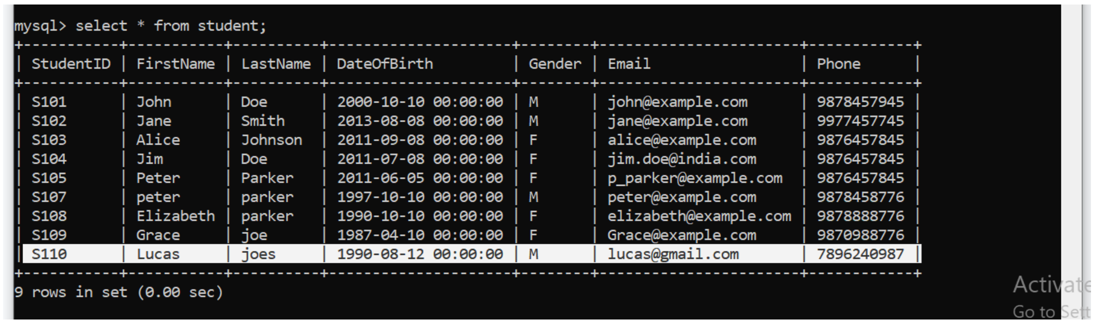
```
                                ps.setString(1, "S110");
                                ps.setString(2, "Lucas");
                                ps.setString(3, "joes");
                                ps.setString(4, "1990-08-12");
                                ps.setString(5, "M");
                                ps.setString(6, "lucas@gmail.com");
                                ps.setString(7, "7896240987");

                                int row=ps.executeUpdate();
                                System.out.println(row+ "record is inserted");
                        }
                        catch(Exception e)
                        {
                                System.out.println(e);
                        }
                }

        }
```

Output:



```
<terminated> insertRecpreparedStatement [Java Application] C:\Users\Asus\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.7.v20230425-1502\jre\bin\javaw.ex
1record is inserted
```

If you see the table Student, one record has been added.



```
mysql> select * from student;
+-----------+-----------+----------+---------------------+--------+-----------------------+------------+
| StudentID | FirstName | LastName | DateOfBirth         | Gender | Email                 | Phone      |
+-----------+-----------+----------+---------------------+--------+-----------------------+------------+
| S101      | John      | Doe      | 2000-10-10 00:00:00 | M      | john@example.com      | 9878457945 |
| S102      | Jane      | Smith    | 2013-08-08 00:00:00 | M      | jane@example.com      | 9977457745 |
| S103      | Alice     | Johnson  | 2011-09-08 00:00:00 | F      | alice@example.com     | 9876457845 |
| S104      | Jim       | Doe      | 2011-07-08 00:00:00 | F      | jim.doe@india.com     | 9876457845 |
| S105      | Peter     | Parker   | 2011-06-05 00:00:00 | F      | p_parker@example.com  | 9876457845 |
| S107      | peter     | parker   | 1997-10-10 00:00:00 | M      | peter@example.com     | 9878458776 |
| S108      | Elizabeth | parker   | 1990-10-10 00:00:00 | F      | elizabeth@example.com | 9878888776 |
| S109      | Grace     | joe      | 1987-04-10 00:00:00 | F      | Grace@example.com     | 9870988776 |
| S110      | Lucas     | joes     | 1990-08-12 00:00:00 | M      | lucas@gmail.com       | 7896240987 |
+-----------+-----------+----------+---------------------+--------+-----------------------+------------+
9 rows in set (0.00 sec)
```

Food for thought:

How does a prepared statement differ from a regular SQL statement or query?

III. Callable Interface

CallableStatement interface is used to call the **stored procedures and functions**.

We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled.

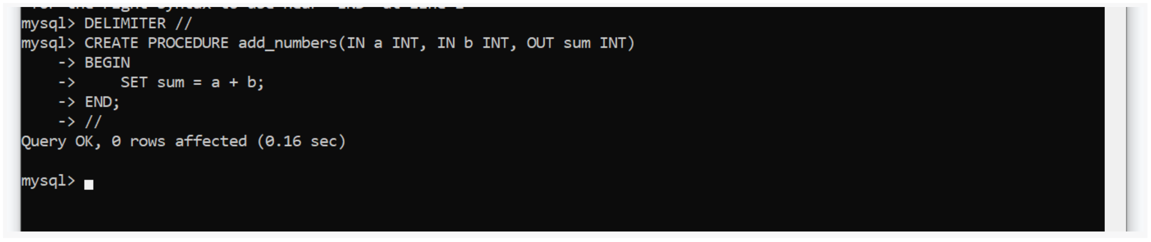
Suppose you need to get the age of the employee based on the date of birth, you may create a function that receives date as the input and returns age of the employee as the output.

Question: What is a CallableStatement, and how does it differ from a PreparedStatement and a Statement in JDBC?

Let's explore a comprehensive example using JDBC:
In this example, we will create a CallableStatement to call a stored procedure in a MySQL database.

Let's assume you have a MySQL stored procedure called add_numbers that takes two integer parameters and returns their sum:

```
DELIMITER //
CREATE PROCEDURE add_numbers(IN a INT, IN b INT, OUT sum INT)
BEGIN
   SET sum = a + b;
END;
//
DELIMITER ;
```



```
mysql> DELIMITER //
mysql> CREATE PROCEDURE add_numbers(IN a INT, IN b INT, OUT sum INT)
    -> BEGIN
    ->     SET sum = a + b;
    -> END;
    -> //
Query OK, 0 rows affected (0.16 sec)

mysql> 
```

Let's break down the code step by step:

1.DELIMITER //: In MySQL, the semicolon (;) is typically used as a statement delimiter. However, when defining stored procedures, triggers, or functions, you often have multiple SQL statements within the body of the procedure. To distinguish between the semicolons used within the procedure and the one used to terminate the entire statement, you can change the delimiter temporarily. In this case, it changes the delimiter to // so that MySQL doesn't treat the semicolon within the procedure body as the end of the statement.

2.CREATE PROCEDURE add_numbers(IN a INT, IN b INT, OUT sum INT): This line defines a new stored procedure named add_numbers. The procedure takes three parameters:

IN a INT: This is an input parameter of type INT named a.
IN b INT: This is an input parameter of type INT named b.
OUT sum INT: This is an output parameter of type INT named sum.

3.BEGIN: This keyword marks the beginning of the procedure's body. All the statements between BEGIN and END constitute the logic of the stored procedure.

4.SET sum = a + b;: This is the main logic of the stored procedure. It calculates the sum of the two input parameters a and b and assigns it to the output parameter sum.

5. END;: This keyword marks the end of the stored procedure's body.

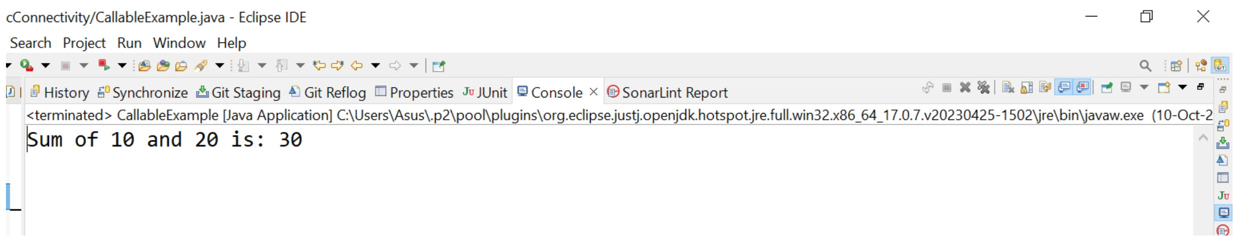
6. //: Since we changed the delimiter earlier with DELIMITER //, this double slash (//) serves as the delimiter for the end of the stored procedure definition.

7. DELIMITER ;: Finally, this line resets the delimiter back to the default semicolon (;) used for individual SQL statements.

Let’s call the stored procedure using jdbc:

```
package jdbcConnectivity;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.Types;
public class CallableExample {
        public static void main(String[] args) {
                try (Connection con = ConnectDB.dbConnect();) {
                        // 3. Prepare the callable statement.
                        String callStatement = "{CALL add_numbers(?, ?, ?)}";
                        CallableStatement callableStatement =
con.prepareCall(callStatement);
                        // 4. Set the input parameters.
                        int a = 10;
                        int b = 20;
                        callableStatement.setInt(1, a);
                        callableStatement.setInt(2, b);
                        // 5. Register the output parameter.
                        callableStatement.registerOutParameter(3, Types.INTEGER);
                        // 6. Execute the stored procedure.
                        callableStatement.execute();
                        // 7. Retrieve the output parameter value.
                        int sum = callableStatement.getInt(3);
                        System.out.println("Sum of " + a + " and " + b + " is: " + sum);
                        // 8. Close the resources.
                        callableStatement.close();
                } catch (Exception e) {
                        System.out.println(e);
                }
        }
}
```

Output:


cConnectivity/CallableExample.java - Eclipse IDE
Search Project Run Window Help
History Synchronize Git Staging Git Reflog Properties JUnit Console SonarLint Report

```
<terminated> CallableExample [Java Application] C:\Users\Asus\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.7.v20230425-1502\jre\bin\javaw.exe (10-Oct-2
Sum of 10 and 20 is: 30
```

Knowledge check…

Now that we've covered PreparedStatement and tables and Callable Interface, it's time to test your understanding. The Trainer will conduct a short poll quiz to assess your knowledge on these topics.

1.Which of the following statements is true regarding prepared statements?

A. They are executed only once and cannot be reused.
B. They are precompiled and can be executed multiple times with different parameter values.
C. They can only be used with SELECT queries, not for INSERT, UPDATE, or DELETE operations.
D. They are more suitable for one-time, complex queries.

2.In a prepared statement, what are placeholders used for?

A. To store the results of a query
B. To specify the order of execution
C. To define input and output parameters
D. To represent values that will be supplied at runtime

3.What is the main purpose of a callable statement in database programming?
A. To execute complex SELECT queries
B. To call stored procedures or functions in the database
C. To retrieve metadata about database tables
D. To perform batch insert operations

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.

IV. Using JDBC with other databases

Let's explore the process of establishing a JDBC connection to an Oracle database.

For connecting java applications with the oracle database, you need to follow 5 steps to perform database connectivity. In this example we are using Oracle as the database. So we need to know following information for the oracle database:

1. Driver class: The driver class for the oracle database is oracle.jdbc.driver.OracleDriver.

2. Connection URL: The connection URL for the oracle database is jdbc:oracle:thin:@localhost:1521:xewhere JDBC is the API, oracle is the database, thin is the driver, localhost is the server name on which oracle is running, we may also use IP address, 1521 is the port number and XE is the Oracle service name.1. You may get all these information from the tnsnames.ora file.

3. Username: The default username for the oracle database is system.
Password: Password is given by the user at the time of installing the oracle database.

4. Password: Password is given by the user at the time of installing the oracle database.

Note: To connect a java application with the Oracle database ojdbc14.jar file is required to be loaded.

First Create Table In oracle:

```
CREATE TABLE Student (
    StudentID VARCHAR2(10) PRIMARY KEY,
    FirstName VARCHAR2(25) NOT NULL,
   LastName  VARCHAR2(25) NOT NULL,
   DateOfBirth Date NOT NULL,
   Gender VARCHAR2(25) NOT NULL,
   Email VARCHAR2(30) UNIQUE NOT NULL,
  Phone VARCHAR2(25) NOT NULL
);
```

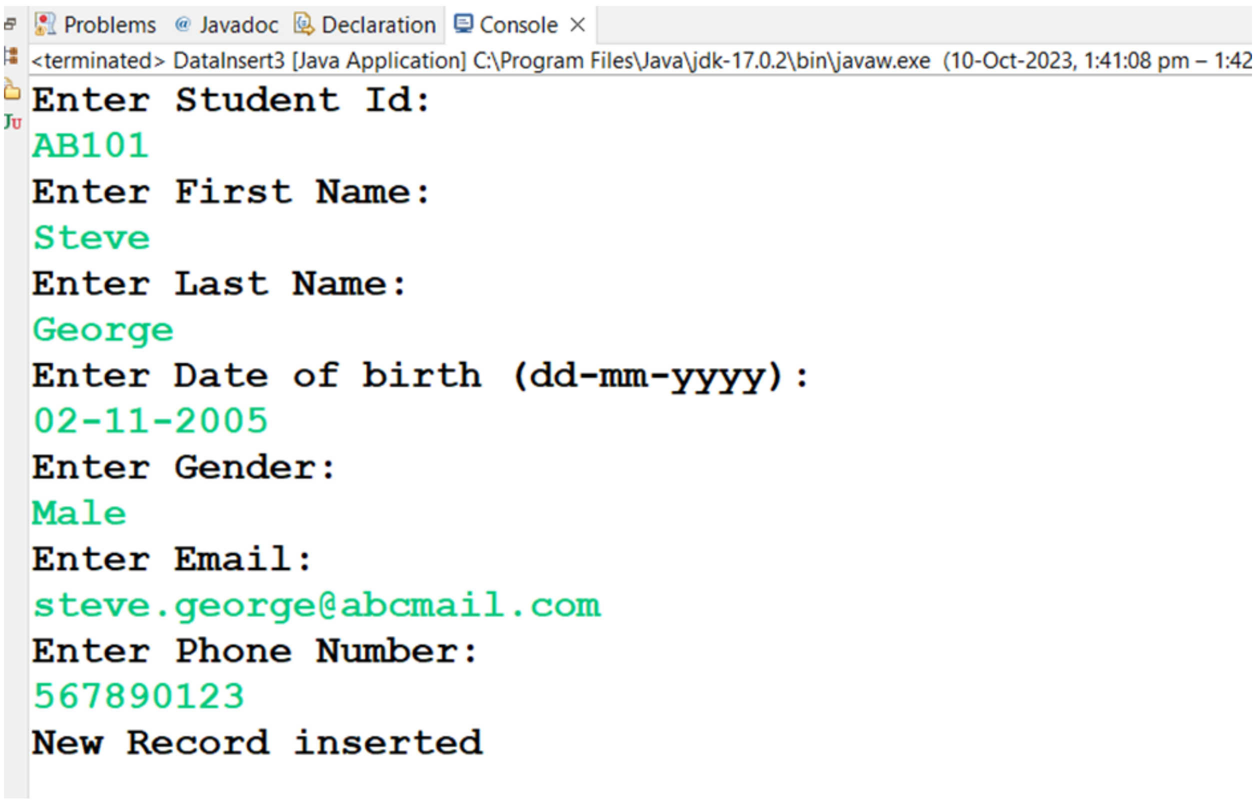
Let us see the complete example using JDBC:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.util.Scanner;

public class DataInsert3 {
 public static void main(String[] args)throws Exception {
          Scanner scanner=new Scanner(System.in);

         //taking input from user
          System.out.println("Enter Student Id:");
```

```
            String id=scanner.nextLine();
            System.out.println("Enter First Name:");
            String fname=scanner.nextLine();
            System.out.println("Enter Last Name:");
            String lname=scanner.nextLine();
            System.out.println("Enter Date of birth (dd-mm-yyyy):");
            String dob=scanner.nextLine();
            System.out.println("Enter Gender:");
            String gender=scanner.nextLine();
            System.out.println("Enter Email:");
            String email=scanner.nextLine();
            System.out.println("Enter Phone Number:");
            String phone=scanner.nextLine();

            //establishing Driver
            Class.forName("oracle.jdbc.driver.OracleDriver");

//establishing Connection
            Connection
connection=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott",
"tiger");

      //Creating query

            String sqlStatement="insert into student values
(?,?,?,to_date(?,'DD-MM-YYYY'),?,?,?)";
            PreparedStatement statement=connection.prepareStatement(sqlStatement);

           //Setting values in all Parameters

            statement.setString(1,id);
            statement.setString(2,fname);
            statement.setString(3,lname);
            statement.setString(4,dob);
            statement.setString(5,gender);
            statement.setString(6,email);
            statement.setString(7,phone);
            statement.executeUpdate();
            System.out.println("New Record inserted");
```

```
          //Closing the connection
           connection.close();
  }
}
```

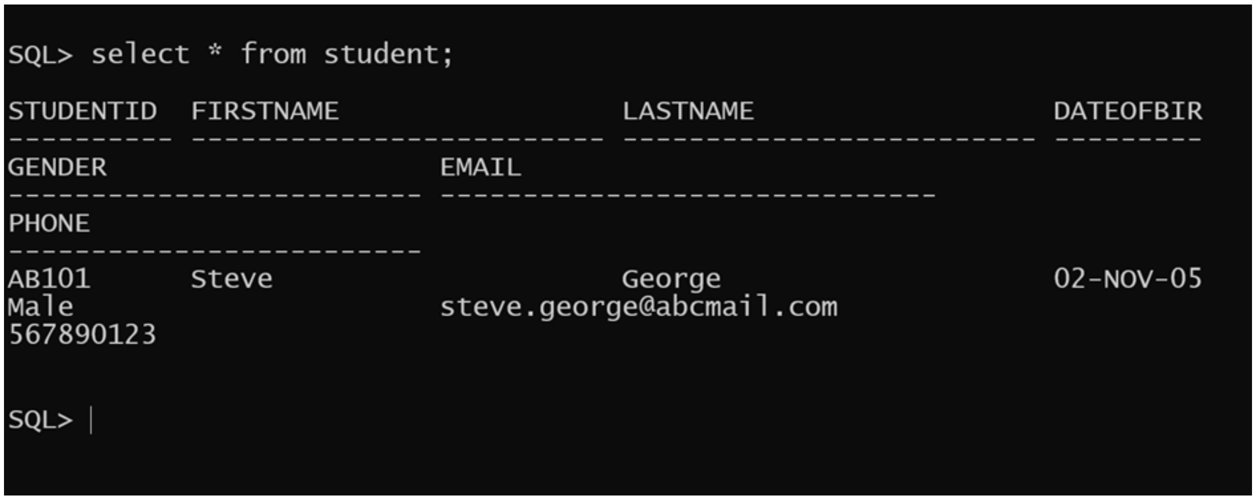
Output:



Problems Javadoc Declaration Console

```
<terminated> DataInsert3 [Java Application] C:\Program Files\Java\jdk-17.0.2\bin\javaw.exe  (10-Oct-2023, 1:41:08 pm – 1:42
Enter Student Id:
AB101
Enter First Name:
Steve
Enter Last Name:
George
Enter Date of birth (dd-mm-yyyy):
02-11-2005
Enter Gender:
Male
Enter Email:
steve.george@abcmail.com
Enter Phone Number:
567890123
New Record inserted
```

Database Console:



```
SQL> select * from student;

STUDENTID  FIRSTNAME                 LASTNAME                  DATEOFBIR
---------- ------------------------- ------------------------- ---------
GENDER                    EMAIL
------------------------- ------------------------------
PHONE
-------------------------
AB101      Steve                     George                    02-NOV-05
Male                      steve.george@abcmail.com
567890123


SQL>
```

Exercise:

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

I've been studying MySQL databases, but I've developed an interest in learning Oracle as well. Could you provide me with some tutorials for Oracle Database?