Viet Nam National University HCM CITY



**University of Information Technology**

**FACULTY OF INFORMATION SYSTEMS**

🙡🕮🙣

**SUBJECT REPORT OF**

**DATABASE MANAGEMENT SYSTEM**

**Logo

Description automatically generated**

**TOPIC:**

**ORACLE JDBC**

Practical instructor: Dr. Nguyen Thanh Binh

Class: IS201.M22.HTCL

|  |  |  |
| --- | --- | --- |
| **Full Name** |  | **Student code** |
| Tran Anh Huy |  | 20520551 |
| Ho Chi Minh City, May 2022 | | |

**CONTENTS**

[1. ORACLE JDBC DESCRIPTION 2](#_Toc105399278)

[1.1. Introduction 2](#_Toc105399279)

[1.2. Overview of Oracle JDBC Drivers 2](#_Toc105399280)

[1.3. Choosing the Appropriate Driver 3](#_Toc105399281)

[1.4. Supported JDK and JDBC Versions 4](#_Toc105399282)

[1.5. RDBMS and JDK Version Compatibility for Oracle JDBC Drivers 4](#_Toc105399283)

[1.6. JDBC và IDEs 5](#_Toc105399284)

[2. USING ORACLE JDBC 5](#_Toc105399285)

[2.1. Determining the Version of the JDBC Driver 5](#_Toc105399286)

[2.2. Importing Packages 6](#_Toc105399287)

[2.3. Opening a Connection to a Database 7](#_Toc105399288)

[2.4. Creating a Statement Object 7](#_Toc105399289)

[2.5. Running a Query and Retrieving a Result Set Object 8](#_Toc105399290)

[2.6. Processing the Result Set Object 8](#_Toc105399291)

[2.7. Closing the Result Set and Statement Objects 9](#_Toc105399292)

[2.8. Making Changes to the Database 9](#_Toc105399293)

[2.9. Closing the Connection 11](#_Toc105399294)

[2.10. Stored Procedure Calls in JDBC Programs 11](#_Toc105399295)

[2.11. About Processing SQL Exceptions 12](#_Toc105399296)

[2.12. Oracle Extensions 14](#_Toc105399297)

[2.13. Oracle JDBC Packages 15](#_Toc105399298)

[2.14. Sample: Connecting, Querying, Processing the Results, DML operations and PL/SQL Stored Procedures 15](#_Toc105399299)

[REFERENCES 22](#_Toc105399300)

1. **ORACLE JDBC DESCRIPTION**
   1. **Introduction**

Java Database Connectivity (JDBC) is a Java standard that provides the interface for connecting from Java to relational databases.

The JDBC standard is defined and implemented through the standard java.sql interfaces. This enables individual providers to implement and extend the standard with their own JDBC drivers.

* 1. **Overview of Oracle JDBC Drivers**

Oracle provides the following JDBC drivers:

* Thin driver:

The JDBC Thin driver is a pure Java, Type IV driver that can be used in applications. It is platform-independent and does not require any additional Oracle software on the client-side. The JDBC Thin driver communicates with the server using Oracle Net Services to access Oracle Database.

The JDBC Thin driver enables a direct connection to the database by providing an implementation of Oracle Net Services on top of Java sockets. The driver supports the TCP/IP protocol and requires a TNS listener on the TCP/IP sockets on the database server.

* Oracle Call Interface (OCI) driver:

It is used on the client-side with an Oracle client installation. It can be used only with applications.

The JDBC OCI driver is a Type II driver used with Java applications. It requires platform-specific OCI libraries. It supports all installed Oracle Net adapters, including interprocess communication (IPC), named pipes, TCP/IP, and Internetwork Packet Exchange/Sequenced Packet Exchange (IPX/SPX).

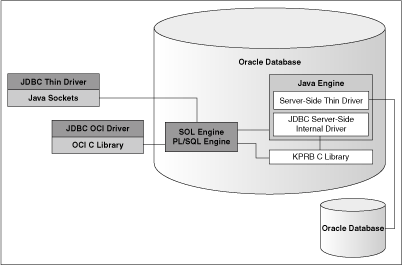
The JDBC OCI driver, written in a combination of Java and C, converts JDBC invocations to calls to OCI, using native methods to call C-entry points. These calls communicate with the database using Oracle Net Services.

* Server-side Thin driver:

The JDBC server-side Thin driver offers the same functionality as the JDBC Thin driver that runs on the client-side. However, the JDBC server-side Thin driver runs inside Oracle Database and accesses a remote database or a different session on the same database for use with Java in the database.

* Server-side internal driver:

The JDBC server-side internal driver supports any Java code that runs inside Oracle Database, such as in a Java stored procedure, and accesses the same database. It lets the Oracle Java Virtual Machine (Oracle JVM) to communicate directly with the SQL engine for use with Java in the database.



*Figure 1-1 Architecture of Oracle JDBC Drivers and Oracle Database"*

* 1. **Choosing the Appropriate Driver**

Consider the following when choosing a JDBC driver for your application or applet:

* In general, unless you need OCI-specific features, such as support for non-TCP/IP networks, use the JDBC Thin driver.
* If you want maximum portability and performance, then use the JDBC Thin driver. You can connect to Oracle Database from an application using the JDBC Thin driver.
* If you want to use Lightweight Directory Access Protocol (LDAP) over Secure Sockets Layer (SSL)/Transport Layer Security (TLS), then use the JDBC Thin driver.
* If you are writing a client application for an Oracle client environment and need OCI-driver-specific features, such as support for non-TCP/IP networks, then use the JDBC OCI driver.
* For code that runs in the database server and needs to access a remote database or another session within the same database instance, use the JDBC server-side Thin driver.
* If your code runs inside the database server and needs to access data locally within the session, then use the JDBC server-side internal driver to access that server.
  1. **Supported JDK and JDBC Versions**

In Oracle Database 21c, all the JDBC drivers are compatible with JDK 8, JDK 11, JDK 12, JDK 13, JDK 14, and JDK 15, and the ojdbc8.jar and ojdbc11.jar files provide the support to these JDK versions.

* When to Use ojdbc8.jar File

Use the ojdbc8.jar file when you want JDBC 4.2 features and need to compile your code with JDK 8, JDK11, JDK12, JDK13, JDK14, and JDK15.

* When to Use ojdbc11.jar File

Use the ojdbc11.jar file when you want JDBC 4.3 features and need to compile your code with JDK 11, JDK 12, JDK13, JDK14, and JDK15.

* 1. **RDBMS and JDK Version Compatibility for Oracle JDBC Drivers**

The following table describes the JDBC and Oracle Database interoperability matrix or the certification matrix:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| JDBC Driver Version | Database 21.x | Database 19.x | Database 18.3 | Database 12.2 and 12.1 | Database 11.2.0.4 |
| JDBC 21.x | Yes | Yes | Yes | Yes | Yes |
| JDBC 19.x | Yes | Yes | Yes | Yes | Yes |
| JDBC 18.3 | Yes | Yes | Yes | Yes | Yes |
| JDBC 12.2 and 12.1 | Yes | Yes | Yes | Yes | Yes |
| JDBC 11.2.0.4 | Yes | Yes | Yes | Yes | Yes |

Oracle JDBC Drivers are always compliant to the latest JDK version for every new release. For some versions, JDBC drivers support multiple JDK versions. The following table describes the release-specific JDBC JAR files and supported JDK versions for various Oracle Database versions:

|  |  |
| --- | --- |
| Oracle Database Version | Release-Specific JDBC JAR File with Supported JDK Version |
| 21.x | - ojdbc11.jar with JDK 11, JDK 12, JDK 13, JDK 14 and JDK 15  - ojdbc8.jar with JDK 8, JDK 11, JDK 12, JDK 13, JDK 14 and JDK 15 |
| 19.x | - ojdbc10.jar with JDK 10, JDK11  - ojdbc8.jar with JDK 8, JDK 9, JDK 11 |
| 18.3 | ojdbc8.jar with JDK 8, JDK 9, JDK 10, JDK 11 |
| 12.2 or 12cR2 | ojdbc8.jar with JDK 8 |
| 12.1 or 12cR1 | - ojdbc7.jar with JDK 7, JDK 8  - ojdbc6.jar with JDK 6 |
| 11.2 or 11gR2 | - ojdbc6.jar with JDK 6, JDK 7, JDK 8  - ojdbc5.jar with JDK 5 |

* 1. **JDBC và IDEs**

The Oracle JDeveloper Suite provides developers with a single, integrated set of products to build, debug, and deploy component-based database applications for the Internet. The Oracle JDeveloper environment contains integrated support for JDBC, including the JDBC Thin driver and the native OCI driver. The database component of Oracle JDeveloper uses the JDBC drivers to manage the connection between the application running on the client and the server.

1. **USING ORACLE JDBC**
   1. **Determining the Version of the JDBC Driver**

To determine the version of the JDBC driver, call the getDriverVersion method of the OracleDatabaseMetaData class as shown in the following sample code:

import java.sql.\*;

import oracle.jdbc.\*;

import oracle.jdbc.pool.OracleDataSource;

class JDBCVersion

{

public static void main (String args[]) throws SQLException

{

OracleDataSource ods = new OracleDataSource();

ods.setURL("jdbc:oracle:thin:HR/hr@<host>:<port>:<service>");

Connection conn = ods.getConnection();

// Create Oracle DatabaseMetaData object

DatabaseMetaData meta = conn.getMetaData();

// gets driver info:

System.out.println("JDBC driver version is " + meta.getDriverVersion());

}

}

You can also determine the version of the JDBC driver by executing the following commands:

* java -jar ojdbc8.jar
* java -jar ojdbc11.jar
  1. **Importing Packages**

Regardless of which Oracle JDBC driver you use, include the import statements shown in Table 3-1 at the beginning of your program using the following syntax:

import <*package\_name*>;

|  |  |
| --- | --- |
| Import statement | Provides |
| import java.sql.\*; | Standard JDBC packages. |
| import java.math.\*; | The BigDecimal and BigInteger classes. You can omit this package if you are not going to use these classes in your application. |
| import oracle.jdbc.\*;  import oracle.jdbc.pool.\*;  import oracle.sql.\*; | Oracle extensions to JDBC. This is optional.  OracleDataSource.  Oracle type extensions. This is optional. |

*Table 3-1 Import Statements for JDBC Driver*

* 1. **Opening a Connection to a Database**

First, specifying a Database URL, User Name, and Password

The following example connects user “c##baocaodoan” with password “password” to a database with service orcl through port 1521 of the host localhost, using the JDBC Thin driver:

String url = "jdbc:oracle:thin:@localhost:1521:orcl";

String user = "c##baocaodoan";

String password = "password";

Connection con = DriverManager.getConnection(url, user, password);

* 1. **Creating a Statement Object**

Once you connect to the database and, in the process, create a Connection object, the next step is to create a Statement object. The createStatement method of the JDBC Connection object returns an object of the JDBC Statement type. To continue the example from the previous section, where the Connection object conn was created, here is an example of how to create the Statement object:

Statement stmt = con.createStatement();

* 1. **Running a Query and Retrieving a Result Set Object**

To query the database, use the executeQuery method of the Statement object. This method takes a SQL statement as input and returns a JDBC ResultSet object.

You create the Statement object stmt, the next step is to run a query that returns a ResultSet object with the contents of SQL statement.

ResultSet rset = stmt.executeQuery ("SQL statement");

* 1. **Processing the Result Set Object**

Once you run your query, use the next() method of the ResultSet object to iterate through the results. This method steps through the result set row by row, detecting the end of the result set when it is reached.

while (rset.next()){

//Command processing

}

* 1. **Closing the Result Set and Statement Objects**

You must explicitly close the ResultSet and Statement objects after you finish using them. This applies to all ResultSet and Statement objects you create when using Oracle JDBC drivers. If you do not explicitly close the ResultSet and Statement objects, serious memory leaks could occur. You could also run out of cursors in the database. Closing both the result set and the statement releases the corresponding cursor in the database. If you close only the result set, then the cursor is not released.

For example, if your ResultSet object is rset and your Statement object is stmt, then close the result set and statement with the following lines of code:

rset.close();

stmt.close();

* 1. **Making Changes to the Database**
* **DML Operations**

To perform DML (Data Manipulation Language) operations, such as INSERT or UPDATE operations, you can create either a Statement object or a PreparedStatement object. PreparedStatement objects enable you to run a statement with varying sets of input parameters. The prepareStatement method of the JDBC Connection object lets you define a statement that takes variable bind parameters and returns a JDBC PreparedStatement object with your statement definition.

The following example shows how to use a prepared statement to run INSERT operations that add two rows to the EMPLOYEES table:

// Prepare to insert new names in the EMPLOYEES table

PreparedStatement pstmt = null;

try{

pstmt = conn.prepareStatement ("insert into EMPLOYEES (EMPLOYEE\_ID, FIRST\_NAME) values (?, ?)");

// Add ANNA as employee number 1

pstmt.setInt (1, 1); // The first ? is for EMPLOYEE\_ID

pstmt.setString (2, "ANNA"); // The second ? is for FIRST\_NAME

// Do the insertion

pstmt.execute();

// Add PERCY as employee number 2

pstmt.setInt (1, 2); // The first ? is for EMPLOYEE\_ID

pstmt.setString (2, "PERCY"); // The second ? is for FIRST\_NAME

// Do the insertion

pstmt.execute();

}

finally{

if(pstmt!=null)

// Close the statement

pstmt.close();

}

* **DDL operations**

To perform data definition language (DDL) operations, you must create a Statement object. The following example shows how to create a table in the database:

//create table EMPLOYEES with columns EMPLOYEE\_ID and FIRST\_NAME

String query;

Statement stmt=null;

try{

query="create table EMPLOYEES " +

"(EMPLOYEE\_ID int, " +

"FIRST\_NAME varchar(50))";

stmt = conn.createStatement();

stmt.executeUpdate(query);

}

finally{

//close the Statement object

stmt.close();

}

* 1. **Closing the Connection**

You must close the connection to the database after you have performed all the required operations and no longer require the connection. You can close the connection by using the close method of the Connection object, as follows:

conn.close();

* 1. **Stored Procedure Calls in JDBC Programs**

JDBC supports the invocation of PL/SQL procedures/functions and anonymous blocks, using either JDBC escape syntax or PL/SQL block syntax. The following PL/SQL calls would work with any Oracle JDBC driver:

// JDBC escape syntax

CallableStatement cs1 = conn.prepareCall

( "{call proc (?,?)}" ) ; // stored proc

CallableStatement cs2 = conn.prepareCall

( "{? = call func (?,?)}" ) ; // stored func

// PL/SQL block syntax

CallableStatement cs3 = conn.prepareCall

( "begin proc (?,?); end;" ) ; // stored proc

CallableStatement cs4 = conn.prepareCall

( "begin ? := func(?,?); end;" ) ; // stored func

* 1. **About Processing SQL Exceptions**

To handle error conditions, Oracle JDBC drivers throw SQL exceptions, producing instances of the java.sql.SQLException class or its subclass. Errors can originate either in the JDBC driver or in the database itself. Resulting messages describe the error and identify the method that threw the error. Additional run-time information can also be appended.

Basic exception handling can include retrieving the error message, retrieving the error code, retrieving the SQL state, and printing the stack trace. The SQLException class includes functionality to retrieve all of this information, when available.

* **Retrieving Error Information**

You can retrieve basic error information with the following methods of the SQLException class:

* *getMessage* class includes functionality to retrieve all of this information, when available.
* *getErrorCode* class includes functionality to retrieve all of this information, when available.
* *getSQLState* class includes functionality to retrieve all of this information, when available.

The following example prints output from a getMessage method call:

catch(SQLException e)

{

System.out.println("exception: " + e.getMessage());

}

This would print the output, such as the following, for an error originating in the JDBC driver:

exception: Invalid column type

* **Printing the Stack Trace**

The SQLException class provides the printStackTrace() method for printing a stack trace. This method prints the stack trace of the Throwable object to the standard error stream. You can also specify a java.io.PrintStream object or java.io.PrintWriter object for output.

The following code fragment illustrates how you can catch SQL exceptions and print the stack trace.

try { <some code> }

catch(SQLException e) { e.printStackTrace (); }

To illustrate how the JDBC drivers handle errors, assume the following code uses an incorrect column index:

// Iterate through the result and print the employee names

// of the code

try {

while (rset.next ())

System.out.println (rset.getString (5)); // incorrect column index

}

catch(SQLException e) { e.printStackTrace (); }

Assuming the column index is incorrect, running the program would produce the following error text:

java.sql.SQLException: Invalid column index

at oracle.jdbc.OracleDriver.OracleResultSetImpl.getDate(OracleResultSetImpl.java:1556)

* 1. **Oracle Extensions**

Beyond standard features, Oracle JDBC drivers provide Oracle-specific type extensions and performance extensions. These extensions are provided through the following Java packages:

* oracle.sql

Provides classes that represent SQL data in Oracle format.

* oracle.jdbc

Provides interfaces to support database access and updates in Oracle type formats.

**Features of the Oracle Extensions:**

* Database Management Using JDBC
* Support for Oracle Data Types
* Support for Oracle Objects
* Support for Schema Naming
* DML Returning
* PL/SQL Associative Arrays
  1. **Oracle JDBC Packages**
* **Package oracle.sql**

General oracle.sql.\* Data Type Support:

* Data storage as Java byte arrays for SQL data
* A getBytes() method, which returns the SQL data as a byte array
* A toJdbc() method that converts the data into an object of a corresponding Java class as defined in the JDBC specification
* The JDBC driver does not convert Oracle-specific data types that are not part of the JDBC specification, such as BFILE. The driver returns the object in the corresponding oracle.sql.\* format
* **Package oracle.sql.json**

The oracle.sql.json package provides functionality to work with the JSON type values:

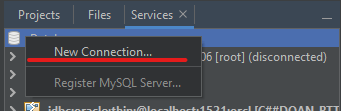
* Store and retrieve JSON type values in the database
* Read, create, and modify JSON type values
* Encode or decode JSON type values in the same binary JSON storage format as used by the database
* Convert JSON type values to and from JSON text
* Bind and access JSON type values using the JSON-P interfaces like javax.json.\*
* **Package oracle.jdbc**

The interfaces of the oracle.jdbc package define the Oracle extensions to the interfaces in java.sql. These extensions provide access to Oracle SQL-format data and other Oracle-specific functionality, including Oracle performance enhancements.

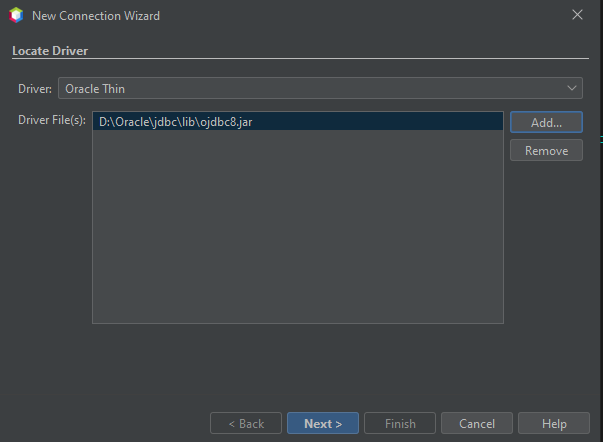
* 1. **Sample: Connecting, Querying, Processing the Results, DML operations and PL/SQL Stored Procedures**

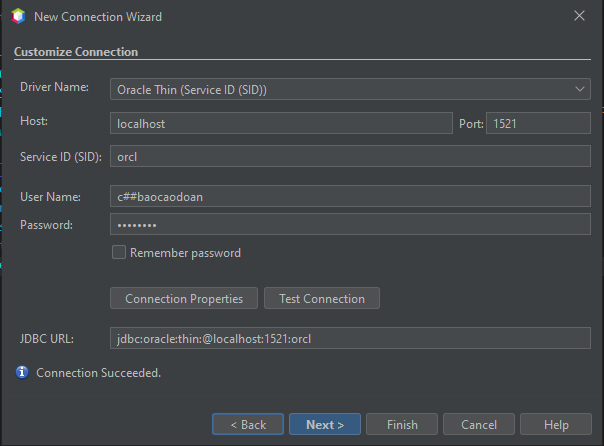
Specifically, illustration with Netbeans IDE through Server connection.

* Step 1. Select Services and Right-click on Database and select New Connection.



* Step 2. Select Oracle Thin Driver and Add Driver (.jar file) (Suitable) and Select Next.



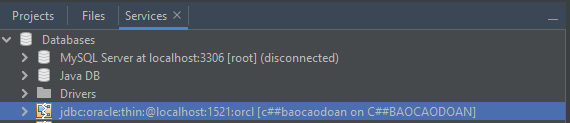
* ****Step 3. Enter the required information.

Check connection

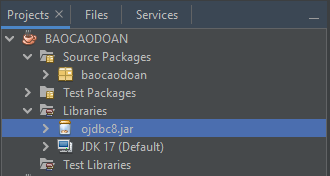
Click Next

Successful connection

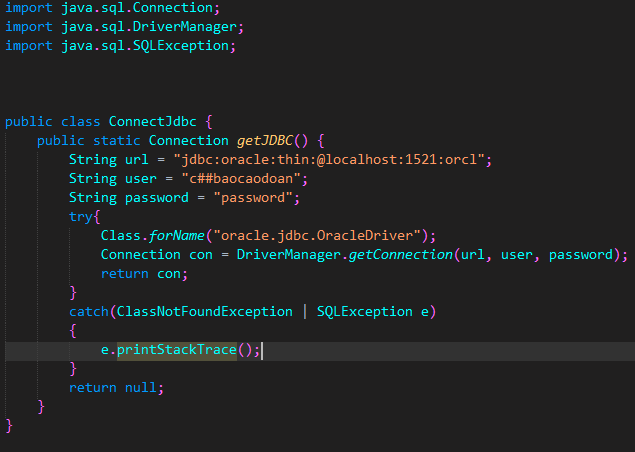
Enter the same username and password as the Oracle you entered

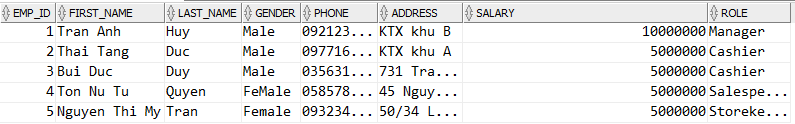
****Successful connection.

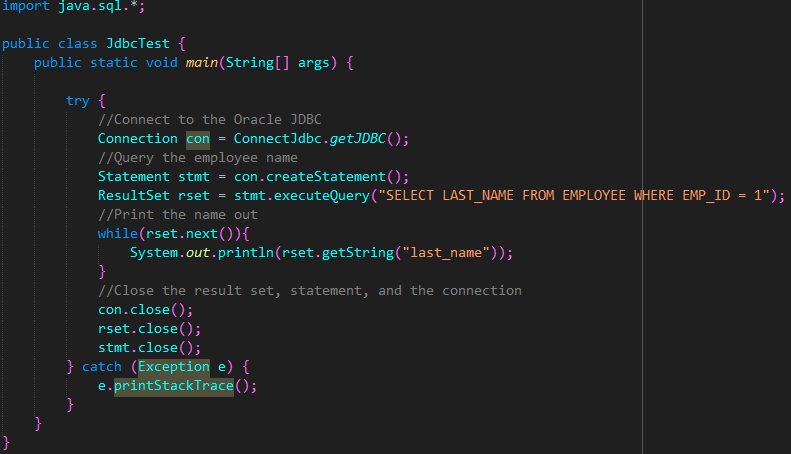
* Step 4. Add Driver (.jar file) to the Libraries.



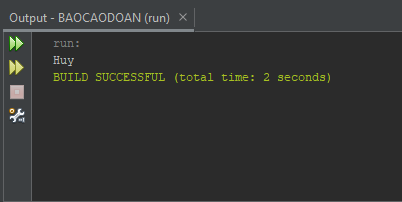
* Step 5. Create connect to Oracle in Java program.

****

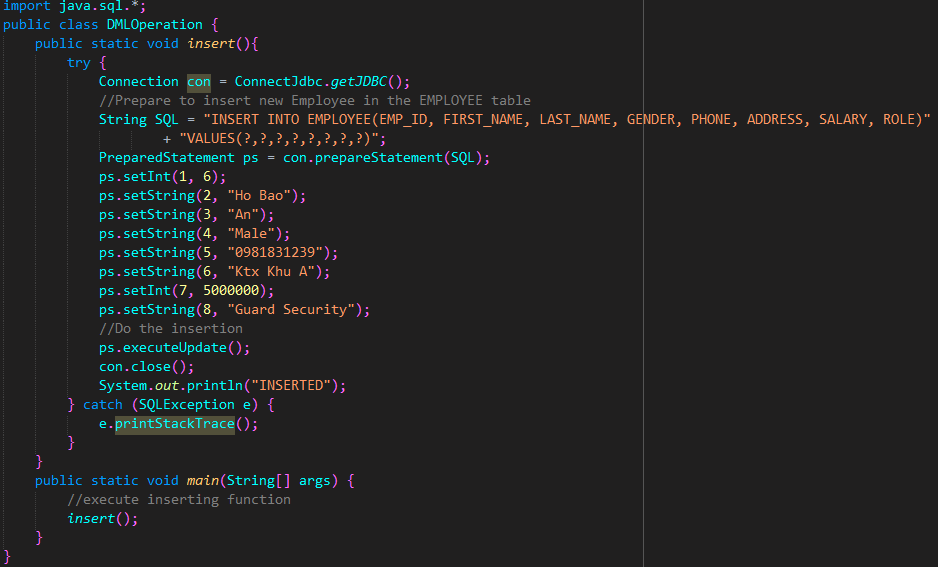
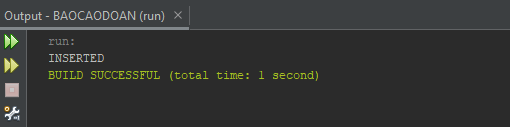
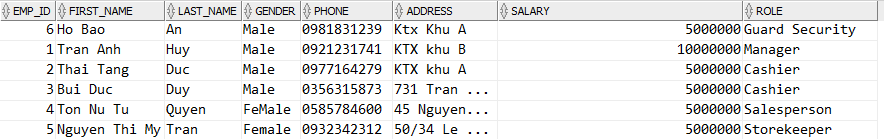
* **Sample: Database**
* **Sample: Connecting, Querying, Processing the Results**



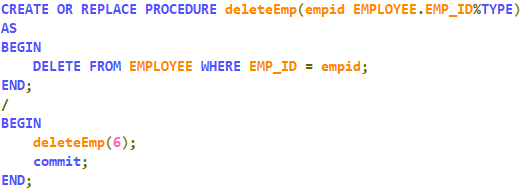
Result:

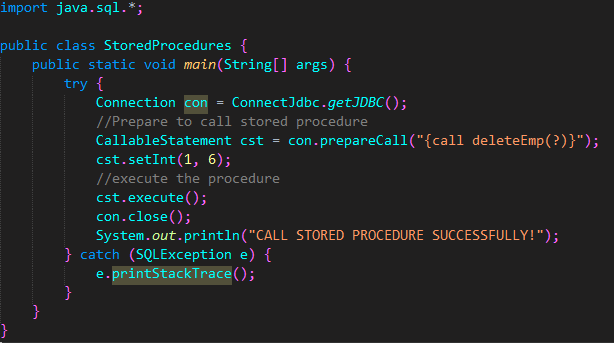


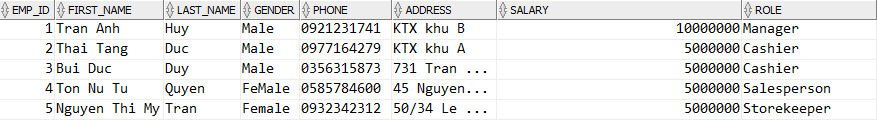
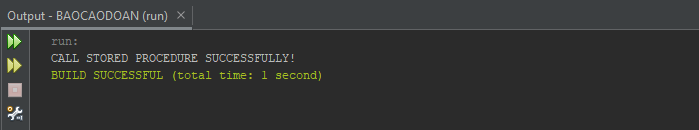
* **Sample: DML operations**

Result:

* **Sample: PL/SQL Stored Procedures**

****

****

Result: 

# **REFERENCES**

[1] Nguyen Thanh Binh, “Oracle & PL-SQL.” University of Information

Technology VNU-HCM, Feb. 01, 2020. [Online]. Available:

[https://drive.google.com/file/d/1O96IHl3\_GPxqPneBmJWEN2VyzFte4jM/view?usp=sharing](https://drive.google.com/file/d/1O96IHl3_GPxqPneBmJWEN2VyzFte-4jM/view)

[2] Oracle, “JDBC Developer's Guide and Reference.”.

<https://docs.oracle.com/en/database/oracle/oracle-database/21/jjdbc/lot.html>