Day11: Java Database Connectivity (java.sql package)

- MySQL
- JDBC Driver Types
- Steps for Database Connectivity
- Statement, PreparedStatement and CallableStatement

MySQL

Downloading MySQL Server

https://dev.mysql.com/downloads/installer/

MySQL Server

1. Execute the Setup.
2. Select Custom Installation, click on Next.
3. Within Check Requirements, click on "Execute" and install all the requirements and then click on Finish.

NOTE: MySQL Server, MySQL WorkBench and MySQL JConnector are the only required tools, remaining can be ignored.

<u>To Verify</u>

- Start | Run | MySQL 8.o Command Line Client
- Specify the password and press enter key.

MySQL WorkBench

- It is tool/utility used to work on MySQL Server
- To Open MySQL WorkBench
  Start | Run | MySQL WorkBench

To Create a New Database/Schema

1. Within Navigator | Schemas | Create Schema
2. Specify the Schema Name
3. Click on Apply

To perform any operation on Schema, it needs to activated or selected

Within Navigator | Schemas | Select the Schema | Set As Default Schema

To Backup the Schema

1. Within Navigator | Data Export
2. Select the Schema and Objects to be exported
3. Select Export to Self Contained File
4. Specify the name and location of the SQL file by clicking on the ...
5. Select Start Export

To Import/Restore the Schema

1. Within Navigator | Data Import/Restore
2. Select Import from Self Contained File
3. Specify the name and location of the SQL file by clicking on the ...
4. Select Start Import

To Create a New Table

1. Within Navigator | Schemas | Select the Schema | Select Tables | Create Table

2. Specify Table Name and column details and click on Apply

To Alter/Delete/Truncate Table
Within Navigator | Schemas | Select the Schema | Select Tables | Alter/Delete/Truncate Table

Sample Table: categories

categoryid	int
categoryname	varchar(45)
categorydetails	varchar(100)

Insert: To insert row in a table
1. Inserting values of all columns
Insert into categories values(1,'Choclates','Sweet Choclates');

2. Inserting values of selected columns
Insert into categories(categoryname, categorydetails) values('Ice-Cream','Frozen Icecream');

Delete: To Delete rows of a table
1. Deleting all rows
Delete from categories;

2. Deleting Selected rows
Delete from categories where categoryid=1;

Update: To Update rows of a Table
1. Updating all rows
update categories set categorydetails='';

2. Updating a selected row
update categories set categoryname='Wafers', categorydetails='Salted Wafers' where categoryid=1;

Select: To fetch rows of a table
1. All rows, All Columns
Select * from categories;

2. All rows, selected Columns
Select categoryname from categories;
3. Selected rows, all columns
Select * from categories where categoryid=1;

4. Selected row, Selected columns
Select categoryname from categories where categoryid=1;

MySQL: Assignment

1. Create a database by the name **staff** in MySQL.

2. Create two tables within the **staff** database.

   a. **departments**

deptid	integer	auto number	primary key
deptname	varchar(45)		
deptdetails	varchar(45)		

   b. **employees**

empid	integer	auto number	primary key
empname	varchar(45)		
deptid	integer		foreign key
designation	varchar(45)		
doj	date		
salary	integer		

3. Insert 3 rows in department table with deptname Admin, Sales and Marketing.

4. Insert 15 rows in employees table, 5 in each department.

5. Take back up of the staff database and try to restore it.

Java Database Connnectivity

What is JDBC Driver?

JDBC drivers implement the defined interfaces in the JDBC API, for interacting with your database server. For example, using JDBC drivers enable you to open database connections and to interact with it by sending SQL or database commands then receiving results with Java.

The Java.sql package that ships with JDK, contains various classes with their behaviours defined and their actual implementations are done in third-party drivers. Third party vendors implement the java.sql.Driver interface in their database driver

.

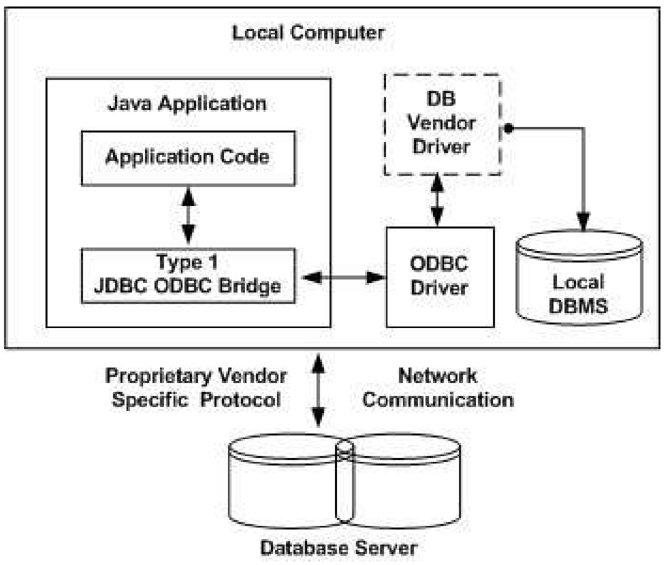
JDBC Drivers Types

JDBC driver implementations vary because of the wide variety of operating systems and hardware platforms in which Java operates. Sun has divided the implementation types into four categories, Types 1, 2, 3, and 4, which is explained below –

Type 1: JDBC-ODBC Bridge Driver

In a Type 1 driver, a JDBC bridge is used to access ODBC drivers installed on each client machine. Using ODBC, requires configuring on your system a Data Source Name (DSN) that represents the target database.

When Java first came out, this was a useful driver because most databases only supported ODBC access but now this type of driver is recommended only for experimental use or when no other alternative is available.



T

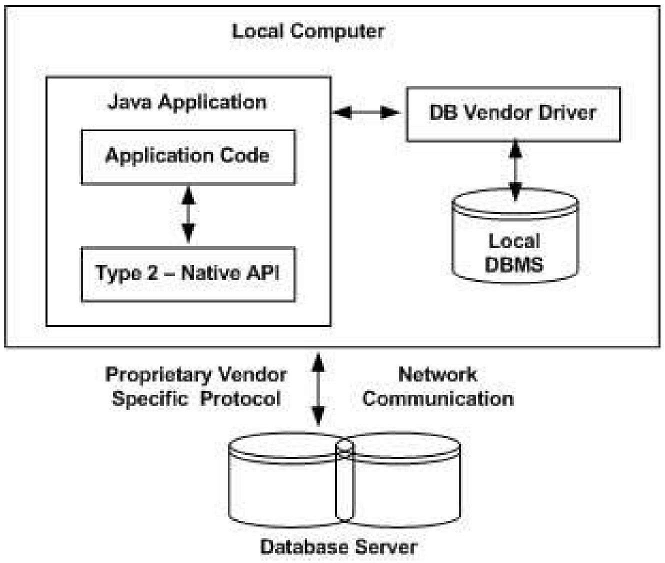
he JDBC-ODBC Bridge that comes with JDK 1.2 is a good example of this kind of driver.

Type 2: JDBC-Native API

In a Type 2 driver, JDBC API calls are converted into native C/C++ API calls, which are unique to the database. These drivers are typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge. The vendor-specific driver must be installed on each client machine.

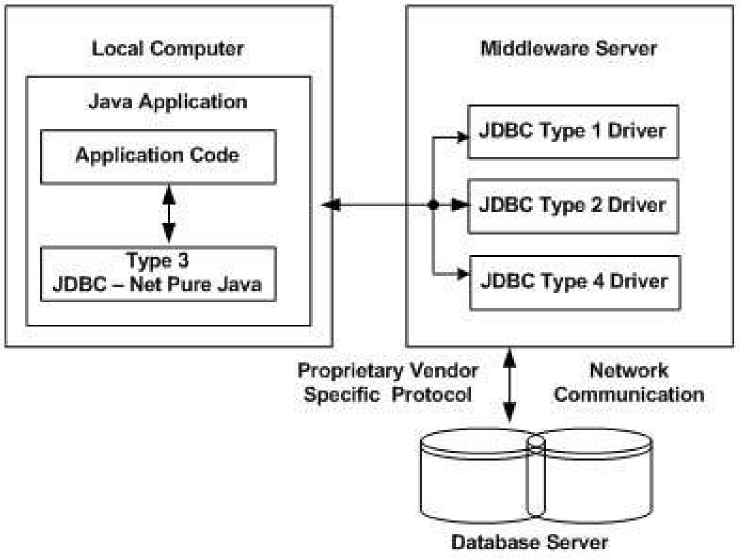
If we change the Database, we have to change the native API, as it is specific to a database and they are mostly obsolete now, but you may realize some speed increase with a Type 2 driver, because it eliminates ODBC's overhead.

The Oracle Call Interface (OCI) driver is an example of a Type 2 driver.



Type 3: JDBC-Net pure Java

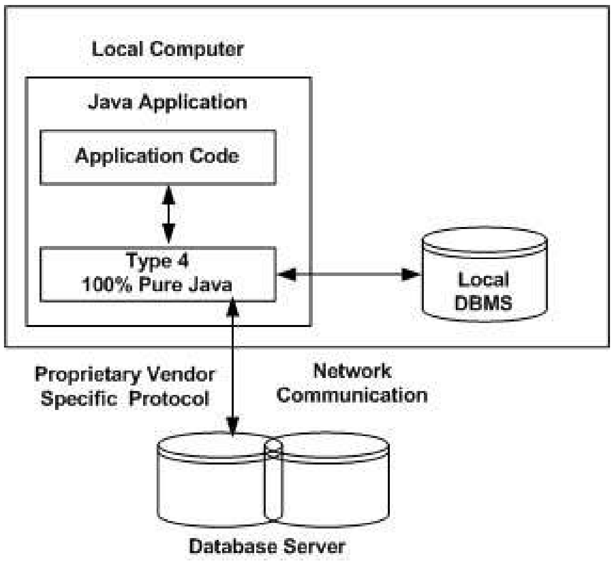
In a Type 3 driver, a three-tier approach is used to access databases. The JDBC clients use standard network sockets to communicate with a middleware application server. The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.
This kind of driver is extremely flexible, since it requires no code installed on the client and a single driver can actually provide access to multiple databases.



You can think of the application server as a JDBC "proxy," meaning that it makes calls for the client application. As a result, you need some knowledge of the application server's configuration in order to effectively use this driver type.
Your application server might use a Type 1, 2, or 4 driver to communicate with the database, understanding the nuances will prove helpful.

Type 4: 100% Pure Java
In a Type 4 driver, a pure Java-based driver communicates directly with the vendor's database through socket connection. This is the highest performance driver available for the database and is usually provided by the vendor itself.

This kind of driver is extremely flexible, you don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.



MySQL's Connector/J driver is a Type 4 driver. Because of the proprietary nature of their network protocols, database vendors usually supply type 4 drivers.

Which Driver should be Used?

- If you are accessing one type of database, such as Oracle, Sybase, or IBM, the preferred driver type is 4.
- If your Java application is accessing multiple types of databases at the same time, type 3 is the preferred driver.
- Type 2 drivers are useful in situations, where a type 3 or type 4 driver is not available yet for your database.
- The type 1 driver is not considered a deployment-level driver, and is typically used for development and testing purposes only.

Steps for Database Connectivity

1. Register the Database Driver
2. Create the Connection
3. Create Statement/PreparedStatement/CallableStatement
4. execute/executeQuery/executeUpdate

I. Register the Database Driver

Approach I - Class.forName()

```
try {
  Class.forName("com.mysql.jdbc.Driver ");
}
catch(ClassNotFoundException ex) {
  System.out.println("Error: unable to load driver class!");
}
```

Approach II - DriverManager.registerDriver()

```
try {
  Driver myDriver = new com.mysql.jdbc.Driver();
  DriverManager.registerDriver( myDriver );
}
catch(ClassNotFoundException ex) {
  System.out.println("Error: unable to load driver class!");
}
```

II. <u>Create the Connection</u>

<u>Option 1</u>: getConnection(String url, String user, String password)

```
String URL = "jdbc:oracle:thin:@amrood:1521:EMP";
String USER = "username";
String PASS = "password"
Connection conn = DriverManager.getConnection(URL, USER, PASS);
```

<u>Option 2</u>: getConnection(String url)

```
String URL = "jdbc:oracle:thin:username/password@amrood:1521:EMP";
Connection conn = DriverManager.getConnection(URL);
```

<u>Option 3</u>: getConnection(String url, Properties prop)

```
import java.util.*;

String URL = "jdbc:oracle:thin:@amrood:1521:EMP";
Properties info = new Properties( );
info.put( "user", "username" );
info.put( "password", "password" );
Connection conn = DriverManager.getConnection(URL, info);
```

<u>Closing JDBC Connections</u>

conn.close()

RDBMS	JDBC Driver Name	URL Format
MySQL	com.mysql.cj.jdbc.Driver	jdbc:mysql://hostname/ databaseName
Oracle	oracle.jdbc.driver.OracleDriver	jdbc:oracle:thin:@hostname:port Number:databaseName
DB2	com.ibm.db2.jdbc.net.DB2Driver	jdbc:db2:hostname:port Number/databaseName
Sybase	com.sybase.jdbc.SybDriver	jdbc:sybase:Tds:hostname: port Number/databaseName

III. <u>Create Statement/PreparedStatement/CallableStatement</u>
Once a connection is obtained we can interact with the database. The JDBC Statement, CallableStatement, and PreparedStatement interfaces define the methods and properties that enable you to send SQL or PL/SQL commands and receive data from your database.

They also define methods that help bridge data type differences between Java and SQL data types used in a database.

Interfaces	Recommended Use
Statement	Use the for general-purpose access to your database. Useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters.
PreparedStatement	Use the when you plan to use the SQL statements many times. The PreparedStatement interface accepts input parameters at runtime.
CallableStatement	Use the when you want to access the database stored procedures. The CallableStatement interface can also accept runtime input parameters.

Statement

```
try (Statement stmt = conn.createStatement( )) {
          String SQL = "Update Employees SET age = 35 WHERE id = 101";
           int x = stmt.executeUpdate(SQL);
}
catch (SQLException e) {
  . . .
}
```

Once you've created a Statement object, you can then use it to execute an SQL statement with one of its three execute methods.

boolean execute (String SQL): Returns a boolean value of true if a ResultSet object can be retrieved; otherwise, it returns false. Use this method to execute SQL DDL statements or when you need to use truly dynamic SQL.

int executeUpdate (String SQL): Returns the number of rows affected by the execution of the SQL statement. Use this method to execute SQL statements for which you expect to get a number of rows affected - for example, an INSERT, UPDATE, or DELETE statement.

ResultSet executeQuery (String SQL): Returns a ResultSet object. Use this method when you expect to get a result set, as you would with a SELECT statement.

JM1-SQLSyntax: Pre-Requisites

1. Restore the database - JDBC Examples – Eclipse\Database.sql
2. Open within Eclipse Project
3. To set CLASSPATH with Eclipse
   - Project | Properties | Java Build Path | Libraries | External Jar Files
   - Select the MySqlConnector.jar from C:\Program Files (x86)\MySQL\Connector J 8.0
4. Within the Program change the password

Example: JM1-SQLSyntax\FirstExample.java

PreparedStatement

```
String SQL = "Update Employees SET age = ? WHERE id = ?";
try (PreparedStatement pstmt =   pstmt = conn.prepareStatement(SQL)) {
  pstmt.setInt(1,35);
  pstmt.setInt(2,101);
 int x = pstmt.executeUpdate();
}
catch (SQLException e) {
  . . .
}
```

Example: JM1-SQLSyntax\PSExample.java

CallableStatement

```
String SQL = "{call getEmpName (?, ?)}";
try (CallableStatement cstmt = conn.prepareCall (SQL) ) {
  . . .
}
catch (SQLException e) {
  . . .
}
```

Example: JM1-SQLSyntax \CSExample.java

Sample MySQL Procedure

```
CREATE PROCEDURE `emp`.`getEmpName`
   (IN EMP_ID INT, OUT EMP_FIRST VARCHAR(255))
BEGIN
   SELECT first INTO EMP_FIRST
   FROM Employees
   WHERE ID = EMP_ID;
END
```

Question 1: Managing Employee Details using JDBC

Problem Statement:

Write a **menu-driven Java program using JDBC** to perform the following operations on an Employee table in a relational database:

1. Insert new employee details
2. Retrieve and display all employee details
3. Update the salary of an existing employee based on empId
4. Delete an employee record based on empId
5. Exit the program

The program should handle SQL exceptions using try/catch/finally blocks and maintain a proper database connection lifecycle.

Table Schema (MySQL):

```
CREATE TABLE Employee (
   empId INT PRIMARY KEY,
   empName VARCHAR(100),
   salary DOUBLE
);
```

Program Requirements:

- Display a menu with the options mentioned above.
- Prompt the user to choose an option and perform the corresponding action.
- Use **PreparedStatement** for insert, update, and delete operations.
- Display employee records in a user-friendly format.
- After each operation, return to the main menu until the user chooses to exit.
- Display meaningful messages like:
  - "Database connected successfully."
  - "Employee inserted/updated/deleted successfully."
  - "Employee not found." (when applicable)
  - "Connection closed successfully."

Sample Menu:

```
Menu:
1. Insert Employee
```

2. Display All Employees

3. Update Employee Salary

4. Delete Employee

5. Exit

Input Format:

Each menu option will prompt for appropriate input:

- For insert:
    - Number of employees to insert (int)
    - For each employee:
        - Employee ID (int)
        - Name (String)
        - Salary (double)
- For update:
    - Employee ID (int)
    - New Salary (double)
- For delete:
    - Employee ID (int)

Output Format:

- Display inserted/updated/deleted employee details.
- Show total number of rows affected (if applicable).
- Show all employees in tabular form for display option.

Question 2: Student Academic Portal using JDBC

Problem Statement:

Create a JDBC-based menu-driven program to manage student academic records.

Menu:

1. Add Student
2. Display All Students
3. Update Major of a Student
4. Display Students with GPA above a given threshold
5. Delete a Student by ID
6. Exit

Table Schema:

```
CREATE TABLE Student (
   studentId INT PRIMARY KEY,
   name VARCHAR(100),
   major VARCHAR(100),
   gpa DOUBLE
);
```

Input Format:

- For insert:
    - Student ID (int)
    - Name (String)
    - Major (String)
    - GPA (double)
- For update: studentId, new major
- For filter: GPA threshold (double)

Question 3: Product Inventory with Category Filter

Problem Statement:

Develop a JDBC menu-driven program to manage product inventory with category-based search.

Menu:

1. Insert Product
2. Display Products by Category
3. Update Price of a Product
4. Delete Product
5. Exit

Table Schema:

```
CREATE TABLE Product (
   productId INT PRIMARY KEY,
   name VARCHAR(100),
   category VARCHAR(50),
   price DOUBLE,
   quantity INT
);
```

Input Format:

- For insert: ID, name, category, price, quantity
- For update: productId, new price
- For display: category name (String)

Question 4: Online Course Platform using JDBC

Problem Statement:

Write a JDBC menu-driven program to manage online courses.

Menu:

1. Add New Course
2. Display All Courses
3. Display Courses with Duration Less Than X Hours
4. Update Fee of a Course
5. Exit

Table Schema:

```
CREATE TABLE Course (
   courseId INT PRIMARY KEY,
   courseName VARCHAR(100),
   instructor VARCHAR(100),
   duration INT, -- in hours
   fee DOUBLE
);
```

Input Format:

- For insert: ID, name, instructor, duration, fee
- For filter: maxDuration (int)
- For update: courseId, new fee

Question 5: Vehicle Management System using JDBC

Problem Statement:

Build a JDBC application to manage vehicle data for a dealership.

Menu:

1. Add Vehicle
2. Show All Vehicles
3. Search Vehicles by Brand
4. Update Model Year
5. Delete Vehicle by Registration Number
6. Exit

Table Schema:

```
CREATE TABLE Vehicle (
   regNumber VARCHAR(15) PRIMARY KEY,
   brand VARCHAR(50),
   model VARCHAR(50),
   modelYear INT,
   price DOUBLE
);
```

Input Format:

- For insert: regNumber, brand, model, year, price
- For update: regNumber, new modelYear
- For search: brand name (String)

Question 6: Employee HR System using JDBC

Problem Statement:

Create a JDBC-based menu-driven Java program to manage employee records that includes department-based filtering and salary updates.

Menu Options:

1. Add New Employee
2. Display All Employees

3. Display Employees by Department
4. Update Employee Salary
5. Delete Employee by ID
6. Exit

Table Schema:

```
CREATE TABLE Employee (
  empId INT PRIMARY KEY,
  empName VARCHAR(100),
  department VARCHAR(50),
  salary DOUBLE
);
```

Input Format:

- **For Insert:**
  - Employee ID (int)
  - Name (String)
  - Department (String)
  - Salary (double)
- **For Display by Department:**
  - Department name (String)
- **For Update Salary:**
  - Employee ID (int)
  - New Salary (double)
- **For Delete:**
  - Employee ID (int)

Output Format:

- Messages such as:
  - "Database connected successfully."
  - "Employee inserted successfully."
  - "Salary updated successfully."
  - "Employee deleted successfully."

    - "Employee not found."
    - "Connection closed successfully."
- Display all employees or filtered employees in tabular format with headers: ID, Name, Department, Salary.