JDBC

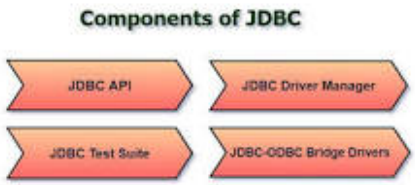
JDBC **Drivers**

Type	Name	Architecture	Advantages	Disadvantages
1	JDBC-ODBC Bridge Driver	JDBC calls to ODBC calls	Easy to use, broad database support	Requires ODBC driver, slower, platform-dependent
2	Native-API Driver	JDBC calls to native API calls	Better performance than Type 1	Requires native libraries, not portable
3	Network Protocol Driver	JDBC calls to middleware server	No client-side native libraries, multi-database support	Additional network trip, requires middleware
4	Thin Driver	Pure Java driver, direct protocol	Platform-independent, best performance	Database-specific drivers required

Choosing the Right Driver

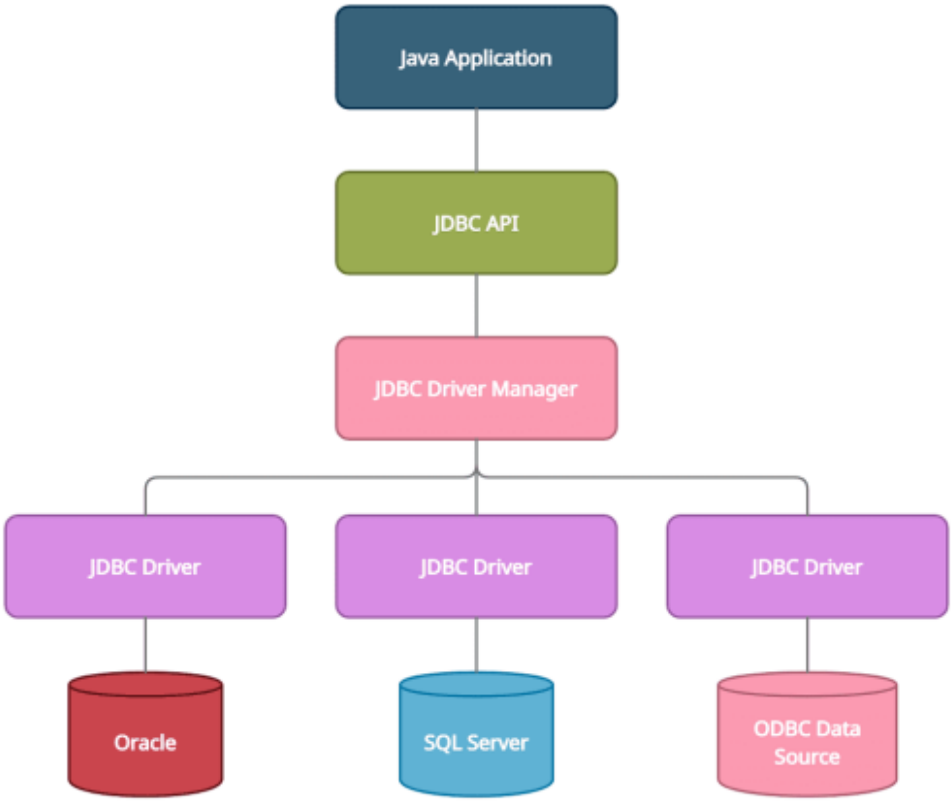
type-2 driver is vendor specific driver, that is it will only work for the specific database as the installed libraries on the client side .

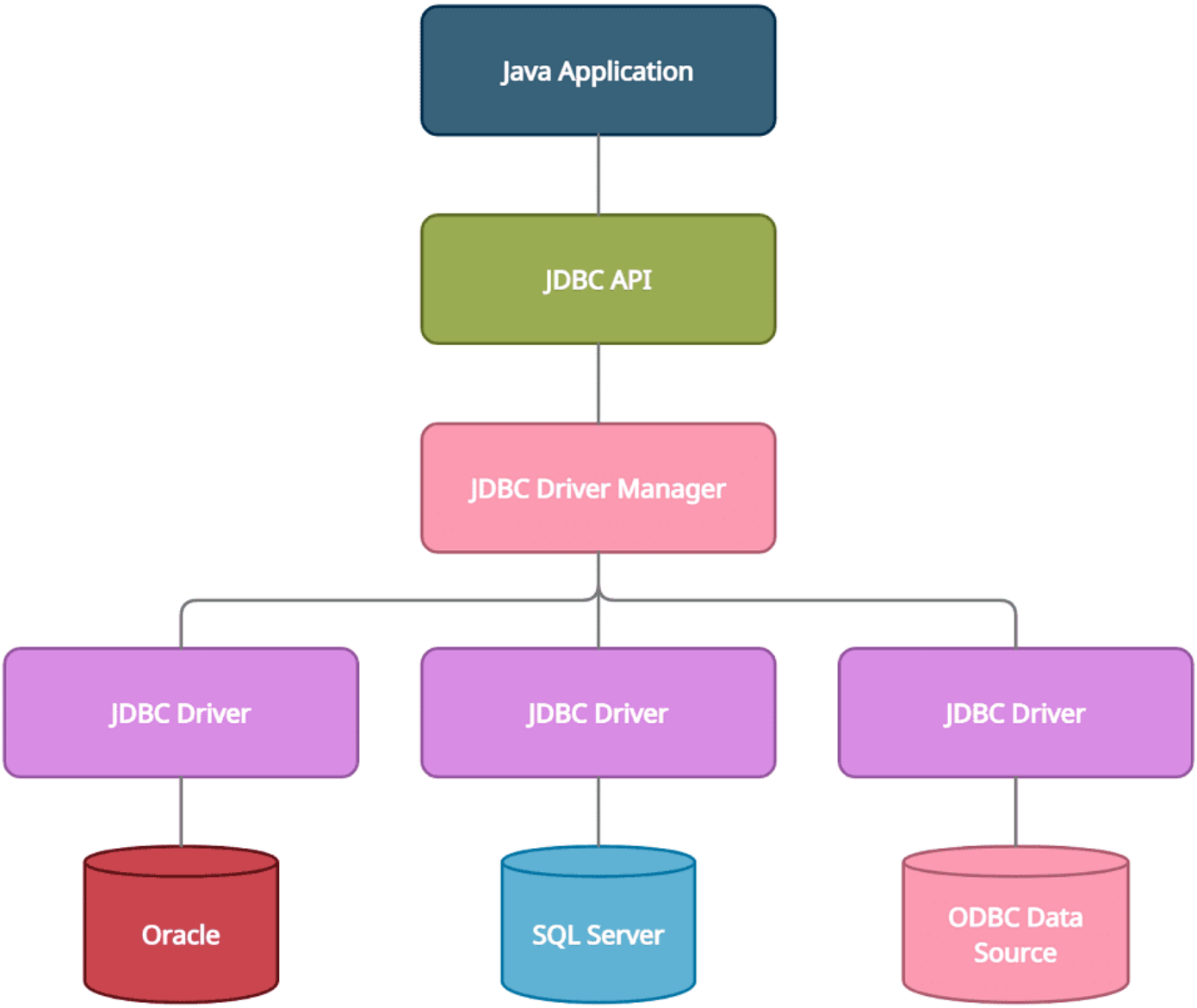
JDBC Components



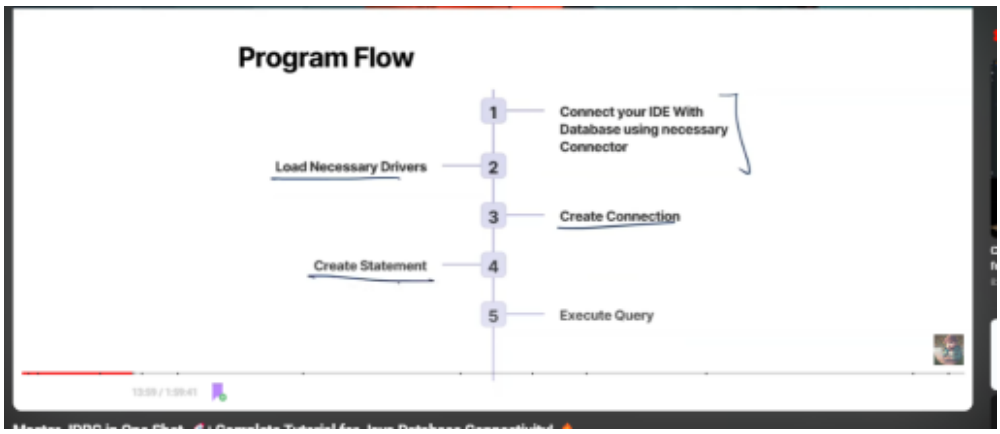
- **JDBC API**. - contains driverManager class, connection , resultSet , statement and preparedStatement interfaces WORA(write once run anywhere) ***import java.sql and javax.sql***
- **JDBC Driver Manager**. It loads a database-specific driver in an application to establish a connection with a database. It is used to make a database-specific call to the database to process the user request.
- **JDBC test suite** It is used to test the operation(such as insertion, deletion, updation) being performed by JDBC Drivers.

JDBC-ODBC Bridge Drivers. It connects database drivers to the database. This bridge translates the JDBC method call to the ODBC function call. It makes use of the **sun.jdbc.odbc** package which includes a native library to access ODBC.





Program Flow

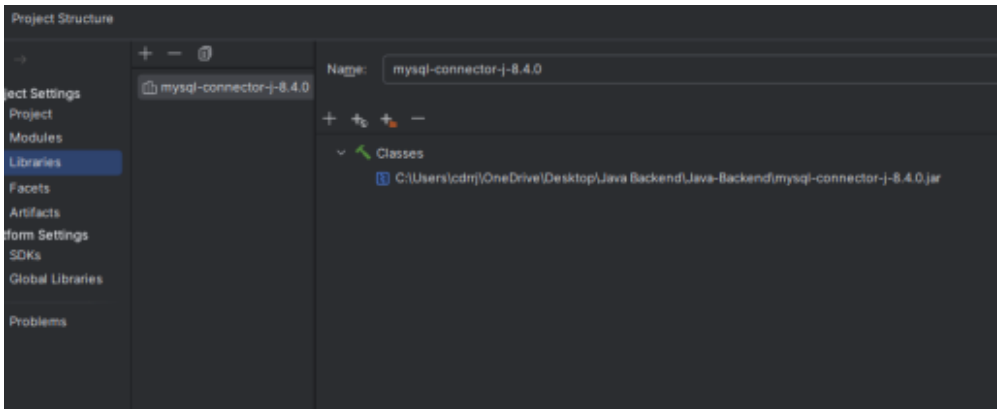


Setup and Start JDBC

1) setup the database that you are using here (mysql)
download mysql connector j , mysql workbench , mysql server
add mysql connector to project libraries , and setup workbench
Setup Workbench and How to get Username and pass

Download all necessary things to start workbench
Setup MySql and open workbench
it shoudl shoud a local instance MySql
then copy JDBC Connection String to Clipboard
p s f string username = "jdbc:mysql://localhost:3306/localhost"
replace this local host with the name of database u wrkn
user = root

2) Include JDBC Driver's in your project



You need the JDBC driver for the database you want to connect to. This driver is usually provided by the database vendor. For example:

- **MySQL**: You need `mysql-connector-java.jar`.
- **PostgreSQL**: You need `postgresql.jar`.

You can add the driver to your project by downloading the JAR file and adding it to your project's classpath. If you are using Maven, you can include it as a dependency in your `pom.xml`.

```
<dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
            <version>8.0.26</version>
</dependency>
```

or add in project structure > library > +

3) Steps for JDBC Program

1) load drivers
   import java.sql.*;
   Class.forName("")
2) Establish Connection
   // Here in the url you need to mention db at the end

p s f String url , p s f password , p s f username

Connection con = DriverManager.getConnectino (url,username,pass);

Statement = con.prepareStatement();

3) On MySQL command line

Here we have to create a DB and then create a table inside that db , containing some information (row and cols).

```
CREATE DATABASE IF NOT EXISTS dbforjdbc;
use dbforjdbc;
CREATE TABLE student( id INT AUTO_INCREMENT PRIMARY KEY,
name VARCHAR(255) NOT NULL , age INT NOT NULL , marks DOUBLE
NOT NULL ) ;
INSERT INTO students(name , age , marks ) VALUES ( "nd" , 20, 94.6);
```

4) Prepare Statement

```
String query = "SELECT * FROM mytable WHERE name = ?";
PreparedStatement pstmt=con.prepareStatement(query);
```

Setting Parameters

Use the appropriate setter methods to set the values for the placeholders (?).

```
java
pstmt.setString(1, "John Doe");
Common setter methods:
setString(int parameterIndex, String value)
setInt(int parameterIndex, int value)
setDouble(int parameterIndex, double value)
setDate(int parameterIndex, java.sql.Date value)
// Prepare S. helps / prevents from sql injection
```

Execution in Batches

Format Specifier

%%	Inserts a % sign
%x %X	Integer hexadecimal
%t %T	Time and Date
%s %S	String
%n	Inserts a newline character
%o	Octal integer
%f	Decimal floating-point
%e %E	Scientific notation
%g	Causes Formatter to use either %f or %e, whichever is shorter
%h %H	Hash code of the argument
%d	Decimal integer
%c	Character
%b %B	Boolean
%a %A	Floating-point hexadecimal

This helps in writing query statements
String.format("INSERT INTO students(name,age,marks) VALUES('%s' ,%o , %f)" , "Rahul" , 21 , 94,6)
Remeber : for string use '%s'
For running multiple sql commands with a **single statement/query we use batchProcessing**

```
Statement statement;
while(){
     String query;
     statement.addBatch(query)
}
statemnt.executeBatch();
```

Transaction Handling in JDBC

Key Concepts

- **Auto-commit Mode**: By default, each SQL statement is committed automatically.
- **Manual Commit Mode**: Disable auto-commit to manage transactions manually. Explicitly commit or rollback transactions.

Steps for Transaction Handling

1. **Disable Auto-commit**:

   ```
   conn.setAutoCommit(false);
   ```

2. **Perform SQL Operations**:
   - Prepare and execute SQL statements within the transaction.
3. **Commit or Rollback**:
   - **Commit** if all operations succeed: (to be used under / in try block)

     ```
     conn.commit();
     ```

   - **Rollback** if any operation fails: (to be used under exception catch block)

     ```
     conn.rollback();
     ```

4. **Handle Exceptions**:
   - Use try-catch blocks to manage exceptions and ensure proper rollback in case of errors.
5. **Close Resources**:
   - Always close `Connection`, `PreparedStatement`, and `ResultSet` objects to free resources.

```
pstmt.close();      conn.close();
```

JDBC Project on Bus Agency