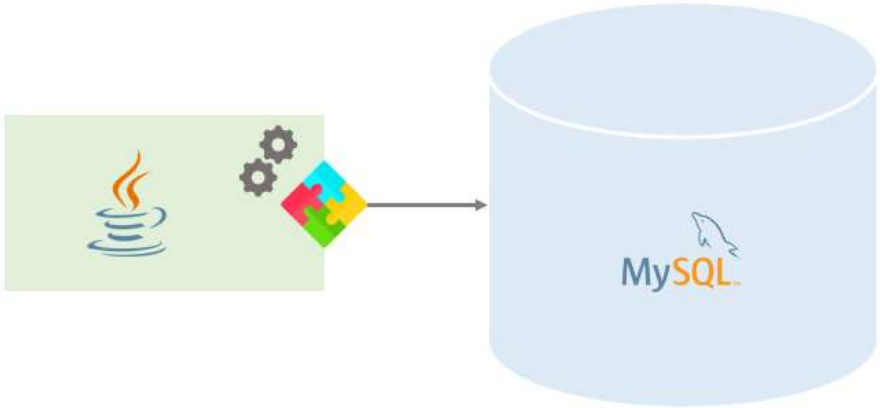
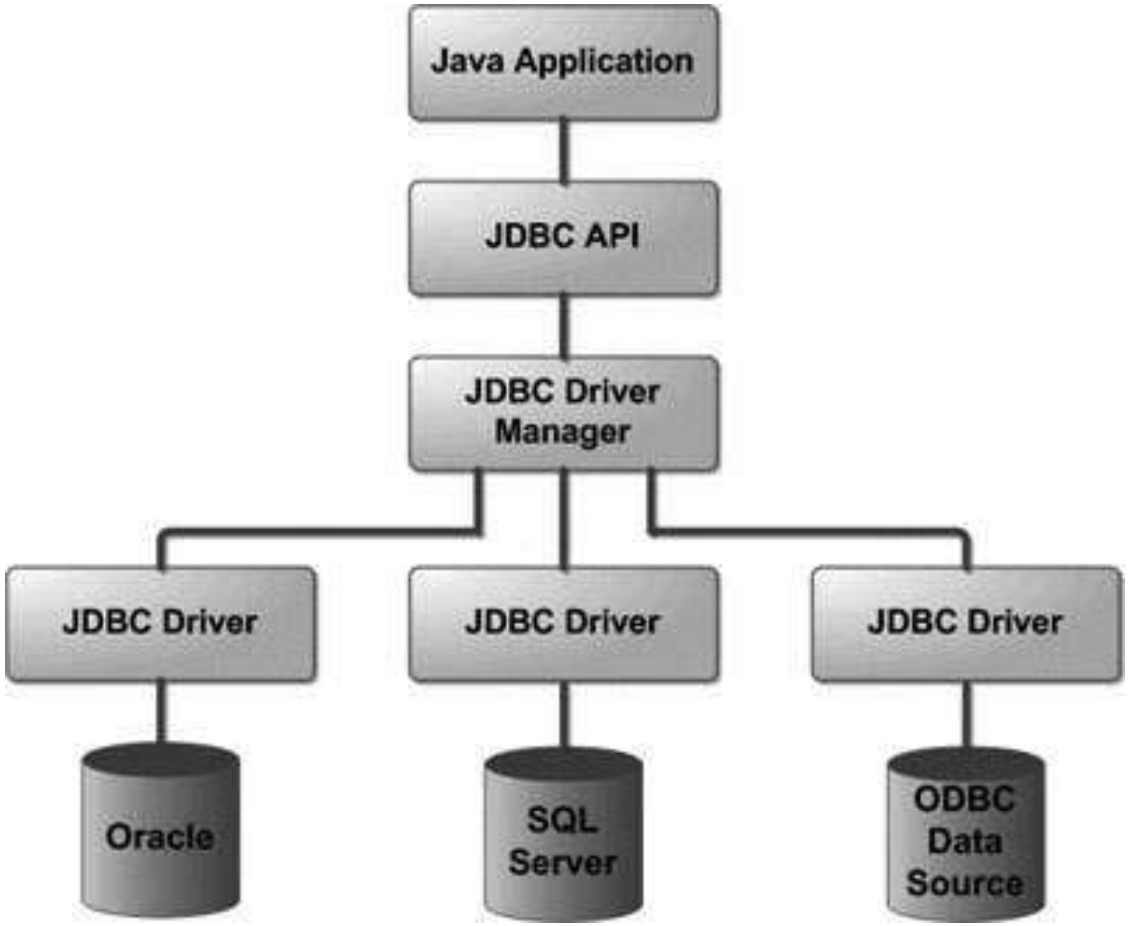
# Java Database Connectivity

# JDBC

- Java JDBC API provides a standard interface to interact with any relational databases.
- JDBC API consists of the following main components:
  1. JDBC Driver
  2. Connection
  3. Statement
  4. ResultSet




Java Application
JDBC API
JDBC Driver Manager
JDBC Driver
JDBC Driver
JDBC Driver
Oracle
SQL Server
ODBC Data Source

# JDBC Driver

- A JDBC driver is set of Java classes that implement JDBC interfaces for interacting with a specific database. Almost all database vendors such as MySQL, Oracle, Microsoft SQL Server, provide JDBC drivers. For example, MySQL provides a JDBC driver called MySQL Connection/J that allows you to work with MySQL database through a standard JDBC API.
- There are four types of JDBC drivers including JDBC-native API Driver, JDBC-net Driver, and JDBC Driver.
- For details refer: JDBC Driver

# Database Connection Steps

- Import package
- Load and Register Driver
- Establish connection
- Create statement
- Execute query
- Process result
- Close

# Connection

- Most important component of JDBC is the Connection object. In a Java application, you first load a JDBC driver and then establish a connection to the database. Through the Connection object, you can interact with the database e.g., creating a Statement to execute SQL queries against tables.
- You can open more than one connection to a database at a time.

# Statement

- To execute a SQL query e.g., SELECT, INSERT, UPDATE, DELETE, etc., you use a Statement object.
- You create the Statement object through the Connection object.
- JDBC provides several types of statements for different purposes such as PreparedStatement , CallableStatement .

# ResultSet

- After querying data from the database, you get a ResultSet object. The ResultSet object provides a set of API that allows you to traverse result of the query.

# JDBC Flow

- The typical flow of using JDBC is as follows:
- First, load the JDBC driver and create a connection to the database.
- Then, create a Statement and execute the query to get a ResultSet.
- Next, traverse and process the ResultSet .
- Close the ResultSet , Statement , and Connection .

# Prerequisite for Driver

- To connect to MySQL database from a Java program, you need to do the following steps:

- Load the MySQL Connector/J into your program.
- Create a new Connection object from the DriverManager class. Then you can use this Connection object to execute queries.

# Loading Driver

- Download MySQL connector from https://repo1.maven.org/maven2/mysql/mysql-connector-java/8.0.27/mysql-connector-java-8.0.27.jar
- Add it to your project

# Steps

1. **Driver class:** The driver class for the mysql database is **com.mysql.cj.jdbc.Driver**.
2. **Connection URL:** The connection URL for the mysql database is **jdbc:mysql://localhost:3306/db** where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and db is the database name. **Username:** The default username for the mysql database is **root**.
3. **Password:** It is the password given by the user at the time of installing the mysql database.

# Creating Statements

- There are three different kinds of statements:
  - Statement: Used to implement simple SQL statements with no parameters.
  - PreparedStatement: (Extends Statement.) Used for precompiling SQL statements that might contain input parameters.
  - CallableStatement: (Extends PreparedStatement.) Used to execute stored procedures that may contain both input and output parameters.

# Executing Queries

- To execute a query, call an execute method from Statement such as the following:
  - execute: Returns true if the first object that the query returns is a ResultSet object. Use this method if the query could return one or more ResultSet objects. Retrieve the ResultSet objects returned from the query by repeatedly calling Statement.getResultSet.
  - executeQuery: Returns one ResultSet object.
  - executeUpdate: Returns an integer representing the number of rows affected by the SQL statement. Use this method if you are using INSERT, DELETE, or UPDATE SQL statements.