Agenda

- JDBC

Java Database Connectivity (JDBC)

- RDBMS understand SQL language only.
- JDBC driver converts Java requests in database understandable form and database response in Java understandable form.
- JDBC drivers are of 4 types
    - Type I - Jdbc Odbc Bridge driver
        - ODBC is standard of connecting to RDBMS (by Microsoft).
        - Needs to create a DSN (data source name) from the control panel.
        - From Java application JDBC Type I driver can communicate with that ODBC driver (DSN).
        - The driver class: sun.jdbc.odbc.JdbcOdbcDriver -- built-in in Java.
        - database url: jdbc:odbc:dsn
        - Advantages:
            - Can be easily connected to any database.
        - Disadvantages:
            - Slower execution (Multiple layers).
            - The ODBC driver needs to be installed on the client machine.
    - Type II - Partial Java/Native driver
        - Partially implemented in Java and partially in C/C++. Java code calls C/C++ methods via JNI.
        - Different driver for different RDBMS. Example: Oracle OCI driver.
        - Advantages:
            - Faster execution
        - Disadvantages:
            - Partially in Java (not truely portable)
            - Different driver for Different RDBMS
    - Type III - Middleware/Network driver
        - Driver communicate with a middleware that in turn talks to RDBMS.
        - Example: WebLogic RMI Driver
        - Advantages:
            - Client coding is easier (most task done by middleware)
        - Disadvantages:
            - Maintaining middleware is costlier
            - Middleware specific to database
    - Type IV
        - Database specific driver written completely in Java.
        - Fully portable.
        - Most commonly used.
        - Example: Oracle thin driver, MySQL Connector/J, ...

MySQL Programming Steps

- step 0: Add JDBC driver into project/classpath.

    - Project Properties -> Java Build Path -> Libraries - Classpath -> Add External Jar -> select mysql driver jar -> Ok
- step 1: Load and register JDBC driver class. These drivers are auto-registered when loaded first time in JVM. This step is optional in Java SE applications from JDBC 4 spec.

```
Class.forName("com.mysql.cj.jdbc.Driver");
    // for Oracle: Use driver class oracle.jdbc.driver.OracleDriver
```

- step 2: Create JDBC connection using helper class DriverManager.

```
// db url = jdbc:dbname://db-server:port/database
Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/dbname", "root",
"manager");
    // for Oracle: jdbc:oracle:thin:@localhost:1521:sid
```

- step 3: Create the statement.

```
Statement stmt = con.createStatement();
```

- step 4: Execute the SQL query using the statement and process the result.

```
String sql = "non-select query";
int count = stmt.executeUpdate(sql); // returns number of rows affected
```

    - OR

```
String sql = "select query";
ResultSet rs = stmt.executeQuery(sql);
while(rs.next()) // fetch next row from db (return false when all rows
completed)
{
    x = rs.getInt("col1");      // get first column from the current row
    y = rs.getString("col2");   // get second column from the current row
    z = rs.getDouble("col3");   // get third column from the current row
    // process/print the result
}
rs.close();
```

- step 5: Close statement and connection.

```
stmt.close();
con.close();
```

MySQL Driver Download

- https://mvnrepository.com/artifact/com.mysql/mysql-connector-j/8.4.0

JDBC concepts

java.sql.Driver

- Implemented in JDBC drivers.
  - MySQL: com.mysql.cj.jdbc.Driver
  - Oracle: oracle.jdbc.OracleDriver
  - Postgres: org.postgresql.Driver
- Driver needs to be registered with DriverManager before use.
- When driver class is loaded, it is auto-registered (Class.forName()).
- Driver object is responsible for establishing database "Connection" with its connect() method.
- This method is called from DriverManager.getConnection().

java.sql.Statement

- Represents SQL statement/query.
- To execute the query and collect the result.

```
Statement stmt = con.createStatement();
```

```
ResultSet rs = stmt.executeQuery(selectQuery);
```

```
int count = stmt.executeUpdate(nonSelectQuery);
```

- Since query built using string concatenation, it may cause SQL injection.

java.sql.ResultSet

- ResultSet represents result of SELECT query. The result may have one/more rows and one/more columns.
- Can access only the columns fetched from database in SELECT query (projection).

```
// SELECT id, quote, created_at FROM quotes
ResultSet rs = stmt.executeQuery();
```

```
while(rs.next()) {
    int id = rs.getInt("id");
    String quote = rs.getString("quote");
    Timestamp createdAt = rs.getTimestamp("created_at"); //
java.sql.Timestamp
    // ...
}
```

```
// SELECT id, quote, created_at FROM quotes
ResultSet rs = stmt.executeQuery();
while(rs.next()) {
    int id = rs.getInt(1);
    String quote = rs.getString(2);
    Timestamp createdAt = rs.getTimestamp(3); // java.sql.Timestamp
    // ...
}
```