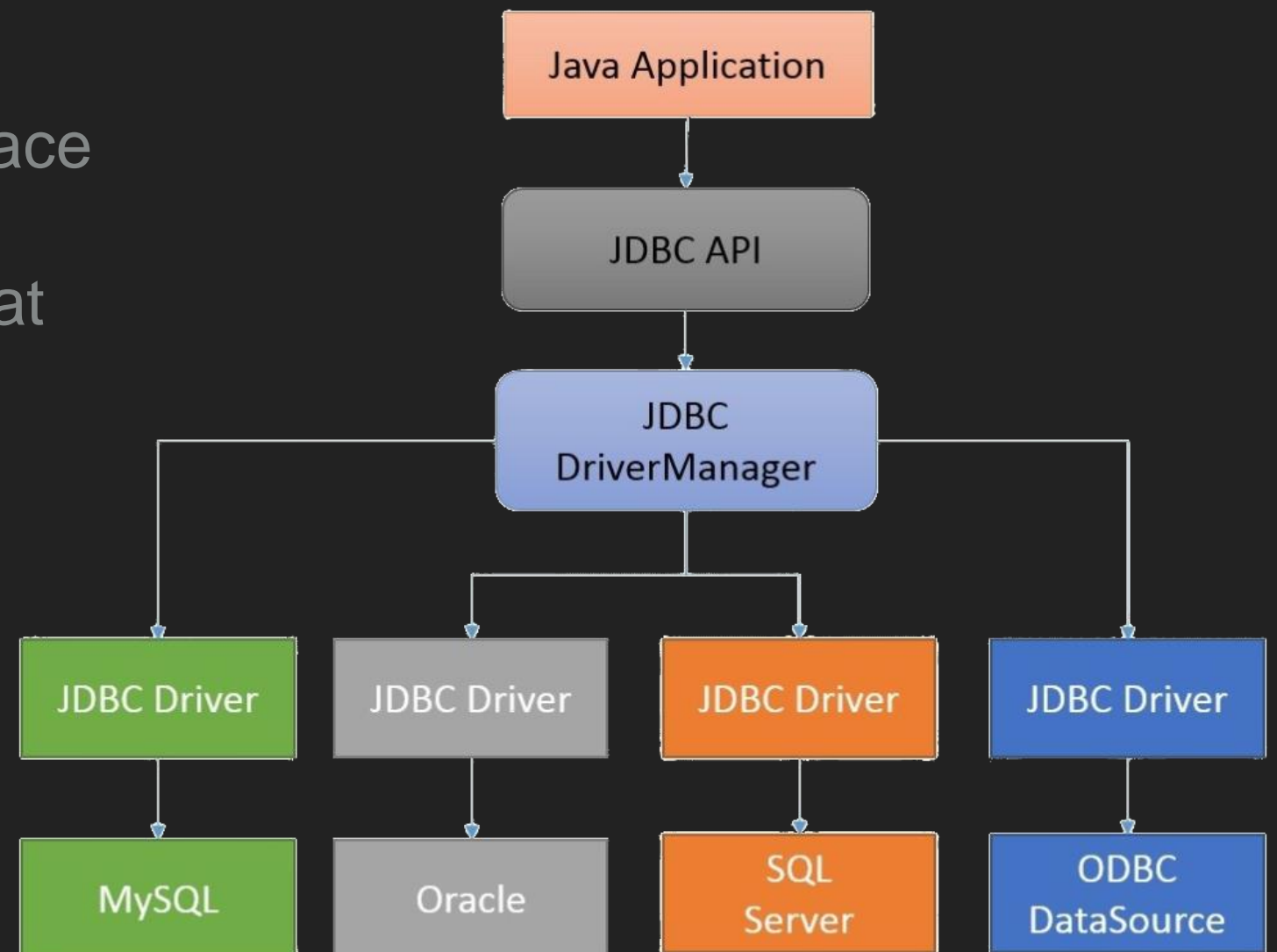

JAVA WITH DB

WHAT IS JDBC

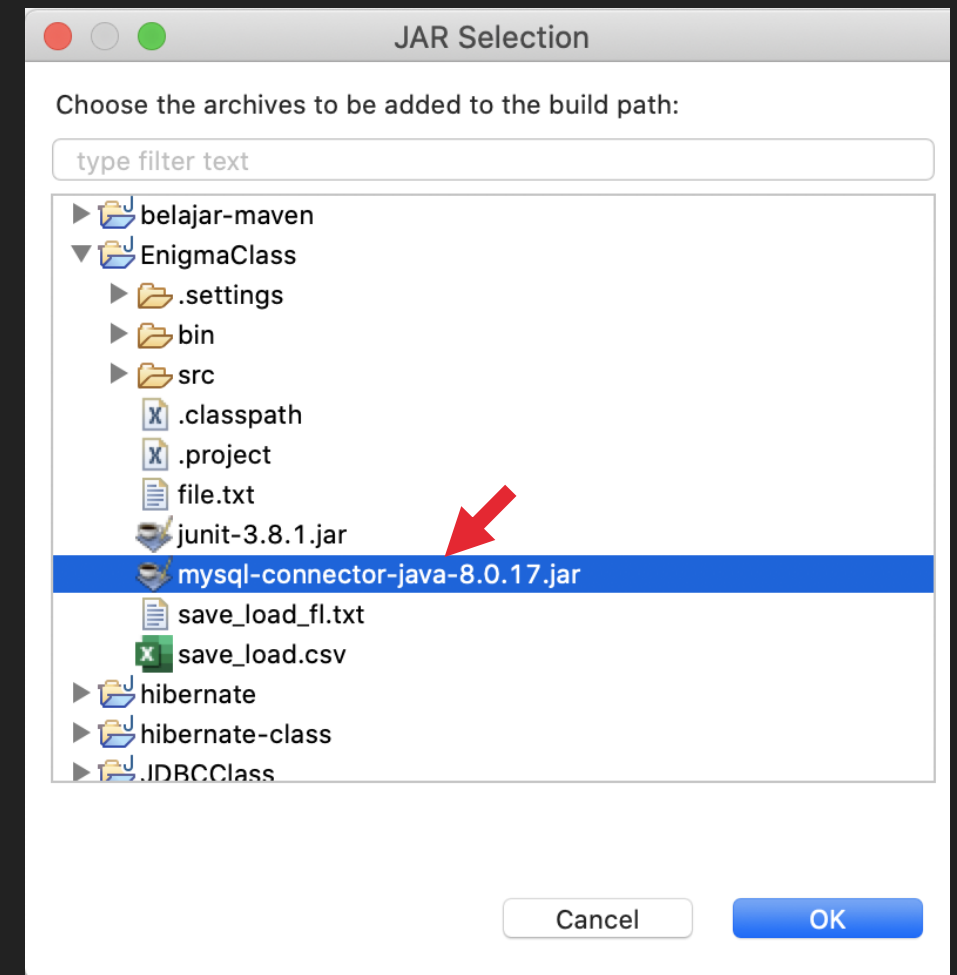
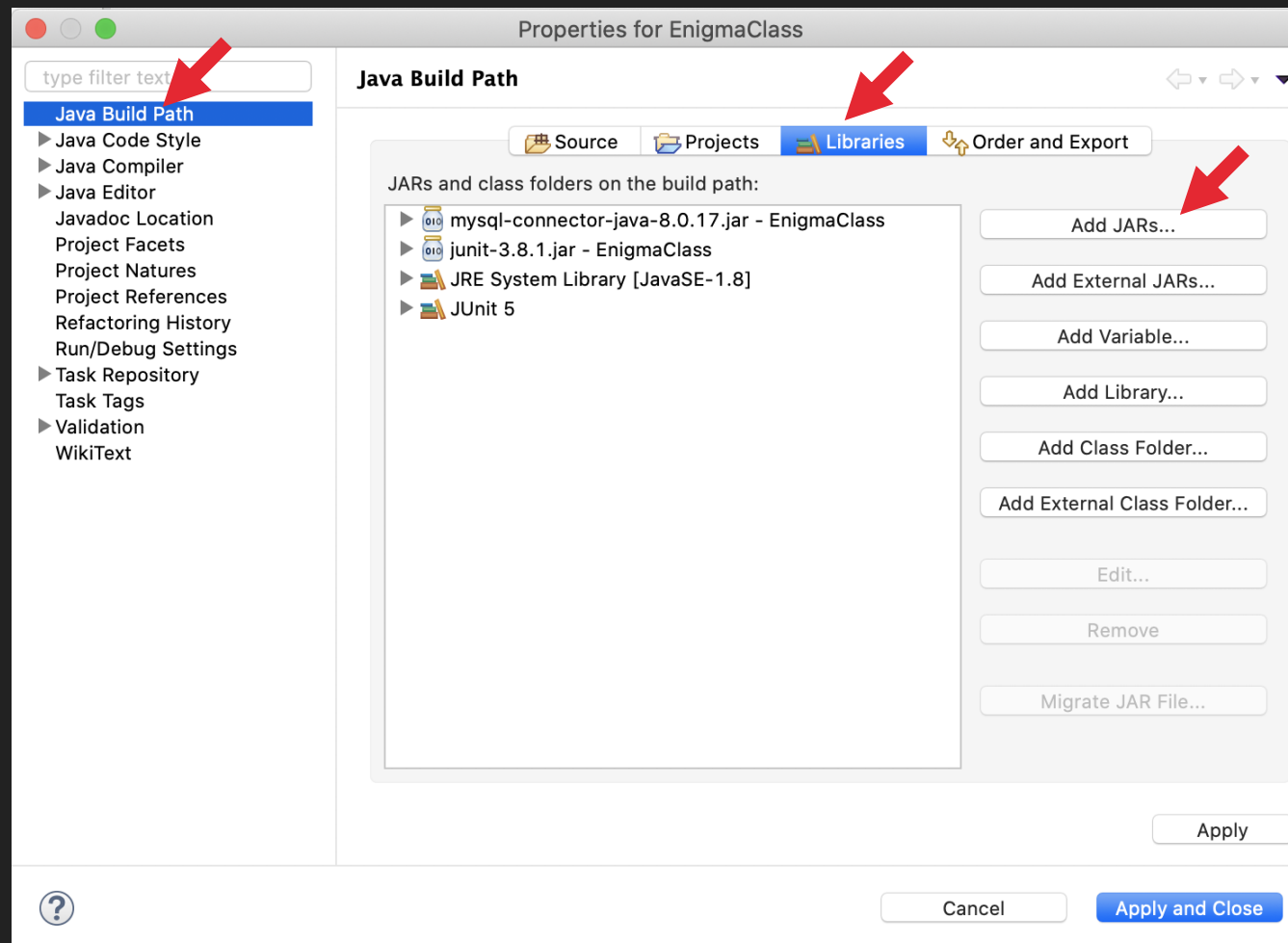
- ▶ JDBC adalah spesifikasi standard API untuk terhubung dengan Database.
- ▶ JDBC berisi sekumpulan interface standard dan masing-masing vendor database akan membuat driver yang compatible



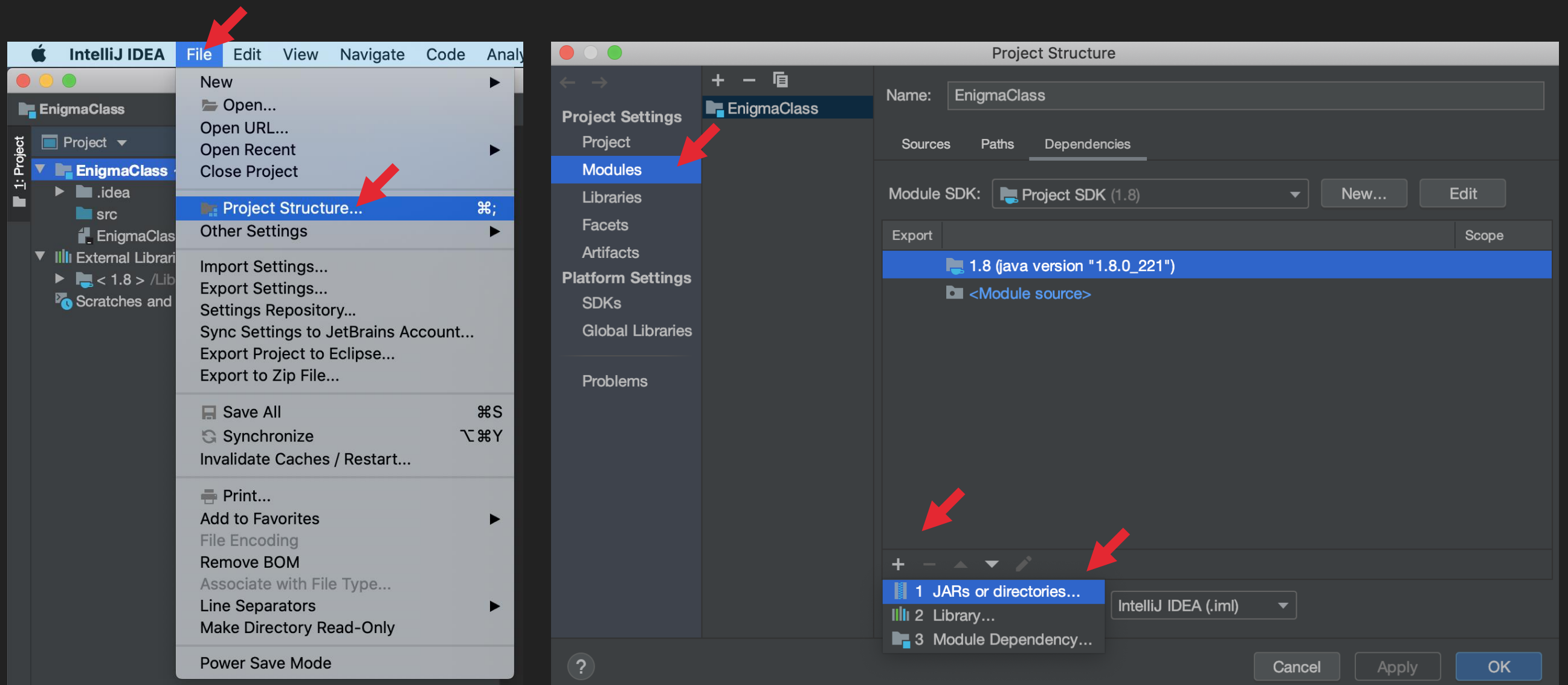
ISTILAH JDBC

- ▶ **DriverManager** : Class yang mengelola driver.
- ▶ **Driver** : Komponen software yang memungkinkan aplikasi java untuk berinteraksi dengan database.
- ▶ **Connection**: Object yang maintain sesi koneksi(session connection) dengan database
- ▶ **Statement** : Object untuk mengeksekusi query
- ▶ **ResultSet** : Object untuk menampung data hasil query

ADD DRIVER ON ECLIPSE



ADD DRIVER ON INTELLIJ IDEA



The screenshot illustrates the steps to add a driver in IntelliJ IDEA:

- Open the **File** menu and select **Project Structure...**.
- In the **Project Structure** dialog, select the **EnigmaClass** module and the **Dependencies** tab.
- Click the **+** button to add a new dependency.
- Select **1 JARs or directories...** from the dropdown menu.



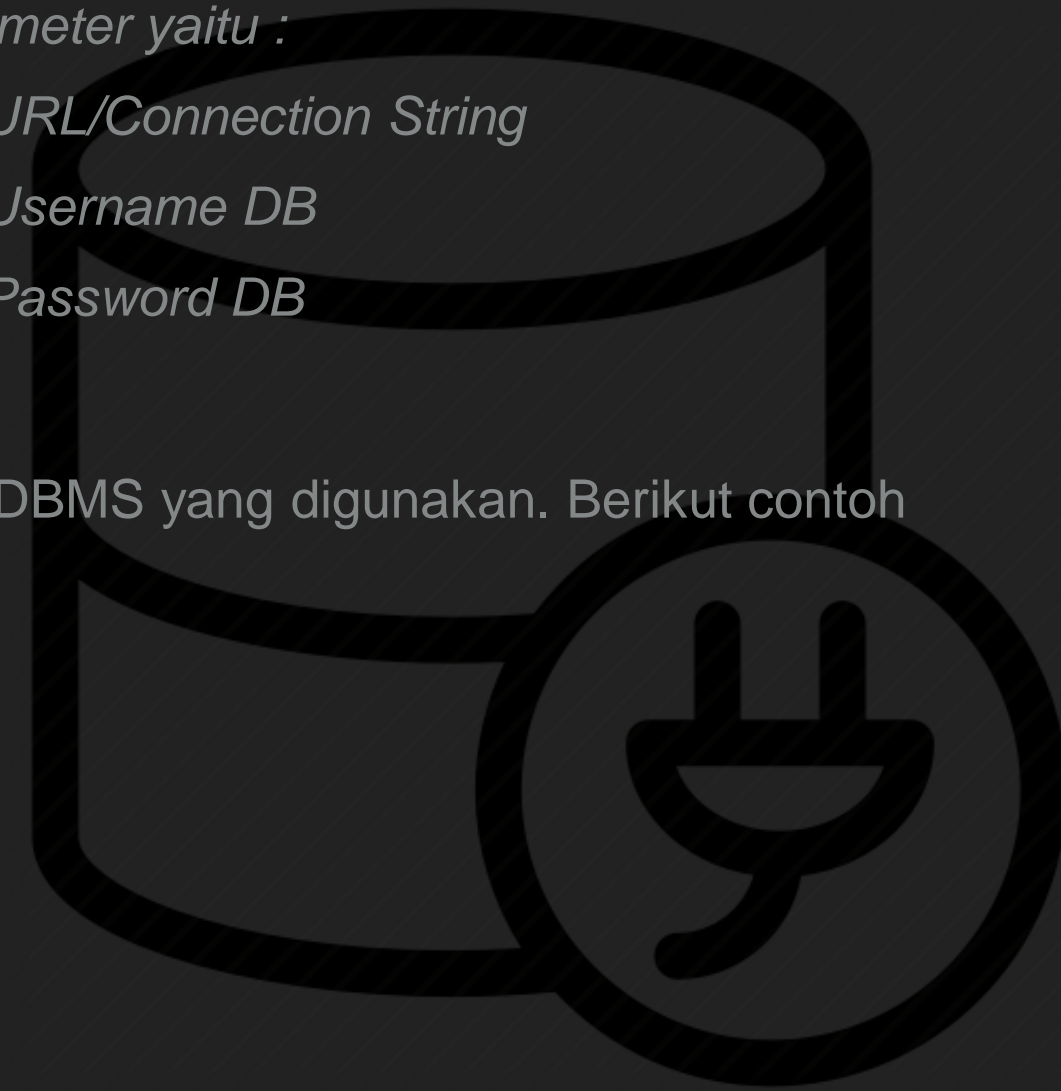
CONNECT TO DATABASE

Connection conn = null;

```
try {
    Class.forName(JDBC_DRIVER);
    conn = DriverManager.getConnection(URL, USER, PASS);
} catch (ClassNotFoundException | SQLException e) {
    e.printStackTrace();
} finally {
    if(conn!=null) {
        try {
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

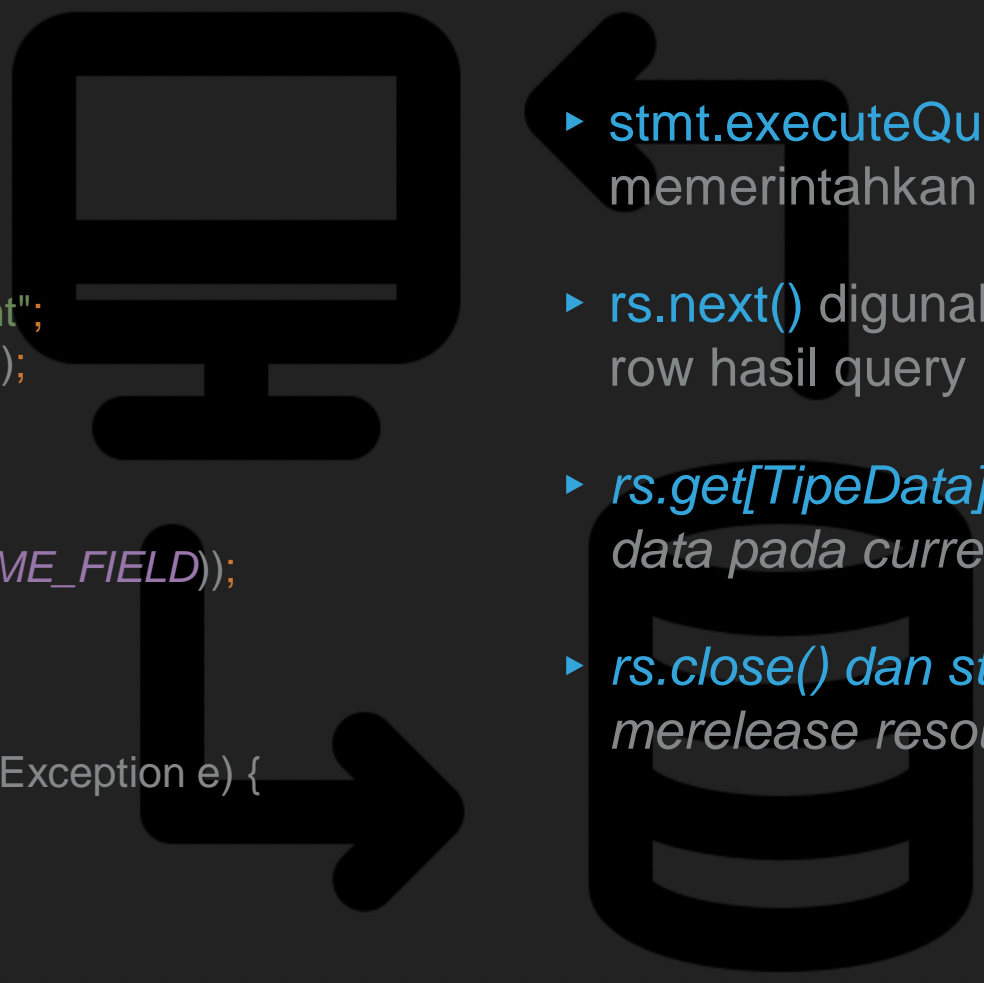
- ▶ URL/Connection String bervariasi tergantung dari driver dan RDBMS yang digunakan. Berikut contoh URL/Connection String :
 - ▶ MySQL : jdbc:mysql://[host]:[port]/[database]
 - ▶ Oracle : jdbc:oracle:thin:@[host]:[port]:[sid]
- ▶ `conn.close()` untuk menutup koneksi ke Database

- ▶ `Class.forName()` untuk mendaftarkan driver yang akan digunakan. Contoh :
 - ▶ MySQL : `com.mysql.cj.jdbc.Driver`
 - ▶ Oracle : `oracle.jdbc.driver.OracleDriver`
- ▶ `DriverManager.getConnection()` untuk memulai koneksi ke Database, menerima 3 parameter yaitu :
 - ▶ URL/Connection String
 - ▶ Username DB
 - ▶ Password DB



RETRIEVING RESULT SET

```
try {  
    ...  
  
    stmt = conn.createStatement();  
  
    String sql = "SELECT * FROM student";  
    ResultSet rs = stmt.executeQuery(sql);  
  
    while(rs.next()) {  
        System.out.println(rs.getInt(1));  
        System.out.println(rs.getString(NAME_FIELD));  
    }  
  
    rs.close();  
} catch (ClassNotFoundException | SQLException e) {  
    e.printStackTrace();  
} finally {  
    if(stmt!=null) {  
        try {  
            stmt.close();  
        } catch (SQLException e) {  
            e.printStackTrace();  
        }  
    }  
}  
  
    ...  
}
```

- 
- ▶ `stmt.executeQuery` digunakan untuk memerintahkan DB untuk menjalankan query
 - ▶ `rs.next()` digunakan untuk melakukan iterasi row hasil query
 - ▶ `rs.get[TipeData]()` digunakan untuk mengambil data pada current row
 - ▶ `rs.close()` dan `stmt.close()` digunakan untuk merelease resource setelah selesai dipakai


PREPARE STATEMENT

```
try {  
    ...  
  
    String sql = "SELECT * FROM student WHERE id = ? ";  
    stmt = conn.prepareStatement(sql);  
    stmt.setInt(1, 1);  
  
    ResultSet rs = stmt.executeQuery();  
  
    while(rs.next()) {  
        System.out.println(rs.getInt(1));  
        System.out.println(rs.getString(NAME_FIELD));  
    }  
  
    rs.close();  
} catch (ClassNotFoundException | SQLException e) {  
    e.printStackTrace();  
} finally {  
    if(stmt!=null) {  
        try {  
            stmt.close();  
        } catch (SQLException e) {  
            e.printStackTrace();  
        }  
    }  
}  
...  
}
```

- ▶ **PreparedStatement** adalah **Statement** dengan fitur tambahan
- ▶ *Kelebihan **PreparedStatement** :*
 - ▶ Memudahkan mengisi parameter
 - ▶ Bisa di-reuse untuk mengeksekusi query yang sama dengan parameter berbeda
 - ▶ Memudahkan batch query
 - ▶ Mencegah *SQLInjection*

MODIFYING DATA

```
try {  
    ...  
  
    String sql = "INSERT INTO student(id, name) VALUES (?, ?) ";  
    stmt = conn.prepareStatement(sql);  
    stmt.setInt(1, 2);  
    stmt.setString(2, "Joe");  
  
    stmt.executeUpdate();  
} catch (ClassNotFoundException | SQLException e) {  
    e.printStackTrace();  
} finally {  
    if(stmt!=null) {  
        try {  
            stmt.close();  
        } catch (SQLException e) {  
            e.printStackTrace();  
        }  
    }  
}  
...  
}
```

- 
- ▶ `executeUpdate` digunakan untuk menggantikan `executeQuery` saat menjalankan perintah untuk Insert, Update, atau Delete
 - ▶ Disarankan untuk digunakan Bersama `prepareStatement` untuk menghindari SQL Injection
 - ▶ `executeUpdate` tidak mengembalikan `ResultSet` tetapi mengembalikan jumlah row yang terdampak dari query



WORKING WITH TRANSACTION

```
try {  
    ...  
    conn.setAutoCommit(false);  
  
    String sqlInsert = "INSERT INTO student(id, name) VALUES (?, ?) ";  
    stmtInsert = conn.prepareStatement(sqlInsert);  
    stmtInsert.setInt(1, 2);  
    stmtInsert.setString(2, "Joe");  
    stmtInsert.executeUpdate();  
  
    String sqlUpdate = "UPDATE student SET name = ? WHERE id = ? ";  
    stmtUpdate = conn.prepareStatement(sqlUpdate);  
    stmtUpdate.setString(1, "Irfan");  
    stmtUpdate.setInt(2, 1);  
    stmtUpdate.executeUpdate();  
  
    conn.commit();  
} catch (ClassNotFoundException | SQLException e) {  
    e.printStackTrace();  
    if(conn!=null) {  
        try {  
            conn.rollback();  
        } catch (SQLException ex) {  
            ex.printStackTrace();  
        }  
    }  
}
```

```
} finally {  
    if(stmtInsert!=null) {  
        try {  
            stmtInsert.close();  
        } catch (SQLException e) {  
            e.printStackTrace();  
        }  
    }  
    if(stmtUpdate!=null) {  
        try {  
            stmtUpdate.close();  
        } catch (SQLException e) {  
            e.printStackTrace();  
        }  
    }  
    ...  
}
```

- ▶ `conn.setAutoCommit(false)` digunakan untuk memulai transaksi
- ▶ `conn.rollback()` untuk mengembalikan data ke transaksi dimulai dan menghilangkan semua perubahan
- ▶ `conn.commit()` untuk membuat perubahan data transaction menjadi permanen

RollbackCOMMIT