

## MODUL 4

### DATABASE STORAGE

#### A. TUJUAN

1. Mahasiswa mengenali ragam pengelolaan database dengan Java
2. Mahasiswa dapat melakukan penyimpanan database dengan Java
3. Mahasiswa dapat melakukan pembacaan database dengan Java

#### B. ALOKASI WAKTU 1 x 50 menit

#### C. DASAR TEORI

##### 1. Database

Secara umum, Database adalah kumpulan data yang terorganisir dengan baik yang disimpan secara elektronik di dalam sistem komputer. Database digunakan untuk menyimpan, mengelola, dan mengakses informasi dengan efisien. Database adalah komponen penting dalam banyak aplikasi perangkat lunak dan sistem informasi yang digunakan di berbagai bidang, termasuk bisnis, ilmu pengetahuan, pendidikan, pemerintahan, dan lainnya.

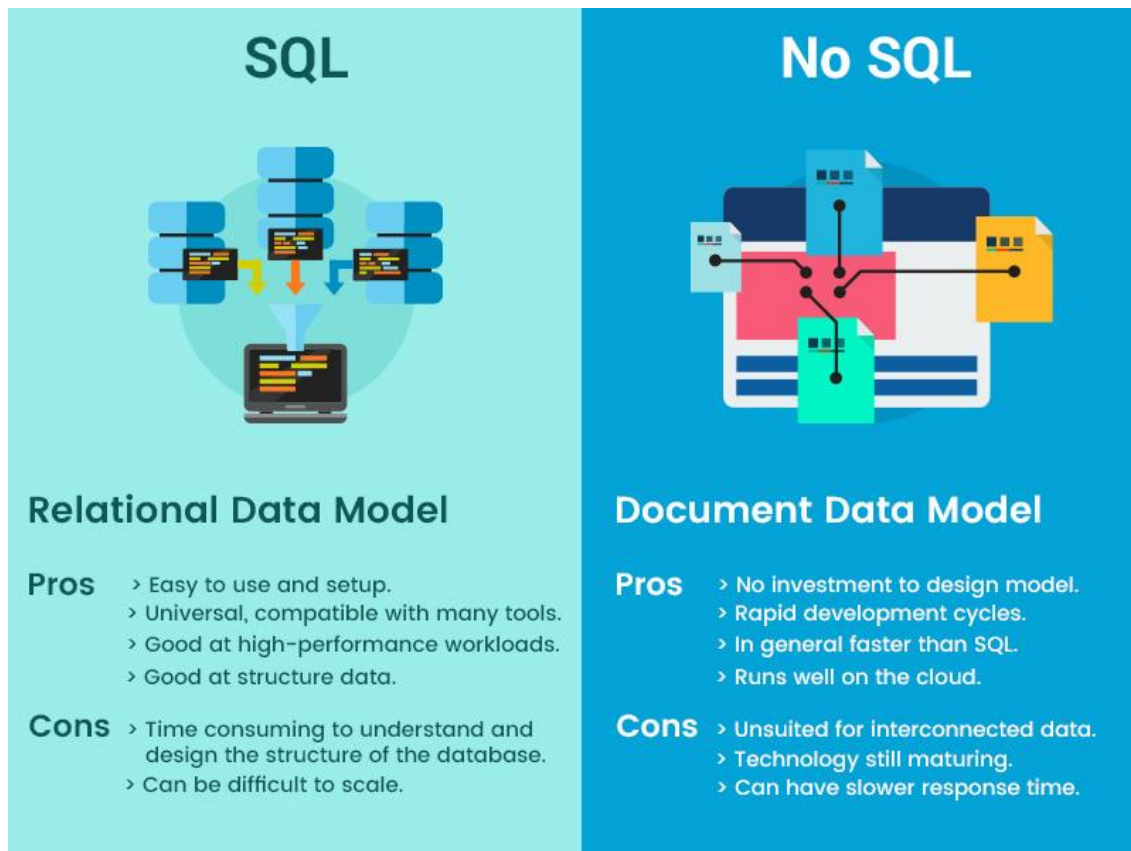
Dalam dunia pemrograman, menjadi salah satu ujung tombak dari keberhasilan pembuatan suatu sistem perangkat lunak. Tanpa adanya database, perangkat lunak yang dibangun tidak akan dapat menjadi dinamis. Artinya, data pada aplikasi tidak akan pernah bisa berubah dan menjadikannya tidak bermanfaat. Berikut adalah beberapa fungsi utama dari database dalam pemrograman:

1. **Penyimpanan Data:** Fungsi utama dari database adalah menyimpan data dengan cara yang terstruktur. Data tersebut bisa berupa informasi pelanggan, pesanan, produk, atau jenis data lainnya yang relevan dengan aplikasi.
2. **Pengaksesan Data:** Database memungkinkan aplikasi untuk mengakses data dengan cepat dan efisien. Ini dilakukan melalui perintah-perintah kueri yang digunakan untuk mengambil, menyaring, dan mengurutkan data sesuai dengan kebutuhan.
3. **Manajemen Data:** Database menyediakan alat-alat untuk manajemen data seperti penyisipan (insertion), pembaruan (update), dan penghapusan (deletion) data. Hal ini memungkinkan pemrogram untuk mengelola perubahan data seiring waktu.
4. **Integritas Data:** Database dapat menerapkan aturan dan batasan yang memastikan integritas data. Ini melibatkan penerapan konstrain referensial, validasi data, dan aturan bisnis untuk memastikan data yang disimpan tetap konsisten dan akurat.
5. **Keamanan:** Database menyediakan tingkat keamanan yang berbeda untuk data, memungkinkan pengaturan izin akses yang sesuai. Ini memastikan bahwa hanya pengguna yang diotorisasi yang dapat mengakses dan memodifikasi data tertentu.

6. **Pemulihan Data:** Database mencadangkan data secara teratur dan menyediakan mekanisme pemulihan data jika terjadi kegagalan sistem atau kesalahan pengguna. Ini membantu melindungi data dari kehilangan atau kerusakan.
7. **Pemrosesan Transaksi:** Database mendukung pemrosesan transaksi, yang memungkinkan aplikasi untuk menangani operasi yang melibatkan sejumlah besar data dengan aman dan konsisten. Contoh transaksi meliputi pembayaran, pemesanan, dan perubahan stok.
8. **Skalabilitas:** Database dapat ditingkatkan kapasitasnya atau diubah struktur datanya untuk mengakomodasi pertumbuhan data. Ini memungkinkan aplikasi untuk berkembang seiring waktu tanpa perlu mengganti sistem penyimpanan data.
9. **Optimasi Kinerja:** Database menyediakan alat dan teknik untuk mengoptimalkan kinerja aplikasi dengan cara mengindeks data, mengoptimalkan kueri, dan melakukan tuning database.

## 2. SQL

Jenis database yang paling umum digunakan dalam pemrograman adalah database relasional, yang menggunakan tabel untuk menyimpan data dan menggunakan bahasa SQL (Structured Query Language) untuk mengakses dan mengelola data. Namun, ada juga jenis database lain seperti database NoSQL yang digunakan untuk kasus penggunaan tertentu di mana lebih mengutamakan skema fleksibel dan kinerja tinggi.

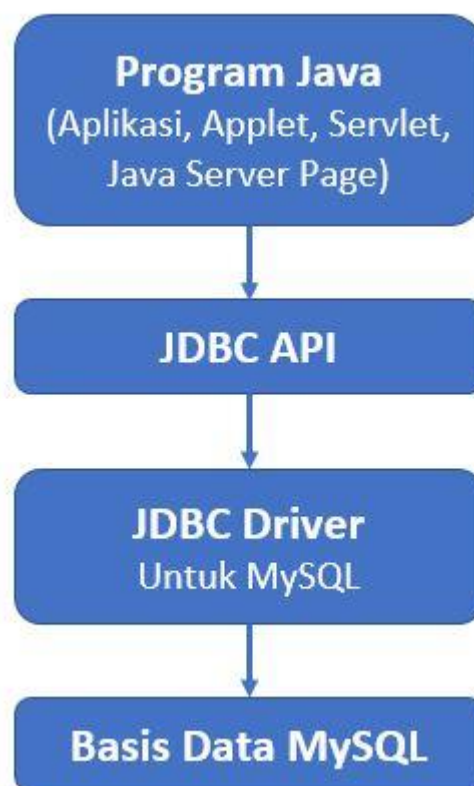


### 3. JDBC

Bahasa pemrograman Java menyediakan standard API (application programming interface) untuk pengembangan program aplikasi basis data (database) yang disebut dengan JDBC API. JDBC adalah API Java untuk memanipulasi basis data. Dengan JDBC API, para pengembang aplikasi dan applet Java diberi kemudahan untuk mengakses berbagai tipe basis data dari berbagai penyedia basis data (database vendors) seperti MySQL Server, SQL Server, Oracle, Sybase dan sebagainya.

JDBC merupakan singkatan dari Java Database Connectivity dan menjadi perantara antara Java dengan basis data. JDBC adalah sebuah spesifikasi yang menyediakan sekumpulan interface yang membolehkan akses portabel ke semua basis data. Dapat dikatakan pula bahwa JDBC hanya menyediakan interface standar, sedangkan masing-masing database vendors membuat driver yang diperlukan sebagai interface yang sebenarnya antara program Java (aplikasi, applet, servlet atau JSP) dengan basis data.

Sebuah program Java yang mengakses data dari basis data harus menggunakan JDBC driver yang khusus untuk basis data tersebut. Contohnya, jika program Java ingin berinteraksi dengan basis data MySQL, harus digunakan JDBC driver dari MySQL. MySQL JDBC Driver disebut juga dengan MySQL Connector/J.



#### 4. Koneksi Database dengan Java

Pada pemrograman Java dengan menggunakan JDBC, ada beberapa langkah yang secara umum harus dilakukan sehingga aplikasi tersebut dapat berinteraksi dengan database server. Langkah-langkah untuk berinteraksi dengan database server dengan menggunakan JDBC adalah sebagai berikut :

##### a. Mengimpor package java.sql

Pertama-tama yang harus dilakukan sebelum kita membuat program JDBC adalah mengimpor package java.sql terlebih dahulu, karena di dalam package java.sql tersebut terdapat kelas-kelas yang akan digunakan dalam proses-proses berinteraksi dengan database server misalnya kelas DriverManager, Connection, dan ResultSet. Adapun listing untuk mengimpor package java.sql adalah sebagai berikut :

```
Import java.sql.*;
```

##### b. Memanggil Driver JDBC

Untuk membuat koneksi dengan database kita harus memanggil JDBC Driver dari database server yang kita gunakan. Driver adalah library yang digunakan untuk berkomunikasi dengan database server. Driver dari setiap database server berbeda-beda, sehingga Anda harus menyesuaikan Driver JDBC sesuai dengan database server yang Anda gunakan.

Berikut ini adalah listing program untuk memanggil driver JDBC.

```
Class.forName(namaDriver);
```

atau

```
Class.forName(namaDriver).newInstance();
```

Kedua cara di atas memiliki fungsi yang sama yaitu melakukan registrasi class driver dan melakukan intansiasi. Apabila driver yang dimaksud tidak ditemukan, maka program akan menghasilkan exception berupa **ClassNotFoundException**. Untuk menghasilkan exception apabila driver tidak ditemukan, maka diperlukan penambahan **try-catch**. Adapun cara menambahkan **try-catch** untuk penanganan error apabila driver tidak ditemukan adalah sebagai berikut :

```

try {
    Class.forName("com.mysql.cj.jdbc.Driver");
} catch (ClassNotFoundException cnfe) {
    System.out.println("Pesan Error : " + cnfe)
}

```

Dari contoh listing di atas dapat dijelaskan bahwa apabila driver yang dipanggil tidak ditemukan, maka program akan menampilkan pesan pada consule dengan isi pesan adalah **“Java.lang.ClassNotFoundException : com.mysql.jdbc.Driver”**. Penanganan error sangat penting dilakukan karena dapat membantu kita dalam mengetahui kesalahan-kesalahan yang terjadi dalam menjalankan program sehingga kita dapat mengatasi kesalahan-kesalahan tersebut. Berikut ini adalah daftar nama-nama driver dari beberapa database server yang sering digunakan.

Database Server	Nama Driver
JDBC-ODBC	sun.jdbc.odbc.JdbcOdbcDriver
MySQL	com.mysql.jdbc.Driver
PostgreSQL	org.postgresql.Driver
Microsoft SQLServer	com.microsoft.jdbc.sqlserver.SQLServerDriver
Oracle	oracle.jdbc.driver.OracleDriver
IBM DB2	COM.ibm.db2.jdbc.app.DB2Driver

### c. Membangun Koneksi

Setelah melakukan pemanggilan terhadap driver JDBC, langkah selanjutnya adalah membangun koneksi dengan menggunakan interface **Connection**. Object Connection yang dibuat untuk membangun koneksi dengan database server tidak dengan cara membuat object baru dari interface Connection melainkan dari class **DriverManager** dengan menggunakan method **getConnection()**.

```
Connection koneksi = DriverManager.getConnection(<argumen>)
```

Ada beberapa macam argumen yang berbeda dari method **getConnection()** yang dipanggil dari **DriverManager**, yaitu :

- **getConnection(String url)**

Pada method kita hanya memerlukan argumen URL, sedangkan untuk data user dan password sudah diikutkan secara langsung pada URL sehingga tidak perlu lagi secara terpisah mendefinisikan data user dan password. Adapun penulisan nilai URL dari method **getConnection(String url)** adalah sebagai berikut :

`jdbc:<DBServer>://[Host][:Port]/<namaDB>?<user=User>&<password=Pasword>`

Misalkan kita menggunakan database server berupa **MySQL** dengan spesifikasi menggunakan host adalah **localhost** dan port default (**3306**), nama database adalah **Dbase**, nama user adalah **ptik**, dan password adalah **uns**. Maka penulisan URL adalah sebagai berikut :

```
try {
    String url = "jdbc:mysql://localhost:3306/Dbase?
user=ptik&password=uns";
    Connection koneksi = DriverManager.getConnection(url);
    System.out.println("Proses apabila koneksi sukses");
} catch (SQLException sqle) {
    System.out.println("Proses apabila koneksi gagal
dilakukan");
}
```

- **getConnection(String url, Properties info)**

Pada method ini memerlukan **URL** dan sebuah object **Properties**. Sebelum menggunakan method ini, Anda harus melakukan import package berupa **java.util.\***, ini dikarenakan object Properties terdapat pada package tersebut. Object Properties berisikan spesifikasi dari setiap parameter database misalnya user name, password, autocommit, dan sebagainya. Berikut ini contoh penggunaan method ini didalam program :

```
try {
    String url = "jdbc:mysql://localhost:3306/Dbase";
    Properties prop = new java.util.Properties();
    prop.put("username", "ptik");
    prop.put("password", "uns");
    Connection koneksi = DriverManager.getConnection(url,
prop);
    System.out.println("Proses apabila koneksi sukses");
} catch (SQLException sqle) {
    System.out.println("Proses apabila koneksi gagal
dilakukan");
}
```

- **getConnection(String url, String user, String password)**

Pada method ini memerlukan argumen berupa **URL**, **user name**, dan **password**. Method ini secara langsung mendefinisikan nilai URL, user name dan password. Berikut ini contoh penggunaan method ini didalam program :

```

        String url = "jdbc:mysql://localhost:3306/Dbase";
        String user = "ptik";
        String password "uns";
    try {
        Connection koneksi =
        DriverManager.getConnection(url, user, password);
        System.out.println("Proses apabila koneksi
        sukses");
    } catch (SQLException sqle) {
        System.out.println("Proses apabila koneksi gagal
        dilakukan");
    }
}

```

Berikut ini adalah daftar penulisan URL dari beberapa database server yang sering digunakan.

Database Server	Nama URL	Contoh penggunaan
JDBC-ODBC	jdbc:odbc:<NamaDatabase>	jdbc:odbc:Dbase
MySQL	jdbc:mysql://<nmHost>:<port>/<nmDB>	jdbc:mysql://localhost:3306/Dbase
PostgreSQL	jdbc:postgresql://<nmHost>:<port>/<nmDB>	jdbc:postgresql://localhost:5432/Dbase
Microsoft SQLServer	jdbc:microsoft:sqlserver://<nmHost>:<port>; DatabaseName=<namaDatabase>	jdbc:microsoft:sqlserver://localhost:1433; DatabaseName=Dbase
Oracle	jdbc:oracle:thin:@<nmHost>:<port>:<nmDB>	jdbc:oracle:thin:@localhost:1521:Dbase
IBM DB2	jdbc:db2:<NamaDatabase>	jdbc:db2:Dbase

#### d. Membuat Statement

JDBC API menyediakan interface yang berfungsi untuk melakukan proses pengiriman statement SQL yang terdapat pada package java.sql. Di dalam JDBC API disediakan tiga buah interface untuk fungsi tersebut yaitu :

- **Statement**

Interface ini dibuat oleh method **Connection.createStatement()**. Object Statement digunakan untuk pengiriman statement SQL tanpa parameter.

```
Statement stat = Connection.createStatement();
```

- **PreparedStatement**

Interface ini dibuat oleh method **Connection.prepareStatement()**. Object PreparedStatement digunakan untuk pengiriman statement SQL dengan atau tanpa parameter. Dengan object ini, kita dapat menampung satu atau lebih

parameter sebagai argumen input (parameter IN). Interface ini dapat menjalankan beberapa proses dalam sekali pengiriman perintah SQL.

```
PreparedStatement stat = Connection.prepareStatement();
```

- **CallableStatement**

Interface ini dibuat oleh method **Connection.prepareStatement()**. Object **CallableStatement** digunakan untuk menjalankan store procedure SQL.

```
CallableStatement stat = Connection.prepareCall();
```

e. **Melakukan Query**

Setelah kita memiliki object statement, kita dapat menggunakannya untuk melakukan pengiriman perintah SQL dan mengeksekusinya. Method eksekusi yang digunakan untuk perintah SQL terbagi menjadi dua bagian yaitu untuk perintah **SELECT** method eksekusi yang digunakan adalah **executeQuery()** dengan nilai kembaliannya adalah **ResultSet**, dan untuk perintah **INSERT**, **UPDATE**, **DELETE** method eksekusi yang digunakan adalah **executeUpdate()**. Berikut ini adalah contoh melakukan eksekusi perintah SQL dan mengambil hasilnya (**ResultSet**) dengan menggunakan perintah **SELECT**.

```
String sql = "SELECT kode, nama, alamat, kelas FROM  
dataSiswa";  
ResultSet set = stat.executeQuery(sql);  
while (set.next()) {  
    String kode = set.getString("kode");  
    String nama = set.getString("nama");  
    String alamat = set.getString("alamat");  
    String kelas = set.getString("kelas");  
}
```

f. **Menutup Koneksi**

Penutupan terhadap koneksi database perlu dilakukan agar sumber daya yang digunakan oleh object **Connection** dapat digunakan lagi oleh proses atau program yang lain. Sebelum kita menutup koneksi database, kita perlu melepas object **Statement** dengan kode sebagai berikut :

```
statement.close();
```

Untuk menutup koneksi dengan database server dapat kita lakukan dengan kode sebagai berikut :





```
connection.close();
```

## D. PRAKTIKUM

### 1. Persiapan Database

- Buatlah database dengan nama **DatabaseDemo**.
- Buatlah tabel dengan nama **mobil** seperti berikut

Fields	Indexes	Foreign Keys	Checks	Triggers	Options	Comment	SQL Preview						
Name				Type	Length	Decimals	Not null	Virtual	Key				
 id				int	11		<input checked="" type="checkbox"/>	<input type="checkbox"/>	 1				
vendor				varchar	255		<input type="checkbox"/>	<input type="checkbox"/>					
tipe				varchar	255		<input type="checkbox"/>	<input type="checkbox"/>					
mesin				varchar	255		<input type="checkbox"/>	<input type="checkbox"/>					
maxSpeed				double			<input type="checkbox"/>	<input type="checkbox"/>					

- Masukkan data berikut

id	vendor	tipe	mesin	maxSpeed
1	Porsche	911	4000	193
2	Audi	R8	5000	205
3	Pagani	Huayra	6000	225
4	Toyota	Supra	3000	155

### 2. Menulis Kode

- Import semua library yang dibutuhkan

```
package com.mycompany.databasedemo;

import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.DriverManager;
```

- Di dalam fungsi main(), definisikan config

```
String url = "jdbc:mysql://localhost:3306/databasedemo";
String username = "root";
String password = "";
```

- Buat perintah koneksi di dalam blok try catch

```
try {
    Class.forName(className: "com.mysql.cj.jdbc.Driver");
    Connection koneksi = DriverManager.getConnection(url, user: username, password);
    Statement st = koneksi.createStatement();
    System.out.println("Koneksi berhasil");
} catch (ClassNotFoundException ex) {
    System.out.println("ex.getMessage()");
}
```

- Modifikasi blok try catch dengan menambahkan statement dan mencetak query

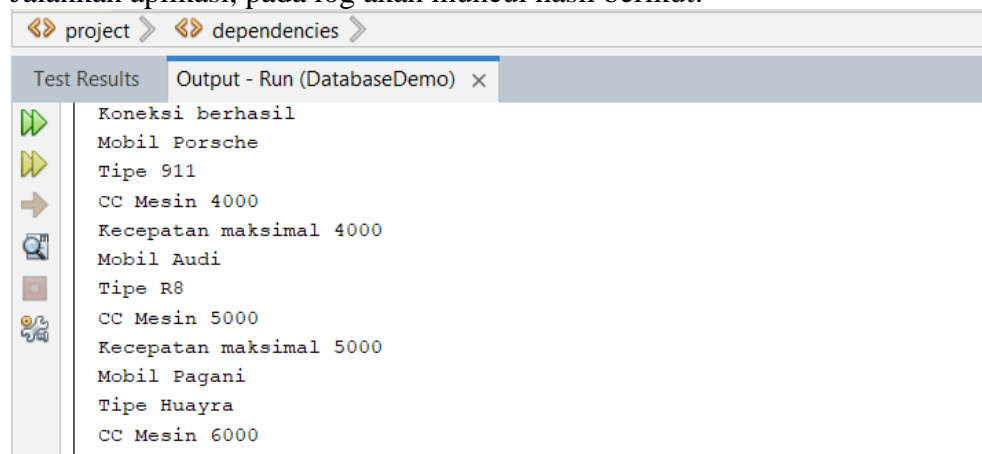
```
String query = "select *from mobil";
ResultSet rs = st.executeQuery(sql: query);

while (rs.next()) {
    System.out.println("Mobil " + rs.getString(columnLabel: "vendor"));
    System.out.println("Tipe " + rs.getString(columnLabel: "tipe"));
    System.out.println("CC Mesin " + rs.getString(columnLabel: "mesin"));
    System.out.println("Kecepatan maksimal " + rs.getInt(columnIndex: 4));
}
```

- Tutup koneksi

```
st.close();
koneksi.close();
System.out.println("Koneksi ditutup....");
```

- Jalankan aplikasi, pada log akan muncul hasil berikut:



### 3. Penyelesaian Masalah

- Jika ditemui masalah seperti ini, maka kemungkinan project kalian belum menggunakan JDBC driver

```
--- compiler:3.10.1:compile (default-compile) @ DatabaseDemo ---
Changes detected - recompiling the module!
Compiling 2 source files to C:\Users\nurca\Documents\NetBeansProjects\l

--- exec:3.1.0:exec (default-cli) @ DatabaseDemo ---
com.mysql.cj.jdbc.Driver

-----
BUILD SUCCESS
-----

Total time: 2.169 s
Finished at: 2023-09-16T11:41:49+07:00
-----
```

- Buka <https://mvnrepository.com/artifact/mysql/mysql-connector-java> dan pilih versi terbaru
- Pada halaman driver, perhatikan pada tab maven. Copy kode tersebut

Categories	JDBC Drivers
Tags	database sql jdbc driver connector mysql
Date	Apr 18, 2023
Files	<a href="#">pom (2 KB)</a> <a href="#">jar</a> <a href="#">View All</a>
Repositories	Central
Ranking	#68 in MvnRepository (See Top Artifacts) #1 in JDBC Drivers
Used By	7,268 artifacts

**Note:** This artifact was moved to:

[com.mysql » mysql-connector-j](#)

MySQL Connector/J artifacts moved to reverse-DNS compliant Maven 2+ coordinates.

[Maven](#) [Gradle](#) [Gradle \(Short\)](#) [Gradle \(Kotlin\)](#) [SBT](#) [Ivy](#) [Grape](#) [Leiningen](#) [Buildr](#)

```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.33</version>
</dependency>
```

☒ Include comment with link to declaration

- Buka pom.xml pada root project kalian.
- Tambahkan tag **<dependencies></dependencies>** lalu paste kode maven tersebut di dalam tag **<dependencies>**

```
<maven.compiler.source>20</maven.compiler.source>
<maven.compiler.target>20</maven.compiler.target>
<exec.mainClass>com.mycompany.databasedemo.DatabaseDemo</exec.mainClass>
</properties>
<dependencies>
  <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.33</version>
  </dependency>
</dependencies>
</project>
```

## E. LATIHAN

1. Modifikasilah file hasil praktikum kalian di atas.
  - Pisahkan bagian yang memuat koneksi database ke dalam kelas tersendiri sehingga program kalian menerapkan design pattern Singleton!
  - Buat query yang dapat memasukkan data baru ke dalam tabel mobil!Simpan kode dengan nama **Latihan1.zip**
2. Buatlah sebuah program yang dapat melakukan pengisian dan pembacaan data anggota perpustakaan. Operasi yang perlu tersedia adalah mendaftarkan data anggota dan menampilkan semua data anggota. dengan nama **Latihan2.zip!**

[SELAMAT DATANG DI PERPUS PUSAT UNS]

1. Pendaftaran anggota
2. Tampilkan data anggota
3. Keluar

=====

Masukkan perintah : 1

Masukkan Nama : Nurcahya Pradana

Masukkan NIM : K353535

Masukkan Prodi : PTIK

Data berhasil dibuat

=====

Masukkan perintah : 2

[DATA ANGGOTA]

Budi - K353531 - PTIK

Randy - M0509051 - Informatika

Nurcahya - K353535 - PTIK

=====

Masukkan perintah : 3

Sampai jumpa!