**Java Database**

## **Pendahuluan**

**Sebelumnya harus sudah paham materi terkait** :

* Java Dasar
* Java Object Oriented Programming
* Java Generic
* Java Collection
* Java Lambda
* Apache Maven
* Java Unit Test
* Java Stream
* MySQL Database

**Agenda**

* Pengenalan JDBC
* Driver
* Data Source
* Connection
* JDBC Data Type
* Statement
* ResultSet
* Prepared Statement
* SQL Injectio
* Metadata
* Dan lain lain

## **Pengenalan JDBC**

* JDBC singkatan dari Java Database Connectivity
* JDBC merupakan spesifikasi API standard untuk mengakses database di java
* JDBC tidak bisa langsung digunakan, karena isinya hanyalah interface-interface kontrak untuk berinteraksi dengan database
* JDBC perlu implementasi, atau kita sebut dengan Driver, seperti MySQL Driver, PostgreSQL Driver, OracleDB Driver, dan lain – lain
* Semua interface API JDBC terdapat di package java.sql dan javax.sql
* <https://docs.oracle.com/en/java/javase/15/docs/api/java.sql/java/sql/package-summary.html>
* <https://docs.oracle.com/en/java/javase/15/docs/api/java.sql/javax/sql/package-summary.html>

**Cara Kerja JDBC**

Database

Driver

JDBC

App

## **Membuat Project**

* Kita akan menggunakan Apache Maven untuk membuat project
* Kita bisa menggunakan perintah :
* mvn archetype:generate
* Lalu cari : maven-archetype-quickstart

**JUnit 5**

* Kita akan belajar Java database menggunkan unit test, jadi pastikan menambahkan dependency JUnit 5 di project nya
* Silahkan cari junit-jupiter di <https://search.maven.org/>

## **Driver**

* Driver adalah jembatan penghubung antara JDBC dan Database Management System yang akan kita gunakan
* Sebenarnya Driver itu berisikan class – class implementasi dari interface yang terdapat di JDBC
* Tanpa menggunkan Driver, kita JDBC tidak bisa terkoneksi ke DBMS
* Driver di JDBC direpresentasikan oleh interface java.sql.Driver
* <https://docs.oracle.com/en/java/javase/15/docs/api/java.sql/java/sql/Driver.html>

**MySQl Driver**

* MySQL sudah menyediakan driver untuk JDBC
* Kita bisa cari MySQL Driver dengan menggunakan kata kunci mysql-connector-java di https://search.maven .org/
* Lalu tambahkan ke dependency project kita

**Registrasi Driver ke JDBC**

* Setelah menambah dependency MySQL Driver ke project kita
* Kita perlu melakukan registrasi Driver terlebih dahulu
* Untuk melakukan registrasi, kitab isa menggunakan static method registerDriver milik class java.sql.DriverManager

## **Connection**

* Setelah melakukan registrasi Driver ke JDBC, kita sekarang bisa mulai melakukan koneksi ke database
* Untuk melakukan koneksi ke database, kita harus memberi tahu jenis database, host, port, username dan password untuk terkoneksi ke database
* Semua itu biasanya digabungkan dalam sebuah string yang biasa disebut JDBC URL
* Koneksi di ke database di representasikan oleh interface java.sql.Connection
* <https://docs.oracle.com/en/java/javase/15/docs/api/java.sql/java/sql/Connection.html>

**MySQL JDBC URL**

* Tiap database biasanya punya format jdbc url sendiri – sendiri
* Contohnya di MySQ, kitab isa menggunakan format seperti ini
* Jdbc:mysql://host:port/namadatabase

**Membuat Connection**

* Untuk membuat Connection ke database, kitab isa menggunakan static method getConnection() di class java.sql.DriverManager
* Semua method di JDBC rata-rata selalu akan menghasilkan SQLException

**Menutup Connection**

* Saat kita selesai menggunkan Connection, disarankan untuk selalu menutup Connection tersebut
* Jika sebuah Connection tidak kita tutup, maka selama aplikasi kita berjalan, koneksi ke database akan selalu terbuka
* Jika Connection yang terbuka ke database terlalu banyak, ditakutkan nanti kita tidak bisa membuka koneksi lagi ke database dikarenakan sudah menyentuh nilai maksimum koneksi yang bisa di tangani oleh database nya
* Contohnya, maksimum Connection di MySQL adalah 151
* <https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html#sysvar_max_connections>

## **DataSource**

**Masalah Dengan Connection**

* Connection adalah sebuah resource yang sangat mahal
* Jika aplikasi kita sangat ketergantungan dengan database, maka membuka tutup koneksi setiap ada request sangatlah mahal harganya
* Connection itu sangat lambat ketika pertama kali dibuat, dan sangat mahal memakan memory
* Oleh karena itu, melakukan manajemen Connection secara manual sangatlah tidak direkomendasikan saat nanti kita membuat aplikasi

**Connection Pool**

* Connection Pool adalah konsep dimana dibanding kita membuat koneksi baru setiap ada request ke yang membutuhkan database
* Lebih baik di awal kita buatkan banyak Connection terlebih dahulu, sehingga hanya lambat diawal ketika aplikasi berjalan
* Selanjutnya kita ada request yang butuh koneksi, kita hanya cukup menggunakan salah satu Connection, dan setelah selesai, kita kembalikan lagi Connection nya
* Jika semua Connection sedang terpakai semua, ketika ada request yang butuh koneksi lagi, request tersebut diminta untuk menunggu terlebih dahulu, dengan demikian penggunaan memory untuk Connection tidak akan terlalu bengkak
* Connection Pool di JDBC direpresentasikan dengan interface javax.sql.DataSource

**HikariCP**

* Membuat connection pool secara manual bukanlah hal bijak, lebih baik kita menggunakan library connection pool yang sudah terbukti bekerja dengan baik
* HikariCP adalah salah satu connection pool library paling populer saat ini di Java
* Kita bisa menggunakan HikariCP ini untuk melakukan connection pool terhadap koneksi database di aplikasi kita
* <https://github.com/brettwooldridge/HikariCP>

## **Statement**

* Java.sql.Statement adalah interface yang bisa kita gunakan mengirim SQL ke database. Sekaligus menerima response data dari database
* Ada banyak method yang bis akita gunakan di Statement untuk mengirim perintah SQL, kita akan bahas satu persatu
* <https://docs.oracle.com/en/java/javase/15/docs/api/java.sql/java/sql/Statement.html>
* Untuk membuat statement kitab isa menggunakan method createStatement() milik Connection

**Statement.executeUpdate(sql)**

* Method pertama yang akan kita bahas adalah executeUpdate(sql)
* Method ini digunakan untuk mengirim perintah SQL INSERT, UPDATE, DELETE atau apapun yang tidak membutuhkan result
* Bahkan bisa perintah SQL DDL (create table, create index, dan lain-lain), walaupun best practice nta, erintah DDL lebih baik dilakukan langsung di database, atau menggunakan migration script, tidak dari aplikasi
* executeUpdate(sql) mengembalikan return int, dimana ini biasanya mengembalikan berapa banyak record di database yang terkena impact perintah SQL kita, misal ketika mengirim perintah UPDATE, berapa banyak record yang ter-update misalnya

**Statement.executeQuery(sql)**

* Jika kita ingin mengirim perintah SQL yang mengembalikan data, maka kita bisa menggunakan method executeQuery(sql)
* Method ini akan mengembalikan object java.sql.ResultSet, yaitu berisikan data-data hasil query SQL yang kita kirim
* Pembahasan interface ResultSet akan kita bahas lebih detail lagi nanti di chapter tersendiri

## **ResultSet**

* Seperti yang sudah kita bahas sebelumnya di Statement, ketika kita melakukan query, maka akan menghasilkan java.sql.ResultSet
* ResultSet adalah representasi data hasil query dari database
* ResultSet itu mirip seperti iterator, jadi kita bisa melakukan perulangan di ResultSet untuk mendapatkan data tiap record nya
* <https://docs.oracle.com/en/java/javase/15/docs/api/java.sql/java/sql/ResultSet.html>

**Mengambil Data Kolom di ResultSet**

* Cara kerja ResultSet adalah seperti cursor, dimana untuk maju kita menggunakan method next() dan untuk mundur, kita bisa menggunakan method previous()
* Untuk mendapatkan data tiap kolom saat sekarang kita berada di lokasi cursor nya, kita bisa menggunkan banyak sekali method getXxx(column) di ResultSet
* Kita bisa sesuaikan dengan tipe data kolom tersebut, misal getString(column), getInt(column), dan lain – lain

## **SQL Injection**

**SQL dengan Parameter**

* Saat membuka aplikasi, kita tidak mungkin akan melakukan hardcode perintah SQL di kode Java kita
* Biasanya kita akan menerima input data dari user, lalu membuat perintah SQL dari input user, dan mengirimnya menggunakan perintah SQL

**SQL Injection**

* SQL Injection adalah sebuah teknik yang menyalahgunakan sebuah celah keamanan yang terjadi dalam lapisan basis data sebuah aplikasi.
* Biasa, SQL Injection dilakukan dengan mengirim input dari user dengan perintah yang salah, sehingga menyebabkan hasil SQL yang kitab buat menjadi tidak valid
* SQL Injection sangat berbahaya, jika sampai kita salah membuat SQL, bisa jadi data kita tidak aman

**Solusinya?**

* Statement tidak didesain untuk bisa menangani SQL Injecton
* Oleh karena itu, jika SQL query yang kita gunakan dibuat berdasarkan input dari user, maka kita jangan menggunakan Statement
* Untuk menghindari SQL injection, terdapat interface bernama PreparedStatement, ini adalah jenis statement yang bisa menangani SQL Injection dengan baik
* Kita akan bahas PreparedStatement di chapter tersendiri

## **PreparedStatement**

**Membuat PreparedStatement**

* Berbeda dengan Statement, ketika pembuatannya, sudah ditentukan SQL apa yang akan kita gunakan
* Oleh karena itu, PreparedStatement biasanya digunakan untuk sekali mengirim perintah SQl, jika ingin mengirim perintah SQL lagi, kita harus membuat PreparedStatement baru
* Untuk membuat PreparedStatement, kitab isa menggunakan method preparedStatement(sql) milik Connection

**Menerima Input User**

* Sekarang pertanyaannya, bagaimana cara menerima input user meggunakan PreparedStatement?
* Untuk menerima input dari user, SQL yang kita buat harus diubah juga
* Input dari user, perlu kita ubah manjadi ?(tanda tanya)
* Nanti ketika pembuatan object, kita bisa substitusi datanya menggunakan set Xxx(index, value) sesuai dengan tipe datanya, misal setString(), setInt() dan lain - lain

## **Batch Process**

**Request dan Response**

* Secara default, komunikasi antara database dan aplikasi Java kita adalah request dan response
* Artinya, setiap kali kita mengirim perintah SQL, maka aplikasi kita akan menunggu sampai database melakukan response dari perintah SQL tersebut
* Proses tersebut kadang terlalu chatty kalo tujuan misal ingin mengirim data secara banyak ke database

**Batch Process**

* Batch process adalah proses mengirim perintah secara banyak sekaligus.
* Biasanya batch process dilakukan dalam kasus tertentu saja, misal ketika kita ingin mengirim import data dari file excel ke database yang jumlahnya jutaan.
* Biasanya dalam batch process, yang diutamakan adalah kecepatan, karena jika perintah SQL nya di execute satu satu dan menunggu response satu satu, maka sudah pasti akan sangat lambat sekali

**Batch Process di JDBC**

* JDBC mendukung proses eksekusi perintah SQL secara batch di Statement ataupun di PreparedStatement
* Di Statement, terdapat method addBatch(sql) untuk menambahkan perintah ke proses batch
* Sedangkan di PreparedStatement terdapat method addBatch() untuk menambahkan proses ke batch, lalu bisa gunakan method clearParameters() untuk menghapus parameter input user sebelumnya
* Setelah proses batch selesai, untuk megeksekusinya, kita bisa gunakan perintah executeBatch()
* Jadi jika butuh query sql yang berbeda- beda(ada insert, update, delet) dalam satu batch proses sebaiknaya menggunkan statement, kaerna untuk preparedstatement hanya bisa melkukan satu jenis query saja.

**PERINGATAN**

* Proses batch akan disimpan di memory sebelum dikirim ke database
* Oleh karena itu, bijaklah membuat proses batch, jangan terlalu banyak menambahkan ke batch, misal per 100 atau per 1000
* Jika sudah mencapai 100 atau 1000, kita bisa mengirim batch tersebut menggunakan perintah executeBatch()

## **Auto Increment**

* Kadang setelah melakukan INSERT data ke database yang memiliki primary key auto increment, kita ingin mendapatkan data id terbarunya
* Di MySQL sebenarnya kita bisa melakukan query SELECT LAST\_INSERT\_ID(), namun berarti kita harus melakukan query ulang dengan Statement dan melakukan iterasi lagi dengan ResultSet
* Untungnya di JDBC, ada kemampuan untuk mendapatkan auto generate data seperti auto increment dengan method getGenerateKeys() yang mengembalikan ResultSet
* Selanjutnya kitab isa melakukan iterasi terhadap ResultSet tersebut

**Mendapatkan Auto Generated Key**

* Secara default, Statement ataupun PreparedStatement tidak mengerti untuk mengambil auto generate key
* Kita perlu memberi tahunya agar Statement atupun PreparedStatement mengambil auto generate id secara otomatis
* Untuk Statement, kita perlu memberi tahu ketika memanggil method executeUpdate(sql, Statement.RETURN\_GENERATED\_KEYS)
* Sedangkan untuk PreparedStatement, kita perlu memberi tahu ketika memanggil method prepareStatement(sql, Statement.RETURN\_GENERATED\_KEYS)
* Setelah itu, untuk mendapatkan auto generate key, kita bisa menggunakan method getGeneratedKeys(), method ini akan error jika kita lupa mengirim parameter generated keys

## **Date, Time dan Timestamp**

* Seperti kita ketahui, tipe tanggal dan waktu di database biasanya banyak, ada Date, Time, ada juga Timestamp
* Sedangkan di java hanya terdapat tipe data java.util.Date
* Agar bisa menghandle hal ini, terdapat class-class turunan java.util.Date di package java.sql yang bernama Date, Time, dan Timestamp
* Sesuai dengan namanya, class – class tersebut digunakan sebagai representasi di Java
* Secara otomatid JDBC bisa melakukan konversi tipe data tersebut dari database menjadi object di Java

## **Database Transaction**

* Fitur paling bermanfaat di database salah satunya adalah database transaction
* Fitur database transaction pun bisa kita jalankan menggunakan JDBC
* Jika belum mengerti apa itu database transaction, silahkan buka materi tentang database MySQL

**Auto Commit**

* Secara default, Connection yang kita buat menggunakan JDBC memililiki sifat auto commit
* Auto commit artinya setiap perintah SQL yang kita kirim akan langsung di commit secara otomatis
* Karena dalam database transaction, kita biasanya melakukan commit transaction setelah semua proses selesai, maka kita perlu mematikan auto commit di JDBC
* Untuk mematikan fitur auto commit di JDBC, kita bisa menggunakan method di Connection bernama setAutoCommit(false)
* Setelah selesai melakukan proses, kita bisa melakukan commit transaction dengan menggunakan method commit() milik Connection
* Untuk membatalkan proses transaksi(rollback), kita bisa menggunakan method rollback() milik connection

**MetaData**

* Kadang kita ingin informasi seputar database yang kita gunakan
* Informasi tersebut bernama MetaData
* Ada banyak jenis metadata, seperti DatabaseMetaData, ParameterMetaData, ResultSetMetaData
* <https://docs.oracle.com/en/java/javase/15/docs/api/java.sql/java/sql/DatabaseMetaData.html>
* <https://docs.oracle.com/en/java/javase/15/docs/api/java.sql/java/sql/ParameterMetaData.html>
* <https://docs.oracle.com/en/java/javase/15/docs/api/java.sql/java/sql/ResultSetMetaData.html>

**DatabaseMetaData**

DatabseMetaData adalah informasi seputar seluruh database yang kita gunakan, seperti misal :

* Nama database
* Versi database
* Table yang ada di database
* dan lain-lain

Untuk membuat DatabaseMetaData, kitab isa menggunakan method getMetaData() dari object Collection

**ParameterMetadata**

* ParameterMetaData adalah informasi seputar parameter yang terdapat di PreparedStatement
* Dengan ParameterMetadata, kitab isa mendapat banyak informasi parameter yang bisa digunakan untuk input di PreparedStatement, seperti berapa banyak parameter, tipe data parameter, dan lain-lain
* Namun peril diperhatikan, beberapa Driver mungkin tidak mendukung untuk mendapatkan jenis tipe parameter di ParameterMetadata

**ResultSetMetadata**

* ResultSetMetaData adalah informasi seputar hasil ResultSet
* Dengan ResultSetMetaData, kita bisa mendapatkan informasi tentang jumlah kolom, nama kolom, tipe data tiap kolom nya, dan lain-lain