Materi Pelatihan

JDBC

Ifnu Bima
ifnu@artivisi.com
http://ifnu.artivisi.com

http://www.artivisi.com
Jakarta, 2009

Daftar Isi

Koneksi Database Dengan JDBC

Mengenal JDBC

Java Database Connectivity adalah API yang digunakan Java untuk melakukan koneksi dengan aplikasi lain atau dengan berbagai macam database. JDBC memungkinkan kita untuk membuat aplikasi Java yang melakukan tiga hal: konek ke sumber data, mengirimkan query dan statement ke database, menerima dan mengolah resultset yang diperoleh dari database.

JDBC mempunyai empat komponen :

1. JDBC API

   JDBC API menyediakan metode akses yang sederhana ke sumber data relational (RDBMS) menggunakan pemrograman Java. dengan menggunakan JDBC API, kita bisa membuat program yang dapat mengeksekusi SQL, menerima hasil ResultSet, dan mengubah data dalam database. JDBC API juga mempunyai kemampuan untuk berinteraksi dengan lingkungan terdistribusi dari jenis sumber data yang berbeda-beda.

   JDBC API adalah bagian dari Java Platform yang disertakan dalam library JDK maupun JRE. JDBC API sekarang ini sudah mencapai versi 4.0 yang disertakan dalan JDK 6.0. JDBC API 4.0 dibagi dalam dua package yaitu : java.sql dan javax.sql.

2. JDBC Driver Manager

   Class DriverManager dari JDBC bertugas untuk mendefisikan object-object yang dapat digunakan untuk melakukan koneksi ke sebuah sumber data. Secara tradisional DriverManager telah menjadi tulang punggung arsitektur JDBC.

3. JDBC Test Suite

   JDBC Test Suite membantu kita untuk mencara driver mana yang cocok digunakan untuk melakukan sebuah koneksi ke sumber data tertentu. Tes yang dilakukan tidak memerlukan resource besar ataupun tes yang komprehensif, namun cukup tes-tes sederhana yang memastikan fitur-fitur penting JDBC dapat berjalan dengan lancar.

4. JDBC-ODBC Bridge

   Brige ini menyediakan fasilitas JDBC untuk melakukan koneksi ke sumber data menggunakan ODBC (Open DataBase Connectivity) driver. Sebagai catatan, anda perlu meload driver ODBC di setiap komputer client untuk dapat menggunakan bridge ini. Sebagai konsekuensinya, cara ini hanya cocok dilakukan di lingkungan intranet dimana isu instalasi tidak menjadi masalah.

Dengan keempat komponen yang dipunyainya, JDBC menjadi tools yang dapat diandalkan untuk melakukan koneksi, mengambil data dan merubah data dari berbagai macam sumber data. Modul ini hanya akan membahas dua komponen pertama dari keempat komponen yang dipunyai oleh JDBC, yaitu JDBC API dan DriverManager. Sumber data yang digunakan adalah Relational Database.

Database Driver

JDBC memerlukan database driver untuk melakukan koneksi ke suatu sumber data. Database driver ini bersifat spesifik untuk setiap jenis sumber data. Database driver biasanya dibuat oleh pihak pembuat sumber datanya, namun tidak jarang juga komunitas atau pihak ketiga menyediakan database driver untuk sebuah sumber data tertentu.

Perlu dipahami sekali lagi bahwa database driver bersifat spesifik untuk setiap jenis sumber data. Misalnya, Database Driver MySql hanya bisa digunakan untuk melakukan koneksi ke database MySql dan begitu juga database driver untuk Postgre SQL juga hanya bisa digunakan untuk melakukan koneksi ke database Postgre SQL.

Database driver untuk setiap DBMS pada umumnya dapat didownload dari website pembuat DBMS tersebut. Beberapa vendor DBMS menyebut Database driver ini dengan sebutan Java Connector (J/Connector). Database driver biasanya dibungkus dalam file yang berekstensi jar. Setiap database driver harus mengimplement interface java.sql.Driver.

Membuat Koneksi

Melakukan koneksi ke database melibatkan dua langkah: Meload driver dan membuat koneksi itu sendiri. Cara meload driver sangat mudah, pertama letakkan file jar database driver ke dalam classpath. Kemudian load driver dengan menambahkan kode berikut ini:

```
Class.forName("com.mysql.jdbc.Driver");
```

Nama class database driver untuk setiap DBMS berbeda, anda bisa menemukan nama class tersebut dalam dokumentasi driver database yang anda gunakan. Dalam contoh ini, nama class database driver dari MySql adalah com.mysql.jdbc.Driver.

Memanggil method Class.forName secara otomatis membuat instance dari database driver, class DriverManager secara otomatis juga dipanggil untuk mengelola class database driver ini. Jadi anda tidak perlu menggunakan statement new untuk membuat instance dari class database driver tersebut.

Langkah berikutnya adalah membuat koneksi ke database menggunakan database driver yang sudah diload tadi. Class DriverManager bekerja sama dengan interface Driver untuk mengelola driver-driver yang diload oleh aplikasi, jadi dalam satu sesi anda bisa meload beberapa database driver yang berbeda.

Ketika kita benar-benar melakukan koneksi, JDBC Test Suite akan melakukan serangkaian tes untuk menentukan driver mana yang akan digunakan. Parameter yang digunakan untuk menentukan driver yang sesuai adalah URL. Aplikasi yang akan melakukan koneksi ke database menyediakan URL pengenal dari server databse tersebut. Sebagai contoh adalah URL yang digunakan untuk melakukan koneksi ke MySql :

```
jdbc:mysql://[host]:[port]/[schema]
```

contoh konkritnya :

```
jdbc:mysql://localhost:3306/latihan
```

Setiap vendor DBMS akan menyertakan cara untuk menentukan URL ini di dalam dokumentasi. Anda tinggal membaca dokumentasi tersebut tanpa harus khawatir tidak menemukan informasi yang anda perlukan.

Method DriverManager.getConnection bertugas untuk membuat koneksi:

```
Connection conn =
      DriverManager.getConnection(
       "jdbc:mysql://localhost:3306/latihan");
```

Dalam kebanyakan kasus anda juga harus memasukkan parameter username dan password untuk dapat melakukan koneksi ke dalam database. Method getConnection menerima Username sebagai parameter kedua dan pasword sebagai parameter ketiga, sehingga kode diatas dapat dirubah menjadi :

```
Connection conn =
      DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/latihan",
            "root","");
```

Jika salah satu dari driver yang diload berhasil digunakan untuk melakukan koneksi dengan URL tersebut, maka koneksi ke database berhasil dilaksanakan. Class Connection akan memegang informasi koneksi ke database yang didefinisikan oleh URL tersebut.

Setelah sukses melakukan koneksi ke database, kita dapat mengambil data dari database menggunakan perintah query ataupun melakukan perubahan terhadap database. bagian berikut ini akan menerangkan bagaimana cara mengambil dan memanipulasi data dari database.

Mengambil dan Memanipulasi Data dari Database

Proses pengambilan data dari database memerlukan suatu class untuk menampung data yang berhasil diambil, class tersebut harus mengimplement interface ResultSet.

Object yang bertipe ResultSet dapat mempunyai level fungsionalitas yang berbeda, hal ini tergantung dari tipe dari result set. Level fungsionalitas dari setiap tipe result set dibedakan berdasarkan dua area:

- Dengan cara bagaimana result set itu dapat dimanipulasi
- Bagaimana result set itu menangani perubahan data yang dilakukan oleh proses lain secara bersamaan (concurrent).

JDBC menyediakan tiga tipe result set untuk tujuan berbeda:

1. TYPE_FORWARD_ONLY : result set tersebut tidak bisa berjalan mundur, reslut set hanya bisa berjalan maju dari baris pertama hingga baris terakhir. result set hanya menggambarkan keadaan data ketika query dijalankan atau ketika data diterima oleh resul set. Jika setelah itu ada perubahan data dalam database, result set tidak akan diupdate alias tidak ada perubahan dalam result set tipe ini.

2. TYPE_SCROLL_INSENSITIVE : result set dapat berjalan maju mundur. result set dapat berjalan maju dari row pertama hingga terakhir atau bergerak bebas berdasarkan posisi relatif atau absolute.

3. TYPE_SCROLL_SENSITIVE : result set dapat berjalan maju mundur. result set dapat berjalan maju dari row pertama hingga terakhir atau bergerak bebas berdasarkan posisi relatif atau absolute.

Instance dari object bertipe ResultSet diperlukan untuk menampung hasil kembalian data dari database. Sebelum kita bisa memperoleh instance dari ResultSet, kita harus membuat instance

dari class Statement. Class Statement mempunyai method executeQuery yang digunakan untuk menjalankan perintah query dalam database kemudian mengembalikan data hasil eksekusi query ke dalam object ResultSet.

Berikut ini adalah contoh kode untuk membuat instance class Statement, kemudian menjalankan query untuk mengambil data dari database yang hasilnya dipegang oleh ResultSet :

```
Statement statement =
      conn.createStatement(
            ResultSet.TYPE_SCROLL_SENSITIVE,
            ResultSet.CONCUR_READ_ONLY);
ResulSet rs =
      statement.executeQuery("select * from Customers");
```

ResultSet akan meletakkan kursornya (posisi pembacaan baris) di sebuah posisi sebelum baris pertama. Untuk menggerakkan kursor maju, mundur, ke suatu posisi relatif atau ke suatu posisi absolute tertentu, gunakan method-method dari ResultSet:

- next() -- mengarahkan kursor maju satu baris.
- previous() -- mengarahkan kursor mundur satu baris.
- first() -- mengarahkan kursor ke baris pertama.
- last() -- mengarahkan kursor ke baris terakhir.
- beforeFirst() -- mengarahkan kursor ke sebelum baris pertama.
- afterLast() -- mengarahkan kursor ke setelah baris terakhir.
- relative(int rows) -- mengarahkan kursor relatif dari posisinya yang sekarang. Set nilai rows dengan nilai positif untuk maju, dan nilai negatif untuk mundur.
- absolute(int rowNumber) – mengarahkan kursor ke posisi tertentu sesuai dengan nilai rowNumber, dan tentu saja nilainya harus positif.

Interface ResultSet menyediakan method getter untuk mengakses nilai dari setiap kolom dalam baris yang sedang aktif. Parameter fungsi getter bisa menerima nilai index dari kolom ataupun nama kolomnya. Namun begitu, penggunaan nilai index lebih efisien dibanding menggunakan nama kolom.

Nilai index dimulai dengan satu hingga banyaknya kolom. Penggunaan nama kolom adalah case insensitive, artinya huruf kecil atau huruf besar tidak menjadi masalah.

getString digunakan untuk mengambil kolom dengan tiper data char, varchar atau tipe data string lainya. getInt digunakan untuk mengambil kolom dengan tipe data integer.

Berikut ini dalah contoh program lengkap dari melakukan koneksi hingga mengambil data dari database.

```
Class.forName("com.mysql.jdbc.Driver);
Connection conn =
      DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/latihan",
            "root","");
Statement statement =
      conn.createStatement(
            ResultSet.TYPE_SCROLL_SENSITIVE,
            ResultSet.CONCUR_READ_ONLY);
ResulSet rs =
      statement.executeQuery("select * from Customers");
while(rs.next()){
      System.out.println(rs.getInt("id"));
      System.out.println(rs.getString("Nama"));
}
```

Method executeQuery hanya dapat menjalankan perintah SQL select, gunakan method executeUpdate untuk menjalankan perintah insert, update dan delete. Hasil dari eksekusi insert, update dan delete tidak mengembalikan result set, tetapi mengembalikan sebuah nilai integer yang merepresentasikan status hasil eksekusi method executeUpdate.

Berikut ini contoh insert, update dan delete :

```
result = statement.executeUpdate(
      "update Customers set nama ='roby' where
      nama='andy'");
result = statement.executeUpdate(
      "delete Customers where nama='andy'");
```

Menggunakan executeQuery dan executeUpdate sangat mudah dan fleksible, namun sangat tidak efisien, PreparedStatement menawarkan keunggulan dalam bentuk efisiensi.

Menggunakan PreparedStatement

Memanggil method executeUpdate berulang-ulang, misalnya melakukan insert ratusan atau ribuan baris, sangat tidak efisien. Hal ini disebabkan karena DBMS harus memproses setiap query yang dikirimkan dalam beberapa langkah: memparsing query, mengcompile query dan kemudian baru mengeksekusi query tersebut.

PreparedStatement menawarkan solusi yang lebih baik dalam menangani keadaan tersebut. PreparedStatement menyaratkan query yang akan dieksekusi didefinisikan terlebih dahulu ketika PreparedStatement dibuat. Kemudian query tersebut dikirimkan ke dalam database untuk dicompile terlebih dahulu sebelum digunakan. Konsekuensinya, PreparedStatement bukan hanya mempunyai query, tetapi mempunyai query yang sudah dicompile. Ketika PreparedStatement dijalankan, DBMS tidak perlu melakukan kompilasi ulang terhadap query yang dijalankan PreparedStatement. Hal inilah yang menyebabkan PreparedStatement jauh lebih efisien dibandingkan menggunakan method Statement.executeUpdate.

Berikut ini contoh pembuatan PreparedStatement menggunakan class Connection yang telah dibuat sebelumnya :

```
PreparedStatement ps = conn.prepareStatement(
      “update T_PERSON set name = ? where name = ?”);
```

Perhatikan tanda ? yang ada dalam query diatas, tanda ? disebut sebagai parameter. Kita bisa memberikan nilai yang berbeda ke dalam parameter dalam setiap pemanggilan PreparedStatement.

Method setString, setFloat, setInt dan beberapa method lain

digunakan untuk memasukkan nilai dari setiap parameter. Method tersebut mempunyai dua parameter, parameter pertama adalah int yang digunakan untuk menentukan parameter PreparedStatement mana yang akan diberi nilai. Parameter kedua adalah nilai yang akan dimasukkan ke dalam PreparedStatement, tipe data dari parameter kedua tergantung dari method yang digunakan. Berdasarkan kode diatas, berikut ini contoh penggunaan method PreparedStatement.setString :

```
ps.setString(1,”andy”);
ps.setString(2,”rizal”);
```

Kode diatas memberikan contoh bagaimana memasukkan nilai ke dalam parameter PreparedStatement. Baris pertama memasukkan String “andy” ke dalam parameter pertama dan baris kedua memasukkan String “rizal” ke parameter kedua. Sehingga pemanggilan query oleh PreparedStatement berdasarkan kode diatas sama dengan query statement di bawah ini :

```
“update T_PERSON set name = 'andy' where name = 'rizal'”
```

Berikut ini contoh lengkap penggunaan PreparedStatement untuk melakukan update dan insert data :

```
PreparedStatement pInsert = conn.prepareStatement(
      "insert into Person(name) values(?)");
pInsert.setString(1,"dian");
pInsert.executeUpdate();
PreparedStatement pUpdate = conn.prepareStatement(
      "update Person set name=? where name=?");
pUpdate.setString(1,"andry");
pUpdate.setString(2,"andri");
pUpdate.executeUpdate();
```

Dalam contoh diatas, insert dan update data hanya dilaksanakan sekali saja, hal ini tidak memberikan gambaran yang tepat untuk melihat keunggulan PreparedStatement dibandingkan Statement.executeUpdate.

Batch Execution

Misalnya kita ingin meng-insert seratus baris data dalam sebuah loop, kita bisa menggunakan fasilitas batc execution dari PreparedStatement. batch execution mengumpulkan semua eksekusi program yang akan dilaksanakan, setelah semuanya terkumpul batch execution kemudian mengirimkan kumpulan

eksekusi program secara bersamaan ke DBMS dalam satu kesatuan. Metode ini sangat efisien karena mengurangi overhead yang diperlukan program untuk berkomunikasi dengan DBMS.

Dalam contoh di bawah ini kita akan menggunakan batch execution untuk melakukan insert data sebanyak seratus kali.

```
PreparedStatement pInsert = conn.prepareStatement(
      "insert into Person(nama) values(?)");
for(int i=0;i<100;i++){
      pInsert.setString(1,"user ke " + i);
      pInsert.addBatch();
}
pInsert.executeBatch();
```

Setiap kali iterasi, method setString dipanggil untuk mengisikan sebuah string ke dalam PreparedStatement, kemudian method addBatch dipanggil untuk mengumpulkan batch dalam satu wadah. Setelah iterasi selesai, method executeBatch dipanggil untuk melaksanakan semua keseratus instruksi insert secara berurut dengan sekali saja melaksanakan koneksi ke database.

Menangani Transaction

Dukungan transaction oleh JDBC tergantung dengan Databasenya, karena ada database yang mendukung transaction dan ada pula database yang tidak mendukung transaction. MySQL mendukung transaction jika kita menggunakan InnoDB sebagai sistem tablenya, kalau kita menggunakan MyISAM maka transaction tidak didukung.

Transaction merupakan konsep penting dari database. Transaction memastikan perubahan data dilaksanakan dengan kaidah ACID (Atomicity, Consistency, Isolation, Durability). Kaidah ini memastikan semua proses perubahan data berjalan secara benar, jika ada yang salah maka semua perubahan dalam satu kesatuan logika harus dibatalkan (rollback).

Mari kita evaluasi kode diatas agar menggunakan transaction, sehingga jika satu proses insert gagal, maka semua insert yang dilaksanakan sebelumnya akan dibatalkan :

```
try{
connection.setAutoCommit(false);
PreparedStatement pInsert = conn.prepareStatement(
      "insert into Person(nama) values(?)");
for(int i=0;i<100;i++){
      pInsert.setString(1,"user ke " + i);
      pInsert.addBatch();
}
pInsert.executeBatch();
connection.commit();
connection.setAutoCommit(true);
} catch (SQLException ex) {
      try{
            connection.rollback();
      }catch(SQLException e){
      }
}
```

DAO dan Service Pattern

Akses terhadap database merupakan bagian yang sangat penting dari aplikasi database. Penggunaan pattern yang sesuai dapat memberikan manfaat sangat besar. Pattern yang sering digunakan dalam akses database adalah DAO (Data Access Object) dan Service/Facade pattern.

Kedua pattern ini digunakan untuk menerapkan “separation of concern” atau pemisahan kode proram berdasarkan fungsi kode program. Semua kode untuk akses data harus dipisahkan dengan kode untuk pengaturan user inteface. Hal ini memungkinkan kode akses data yang dibuat untuk aplikasi desktop, dengan mudah digunakan untuk aplikasi web.

Penerapan konsep separation of concern secara disiplin, dapat menghasilkan kode program yang dapat dites secara otomatis menggunakan JUnit atau DBUnit. Unit testing merupakan paramater utama dalam menentukan apakah kode program yang kita hasilkan mempunyai mutu yang tinggi atau tidak. Coverage unit testing yang tinggi mencerminkan kode program yang berkualitas tinggi pula.

Dao pattern berisi semua kode untuk mengakses data, seperti query. Semua kode yang sepesifik terhadap implementasi akses data berhenti di sini, lapisan lebih atas tidak boleh tahu bagaimana

akses data diterapkan, apakah menggunakan JDBC murni atau Hibernate atau JPA. Lapisan lainya hanya perlu tahu fungsionalitas dari suatu method di dalam DAO class, tidak perlu tahu bagimana method tersebut diimplementasikan. Class DAO akan mempunyai method seperti save, delete, getById atau getAll. Praktek yang lazim digunakan adalah satu buah Entity/Table akan mempunyai satu buah class DAO.

Di bawah ini adalah contoh Class DAO menggunakan JDBC untuk table T_CATEGORY :

```
public class CategoryDaoJdbc {

    private Connection connection;
    private PreparedStatement insertStatement;
    private PreparedStatement updateStatement;
    private PreparedStatement deleteStatement;
    private PreparedStatement getByIdStatement;
    private PreparedStatement getAllStatement;
    private final String insertQuery =
      "insert into T_CATEGORY(name,description)" +
      " values(?,?)";
    private final String updateQuery =
      "update T_CATEGORY set name=?, description=? " +
      " where id=?";
    private final String deleteQuery =
      "delete from T_CATEGORY where id=?";
    private final String getByIdQuery =
      "select * from T_CATEGORY where id =?";
    private final String getAllQuery =
      "select * from T_CATEGORY";
    public void setConnection(Connection connection) {
        try {
            this.connection = connection;
      insertStatement =
        this.connection.prepareStatement(
      insertQuery,Statement.RETURN_GENERATED_KEYS);
      updateStatement =
        this.connection.prepareStatement(
        updateQuery);
      deleteStatement =
        this.connection.prepareStatement(
        deleteQuery);
      getByIdStatement =
        this.connection.prepareStatement(
        getByIdQuery);
      getAllStatement =
        this.connection.prepareStatement(
        getAllQuery);
        } catch (SQLException ex) {
        }
    }

    public void save(Category category) {
        try {
      if (category.getId() == null) {
        insertStatement.setString(1,
```

```
            category.getName());
        insertStatement.setString(2,
            category.getDescription());
        int id = insertStatement.executeUpdate();
            category.setId(id);
      } else {
        updateStatement.setString(1,
            category.getName());
        updateStatement.setString(2,
            category.getDescription());
        updateStatement.setInt(3, category.getId());
              updateStatement.executeUpdate();
      }
      } catch (SQLException ex) {
      }
    }

    public void delete(Category category) {
      try {
      deleteStatement.setInt(1, category.getId());
      deleteStatement.executeUpdate();
      } catch (SQLException ex) {
      }
    }

    public Category getById(Long id) {
      try {
      getByIdStatement.setLong(1, id);
      ResultSet rs = getByIdStatement.executeQuery();
      //proses mapping dari relational ke object
      if (rs.next()) {
        Category category = new Category();
        category.setId(rs.getInt("id"));
        category.setName(rs.getString("name"));
        category.setDescription(
            rs.getString("description"));
        return category;
      }
      } catch (SQLException ex) {
      }
      return null;
    }
    public List<Category> getAll() {
      try {
      List<Category> categories =
            new ArrayList<Category>();
```

```
      ResultSet rs = getAllStatement.executeQuery();
      while(rs.next()){
        Category category = new Category();
        category.setId(rs.getInt("id"));
        category.setName(rs.getString("name"));
        category.setDescription(
             rs.getString("description"));
        categories.add(category);
      }
      return categories;
      } catch (SQLException ex) {
      }
      return null;
    }
}
```

Service pattern digunakan utamanya untuk menyederhanakan class-class DAO yang ada, misalnya kita mempunyai 50 buah table maka lazimnya akan ada 50 buah class DAO. Class DAO tersebut perlu disederhanakan, caranya adalah dengan mengelompokkan class-class DAO dalam satu modul aplikasi ke class Service. Misalnya DAO yang berhubungan dengan user management ke dalam class UserService.

Transaction diatur dalam class Service, praktek yang lazim digunakan adalah satu method dalam class service adalah satu scoop transaction. Jadi ketika method dalam service mulai dieksekusi transaction akan dimulai (begin), ketika method akan berakhir, transaction akan dicommit.

Berikut ini adalah contoh class Service :

```
public class ServiceJdbc {

    private CategoryDaoJdbc categoryDao;
    private Connection connection;

    public void setDataSource(DataSource dataSource){
        try {
            connection = dataSource.getConnection();
            categoryDao = new CategoryDaoJdbc();
            categoryDao.setConnection(connection);
        } catch (SQLException ex) {
        }
    }

    public void save(Category category){
        try {
            connection.setAutoCommit(false);
            categoryDao.save(category);
            connection.commit();
            connection.setAutoCommit(true);
        } catch (SQLException ex) {
            try{
                connection.rollback();
            }catch(SQLException e){
            }
        }
    }
    public void delete(Category category){
        try {
            connection.setAutoCommit(false);
            categoryDao.save(category);
            connection.commit();
            connection.setAutoCommit(true);
        } catch (SQLException ex) {
            try{
                connection.rollback();
            }catch(SQLException e){
            }
        }
   }

    public Category getGroup(Long id){
        return categoryDao.getById(id);
    }

    public List<Category> getGroups(){
```

```
        return categoryDao.getAll();
    }
}
```

Setelah class DAO dan service berhasil dibuat, mari kita lihat bagaimana cara menggunakannya :

```
public class MainJdbc {

    public static void main(String[] args) {

        MysqlDataSource dataSource =
             new MysqlDataSource();
        dataSource.setUser("root");
        dataSource.setPassword("admin");
        dataSource.setDatabaseName("latihan");
        dataSource.setServerName("localhost");
        dataSource.setPortNumber(1527);

        ServiceJdbc service = new ServiceJdbc();
        service.setDataSource(dataSource);

        Category category = new Category();
        category.setName("administrator");
        service.save(category);

        System.out.println("id : " + category.getId());
        System.out.println("name : " +
             category.getName());

        try {
            dataSource.getConnection().close();
        } catch (SQLException ex) {
        }
    }

}
```

Setelah mengenal JDBC, kita akan membahas cara mengakses database yang lebih baik dengan manggunakan JPA. Dengan menggunakan JPA kode program yang akan dibuat akan lebih ringkas dan terlihat konsep OOP dibanding dengan JDBC murni. JPA adalah framework ORM untuk memetakan tabel dalam database dan class dalam konsep OOP. Dengan menggunakan JPA kode program kita akan lebih rapi dan terlihat OOP.