

## BAB 9 JDBC

### 9.1 Tujuan Pembelajaran

Mahasiswa dapat mengimplementasikan JDBC pada project yang sudah dibuat sebelumnya

### 9.2 Dasar Teori

#### 9.2.1 Connection

Connection adalah sesi antara aplikasi Java dan database. Ini membantu untuk membuat koneksi dengan database. Antarmuka *connection* adalah kumpulan pernyataan seperti: *PreparedStatement*, dan *DatabaseMetaData*, objek Connection dapat digunakan untuk mendapatkan objek Statement dan DatabaseMetaData. Antarmuka Connection menyediakan banyak metode untuk manajemen transaksi seperti *commit()*, *rollback()*, *setAutoCommit()*, *setTransactionIsolation()*, dll.

Metode antarmuka Koneksi yang umum digunakan:

1. **createStatement()**: membuat objek pernyataan yang dapat digunakan untuk mengeksekusi query SQL.
2. **createStatement(int resultSetType,int resultSet Concurrency)**: Membuat objek Pernyataan yang akan menghasilkan objek ResultSet dengan tipe dan konkurensi yang diberikan.
3. **setAutoCommit(status boolean)**: digunakan untuk mengatur status komit. Secara default, itu benar.
4. **commit()**: menyimpan perubahan yang dibuat sejak commit/rollback sebelumnya bersifat permanen.
5. **rollback()**: Menjatuhkan semua perubahan yang dilakukan sejak commit/rollback sebelumnya.
6. **close()**: menutup koneksi dan segera merilis resource JDBC.

Contoh implementasi file connection:

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

public class ConnectionDB {

    private Connection conn;

    public Connection connect() throws
SQLException { String host = "localhost:3306";

        String
dbName = "pbo_db"; String
dbuser = "root"; String
dbpassword = "";

        try {

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

        } catch
(ClassNotFoundException e) {
e.printStackTrace();
        }

        Connection conn =
DriverManager.getConnection("jdbc:mysql://" + host + "/" + dbName,
                                dbuser,
```

Tambahkan dependencies library MySQL pada project maven anda:

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

    <groupId>mysql</groupId>

    <artifactId>mysql-connector-java</artifactId>
```

Pada terminal project anda install library tersebut dengan memasukan syntax  
Mvn clean install package

## 9.2.2 Prepared Statement

Antarmuka Prepared Statement adalah sub antarmuka dari Pernyataan. Ini digunakan untuk mengeksekusi query berparameter. Berikut adalah contoh query berparameter:

Simbol tanda tanya mengartikan bentuk pengiriman parameter (?) untuk nilainya. Nilainya akan ditetapkan dengan memanggil metode penyetel dari

*PreparedStatement*.

**Mengapa menggunakan *PreparedStatement*:**

Meningkatkan performa: Performa aplikasi akan lebih cepat jika Anda menggunakan interface *PreparedStatement* karena query hanya dikompilasi satu kali.

Metode antarmuka *PreparedStatement*:

<b>Meth od</b>	<b>Descripti on</b>
setInt(int paramIndex, int value)	menetapkan nilai integer ke indeks parameter yang diberikan.
setString(int paramIndex, String value)	menetapkan nilai string ke indeks parameter yang diberikan.
setFloat(int paramIndex, float value)	menetapkan nilai float ke indeks parameter yang diberikan.
setDouble(int paramIndex, double value)	menetapkan nilai double ke indeks parameter yang diberikan.
executeUpdate()	mengeksekusi query. Ini digunakan untuk membuat, menjatuhkan, menyisipkan, memperbarui, menghapus, dll.
ResultSet executeQuery()	mengeksekusi query pemilihan. Ini mengembalikan instance dari ResultSet.

### 9.2.3 ResultSet

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 resultset dibedakan berdasarkan dua area:

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

Metode antarmuka ResultSet yang umum digunakan:


<b>Meth od</b>	<b>Deskri psi</b>
next():	digunakan untuk memindahkan index ke satu baris berikutnya dari posisi saat ini.
previous():	digunakan untuk memindahkan index ke satu baris sebelumnya dari posisi saat ini.
first():	digunakan untuk memindahkan index ke baris pertama dalam objek set hasil.
last():	digunakan untuk memindahkan index ke baris terakhir dalam objek set hasil.
absolute(int row):	digunakan untuk memindahkan index ke nomor baris yang ditentukan dalam objek ResultSet.
relative(int row):	digunakan untuk memindahkan index ke nomor baris relatif pada objek ResultSet, bisa positif atau negatif.
getInt(int columnIndex):	digunakan untuk mengembalikan data indeks kolom tertentu dari baris saat ini sebagai int.
getInt(String columnName):	digunakan untuk mengembalikan data nama kolom tertentu dari baris saat ini sebagai int.
getString(int columnIndex):	digunakan untuk mengembalikan data indeks kolom tertentu dari baris saat ini sebagai String.
getString(String columnName):	digunakan untuk mengembalikan data nama kolom tertentu dari baris saat ini sebagai String.

## 9.2.4 Implementasi dari PreparedStatement dan ResultSet

### 9.2.4.1 Setting Up

Buatlah database dengan nama `pbo_db` dimana database tersebut memiliki table bernama `students`. Berikut ini adalah contoh desain table students:

Buatlah 1 file class DTO bernama Students, class encapsulation

Name	Type	Length	Decimals	Not Null	Virtual	Key
id	int	11		<input checked="" type="checkbox"/>	<input type="checkbox"/>	 1
npm	varchar	15		<input type="checkbox"/>	<input type="checkbox"/>	
firstname	varchar	20		<input type="checkbox"/>	<input type="checkbox"/>	
middlename	varchar	20		<input type="checkbox"/>	<input type="checkbox"/>	
lastname	varchar	20		<input type="checkbox"/>	<input type="checkbox"/>	
program_id	int	11		<input checked="" type="checkbox"/>	<input type="checkbox"/>	
program_study_i	int	11		<input type="checkbox"/>	<input type="checkbox"/>	
email	varchar	50		<input type="checkbox"/>	<input type="checkbox"/>	
birthdate	date			<input type="checkbox"/>	<input type="checkbox"/>	

ini berisi attribute dan setter dan getter berdasarkan field dari table students. Contoh sebagai berikut:

```
public class Students {

    private Integer
    id; private String npm;
    private String firstname;
    private String middlename;
    private String lastname;
    private String email;
    private String birthdate;


    public String
    getNpm() {

        return npm;
    }

    public void setNpm(String npm) {

        this.npm = npm;
    }

    public String getFirstname() {

        return firstname;
    }

    public void setFirstname(String firstname) {
```

```
}

    public String getMiddlename() {

        return middlename;

    }

    public void setMiddlename(String middlename) {

        this.middlename = middlename;

    }

    public String getLastname() {

        return lastname;

    }

    public void setLastname(String lastname) {

        this.lastname = lastname;

    }

    public String getEmail() {

        return email;

    }

    public void setEmail(String email) {

        this.email = email;

    }

    public String getBirthdate() {

        return birthdate;

    }

    public void setBirthdate(String birthdate) {

        this.birthdate = birthdate;

    }

    public Integer getId() {

        return id;

    }

    public void setId(Integer id) {

        this.id = id;

    }

}
```

Buatlah 1 file class DAO bernama Students Dao, class repository ini berisi query yang akan dieksekusi dan dioperasikan dalam bentuk method.

Sebagai contoh:

```
import java.sql.Connection;

import java.sql.PreparedStatement;

import
java.sql.ResultSet;
import
java.sql.SQLException;
import
java.util.ArrayList;
import java.util.List;

public class StudentsDao {

    //method-method berisi CRUD
}
```

#### 9.2.4.2 Insert Data

Terdapat sebuah form isian untuk mengisi table students seperti dibawah ini:



Pada button Register diberikan aksi validator dan memasukan semua isian form diatas kedalam database. Berikut adalah setingan untuk membuat insert data dengan JDBC:

1. Pada class DAO milik StudentsDao, buatlah method bernama saved() dan attribute bernama queryInsert.



```

public class StudentsDao {

    private String queryInsert = "insert into students (npm,firstname,
middlename, lastname, email, birthdate) " + "values (?, ?, ?, ?, ?, ?)";

    public void saved(Students students) throws Exception {

        Connection c = new ConnectionDB().connect();

        PreparedStatement psInsert = c.prepareStatement(queryInsert);

        psInsert.setString(1, students.getNpm());
        psInsert.setString(2, students.getFirstname());
        psInsert.setString(3, students.getMiddlename());
        psInsert.setString(4, students.getLastname());
        psInsert.setString(5, students.getEmail());
        psInsert.setString(6, students.getBirthdate());

        psInsert.execute

        Update(): c.close();
    }
}

```

2. Pada view swing button register berikan aksi ActionListener agar dapat mengelola logical validasi dan insert data kedalam database.

```

...

JButton btnNewButton_1 = new JButton("Register");
btnNewButton_1.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent
e) { ValidatorForm();

    }

});

```

```

private void ValidatorForm() {

    if(...statement check if field is empty...){

        if(!(...statement if password and repassword is
match...)) { JOptionPane.showMessageDialog(null,
"Password not matched");

        }else {

            try {

                Insert2database
                ();
                JOptionPane.sho
wMessageDialog(
null,
"Successfully
saved");

            } catch (Exception e) {
                JOptionPane.showMe
ssageDialog(null,
"Failed save.
Error:" + e.getMessa
ge());
                e.printStackTrace(
);

            }

        }

    }else{

        Object message = "Please fill up the form with
correctly"; JOptionPane.showMessageDialog(null,
message);

    }

}

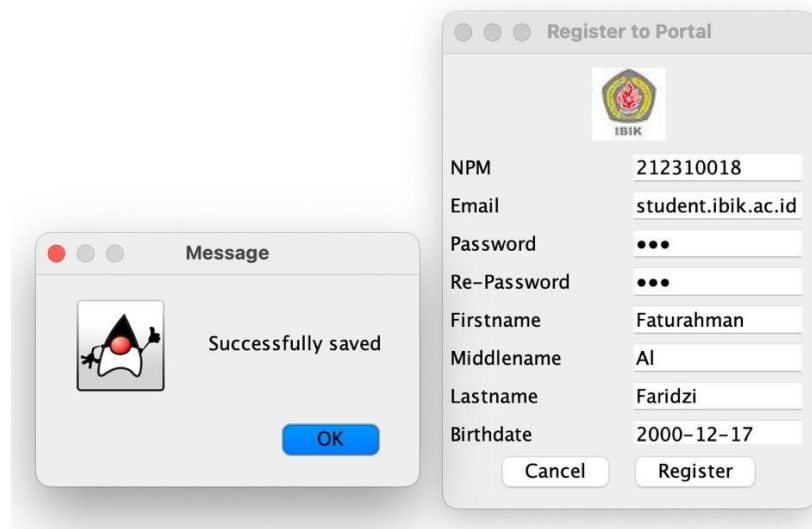
private void Insert2database()
throws Exception {
    Students students = new
Students();

    students.setNpm(textNpm.getText());
    students.setFirstname(textFirstname.getText());
    students.setMiddlename(textMiddlename.getText());
    students.setLastname(textLastname.getText());
    students.setEmail(textEmail.getText());
    students.setBirthdate(textBirthdate.getText());

    StudentsDao stdDao
= new
StudentsDao();
    stdDao.saved(studen
ts);
}

```

Maka output dari program tersebut akan seperti berikut:



#### 9.2.4.3 Select Data

Berikut ini adalah contoh melakukan retrieve data dengan query select all tanpa ada kondisi tertentu (tanpa menggunakan PreparedStatement). Dan hasil data akan di retrieve ke dalam JTable. Pada class Students DO buatlah method bernama findAll() dan attribute querySelectorAll.

```

public class StudentsDao {

    ...

    private String querySelectAll = "select * from students";

    ...

    public List<Students> findAll() throws Exception
    { List<Students> hasil = new ArrayList<Students>();

        Connection c = new
        ConnectionDB().connect(); PreparedStatement
        psCariSemuaProduk =
        c.prepareStatement(querySelectAll);

        ResultSet rs = psCariSemuaProduk.executeQuery();

        while(rs.next()){

            Students students =
            konversiResultSet(rs);
            hasil.add(students);

        }

        c.close();

        return hasil;

    }

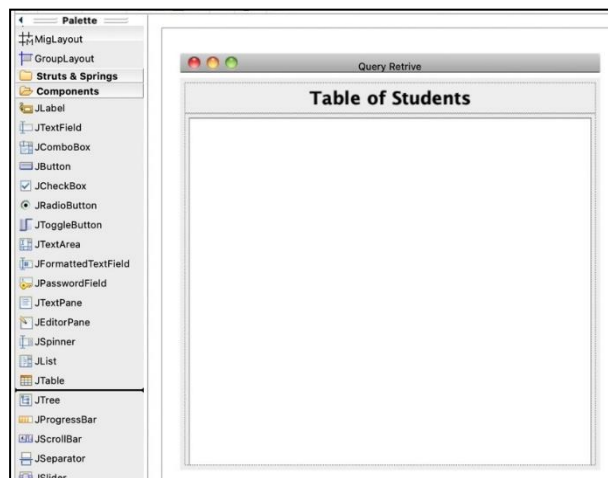
    private Students konversiResultSet(ResultSet rs) throws
    SQLException{ Students students = new Students();
    students.setId(rs.getInt("id"));
    students.setNpm(rs.getString("npm"));
    students.setFirstname(rs.getString("firstname"));

        students.setMiddlename(rs.getString("middlena
    me")); students.setLastname(rs.getString("lastname"));
    students.setEmail(rs.getString("email"));
    students.setBirthdate(rs.getString("birthdate"));
    return students;

    }

```

Pada UI Swing untuk menampung data retrieve akan membentuk sebagai berikut:



```
import
java.awt.EventQueue; import
javax.swing.JFrame; import
javax.swing.JPanel;

import javax.swing.border.EmptyBorder;
import javax.swing.table.DefaultTableModel;
import java.awt.BorderLayout;

import javax.swing.JLabel;

import javax.swing.SwingConstants;

import java.awt.Font;

import java.sql.SQLException;

import java.util.List;

import javax.swing.JTable;

import javax.swing.JScrollPane;

public class AppRetrieveData extends JFrame {

    private JPanel contentPane;

    private JTable jt;

    /**
     * Launch the application.
     */

    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {

            public void run() {

                try {
```

```

AppRetrieveData frame =
new AppRetrieveData(); frame.setVisible(true);

    }

    catch (Exception e)
    {
        e.printStackTrace();
    }

    }

});

}

public
AppRetrieveData () {
    setTitle("Query Retrieve");

        setDefaultCloseOperation(JFrame.EXIT
ON_CLOSE); setBounds(100, 100, 474, 471);

        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

        contentPane.setLayout(new
BorderLayout(0, 0));
        setContentPane(contentPane);

        JPanel panel = new JPanel();
        contentPane.add(panel, BorderLayout.NORTH);

        JLabel lblNewLabel = new JLabel("Table of
Students"); panel.add(lblNewLabel);

        JPanel panel_1 = new JPanel();
        contentPane.add(panel_1, BorderLayout.CENTER);

        DefaultTableModel tableModel =
FetchDataSelect(); jt=new JTable(tableModel);

        jt.addMouseListener
(new MouseAdapter() { @Override

            public void
mouseClicked(MouseEvent e) {
                GetSelectedData();
            }

        });

        JScrollPane sp=new
JScrollPane(jt); panel_1.add(sp);
    }

```

```

tableModel.addRow(data);

    }

    }catch (Exception e) {
OptionPane.showMessageDialog(null, "Failed
load data. Error:"+e.getMessage());
    }

    return tableModel;
}

protected void GetSelectedData() {
    DefaultTableModel jtModel = (DefaultTableModel)
jt.getModel();

    int selectedRow = jt.getSelectedRow();
System.out.println("You just click row "+selectedRow);

    String npm = jtModel.getValueAt(selectedRow,

                                0).toString();

                                String name =
                                jtModel.getValueAt(selectedRow,
                                1).toString();

```

Output dari program diatas sebagai berikut:

The screenshot shows a Java application window titled "Query Retrive" with a table titled "Table of Students". The table has four columns: NPM, Fullname, Email, and Birthdate. The table contains 24 rows of student data. The 24th row (index 23) is highlighted in blue, showing NPM: 212310018, Fullname: Faturahman Al Faridzi, Email: 212310018@student.ibik.ac.id, and Birthdate: 2000-12-17.

Below the table, a terminal window shows the output of the application. It displays the message "You just click row 24" and the data for the selected row:

```

[INFO] --- exec-maven-plugin:3.0.0:java (default-cli) @ pbo-project-maven ---
Click row
You just click row 24
Data:
NPM: 212310018
Fullname: Faturahman Al Faridzi
Email: 212310018@student.ibik.ac.id
Birthdate: 2000-12-17

```

#### 9.2.4.4 Update Data

Untuk melakukan update data sama halnya dengan membuat method dan attribute query seperti Insert Data (poin IV.B). Yang berbeda hanyalah isian dari value attribute berisi query update pada class Student DAO.

```
public class StudentsDao {  
  
    ...  
  
    private String queryUpdate = "update students set npm=?, firstname=?,  
    middlename=?, lastname=?, email=?, birthdate=? " + "where id = ?";  
  
    ...  
  
    public void updated(Students students) throws  
    Exception { Connection c = new  
    ConnectionDB().connect();  
  
        PreparedStatement psUpdate =  
    c.prepareStatement(queryUpdate); psUpdate.setString(1,  
    students.getNpm()); psUpdate.setString(2,  
    students.getFirstname()); psUpdate.setString(3,  
    students.getMiddlename()); psUpdate.setString(4,  
    students.getLastname()); psUpdate.setString(5,  
    students.getEmail()); psUpdate.setString(6,  
    students.getBirthdate());  
  
        psUpdate.setInt(7, students.getId());  
  
        psUpdate.executeUpdate();  
  
        c.close();  
  
    }  
  
    ...  
}
```



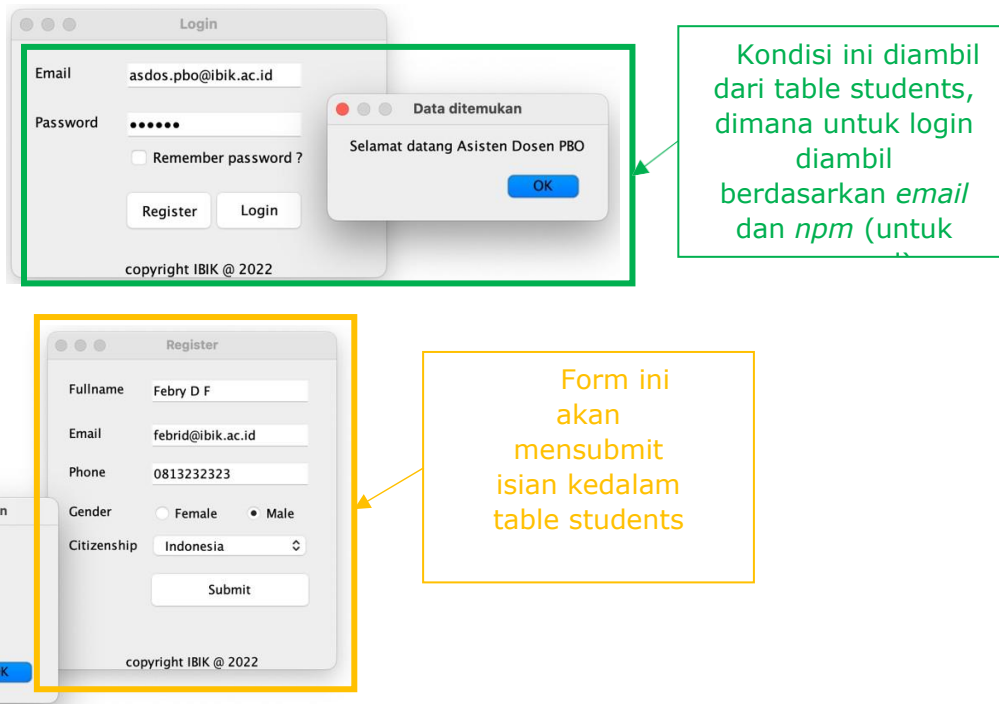
#### 9.2.4.5 Delete Data

Sedangkan untuk melakukan penghapusan data membutuhkan kondisi statement pada querynya. Oleh karenanya dapat menggunakan PreparedStatement untuk memberikan statement kondisi query. Contoh pada class Students DAO:

```
public class StudentsDao {  
  
    ...  
  
    private String queryRemoveById = "delete from students where id = ?";  
  
    ...  
  
    public void removed(Integer id) throws Exception{  
  
        if(i  
        d == null){  
            return;  
        }  
  
        Connection c = new ConnectionDB().connect();  
  
        PreparedStatement psHapusById =  
        c.prepareStatement(queryRemoveById); psHapusById.setInt(1, id);  
  
        psHapusById.execute  
        Update(); c.close();  
    }  
  
    ...  
  
}
```

### 9.3 Latihan Pembelajaran

1. Buatlah project maven dengan nama PBO-NPM-Pembelajaran-9 dan package groupid bernama com.ibik.pbo.Pembelajaran.
2. Berdasarkan UI pada soal Pembelajaran-8 soal nomor 2 buatlah flow aplikasi sebagai berikut dengan JDBC:



3. Pada form dibawah ini buatlah **Create Retrieve Update Delete** dimana seluruh data dibawah ini disimpan di dalam table student\_score. Silahkan anda rancang skema database yang baik jika memiliki rancangan seperti dibawah ini:

The image shows a screenshot of a Java Swing application window titled 'Latihan 05'. The window has a menu bar with 'File', 'Edit', and 'Help'. Below the menu bar is a toolbar with 'Open Student Data' and 'Exit' buttons. The main area is titled 'FORM PENILAIAN MATAKULIAH PBO'. It contains three input fields: 'NPM', 'Nama', and 'Nilai'. Below the 'Nilai' field are radio buttons for 'A', 'B', 'C', 'D', 'E', and 'F'. At the bottom are three buttons: 'Save', 'Delete', and 'Clear'. On the right side of the form, there is a table with the following data:

NPM	Nama	Nilai
212310018	FATHURAHMAN AL FAR...	B
212310019	MUHAMMAD RAFI ZUH...	C
212310020	MUHAMMAD SUBHAN RIZ...	A

Pada gambar diatas memiliki beberapa kondisi tertentu, seperti:

- Jika mengklik tombol save dimana keadaan formnya masih kosong, maka

akan melakukan logical validasi dan insert ke dalam database student score.

- Jika mengklik tombol save dimana keadaan formnya telah terisi (value diambil dari aksi jTable), maka akan melakukan logical update data kedalam database student score

Jika mengklik tombol delete, namun belum ada data yang terpilih (data dari jTable) maka akan menampilkan pesan error “Silakan pilih data terlebih dahulu”, namun jika sudah terpilih maka akan melakukan logical delete data ke dalam database berdasarkan item yang terpilih.