


WEEK 7: GUIDELINES

JDBC & PostgreSQL Database Integration

Week: 7 of 10 | **Topics:** JDBC, PostgreSQL, Database Connection, CRUD Operations | **Practice:** Connect to Database


 *NOTE: This is practice only - no assignment defense this week!*



LEARNING OBJECTIVES

- Understand what JDBC is and why we need it
- Install and setup PostgreSQL database
- Connect Java application to PostgreSQL
- Create database table matching your entity
- Insert data into database (CREATE)
- Read data from database (READ)
- Understand PreparedStatement vs Statement
- Handle database exceptions properly

TECHNICAL REQUIREMENTS

 **MINIMUM REQUIREMENTS:** By the end of this week, you **MUST** have these working:

| Requirement | What You Must Do |
|------------------------------|---|
| 1. PostgreSQL Installed | <ul style="list-style-type: none">✓ Download and install PostgreSQL✓ Remember your postgres user password✓ Can open pgAdmin or psql terminal |
| 2. Database Created | <ul style="list-style-type: none">✓ Create database with your project name✓ Example: restaurant_db, hospital_db, gym_db, etc.✓ Database is accessible |
| 3. One Table Created | <ul style="list-style-type: none">✓ Create ONE table for your main entity✓ Example: staff, patient, member, product, etc.✓ Table has PRIMARY KEY✓ Columns match your Java class fields |
| 4. JDBC Driver Added | <ul style="list-style-type: none">✓ Download PostgreSQL JDBC driver (JAR file)✓ Add to project classpath✓ Can import org.postgresql.Driver |
| 5. Database Connection Works | <ul style="list-style-type: none">✓ Can connect from Java to PostgreSQL✓ No connection errors✓ Connection closes properly |
| 6. INSERT Works | <ul style="list-style-type: none">✓ Can insert at least 1 record from Java✓ Uses PreparedStatement✓ Record appears in database |
| 7. SELECT Works | <ul style="list-style-type: none">✓ Can read records from database✓ Display in console✓ ResultSet processed correctly |



WHAT IS JDBC?

JDBC = Java Database Connectivity

JDBC is a Java API that allows Java programs to connect to databases and execute SQL queries.

Why Do We Need JDBC?

- Store data permanently (not lost when program closes)
- Handle large amounts of data efficiently
- Share data between multiple applications
- Perform complex queries and searches
- Real-world applications use databases

JDBC Architecture:

Java Application



JDBC API



JDBC Driver (postgresql-XX.jar)



PostgreSQL Database



STEP 1: INSTALL POSTGRESQL

Download:

1. Go to: <https://www.postgresql.org/download/>
2. Select your operating system (Windows/Mac/Linux)
3. Download PostgreSQL installer
4. Run installer

During Installation:

- ✓ Remember the password you set for "postgres" user
- ✓ Default port: 5432 (keep this)
- ✓ Install pgAdmin (included with PostgreSQL)
- ✓ Complete installation



CRITICAL: WRITE DOWN YOUR PASSWORD! You will need it to connect from Java.

STEP 2: CREATE DATABASE

Method 1: Using pgAdmin (GUI)

5. Open pgAdmin
6. Connect to PostgreSQL (enter your password)
7. Right-click on "Databases"
8. Select "Create" → "Database"
9. Enter database name (example: restaurant_db)
10. Click "Save"

Method 2: Using SQL (psql)

```
-- Open psql terminal
-- Then run:
CREATE DATABASE restaurant_db;

-- Verify:
\l
```

Database Naming Examples:

- Restaurant project: restaurant_db
- Hospital project: hospital_db
- Gym project: gym_db
- Vet Clinic project: vet_clinic_db
- Grocery Store project: grocery_db
- Clothing Store project: clothing_db



STEP 3: CREATE TABLE

Choose Your Main Entity:

- Restaurant: staff, menu_item, or customer
- Hospital: patient, doctor, or appointment
- Gym: member, trainer, or workout
- Vet: animal, owner, or treatment
- Grocery: product, employee, or order
- Clothing: clothing_item, customer, or sale

Example: Restaurant Staff Table

If your Java class looks like this:

```
public class Staff {  
    private int staffId;  
    private String name;  
    private double salary;  
    private int experienceYears;  
}
```

Your SQL table should be:

```
CREATE TABLE staff (  
    staff_id SERIAL PRIMARY KEY,  
    name VARCHAR(100) NOT NULL,  
    salary DECIMAL(10, 2) NOT NULL,  
    experience_years INTEGER NOT NULL  
);
```

JAVA ↔ POSTGRESQL DATA TYPE MAPPING

| Java Type | PostgreSQL Type | Example |
|----------------|-------------------|----------------------|
| int | INTEGER or SERIAL | staff_id SERIAL |
| String | VARCHAR(n) | name VARCHAR(100) |
| double | DECIMAL(p,s) | salary DECIMAL(10,2) |
| boolean | BOOLEAN | is_active BOOLEAN |
| Date | DATE or TIMESTAMP | birth_date DATE |

Important Notes:

- SERIAL = auto-incrementing integer (like ID)
- VARCHAR(n) = variable length string, max n characters
- DECIMAL(10,2) = decimal number, 10 total digits, 2 after decimal point
- NOT NULL = field cannot be empty
- PRIMARY KEY = unique identifier for each row

MORE TABLE EXAMPLES

Hospital - Patient Table:

```
CREATE TABLE patient (  
    patient_id SERIAL PRIMARY KEY,  
    name VARCHAR(100) NOT NULL,  
    age INTEGER NOT NULL,  
    diagnosis VARCHAR(200),  
    admission_date DATE NOT NULL  
);
```

Gym - Member Table:

```
CREATE TABLE member (  
    member_id SERIAL PRIMARY KEY,  
    name VARCHAR(100) NOT NULL,  
    membership_type VARCHAR(50) NOT NULL,  
    monthly_fee DECIMAL(10,2) NOT NULL,  
    join_date DATE NOT NULL  
);
```

Vet Clinic - Animal Table:

```
CREATE TABLE animal (  
    animal_id SERIAL PRIMARY KEY,  
    name VARCHAR(100) NOT NULL,  
    species VARCHAR(50) NOT NULL,  
    age INTEGER,  
    owner_name VARCHAR(100) NOT NULL  
);
```

STEP 4: ADD JDBC DRIVER

Download JDBC Driver:

11. Go to: <https://jdbc.postgresql.org/download/>
12. Download the .jar file (e.g., postgresql-42.7.1.jar)
13. Save it to your computer

Add to IntelliJ IDEA:

14. Open your project in IntelliJ
15. File — Project Structure (Ctrl+Alt+Shift+S)
16. Select "Libraries"
17. Click "+" — "Java"
18. Navigate to downloaded postgresql-XX.jar file
19. Click "OK"
20. Apply and Close

Verify: Try importing:

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

If no errors - driver is added correctly!

STEP 5: CREATE DATABASE CONNECTION CLASS

Create a new package: database

Create file: DatabaseConnection.java

```
package database;

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

public class DatabaseConnection {
    private static final String URL = "jdbc:postgresql://localhost:5432/restaurant_db";
    private static final String USER = "postgres";
    private static final String PASSWORD = "your_password_here";

    public static Connection getConnection() {
        Connection connection = null;
        try {
            connection = DriverManager.getConnection(URL, USER, PASSWORD);
            System.out.println("✅ Connected to database successfully!");
        } catch (SQLException e) {
            System.out.println("❌ Connection failed!");
            e.printStackTrace();
        }
        return connection;
    }

    public static void closeConnection(Connection connection) {
        if (connection != null) {
            try {
                connection.close();
                System.out.println("Connection closed.");
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

IMPORTANT: Change These Values!

- **URL:** Replace "restaurant_db" with YOUR database name
- **USER:** Usually "postgres" (default user)
- **PASSWORD:** Replace with YOUR postgres password

STEP 6: TEST DATABASE CONNECTION

Create a simple test to verify connection works:

```
public class TestConnection {  
    public static void main(String[] args) {  
        Connection connection = DatabaseConnection.getConnection();  
  
        if (connection != null) {  
            System.out.println("Connection test passed! ✓");  
            DatabaseConnection.closeConnection(connection);  
        } else {  
            System.out.println("Connection test failed! ✗");  
        }  
    }  
}
```

Expected Output:

```
✓ Connected to database successfully!  
Connection test passed! ✓  
Connection closed.
```

Common Connection Errors:

ERROR: password authentication failed

Solution: Wrong password in DatabaseConnection.java

ERROR: database "xxx" does not exist

Solution: Database name in URL is wrong or database not created

ERROR: No suitable driver found

Solution: JDBC driver not added to project

ERROR: Connection refused

Solution: PostgreSQL is not running

STEP 7: INSERT DATA (CREATE)

Create StaffDAO.java (Data Access Object):

```
package database;

import model.Staff;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class StaffDAO {

    public void insertStaff(Staff staff) {
        String sql = "INSERT INTO staff (name, salary, experience_years) VALUES (?, ?, ?)";

        Connection connection = DatabaseConnection.getConnection();

        try {
            PreparedStatement statement = connection.prepareStatement(sql);

            // Set parameters (? → actual values)
            statement.setString(1, staff.getName());
            statement.setDouble(2, staff.getSalary());
            statement.setInt(3, staff.getExperienceYears());

            // Execute INSERT
            int rowsInserted = statement.executeUpdate();

            if (rowsInserted > 0) {
                System.out.println("✅ Staff inserted successfully!");
            }

            statement.close();
        } catch (SQLException e) {
            System.out.println("❌ Insert failed!");
            e.printStackTrace();
        } finally {
            DatabaseConnection.closeConnection(connection);
        }
    }
}
```

KEY CONCEPTS IN INSERT

1. PreparedStatement

PreparedStatement is BETTER than Statement because:

- ✓ Prevents SQL injection attacks
- ✓ Faster for repeated queries
- ✓ Cleaner code

2. The ? Placeholder

The ? in SQL is a placeholder:

```
String sql = "INSERT INTO staff (name, salary) VALUES (?, ?)";
                                                    ↑      ↑
                                                    index 1 index 2

statement.setString(1, "Aibek");    // Replace first ?
statement.setDouble(2, 500000);    // Replace second ?
```

3. executeUpdate()

executeUpdate() returns number of rows affected:

- 1 = one row inserted
- 0 = no rows inserted (something wrong)

TEST INSERT

```
public class TestInsert {
    public static void main(String[] args) {
        // Create staff object
        Staff staff = new Staff(0, "Aibek", 450000, 3);

        // Insert into database
        StaffDAO dao = new StaffDAO();
        dao.insertStaff(staff);

        // Check in pgAdmin or psql:
        // SELECT * FROM staff;
    }
}
```

Verify in PostgreSQL:

```
-- Open pgAdmin or psql
-- Run:
SELECT * FROM staff;

-- You should see your inserted record!
```

STEP 8: READ DATA (SELECT)

Add method to StaffDAO.java:

```
public void getAllStaff() {
    String sql = "SELECT * FROM staff";

    Connection connection = DatabaseConnection.getConnection();

    try {
        PreparedStatement statement = connection.prepareStatement(sql);
        ResultSet resultSet = statement.executeQuery();

        System.out.println("\n--- ALL STAFF FROM DATABASE ---");

        while (resultSet.next()) {
            int id = resultSet.getInt("staff_id");
            String name = resultSet.getString("name");
            double salary = resultSet.getDouble("salary");
            int experience = resultSet.getInt("experience_years");

            System.out.println("ID: " + id);
            System.out.println("Name: " + name);
            System.out.println("Salary: " + salary);
            System.out.println("Experience: " + experience + " years");
            System.out.println("---");
        }

        resultSet.close();
        statement.close();

    } catch (SQLException e) {
        System.out.println("❌ Select failed!");
        e.printStackTrace();
    } finally {
        DatabaseConnection.closeConnection(connection);
    }
}
```



UNDERSTANDING RESULTSET

ResultSet is like a cursor that points to rows in result:

Database table:

| staff_id | name | salary | experience_years |
|----------|-------|--------|------------------|
| 1 | Aibek | 450000 | 3 |
| 2 | Murat | 600000 | 12 |
| 3 | Dana | 300000 | 4 |

← resultSet.next()

← resultSet.next()

← resultSet.next()

```
while (resultSet.next()) { // Move to next row
    int id = resultSet.getInt("staff_id"); // Get column value
    String name = resultSet.getString("name");
    // ...
}
```

Key Methods:

- resultSet.next() - Move to next row, returns true if row exists
- resultSet.getInt("column_name") - Get integer value from column
- resultSet.getString("column_name") - Get string value
- resultSet.getDouble("column_name") - Get double value



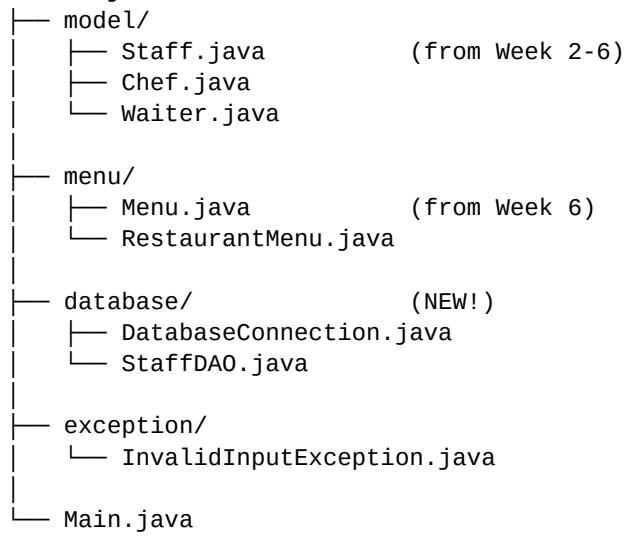
TEST SELECT

```
public class TestSelect {  
    public static void main(String[] args) {  
        StaffDAO dao = new StaffDAO();  
        dao.getAllStaff();  
    }  
}
```

```
// Expected output:  
// --- ALL STAFF FROM DATABASE ---  
// ID: 1  
// Name: Aibek  
// Salary: 450000.0  
// Experience: 3 years  
// ---
```

FINAL PROJECT STRUCTURE

YourProject/



WEEK 7 PRACTICE CHECKLIST

- ☐ PostgreSQL installed
- ☐ Database created
- ☐ One table created matching your entity
- ☐ JDBC driver added to project
- ☐ DatabaseConnection class created
- ☐ Connection test successful
- ☐ DAO class created
- ☐ INSERT method works (can add record)
- ☐ SELECT method works (can read records)
- ☐ PreparedStatement used (not Statement)
- ☐ try-catch blocks for all database operations
- ☐ Connections closed properly

COMMON ISSUES & SOLUTIONS

Cannot connect to database

Check:

- PostgreSQL is running
- Password is correct
- Database name is correct
- Port is 5432

ClassNotFoundException: org.postgresql.Driver

JDBC driver not added to project. Add the JAR file again.

Column "xxx" does not exist

Column name in SQL doesn't match table. Check spelling and case.

NULL values in database

Check if you're using `setString`/`setInt` correctly with right index numbers.

ResultSet is empty

No data in table. Insert some records first, then `SELECT`.

Connection leak

Always close connections in `finally` block!



TIPS FOR SUCCESS

- Start with database setup FIRST - don't wait until last minute
- Test connection before writing any DAO code
- Insert one record, then SELECT it to verify
- Use PreparedStatement, NEVER Statement
- Always close connections in finally block
- Print SQL queries to console for debugging
- Check pgAdmin after every INSERT to verify data
- Keep your postgres password safe



NEXT WEEK (Week 8)

Week 8 Topics:

- UPDATE operation
- DELETE operation
- Complete CRUD
- SOLID Principles
- Assignment 4 issued

Master database basics this week!