Lecture 1: JDBC Introduction

What is JDBC?

JDBC (Java Database Connectivity) is a Java API that enables Java applications to interact with databases. It provides a standard interface for connecting to and working with relational databases.

What is Data?

Data is information that can be stored, processed, and retrieved. In software applications, data represents the core information that needs to be persisted beyond the application's runtime.

Where Should Data Be Stored?

Text Files - Problems:

- **No Structure**: Difficult to organize and search
- **No Data Integrity:** No validation or constraints
- **Poor Performance**: Linear search required
- **Concurrency Issues**: Multiple users can't access simultaneously
- **No Security**: No access control mechanisms

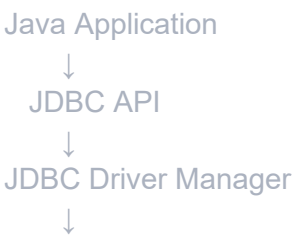
RDBMS (Relational Database Management System) - Solution:

- **Structured Storage**: Tables, rows, columns
- **Data Integrity**: Constraints, validation rules
- **Efficient Queries**: SQL for complex operations
- **Concurrency Control**: Multiple users can access safely
- **Security Features**: User authentication and authorization
- **ACID Properties**: Atomicity, Consistency, Isolation, Durability

Why JDBC?

- **Database Independence**: Same code works with different databases
- **Standardized API**: Consistent interface across database vendors
- **Flexibility**: Easy to switch between databases
- **Supported Databases**: PostgreSQL, MySQL, Oracle, H2, SQL Server, etc.

JDBC Architecture:



Database-Specific Driver

↓

Database

Lecture 2: PostgreSQL Setup

Complete PostgreSQL Installation Steps

Step 1: Download PostgreSQL

1. Visit postgresql.org
2. Click "Download" → Select your operating system
3. Download the installer for your platform

Step 2: Installation Process

Windows:

1. Run the downloaded .exe file as administrator
2. Follow the installation wizard:
    - Choose installation directory (default: `C:\Program Files\PostgreSQL\15`)
    - Select components (PostgreSQL Server, pgAdmin 4, Command Line Tools)
    - Choose data directory (default: `C:\Program Files\PostgreSQL\15\data`)
    - Set password for postgres superuser (remember this!)
    - Set port (default: 5432)
    - Choose local (default is fine)
3. Complete installation

macOS:

1. Run the downloaded .dmg file
2. Follow similar steps as Windows
3. Or use Homebrew: `brew install postgresql`

Linux (Ubuntu/Debian):

```
sudo apt update
sudo apt install postgresql postgresql-contrib
sudo systemctl start postgresql
sudo systemctl enable postgresql
```

Step 3: Verify Installation

1. Open Command Prompt/Terminal
2. Test connection: `psql -U postgres`
3. Enter the password you set during installation

Step 4: Using pgAdmin

1. Launch pgAdmin 4 from Start Menu/Applications
2. Create master password for pgAdmin
3. Connect to the PostgreSQL server:
    - Host: localhost
    - Port: 5432
    - Username: postgres
    - Password: (your set password)

Step 5: Create Database and Table

1. In pgAdmin, right-click "Databases" → "Create" → "Database"
2. Name: `demo`
3. Click "Save"

Create Table using SQL:

```
-- Connect to demo database
-- Create student table
CREATE TABLE student (
    sid INTEGER PRIMARY KEY,
    sname TEXT NOT NULL,
    marks INTEGER
);

-- Insert sample data
INSERT INTO student VALUES (1, 'Navin', 80);
```

Lecture 4: PostgreSQL JDBC Driver Setup

Complete IntelliJ IDEA Setup:

Step 1: Create New Project

1. Open IntelliJ IDEA
2. File → New → Project
3. Choose "Java" → Next
4. Project name: `JDBCCourse`
5. Choose project location and JDK version
6. Finish

Step 2: Download PostgreSQL JDBC Driver

1. Visit jdbc.postgresql.org
2. Download latest JDBC driver (.jar file)
3. Save to a known location (e.g., `libs` folder in the project).

Step 3: Add Driver to Project

Method 1: IntelliJ IDEA

1. File → Project Structure (Ctrl+Alt+Shift+S)
2. Select "Modules" → Dependencies tab
3. Click "+" → "JARs or directories"
4. Navigate to downloaded postgresql-xx.xx.jar file
5. Select and click OK
6. Apply and OK

Method 2: Manual Library Addition

1. Create `libs` folder in project root
2. Copy `postgresql-xx.xx.jar` to libs folder
3. Right-click jar file → "Add as Library"

Step 4: Verify Setup

Create a test class to verify the driver is accessible:

```
public class DriverTest {
    public static void main(String[] args) {
        try {
            Class.forName("org.postgresql.Driver");
            System.out.println("PostgreSQL Driver loaded successfully!");
        } catch (ClassNotFoundException e) {
            System.out.println("Driver not found: " + e.getMessage());
        }
```

```
    }
}
```

Lecture 6: Execute and Process Queries

Complete Query Execution Code:

```
import java.sql.*;

public class DemoJdbc {
    public static void main(String[] args) {
        String url = "jdbc:postgresql://localhost:5432/demo";
        String username = "postgres";
        String password = "0000";

        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;

        try {
            // Load driver and create connection
            Class.forName("org.postgresql.Driver");
            conn = DriverManager.getConnection(url, username, password);
            System.out.println("Connection established");

            // SQL query
            String query = "SELECT sname FROM student WHERE sid = 1";

            // Create statement
            stmt = conn.createStatement();

            // Execute query
            rs = stmt.executeQuery(query);

            // Process result
            if (rs.next()) {
                String studentName = rs.getString("sname");
                System.out.println("Student name: " + studentName);
            } else {
                System.out.println("No student found with sid = 1");
            }

        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
            e.printStackTrace();
        } finally {
            // Close resources in reverse order
            try {
                if (rs != null) rs.close();
                if (stmt != null) stmt.close();
```

```
            if (conn != null) conn.close();
            System.out.println("Connection closed");
        } catch (SQLException e) {
            System.out.println("Error closing resources: " + e.getMessage());
        }
    }
}
}
```

Understanding ResultSet:

- **rs.next()**: Moves cursor to next row, returns true if row exists
- **rs.getString("column_name")**: Gets string value from named column
- **rs.getInt("column_name")**: Gets integer value from named column
- **rs.getString(1)**: Gets string value from first column (1-indexed)

Lecture 10: Prepared Statement Solution

What is PreparedStatement?

PreparedStatement is a precompiled SQL statement that:

- Prevents SQL injection attacks
- Improves performance through query plan reuse
- Provides cleaner, more readable code
- Handles data type conversions automatically

Complete PreparedStatement Example:

```
import java.sql.*;

public class PreparedStatementDemo {
    public static void main(String[] args) {
        String url = "jdbc:postgresql://localhost:5432/demo";
        String username = "postgres";
        String password = "0000";

        Connection conn = null;
        PreparedStatement pstmt = null;

        try {
            Class.forName("org.postgresql.Driver");
            conn = DriverManager.getConnection(url, username, password);

            // SQL with placeholders (?)
            String sql = "INSERT INTO student VALUES (?, ?, ?)";

            // Create PreparedStatement
            pstmt = conn.prepareStatement(sql);

            // Set parameters (1-indexed)
            pstmt.setInt(1, 10);          // sid
            pstmt.setString(2, "Ananya");  // sname
            pstmt.setInt(3, 90);          // marks

            // Execute the statement
            int rowsAffected = pstmt.executeUpdate();
            System.out.println("Rows inserted: " + rowsAffected);

        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
            e.printStackTrace();
        } finally {
```

```
            try {
                if (pstmt != null) pstmt.close();
                if (conn != null) conn.close();
            } catch (SQLException e) {
                System.out.println("Error closing resources: " + e.getMessage());
            }
        }
    }
}
```

Understanding Placeholders (?):

- **?** represents a parameter placeholder
- Parameters are 1-indexed (first ? is index 1)
- Use appropriate setter methods based on data type:
    - `setInt(index, value)` for integers
    - `setString(index, value)` for strings
    - `setDouble(index, value)` for doubles
    - `setDate(index, value)` for dates
    - `setBoolean(index, value)` for booleans

PreparedStatement vs Statement Comparison:

Aspect	Statement	PreparedStatement
SQL Injection	Vulnerable	Safe
Performance	Slower (compiled each time)	Faster (precompiled)
Code Readability	Complex concatenation	Clean and simple
Parameter Handling	Manual string manipulation	Automatic type handling
Reusability	Limited	High (can reuse with different parameters)

Complete CRUD with PreparedStatement:

```
public class PreparedStatementCRUD {
    private static final String URL = "jdbc:postgresql://localhost:5432/demo";
    private static final String USERNAME = "postgres";
    private static final String PASSWORD = "0000";
```

```
    // CREATE
    public static void insertStudent(int sid, String sname, int marks) {
        String sql = "INSERT INTO student VALUES (?, ?, ?)";
        try (Connection conn = DriverManager.getConnection(URL, USERNAME,
PASSWORD);
             PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setInt(1, sid);
            pstmt.setString(2, sname);
            pstmt.setInt(3, marks);

            int rows = pstmt.executeUpdate();
            System.out.println("Inserted " + rows + " record(s)");

        } catch (SQLException e) {
            System.out.println("Insert error: " + e.getMessage());
        }
    }

    // READ
    public static void getStudent(int sid) {
        String sql = "SELECT * FROM student WHERE sid = ?";
        try (Connection conn = DriverManager.getConnection(URL, USERNAME,
PASSWORD);
             PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setInt(1, sid);
            ResultSet rs = pstmt.executeQuery();

            if (rs.next()) {
                System.out.println("ID: " + rs.getInt("sid") +
                                   ", Name: " + rs.getString("sname") +
                                   ", Marks: " + rs.getInt("marks"));
            } else {
                System.out.println("Student not found");
            }

        } catch (SQLException e) {
            System.out.println("Select error: " + e.getMessage());
        }
    }

    // UPDATE
    public static void updateStudent(int sid, String newName, int newMarks) {
        String sql = "UPDATE student SET sname = ?, marks = ? WHERE sid = ?";
        try (Connection conn = DriverManager.getConnection(URL, USERNAME,
PASSWORD);
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
```

```
            pstmt.setString(1, newName);
            pstmt.setInt(2, newMarks);
            pstmt.setInt(3, sid);

            int rows = pstmt.executeUpdate();
            System.out.println("Updated " + rows + " record(s)");

        } catch (SQLException e) {
            System.out.println("Update error: " + e.getMessage());
        }
    }

    // DELETE
    public static void deleteStudent(int sid) {
        String sql = "DELETE FROM student WHERE sid = ?";
        try (Connection conn = DriverManager.getConnection(URL, USERNAME,
PASSWORD);
             PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setInt(1, sid);

            int rows = pstmt.executeUpdate();
            System.out.println("Deleted " + rows + " record(s)");

        } catch (SQLException e) {
            System.out.println("Delete error: " + e.getMessage());
        }
    }
}
```

Lecture 1: JDBC Introduction

What is JDBC?

JDBC (Java Database Connectivity) is a Java API that enables Java applications to interact with databases. It provides a standard interface for connecting to and working with relational databases.

What is Data?

Data is information that can be stored, processed, and retrieved. In software applications, data represents the core information that needs to be persisted beyond the application's runtime.

Where Should Data Be Stored?

Text Files - Problems:

- **No Structure**: Difficult to organize and search
- **No Data Integrity:** No validation or constraints
- **Poor Performance**: Linear search required
- **Concurrency Issues**: Multiple users can't access simultaneously
- **No Security**: No access control mechanisms

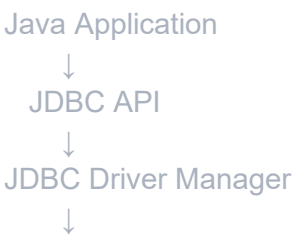
RDBMS (Relational Database Management System) - Solution:

- **Structured Storage**: Tables, rows, columns
- **Data Integrity**: Constraints, validation rules
- **Efficient Queries**: SQL for complex operations
- **Concurrency Control**: Multiple users can access safely
- **Security Features**: User authentication and authorization
- **ACID Properties**: Atomicity, Consistency, Isolation, Durability

Why JDBC?

- **Database Independence**: Same code works with different databases
- **Standardized API**: Consistent interface across database vendors
- **Flexibility**: Easy to switch between databases
- **Supported Databases**: PostgreSQL, MySQL, Oracle, H2, SQL Server, etc.

JDBC Architecture:



Database-Specific Driver

↓

Database

Lecture 2: PostgreSQL Setup

Complete PostgreSQL Installation Steps

Step 1: Download PostgreSQL

1. Visit postgresql.org
2. Click "Download" → Select your operating system
3. Download the installer for your platform

Step 2: Installation Process

Windows:

1. Run the downloaded .exe file as administrator
2. Follow the installation wizard:
    - Choose installation directory (default: `C:\Program Files\PostgreSQL\15`)
    - Select components (PostgreSQL Server, pgAdmin 4, Command Line Tools)
    - Choose data directory (default: `C:\Program Files\PostgreSQL\15\data`)
    - Set password for postgres superuser (remember this!)
    - Set port (default: 5432)
    - Choose local (default is fine)
3. Complete installation

macOS:

1. Run the downloaded .dmg file
2. Follow similar steps as Windows
3. Or use Homebrew: `brew install postgresql`

Linux (Ubuntu/Debian):

```
sudo apt update
sudo apt install postgresql postgresql-contrib
sudo systemctl start postgresql
sudo systemctl enable postgresql
```

Step 3: Verify Installation

1. Open Command Prompt/Terminal
2. Test connection: `psql -U postgres`
3. Enter the password you set during installation

Step 4: Using pgAdmin

1. Launch pgAdmin 4 from Start Menu/Applications
2. Create master password for pgAdmin
3. Connect to the PostgreSQL server:
    - Host: localhost
    - Port: 5432
    - Username: postgres
    - Password: (your set password)

Step 5: Create Database and Table

1. In pgAdmin, right-click "Databases" → "Create" → "Database"
2. Name: `demo`
3. Click "Save"

Create Table using SQL:

```
-- Connect to demo database
-- Create student table
CREATE TABLE student (
    sid INTEGER PRIMARY KEY,
    sname TEXT NOT NULL,
    marks INTEGER
);

-- Insert sample data
INSERT INTO student VALUES (1, 'Navin', 80);
```

Lecture 4: PostgreSQL JDBC Driver Setup

Complete IntelliJ IDEA Setup:

Step 1: Create New Project

1. Open IntelliJ IDEA
2. File → New → Project
3. Choose "Java" → Next
4. Project name: `JDBCCourse`
5. Choose project location and JDK version
6. Finish

Step 2: Download PostgreSQL JDBC Driver

1. Visit jdbc.postgresql.org
2. Download latest JDBC driver (.jar file)
3. Save to a known location (e.g., `libs` folder in the project).

Step 3: Add Driver to Project

Method 1: IntelliJ IDEA

1. File → Project Structure (Ctrl+Alt+Shift+S)
2. Select "Modules" → Dependencies tab
3. Click "+" → "JARs or directories"
4. Navigate to downloaded postgresql-xx.xx.jar file
5. Select and click OK
6. Apply and OK

Method 2: Manual Library Addition

1. Create `libs` folder in project root
2. Copy `postgresql-xx.xx.jar` to libs folder
3. Right-click jar file → "Add as Library"

Step 4: Verify Setup

Create a test class to verify the driver is accessible:

```
public class DriverTest {
    public static void main(String[] args) {
        try {
            Class.forName("org.postgresql.Driver");
            System.out.println("PostgreSQL Driver loaded successfully!");
        } catch (ClassNotFoundException e) {
            System.out.println("Driver not found: " + e.getMessage());
        }
```

```
    }
}
```

Lecture 6: Execute and Process Queries

Complete Query Execution Code:

```
import java.sql.*;

public class DemoJdbc {
    public static void main(String[] args) {
        String url = "jdbc:postgresql://localhost:5432/demo";
        String username = "postgres";
        String password = "0000";

        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;

        try {
            // Load driver and create connection
            Class.forName("org.postgresql.Driver");
            conn = DriverManager.getConnection(url, username, password);
            System.out.println("Connection established");

            // SQL query
            String query = "SELECT sname FROM student WHERE sid = 1";

            // Create statement
            stmt = conn.createStatement();

            // Execute query
            rs = stmt.executeQuery(query);

            // Process result
            if (rs.next()) {
                String studentName = rs.getString("sname");
                System.out.println("Student name: " + studentName);
            } else {
                System.out.println("No student found with sid = 1");
            }

        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
            e.printStackTrace();
        } finally {
            // Close resources in reverse order
            try {
                if (rs != null) rs.close();
                if (stmt != null) stmt.close();
```

```
            if (conn != null) conn.close();
            System.out.println("Connection closed");
        } catch (SQLException e) {
            System.out.println("Error closing resources: " + e.getMessage());
        }
    }
}
}
```

Understanding ResultSet:

- **rs.next()**: Moves cursor to next row, returns true if row exists
- **rs.getString("column_name")**: Gets string value from named column
- **rs.getInt("column_name")**: Gets integer value from named column
- **rs.getString(1)**: Gets string value from first column (1-indexed)

Lecture 10: Prepared Statement Solution

What is PreparedStatement?

PreparedStatement is a precompiled SQL statement that:

- Prevents SQL injection attacks
- Improves performance through query plan reuse
- Provides cleaner, more readable code
- Handles data type conversions automatically

Complete PreparedStatement Example:

```
import java.sql.*;

public class PreparedStatementDemo {
    public static void main(String[] args) {
        String url = "jdbc:postgresql://localhost:5432/demo";
        String username = "postgres";
        String password = "0000";

        Connection conn = null;
        PreparedStatement pstmt = null;

        try {
            Class.forName("org.postgresql.Driver");
            conn = DriverManager.getConnection(url, username, password);

            // SQL with placeholders (?)
            String sql = "INSERT INTO student VALUES (?, ?, ?)";

            // Create PreparedStatement
            pstmt = conn.prepareStatement(sql);

            // Set parameters (1-indexed)
            pstmt.setInt(1, 10);          // sid
            pstmt.setString(2, "Ananya");  // sname
            pstmt.setInt(3, 90);          // marks

            // Execute the statement
            int rowsAffected = pstmt.executeUpdate();
            System.out.println("Rows inserted: " + rowsAffected);

        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
            e.printStackTrace();
        } finally {
```

```
            try {
                if (pstmt != null) pstmt.close();
                if (conn != null) conn.close();
            } catch (SQLException e) {
                System.out.println("Error closing resources: " + e.getMessage());
            }
        }
    }
}
```

Understanding Placeholders (?):

- **?** represents a parameter placeholder
- Parameters are 1-indexed (first ? is index 1)
- Use appropriate setter methods based on data type:
    - `setInt(index, value)` for integers
    - `setString(index, value)` for strings
    - `setDouble(index, value)` for doubles
    - `setDate(index, value)` for dates
    - `setBoolean(index, value)` for booleans

PreparedStatement vs Statement Comparison:

Aspect	Statement	PreparedStatement
SQL Injection	Vulnerable	Safe
Performance	Slower (compiled each time)	Faster (precompiled)
Code Readability	Complex concatenation	Clean and simple
Parameter Handling	Manual string manipulation	Automatic type handling
Reusability	Limited	High (can reuse with different parameters)

Complete CRUD with PreparedStatement:

```
public class PreparedStatementCRUD {
    private static final String URL = "jdbc:postgresql://localhost:5432/demo";
    private static final String USERNAME = "postgres";
    private static final String PASSWORD = "0000";
```

```
    // CREATE
    public static void insertStudent(int sid, String sname, int marks) {
        String sql = "INSERT INTO student VALUES (?, ?, ?)";
        try (Connection conn = DriverManager.getConnection(URL, USERNAME,
PASSWORD);
             PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setInt(1, sid);
            pstmt.setString(2, sname);
            pstmt.setInt(3, marks);

            int rows = pstmt.executeUpdate();
            System.out.println("Inserted " + rows + " record(s)");

        } catch (SQLException e) {
            System.out.println("Insert error: " + e.getMessage());
        }
    }

    // READ
    public static void getStudent(int sid) {
        String sql = "SELECT * FROM student WHERE sid = ?";
        try (Connection conn = DriverManager.getConnection(URL, USERNAME,
PASSWORD);
             PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setInt(1, sid);
            ResultSet rs = pstmt.executeQuery();

            if (rs.next()) {
                System.out.println("ID: " + rs.getInt("sid") +
                                   ", Name: " + rs.getString("sname") +
                                   ", Marks: " + rs.getInt("marks"));
            } else {
                System.out.println("Student not found");
            }

        } catch (SQLException e) {
            System.out.println("Select error: " + e.getMessage());
        }
    }

    // UPDATE
    public static void updateStudent(int sid, String newName, int newMarks) {
        String sql = "UPDATE student SET sname = ?, marks = ? WHERE sid = ?";
        try (Connection conn = DriverManager.getConnection(URL, USERNAME,
PASSWORD);
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
```

```
            pstmt.setString(1, newName);
            pstmt.setInt(2, newMarks);
            pstmt.setInt(3, sid);

            int rows = pstmt.executeUpdate();
            System.out.println("Updated " + rows + " record(s)");

        } catch (SQLException e) {
            System.out.println("Update error: " + e.getMessage());
        }
    }

    // DELETE
    public static void deleteStudent(int sid) {
        String sql = "DELETE FROM student WHERE sid = ?";
        try (Connection conn = DriverManager.getConnection(URL, USERNAME,
PASSWORD);
             PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setInt(1, sid);

            int rows = pstmt.executeUpdate();
            System.out.println("Deleted " + rows + " record(s)");

        } catch (SQLException e) {
            System.out.println("Delete error: " + e.getMessage());
        }
    }
}
```