# ACCESS TO DATABASES

## Java Applications and Access to Databases

# AGENDA

A quick recap to databases
Database modelling
JDBC, SQL & SQLite
Java DB & MySQL
Build tables and perform queries using command line and java code

# Recap of databases

- What is database?
  - A database is an organised collection of structured information or data, typically stored electronically in a computer system.

- Can you list some types of databases?
  - Relational Databases (RDBMS)
  - NoSQL Databases
  - In-Memory Databases
  - Object-Oriented Databases
  - Hierarchical Databases

- Which is the common type and how it works?
  - Relational Database with SQL is used to query and manage the data

# Data Storage

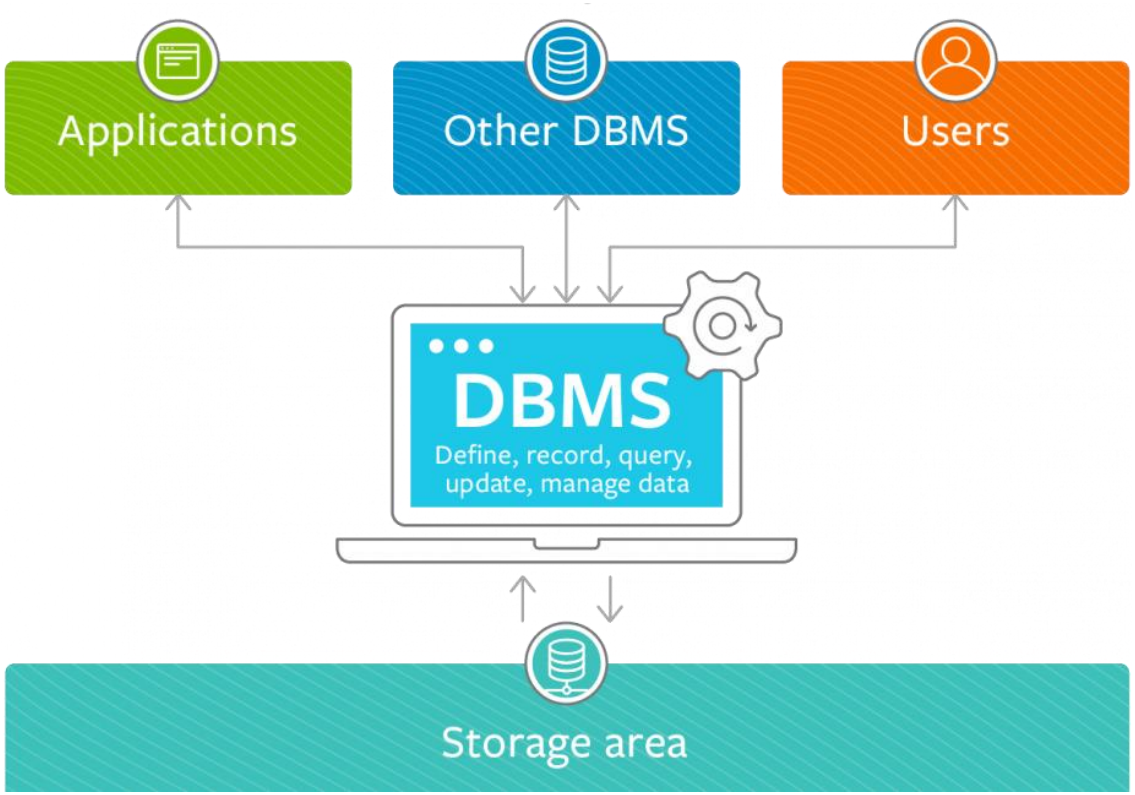
- When we store anything, we generally need to consider:
  - Organisation (a system or model for arranging the stuff)
    - In a library, books are arranged based on categories such as fiction, non-fiction, history, science, etc. They are further organized alphabetically by author or title within each category
  - Physical storage (where stuff goes)
    - Books are stored physically on shelves. Each shelf might be labeled with the genre or subject, and the books are placed in rows based on their category, author name, and title
  - Access (how we manage the stuff after we stored it)
    - A catalog system (either manual or computerized) helps users search for books. Users can check out or return books, and the library staff manages the inventory to ensure that the books are properly stored, available, and accessible to the public

# Data Model

- Data modelling is the first step in database design
- The data model is a representation of the content, relationships and business rules needed to support the requirements of an application
- The data model emphasises what data is needed and how it should be organised
- It should reinforce the business rules of the application
- It should ensure the security, consistency and integrity of data

# Data Access

- For data to be accessible, we need a way to interface with it
- A database management system (DBMS) provides this interface



# Database Management System (DBMS)

- Program (or collection of programs) designed to
  - manage the Creation, Retrieval, Updating and Deletion of data stored in a database (aka CRUD operations)

# Relational Database Management Systems (RDBMSs)

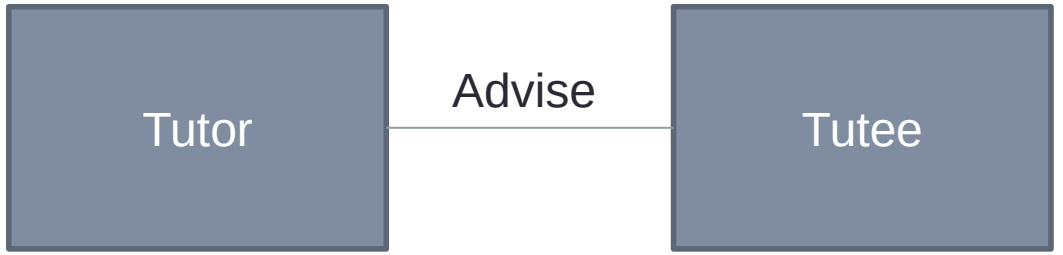
- A program that allows you to create, update, and administer a relational database
- Some popular original relational database management systems (RDBMSs) are Microsoft SQL Server, Oracle, Sybase and IBM DB2, PostgreSQL, MariaDB are popular open-source DBMSs that can be downloaded and used freely by anyone

# Relational database terminology

- Database schema (skeleton structure of the database)
- Database tables (entities or relations)
- Database columns (attributes or fields)
- Database rows (records or instances)
- Database relationships (e.g. ERD)

*one to one*
*one to many*
*many to one*
*many to many*

Which relationship best represents student support system? Why?



# What is JDBC?

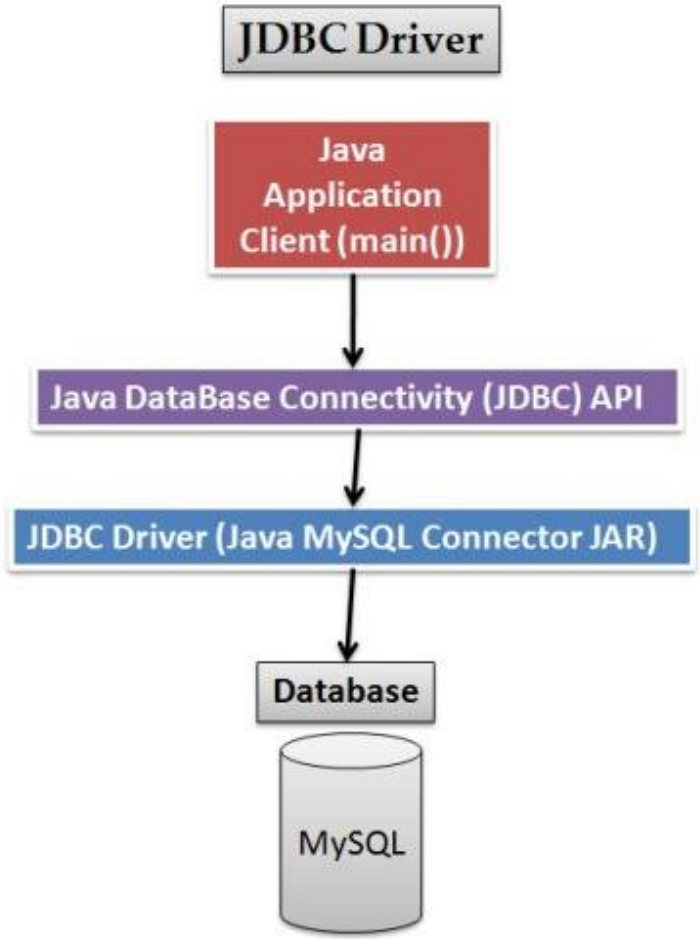
- JDBC (Java Database Connectivity) is a Java API that allows Java applications to interact with databases
- It provides a standard abstraction for Java applications to communicate with various databases
  - What is an API? (check this site for more info)
  - JDBC Documentation

# Introducing Java DB

- First introduced by Sun Microsystems as part of JDK 6 (and not included in the JRE)
- So, you need to add the connector to java DB (JDBC) to your project manually (I will guide you through this)
- It’s secure, supports JDBC and SQL (including transactions, stored procedures, and concurrency), and has a small footprint
- All database venders have connectivity drivers to connect with Java

# JDBC Driver

- Because data sources are accessed in different ways, JDBC uses drivers to abstract over their implementations
- This abstraction lets you write an application that can be adapted to various data source without having to change a single line of code (in most cases)
- Drivers are implementations of the ***java.sql.Driver*** interface
- A JDBC driver is a classfile plug-in for communicating with a database



# Basic Steps to use a Database in Java

- Establish a connection
- Create JDBC Statements
- Excute SQL/SQLite Statements
- Get ResultSet
- Close connections

# Data Sources, Drivers, and Connections

## DriverManger class

- To connect to a data source and obtain a Connection instance, use the DriverManager class

## getConnection() methods

- Define a url argument that specifies a string-based URL that starts with the jdbc: prefix
- Continues with data source-specific syntax.DriverManager's getConnection() methods

## createStatement() method

- Create a statement argument that executes SQL query using Connection's createStatement() methods
- JDBC methods throw SQLException or one of its subclasses when something goes wrong
- They provide vendor codes, SQL state strings, and other kinds of information (e.g. code 1022 means duplicate primary key)

# Establish a connection

- **import java.sql.*;**
- **Load the vendor specific driver**
  - Class.forName("oracle.jdbc.driver.OracleDriver");
    - What do you think this statement does, and how?
    - Dynamically loads a driver class, for Oracle database
- **Make the connection**
  - Connection con = DriverManager.getConnection( "jdbc:oracle:thin:@oracle-prod:1521:OPROD", username, passwd);
    - What do you think this statement does?
    - Establishes connection to database by obtaining a Connection object

# Create JDBC statement(s)

- stmt = con.createStatement() ;
  - Creates a Statement object for sending SQL statements to the database

# Executing SQL Statements

- String createStudent = " Create table Student " + " (Id Integer not null, Name VARCHAR(32), " + " Marks Integer) " ;

  stmt.executeUpdate(createStudent);
  - What does this statement do?
- String insertStudent = " Insert into Student values “ + " (123456789,abc,100) " ;

  stmt.executeUpdate(insertStudent);

# Get ResultSet

- String queryStudent = " select * from Student " ;
- ResultSet rs = Stmt.executeQuery(queryStudent);

```
while (rs.next()) {
    int ssn = rs.getInt( " SSN " );
    String name = rs.getString( " NAME " );
    int marks = rs.getInt( " MARKS " );
}
```

# Close Connection

- stmt.close();
- con.close();

# Transactions and JDBC

- JDBC allows SQL statements to be grouped together into a single transaction
- Transaction control is performed by the Connection object, default mode is auto-commit, I.e., each sql statement is treated as a transaction
- We can turn off the auto-commit mode with con.setAutoCommit(false);
- And turn it back on with con.setAutoCommit(true);
- Once auto-commit is off, no SQL statement will be committed until an explicit is invoked con.commit();
- At this point all changes done by the SQL statements will be made permanent in the database.

# Handling Errors with Exceptions

- Programs should recover and leave the database in a consistent state
- If a statement in the try block throws an exception or warning, it can be caught in one of the corresponding catch statements
  - How might a finally {…} block be helpful here?

- E.g., you could rollback your transaction in a catch { …} block or close database connection and free database related resources in finally {…} block

# Student Grade Example

- Download and add the Microsoft SQL Server JDBC driver (e.g., mssql-jdbc-8.4.1.jre8.jar) to your project's classpath
- Ensure that a SQL Server instance is running and that you have a students table similar to the one below
- Use the instructions attached to Moodle to connect NetBeans to MySQL Driver

```
CREATE TABLE students (
    student_id INT PRIMARY KEY,
    student_name NVARCHAR(100),
    grade FLOAT
);
```

# Student Grade Example Cont

- Import the important packages and classes

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

# Student Grade Example Cont

## main class

```
public class StudentGradesSQLServer {

    // SQL Server connection details
    private static final String DB_URL =
"jdbc:sqlserver://localhost:1433;databaseName=SchoolDB"; //
Update your DB details
    private static final String DB_USER = "your_username"; // Replace
with your username
    private static final String DB_PASSWORD = "your_password"; //
Replace with your password

    // JDBC Connection
    private Connection conn = null;
```

## constructor to establish the DB connection

```
// Constructor to initialize the DB connection
    public StudentGradesSQLServer() {
        try {
            // Load the SQL Server JDBC driver
            Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");


            // Establish connection
            conn = DriverManager.getConnection(DB_URL, DB_USER,
DB_PASSWORD);
            System.out.println("Connected to SQL Server database successfully.");
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
    }
```

# Student Grade Example Cont

## Method to insert a new student with grade

```
public void insertStudent(int studentId, String studentName, double grade) {
    String insertSQL = "INSERT INTO students (student_id, student_name, grade)
VALUES (?, ?, ?)";
    try (PreparedStatement pstmt = conn.prepareStatement(insertSQL)) {
      pstmt.setInt(1, studentId);
      pstmt.setString(2, studentName);
      pstmt.setDouble(3, grade);

      int rowsInserted = pstmt.executeUpdate();
      if (rowsInserted > 0) {
        System.out.println("A new student record inserted successfully!");
      }
    } catch (SQLException e) {
      e.printStackTrace();
    }
```

## Method to retrieve all students and their grades

```
public void getAllStudentGrades() {
    String selectSQL = "SELECT student_id, student_name, grade FROM students";
    try (PreparedStatement pstmt = conn.prepareStatement(selectSQL);
      ResultSet rs = pstmt.executeQuery()) {
      System.out.println("Student Grades:");
      while (rs.next()) {
        int studentId = rs.getInt("student_id");
        String studentName = rs.getString("student_name");
        double grade = rs.getDouble("grade");
        System.out.println("ID: " + studentId + ", Name: " + studentName + ", Grade: " +
grade);
      }
    } catch (SQLException e) {
      e.printStackTrace();
    }
  }
```

# Student Grade Example Cont

## Close Connection

```
// Close the DB connection
  public void closeConnection() {
     try {
        if (conn != null && !conn.isClosed()) {
           conn.close();
           System.out.println("Database connection closed.");
        }
     } catch (SQLException e) {
        e.printStackTrace();
     }
  }
```

# Student Grade Example Cont

## Main method

```
public static void main(String[] args) {
    // Initialize the student grades management
    StudentGradesSQLServer studentGrades = new StudentGradesSQLServer();

    // Insert sample student data
    studentGrades.insertStudent(1, "John Doe", 88.5);
    studentGrades.insertStudent(2, "Jane Smith", 92.3);

    // Retrieve and display all student grades
    studentGrades.getAllStudentGrades();

    // Close the database connection
    studentGrades.closeConnection();
  }
}
```

# Tasks

- Complete the tutorial here: Connect MySQL Database to NetBeans in Java | JDBC
- Create a database for the banking system and run the example.