Databases

BCS1510

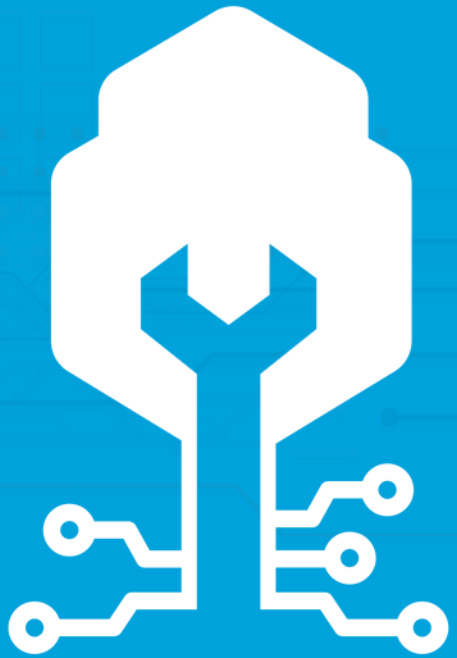
Dr. Katharina Schneider &
Dr. Tony Garnock-Jones*

🗓️ Week 4 - Lecture 1

📍 EPD150 MSM Conference Hall

Adapted with gratitude from the original lectures by Dr. Ashish Sai

*CC both of us when emailing please!



Maastricht University

What have we looked at so far?

- Database and SQL Fundamentals
  - Overview of databases, DBMS evolution, data models, and SQL essentials.
  - Introduction to relational algebra (see also lab 1) and SQL operations (table creation + basic data manipulation)
- Advanced SQL Techniques and Operations
  - Advanced query techniques (handling NULL values, using complex conditions, and combining data across multiple tables through different JOIN operations).
- Subqueries and Operators
  - Introduction to the role of subqueries in SQL and their usage in SELECT, FROM and WHERE clauses
  - Detailed discussion on operators such as IN, EXISTS, ANY and ALL
- Aggregation, Grouping and Subquery Applications
  - Use of SQL aggregate functions (SUM, AVG, COUNT, MIN, MAX)
  - Understanding grouping data with the GROUP BY clause and applying the HAVING clause
  - Practical applications of subqueries in complex SQL series

Learning Objectives

- Understand Set Operations in SQL
  - Learn to use set operations like UNION, INTERSECT and MINUS to combine, intersect, or subtract results from different queries
  - Understand the impact of these operations on the result sets
- Exploring Built-In Functions
  - Gain familiarity with SQL's built-in functions (including their application in data manipulation and query refinement).
  - Emphasize the usage of standard functions like SUM, AVG, MIN, MAX, and COUNT
- Mastering Data Modification Statements:
  - Develop skills to effectively use INSERT, DELETE and UPDATE statements in SQL to modify data within databases
  - Learn how to handle default values and use subqueries within these statements to manipulate large datasets
- Applying Advanced SQL Techniques
  - Apply advanced SQL techniques in practical scenarios, such as using subqueries in data modification, handling complex conditions in deletion, and managing database updates to enforce business rules or constraints
- Using SQL in Host Language Environment
  - Understand the three tier architecture
  - Learn how to process SQL statement with JDBC

Set Operators

UNION, UNION ALL, INTERSECT and MINUS

Set Operators

- Set operators and joins both combine data from multiple tables

- Remember JOINs:
  - Combine columns from multiple tables
  - Can join tables with different columns
  - Aim: bring related data together

- So what do Set Operators do?

Set Operators

- Combine results of entire queries (rows)
- Unite, intersect, or subtract only the results of subqueries with the same schema, i.e., same number of attributes, in the same order and with the same data types
- Set Operators in SQL remove duplicated rows in the results by default (Use UNION ALL if you want to suppress the deletion of duplicates)

Set Operator: UNION

- <subquery> UNION <subquery>

- Q: Using tables PCs, Laptops and Printers, find the list of prices of all products. List the model number and price for each product

```
SELECT model, price FROM PCs
UNION
SELECT model, price FROM Laptops
UNION
SELECT model, price FROM Printers;
```

- Note that the three subqueries above have the same schema

Set Operator: MINUS

- <subquery> MINUS <subquery>

- Q: Using Products(maker, model, type), find the makers who make at least one laptop model but no PC models

```
SELECT maker FROM Products WHERE type = 'laptop'
MINUS
SELECT maker FROM Products WHERE type = 'pc';
```

- Note that MINUS is not implemented in MySQL
- How can we do it instead?

MINUS - Alternative

- Q: Using Products(maker, model, type), find the makers who make at least one laptop model but no PC models

```
SELECT DISTINCT maker
 FROM Products
 WHERE type = 'laptop' AND maker NOT IN
  (SELECT maker FROM Products WHERE type = 'pc');
```

Set Operator: INTERSECT

- <subquery> INTERSECT <subquery>

- Q: Using Products, PCs and Laptops, find the makers who make at least one PC model with price above 500 and at least one laptop model also with price above 500.

```
SELECT maker FROM Products NATURAL JOIN PCs WHERE price > 500
INTERSECT
SELECT maker FROM Products NATURAL JOIN Laptops WHERE price > 500;
```

- Note that INTERSECT is not implemented in MySQL either
- How can we do it instead?

INTERSECT - Alternative

- Q: Using Products, PCs and Laptops, find the makers who make at least one PC model with price above 500 and at least one laptop model also with price above 500.

```
SELECT DISTINCT maker FROM Products NATURAL JOIN PCs
    WHERE price > 500 AND maker IN
     (SELECT maker FROM Products NATURAL JOIN Laptops
       WHERE price > 500);
```

Built-In Functions

Built-In Functions

- SQL standard defines a set of standard SQL built-in functions
- Only some aggregate functions (SUM, AVG, MIN, MAX, COUNT) are covered in this course
- but you are highly encouraged to use a wider range of built-in functions in your project

Usage of Built-in Functions

- See aggregate functions in last lecture

Built-in Functions: Some references

- Standard built-in functions in SQL92
  - http://db.apache.org/derby/docs/10.4/ref/rrefsqlj29026.html
- Built-in functions in MySQL:
  - http://dev.my_sql.com/doc/refman/5.7/en/functions.html
- Built-in functions in Oracle:
  - https://docs.oracle.com/cd/B28359_01/server.111/b28286/functions001.htm#SQLRF51174
- Built-in functions in the Microsoft SQL Server:
  - http://db.apache.org/derby/docs/10.4/ref/rrefsqlj29026.html

Data Modification Statements

INSERT, DELETE and UPDATE

Data Modification Statements

- A modification command does not return a result but changes the database in some way

- We already saw the basic data modification statements in lecture 2

- Remember:

INSERT

Inserting a Tuple

- To insert a single new tuple:

```
INSERT INTO table_name(column1, column2, column3, …)
VALUES (value_column1, value_column2, value_column3);
```

- Example:

```
INSERT INTO customers(customer_id, firstname,
     lastname, city)
VALUES ('0000000000', 'John', 'Doe', 'Maastricht');
```

Default Values

- In a CREATE TABLE statement, we can follow an attribute by DEFAULT and a value

```
CREATE TABLE students (
      id INT primary key,
      name VARCHAR(100),
      grade INT DEFAULT 10);
```

- When an inserted tuple has no value for that attribute, the default will be used

Inserting a Tuple

- Do we really need to specify the attributes in addition to the table?

- No, not necessarily

- But it is helpful if
  - We forgot the standard order of attributes for the relation
  - We don’t have values for all attributes and we want the system to fill in missing components with NULL or a default value

Inserting a Tuple Without Specifying Attributes

- To insert a single new tuple:

```
INSERT INTO table_name
VALUES (<list of values>);
```

- Example

```
INSERT INTO customers
VALUES ('0000000001', 'John', 'Doe', 'Maastricht',
    'Paul-Henri-Spaaklaan 1', NULL);
```

Inserting Many Tuples from a Subquery

- We may insert the entire result of a query into a relation using:

```
INSERT INTO table_name
(<subquery>);
```

Inserting Many Tuples from a Subquery

- Create a copy of Products(maker, model, type) called Products_copy

```
CREATE TABLE Products_copy(
      maker CHAR(1),
      model CHAR(4) Primary key,
      type VARCHAR(8) DEFAULT 'pc');
```

```
INSERT INTO Products_copy
 (SELECT * FROM Products);
```

DELETE

Deleting Rows

- To delete a row/multiple rows:

```
DELETE FROM table_name
WHERE condition;
```

- Example:

```
DELETE FROM Products_copy
WHERE type = 'printer' AND maker = 'A';
```

- Without condition (WHERE), all tuples in the relation are deleted

Deleting Tuples: Example

- Delete from Products_copy(maker, model, type) all models for which there is another mode by the same maker

- Suggestions?

Deleting Tuples: Example

- Delete from Products_copy(maker, model, type) all models for which there is another mode by the same maker

```
DELETE FROM Products_copy p
WHERE EXISTS ( SELECT model FROM products_copy WHERE
     maker = p.maker AND model <> p.model);
```

Deleting some tuples – Semantics

- Delete from Products(maker, model, type) all models for which there is another model by the same maker:
- Suppose maker H makes only products 3006 and 3007
- We go through the tuples one by one and check whether there is another model for the current maker
- Suppose we come to the tuple of product 3006 first
- So there is another model by the same maker and we delete the tuple of product 3006
- What happens to the 3007 tuple?

Deleting some tuples – Semantics

- Answer: we delete the 3007 tuple as well

- The reason is that the deletion proceeds in two stages:
  - Mark all tuples for which the WHERE condition is satisfied
  - Delete the marked tuples

UPDATE

UPDATE

- To change certain attributes in certain tuples of a relation

```
UPDATE <relation>
SET <list of attribute assignments>
 WHERE <condition on tuples>
```

- Example:

```
UPDATE PCs
SET price = 1999.0
 WHERE model='1001';
```

UPDATE Several Tuples

- Make 1999.0 the maximum price for PCs

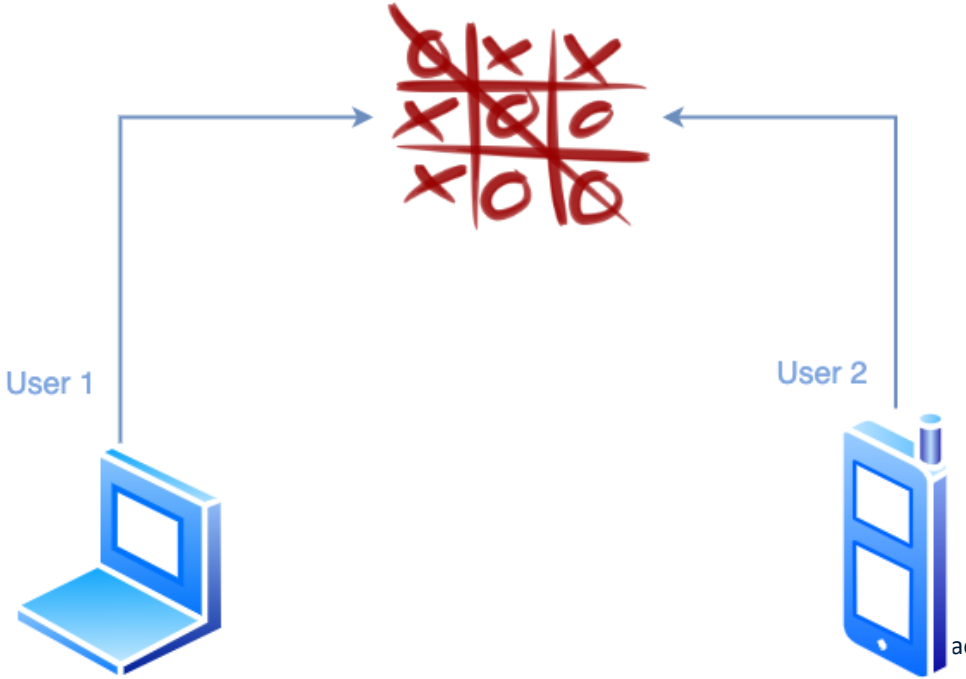
```
UPDATE PCs
SET price = 1999.0
WHERE price > 1999.0;
```

Short Break

- **10** minutes

Tic Tac Toe Game and Database Systems

- Imagine two players play a tic tac toe game together, one using a computer, one using a mobile phone



Tic Tac Toe Game and Database Systems

- What may be stored in the database?
  - Move records
  - Players
  - Game states
  - …

- Example:

Move_id	Game_id	Player_id	Position	Symbol	Move_time
1	1	23	0	X	2025-04-15 16:00:01
2	1	45	1	O	2025-04-15 16:00:10

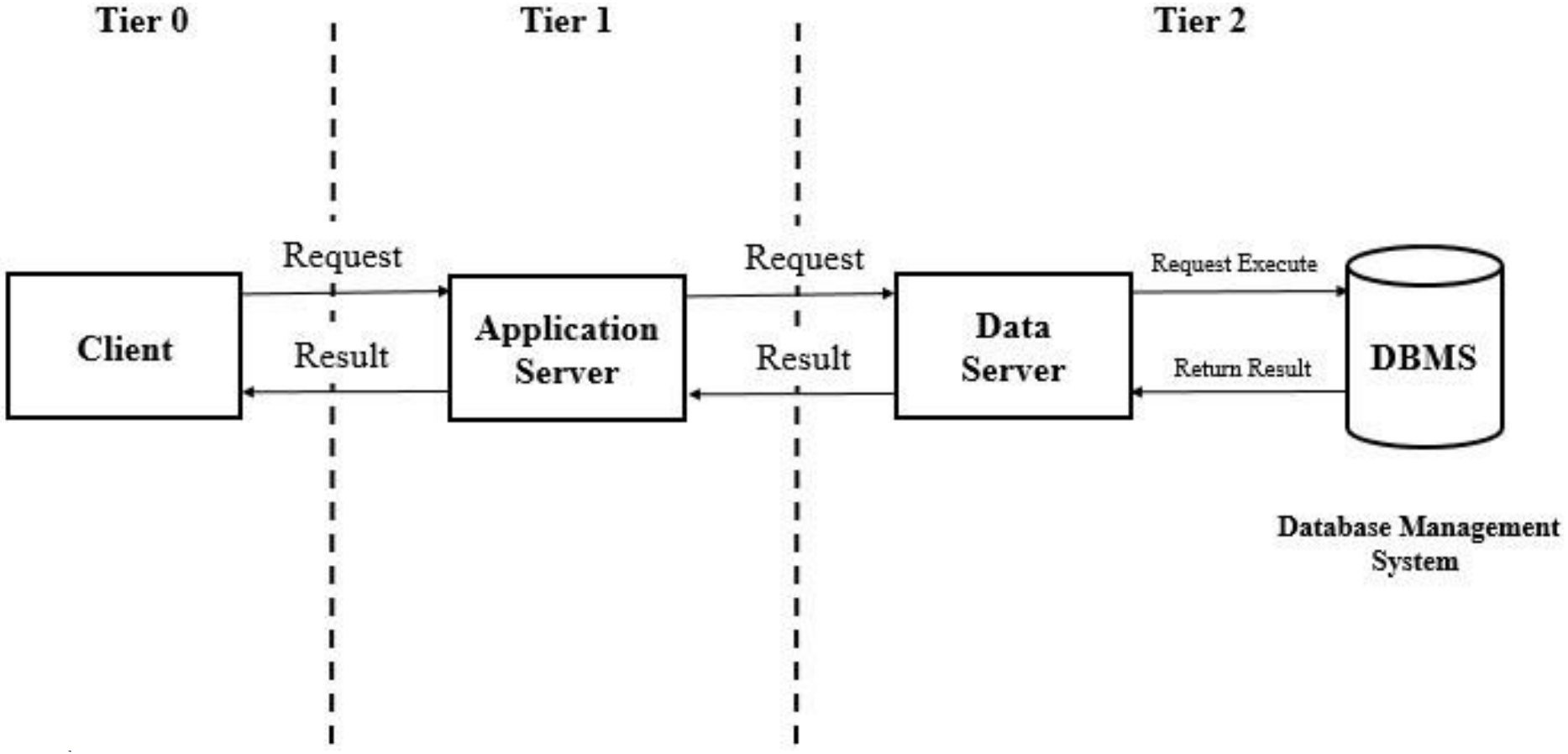
Architecture Realization

- A common environment for using a database has three tiers (three-tier client-server architecture)

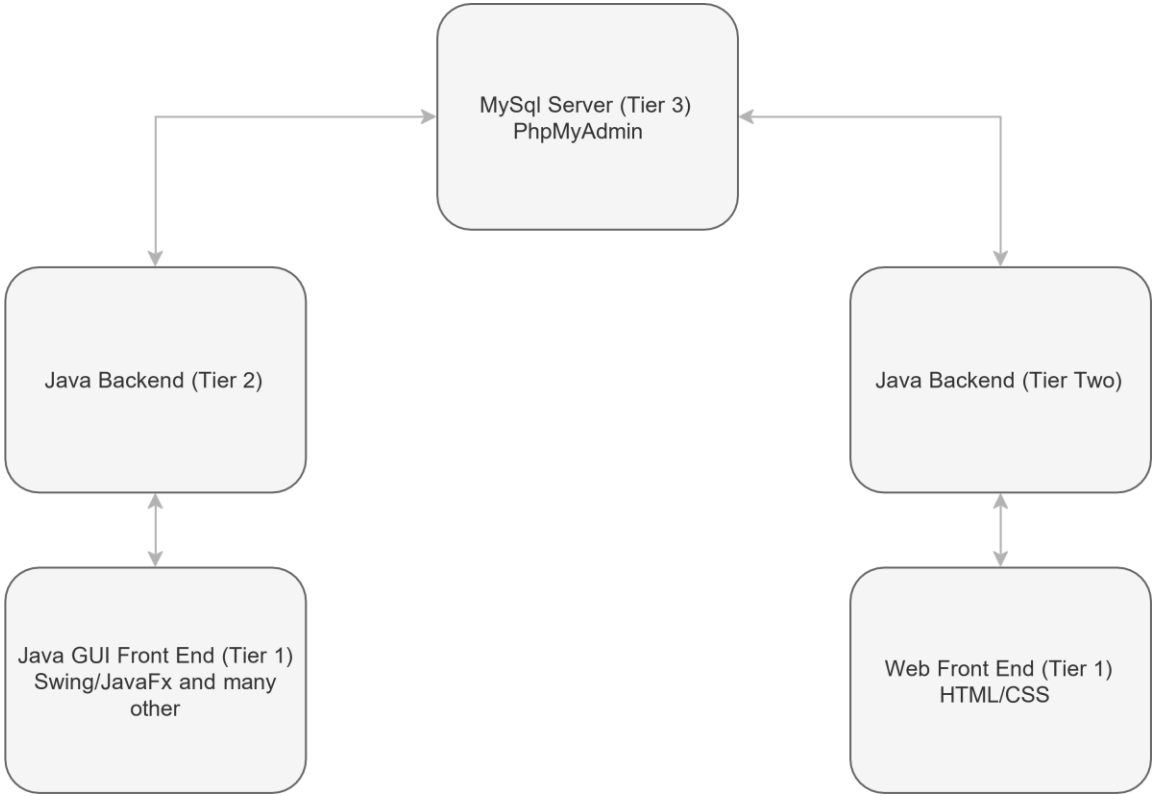
- First Tier – Client (front-end)
  - User Interface

- Second Tier – Application Server (back-end)
  - Business logic
  - Data processing logic

- Third Tier – Database server
  - Data validation
  - Database access


Tier 0
Tier 1
Tier 2
Request
Result
Request
Result
Request Execute
Return Result
Client
Application Server
Data Server
DBMS
Database Management System

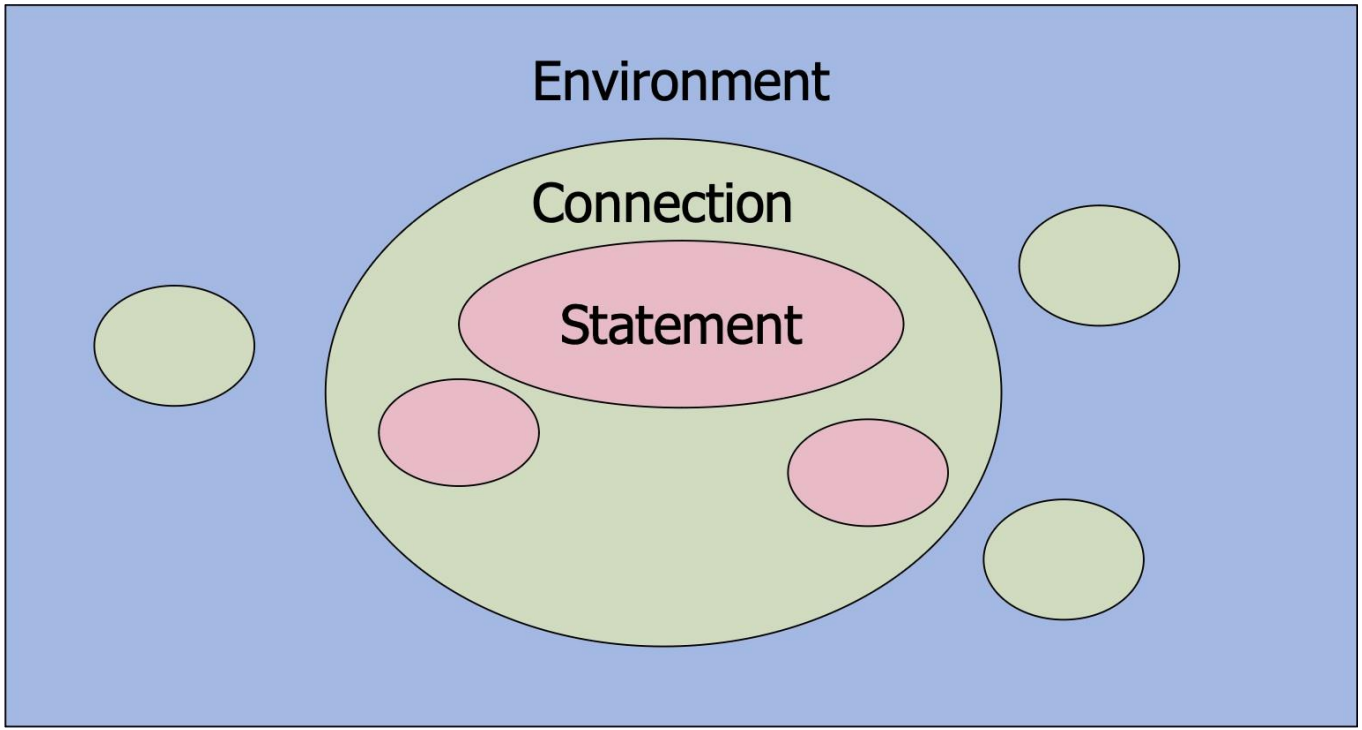
Three-Tier Architecture Tic Tac Toe



Environments, Connections, Queries

- The database is, in many DB-access languages an environment
- Database servers maintain some number of connections, so app servers can ask queries or perform modifications
- The app server issues statements: queries and modifications, usually

Environments, Connections, Queries



JDBC Java Database Connectivity

https://docs.oracle.com/javase/tutorial/jdbc/basics/index.html

Processing SQL Statements with JDBC

- In general, to process any SQL statement with JDBC, you follow these steps
  - Establishing a connection
  - Create a statement
  - Execute the query
  - Process the ResultSet object
  - Close the connection

Establish a connection

Establish a Connection

```
import java.sql.*;

public class main {
private static String dbUrl = "jdbc:mysql://localhost:3306/pcshop";
private static String dbUsername = "root";
private static String dbPassword = "…";

public static void main(String[] args) {
try {
    Connection myCon = DriverManager.getConnection(dbUrl,
    dbUsername, dbPassword) ;
…
```

Create a Statement

Create a statement

- JDBC provides classes for three different kinds of statements
  - Statement: Used to implement simple SQL statements with no parameters
  - PreparedStatement: (extends Statement) Used for precompiling SQL statements that might contain input parameters
  - CallableStatement: (extends PreparedStatement) Used to execute stored procedures that may contain both input and output parameters

Create a Statement

```
Connection conn = DriverManager.getConnection(...); // Establish conn

Statement stmt = conn.createStatement(); // Create the statement
String sql = "SELECT * FROM Products";  // Create query string
```

- https://docs.oracle.com/javase/tutorial/jdbc/basics/processingsqlstatements.html

Create a Statement

```
Connection conn = DriverManager.getConnection(...); // Establish conn

PreparedStatement pstmt = conn.prepareStatement("SELECT * FROM customers
                          WHERE customer_id = ?");
pstmt.setString(1,"0000000001");
```

- Placeholder for parameters: ?
- https://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html

When to use Statement when PreparedStatement?

- Static query with no user input: Statement is okay
- Query with user input of variables: Always use PreparedStatement (SQL injection)
- Repeated execution with different values: PerparedStatement

SQL injection (Real-World Attack Scenario)

- Scenario:
  - Imagine you have a login form and someone enters
  - Username: admin’ --
  - Password: anything

SQL injection - Statement

```
Connection conn = DriverManager.getConnection(...);

Statement stmt = conn.createStatement();
String sql = "SELECT * FROM Users
    WHERE username = 'admin' --' AND password = 'anything'";
```

- -- is a comment in SQL, so the right part gets ignored
- Result: login as an admin without knowing the password

SQL injection avoided - PreparedStatement

```
Connection conn = DriverManager.getConnection(...);

String sql = "Select * FROM Users WHERE username = ? AND password = ?";
PreparedStatement pstmt = conn.createStatement(sql);
pstmt.setString(1,admin);
pstmt.setString(2,password);
```

Execute a Query

Executing Statements - Updates vs Queries

- JDBC distinguishes queries from modifications, which it calls “updates”

- Statement and PreparedStatement each have methods executeQuery and executeUpdate
  - Statements: executeQuery(sql) – with argument (the query or modification to be executed)
  - PreparedStatements: executeQuery() – without argument

Example: Query + Statement

```
Connection conn = DriverManager.getConnection(...); // Establish conn

Statement stmt = conn.createStatement(); // Create the statement
String sql = "SELECT * FROM Products";  // Create query string
ResultSet rs = stmt.executeQuery(sql); // Execute the query here
```

Example: Query + PreparedStatement

```
Connection conn = DriverManager.getConnection(...);

String sql = "Select * FROM Users WHERE username = ? AND password = ?";
PreparedStatement pstmt = conn.createStatement(sql);
pstmt.setString(1,admin);
pstmt.setString(2,password);
ResultSet rs = pstmt.executeQuery(); // Execute the query here
```

Process the ResultSet Object

Accessing the ResultSet

- An object of type ResultSet is something like a cursor
- Method next() advances the “cursor” to the next tuple
- The first time next() is applied, it gets the first tuple
- If there are no more tuples, next() returns the values false

Accessing Components of Tuples

- When a ResultSet is referring to a tuple, we can get the components of that tuple by applying certain methods to the ResultSet
- Method getX(i) (where X is some type, and i is the attribute/column number) returns the value of that attribute
- The value must have type X

Example: Accessing Components

- Q: Give me the total sales for each model.
- SQL query:

```
SELECT model, SUM(quantity)
FROM pcshop.Sales
GROUP BY model;
```

- Java-Code:

```
String sql = "Select model, SUM(quantity) FROM pcshop.sales GROUP BY model";
PreparedStatement pstmt = conn.createStatement(sql);
ResultSet totalSales = pstmt.executeQuery();

while(totalsales.next()){
String m = totalsales.getString(1); //gets the value in column 1
int tq = totalsales.getFloat(2); //gets the value in column 2
}
```

Contact details

- If you have any issues throughout the course, contact us via email:
- Tony: tony.garnock-jones@maastrichtuniversity.nl
- Katharina: k.schneider@maastrichtuniversity.nl
- Or send a message on Discord
- (https://discord.gg/Be2KSF8QG6)

Exercises from last lecture

- Q4: Find the customer who bought the most products
- Q5: Find customers who have made purchases on more than one day

Questions?

This was my last lecture in that course

See you hopefully around