K29 Java Slides

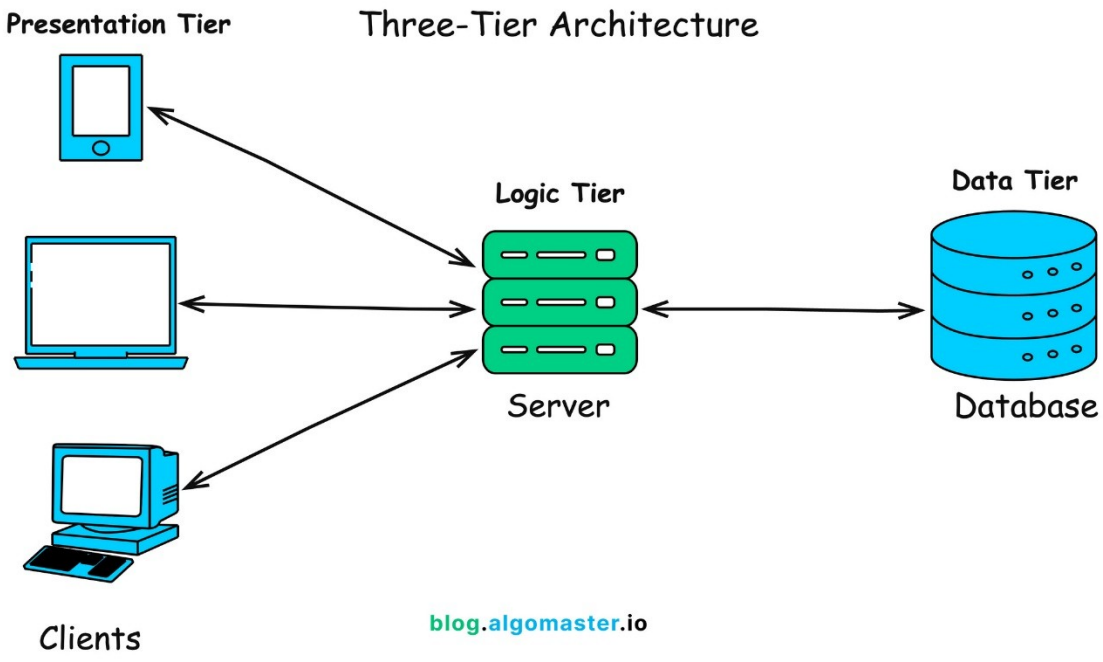
ΣΧΕΔΙΑΣΗ & ΧΡΗΣΗ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ

ΚΟΡΡΕΣ ΜΙΧΑΗΛ

2026

OVERALL SYSTEM ARCHITECTURE – 3-Tier Architecture

- Web Client => HTML,JS,CSS
- Application Server => Java, Python, Go
- DB Server => MySQL, Postgres

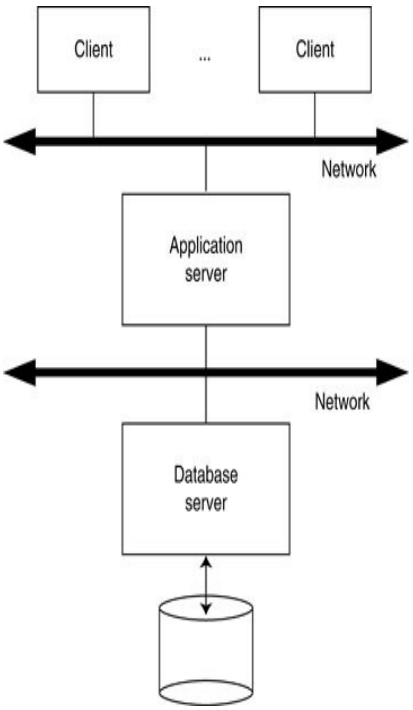


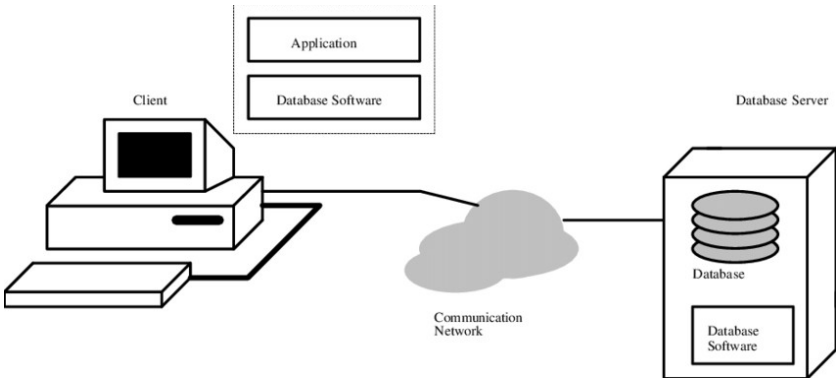
DB Client - Server

SOS: Our *Application Server* acts as a *Database Client!*

Other DB clients include:

- PgAdmin
- DBeaver
- DataGrip





Java

We will not frustrate ourselves with the **best programming language**

Nevertheless…

- **Maven => Build Tool for Java + Applies specific structure**
- **Please use IntelliJ**

A Maven project **structure**:

```
my-app/
├── pom.xml            -> Project Info + Dependencies
└── src/
├── main/
│ ├── java/
│ │ └── com/example/App.java -> A beautiful Java program
```

A **minimal *pom.xml:***

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0 </version>
</project>
```

Let’s add JDBC dependency (Java Database Client)

```
<!-- Example for MySQL -->
<dependency>
  <groupId>com.mysql</groupId>
  <artifactId>mysql-connector-j</artifactId>
  <version>8.3.0</version>
</dependency>
```

JDBC

Alright, we are currently set up.

Now, our **goal** is to **execute queries to a DB.**

In order **to interact with the DB**, any kind of client (Java or Python, GUI or terminal or code, etc.), **must connect to it!**

So, we have this:

```
Connection conn = DriverManager.getConnection(
  "jdbc:mysql://localhost:3306/mydb",
  "username",
  "password"
);
```

This Java code creates a **Connection object** (particularly conn), which is **our interface with the db!**

But what did we pass as arguments ??

1. “jdbc:mysql://localhost:3306/mydb”
    - This seems like a URL
    - **localhost:3306 =>** *localhost* is a way to *refer to our machine. 3306* is *port*. S, something is running on that port in our machine and we want to interact with it.
    - **jdbc:mysql =>** The JDBC URL scheme. As we can see, we communicate with a *MySQL* database. **Its server in particular!**
    - **mydb =>** This is the database in which we want to connect to. The **DB server** running on the respective port of our machine (or a remote machine), **manages multiple databases.**

2. "username"
3. “password
    - Both username & password refer to credentials needed to connect to the db.
    - The **credentials correspond to a DB user/admin**, not an application user !
    - They are stored in a particular table managed from the DBMS.
    - Usually, a default user already exists, but we can also create ours.

Query Execution

Each time we want to send a query to the db, we have to create some kind of **statement**

There are multiple flavors:

- **Statement → simple SQL**
- PreparedStatement → precompiled, parameterized SQL
- CallableStatement → stored procedures

We will stick to a simple **Statement.**

```
Statement stmt = conn.createStatement();
```

Think of **Statement** as the **execution strategy we want to follow for our query.**

Now that we have a Statement, we may as well a execute a query:

ResultSet rs = stmt.executeQuery("SELECT * FROM users");

As we can see, the returned structure is a special object called **ResultSet,** which, as its name implies, **contains the results of our executed query**.

The underlying ResultSet implementation is considered unknown to us.

Nevertheless, to access the records contained inside it, we **access it sequentially,** in a pattern that reminds us a **linked list.**

while (rs.next()) {

System.out.println(rs.getString("name"));

}

Finally, as **top-tier C practicioners, we CLOSE THE RESOURCES THAT WE FINISHED USING!**

rs.close();

stmt.close();

conn.close();

The Spring Framework

Alright, now that we mastered the power of **Java & JDBC,** let's have a look on something more **industry-oriented,** more **abstract** and more "**convention over configuration".**

The Spring framework umbrella is quite wide !

It contains implementations for making Java developer lives easier (probably), that tend to follow uniform principles and aim to provide a "batteries included framework", meaning equip Java developers with all necessary libraries for production backend systems.

BUT WHAT IS A FRAMEWORK ???

A framework is just a library per se, but **strongly opinionated** and which **applies a particular project structure** that every user must follow !

Spring JPA

Now, an important piece of the Spring umbrella is **the Spring Java Persistence API**

Hmmmmm....

Persistence => "Save permanently" => Secondary mem (Hard Drive)

But what is persisted on the disk other than files ? **DB RECORDS !!!**

Spring JPA uses the Hibernate ORM underneath to provide ready to use methods and an easy to use application API to communicate with the DB, execute queries, etc.

Feature	JDBC	JPA (Spring Boot)
Level	Low-level	High-level (ORM)
SQL	Manual	Optional
Boilerplate	High	Low
Control	Full	Abstracted
Learning curve	Easier start	Concept heavier