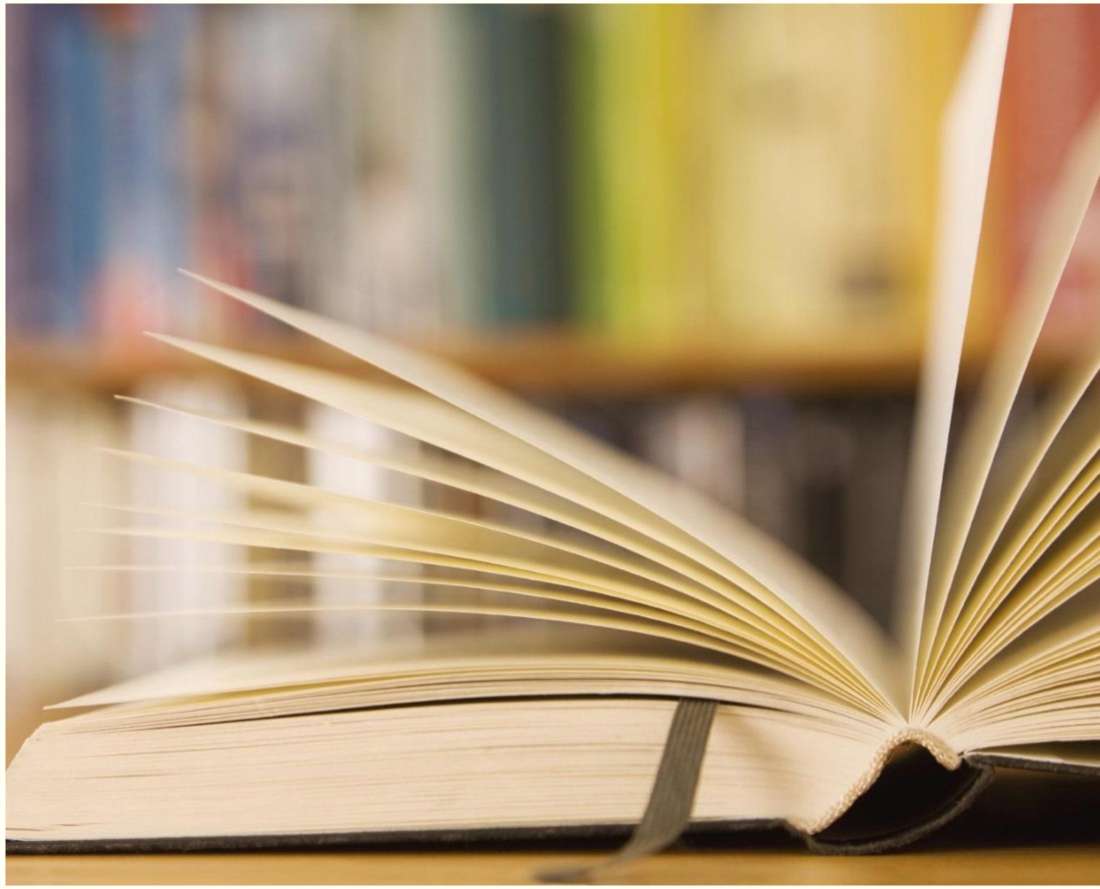
# ENTERPRISE ARCHITECTURE

Najeeb Najeeb, PhD

Version 2.1 ©2022



## LESSON 03 DYNAMIC ASPECTS OF PERSISTENCE

## The problem

- When data is stored in RDBMS, data manipulation requires learning and using SQL.
- Writing SQL queries requires the full knowledge of the DB schema.
- Changes to DB schema results in changes to relevant SQL queries.
- Complex queries, involving several entitles, may result in SQL queries that are hard to understand and maintain.
- SQL is vulnerable to:
  - Injection
  - Runtime errors due to typos in the queries.
- Any solution should still enable backwards compatibility and query optimization.

## JPQL

## Jakarta Persistence Query Language

- JPQL syntax is very similar to SQL syntax.
- What is the advantage of having similar syntax?
- SQL retrieves result set from database, JPQL retrieves entity object(s).
- Typical JPQL structure
  - SELECT … FROM … [WHERE …] [GROUP BY … [HAVING …]] [ORDER BY …]
- DELETE structure
  - DELETE FROM … [WHERE …]
- UPDATE structure
  - UPDATE … SET … [WHERE …]

## DYNAMIC QUERIES

## Java Persistence Query Language

- What is wrong with a normal query?
- What could go wrong when not using a Typed Query?
- Why do we have a non-typed query?
- What are disadvantages of using dynamic queries?

```
String queryStr = "SELECT s FROM Student s";

Query query = em.createQuery(queryStr);

List<Student> students = query.getResultList();


TypedQuery<Student> query = em.createQuery(queryStr, Student.class);

List<Student> result = query.getResultList();

Stream<Student> students
   = result.getResultStream();
```

## Parameters with Dynamic Queries

- What does this help prevent?
- When is this query going to be converted to SQL?
- What could go wrong?

```
String queryStr = "SELECT s FROM
Student s WHERE s.gpa = :gpa";

TypedQuery<Student> query
= em.createQuery(queryStr, Student.class)
;

query.setParameter("gpa", 3.0);

List<Student>
   students = query.getResultList();
```

## Main Point

- When working with SQL one must sometimes issue statements in certain order due to how data is stored and related. In JPQL the data dependencies are managed by the ORM, and the developer just focuses on writing the queries. The ORM will issue queries in the proper order or may not issue them at all.
- The highest first principle enables us to focus on what is important and everything else is taken care of automatically. Water the root and enjoy the fruit.