The scripts folder contains the SQL script for creating the database, tables and some initial dummy data. The query folder contains the script for each query. The query results folder contains screenshots documenting the results. Functions folder contains the 2 functions and a sub folder documenting the results. The Trigger folder contains 2 sub folders, “after insert” and “before insert”, each containing the scripts and documentation. The stored procedures folder contains the scripts for the 3 stored procedures and a sub folder documenting the results. The using\_db\_in\_app folder contains documentation for the implementation. Finally, a copy of the project is included and is named names esports1.

**Overview of SQL Tools and Techniques Used and available**

I used stored procedures as required. I used the Spring data JPA.Spring Data JPA is a Spring module that simplifies the integration of JPA into your Spring application by providing a repository layer. Spring Data JPA provides repositoryinterfaces that extend *JpaRepository*, and Spring automatically implements these methods at runtime.

**1. Stored Procedures (Database-Driven Approach)**

A **stored procedure** is a precompiled SQL script stored in the database and executed directly by the database engine.

**Example: Stored Procedure for Joining a Tournament**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**2. JPA (Spring Data JPA)**

JPA (Java Persistence API) is an **ORM (Object-Relational Mapping)** framework that abstracts database interactions using entity classes.

**3. JDBC with PreparedStatement (Low-Level Database Access)**

JDBC (Java Database Connectivity) is a **low-level API** for executing SQL queries manually.

**4. Spring JDBC Template (Simplified JDBC)**

Spring’s JdbcTemplate simplifies JDBC by managing connections and reducing boilerplate code.

**5. EntityManager (JPA Alternative for Custom Queries)**

EntityManager is a JPA component that allows **manual query execution** while still leveraging JPA

| **Approach** | **Performance** | **Ease of Use** | **Database Independence** | **Best For** |
| --- | --- | --- | --- | --- |
| **Stored Procedures** | ✅ High | ❌ Hard | ❌ No | Complex, DB-intensive tasks |
| **JPA Repository** | ⚠️ Moderate | ✅ Easy | ✅ Yes | General CRUD operations |
| **JDBC (PreparedStatement)** | ✅ High | ❌ Hard | ⚠️ Partial | Low-level DB access |
| **Spring JDBC Template** | ✅ High | ✅ Easy | ⚠️ Partial | Simpler JDBC operations |
| **EntityManager** | ⚠️ Moderate | ⚠️ Medium | ✅ Yes | Custom JPA queries |

**1. Data Structure**

* **SQL (Relational Databases)**:
  + **Schema-based**: SQL databases rely on a **fixed schema** that defines the tables, columns, and types of data stored.
  + **Structured Data**: Best for applications where the data is highly structured and fits into a table with relationships (i.e., rows and columns).
  + **Relationships**: Strong support for **JOIN operations** to relate different tables using **foreign keys**.
* **NoSQL (Non-Relational Databases)**:
  + **Schema-less**: NoSQL databases are **schema-flexible**, allowing data to be stored in a more **dynamic and unstructured format** (e.g., key-value, document, column-family, or graph).
  + **Document-Oriented** (MongoDB): Data is stored in **JSON-like documents**, allowing for **nested data**.
  + **No Complex Joins**: NoSQL databases typically avoid complex joins, but they may **denormalize** data (i.e., store related data in the same document) to improve read performance.