**What gets persisted to a database, classes or objects? Why is this needed?**

**Objects** are persisted to a database, not classes. Classes are blueprints or templates that define the structure and behavior of objects. Objects are instances of classes that hold actual data. Persisting objects is necessary because:

* **Data Storage**: Objects contain the actual data that needs to be stored for later retrieval.
* **State Management**: Persisting objects allows the application to save the state of an object at a particular point in time.
* **Persistence Across Sessions**: Data needs to be stored in a database so that it can be retrieved and used across different application sessions or even after the application has been restarted.

**Explain the role of domain classes in an MVC architecture.**

In an **MVC (Model-View-Controller)** architecture, **domain classes** are part of the **Model** layer. The Model represents the application's data and business logic. Domain classes are crucial because:

* **Data Representation**: They represent the core entities and their relationships in the application (e.g., Student, Course).
* **Business Logic**: They encapsulate the rules and logic that govern how data is manipulated and validated.
* **Separation of Concerns**: By separating the domain logic (Model) from the presentation (View) and user interaction (Controller), the application becomes more modular, maintainable, and scalable.

**UML Class Diagram for Student Microservice**

Based on the example provided, here is a simple UML class diagram for the Student and Course domain model:

plaintext

Copy

+----------------+ +----------------+

| Student | | Course |

+----------------+ +----------------+

| - studentId: Long | | - courseId: Long |

| - name: String | | - title: String |

| - email: String | | - credits: int |

+----------------+ +----------------+

| + enroll(course: Course) | + addStudent(student: Student) |

+----------------+ +----------------+

| |

| |

+--------------------------+

|

v

+----------------+

| Enrollment |

+----------------+

| - studentId: Long |

| - courseId: Long |

| - enrollmentDate: Date |

+----------------+

* A Student can enroll in many Courses.
* A Course can have many Students.
* The Enrollment class represents the many-to-many relationship between Student and Course.

**Object-Relational Mapping (ORM)**

**What are the advantages of using an ORM instead of embedding SQL statements in the code?**

1. **Abstraction**: ORM abstracts the database interactions, allowing developers to work with objects instead of writing raw SQL queries.
2. **Productivity**: Reduces boilerplate code and speeds up development by automating CRUD operations.
3. **Database Agnostic**: ORMs often support multiple databases, making it easier to switch databases without rewriting queries.
4. **Type Safety**: ORMs provide type-safe queries, reducing runtime errors.
5. **Maintainability**: Code is cleaner and easier to maintain since database logic is centralized in the ORM layer.
6. **Security**: ORMs help prevent SQL injection attacks by using parameterized queries.

**Database Configuration and Verification**

**Would it be prudent to use the H2 database in a production setting? Why or why not?**

No, it is generally not prudent to use the **H2 database** in a production setting because:

* **In-Memory Database**: H2 is primarily designed for development and testing purposes. It is often used as an in-memory database, which means data is lost when the application restarts.
* **Scalability**: H2 is not designed to handle large-scale production workloads or high concurrency.
* **Persistence**: While H2 can persist data to disk, it lacks the robustness, reliability, and advanced features (e.g., replication, clustering) offered by production-grade databases like PostgreSQL, MySQL, or Oracle.
* **Support**: Production databases come with enterprise-level support and community resources, which are critical for production environments.

**Demonstrate that your records have been inserted in the correct database table.**

To verify that records have been inserted correctly, you can:

1. Query the database directly using a database client or command-line tool.
2. Use an ORM's built-in methods to retrieve and display records.

Example (using SQL):

sql

Copy

SELECT \* FROM Student;

SELECT \* FROM Course;

SELECT \* FROM Enrollment;

**Demonstrate that the database schema has been created correctly.**

To verify the schema, you can:

1. Use a database client to inspect the table structure.
2. Run a DESCRIBE or SHOW CREATE TABLE command.

Example (using SQL):

sql

Copy

DESCRIBE Student;

DESCRIBE Course;

DESCRIBE Enrollment;

**Add screenshots showing that your sample data has been inserted in the correct tables.**

(Note: Screenshots cannot be provided here, but you can take screenshots of your database client showing the tables and their data.)

**Generate (or manually draw) an Entity Relationship Diagram (ERD) for your application.**

Here is a textual representation of the ERD for the Student and Course microservice:

plaintext

Copy

+----------------+ +----------------+

| Student | | Course |

+----------------+ +----------------+

| - studentId (PK)| | - courseId (PK) |

| - name | | - title |

| - email | | - credits |

+----------------+ +----------------+

| |

| |

+--------------------------+

|

v

+----------------+

| Enrollment |

+----------------+

| - studentId (FK)|

| - courseId (FK) |

| - enrollmentDate|

+----------------+

* Student and Course have a many-to-many relationship.
* The Enrollment table acts as a junction table with foreign keys (studentId and courseId) referencing the primary keys of Student and Course.