# AeroAspire - SDE Intern
## Gokul Krishna S

### Week 4 – Day 2 (October 14)

**Questions/Reflections:**

1. What is ORM, what are its advantages & disadvantages?

- ORM (Object-Relational Mapping) is a programming technique that allows developers to interact with a database using object-oriented code instead of writing raw SQL queries. It maps the objects (classes) in your code to corresponding tables in the database. For example, a User class in Python or Java could directly represent a users table in your database.
- ORMs like SQLAlchemy (Python), Hibernate (Java), or Entity Framework (C#) handle most of the data persistence work by translating between object models and database tables.

**Advantages of ORM:**

- Faster development. Eliminates repetitive SQL, speeding up feature creation.
- Database independence. You can switch between MySQL, PostgreSQL, or SQLite with minimal code changes.
- Less boilerplate code. CRUD (Create, Read, Update, Delete) operations become simple method calls.
- Improved security. Built-in protection against SQL injection via automatic parameterization.
- Easier maintenance. The code is more readable and consistent with object-oriented design.

**Disadvantages of ORM:**

- Performance overhead. Automatically generated queries can be slower than hand-written SQL.
- Limited control. Developers have less insight into low-level SQL optimizations.

- Learning curve. Understanding relations, migrations, and ORM methods takes time.
- Not ideal for complex queries. Large or highly optimized joins can be cumbersome to express.
- Debugging difficulty. ORM abstraction can make troubleshooting SQL-related issues harder.

2. How does parameterized query prevent SQL injection?

- A parameterized query (or prepared statement) separates SQL code from user input. Instead of embedding user data directly into an SQL query string, ORM frameworks or database APIs send the structure of the query and the parameters separately.
- This prevents attackers from injecting malicious SQL statements — because input data is treated strictly as a value, not as executable code.

- Example:

- # Unsafe SQL
  cursor.execute(f"SELECT * FROM users WHERE username = '{username}'")

- # Safe, parameterized query
  cursor.execute("SELECT * FROM users WHERE username = %s", (username,))

- In the safe version, the ORM or database driver ensures that even if username contains SQL keywords (like '; DROP TABLE users;--), those are treated as ordinary text, not commands. This technique effectively mitigates SQL injection attacks.

3. What is the flow from request → ORM / SQL → DB → return result → commit / rollback?

    - **Request received:** The web client (e.g., browser or API call) sends an HTTP request to your server.
    - **Controller or route handler:** The corresponding route in your Flask/Django/Express app gets triggered.
    - **ORM interaction:** The route handler calls ORM methods (e.g., User.query.get(id) or db.session.add(new_user)).
    - **ORM → SQL translation:** The ORM generates the appropriate SQL queries and sends them to the database engine.
    - **Database execution:** The DBMS (like PostgreSQL or MySQL) executes the SQL, returning any result sets.
    - **Result processing:** The ORM converts SQL results into objects (like a User instance) and returns them to your code.
    - **Transaction control:**
      **If the operation succeeds:** ORM calls commit() to save changes.
      **If an error occurs:** ORM calls rollback() to undo partial changes.
    - **Response to client:** The controller serializes the result (e.g., JSON) and sends it back as an HTTP response.