AeroAspire - SDE Intern

Gokul Krishna S

Week 4 – Day 3 (October 15)

Questions/Reflections:

1. How do you design efficient query for filtering? What is index; when do you use it?

    - A filter in SQL narrows down the rows returned from a table. To make filtering efficient, you want to:
    - Apply filters early — fetch only what you need. Let the WHERE clause or subquery do the trimming before any sorting or grouping.
    - Avoid functions on filtered columns. For example, writing WHERE YEAR(hire_date) = 2024 prevents the database from using an index effectively. Instead, use range-based filtering:

    - Sql :

      ```
      SELECT * FROM employees
      WHERE hire_date >= '2024-01-01' AND hire_date < '2025-01-01';
      ```

    - Filter before sorting: if you sort a smaller dataset, you compute less. This can drastically improve query performance.
    - Use equality (=) or range comparisons (>, <) instead of LIKE whenever possible. Comparing indexed columns with %LIKE% forces full scans.
    - **What is an index and when do you use it?**
    - An index is like a look-up table — a sorted data structure (like a book's index) that lets the database find specific rows faster without scanning the entire table.
    - For example, an index on the email column means when you run:

- sql
  SELECT * FROM users WHERE email ='alex@example.com';

- the database jumps straight to matching rows instead of reading every user.
- Use an index when:
- The column is frequently used in WHERE, JOIN, or ORDER BY clauses.
- It contains highly selective values (many unique entries).
- Use with care: indexes speed up reads, but slow down writes (INSERT, UPDATE, DELETE) since the index must update too. Also, indexes consume extra storage.

2. How does pagination work (offset/limit etc.).
   - Pagination is how you fetch results in smaller chunks — common for APIs showing tables or lists.

   - **Basic syntax:**
     SELECT * FROM books ORDER BY published_at DESC LIMIT 10 OFFSET 20;

   - LIMIT: number of rows to return.
   - OFFSET: number of rows to skip first.
   - **For example:**
     Page 1 → LIMIT 10 OFFSET 0
     Page 2 → LIMIT 10 OFFSET 10

   - Performance tip: large offsets are slow because the database still scans skipped rows. An alternative is cursor or keyset pagination:

   - sql
     SELECT * FROM books
     WHERE id > 1000 ORDER BY id ASC LIMIT 10;

   - This uses an indexed column (id) for pagination and avoids skipping.

3. What’s the flow of building these endpoints, receiving query params, applying them in SQL / ORM, returning results.

- When you create an endpoint like /api/products, the backend usually:
- **Receives query parameters from the client:**
  /api/products?category=shirts&limit=10&page=2.
- **Parses and validates them in your backend code:**
  const { category, limit = 10, page = 1 } = req.query;
  const offset = (page - 1) * limit;
- **Builds a parameterized SQL or ORM query:**
  SELECT * FROM products
  WHERE category = $1
  ORDER BY created_at DESC
  LIMIT $2 OFFSET $3;
- **or with an ORM like Sequelize:**
  Product.findAll({
  where: { category },
  order: [['createdAt', 'DESC']],
  limit,
  offset
  });
- Executes the query — the database handles filtering, sorting, and pagination efficiently when columns are properly indexed.
- **Returns a structured response:**
  {
  "page": 2,
  "limit": 10,
  "total": 125,
  "data": [...products]
  }