1. What is the difference between WHERE and HAVING?

- **WHERE** → Filters rows **before** grouping (used with raw data).
- **HAVING** → Filters rows **after** grouping (used with aggregate functions like SUM, COUNT).

Example:

```
-- WHERE filters before grouping
SELECT * FROM sales WHERE region = 'East';

-- HAVING filters after grouping
SELECT region, SUM(sales)
FROM sales
GROUP BY region
HAVING SUM(sales) > 1000;
```

---

2. What are the different types of joins?

- **INNER JOIN** → Returns only matching rows from both tables.
- **LEFT JOIN** → Returns all rows from the left table + matching rows from the right table.
- **RIGHT JOIN** → Returns all rows from the right table + matching rows from the left table.
- **FULL JOIN** → Returns all rows from both tables (matches + non-matches).
- **CROSS JOIN** → Combines every row of one table with every row of another (Cartesian product).

---

3. How do you calculate average revenue per user in SQL?
Divide total revenue by the number of unique users.

```
SELECT SUM(revenue) / COUNT(DISTINCT user_id) AS avg_revenue_per_user
FROM sales;
```

---

4. What are subqueries?
A query inside another query.

- Used to get intermediate results for the main query.
- Can be in `SELECT`, `FROM`, or `WHERE` clauses.

Example:

```
SELECT *

FROM sales
WHERE revenue > (SELECT AVG(revenue) FROM sales);
```

---

5. How do you optimize a SQL query?

- Use **indexes** on frequently searched columns.
- Avoid `SELECT *` — only select needed columns.
- Use proper **JOINs** instead of subqueries when possible.
- Use **LIMIT** to avoid unnecessary data loading.
- Analyze queries using `EXPLAIN` to see performance.

---

6. What is a view in SQL?

- A **view** is a saved SQL query that acts like a virtual table.
- It doesn't store data itself but fetches it when used.
- Helpful for simplifying complex queries and improving security.

Example:

```
CREATE VIEW top_customers AS
SELECT customer_id, SUM(sales) AS total_sales
FROM sales
GROUP BY customer_id
HAVING total_sales > 5000;
```

---

7. How would you handle null values in SQL?

- Use `IS NULL` or `IS NOT NULL` to check null values.
- Use `COALESCE()` to replace nulls with a default value.
- Use `NULLIF()` to return null if two values are the same.

Example:

```
SELECT COALESCE(discount, 0) AS discount_value
FROM sales;
```