ChatGPT

10-Day Interactive SQL Plan

Overview

This interactive plan is designed for a **Senior Data Engineer** to master SQL for interviews. Each day contains practical tasks. Check off each task as you finish. Use any relational database (PostgreSQL, MySQL, or SQLite) and a realistic dataset (you can create your own or use open data). Keep notes on what you learn.

How to use this plan

- **Work in your own database.** Create a sandbox database and tables to experiment with. Keep your test data separate from production data.
- **Complete all tasks.** Each day lists tasks (`[ ]`) you should complete. Check them off when done.
- **Take notes.** Make notes about errors or insights to review later.
- **Discuss with peers.** Share questions and insights with teammates; teaching others solidifies your knowledge.

Day 1 – Build Your Data Model

Goals

- Set up your database and initial schema.
- Practice creating tables and enforcing data integrity.

Tasks

1. `[ ]` **Set up a database environment.** Install PostgreSQL or MySQL locally or use an online SQL playground. Create a new database named `interview_practice`.
2. `[ ]` **Create core tables.** Write `CREATE TABLE` statements for at least these tables:
3. `employees` (id, name, department_id, manager_id, hire_date, salary)
4. `departments` (id, name, budget)
5. `customers` (id, name, email, signup_date)
6. `orders` (id, customer_id, order_date, amount)
7. `[ ]` **Add primary and foreign keys** to enforce relationships:
8. `employees.department_id` references `departments.id` [1].
9. `employees.manager_id` references `employees.id` (allow NULL for top management).
10. `orders.customer_id` references `customers.id`.
11. `[ ]` **Add constraints**:
12. `customers.email` must be unique [2].
13. `employees.salary` must be greater than zero (CHECK constraint).
14. `orders.amount` must not be null and must be positive.

15. `[ ]` **Insert test data** (about 20–30 rows per table). Use realistic values (e.g., salaries, dates). Commit your work.

Day 2 – CRUD and Simple Queries

Goals

- Practice inserting, updating, and deleting data.
- Write simple `SELECT` queries with filtering and sorting.

Tasks

1. `[ ]` **Insert additional data** into your tables, including edge cases (e.g., a department with no employees).
2. `[ ]` **Update data**: give a 10 % salary raise to employees in a specific department. Record the before/after values.
3. `[ ]` **Delete data**: remove an order and observe how foreign key constraints work (does it cascade? does it fail?). Adjust constraints if needed.
4. `[ ]` **Write and execute queries**:
5. Retrieve all employees sorted by hire date.
6. Filter customers whose email contains `@gmail.com` using `WHERE email LIKE '%@gmail.com%'`.
7. Select the top 5 highest paid employees.
8. `[ ]` **Write queries with** `CASE`: classify employees into salary bands (e.g., low, medium, high) using a `CASE WHEN` expression.
9. `[ ]` **Practice** `DISTINCT`: list distinct domains from the `customers.email` column.

Day 3 – Aggregations & Grouping

Goals

- Use aggregate functions and `GROUP BY` to answer business questions.
- Understand the difference between `WHERE` and `HAVING` [3].

Tasks

1. `[ ]` **Total sales per customer.** Write a query that sums `orders.amount` grouped by `customer_id` and join it with `customers` to display names.
2. `[ ]` **Average salary per department.** Group employees by department and calculate the average salary.
3. `[ ]` **Departments with total salary > $300,000.** Use `HAVING` to filter groups after aggregation.
4. `[ ]` **Count orders per month.** Extract the year and month from `order_date` and count the number of orders for each month. Use `ORDER BY` to sort chronologically.

5. [ ] **Find duplicates.** Identify any duplicate emails in the `customers` table using `GROUP BY email HAVING COUNT(*) > 1`.

Day 4 – Joins Deep Dive

Goals

- Master different join types.
- Learn to query across multiple tables.

Tasks

1. [ ] **Inner join** employees with departments to list each employee's department name.
2. [ ] **Left join** customers with orders to list all customers and their order totals (NULL for customers with no orders).
3. [ ] **Right or full join** (if supported) to list all departments and their employees, including departments with no employees. [4]
4. [ ] **Cross join**: create a calendar table (dates for a month) and cross join it with a small product list to generate a schedule (shows Cartesian product) [5].
5. [ ] **Self join**: write a query that pairs each employee with their manager using `employees` joined to itself [6]. Display employee and manager names.
6. [ ] **Join conditions vs filters**: compare placing conditions in the `ON` clause vs the `WHERE` clause and observe differences.

Day 5 – Subqueries & Set Operations

Goals

- Use subqueries (both correlated and non-correlated).
- Perform set operations (`UNION`, `INTERSECT`, `EXCEPT`).

Tasks

1. [ ] **Non-correlated subqueries**: list customers with total orders > $1,000. Use a subquery that sums orders per customer.
2. [ ] **Correlated subquery**: find employees whose salary is above the average salary of their department [7].
3. [ ] **Use** `EXISTS` **vs** `IN`: write queries that check if a customer has placed any order using both approaches. Compare performance on large data sets.
4. [ ] **CTEs**: rewrite one of the subquery tasks using a Common Table Expression (`WITH` clause). CTEs improve readability [8].
5. [ ] **Set operations**:
6. Create two small tables (e.g., `new_customers` and `vip_customers`).

7. Combine them with `UNION` and `UNION ALL`. Demonstrate how `UNION` removes duplicates while `UNION ALL` keeps them.
8. Find common elements using `INTERSECT`.
9. Find customers who are in `new_customers` but not in `vip_customers` using `EXCEPT` [9].

Day 6 – Window Functions

Goals

- Use window functions to compute rankings, running totals, and comparisons.

Tasks

1. `[ ]` **Running totals.** For each customer, compute a running sum of their order amounts ordered by `order_date` using `SUM(amount) OVER (PARTITION BY customer_id ORDER BY order_date)` [10].
2. `[ ]` **Ranking functions**: use `ROW_NUMBER`, `RANK`, and `DENSE_RANK` to rank employees by salary within each department. Understand how ties are handled differently [11].
3. `[ ]` **Lead/Lag**: compute the difference between an order's amount and the previous order amount per customer using `LAG(amount) OVER ...`.
4. `[ ]` **Moving average.** Calculate a three-month moving average of sales per customer.
5. `[ ]` **Partition vs no partition**: compare results of window functions with and without `PARTITION BY`.

Day 7 – Advanced CTEs & Data Manipulation

Goals

- Use CTEs to simplify complex queries.
- Practice inserting/updating via CTEs.

Tasks

1. `[ ]` **Recursive CTE**: create a small hierarchy table (e.g., categories with parent_id). Write a recursive CTE to traverse the hierarchy.
2. `[ ]` **Chain CTEs**: build a multi-step query using multiple CTEs (e.g., a CTE to filter orders, another to calculate totals, and a final one to join with customers).
3. `[ ]` **Merge statement (if supported)**: perform an upsert using `MERGE` or `INSERT ... ON CONFLICT`.
4. `[ ]` **Delete with CTE**: use a CTE to identify records to delete (e.g., orphaned orders) and delete them in one statement.

Day 8 – Stored Procedures, Functions & Triggers

Goals

- Encapsulate logic inside stored procedures and functions.
- Use triggers for automatic actions.

Tasks

1. [ ] **Write a stored procedure** that adds a new order and updates the customer's last_order_date. Pass customer_id and order_amount as parameters. Use transactions inside the procedure.
2. [ ] **Create a scalar function** that calculates annual bonus as 10 % of salary. Use this function in a query to display each employee's bonus.
3. [ ] **Create a BEFORE UPDATE trigger** on the `employees` table that logs old salary values into an audit table before any salary change [12].
4. [ ] **Create an AFTER INSERT trigger** that updates a summary table whenever a new order is inserted (e.g., updates total_sales per customer).
5. [ ] **Compare triggers vs procedures**: discuss when you would use each. Note that procedures must be called explicitly while triggers fire automatically.

Day 9 – Transactions & Concurrency

Goals

- Understand ACID properties and control transactions. [13] [14] [15]
- Explore isolation levels and locking behavior.

Tasks

1. [ ] **ACID practice**: write a script that transfers funds from one account to another. Test failure scenarios and ensure atomicity and consistency.
2. [ ] **Use transactions**: wrap multiple `INSERT` statements in a `BEGIN…COMMIT` block. Intentionally trigger an error and roll back.
3. [ ] **Savepoints**: create a savepoint within a transaction and roll back to it while retaining earlier changes.
4. [ ] **Isolation levels**: in two separate sessions, run concurrent transactions that update the same data. Test `READ COMMITTED`, `REPEATABLE READ` and `SERIALIZABLE` levels. Observe anomalies like phantom reads or deadlocks.
5. [ ] **Deadlock simulation**: deliberately create a deadlock by having two transactions lock resources in opposite order. Observe how the DBMS resolves it.

Day 10 – Indexing, Performance & Data Modeling

Goals

- Improve query performance with indexes.
- Design schemas for analytical workloads.

Tasks

1. `[ ]` **Create indexes** on frequently queried columns (e.g., `orders.customer_id`, `employees.department_id`). Explain the difference between clustered and non-clustered indexes (a clustered index orders the table physically [16]; non-clustered indexes maintain a separate structure with pointers [17]).
2. `[ ]` **Test performance**: compare execution times with and without indexes using `EXPLAIN` or `EXPLAIN ANALYZE`.
3. `[ ]` **Composite index**: create a multi-column index (e.g., on `orders` for `(customer_id, order_date)`) and show how it speeds up queries.
4. `[ ]` **Optimize queries**: rewrite any slow queries to avoid `SELECT *` and use `WHERE` clauses that filter on indexed columns [18]. Use `LIMIT` to restrict result size [19].
5. `[ ]` **Star vs snowflake schema**: design a small data warehouse for sales. Use a star schema (single fact table with denormalized dimensions) and then redesign it as a snowflake schema (normalized dimensions). Discuss advantages and trade-offs [20].
6. `[ ]` **Partitioning**: if your DB supports it, partition a large table by date (range partition) or by hash of customer_id. Query individual partitions and compare performance.

After 10 Days

- Review your notes and highlight topics you found challenging.
- Re-run tasks you struggled with; refine your queries.
- Apply these skills to real interview questions from sites like LeetCode or Hackerrank.
- Continue experimenting with new DBMS features (JSON columns, full-text search, window frames, pivot/unpivot).

---

1 3 4 100+ SQL Interview Questions and Answers (2025)
https://www.wecreateproblems.com/interview-questions/sql-interview-questions

2 SQL | UNIQUE Constraint - GeeksforGeeks
https://www.geeksforgeeks.org/sql/sql-unique-constraint/

5 Difference between Natural join and Cross join in SQL - GeeksforGeeks
https://www.geeksforgeeks.org/dbms/difference-between-natural-join-and-cross-join-in-sql/

6 SQL Self Join - GeeksforGeeks
https://www.geeksforgeeks.org/sql/sql-self-join/

7 SQL Correlated Subqueries - GeeksforGeeks
https://www.geeksforgeeks.org/sql/sql-correlated-subqueries/

8 SQL WITH Clause - GeeksforGeeks
https://www.geeksforgeeks.org/sql/sql-with-clause/

9 MySQL :: MySQL 8.4 Reference Manual :: 15.2.14 Set Operations with UNION, INTERSECT, and EXCEPT
https://dev.mysql.com/doc/refman/8.4/en/set-operations.html

10 11 Window Functions in SQL - GeeksforGeeks
https://www.geeksforgeeks.org/sql/window-functions-in-sql/

12 Different types of MySQL Triggers (with examples) - GeeksforGeeks
https://www.geeksforgeeks.org/dbms/different-types-of-mysql-triggers-with-examples/

13 14 15 ACID Properties in DBMS - GeeksforGeeks
https://www.geeksforgeeks.org/dbms/acid-properties-in-dbms/

16 17 Clustered and Non-Clustered Indexing - GeeksforGeeks
https://www.geeksforgeeks.org/sql/clustered-and-non-clustered-indexing/

18 19 SQL Query Optimizations - GeeksforGeeks
https://www.geeksforgeeks.org/sql/best-practices-for-sql-query-optimizations/

20 Star Schema vs Snowflake Schema in Data Engineering - GeeksforGeeks
https://www.geeksforgeeks.org/data-engineering/star-schema-vs-snowflake-schema-in-data-engineering/