go-to SQL Bible for Placements

✨ *“In every placement drive, there’s one universal language that recruiters trust beyond resumes—SQL. Master it once, and you’ll unlock doors to analytics, software, and data roles everywhere.”*

Introduction to SQL

What is SQL?

SQL (**Structured Query Language**) is the most widely used language to interact with relational databases. It allows you to **create, retrieve, update, and delete (CRUD)** data.
👉 In simple words, SQL is the **bridge between raw data and useful information**.

Example:

```
SELECT name, salary
FROM Employee
WHERE salary > 50000;
```

This query fetches employees with salaries above 50,000 — something you'll often be asked in interviews.

Why SQL Matters in Placements ?

SQL is **not just a technical skill**, it's a **core interview filter** for multiple roles:

- **Analytics & Data Roles** → Business Analysts, Data Analysts, Data Scientists.
- **Software Development** → Backend engineers, full-stack developers.
- **Consulting/Finance Roles** → Big 4, consulting firms where client data analysis is common.
- **Product Companies** → Understanding how products generate and use data is crucial.

💡 **Placement Tip:** Recruiters often begin interviews with SQL because:

1. It tests **logical thinking**.
2. It reveals **problem-solving skills**.
3. It applies across multiple domains (not limited to coding-heavy roles).

Popular Companies that Ask SQL

Global Tech Giants (FAANG & Tier-1):

- Facebook (Meta) – Product analyst, data scientist roles.
- Amazon – Business intelligence, data engineer interviews.
- Apple – Data analytics and system design roles.
- Netflix – Data-heavy product & recommendation system teams.
- Google – Data analysis + backend roles.

Consulting & Big 4:

- Deloitte, EY, PwC, KPMG → Use SQL in audit, risk, and analytics consulting interviews.

Indian Product/Startup Ecosystem:

- Flipkart, Zomato, Swiggy, Paytm, Razorpay, Ola, PhonePe → Strong SQL focus for analyst + product engineering roles.
- Byju's, Unacademy → SQL in EdTech for tracking student/product data.

Financial Services:

- Goldman Sachs, JP Morgan, Barclays, Macquarie → SQL in risk analytics and finance data pipelines.

💡 **Placement Tip:** If your goal is a **top product or finance company in India**, SQL will 100% appear in your interview rounds.

Types of SQL Databases

MySQL – Popular open-source DB, widely used in startups & web apps.
Common in e-commerce dataset questions.

PostgreSQL – Feature-rich open-source DB, handles complex queries well.
Preferred by data-heavy startups.

MS SQL Server – Enterprise-focused with Excel/BI integration.
Common in service-based firms & IT MNCs.

Oracle Database – Enterprise-grade, reliable & secure.
Heavily used in banking, telecom, and government roles.

SQL Basics (Beginner Level)

Database Fundamentals & Data Types

- **Database** → A structured collection of data.
- **Table** → Stores data in rows (records) and columns (fields).
- **Primary Key** → Unique identifier for each record.
- **Foreign Key** → Links tables together.

Common Data Types:

- **INT / DECIMAL** → Numbers
- **VARCHAR / TEXT** → Strings
- **DATE / TIME / TIMESTAMP** → Dates and times
- **BOOLEAN** → True/False values

💡 **Placement Tip:** Questions often begin with *"Explain primary vs foreign key"* or *"What are SQL data types?"*.

Basic SELECT Queries

The **SELECT** statement fetches data from a table.

Syntax:

```
SELECT column1, column2
FROM table_name;
```

Example:

```
SELECT name, department
FROM Employee;
```

👉 Returns names and departments of all employees.

Filtering with WHERE, AND, OR, NOT

WHERE adds conditions to queries.

```sql
-- Employees earning more than 50,000
SELECT name, salary
FROM Employee
WHERE salary > 50000;

-- Employees in HR with salary > 40,000
SELECT name
FROM Employee
WHERE department = 'HR' AND salary > 40000;
```

Sorting with ORDER BY

Sort results in **ascending (ASC)** or **descending (DESC)** order.

```sql
-- Highest paid employees first
SELECT name, salary
FROM Employee
ORDER BY salary DESC;
```

Using DISTINCT

Removes duplicates from query results.

```sql
-- Unique departments
SELECT DISTINCT department
FROM Employee;
```

Practice Set 1 – Beginner Level

Assume a table `Employee` with columns:

```
id, name, department, salary, joining_date, job_title
```

1. Show names of all employees in the "IT" department.
2. Fetch all employees with salary less than 30,000.
3. List employees who joined after 2020.
4. Show names and salaries of employees in "Finance" OR "HR."
5. Display all unique job titles.
6. Get employees with salary > 50,000 and working in "Sales."
7. Show employee names in alphabetical order.
8. Fetch the top 5 highest paid employees.
9. List employees who are **not** from the "Marketing" department.
10. Show employee names with salaries between 40,000 and 70,000.

💡 **Placement Tip:** Most **first-round SQL tests** (Flipkart, Deloitte, TCS, Infosys) will ask questions from this level.

Intermediate SQL

Aggregate Functions

Used to perform calculations on groups of data.

Common Functions:

- `COUNT()` → Number of rows
- `SUM()` → Total of values
- `AVG()` → Average
- `MAX()` / `MIN()` → Highest & lowest

Example:

```sql
-- Average salary of employees
SELECT AVG(salary) AS avg_salary
FROM Employee;
```

💡 **Placement Tip:** Be ready for questions like *"Find the highest-paid employee per department."*

GROUP BY and HAVING

- GROUP BY groups rows with same values.
- HAVING filters groups (like WHERE but for aggregates).

Example:

```sql
-- Average salary per department
SELECT department, AVG(salary) AS avg_salary
FROM Employee
GROUP BY department;

-- Departments with average salary > 60,000
SELECT department, AVG(salary) AS avg_salary
FROM Employee
GROUP BY department
HAVING AVG(salary) > 60000;
```

String & Date Functions

- `CONCAT(str1, str2)` → Combine strings.
- `SUBSTR(column, start, length)` → Extract substring.
- `UPPER()/LOWER()` → Convert case.
- `NOW(), DATE(), YEAR(), MONTH()` → Work with dates.

Example:

```sql
-- Full name from first and last name
SELECT CONCAT(first_name, ' ', last_name) AS full_name
FROM Employee;

-- Employees who joined in 2022
SELECT name
FROM Employee
WHERE YEAR(joining_date) = 2022;

-- First 5 letters of employee name
SELECT SUBSTR(name, 1, 5) AS short_name
FROM Employee;

-- Names in uppercase
SELECT UPPER(name) AS name_upper
FROM Employee;

-- Employees who joined this month
SELECT name
FROM Employee
WHERE MONTH(joining_date) = MONTH(NOW());
```

Joins

Used to combine data from multiple tables.

Types of Joins:

- **INNER JOIN** → Matching records from both tables.
- **LEFT JOIN** → All from left table + matches from right.
- **RIGHT JOIN** → All from right table + matches from left.
- **FULL JOIN** → All records, whether matching or not.

Example:

Tables: `Employee` `(id, name, dept_id)` & `Department` `(dept_id, dept_name)`

```sql
-- Employees with their department names
SELECT E.name, D.dept_name
FROM Employee E
INNER JOIN Department D
ON E.dept_id = D.dept_id;

-- All employees (including those without a dept)
SELECT E.name, D.dept_name
FROM Employee E
LEFT JOIN Department D
ON E.dept_id = D.dept_id;

-- All departments (including empty departments)
SELECT E.name, D.dept_name
FROM Employee E
RIGHT JOIN Department D
ON E.dept_id = D.dept_id;
```

💡 **Placement Tip:** *Joins are the #1 topic in SQL interviews.* Practice INNER + LEFT thoroughly.

Subqueries

A query inside another query.

Example (single-row subquery):

```sql
-- Employees earning more than the average salary
SELECT name, salary
FROM Employee
WHERE salary > (SELECT AVG(salary) FROM Employee);
```

Example (correlated subquery):

```sql
-- Employees with max salary in their department
SELECT name, department, salary
FROM Employee E
WHERE salary = (
  SELECT MAX(salary)
  FROM Employee
  WHERE department = E.department
);
```

💡 **Placement Tip:**
Interviewers often test subqueries to check if you understand *when to use them versus JOINs*.

- **Single-row subqueries** (like using `AVG`, `MAX`, `MIN`) show your ability to compare with aggregated values.
- **Correlated subqueries** prove you can handle row-by-row logic (like "max salary *within* each department").
  👉 Practice both, but also be ready to **rewrite a subquery as a JOIN**—that's a common follow-up question in interviews.

Practice Set 2 – Intermediate Level

Assume a table `Employee` with columns:

```
id, name, department, salary, joining_date, job_title
```

1. Find the total number of employees in each department.
2. Get the highest salary from the Employee table.
3. Show departments with more than 10 employees.
4. Find employees who earn above the overall average salary.
5. Display names of employees along with their department names.
6. Get employees who joined in the last 6 months.
7. List all employees who don't have a department assigned (use LEFT JOIN).
8. Find the second-highest salary in the Employee table.
9. Show the number of employees hired per year.
10. Get employees whose salary is equal to the max salary in their department.

💡 **Placement Tip:** Most **case studies in Deloitte, Infosys, TCS, and Amazon** use GROUP BY + Joins + Subqueries.

Advanced SQL (Placement Core)

Subqueries & Nested Queries

- Using queries inside queries (single-row vs multi-row subqueries).

Common examples:

```
-- Find employees earning more than the average salary
SELECT name, salary
FROM Employee
WHERE salary > (SELECT AVG(salary) FROM Employee);

-- Get customers who purchased the same product as Customer_ID =
101
SELECT DISTINCT customer_id
FROM Orders
WHERE product_id IN (
    SELECT product_id
    FROM Orders
    WHERE customer_id = 101
);
```

Window Functions
(# Game-Changer in Placements)

- **ROW_NUMBER(), RANK(), DENSE_RANK()** – ordering and ranking results.

```
-- Rank employees by salary within each department
SELECT
    name, department, salary,
    RANK() OVER (PARTITION BY department ORDER BY salary DESC) AS
dept_rank
FROM Employee;
```

- **LEAD() / LAG()** – comparing current vs previous row.

```sql
-- Compare each month's sales with the previous month
SELECT
    month, sales,
    LAG(sales, 1) OVER (ORDER BY month) AS prev_month_sales
FROM Sales;
```

- **SUM() OVER(), AVG() OVER()** – running totals, moving averages.

```sql
-- Running total of sales
SELECT
    month, sales,
    SUM(sales) OVER (ORDER BY month) AS running_total
FROM Sales;
```

Placement Example:

```sql
-- Find the second highest salary per department
WITH ranked AS (
    SELECT
        department, name, salary,
        DENSE_RANK() OVER (PARTITION BY department ORDER BY salary
DESC) AS rnk
    FROM Employee
)
SELECT * FROM ranked WHERE rnk = 2;

-- Find top 3 earners in each department
SELECT
    department, name, salary,
    ROW_NUMBER() OVER (PARTITION BY department ORDER BY salary
DESC) AS row_num
FROM Employee
WHERE row_num <= 3;
```

Complex Joins & Set Operations

- **Self Joins**

```
-- Employee with their manager name
SELECT e.name AS employee, m.name AS manager FROM Employee e
JOIN Employee m ON e.manager_id = m.emp_id;
```

- **FULL OUTER JOIN**

```
-- Customers who purchased from Store A or Store B
SELECT customer_id FROM Orders WHERE store = 'A'
UNION
SELECT customer_id FROM Orders WHERE store = 'B';

-- Customers who purchased from Store A but not Store B
SELECT customer_id FROM Orders WHERE store = 'A'
EXCEPT
SELECT customer_id FROM Orders WHERE store = 'B';
```

- **UNION, INTERSECT, EXCEPT**

```
-- Find the second highest salary per department
WITH ranked AS (
    SELECT
        department, name, salary,
        DENSE_RANK() OVER (PARTITION BY department ORDER BY salary
DESC) AS rank FROM Employee
)
SELECT * FROM ranked WHERE rank = 2;

-- Find top 3 earners in each department
SELECT
    department, name, salary,
    ROW_NUMBER() OVER (PARTITION BY department ORDER BY salary
DESC) AS row_num
FROM Employee
WHERE row_num <= 3;
```

Case Studies (A Must-Do for Practice)

```sql
-- E-commerce: Top 3 best-selling products per category
SELECT *
FROM (
    SELECT
        category, product_id, SUM(quantity) AS total_sales,
        RANK() OVER (PARTITION BY category ORDER BY SUM(quantity) DESC)
AS rank
    FROM Orders
    GROUP BY category, product_id
) ranked
WHERE rank <= 3;

-- Banking: Detect duplicate transactions
SELECT transaction_id, amount, COUNT(*)
FROM Transactions
GROUP BY transaction_id, amount
HAVING COUNT(*) > 1;

-- HR: Find employees with consecutive absences
SELECT emp_id, date,
       LAG(date,1) OVER (PARTITION BY emp_id ORDER BY date) AS prev_date
FROM Attendance
WHERE status = 'Absent';
```

👉 Placement Tip - By mastering **Window Functions, CTEs, and Optimization**, a candidate can crack **90% of SQL placement rounds**.

Real-World SQL Scenarios

Employee Database Case Study

Concept: Classic company dataset with employees, departments, and salaries.

Sample Tables:

- **Employees** `(emp_id, name, dept_id, salary, join_date)`
- **Departments** `(dept_id, dept_name)`

Queries:

1. Find the highest salary in each department.

```
SELECT d.dept_name, MAX(e.salary) AS highest_salary
FROM Employees e
JOIN Departments d ON e.dept_id = d.dept_id
GROUP BY d.dept_name;
```

2. List employees who joined in the last 1 year.

```
SELECT name, join_date
FROM Employees
WHERE join_date >= DATE_SUB(CURDATE(), INTERVAL 1 YEAR);
```

3. Find departments with more than 5 employees

```
SELECT d.dept_name, COUNT(e.emp_id) AS total_employees
FROM Employees e
JOIN Departments d ON e.dept_id = d.dept_id
GROUP BY d.dept_name
HAVING COUNT(e.emp_id) > 5;
```

Placement Insight: Frequently asked in service-based companies and analytics interviews.

E-commerce Transactions

Concept: Simulating Flipkart/Amazon-style datasets.

Sample Tables:

- **Orders** `(order_id, customer_id, order_date, amount)`
- **Customers** `(customer_id, name, city)`

Queries:

1. Find the top 5 customers by spending.

```
SELECT c.name, SUM(o.amount) AS total_spent
FROM Orders o
JOIN Customers c ON o.customer_id = c.customer_id
GROUP BY c.name
ORDER BY total_spent DESC
LIMIT 5;
```

2. Monthly revenue trend for the last 6 months.

```
SELECT DATE_FORMAT(order_date, '%Y-%m') AS month, SUM(amount) AS
revenue
FROM Orders
WHERE order_date >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH)
GROUP BY month
ORDER BY month;
```

3. Find the city contributing the highest revenue.

```
SELECT c.city, SUM(o.amount) AS total_revenue
FROM Orders o
JOIN Customers c ON o.customer_id = c.customer_id
GROUP BY c.city
ORDER BY total_revenue DESC
LIMIT 1;
```

Placement Insight: Common in product-based startup interviews.

Social Media Analytics

Concept: Twitter/Instagram-like database.

Sample Tables:

- **Posts** `(post_id, user_id, post_date, likes, comments)`
- **Users** `(user_id, username, city)`

Queries:

1. Find the most active users (top 10 by posts).

```sql
SELECT u.username, COUNT(p.post_id) AS post_count
FROM Users u
JOIN Posts p ON u.user_id = p.user_id
GROUP BY u.username
ORDER BY post_count DESC
LIMIT 10;
```

2. Average likes per post in the last 30 days.

```sql
SELECT AVG(likes) AS avg_likes
FROM Posts
WHERE post_date >= DATE_SUB(CURDATE(), INTERVAL 30 DAY);
```

3. Find the top 5 cities by total posts.

```sql
SELECT u.city, COUNT(p.post_id) AS total_posts
FROM Users u
JOIN Posts p ON u.user_id = p.user_id
GROUP BY u.city
ORDER BY total_posts DESC
LIMIT 5;
```

Placement Insight: Used in analytics and data science interviews.

Banking Transactions

Concept: Banking/FinTech dataset.

Sample Tables:

- **Accounts** `(acc_id, customer_id, balance)`
- **Transactions** `(txn_id, acc_id, txn_date, amount, txn_type)`

Queries:

1. Find customers with withdrawals > ₹1,00,000 in the last 3 months.

```
SELECT acc_id, SUM(amount) AS total_withdrawals
FROM Transactions
WHERE txn_type = 'withdrawal'
  AND txn_date >= DATE_SUB(CURDATE(), INTERVAL 3 MONTH)
GROUP BY acc_id
HAVING total_withdrawals > 100000;
```

2. List top 3 customers with the highest account balance.

```
SELECT customer_id, balance
FROM Accounts
ORDER BY balance DESC
LIMIT 3;
```

3. Find the month-wise total deposits in the current year.

```
SELECT DATE_FORMAT(txn_date, '%Y-%m') AS month, SUM(amount) AS
total_deposits
FROM Transactions
WHERE txn_type = 'deposit'
  AND YEAR(txn_date) = YEAR(CURDATE())
GROUP BY month
ORDER BY month;
```

Placement Insight: Common in BFSI and analytics roles.

Placement-Oriented Practice Sets

LeetCode

Here are some medium-to-hard LeetCode SQL problems, many frequently used in technical interviews—complete with their problem numbers and titles:

Referencing lists from GitHub repositories and curated blog summaries:

1. LeetCode 176: Second Highest Salary (Medium) Career Ten
2. LeetCode 177: Nth Highest Salary (Medium) Career Ten
3. LeetCode 178: Rank Scores (Medium) Career Ten
4. LeetCode 182: Duplicate Emails (Easy/Medium) GitHub
5. LeetCode 183: Customers Who Never Order (Easy/Medium) GitHub
6. LeetCode 185: Department Top Three Salaries (Medium) GitHub
7. LeetCode 262: Trips and Users (Hard) GitHub
8. LeetCode 570: Managers with at Least 5 Direct Reports (Medium) GitHub
9. LeetCode 608: Tree Node (Medium) GitHub
10. LeetCode 1045: Customers Who Bought All Products (Medium) GitHub
11. LeetCode 1068: Product Sales Analysis I (Easy/Medium) GitHub
12. LeetCode 1174: Immediate Food Delivery II (Medium) GitHub
13. LeetCode 1193: Monthly Transactions I (Easy/Medium) GitHub
14. LeetCode 1204: Last Person to Fit in the Bus (Medium) GitHub
15. LeetCode 262 (Trips and Users) (Hard) — already listed above.

InterviewBit—SQL Challenge Problems

1. **Nth Highest Salary** – solutions often in InterviewBit's discussions or shared repos.
2. **Deals Away**
3. **Country vs. Continent Statistics**
4. **Most Played Artist**
5. **Daily Transactions Analysis**
6. **Rank Employees by Salary Percentile**
7. **Manager vs Subordinate Tenure Comparison**
8. **Moving Average Revenue**
9. **Frequent Customers**
10. **Inventory Restock Alerts**
11. **Customer Churn Detection**
12. **Category-wise Revenue Growth**
13. **Correlated Subquery for Missing Orders**
14. **Top N per Group**
15. **Flag First Purchase**
16. **Employee Reporting Hierarchy Depth**
17. **Percentile Rank within City**
18. **Cumulative Balance per Account**
19. **Identify Duplicate Records**
20. **Pivot Monthly Sales per Region**

SQL Mock Test (Interview Simulation)

Concept: Mixed practice problems combining different SQL topics (joins, subqueries, window functions, grouping).

Sample Tables:

- **Employees** `(emp_id, name, department, salary, join_date)`
- **Departments** `(dept_id, dept_name)`
- **Orders** `(order_id, customer_id, order_date, amount)`
- **Customers** `(customer_id, name, city)`

Practice Questions (25 Queries):

1. Find employees who earn more than the average salary of their department.

2. List departments where the highest salary is more than ₹80,000.

3. Find the second highest salary in each department.

4. Show top 3 earners in each department using window functions.

5. List employees who joined in the last 2 years but earn above the overall company average.

6. Find the department contributing the highest total salary expense.

7. Identify employees whose salary is above their manager's salary (assume `manager_id` in Employees table).

8. Find customers who have placed more than 5 orders in the last 6 months.

9. Show the city-wise top spender customers (highest order amount total per city).

10. Find the month with the highest revenue in the last 12 months.

11. Find customers who have not placed any order in the last 1 year.

12. List accounts with more than 10 withdrawals in the past 3 months.

13. Show the month-wise total deposits vs withdrawals for the current year.

14. Find the account with the maximum net balance change in the last year.
15. Find customers with multiple accounts having combined balance > ₹5,00,000.
16. Find employees whose salary rank is in the top 10% company-wide.
17. List departments with less than 3 employees earning above ₹70,000.
18. Find customers who ordered products in **every month** of the last 6 months.
19. Find employees who have the same salary as at least one other employee.
20. List the top 5 cities by average order value (AOV = revenue / number of orders).
21. Find the rolling 3-month revenue trend (moving average) for e-commerce orders.
22. List employees who joined before their manager (based on join_date).
23. Find accounts with transactions on at least 20 different days in the past 6 months.
24. Identify customers who spent more than the **average spending of their city**.
25. Show the department-wise highest paid employee(s) using DENSE_RANK().

Placement Insight:

These 25 queries **span all major SQL topics** (Joins, Aggregations, Group By + Having, Subqueries, Correlated Subqueries, Window Functions, Case-based queries, Time-based filtering) — exactly the mix you'll get in **FAANG/product/startup interviews**.

Quick Revision Cheat Sheets

3-Day Pre-Placement SQL Revision Flow

Day 1 – Core Refresh

- **Morning:** Review:
    - Basic SELECT queries, filtering (`WHERE`, `AND`, `OR`, `NOT`)
    - Sorting & deduplication (`ORDER BY`, `DISTINCT`)
- **Afternoon:** Practice:
    - Aggregates (`COUNT`, `SUM`, `AVG`, `MAX`, `MIN`)
    - Grouping and filters (`GROUP BY`, `HAVING`)
- **Evening:** Drill:
    - Joins: INNER, LEFT, RIGHT, FULL
    - Simple subqueries

Day 2 – Advanced Techniques

- **Morning:** Master:
    - Window Functions (`ROW_NUMBER`, `RANK`, `DENSE_RANK`, `LEAD`, `LAG`)
    - Running totals and analytics with `OVER()`
- **Afternoon:** Work through:
    - CTEs (`WITH … AS …`, recursive queries)
    - Complex subqueries and nested queries
- **Evening:** Strengthen:
    - Set operations (`UNION`, `INTERSECT`, `EXCEPT`)
    - Self-joins and hierarchy queries

Day 3 – Real-World Practice & Interview Mindset

- **Morning:** Drill placement cases:
    - E-commerce analytics (top buyers, monthly revenue)
    - Banking/FinTech: large withdrawals, high balances

- **Afternoon:** Simulate interviews:
    - Walkthrough SQL mock test (45-minute timed set; solve and review)

- **Evening:** Final wrap-up:
    - Quick cheat-sheet review
    - Focus on common pitfalls and syntax patterns (e.g., GROUP vs. HAVING, window vs. group logic)
    - Recap frequently used commands

Placement Focus: Quick-Hit Reminders

- **Start Simple:** Always clarify basic asks (select/filter) before moving to complex logic

- **Optimal Joins:** Prefer `JOIN` over nested subqueries when scalable

- **Rank with Purpose:** Use `ROW_NUMBER()` + partition for clean Top-N results

- **Think in Sets:** Translate real-world problems (daily sales, top customers) into `GROUP BY` or window logic

- **Performance Mindset:** Know when to mention indexes or query execution plans

---

Access Full Revision Cheat Sheet

You can access the complete, detailed cheat sheet using the following link:

https://docs.google.com/document/d/1XrdKEvQVTrpBcqcKdRTFJLpEAsWc2Zyf4e7FEo_7hPk/edit?usp=sharing

---

Cheat Sheet Components

Topic	Key Concepts & Commands
Basic Queries	`SELECT`, `WHERE`, `ORDER BY`, `DISTINCT`
Aggregates & Grouping	`COUNT`, `SUM`, `AVG`, `MAX`, `MIN`, `GROUP BY`, `HAVING`
Joins	`INNER`, `LEFT`, `RIGHT`, `FULL`, join conditions
Subqueries	Single-row, multi-row, correlated subqueries
Window Functions	`ROW_NUMBER`, `RANK`, `DENSE_RANK`, `LEAD`, `LAG`, `OVER()`
CTEs	`WITH ... AS (...)`, and recursive CTEs
Set Operations	`UNION`, `INTERSECT`, `EXCEPT`
Advanced Queries	Self-join, hierarchical queries, running totals
Performance Tips	Index basic (`CREATE INDEX`),`EXPLAIN` usage
Common Pitfalls	`GROUP BY` vs `HAVING`; null handling; ambiguous joins

Guru Dakshina: My Last Gift Before Your Placements

Placements are not just about cracking an interview — they are about proving to yourself that the countless nights of learning, the hours spent debugging queries, and the persistence through rejections were worth it.

You will hear stories of students who "just got lucky," but luck is a myth without preparation. When you sit in front of your interviewer, it will not be the last two weeks of rushed practice that speaks for you. It will be the months — sometimes years — of small efforts: writing that extra query, revising one more concept, solving just one more problem even when your eyes begged for sleep.

There will be moments when you'll question yourself. Maybe after a rejection, maybe when your friend gets placed before you, or maybe when the concepts feel like a mountain. That's normal. Every person who has cleared placements has felt the same weight. But here's the secret — they stood up again, revised again, and kept walking.

Placements are not a race against your peers. They are a test of your patience, your resilience, and your self-belief. And if you've reached this note, it means you've already done the hard part — you've prepared. Now, the only thing left is to trust your preparation and trust yourself.

Even if you fail an interview, remember — it's not the end of the road. It's just feedback wrapped in disappointment. Each attempt is a rehearsal for the stage where you'll shine.

I want you to carry this with you: **No placement defines your worth. But your preparation defines your future.** Companies, job roles, salaries — they will come and go. But the skills you've earned, the discipline you've built, and the courage you've shown — those stay with you forever.

As you walk into your interviews, remember to breathe. Smile when you greet. Think before you speak. And above all, let your confidence tell the story your resume cannot.

This is the end of the workbook, but only the beginning of your journey. May you crack not just interviews, but life itself.

With pride in your effort and faith in your future,

– Your Trainer & Guide

Resources & Next Steps

Preparing for placements is not just about learning SQL commands but also practicing real-world problems and revising systematically. Below are structured resources (videos, websites, and practice platforms) to help you master SQL.

Concept Building (Videos & Notes)

- **Apna College SQL Playlist (Beginner-Friendly)**
  Covers basics of SQL from scratch with practical examples, great for quick learning.
  https://www.youtube.com/playlist?list=PLfqMhTWNBTe3LtFWcvwpqTkUSlB32kJop

- **CodeWithHarry SQL Tutorial (Hindi, Beginner to Advanced)**
  Explains SQL concepts in simple terms with hands-on demos.
  https://www.youtube.com/watch?v=hlGoQC332VM

- **FreeCodeCamp SQL Course (Full 4 Hours, English)**
  Detailed beginner-to-intermediate SQL tutorial covering queries, joins, and more.
  https://www.youtube.com/watch?v=HXV3zeQKqGY

- **GeeksforGeeks SQL Notes & Articles**
  Best for structured learning, concept explanations, and interview-oriented questions.
  https://www.geeksforgeeks.org/sql-tutorial/

Practice-Oriented Learning

- **LeetCode SQL Practice Problems**
  Widely used for FAANG & product-based interviews. Contains a mix of easy to hard SQL problems.
  https://leetcode.com/problemset/database/

- **HackerRank SQL Track**
  Beginner-friendly, structured progression with lots of problems for practice.
  https://www.hackerrank.com/domains/sql

- **StrataScratch SQL Interview Questions**
  Real-world company SQL problems (Google, Amazon, Uber, etc.). Excellent for advanced practice.
  https://www.stratascratch.com/sql

Topic-Wise Strategy

- **Basic Queries & Filtering** → Start with **Apna College videos** + **HackerRank SQL Basic section**.

- **Joins & Subqueries** → Practice on **LeetCode Easy/Medium SQL problems**.

- **Aggregation, GROUP BY, HAVING** → Use **CodeWithHarry tutorials** + practice on **HackerRank**.

- **Window Functions & Advanced SQL** → Learn from **FreeCodeCamp video** + practice on **StrataScratch**.

- **Placement Revision** → Revise from **GeeksforGeeks interview questions**:
  https://www.geeksforgeeks.org/sql-interview-questions/

Recommended Revision Flow (Before Placements)

- **2 Weeks Before Placements** → Learn/revise concepts using Apna College, CodeWithHarry, and GFG.

- **1 Week Before Placements** → Solve at least 20-30 LeetCode SQL questions.

- **2-3 Days Before Placements** → Use your **SQL Cheat Sheet + Quick Notes** and solve 5-10 problems daily.

- **Night Before Interview** → Revise queries on Joins, Aggregation, and Window Functions.

Thank You

🙏 A Note of Gratitude

Thank you for walking with me through this journey of SQL and placement preparation. Remember, concepts will fade if not practiced — but discipline and curiosity will always keep you ahead.

Wishing you the very best for your placements — may you crack every challenge with confidence and clarity.

All the best, always!
— Your Trainer & Guru