SQL Interview Questions: Complete Guide

Introduction

SQL (Structured Query Language) is a crucial skill in today's job market. Nearly all major companies like Netflix, Meta, Uber, Google, and Microsoft rely on SQL in some capacity. SQL has become essential for many roles:

- Python developers
- Testers
- Web developers
- Software developers
- Data analysts
- Data engineers
- Data scientists

A solid understanding of SQL will help you navigate interviews and make a strong impression. This guide covers 15 essential SQL interview questions.

Question 1: Difference Between DBMS and RDBMS

DBMS (Database Management System)

Definition: The basic way to store data in files.

Characteristics:

- Data is stored in the form of individual files
- No connection between files
- Information is scattered across separate files

Example: Imagine a teacher storing student information like roll number, name, address, and phone number in separate files with no linkage. To find information about a specific student, you must look through each file one by one, which is very tedious and time-consuming.

RDBMS (Relational Database Management System)

Definition: A database management system with relationships between data.

Characteristics:

- Data is organized in related tables
- Example: MySQL
- Information is interconnected through tables and columns

Example: Instead of separate files, create a table with columns for roll number, name, and address. All student information is stored in one table, making it easy to search for specific students.

Key Advantages of RDBMS

- **Faster data retrieval**: Especially with large datasets
- **Reduced redundancy**: Only one record per student
- **Multiple user support**: Designed to support many users
- **High security**: Built-in security features
- **Optimized for large data**: Better performance with bigger datasets

---

Question 2: Primary Key vs Foreign Key

Primary Key

Definition: A unique identifier for each row in a table.

Characteristics:

- Uniquely identifies rows in a table
- Ensures each row is distinct and cannot be duplicated
- Does not allow empty or null values
- Each record must have a value

Example: In a school, each student has a unique roll number. No two students have the same roll number. This roll number serves as the primary key to identify each student.

Foreign Key

Definition: A field that links tables together by referencing the primary key in another table.

Characteristics:

- Creates relationships between tables
- Points back to the primary key in another table
- Ensures referential integrity

Example: In a school database:

- Student table has roll number as primary key
- Course table contains course information
- To connect which student is enrolled in which course, use the roll number (from the student table) as a foreign key in the course table
- This creates a link between the two tables

Purpose: Foreign keys help you connect data across tables, ensuring that records in one table can reference related information in another table.

Question 3: Constraints and Their Types

Definition: Constraints are rules you set up for data in your tables to keep data accurate and reliable.

Analogy: Filling a Google Form – certain fields are mandatory, some are optional, and some only accept input within a given range.

Types of Constraints

1. NOT NULL

Purpose: Ensures a column cannot be empty.

How it works: If you make a column NOT NULL, the database won't allow you to add data without providing a value for that column.

Example: If you create a schema with a column named `roll_number` as NOT NULL, you cannot add a student without a roll number. The database will reject the entry.

2. UNIQUE

Purpose: Ensures no two values in a column are the same.

How it works: Every value in the column must be distinct.

Example: In a student table, each email must be unique. Students must have distinct email IDs.

3. PRIMARY KEY

Purpose: Combines NOT NULL and UNIQUE constraints.

How it works: There can be only one primary key per table. It uniquely identifies each record.

Example: A student's roll number serves as the primary key.

4. FOREIGN KEY

Purpose: Links tables together.

How it works: Creates a relationship between two tables.

Example: Connecting a student table to a courses table. The foreign key in the courses table references the student ID in the student table.

5. CHECK

Purpose: Sets a condition on acceptable values.

How it works: Enforces a specific rule on data entry.

Example: If you have a voting age column, you can use CHECK to enforce that only ages over 18 are accepted. If someone tries to add a voter younger than 18, the database will reject the entry.

6. DEFAULT

Purpose: Automatically fills in a default value when no value is specified.

How it works: Assigns a preset value to a column if no value is provided during data entry.

Example: Set a default value of "India" for a country column. If you don't specify a country, it automatically fills in "India".

Question 4: DDL and DML Commands in SQL

DDL (Data Definition Language)

Definition: Commands that define the structure of the database.

Purpose: Setting up and organizing the database structure (like building and organizing a house).

Key DDL Commands:

CREATE

- Used to create new tables and other database objects
- Example: `CREATE TABLE employee` will create a new table named employee

ALTER

- Used to modify existing table structures
- Example: Adding a new column to the employee table

DROP

- Used to delete entire tables or other objects
- Example: `DROP TABLE employee` will delete the entire employee table

Important: DDL commands deal with the database structure, not the actual data.

DML (Data Manipulation Language)

Definition: Commands that work with the actual data inside tables.

Purpose: Managing and changing the data within tables (like managing and organizing everything inside the house).

Key DML Commands:

INSERT

- Used to add new rows of data
- Example:

```
INSERT INTO employee (name, position)
VALUES ('Joe', 'Manager')
```

UPDATE

- Used to change existing data
- Example: Updating an employee's position

DELETE

- Used to remove specific data
- Example: Deleting a specific employee record

Important: DML commands work with the actual data content of tables.

Summary

Aspect	DDL	DML
Purpose	Defines database structure	Works with actual data
Focus	Tables and objects	Data content
Example Commands	CREATE, ALTER, DROP	INSERT, UPDATE, DELETE
Analogy	Setting up a house	Managing contents inside a house

Question 5: Difference Between DELETE, DROP, and TRUNCATE

Analogy: Your bookshelf is your database table, and each book represents a row of data. You have three options to clear the bookshelf.

DELETE

Purpose: Removes specific rows based on conditions.

Characteristics:

- Allows you to choose exactly what you want to remove
- Can be rolled back (reversed) if you change your mind
- Uses WHERE clause to specify conditions
- Slower than TRUNCATE (generates individual delete statements)
- Can be used with WHERE clause for conditional deletion

Example: Remove all red-covered books from the shelf or remove books from the top shelf only.

TRUNCATE

Purpose: Quickly clears all data from a table while keeping the structure intact.

Characteristics:

- Removes all rows from the table
- Keeps the table structure (empty shelf remains)
- Very fast operation

- **Cannot be rolled back** – once done, it's permanent
- Does not generate individual delete statements
- Perfect when you don't need any data but want to keep the structure for future use

Example: Clearing all books from a shelf but keeping the shelf itself for future use.

DROP

Purpose: Completely removes the entire table and all its data.

Characteristics:

- Deletes the entire table structure and data
- Removes the table from the database completely
- Permanent and cannot be rolled back/undone
- Removes all books AND the shelf itself
- Only use if you know you won't need the table again in the future

Example: Taking down the entire shelf and removing it from the room. All books are gone, and you're left with just empty space.

Comparison Table

Feature	DELETE	TRUNCATE	DROP
Removes	Specific/all rows	All rows	Entire table
Conditions	Can use WHERE clause	No conditions	N/A
Speed	Slow	Fast	Fast
Rollback	Yes (in most cases)	No	No
Structure	Kept intact	Kept intact	Completely removed
When to use	Remove specific data	Clear all data, keep structure	Remove table permanently

Question 6: GROUP BY vs ORDER BY Clause

Context: You have an employee table with employee data including department and salary.

GROUP BY

Purpose: Organizes data into groups to perform aggregate calculations.

How it works:

- Groups rows with the same values together
- Creates buckets of data based on specified columns
- Used with aggregate functions (SUM, AVG, COUNT, MIN, MAX)
- Shows summarized data, not individual rows

Use Cases:

- Calculate total salary by department
- Find average salary in each department
- Count total employees in each department

Example Query:

```
SELECT Department, AVG(salary)
FROM employee
GROUP BY Department
```

Result: Shows average salary for each department (HR, Sales, IT, etc.) instead of individual employee salaries.

ORDER BY

Purpose: Sorts data in a specific order.

How it works:

- Arranges rows based on one or more columns
- Can sort in ascending or descending order
- Operates on the entire result set
- Does not aggregate data

Use Cases:

- Sort employees by salary (highest to lowest)
- Arrange names alphabetically
- Order by date (newest to oldest)

Example Query:

```
SELECT *
FROM employees
ORDER BY salary DESC
```

Result: Lists all employees sorted by salary from highest to lowest.

Key Differences

Aspect	GROUP BY	ORDER BY
Purpose	Groups data into categories	Sorts/arranges data
Result	Summary/aggregate view	Individual or grouped rows
Functions used	Aggregate functions (SUM, AVG, COUNT)	No specific functions
Output rows	Fewer rows (one per group)	Same number of rows

Aspect	GROUP BY	ORDER BY
Example	Total salary per department	Employees sorted by salary

Summary

- **GROUP BY**: Like grouping your data into meaningful summaries
- **ORDER BY**: Arranges data in the order you want to view it

---

Question 7: WHERE vs HAVING Clause

WHERE Clause

Purpose: Filters individual rows before grouping.

Analogy: Like a watchman at a society gate who checks conditions before anyone enters. The watchman filters based on criteria before people are grouped inside.

When to use: Filter individual rows based on specific conditions like age, name, or department.

Operates on: Individual rows before GROUP BY is applied.

Example Query:

```
SELECT name, age
FROM student
WHERE age >= 18
```

What happens:

1. The database filters out students who don't meet the age requirement
2. Only rows where age is 18 or older are processed
3. Filtering happens before any grouping

Example Result: Shows only students aged 18 or older.

HAVING Clause

Purpose: Filters groups after grouping and aggregation.

Analogy: Like judges in a talent show who review groups of contestants after they've been grouped by score. The judges filter which groups meet their criteria.

When to use: Filter groups created by GROUP BY based on aggregate results like COUNT or SUM.

Operates on: Groups created by GROUP BY, uses aggregate functions.

Example Query:

```

```

```
SELECT age, COUNT(roll_number) as number_of_students
FROM student
GROUP BY age
HAVING COUNT(roll_number) > 1
```

What happens:

1. First, students are grouped by age
2. COUNT is calculated for each age group
3. Only age groups with more than one student are displayed

Example Result: Shows only age groups that have more than one student.

Key Differences

Aspect	WHERE	HAVING
Applied to	Individual rows	Groups of rows
When it works	Before GROUP BY	After GROUP BY
Functions	Cannot use aggregate functions	Uses aggregate functions
Syntax position	Before GROUP BY	After GROUP BY
Example condition	`WHERE age > 18`	`HAVING COUNT(*) > 5`

Summary

- **WHERE**: Filters rows before grouping (like a gate watchman)
- **HAVING**: Filters groups after grouping (like judges reviewing groups)

Question 8: Aggregate Functions in SQL

Definition: Functions that perform calculations on multiple values in a column and return a single result.

Analogy: Like tools for analyzing a basket of fruit – you can count how many you have, find the average weight, identify the heaviest, or find the lightest one.

Types of Aggregate Functions

COUNT

Purpose: Counts the number of rows or non-null values in a column.

Example:

```
COUNT(*)
```

Returns the total number of records.

Example with data: If you have 6 employees, `COUNT(*)` returns 6.

SUM

Purpose: Adds up all values in a numeric column.

Example:

```
SUM(salary)
```

Returns the total salary of all employees.

Example with data: If employee salaries total to 3120, `SUM(salary)` returns 3120.

AVG

Purpose: Calculates the average of a numeric column.

Calculation: Sum of all values divided by the total number of values.

Example:

```
AVG(salary)
```

Returns the average salary of all employees.

MIN

Purpose: Finds the minimum (smallest) value in a column.

Example:

```
MIN(salary)
```

Returns the lowest salary among all employees.

Example with data: If the lowest salary is 43, `MIN(salary)` returns 43.

MAX

Purpose: Finds the maximum (largest) value in a column.

Example:

```
MAX(salary)
```

Returns the highest salary among all employees.

Example with data: If the highest salary is 82, `MAX(salary)` returns 82.

Quick Reference Table

Function	Purpose	Example
COUNT	Count rows/values	`COUNT(*)` → 6
SUM	Add values	`SUM(salary)` → 3120
AVG	Calculate average	`AVG(salary)` → average value
MIN	Find minimum	`MIN(salary)` → 43
MAX	Find maximum	`MAX(salary)` → 82

Question 9: Indexing in SQL and Clustered Index

Understanding Indexing

Analogy: Imagine a book without page numbers. Finding a specific topic would require flipping through every page one by one, which is very slow and tedious.

In databases: Without indexing, searching for specific data (like employee ID 1120) requires the database to flip through every record in the table.

Purpose of Indexing

- **Speeds up searches**: Instead of checking every record, jump directly to relevant data
- **Improves query performance**: Dramatically reduces search time
- **Makes retrieval efficient**: Particularly important for large datasets

Clustered Index

Definition: An index that physically sorts and stores data rows based on the index key.

How it works:

- Arranges papers/data in exact order (like organizing papers by employee ID from smallest to largest)
- The actual data rows are organized in this order
- There can be only one clustered index per table

Process:

1. Data is physically organized in sorted order
2. When searching for employee ID 1120, the database uses the index to skip larger sections
3. It quickly finds the range (1100 to 1200) and jumps directly to that folder

4. No need to check every single record

Benefit: Indexing cuts down search steps significantly, making data retrieval fast and efficient.

Example Scenario

Without Clustering Index:

- Search through all 1 million employee records
- Very slow process

With Clustered Index on Employee ID:

- Database knows employee IDs are sorted
- Jumps to the 1100-1200 range
- Finds employee 1120 quickly
- Much faster retrieval

Question 10: Normalization and Normal Forms

What is Normalization?

Definition: The process of organizing data in a database to improve efficiency and reduce redundancy.

Purpose: Minimizes duplicate information and prevents issues when inserting, deleting, or updating records.

Why Normalization is Important

1. Reduces Duplication:

- Avoids storing the same information multiple times
- Like not keeping duplicate copies of the same book

2. Prevents Anomalies:

- Helps prevent errors with adding, removing, or updating data
- Maintains data consistency

Types of Normal Forms

1NF (First Normal Form)

Rules:

- Each table cell should contain a single value
- Each column must have a unique name
- No duplicate values are present
- Simplifies data retrieval

Example:

- **Wrong**: Storing multiple hobbies in one cell like "Reading, Gaming, Sports"

- **Correct**: Creating separate rows for each hobby

2NF (Second Normal Form)

Rules:

- All non-key attributes must depend on the primary key
- Each piece of data should relate directly to the main identifier (like student ID)
- Not to each other

Example:

- Student ID should directly determine student name
- But teacher's name should not depend on student ID
- Instead, teacher should relate to the class, not the individual student

Purpose: Ensures that data in each column is functionally dependent on the primary key.

3NF (Third Normal Form)

Rules:

- Every non-key attribute must be independent of other non-key attributes
- No information should rely on another piece of non-key data
- Creates separate tables for related but distinct information

Analogy: Like organizing toys so that no toy categories overlap.

Example:

- If student class is linked to a specific teacher
- This information should be in a separate teacher/class table
- Not mixed with the student table

BCNF (Boyce-Codd Normal Form)

Rules:

- Every determinant (attribute that determines other attributes) must be a candidate key
- All dependencies are based solely on unique identifiers
- Ensures no overlapping between identifiers

Example:

- If both student ID and class are potential identifiers
- Each must uniquely define data in its column without overlap

Normal Forms Progression

Normal Form	Key Requirement	Focus
1NF	Single values per cell	Atomic values

Normal Form	Key Requirement	Focus
2NF	Depend on primary key	Primary key dependency
3NF	Independent non-key attributes	Eliminate transitive dependencies
BCNF	Determinants are candidate keys	Remove all anomalies

Question 11: UNION vs UNION ALL

Understanding the Scenario

Example: You have two friend lists:

- **List A (School friends)**: Alice, Bob, Charlie
- **List B (Work friends)**: Bob, David, Eva

You're throwing a party and want to create a single guest list.

UNION Operator

Purpose: Combines results from two queries and removes duplicates.

How it works:

- Merges two lists into one
- If someone appears in both lists, they appear only once in the final result
- Removes duplicate entries

Example Result:

- Final guest list: Alice, Bob, Charlie, David, Eva
- Notice: Bob appears only once (not twice)

SQL Syntax:

```
SELECT name FROM school_friends
UNION
SELECT name FROM work_friends
```

Performance: Slower because it has to check for and remove duplicates.

UNION ALL Operator

Purpose: Combines results from two queries and keeps all duplicates.

How it works:

- Merges two lists into one
- Includes every entry, even if duplicates exist
- Does not remove duplicate entries

Example Result:

- Final guest list: Alice, Bob, Charlie, Bob, David, Eva
- Notice: Bob appears twice (once from each list)

SQL Syntax:

```
SELECT name FROM school_friends
UNION ALL
SELECT name FROM work_friends
```

Performance: Faster because it doesn't check for duplicates.

Key Differences

Aspect	UNION	UNION ALL
Duplicates	Removed	Kept
Result count	Fewer rows	More rows
Speed	Slower	Faster
Use case	Need unique results	Want all results
Example result	5 guests	6 guests (Bob counted twice)

When to Use

- **UNION**: When you need a unique, consolidated list
- **UNION ALL**: When you need all records, including duplicates, or when you know there are no duplicates

---

Question 12: Finding the Second Highest Salary

Approach Using Nested Subqueries

The SQL query to find the second highest salary operates through a structured flow involving nested subqueries.

Step-by-Step Process

Step 1: Find the Highest Salary

```
SELECT MAX(salary) FROM employee
```

Purpose: Get the maximum salary value from the employee table.

Example: If the highest salary is 100,000, this subquery returns 100,000.

Step 2: Find the Second Highest Salary

```
SELECT MAX(salary)
FROM employee
WHERE salary < (SELECT MAX(salary) FROM employee)
```

What it does:

1. Finds the maximum salary (100,000) from Step 1
2. Filters the employee table to exclude salaries equal to or greater than 100,000
3. Finds the maximum of the remaining salaries
4. **Result**: Returns the second highest salary (example: 90,000)

Step 3: Find Employee(s) with Second Highest Salary

```
SELECT name, salary
FROM employee
WHERE salary = (
    SELECT MAX(salary)
    FROM employee
    WHERE salary < (SELECT MAX(salary) FROM employee)
)
```

What it does:

1. Uses the second highest salary from Step 2
2. Matches it with employee records
3. Returns name and salary of all employees earning the second highest salary

Complete Query Example

```
SELECT name, salary
FROM employee
WHERE salary = (
    SELECT MAX(salary)
    FROM employee
    WHERE salary < (SELECT MAX(salary) FROM employee)
)
```

Key Points

- **Innermost query**: `SELECT MAX(salary) FROM employee` → Highest salary
- **Middle query**: Filters and finds the second highest salary

- **Outer query**: Returns employee names and salaries matching the second highest

Handling Multiple Employees

Important: If multiple employees earn the second highest salary, the query will return each one of them.

Result: Shows all employees who earn the second highest salary in the company.

Question 13: Views in SQL

What is a View?

Definition: A virtual table that doesn't store data on the disk but allows you to see data from one or more tables in a specific way.

Key Characteristic: A view is not a physical table – it's a saved query that presents data from existing tables.

Why Use Views?

1. Limit Sensitive Data: Show only necessary information to specific users **2. Restrict Permissions**: Control what data different users can access **3. Simplify Complex Queries**: Hide complex query logic behind a simple view name **4. Useful for large tables**: When different users need access to different parts of data

Example Scenario

Situation: You have a student database storing student ID, name, address, and other information. Someone wants to see only student names and addresses without seeing IDs or other sensitive data.

Creating a View

Query:

```
CREATE VIEW detail_view AS
SELECT name, address
FROM student
WHERE ID < 5
```

Explanation:

- `CREATE VIEW detail_view AS`: Creates a new view named "detail_view"
- `SELECT name, address FROM student`: Selects only name and address columns
- `WHERE ID < 5`: Adds a condition to show only records where ID is less than 5

Viewing the Results

Query to see the view:

```
SELECT * FROM detail_view
```

Result: Displays a virtual table with only the name and address columns for students with ID less than 5.

Benefits of Views

- **Security**: Hide sensitive columns like ID, salary, or personal details
- **Simplicity**: Users see only relevant data
- **Flexibility**: Different views can show different subsets of data to different users
- **Query reduction**: Complex queries are simplified into a view name

Important Notes

- Views do not store data themselves
- Views display data from underlying tables
- When underlying table data changes, the view automatically reflects those changes
- Views are particularly useful in multi-user database environments

---

Question 14: Converting Text to Date Format

Problem

You have text data like "201122" that represents a date but is in text format. You need to convert it to a proper date format.

Solution: String to Date Function

SQL Query:

```
SELECT STR_TO_DATE('201122', '%d%m%y')
```

Explanation:

- `STR_TO_DATE()`: Function to convert string to date format
- `'201122'`: The text input (represents October 22, 2020 in DDMMYY format)
- `'%d%m%y'`: The format pattern
  - `%d` = Day (22)
  - `%m` = Month (10)
  - `%y` = Year (20, which becomes 2020)

Format Pattern Symbols

Symbol	Meaning	Example
`%d`	Day (01-31)	22
`%m`	Month (01-12)	10
`%y`	Year (2-digit)	20

Symbol	Meaning	Example
%Y	Year (4-digit)	2020

How It Works

Input: '201122' (text string)

Processing: SQL reads the format pattern and interprets:

- First 2 digits (20) = Day
- Next 2 digits (11) = Month
- Last 2 digits (22) = Year

Output: October 22, 2022 (properly formatted date)

Important Notes

- The input must match the specified format exactly
- Different databases may use slightly different date functions (SQL Server uses CONVERT, Oracle uses TO_DATE)
- Always specify the correct format pattern for accurate conversion

---

Question 15: Triggers in SQL

What is a Trigger?

Definition: Triggers are automatic actions that run every time a certain event happens in your database.

Analogy: Like a reflex action – when a specific event triggers, an automatic response occurs without manual intervention.

Why Use Triggers?

Scenario: Managing a library where every time someone borrows or returns a book:

- Update the available book count
- Log the transaction for record-keeping

Doing this manually for every transaction would be:

- Very tedious
- Prone to errors
- Time-consuming

Solution: Use triggers to automate these tasks.

When Triggers Run

Triggers are usually set up to run during one of three events:

1. INSERT

- Runs when new data is inserted
- Example: When a new borrowed book record is added

2. UPDATE

- Runs when existing data in a table changes
- Example: When a book's availability status is updated

3. DELETE

- Runs when data is removed from a table
- Example: When a borrowed book record is deleted

Example: Updating Book Stock

Scenario: Every time someone borrows a book, automatically decrease the available stock.

Complete Trigger Code:

```
CREATE TRIGGER update_book_stock
AFTER INSERT ON borrowed_books
FOR EACH ROW
BEGIN
    UPDATE books
    SET available_count = available_count - 1
    WHERE book_id = NEW.book_id;
END
```

Breaking Down the Trigger

`CREATE TRIGGER update_book_stock`:

- Creates a new trigger named "update_book_stock"
- The name reflects what the trigger does (updates stock)

`AFTER INSERT ON borrowed_books`:

- `AFTER INSERT`: Trigger activates after a new record is inserted
- `ON borrowed_books`: Specifically targets the borrowed_books table
- Meaning: Every time a book is borrowed (new record added), this trigger runs

`FOR EACH ROW`:

- This is a looping statement
- Applies the trigger to every individual row being added
- If 5 books are borrowed at once, the trigger runs 5 times

`BEGIN ... END`:

- Marks the start and end of the trigger action

`UPDATE books SET available_count = available_count - 1`:

- The actual update operation
- Reduces the available_count by 1

`WHERE book_id = NEW.book_id`:

- Specifies which book to update
- `NEW.book_id` refers to the book ID of the newly borrowed book

How It Works Step-by-Step

1. **Event occurs**: Someone borrows a book (INSERT event into borrowed_books table)
2. **Trigger activates**: The trigger automatically runs
3. **Action executes**: The UPDATE statement decreases available_count by 1
4. **Result**: Stock count in the books table stays up-to-date automatically

Benefits of Triggers

- **Automation**: No manual intervention needed
- **Accuracy**: Eliminates human error
- **Consistency**: Rules are applied uniformly
- **Audit trails**: Can automatically log transactions
- **Data integrity**: Ensures related data stays synchronized

Important Considerations

- **Performance**: Too many triggers can slow down database performance
- **Complexity**: Triggers can make debugging harder
- **Maintenance**: Multiple triggers can interact in unexpected ways
- **Documentation**: Always document what triggers do for future maintenance

---

Summary

This comprehensive guide covers the 15 essential SQL interview questions:

1. **DBMS vs RDBMS**: Basic data storage vs organized relational data
2. **Primary Key vs Foreign Key**: Unique identification vs table relationships
3. **Constraints**: Rules for data accuracy (NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK, DEFAULT)
4. **DDL vs DML**: Structure definition vs data manipulation
5. **DELETE vs DROP vs TRUNCATE**: Row removal vs table structure vs complete deletion
6. **GROUP BY vs ORDER BY**: Aggregation vs sorting
7. **WHERE vs HAVING**: Row filtering vs group filtering
8. **Aggregate Functions**: COUNT, SUM, AVG, MIN, MAX
9. **Indexing and Clustered Index**: Data retrieval optimization
10. **Normalization**: Data organization to reduce redundancy (1NF, 2NF, 3NF, BCNF)

11. **UNION vs UNION ALL**: Combining queries with/without duplicates
12. **Second Highest Salary**: Nested subquery approach
13. **Views**: Virtual tables for data presentation
14. **Text to Date Conversion**: String formatting with STR_TO_DATE
15. **Triggers**: Automated actions on database events

Key Takeaways for Interview Success

- Understand fundamental database concepts and their real-world applications
- Know the differences between similar concepts (DELETE vs TRUNCATE, WHERE vs HAVING)
- Be able to explain concepts using analogies and examples
- Understand when to use each tool (GROUP BY for aggregation, ORDER BY for sorting)
- Practice writing actual SQL queries and understand execution flow
- Be ready to explain performance implications (indexing, triggers)
- Understand data integrity concepts (constraints, normalization, foreign keys)