SQL Overview & Categories

follow for more →

What is SQL?

SQL (Structured Query Language) is used to interact with relational databases by storing, retrieving, modifying, and managing structured data.

Why SQL is Important

- Works with almost all relational databases
- Used in real-world applications and analytics
- Essential for backend and data-related roles.
- Reliable and structured data handling.

Main Categories of SQL

- **DDL** (Data Definition Language)
- **DML** (Data Manipulation Language)
- **DQL** (Data Query Language)
- **DCL** (Data Control Language)
- **TCL** (Transaction Control Language)

Where You Use SQL Daily

- Fetch reports from databases
- Validate application data
- Support analytics and dashboards.

follow for more →

SQL Data Types

follow for more →

What are SQL Data Types?

SQL data types define the kind of values that can be stored in a table column and determine how the database stores, processes, and validates that data.

Why Data Types Matter

- ✔ Ensure correct type of data is stored
- ✔ Reduce storage and memory usage.
- ✔ Improve query performance.
- ✔ Prevent logical and calculation errors.

Main Categories of SQL Data Types

- **Numeric Data Types** → Used for storing numbers.
- **Character / String Data Types** → Used for storing text-based information.
- **Date & Time Data Types** → Used for storing date-related and time-based values,
- **Boolean Data Type** → Used for storing logical true or false values.

Commonly Used Data Types

- ✔ **INT** → whole numbers
- ✔ **DECIMAL** → precise numeric values
- ✔ **VARCHAR** → variable-length text
- ✔ **DATE** → calendar date
- ✔ **TIMESTAMP** → date with time.

follow for more →

SQL Constraints

follow for more →

What are SQL Constraints?

SQL constraints are rules applied to table columns that restrict the type of data allowed, ensuring accuracy, consistency, and reliability of stored information.

Why SQL Constraints are Important

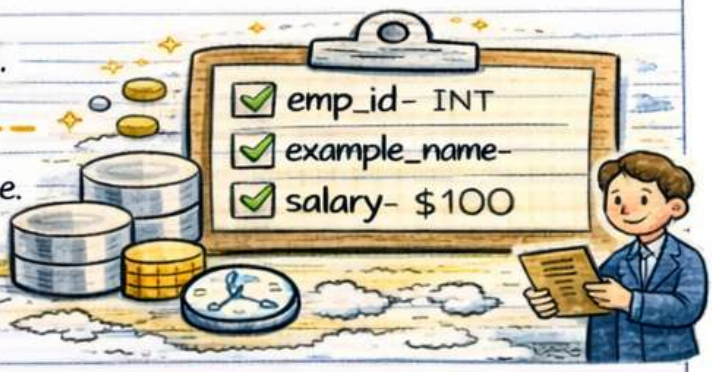
- Prevent invalid or inconsistent data entry.
- Enforce business rules at database level.
- Maintain data integrity automatically.
- Reduce dependency on application-side validation.

Types of SQL Constraints

- **NOT NULL** → A column always contains a value and does not accept NULL entries.
- **UNIQUE** → All values in a column are different and prevents duplicate records.
- **PRIMARY KEY** → Uniquely identifies each row in a table.
- **FOREIGN KEY** → Maintains relationships between tables by referencing a primary key from another table.
- **CHECK** → Validates data based on a specified condition.
- **DEFAULT** → Assigns a default value. when no value is provided.

Real-World Example

- emp_id uniquely identifies each employee.
- email, avoids duplicate entries.
- salary accepts only valid positive values.
- dept_id links employees to departments.



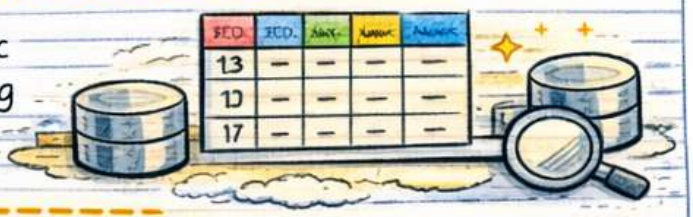
follow for more →

SELECT Query & Execution Basics

follow for more →

What is a SELECT Query?

The SELECT query is used to retrieve specific data from database tables without modifying the stored records.

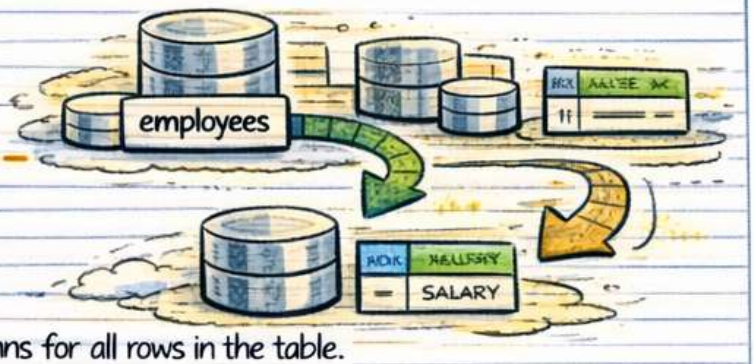


Purpose of SELECT

- Fetch required columns from tables
- View data without changing it.
- Form the base of reporting and analysis queries.

Basic SELECT Syntax

- Columns that are required
- Table from which data is fetched.



SELECT Query - Code Example

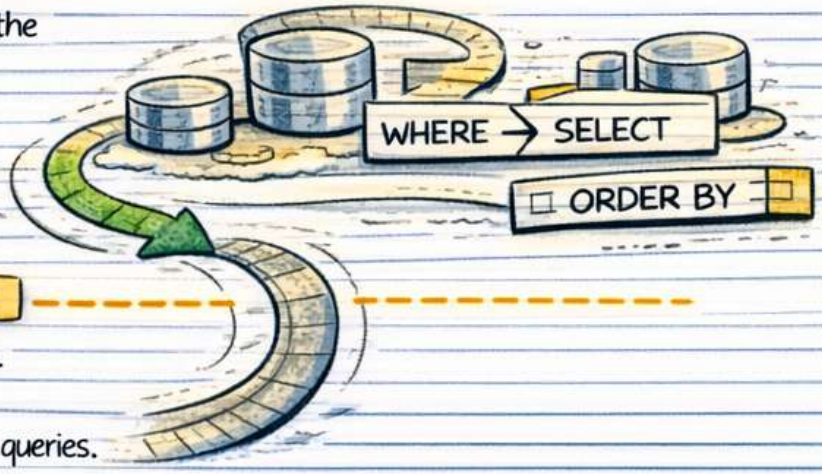
```
SELECT name, salary
FROM  employees;
```

This query returns only the selected columns for all rows in the table.

How SQL Executes a SELECT Query

SQL does not execute queries in the same order they are written,

- The database follows a logical sequence internally to decide
- how data is identified, filtered, and returned.



Why Execution Order Matters

- Prevents incorrect filtering logic.
- Explains alias-related errors
- Helps write correct GROUP BY queries.

follow for more →

WHERE, ORDER BY & LIMIT

follow for more →

Purpose of WHERE Clause

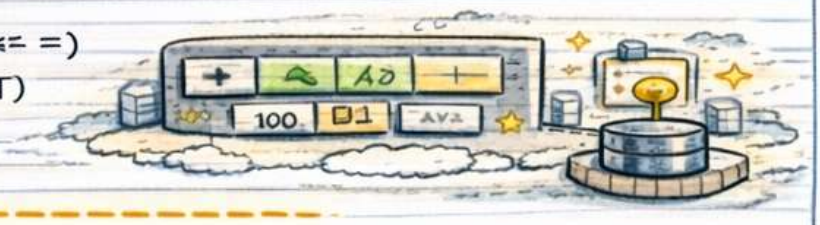
The WHERE clause is used to filter rows based on specific conditions before results are returned.

How WHERE Works

- Applies conditions to table rows.
- Returns only matching records.
- Reduces unnecessary data processing.

Common WHERE Conditions

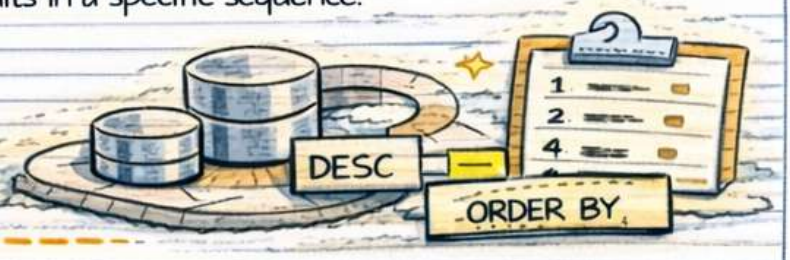
- Comparison operators (>, <, <, ≤= =)
- Logical operators (AND, OR, NOT)
- Range and pattern checks.



Purpose of ORDER BY

ORDER BY is used to sort query results in a specific sequence.

- Sorts data after selection.
- Can be ascending or descending.
- Helps present data meaningfully



Purpose of LIMIT

LIMIT restricts the number of rows returned by a query.

When LIMIT is Useful

- Fetching top results.
- Previewing large datasets.
- Improving query response time.

Query Example

```
SELECT, name, salary
FROM  employees
WHERE salary > 50000
ORDER BY salary DESC
LIMIT 5;
```

This query filters employees, sorts them by salary and returns only the top records.

SQL Aggregate Functions

follow for more →

What are Aggregate Functions?

Aggregate functions perform calculations on multiple rows and return a single summarized result.

Why Aggregate Functions are Used

- Analyze large datasets easily
- Generate summary reports.
- Support decision-making and analytics
- Reduce manual data processing

Common SQL Aggregate Functions

- **COUNT()** – Returns the total number of rows matching a condition.
- **SUM()** – Calculates the total value of a numeric column.
- **AVG()** – Returns the average value of a numeric column.
- **MIN()** – Finds the smallest value in a column.
- **MAX()** – Finds the largest value in a column.

Aggregate Function Example

```
SELECT COUNT(*), AVG(salary)
FROM employees;
```

This query returns the total number of employees and their average salary.

Real-World Use Case

In payroll or sales systems, aggregate functions help calculate:

- Total revenue
- Average salary
- Highest and lowest transactions.

Query Exomple

follow for more →

GROUP BY & HAVING

follow for more →

Purpose of GROUP BY

GROUP BY is used to divide rows into groups based on one or more columns so that aggregate functions can be applied to each group.

How GROUP BY Works

- Rows with the same values are grouped together
- Aggregates are calculated per group.
- Output contains one row per group

GROUP BY Example

```
SELECT department, AVG(salary)
FROM employees:
GROUP BY department;
```

This query calculates average salary for each department separately.

Purpose of HAVING

HAVING is used to filter grouped results after aggregation has been applied.

- WHERE filters rows before grouping
- HAVING filters groups after aggregation.

HAVING Example

```
SELECT department, COUNT(*)
FROM employees.
GROUP BY department;
HAVING COUNT(*) > 5;
```

This query returns only departments with more than five employees.

Real-World Usage

- Department wise reports
- Sales summaries.
- Performance analytics.

Important Rules

- GROUP BY is mandatory
- Sales summaries.

follow for more →

SQL Joins - Overview

follow for more →

What are SQL Joins?

SQL joins are used to combine data from two or more tables based on a related column between them.

Why Joins are Needed

- Data is stored across multiple tables
- Relationships avoid duplication of information
- Joins help retrieve meaningful combined results

How Tables are Related

Tables are usually connected using:

- Primary key in one table
- Foreign key in another table.

This relationship allows rows to be be matched correctly.

Main Types of SQL Joins

INNER JOIN	LEFT JOIN	RIGHT JOIN	FULL JOIN
Returns only rows that have matching values in both tables.	Returns all rows from the left table and matching rows from the right	Returns all rows from the right table and matching rows from the left	Returns all rows when there is a match in either table.

Simple Join Example

```
SELECT orders.order_id, customers.name
FROM orders
JOIN customers.
ON orders.customer_id = customers.customer_id;
```

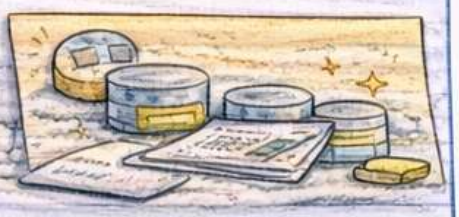
This query combines order data with customer details.

Important Behavior

- Join condition decides how rows are matched
- Incorrect join logic can multiply rows
- Join performance depends on indexing

Real-World Usage

- Order and customer reports
- Employee and department mapping
- Transaction history analysis.



follow for more →

SQL Transactions & ACID

follow for more →

What is a Transaction?

A transaction is a sequence of SQL operations executed together as one logical unit of work.

Why Transactions Matter

- Prevent partial updates.
- Maintain data consistency
- Handle failures safely
- Ensure reliable multi-step operations.

Transaction Result

- COMMIT → changes are saved
- ROLLBACK → changes are undone.

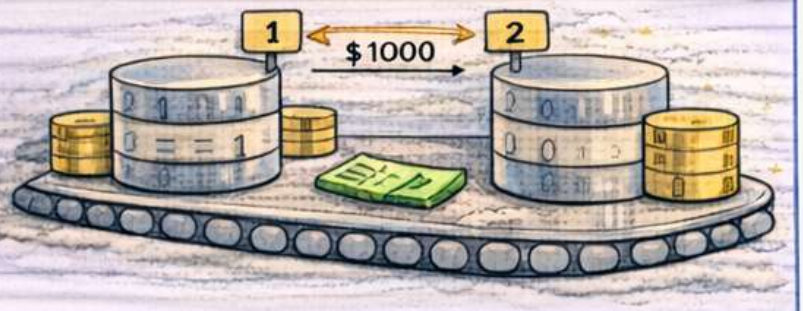
No partial state is allowed.

ACID Properties (Core Guarantee)

- **Atomicity** → All operations succeed together or none are applied
- **Consistency** → Database moves from one valid state to another.
- **Isolation** → Concurrent transactions do not affect each other.
- **Durability** → Committed data remains safe after failures.

Transaction Example

```
BEGIN;
UPDATE accounts
   SET balance = balance + 1000
   WHERE account_id = 1;
UPDATE accounts
   SET balance = balance – 1000
   WHERE account_id = 2;
COMMIT;
```



Used in banking, payments, and order processing systems.

Real-World Usage

- Used in banking, payments,
- and order processing systems.

Key insight

Transactions ensure correctness when multiple operations depend on each other.

follow for more →

Subqueries (Basic Idea)

follow for more →

What is a Subquery?

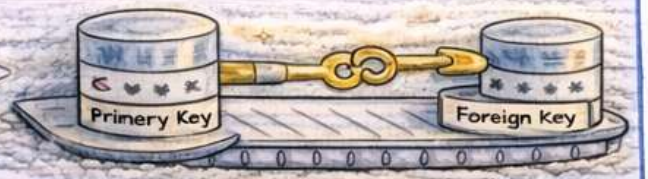
A subquery is a query written inside another SQL query, used to provide intermediate results to the main query

Why Subqueries are Used

- Break complex problems into smaller steps
- Use results of one query inside another
- Improve logical clarity of queries

Where Subqueries Can Appear

1) inside SELECT
2. inside WHERE
3 inside FROM.



Each placement serves a different purpose.

Simple Subquery Example

```
SELECT name
FROM employees
  WHERE salary > (
    SELECT AVG(salary)
  FROM employees);
```

This query returns employees earning more than the average salary.

How Subqueries Work

- Inner query runs first.
- Result is passed to outer query
- Outer query uses that result for filtering or selection.

Important Behavior

- Subqueries must return compatible data types
- Multiple subqueries nested need special operators
- Excessive nesting can impact performance.

Real-World Usage

- Order and customer reports
- Employee and department mapping
- Performance analytics.

Key insight

Subqueries allow SQL to return data using results of another query.

follow for more →

Indexes & Performance Basics

follow for more →

What is an Index in SQL?

An index is a database structure that improves query performance by allowing faster access to rows without scanning the entire table.

Why Indexes are Important

- Speed up data retrieval.
- Improve performance on large tables
- Optimize filtering and join operations
- Reduce full table scans.

This significantly reduces query execution time.

How Indexes Work (Conceptual View)

- Indexes store column values in an ordered structure
- Database uses the index to locate matching rows quickly
- Only required data blocks are accessed.

This significantly reduces query execution time.

When Indexes are Useful

- Columns are frequently used in WHERE conditions.
- Columns are involved in JOIN operations.
- Queries perform sorting or filtering on large datasets.

When Indexes Can Hurt Performance

- Tables have frequent INSERT, UPDATE, or DELETE operations.
- Too many indexes exist on a single table.
- Indexed columns have very few unique valuies.

Performance Insight

- Indexes should be created based on query patterns, not blindly on every column.

follow for more →

SQL Interview Quick Revision Sheet

follow @jobtechmingle

SQL Core Fundamentals

- SQL works with relational databases
- Data is stored in tables (rows & columns)
- Queries fetch data without modifying structure

SQL Query Thinking Order

- Identify required table(s)
- Apply conditions logically
- Select needed columns
- Format final output

This approach avoids most logical errors.

Must-Know SQL Topics

- SELECT & execution order
- WHERE, ORDER BY, LIMIT
- Aggregate functions
- GROUP BY & HAVING
-

- SELECT & execution order
- WHERE, ORDER BY LIMIT
- GROUP BY & HAVING

- Joins & subqueries
- Transactions & ACID

Common interview Mistakes

- ⚠ Confusing WHERE and HAVING
- ⚠ Ignoring execution order
- ⚠ Forgetting GROUP BY rules
- ⚠ Forgetting GROUP BY rules

- ▼ Explain logic before syntax
- ▼ Use simple real-world examples
- ▼ Mention performance impact
- ▼ Clarify assumptions clearly

How to Answer SQL Questions

- Explain logic before syntax
- Use real-world examples
- Mention performance impact
- Clarify assumptions clearly

Final Preparation

Strong SQL fundamentals matter more than memorizing syntax.

Tip:

Practice writing queries daily and focus on understanding "why" SQL behaves the way it does.

follow for more →