**SQL I – Relational Core**

**1. Relational Theory**

Relational theory is the foundation of SQL and relational databases. It models data as **relations**, which are essentially tables.

* **Relations**: Tables with rows and columns.
* **Tuples**: Rows in a table.
* **Attributes**: Columns in a table.
* **Schema**: The structure of a table, defining columns and data types.

Key principles:

* Each table has a **primary key** that uniquely identifies each row.
* Data operations follow **relational algebra** principles:
  + Selection (filter rows)
  + Projection (select columns)
  + Join (combine rows from different tables)
  + Union, Difference, Rename (for set logic)

**2. Primary Key (PK) and Foreign Key (FK)**

**Primary Key (PK)**:

* Uniquely identifies each row.
* Must be unique and not null.
* Ensures entity integrity.

**Foreign Key (FK)**:

* Refers to a primary key in another table.
* Enforces referential integrity between related tables.

Example:

sql

CREATE TABLE Employees (

emp\_id INT PRIMARY KEY,

name VARCHAR(100)

);

CREATE TABLE Projects (

proj\_id INT PRIMARY KEY,

emp\_id INT,

FOREIGN KEY (emp\_id) REFERENCES Employees(emp\_id)

);

**3. SELECT and WHERE Clauses**

Used to retrieve specific data from tables.

Example:

sql

SELECT name, salary

FROM Employees

WHERE department = 'Sales';

Explanation:

* SELECT specifies which columns to return.
* WHERE filters rows based on conditions.

Best practices:

* Use WHERE filters on indexed columns for better performance.
* Avoid applying functions to columns in WHERE clauses, which may prevent index usage.

**4. INNER JOIN and LEFT JOIN**

**INNER JOIN**:

* Returns only rows with matching keys in both tables.

**LEFT JOIN**:

* Returns all rows from the left table, plus matching rows from the right table if they exist; otherwise, NULLs are returned.

Examples:

sql

-- Inner Join

SELECT e.name, p.proj\_id

FROM Employees e

INNER JOIN Projects p ON e.emp\_id = p.emp\_id;

-- Left Join

SELECT e.name, p.proj\_id

FROM Employees e

LEFT JOIN Projects p ON e.emp\_id = p.emp\_id;

**5. GROUP BY and HAVING**

Used to group data and apply aggregate functions.

Example:

sql

SELECT department, COUNT(\*) AS total\_employees

FROM Employees

GROUP BY department

HAVING COUNT(\*) > 5;

Explanation:

* GROUP BY groups rows by one or more columns.
* HAVING filters groups based on conditions on aggregated values.

**6. NULL Semantics**

NULL represents missing or unknown values.

Key points:

* NULL is not equal to anything, including another NULL.
* Use IS NULL or IS NOT NULL to test for NULL.
* COUNT(column) ignores NULL, but COUNT(\*) includes them.
* Use COALESCE(column, default\_value) to handle NULL.

Example:

sql

SELECT name

FROM Employees

WHERE manager\_id IS NULL;

**7. EXPLAIN Basics**

EXPLAIN is used to analyze how the SQL engine will execute a query.

Example:

sql

EXPLAIN SELECT name FROM Employees WHERE emp\_id = 101;

Things to look for:

* Index usage (Index Scan, Index Seek) is preferred.
* Sequential scan (Seq Scan) may indicate missing indexes.
* Join strategy (Nested Loop, Hash Join, etc.)

Use EXPLAIN ANALYZE for actual timing and cost.

**8. Covering Index**

A covering index includes all the columns needed for a query, so the database does not need to read the full row from the table.

Benefits:

* Reduces disk I/O.
* Improves performance significantly.

Example:

sql

CREATE INDEX idx\_dept\_name ON Employees(department, name);

-- Now the query can be covered:

SELECT name FROM Employees WHERE department = 'Sales';

**9. Introduction to ACID**

ACID is a set of properties that guarantee reliable transactions in a database.

* **Atomicity**: All operations in a transaction succeed or none do.
* **Consistency**: Transactions leave the database in a valid state.
* **Isolation**: Transactions do not interfere with each other.
* **Durability**: Once committed, changes persist even after a crash.

These properties ensure data integrity in the presence of concurrency and failure.

**10. Indexed CRUD and JOIN Queries Under 50ms**

To achieve sub-50ms query times:

* Use indexes effectively on WHERE and JOIN columns.
* Avoid SELECT \*; select only needed columns.
* Use EXPLAIN to confirm optimal query plans.

Examples:

CRUD:

sql

-- INSERT

INSERT INTO Employees(emp\_id, name, department)

VALUES (101, 'Alice', 'HR');

-- READ

SELECT name FROM Employees WHERE emp\_id = 101;

-- UPDATE

UPDATE Employees SET department = 'Sales' WHERE emp\_id = 101;

-- DELETE

DELETE FROM Employees WHERE emp\_id = 101;

JOIN:

sql

-- Ensure index on Projects.emp\_id

CREATE INDEX idx\_proj\_emp ON Projects(emp\_id);

-- Optimized join query

SELECT e.name, p.proj\_id

FROM Employees e

JOIN Projects p ON e.emp\_id = p.emp\_id

WHERE e.department = 'Engineering';

Performance tips:

* Always use indexes on foreign keys.
* Analyze queries using EXPLAIN and optimize join order.
* Avoid full-table scans on large datasets.

**Summary**

* Relational databases rely on a strict mathematical foundation.
* Primary and foreign keys enforce data integrity.
* SELECT, JOIN, GROUP BY, and HAVING are core to data retrieval.
* NULLs require careful handling.
* EXPLAIN reveals query execution details.
* Covering indexes and proper indexing can drastically reduce query time.
* ACID properties ensure data reliability in transactional systems.
* Aim for efficient, indexed queries that return results in under 50ms.