Database & DBMS Recap

- **Database**: A collection of data.
- **DBMS**: Software to manage databases.
- **Relational vs. Non-Relational**: Databases can be relational (data stored in tables, e.g., MySQL, Oracle) or non-relational (data not necessarily in tables, e.g., MongoDB).

SQL Overview

- **Why SQL?**: SQL (Structured Query Language) is needed to interact with relational databases, performing operations like Create, Read, Update, Delete (CRUD).
- **What is MySQL?**: MySQL is a specific RDBMS product that uses SQL.
- **History**: Originated in the 1970s at IBM as SEQUEL, later renamed SQL.
- **Purpose**: Communicate with and manipulate data in databases. Tasks include:
    - Retrieving Data (Queries)
    - Manipulating Data (Add, Modify, Remove)
    - Defining Data (Database structure: tables, views, indexes)
    - Controlling Data (Access permissions)

MySQL Installation (Brief)

- The PDF mentions installing MySQL Server (the database engine) and MySQL Workbench (a visual UI tool).

Types of SQL Commands

- **DQL (Data Query Language)**: Retrieves data.
    - SELECT
- **DML (Data Manipulation Language)**: Manipulates data stored in the database.
    - INSERT, UPDATE, DELETE
- **DDL (Data Definition Language)**: Defines or modifies the database structure/schema.
    - CREATE, ALTER, DROP, TRUNCATE, RENAME
- **DCL (Data Control Language)**: Manages permissions and security.
    - GRANT, REVOKE
- **TCL (Transaction Control Language)**: Manages transactions.
    - COMMIT, ROLLBACK, SAVEPOINT

Basic Database & Table Operations

- **Create Database**:
- CREATE DATABASE databaseName;
- CREATE DATABASE IF NOT EXISTS databaseName; -- Avoids error if DB exists
- **Delete Database**:
- DROP DATABASE databaseName;
- DROP DATABASE IF EXISTS databaseName; -- Avoids error if DB doesn't exist
- **Use Database**: Selects the database to work with for subsequent commands.
- USE databaseName;
- **Show Databases**: Lists all databases on the server.
- SHOW DATABASES;
- **Create Table**: Defines a new table structure.
- CREATE TABLE TableName (
- Column1 DataType1 Constraint1,
- Column2 DataType2 Constraint2,
- ...
-);
- -- Example:
- CREATE TABLE employee (
- empId INT PRIMARY KEY,
- name VARCHAR(50),
- salary INT
-);

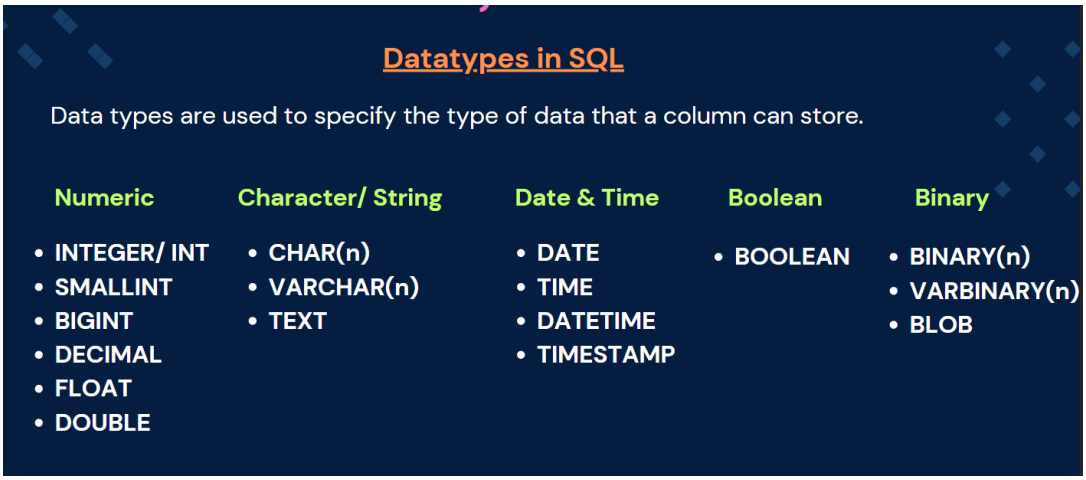
- **Insert Data**: Adds new rows into a table.
- INSERT INTO tableName (Column1, Column2, ...) VALUES (value1, value2, ...);
- -- Or insert values for all columns in order:
- INSERT INTO tableName VALUES (value1, value2, ...);

- **Show Tables**: Lists all tables in the currently selected database.
- SHOW TABLES;

- **Select Data**: Retrieves data from a table.
- SELECT column1, column2 FROM tableName; -- Specific columns
- SELECT * FROM tableName; -- All columns

SQL Datatypes

- Specifies the type of data a column can hold.
- **Numeric**: INT, BIGINT, FLOAT, DOUBLE, DECIMAL(p, s). Can use UNSIGNED for non-negative integers.
- **Character/String**: CHAR(n) (fixed-length), VARCHAR(n) (variable-length), TEXT (variable, large).
- **Date & Time**: DATE (YYYY-MM-DD), TIME (hh:mm:ss), DATETIME/TIMESTAMP (YYYY-MM-DD hh:mm:ss).
- **Boolean**: BOOLEAN (True/False).
- **Binary**: BINARY(n), VARBINARY(n), BLOB (Binary Large Object).



SQL Constraints

- Rules enforced on data columns.
- **NOT NULL**: Ensures a column cannot have a NULL value.
- **UNIQUE**: Ensures all values in a column (or set of columns) are unique. Allows NULLs.
- **PRIMARY KEY**: Uniquely identifies each row. Combination of NOT NULL and UNIQUE.
- **FOREIGN KEY**: Enforces a link between data in two tables, referencing the primary key of another table. Maintains referential integrity.
- **CHECK**: Ensures values in a column satisfy a specific condition (e.g., age >= 18).
- **DEFAULT**: Provides a default value for a column when none is specified during insertion.

Keys in SQL (Recap)

- **Primary Key**: Unique identifier for rows (Unique + Not Null).
- **Foreign Key**: Links to the Primary Key of another table (the "parent" or "referenced" table) from the "child" or "referencing" table.

Cascading Actions for Foreign Keys

- Rules defining automatic actions in the child table when a referenced row in the parent table is modified (updated/deleted). Defined using ON DELETE / ON UPDATE.
- **CASCADE**: Deletes/updates corresponding rows in the child table.
- **SET NULL**: Sets the foreign key columns in the child table to NULL.
- **RESTRICT / NO ACTION**: Prevents the delete/update operation on the parent table if related child rows exist (Default behavior in many systems if not specified).
    - ***PDF Visual (Page 52, 53):*** Shows syntax for *ON DELETE CASCADE* and *ON UPDATE CASCADE*.

DML & DDL Commands (Continued)

- **UPDATE**: Modifies existing records. Often used with WHERE.
- UPDATE tableName SET column1 = value1, column2 = value2 WHERE condition;
- -- Note: SET SQL_SAFE_UPDATES=0; might be needed in MySQL Workbench to allow updates without a key in the WHERE clause.

- **DELETE**: Removes existing records. Often used with WHERE.
- DELETE FROM tableName WHERE condition;

- **ALTER TABLE**: Modifies an existing table's structure.
    - ADD COLUMN: Adds a new column.
    - DROP COLUMN: Removes a column.
    - MODIFY COLUMN: Changes a column's datatype or constraints.
    - CHANGE COLUMN: Renames a column and optionally changes its datatype/constraints.
    - RENAME COLUMN ... TO ...: Renames a column (alternative syntax).
    - ***PDF Visual (Page 66-69):*** Provides syntax examples for these *ALTER TABLE* operations.
- **RENAME**: Changes the name of database objects.
    - RENAME TABLE oldName TO newName;
    - RENAME DATABASE oldName TO newName; (May not be supported/recommended in all systems).
- **TRUNCATE TABLE**: Removes *all* rows from a table quickly. Resets auto-increment counters. Cannot be easily rolled back. Preserves table structure.
- TRUNCATE TABLE tableName;

- **DROP vs DELETE vs TRUNCATE**:
    - DROP: Removes the entire table structure and data.
    - DELETE: Removes specific rows (or all if no WHERE), slower, can be rolled back.
    - TRUNCATE: Removes all rows, faster than DELETE, usually cannot be rolled back easily, keeps structure.
    - ***PDF Visual (Page 73):*** A table compares these three commands.

SQL Operators

- Used in expressions, often within WHERE clauses.
- **Arithmetic**: +, -, *, /, % (Modulus).
- **Comparison**: =, <> or !=, >, <, >=, <=.
- **Logical**: AND, OR, NOT.
- **Membership**: IN (checks if value is in a list), NOT IN.
- **Range**: BETWEEN ... AND ...
- **NULL Check**: IS NULL, IS NOT NULL.
- **Pattern Matching**: LIKE used with wildcards:
    - %: Matches any sequence of zero or more characters.
    - _: Matches any single character.
    - ***PDF Visual (Page 171):*** Table shows examples like *LIKE 'A%'*, *LIKE '%ra%'*, *LIKE 'A____'*, *LIKE '_a%'*.
- **Bitwise**: & (AND), | (OR).

SQL Clauses

- Keywords that modify or refine SQL statements.
- **DISTINCT**: Used with SELECT to return only unique rows/values.
- SELECT DISTINCT column1 FROM tableName;
- **WHERE**: Filters rows based on specified conditions. Applied *before* grouping.
- SELECT * FROM tableName WHERE condition;
- **ORDER BY**: Sorts the result set based on one or more columns. ASC (ascending, default) or DESC (descending). Applied *after* filtering and grouping.
- SELECT * FROM tableName ORDER BY column1 ASC, column2 DESC;
- **LIMIT**: Restricts the number of rows returned. Often used with ORDER BY.
    - LIMIT n: Returns the first n rows.
    - LIMIT m, n: Skips m rows, then returns the next n rows (offset m, count n).
- SELECT * FROM tableName LIMIT 10; -- First 10 rows
- SELECT * FROM tableName LIMIT 5, 10; -- Rows 6 through 15
- **GROUP BY**: Groups rows with the same values in specified columns into a summary row. Used with aggregate functions.
- SELECT column1, COUNT(*) FROM tableName GROUP BY column1;
- **HAVING**: Filters groups based on a specified condition *after* aggregation has occurred. Used with GROUP BY.
- SELECT column1, COUNT(*) FROM tableName GROUP BY column1 HAVING COUNT(*) > 1;

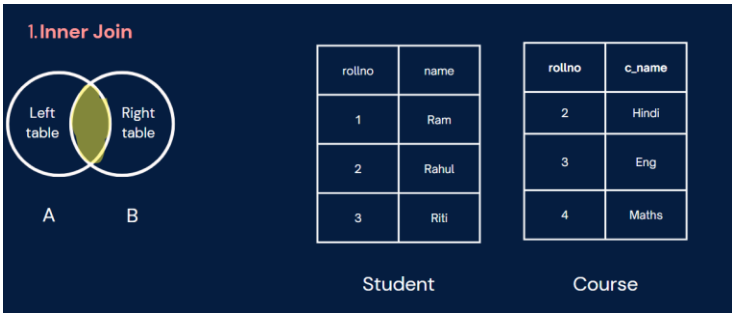
- **WHERE vs HAVING**: WHERE filters rows *before* grouping; HAVING filters groups *after* grouping.
- **General Order of Execution (Logical)**: FROM -> WHERE -> GROUP BY -> HAVING -> SELECT -> ORDER BY -> LIMIT.

Aggregate Functions

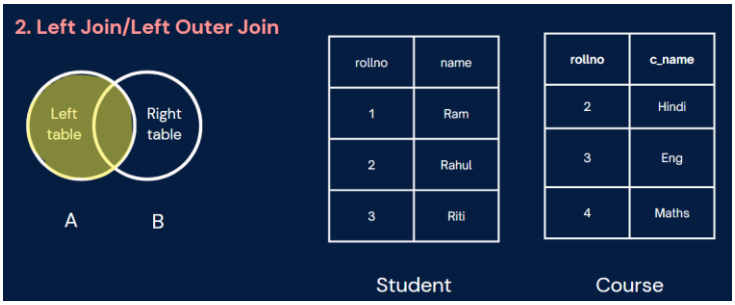
- Perform calculations on a set of rows and return a single summary value. Often used with GROUP BY.
- **COUNT()**: Counts rows or non-null values (COUNT(*) counts all rows).
- **SUM()**: Calculates the sum of values in a numeric column.
- **AVG()**: Computes the average of values in a numeric column.
- **MIN()**: Finds the minimum value in a column.
- **MAX()**: Finds the maximum value in a column.

Joins

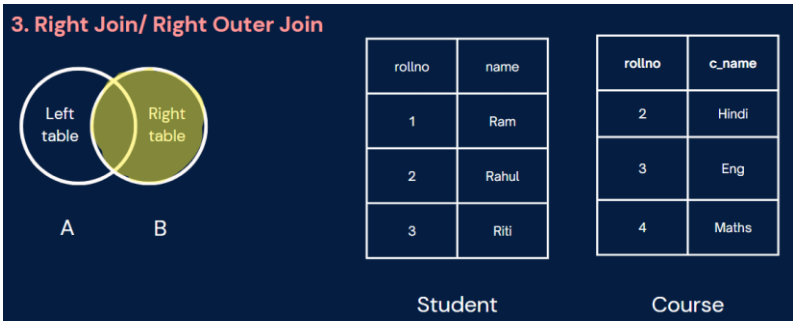
- Combine rows from two or more tables based on a related column.
- **INNER JOIN**: Returns only rows where the join condition is met in *both* tables (matching values).



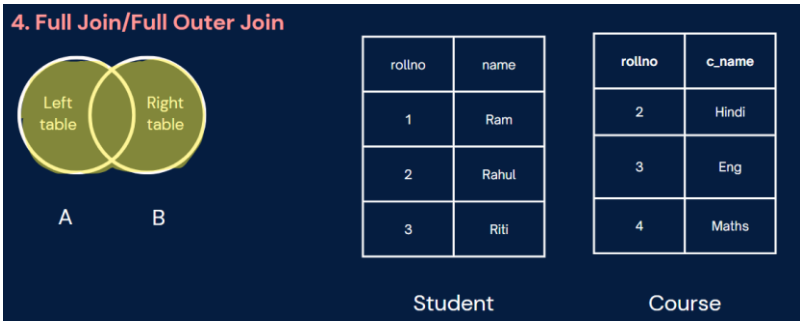
- **LEFT JOIN (or LEFT OUTER JOIN)**: Returns *all* rows from the left table and matching rows from the right table. If no match, NULLs appear for right table columns.



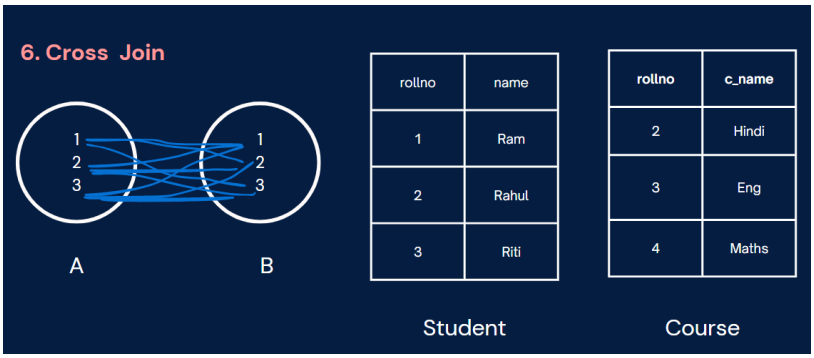
- **RIGHT JOIN (or RIGHT OUTER JOIN)**: Returns *all* rows from the right table and matching rows from the left table. If no match, NULLs appear for left table columns.



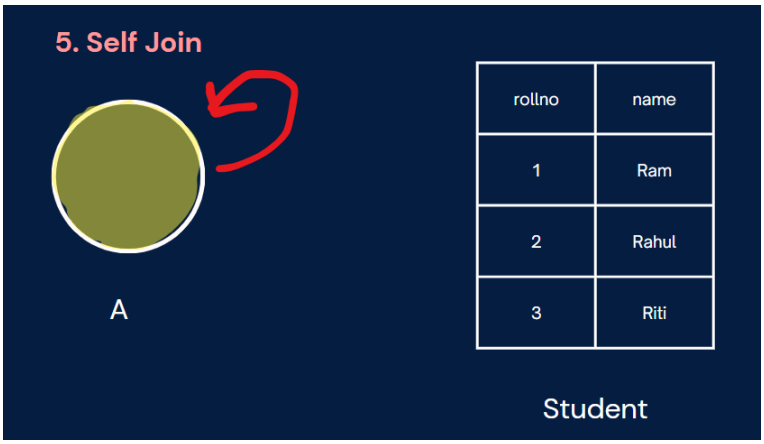
- **FULL JOIN (or FULL OUTER JOIN)**: Returns all rows when there's a match in either the left or right table. Includes all rows from both tables; NULLs fill in where matches are missing. (MySQL simulates using LEFT JOIN ... UNION ... RIGHT JOIN).



- **CROSS JOIN**: Returns the Cartesian product – every row from the first table combined with every row from the second table. (m rows * n rows = m*n result rows).



- **SELF JOIN**: A table is joined with itself, using aliases to distinguish the two instances. Used for comparing rows within the same table (e.g., finding employees and their managers).



- **Exclusive Joins**: Variations of outer joins using a WHERE ... IS NULL clause to find rows that exist in one table but *not* the other.
    - **Left Exclusive**: All rows from the left table with no match in the right (WHERE right_table.key IS NULL).
    - **Right Exclusive**: All rows from the right table with no match in the left (WHERE left_table.key IS NULL).
    - **Full Exclusive**: All non-matching rows from both tables (Union of Left Exclusive and Right Exclusive).
    - ***PDF Visual (Page 129-132):*** Venn diagrams and syntax examples provided.

UNION and UNION ALL

- Combine result sets of two or more SELECT statements. Columns must match in number, order, and compatible datatypes.
- **UNION**: Combines results and removes duplicate rows.
    - ***PDF Visual (Page 133-134):*** Venn diagram shows union. Example combines two lists of IDs, removing duplicates.
- **UNION ALL**: Combines results and includes *all* rows, including duplicates. Generally faster as it doesn't check for duplicates.
    - ***PDF Visual (Page 135-136):*** Example combines two lists of IDs, keeping duplicates.

Subqueries (Nested Queries / Inner Queries)

- A query embedded inside another SQL query (the outer query).
- **Usage**:
    - In WHERE clause: Filter data based on the result of the subquery (e.g., WHERE salary > (SELECT AVG(salary) FROM...)).
    - In SELECT clause: Include an aggregated value or calculated result alongside other columns (e.g., SELECT name, (SELECT AVG(age) FROM...) AS avg_age FROM...).
    - In FROM clause: Use the result set of the subquery as a temporary table (requires an alias) (e.g., FROM (SELECT MIN(age) AS min_age FROM...) AS sub).
    - ***PDF Visual (Page 141-148):*** Provides examples for using subqueries in WHERE, FROM, and SELECT clauses.

Finding Nth Highest Salary

- A common pattern using ORDER BY and LIMIT.
- **Technique**: Select distinct salaries, order descending, use LIMIT n-1, 1 to skip n-1 rows and take the next 1 row (which is the nth highest).
- SELECT DISTINCT Salary
- FROM tableName
- ORDER BY Salary DESC
- LIMIT n-1, 1; -- For the nth highest

Stored Procedures

- Pre-compiled SQL code saved under a name for reuse. Can accept parameters.
- **Create**:
- -- May need DELIMITER // before and DELIMITER ; after in some clients
- CREATE PROCEDURE procedureName(IN param1 INT, OUT param2 VARCHAR(50))
- BEGIN
- -- SQL statements here
- END;

- **Call**:
- CALL procedureName(value1, @outputVar);

Views

- Virtual tables based on the result set of a stored SQL query.
- **Purpose**: Simplify complex queries, provide data security (show only certain columns/rows), abstract underlying table structure.
- **Create**:
- CREATE VIEW viewName AS
- SELECT column1, column2 FROM baseTable WHERE condition;

- **Usage**: Query like a regular table (SELECT * FROM viewName;).
- **Drop**:
- DROP VIEW IF EXISTS viewName;

Conditional Logic

- **CASE Statement**: Provides IF-THEN-ELSE logic within SQL statements.
- SELECT
- CASE
- WHEN condition1 THEN result1
- WHEN condition2 THEN result2
- ELSE defaultResult
- END AS newColumnName
- FROM tableName;
-
- UPDATE tableName SET column = CASE WHEN condition THEN val1 ELSE val2 END WHERE ...;

    - ***PDF Visual (Page 159-160):*** Examples categorize students based on percentage and update grades.
- **IF Function (MySQL specific)**: Simpler conditional logic returning one of two values.
- IF(condition, value_if_true, value_if_false)
-
- -- Example in SELECT
- SELECT IF(percentage > 50, 'Pass', 'Fail') AS status FROM ...;
- -- Example in UPDATE
- UPDATE employee SET gender = IF(gender = 'm', 'f', 'm');

- ***PDF Visual (Page 162-163):*** Examples categorize students and swap gender values.

SQL Interview Questions & Approach

- The PDF includes common interview questions (CREATE/DROP DB/Table, INSERT/UPDATE/DELETE, finding Nth highest salary, duplicates, NULLs, pattern matching, aggregate functions, definitions of terms like NULL/Keys/Joins/Views, WHERE vs HAVING).
- **General Approach**: Understand the requirement -> Analyze schema -> Choose appropriate clauses/functions -> Write query -> Test/Optimize.
- **LeetCode Examples**: Mentions problems like Swap Salary, Duplicate Emails, Employees Earning More Than Managers, Not Boring Movies, Classes More Than 5 Students.