SQL JOINS (Basic Idea)

- Joins are used to **combine data from two or more tables**
- Tables are joined using a **common column (usually primary key & foreign key)**

INNER JOIN

- Returns **only matching records** from both tables
- Intersection of Table A and Table B
- If there is no match, the row is not shown

Example idea:

- Joining film and language
- Only films that have a matching language_id are returned

LEFT JOIN

- Returns **all records from left table**
- Matching records from right table
- If no match, right side columns show **NULL**

Used when:

- You want all customers even if they have no rentals

RIGHT JOIN

- Opposite of LEFT JOIN
- Returns **all records from right table**
- Matching records from left table
- If no match, left side shows NULL

FULL OUTER JOIN

- Returns **everything from both tables**
- Matching + non-matching rows
- If no match, NULL values appear

Note:

- MySQL does not support FULL OUTER JOIN directly
- Achieved using:
    - LEFT JOIN
    - UNION
    - RIGHT JOIN

FULL JOIN using UNION

- LEFT JOIN gives all left records
- RIGHT JOIN gives all right records
- UNION combines both

- This gives full outer join result

SELF JOIN

- Joining a table with **itself**
- Used to compare rows within the same table

Example:

- Finding staff working in the same store
- Table is aliased as s1 and s2
- Condition:
    - same store_id
    - different staff_id (<>)

CROSS JOIN (Cartesian Join)

- Every row of table A joins with every row of table B
- If A has 3 rows and B has 3 rows → result = 9 rows
- Rarely used
- Mostly for testing or combinations

EXISTS (WHERE EXISTS)

- Used to check if a subquery returns at least one row

- Returns TRUE or FALSE
- Stops checking once a match is found (efficient)

Example idea:

- Select customers who have rentals
- EXISTS checks rental table for matching customer_id

JOIN vs EXISTS vs INNER JOIN

- INNER JOIN → returns actual matching rows
- EXISTS → just checks if matching row exists
- EXISTS is faster for checking conditions

Special JOIN Condition (ON 1 = 0)

- Always false condition
- Used with UNION to force no match
- Helps combine data from unrelated tables
- Produces NULLs intentionally

Normalization in SQL

- Process of organizing data
- Reduces redundancy

- Improves data integrity

First Normal Form (1NF)

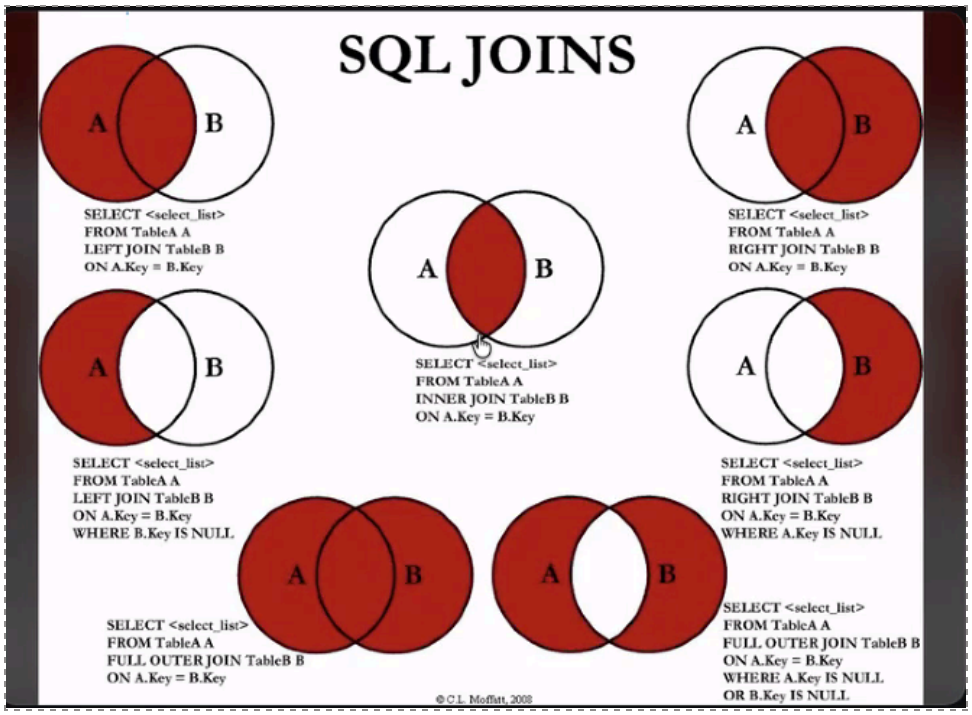
- Each column should contain **atomic values**
- No multiple values in one column
- Each cell holds only one value
- Columns should have unique names

Second Normal Form (2NF)

- Table must already be in 1NF
- No partial dependency
- Non-key columns depend **only on the primary key**
- Important for composite keys

Third Normal Form (3NF)

- Table must be in 2NF
- No transitive dependency
- Non-key columns should not depend on other non-key columns
- Depends only on primary key



Denormalized structure is used in data warehousing