Chapiter3:

SQL: Database Query Language (DQL)

Teacher F.Benabderrahmane

Definition

- **SQL (Structured Query Language)** is the standard language used to:
  - ✓ Query relational databases
  - ✓ Retrieve and manipulate data
  - ✓ Express relational operations in a declarative way

Position in SQL Categories

- SQL includes several sublanguages:
- ✓ **DQL (Data Query Language)** → SELECT *(focus of this chapter)*
- ✓ DDL (Data Definition Language) → CREATE, ALTER, DROP
- ✓ DML (Data Manipulation Language) → INSERT, UPDATE, DELETE
- ✓ DCL (Data Control Language) → GRANT, REVOKE

Example Database Schema: Human Resources

We will use the following database throughout the chapter:

- ✓ **Employee**(emp_id, last_name, first_name, hire_date, job_id#, salary, bonus, manager_id, dept_id#) //all employees of the organization
- ✓ **Department**(dept_id, dept_name, manager_id#) //all departments of the organization
- ✓ **Job**(job_id, job_title, min_salary, max_salary) // all job categories / positions
- ✓ **Job_History**(**emp_id, job_id**, dept_id, start_date, end_date) // past assignments of employees

Employee Table

emp_id	last_name	first_name	hire_date	job_id	salary	bonus	manager_id	dept_id
101	Benali	Amine	2020-02-15	dev	85000	5000	201	10
102	Bensalem	Sara	2021-09-01	hr_assist	62000	NULL	202	20
103	Aït Kaci	Yacine	2019-03-10	account	78000	3000	203	30
104	Bouzid	Imane	2022-01-17	dev	83000	NULL	201	10
105	Messaoudi	Khaled	2018-11-05	sales	70000	8000	204	40
106	Cheriet	Lina	2023-05-20	secretary	50000	NULL	202	20
107	Merabet	Nadir	2017-07-13	manager	120000	10000	NULL	10
108	Gherbi	Houda	2020-10-04	sales	68000	6000	204	null
109	Khelifi	Walid	2021-06-28	it_support	64000	NULL	201	10
110	Rahmani	Asma	2016-04-09	manager	115000	9000	NULL	20

Department Table

dept_id	dept_name	manager_id
10	Informatics	107
20	Human Resources	110
30	Finance	111
40	Sales	112
50	Administration	113

Jobs Table

job_id	job_title	min_salary	max_salary
DEV	Developer	70000	100000
HR_ASSIST	HR Assistant	50000	70000
ACCOUNT	Accountant	65000	90000
SALES	Sales Representative	55000	85000
SECRETARY	Secretary	40000	60000
MNG	MANAGER	100000	140000
IT_SUPPORT	IT Support Technician	55000	75000

Job-History Table

emp_id	job_id	dept_id	start_date	end_date
101	IT_SUPPORT	10	2019-01-01	2020-02-14
103	SECRETARY	30	2017-02-01	2019-03-09
104	IT_SUPPORT	10	2021-01-10	2022-01-16
105	DEV	10	2017-09-01	2018-11-04
108	HR_ASSIST	20	2019-06-15	2020-10-03

Querying Relational Databases : One Core Instruction: SELECT

- In relational databases, **data retrieval is done using a single declarative statement:**
  - ✓SELECT

- SQL is a **declarative language**:
  - ✓You specify **what you want**, not **how to compute it**
  - ✓The DBMS decides the execution strategy

General Role of SELECT

The SELECT statement allows you to:

- ✓ Retrieve data from one or more tables
- ✓ Filter rows (**conditions → WHERE**)
- ✓ Select specific columns (**projection**)
- ✓ Combine tables (**JOINs**)
- ✓ Group data (**GROUP BY**)
- ✓ Filter groups (**HAVING**)
- ✓ Sort results (**ORDER BY**)

Mapping SQL with Relational Algebra

Relational Algebra	**SQL**
Projection (π)	SELECT columns
Selection (σ)	WHERE
Join (*)	FROM + JOIN
Rename (α)	AS

Important Notes

- ✓ SELECT * → returns **all columns**
- ✓ Column order in SELECT = order in result
- ✓ SQL is **case-insensitive** (but convention: **uppercase keywords**)
- ✓ Strings use **single quotes**

How to Express Projection?

Syntax (PostgreSQL)

SELECT [ALL | DISTINCT] value_expression

FROM table_name;

- ✓ SELECT → specifies the **columns to keep** (projection)
- ✓ ALL → keeps duplicates *(default)*
- ✓ DISTINCT → removes duplicates
- ✓ A value expression can be: A **column,** A **constant** , An **arithmetic expression**
- ✓ Arithmetic operators: + - * /
  - ➢ Can combine: columns, constants, expressions

Projection: examples

Q1> SELECT last_name, first_name
FROM employee;

Result

last_name	first_name
Benali	Amine
Bensalem	Sara
Aït Kaci	Yacine
Bouzid	Imane
Messaoudi	Khaled
Cheriet	Lina
Merabet	Nadir
Gherbi	Houda
Khelifi	Walid
Rahmani	Asma

Projection: examples

Q2) SELECT DISTINCT job_id
FROM employee;

Result 

job_id
DEV
HR_ASSIST
ACCOUNT
SALES
SECRETARY
MANAGER
IT_SUPPORT

Projection: examples

Q2) Display employee names and their annual salary (salary × 12) :

```
SQL> SELECT last_name, salary * 12 AS annual_salary
     FROM employee;
```

Result 

last_name	annual_salary
Benali	1020000
Bensalem	744000
Aït Kaci	936000
Bouzid	996000
Messaoudi	840000
Cheriet	600000
Merabet	1440000
Gherbi	816000
Khelifi	768000
Rahmani	1380000

Projection **Insight**

- Projection **reduces columns**, not rows (except with DISTINCT)
- SQL result = **relation instance (table)**
- Each query produces a **new temporary relation**

How to Express Selection?

Syntax (PostgreSQL) :

```
SELECT *
FROM table_name
WHERE condition;
```

Definition : A **selection** filters rows based on a condition. A condition is called a **predicate** in SQL

Predicate : A predicate compares:

- a **column (attribute)** with a **constant** or another column

Tyes of Predicates in SQL

Comparison Predicate : = != < > <= >=

Range Predicate : BETWEEN

Text Predicate: LIKE

NULL Predicate: IS NULL

Membership Predicate: IN

Comparison Predicate : = != < > <= >=

- Example 1 : Which employees have a salary greater than or equal to 80,000?

```
SQL > SELECT last_name, salary
      FROM employee
      WHERE salary >= 80000;
```

Result :

last_name	salary
Benali	85000
Bouzid	83000
Merabet	120000
Rahmani	115000

Range Predicate: BETWEEN

- Select values within an interval (inclusive)
- Which employees have a salary between 70,000 and 90,000?

```
SQL> SELECT last_name, salary
       FROM employee
       WHERE salary BETWEEN 70000 AND 90000;
```

Result :

last_name	salary
Benali	85000
Aït Kaci	78000
Bouzid	83000
Messaoudi	70000

Text Predicate: LIKE

- Pattern matching
  - % → any sequence
  - _ → one character
- Which employees have a last name starting with the letter 'B'?

```
SQL> SELECT last_name
        FROM employee
        WHERE last_name LIKE 'B%';
```

Result :

last_name
Benali
Bensalem
Bouzid

NULL Predicate: IS NULL

- Test missing values
- Which employees have no bonus (i.e., bonus is NULL)?

```
SQL> SELECT last_name
        FROM employee
        WHERE bonus IS NULL;
```

Result :

last_name
Bensalem
Bouzid
Cheriet
Khelifi

Membership Predicate: IN

- Check multiple possible values

- Which employees work in the DEV or SALES job positions?

```
SQL> SELECT last_name, job_id
       FROM employee
       WHERE job_id IN ('DEV', 'SALES');
```

last_name	job_id
Benali	DEV
Bouzid	DEV
Messaoudi	SALES
Gherbi	SALES

Important Notes

- BETWEEN is **inclusive**
- LIKE is **case-sensitive in PostgreSQL**

  (unless using ILIKE)
- IS NULL is mandatory (NOT = NULL)
- IN is equivalent to multiple OR

Applicative exercise

- List employees working in DEV or HR_ASSIST with known bonus

```
SQL> SELECT last_name
        FROM employee
        WHERE job_id IN ('DEV', 'HR_ASSIST')
        AND bonus IS NOT NULL;
```

Cartesian Product in SQL

Definition : The Cartesian product corresponds to A **combination of all rows** from two (or more) tables

How to Express Cartesian Product?

```
SQL> SELECT *
     FROM table1, table2;
```

- ✓No condition → **all possible combinations**
- ✓Number of rows: $|Result| = |R| \times |S|$

Important Warning :

- ➢This is **almost never used alone**
- ➢It produces **huge meaningless results**
- ➢It must be followed by a condition:

Cartesian Product (SQL2 / PostgreSQL)

```
SQL> SELECT columns
        FROM table1
        CROSS JOIN table2;
```

- CROSS JOIN = rarely used alone
- Mostly used as a base for JOIN logic

Aliases (Table Synonyms) in SQL

Why Do We Use Aliases?

Aliases allow you to:

- ✓simplify queries
- ✓avoid ambiguity
- ✓improve readability

Definition : An alias is a temporary name given to a table in the FROM clause

Ways to Reference Attributes using aliases

- **Column name only** *(if no ambiguity)*

  SQL> SELECT last_name

- **Table.column** *(if ambiguity)*

  SQL> SELECT employee.last_name

- **Alias.column** *(recommended)*

  SQL> SELECT e.last_name

  Benefit : Shorter + clearer + easier to read

How to Express a JOIN in SQL?

- **Principle**

A join combines rows from multiple tables **based on a condition**

It is equivalent to: cartesian product followed by a selection

JOIN = Cartesian Product + Selection

How to Express a JOIN in SQL?

Method 1 — Old Style (Product + WHERE)

Question: Retrieve all employees with their corresponding departments.

```
SQL> SELECT *
     FROM employee, department
     WHERE employee.dept_id = department.dept_id;
```

Interpretation :

- ✓ Step 1: Cartesian product
- ✓ Step 2: filter rows using condition

Same query in relational algebra:

$$Res = \sigma(employee.dept\_id = department.dept\_id)(employee \times department)$$

How to Express a JOIN in SQL?

Method 2 — Modern SQL (Recommended)

Question: Retrieve all employees with their corresponding departments.

```
SQL> SELECT *
     FROM employee e
     JOIN department d
     ON e.dept_id = d.dept_id;
```

- **Advantages**
  - ✓ More readable
  - ✓ Safer (avoids forgetting conditions)
  - ✓ Standard in PostgreSQL and industry

How to Express a JOIN in SQL?

Method 3 — Using Subqueries

Question: Retrieve employees who belong to an existing department.

```
SQL> SELECT *
     FROM employee
     WHERE dept_id IN (
                        SELECT dept_id
                        FROM department
                      );
```

Interpretation

- ✓ Uses **query nesting**
- ✓ Less direct for joins, but useful in some cases

NATURAL JOIN

Definition : NATURAL JOIN automatically joins tables using all columns with the same name

Syntax (PostgreSQL) :

```
SQL> SELECT columns
     FROM table1
     NATURAL JOIN table2;
```

Important Remarks :

- Join condition is implicit→ based on identical column names
- **No need** to specify join condition
- Column **duplication is removed**
- **Restriction :**You cannot prefix the join column with table alias
- Avoid NATURAL JOIN in real systems, Prefer explicit JOIN … ON

Example (Projection + Join)

Question: Retrieve the names of employees along with their department names.

```
SQL> SELECT e.last_name, d.dept_name
        FROM employee e
        JOIN department d
        ON e.dept_id = d.dept_id;
```

last_name	dept_name
Benali	Informatics
Ait Ali	Finance
Mansouri	HR

Applicative exercise

Question: Retrieve the names of employees working in the “Informatics” department.

```
SQL> SELECT e.last_name
       FROM employee e
       JOIN department d ON e.dept_id = d.dept_id
        WHERE d.dept_name = 'Informatics';
```

Expressing the Same Query in Different Ways

Question :

Retrieve the **names and departments of employees working as "Manager"**

1st Formulation — JOIN with WHERE (Product + Selection)

Question : Retrieve the **names and departments of employees working as "Manager"**

```
SQL> SELECT e.last_name, d.dept_name
        FROM employee e, department d, job j
        WHERE e.dept_id = d.dept_id
        AND e.job_id = j.job_id
        AND j.job_title = 'Manager';
```

2nd Formulation — Using Subqueries

```
SQL> SELECT last_name, dept_NAME
        FROM employee e , DEPARTMENT d
        WHERE  e.dep-id = d.dep-id
           AND    job_id  = (
                              SELECT job_id
                              FROM job
                              WHERE job_title = 'Manager'
                             );
```

3rd Formulation — Modern JOIN (Recommended)

```
SQL> SELECT e.last_name, d.dept_name
FROM employee e
JOIN department d ON e.dept_id = d.dept_id
JOIN job j ON e.job_id = j.job_id
WHERE j.job_title = 'Manager';
```

INNER JOIN vs LEFT JOIN

JOIN Type	Meaning
INNER JOIN	Only matching rows
LEFT JOIN	All rows from left table + matches (or NULL)

Example :

Retrieve employee names with their department names.

Key difference :

INNER JOIN → filters rows

LEFT JOIN → preserves left table

Employee Table

emp_id	last_name	first_name	hire_date	job_id	salary	bonus	manager_id	dept_id
101	Benali	Amine	2020-02-15	dev	85000	5000	107	10
102	Bensalem	Sara	2021-09-01	hr_assist	62000	NULL	110	20
103	Aït Kaci	Yacine	2019-03-10	account	78000	3000	203	30
104	Bouzid	Imane	2022-01-17	dev	83000	NULL	201	10
105	Messaoudi	Khaled	2018-11-05	sales	70000	8000	204	40
106	Cheriet	Lina	2023-05-20	secretary	50000	NULL	202	20
107	Merabet	Nadir	2017-07-13	manager	120000	10000	NULL	10
108	Gherbi	Houda	2020-10-04	sales	68000	6000	204	null
109	Khelifi	Walid	2021-06-28	it_support	64000	NULL	201	10
110	Rahmani	Asma	2016-04-09	manager	115000	9000	NULL	20

INNER JOIN

```
SQL> SELECT e.last_name, d.dept_name
     FROM employee e
     INNER JOIN department d
     ON e.dept_id = d.dept_id;
```

last_name	dept_name
Benali	Informatics
Ait Ali	Finance

NB : Employees **without department are excluded**

LEFT JOIN

```
SQL> SELECT e.last_name, d.dept_name
     FROM employee e
     LEFT JOIN department d
     ON e.dept_id = d.dept_id;
```

last_name	dept_name
Benali	Informatics
Ait Ali	Finance
Mansouri	NULL

NB : Employees **without department are included**

Applicative exercise

Question :

Retrieve all employees and their jobs, including those without a job.

Which JOIN should you use?

```
SQL> SELECT e.last_name, j.job_title
        FROM employee e
        LEFT JOIN job j
        ON e.job_id = j.job_id;
```

Important Use Case — Ambiguity and Aliases

- When two tables have the same column name: Aliases are recommended

Question : Retrieve employee IDs and department IDs.

```
SQL> SELECT e.emp_id, d.dept_id
        FROM employee e
        JOIN department d
        ON e.dept_id = d.dept_id;
```

Required because both tables contain dept_id

Ambiguity and Aliases: Self Join (Alias is Mandatory)

Question : Retrieve employees and their managers

```
SQL> SELECT e.last_name AS employee, m.last_name AS manager
       FROM employee e
       INNER JOIN employee m
       ON e.manager_id = m.emp_id;
```

Self Join (Joining a Table with Itself)

Principle

A self join is used to **Combine** information from different rows of the **SAME** table.

Important Rule

- You MUST use aliases (table synonyms)
- Otherwise, SQL cannot distinguish the two roles of the same table

Self Join (Joining a Table with Itself)

Example — Manager Relationship

Question : Retrieve employee names along with their manager names.

emp_id	last_name	manager_id
1	Benali	NULL
2	Ait Ali	1
3	Mansouri	1
4	Haddad	2
5	Bouchareb	2

Step 1 — employee Table (Extension)

Query :

```
SQL> SELECT e.last_name AS employee,
            m.last_name AS manager
     FROM employee e
     LEFT JOIN employee m
     ON e.manager_id = m.emp_id;
```

Example — Manager Relationship : Query execution

e

emp_id	last_name	manager_id
1	Benali	NULL
2	Ait Ali	1
3	Mansouri	1
4	Haddad	2
5	Bouchareb	2

m

emp_id	last_name	manager_id
1	Benali	NULL
2	Ait Ali	1
3	Mansouri	1
4	Haddad	2
5	Bouchareb	2

Result

employee	manager
Benali	NULL
Ait Ali	Benali
Mansouri	Benali
Haddad	Ait Ali
Bouchareb	Ait Ali

Second Example — Same Department

Question : Retrieve pairs of employees working in the same department.

```
SQL> SELECT   e1.last_name AS emp1, e2.last_name AS emp2,
                    e1.dept_id
       FROM employee e1
      JOIN employee e2
       ON e1.dept_id = e2.dept_id
       AND e1.emp_id < e2.emp_id;
```

emp_id	last_name	dept_id
1	Benali	10
2	Ait Ali	20
3	Mansouri	10
4	Haddad	20
5	Bouchareb	20

Second Example — Same Department

Result :

emp1	emp2	dept_id
Benali	Mansouri	10
Ait Ali	Haddad	20
Ait Ali	Bouchareb	20
Haddad	Bouchareb	20

Why e1.emp_id < e2.emp_id ?

- **Avoid duplicates: (A, B) and (B, A)**
- **Avoid pairing a row with itself**

Nested Queries (Subqueries) in SQL

Concept :

A powerful feature of SQL is the ability to use a **SELECT inside another query (subquery)**.

- A subquery can be used inside WHERE to filter results dynamically.

Query :

List employees who work in the **same department as employee with ID = 123**

Example (Two Equivalent Solutions)

Solution 1: Using a JOIN (Self Join)

Query : List employees who work in the **same department as employee with ID = 123**

```
SQL> SELECT e1.emp_name
     FROM employee e1
     JOIN employee e2
     ON e1.dept_id = e2.dept_id
     WHERE e2.emp_id = 123
     AND e1.emp_id <> e2.emp_id;
```

Example (Two Equivalent Solutions)
Solution 2: Using a Subquery

Query : List employees who work in the **same department as employee with ID = 123**

```
SQL> SELECT e1.emp_name
     FROM employee
     WHERE  dept_id = (
                        SELECT dept_id
                        FROM employee
                        WHERE emp_id = 123
                       )
     AND emp_id <> 123;
```

Single-Row Subquery

Definition :

A subquery returning ONE value behaves like a constant.

Syntax:

WHERE expression operator (SELECT ...)

Operators : !=, <, >, <= , >=

- Interpretation : The subquery is evaluated first → returns a value → used in comparison

Multi-Row Subquery

- **Available Operators**
  - **IN**
  - **NOT IN**
  - **ANY**
  - **ALL**

  Followed by one of the comparison operators : (=, !=, <, >, <=, >=)

- **General Forms :**

  WHERE expression IN (SELECT ...)

  WHERE expression NOT IN (SELECT ...)

  WHERE expression op ANY (SELECT ...)

  WHERE expression op ALL (SELECT ...)

ANY vs ALL

WHERE expression op ANY (SELECT …)

ANY : Condition is TRUE if it holds for at least one value

WHERE exp op ALL (SELECT …)

ALL : Condition is TRUE only if it holds for ALL values

Example with ALL

Query : List employees who earn **more than all employees in department 30**

```
SQL> SELECT emp_name, salary
        FROM employee
        WHERE salary > ALL (
                                SELECT salary
                                FROM employee
                                WHERE dept_id = 30
                               );
```

- **Important Equivalences**

IN ≡ = ANY

NOT IN ≡ != ALL

Complex Example (Multi-level Subquery)

Query : List employees in department 10 who have the **same job as at least one employee in the "SALES" department**

```
SQL> SELECT emp_name, job_id
        FROM employee
        WHERE dept_id = 10
        AND job_id = ANY (
                                SELECT job_id
                                FROM employee
                                WHERE dept_id = (
                                                        SELECT dept_id
                                                        FROM department
                                                         WHERE dept_name = 'SALES'
                                                    )
                        );
```

Equivalent version (simpler, recommended)

Query : List employees in department 10 who have the **same job as at least one employee in the "SALES" department**

```
SQL> SELECT emp_name, job_id
        FROM employee
        WHERE dept_id = 10
        AND job_id = ANY (
                                SELECT job_id
                                FROM employee
                                WHERE dept_id = (
                                                SELECT dept_id
                                                FROM department
                                                 WHERE dept_name = 'SALES'
                                                )
                        );
```

Sub-Queries Summary

- **Subqueries allow:**
  - dynamic filtering
  - comparison with computed values
  - expressing complex conditions

- **Common Pitfalls (Very Important)**
- Subquery returns multiple rows with = → ERROR
- NOT IN with NULL values → unexpected results
- Forgetting correlation between queries

Set Operators — UNION

Definition : UNION combines the results of two queries into a single result set; result = **set union (∪)**

- duplicates are automatically removed

Syntax :

```
SQL> SELECT columns
        FROM table1
        WHERE condition
UNION
         SELECT columns
        FROM table2
        WHERE condition;
```

Set Operators – UNION

Important Condition

- Both queries must return:
  → same number of columns
  → compatible data types
  → same logical meaning (same schema)

Example :

List (all department IDs where employees are working)

OR (departments with no employees)

```
SQL> SELECT dept_id
       FROM employee
     UNION
     SELECT dept_id
     FROM department;
```

Set Operators — Intersection

Definition : INTERSECT returns only the rows that appear in BOTH query results;

Result = set Intersection (∩)

- duplicates are automatically removed

Syntax :

```
SQL> SELECT columns
        FROM table1
        WHERE condition
INTERSECT
         SELECT columns
        FROM table2
        WHERE condition;
```

Set Operators — Intersection

Important Condition

- Both queries must return:

  → same number of columns

  → compatible data types

  → same logical meaning (same schema)

- Example : List department IDs that correspond to existing departments with employees.

```
SQL> SELECT dept_id FROM employee
        INTERSECT
        SELECT dept_id FROM department;
```

Set Operators — Difference

Definition : EXCEPT returns rows from the first query that DO NOT appear in the second query .

Result = set difference (–)

Syntax :

```
SQL> SELECT columns
        FROM table1
        WHERE condition
EXCEPT
         SELECT columns
        FROM table2
        WHERE condition;
```

Set Operators — Difference

Important Condition

- Both queries must return:

→ same number of columns

→ compatible data types

→ same logical meaning (same schema)

Question : Departments with NO employees?

```
SQL> SELECT dept_id FROM department
        EXCEPT
        SELECT dept_id FROM employee;
```

Aggregate Functions (Group Functions)

Definition :

Aggregate functions compute a single value from a set of rows.

- **Main Functions :**

Function	Role
AVG(col)	Average
MIN(col)	Minimum value
MAX(col)	Maximum value
SUM(col)	Total sum
COUNT(*)	Number of rows
COUNT(col)	Number of non-NULL values
COUNT(DISTINCT col)	Number of distinct non-NULL values
STDDEV(col)	Standard deviation

Group Functions : Important Rule

- Aggregate functions IGNORE NULL values :
  - ✓ SUM(col): the sum of all non-NULL values in column 'col'.
  - ✓ AVG: (sum of non-NULL values) / (number of non-NULL values).

Example : What is the average salary?

```
SQL> SELECT AVG(salary)
       FROM employee;
```

emp_id	salary
1	1000
2	2000
3	NULL

Result : (1000 + 2000) / 2 = 1500

Basic Examples

Query : Count total number of employees

```
SQL> SELECT COUNT(*)
        FROM employee;
```

Query : Compute total bonuses in department 10

```
SQL> SELECT SUM(bonus)
        FROM employee
        WHERE dept_id = 10;
```

Important Rule (Aggregate Functions)

- Aggregate functions and columns must be at the same level of aggregation ; You **cannot mix** in the same WHERE clause :
  - ✓ a row-level column (salary)
  - ✓ with an aggregate (MAX(salary))

Question : List the names and salaries of employees who earn the highest salary in the company.

⚠ **Incorrect Query**

```
SQL> SELECT emp_name, salary
        FROM employe
        WHERE salary = MAX(salary);
```

Important Rule (Aggregate Functions)

- You **cannot mix** in the same WHERE clause :
  - ✓ a row-level column (salary)
  - ✓ with an aggregate (MAX(salary))

Correct Query (using subquery)

```
SQL> SELECT emp_name, salary
        FROM employee
        WHERE salary = (SELECT MAX(salary) FROM employee);
```

- ✓ The subquery SELECT MAX(salary) FROM employee returns a **single value** (e.g., 5000)
- ✓ The main query becomes:
- ✓ WHERE salary = 5000; Now the comparison is valid (same level)

GROUP BY Clause — Syntax

SELECT [Ai1] [,Ai2]….[,Ain] [F1(Aj1)] ,[F2(Aj2)] , … ,[Fv(Ajv)]

FROM < table expression >

[**WHERE** < search condition>]

GROUP BY Ak1 ,[Ak2], … ,[Akw]

[**HAVING** < Group condition>]

where :

- **Fi**: aggregate function (COUNT, AVG, SUM, MAX, …)
- And {Ak1, Ak2,…., Akw} $\supseteq$ {Ai1, Ai2, …., Ain} : Every non-aggregated attribute in the SELECT clause must appear in the GROUP BY
- And {Ak1, Ak2,…., Akw} $\wedge$ {Aj1, Aj2,…., Ajv} = $\varnothing$: Attributes used inside aggregate functions must **not** be part of the grouping attributes.

GROUP BY Clause — Role and Position

- **Role :**
  - ✓ **Group By partitions a table into groups** :
  - ✓ Each group contains all rows that share the **same values for the grouping attributes**.
  - ✓ Formally: GROUP BY exp1, exp2, ...
  - ✓ All rows with identical values of (exp1, exp2, ...) are grouped together and **collapsed into a single result row** after aggregation.

- **Position in the Query :**

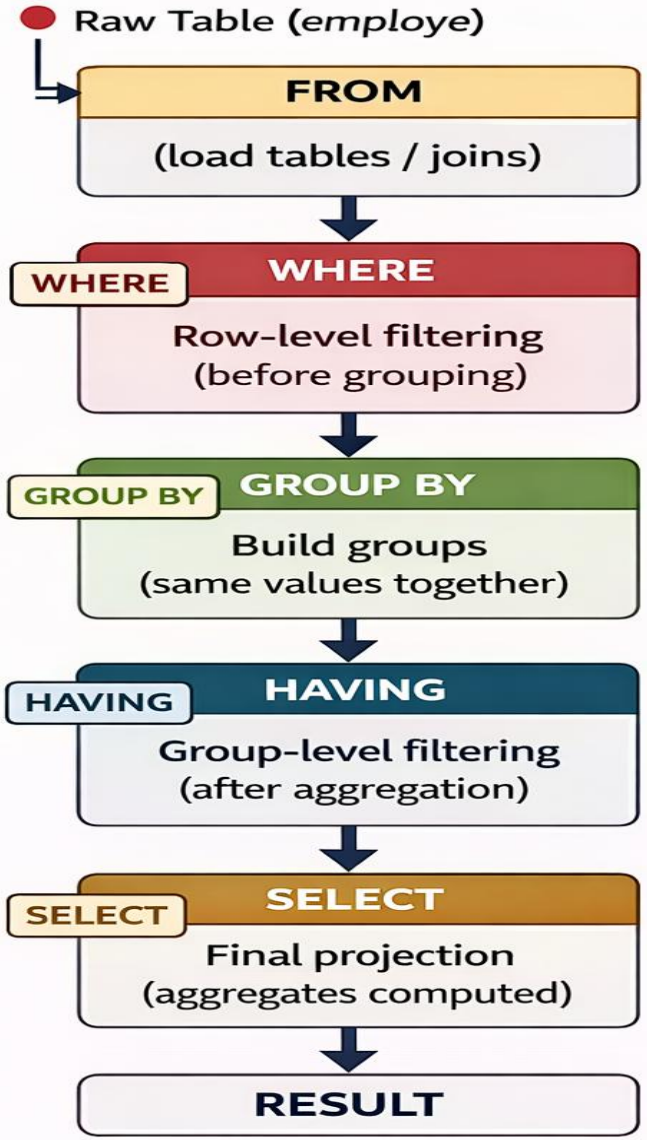
GROUP BY is placed:

- ✓ after the WHERE clause, if it exists
- ✓ otherwise, immediately after the FROM clause

GROUP BY Clause — Execution Process

- **Execute the FROM :** Evaluate the table expression and construct the intermediate result set.
- **Apply the WHERE (if present) :**Filter rows by keeping only those that satisfy the predicate.
- **Execute the GROUP BY :** Partition the remaining rows into groups such that all rows in a group share the same values for the grouping attributes.
  - **NULL values are treated as equal**, meaning they are grouped together and form their own group.
- **Apply aggregate functions (SELECT) :** Compute the specified aggregate functions (e.g., SUM, AVG, COUNT, etc.) for each group.
  - Each group is reduced to **a single row** in the result.
- **Apply the HAVING clause (if present) :** Filter groups based on a condition:
  - Groups for which the condition evaluates to **TRUE** are retained
  - Groups for which it evaluates to **FALSE** or **UNKNOWN** are discarded

SQL Query Execution Flow : WHERE vs GROUP BY vs HAVING



Clause GROUP BY: Examples

- How many employees are there in each department?

```
SQL> SELECT coddep, COUNT(*)
        FROM employe
        GROUP BY coddep;
```

coddep	COUNT(*)
10	3
20	5
30	2

- **Interpretation**

✓ The query **groups employees by department (coddep)**

✓ For each department, it computes: the **number of employees** (COUNT(*))

Clause GROUP BY: Examples

- Compute the number of secretaries per department.

```
SQL >SELECT coddep, COUNT(*)
      FROM employe
      WHERE codposte = 'SECR'
      GROUP BY coddep;
```

- **Interpretation**
  - ✓ WHERE codposte = 'SECR' → keeps only secretaries
  - ✓ GROUP BY coddep → groups them by department
  - ✓ COUNT(*) → counts secretaries in each department

Clause GROUP BY: Examples

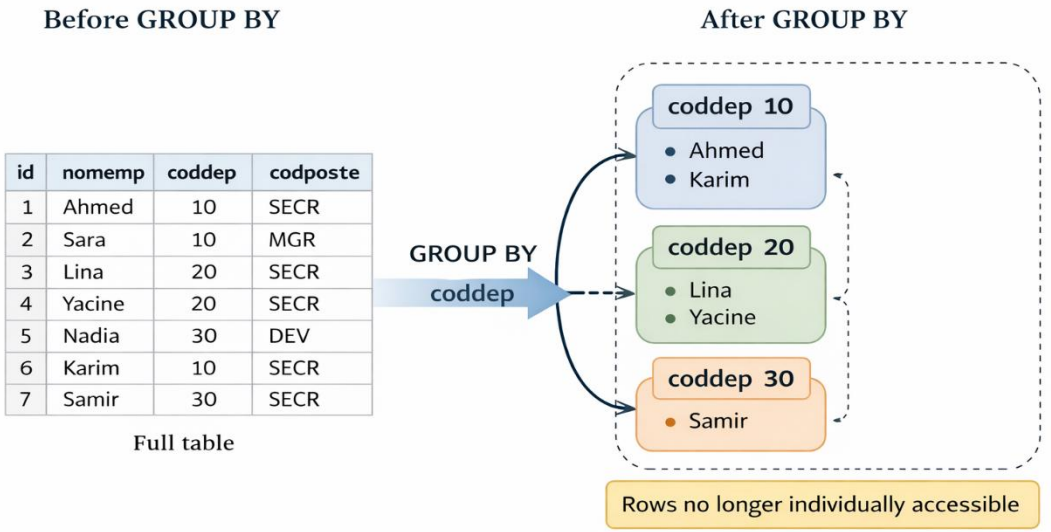
emp_id	name	coddep	codposte
1	Ahmed	10	SECR
2	Sara	10	MGR
3	Lina	20	SECR
4	Yacine	20	SECR
5	Nadia	30	DEV
6	Karim	10	SECR
7	Samir	30	SECR

Clause GROUP BY: Examples

1. Apply WHERE codposte = 'SECR'

Filtered rows:

emp_id	name	coddep	codposte
1	Ahmed	10	SECR
3	Lina	20	SECR
4	Yacine	20	SECR
6	Karim	10	SECR
7	Samir	30	SECR



2. Apply GROUP BY coddep

✓coddep = 10 → {Ahmed, Karim}
✓coddep = 20 → {Lina, Yacine}
✓coddep = 30 → {Samir}

✓Important rule : Après GROUP BY, seules deux choses sont accessibles:

1. Les attributs de regroupement
2. Les fonctions d'agrégation

- 3. Final Result : after Select :

coddep	count
10	2
20	2
30	1

Clause GROUP BY: Examples

Query : Display the number of employees holding the same position in each department.

```
SQL> SELECT coddep, codposte, COUNT(*)
        FROM employe
        GROUP BY coddep, codposte;
```

- The grouping is performed on **two attributes**:
  - coddep (department)
  - codposte (job position)
- Therefore: One group = one (department, position) pair

GROUP BY Constraint

- A SELECT statement that includes a GROUP BY clause must respect the **grouping consistency rule**.

Rule : The SELECT clause must contain only:

- either **grouping columns** (i.e., attributes appearing in the GROUP BY clause),
- or **aggregate functions** (e.g., COUNT, SUM, AVG, MAX, etc.).

Incorrect Query Example :

- *Issue: nomdep is not included in the GROUP BY clause.*

```
SQL> SELECT nomdep, SUM(salaire)
        FROM employe NATURAL JOIN departement
        GROUP BY coddep;
```

Correct vs Incorrect GROUP BY

✕ Incorrect Query

```
SELECT nomdep, SUM(salaire)
FROM employe NATURAL JOIN departement
GROUP BY coddep;
```

- ✕ nomdep is not aggregated
- ✕ nomdep is not in GROUP BY

✕ Violates GROUP BY rule

Rule to Remember

Every column in SELECT must be:

- ✓ An aggregate function (SUM, COUNT, ...)

OR

- ✓ Present in the **GROUP BY** clause

Group	Result
coddep = 10	One row
coddep = 20	One row
coddep = 30	One row

✓ Correct – Option 1

```
SELECT coddep, SUM(salaire)
FROM employe NATURAL JOIN departement
GROUP BY coddep;
```

Group by department ID

coddep	SUM
10	5600
20	7400
30	4200

✓ Correct – Option 2

```
SELECT nomdep, SUM(salaire)
FROM employe NATURAL JOIN departement
GROUP BY nomdep;
```

Group by department name

nomdep	SUM
Sales	5600
IT	7400
HR	4200

Key Takeaway

➡ **You must group by the attribute you want to display.**

Why?

- ✓ Each selected column must describe the group as a whole
- ✓ Not individual rows

HAVING Clause

Definition : A predicate used to specify **which groups should be selected**.

- It is evaluated **after the GROUP BY clause**.
- The **HAVING clause follows the same syntax as WHERE**, but it applies to **groups**, not individual rows.
- It can only refer to:
  - ✓ **aggregate functions** (e.g., COUNT, SUM, AVG, …)
  - ✓ **attributes included in the GROUP BY clause**.

HAVING Clause — Example 1

- **Problem** : Display the departments that have **more than one secretary**.

```
SQL> SELECT dept_id, COUNT(*) AS nb_of_secretaries
        FROM employee
        WHERE job_id = 'SECR'
        GROUP BY dept_id
        HAVING COUNT(*) > 1;
```

Pipeline of execution :

1. FROM → all employees
2. WHERE → keep only 'SECR'
3. GROUP BY → group by coddep
4. HAVING → keep groups with COUNT > 1
5. SELECT → display coddep + count

➢ **Key idea**

- ✓ WHERE filters **rows before grouping**
- ✓ HAVING filters **groups after aggregation**

HAVING Clause – Advanced Example

Problem : Display the **department name** (and its number of secretaries) that has the **maximum number of secretaries**.

```
SQL> SELECT dept_name AS Departement, COUNT(*) AS Nb_secretaires
        FROM employee NATURAL JOIN department
        WHERE job_id = 'SECR'
        GROUP BY dept_name
        HAVING COUNT(*) = (
                                SELECT MAX(COUNT(*))
                                FROM employee
                                WHERE job_id  like 'SECR'
                                GROUP BY dept_id
                          );
```

Advanced example_Pipeline of execution

Step 1 — FROM + JOIN (Initial simplified Table)

After NATURAL JOIN employe ⋈ departement:

Dept_name	Dept_id	Emp_name	Job_id
HR	10	Ali	SECR
HR	10	Sara	SECR
HR	10	Omar	DEV
IT	20	Lina	SECR
IT	20	Samir	DEV
Finance	30	Nadia	SECR
Finance	30	Yacine	SECR
Finance	30	Karim	SECR

Advanced example_Pipeline of execution

Step 2 — After WHERE (codposte = 'SECR')

Dept_name	**Dept_id**	**Emp_name**	**Job_id**
HR	10	Ali	SECR
HR	10	Sara	SECR
IT	20	Lina	SECR
Finance	30	Nadia	SECR
Finance	30	Yacine	SECR
Finance	30	Karim	SECR

Advanced example_Pipeline of execution

Step 3 —After GROUP BY + SELECT COUNT

Department	nb_of_secretaries
HR	2
IT	1
Finance	3

Advanced example_Pipeline of execution

Step4. Extension of sub-query (GROUP BY dept_id)

Dept_id	COUNT
10	2
20	1
30	3

Final Step. After HAVING)

Department	**Nb_of secretaires**
Finance	3

ORDER BY Clause

Definition : ORDER BY specifies **the order in which rows are returned**

Without ORDER BY, the result order is **not guarantee**

Syntax (PostgreSQL) :

SQL> ORDER BY expr1 [ASC | DESC], expr2 [ASC | DESC], …

Note (PostgreSQL Specific) :

✓NULL values are LAST in ASC order

✓NULL values are FIRST in DESC order

Example 1 — Simple Ordering

Question : Retrieve employee names ordered alphabetically.

```
SQL> SELECT last_name
     FROM employee
     ORDER BY last_name ASC;
```

Result :

last_name
Ait Ali
Benali
Mansouri

Example 2 — Descending Order

Question : Retrieve employees ordered by salary (highest first).

```
SQL> SELECT last_name, salary
        FROM employee
        ORDER BY salary DESC;
```

Result :

last_name	salary
Benali	120000
Ait Ali	90000
Mansouri	70000

Example 3 — join + Multiple Columns

Rule :

Sorting priority: first column → second column → …

Question : Retrieve employees ordered by department, then by salary (descending).

```
SQL> SELECT e.last_name, d.dept_name, e.salary
     FROM employee e
     JOIN department d ON e.dept_id = d.dept_id
     ORDER BY d.dept_name ASC, e.salary DESC;
```

NOTES :

- **Order by is performed on the table resulting from the join**
- **Sort by department**
- **Inside each department → sort by salary**

Example 3 — Multiple Columns

- Employee :

emp_id	last_name	dept_id	salary
1	Benali	10	120000
2	Ait Ali	20	90000
3	Mansouri	10	70000
4	Haddad	20	110000
5	Bouchareb	10	95000

Department :

dept_id	dept_name
10	Informatics
20	Finance

- Result :

last_name	dept_name	salary
Haddad	Finance	110000
Ait Ali	Finance	90000
Benali	Informatics	120000
Bouchareb	Informatics	95000
Mansouri	Informatics	70000

ORDER BY — Examples

- **Query** : List the total salaries per department in ascending order

- **Using column position :**

```
SQL> SELECT department_id, SUM(salary) AS total_salaries
     FROM employee
     GROUP BY department_id
     ORDER BY 2;
```

- **Using the aggregate expression :**

```
SQL> SELECT department_id, SUM(salary) AS total_salaries
     FROM employee
     GROUP BY department_id
     ORDER BY SUM(salary);
```

ORDER BY — Examples

➢ **Using the alias**

```
SQL> SELECT department_id, SUM(salary) AS total_salaries
        FROM employee
        GROUP BY department_id
        ORDER BY total_salaries;
```

➢ **Notes :**

- ORDER BY 2 → sorts by the **2nd column in SELECT**
- ORDER BY SUM(salary) → sorts by the **computed aggregate**
- ORDER BY total_salaries → **best practice** (clear and readable)

EXISTS Operator

Definition :

The EXISTS clause is followed by a **subquery enclosed in parentheses** (SELECT …).

- EXISTS returns **TRUE** if the subquery returns **at least one row**.
- Otherwise, it returns **FALSE**.

In many cases, IN can be used instead of EXISTS, but:

EXISTS is generally preferred for **correlated subqueries**.

EXISTS – Example

Problem : Display the names of departments that have **at least one employee earning more than 40,000**.

```
SQL > SELECT department_name
        FROM department
        WHERE EXISTS (
                        SELECT   *
                        FROM     employee e
                        WHERE    e.dept_id = d.dept_id
                        AND      salary > 40000
                        );
```

EXISTS VS IN – Example

Equivalent Query Using IN :

```
SQL > SELECT dept_name
        FROM department d
        WHERE d.dept_id IN (
                              SELECT    e.dept_id
                              FROM      employee e
                              WHERE     e.dept_id = d.dept_id
                              AND       salary > 40000
                              );
```

End of Part SQL (SELECT)