# 3.2 Languages: SQL

- Introduction
- Example database
- SQL sentences
  - Table creation
  - Inserting / updating / deleting rows
  - Queries

#### Introduction

- Structured language used to define, update and query databases
- Proposed by an investigation department of IBM
- Adopted as standard for the database Relational Model in the years 1986-87 (ANSI/X3H2/RDL).
- Since then, several versions have been made: SQL-89, SQL-92,
   SQL:1999, SQL:2003, SQL:2006, SQL:2008, SQL:2011.
- Despite standardization, the different RDBMS manufacturers (Relational Database Management System) offer SQL variants.
- We will see the common SQL core for all RDBMS, considering that for some of RDBMS the syntax of some sentences may differ.
- May be used either in an interactive or in a hosted way in the program sentences.

#### **Example database**

 A relational database consists of Tables (Relations) with a set of Columns (Attributes) and a set of Rows (Tuples).

departments(num_dpt,	name_dpt, flo	oor, buildi	ng,	city_dpt)			
1	DIRECTION 1	10 PAU CLA	RIS	BARCELONA			
2	DIRECTION 8	RIOS RO	SAS	MADRID			
3	MARKETING 1	1 PAU CLA	RIS	BARCELONA			
projects(num_proj, name_proj, product, budget)							
1 IBI	DTEL TELEVISIO	N 1000000					
2 IBE	OVID VIDEO	500000					
employees(num_empl, i	name_empl, sal,	city_empl, nu	ım_dpt, n	um_proj)			
1	CARME 400000	MATARO	1	1			
2 E	UGENIA 350000	TOLEDO	2	2			
3	JOSEP 250000	SITGES	3	1			

#### **Example database**

- Primary key: Each table has a primary key that allows us to identify the rows on the table. E.g. num\_dpt is the primary key of the table departments. This means that each department has a num\_dpt that must be unique among all the departments, i.e. there will never be two departments with the same department number.
- Foreign key. A foreign key to join tuples from different tables or among the tuples of the same table. E.g. num\_dpt is a foreign key of the table employees that refernces the table departments. This means that besides the data from an employee there will also be the department where (s)he belongs. Thanks to the forigh key, we can know the department where an employee works, and also the employees that work on a given department.

departments(num_	_dpt,	name_c	dpt, f	loor,	building	g,	city_dpt)	
	1	DIRECT	ΓΙΟΝ	10	PAU CLAR	RIS	BARCELON	A
	2	DIRECT	ION	8	RIOS ROS	SAS	MADRID	
	3	MARKET	ING	1	PAU CLAR	RIS	BARCELON	A
projects( <u>num_pro</u> j	i, non	n_proj, p	roducte	, buc	dget)			
1	IBI	DTEL T	ELEVISI	0	1000000			
2	IBI	OVID	VIDEO		500000			
employees( <u>num_e</u>	mpl,	name_er	npl, sal	ary, c	city_empl,	num_d	pt num_p	roj)
1		CARME	400000	MAT	ARO	1	1	
2	E	EUGENIA	350000	TOL	EDO	2	2	
3		JOSEP	250000	SIT	GES	3	1	

#### **Table creation**

```
CREATE TABLE <table_name>
    (<column_name> <data_type> [<col_constraints>] [<default_value>]
    [, < column_name> < data_type> [< col_constraints>] [< default_value>]...]
    [< default_table>]);
```

- Data\_type: INTEGER, FLOAT(precision), REAL, CHAR(n), NUMERIC(precision,scale), DECIMAL(precision,scale), SMALLINT, DOUBLE PRECISION, VARCHAR(n), DATE,....
- Default\_value: Default value of a column for a row inserted on the table.
  DEFAULT { < literal> | NULL }.

#### Table creation: table and column constraints

- table\_constraints:
  - UNIQUE (<cols>)
  - PRIMARY KEY (<cols>)
  - FOREIGN KEY (<cols>) REFERENCES [<cols>]
  - CHECK (<conditions>)
- col\_constraints:
  - UNIQUE
  - PRIMARY KEY
  - REFERENCES [<col>]
  - CHECK (<conditions>)
  - NOT NULL

Those constraints affecting more than one column (e.g. primary key formed by two or more columns) are necessarily table constraints.

## **Table creation: Example**

#### **CREATE TABLE** employees

( num\_empl INTEGER,

name\_empl CHAR(30) NOT NULL,

salary INTEGER DEFAULT 100000

**CHECK** (salary>80000),

city\_empl CHAR(30),

num\_dpt INTEGER,

num\_proj **INTEGER**,

PRIMARY KEY (num\_empl),

FOREIGN KEY (num\_dpt) REFERENCES departments(num\_dpt),

FOREIGN KEY (num\_proj) REFERENCES projects(num\_proj));

#### Inserting rows into a table

```
INSERT INTO <table_name> [(<columns>)]
( VALUES {<value<sub>1</sub>> | NULL}, ..., {<value<sub>n</sub>> | NULL} ) | <query> ;
```

- In case we do no specify the column names right after the table\_name, the values must be written exactly in the same order specified in the CREATE\_TABLE sentence.
- In case we specify the column names, the values must correspond to the explicited column order.
- The values of a column of the row or rows to insert can be obtained as a result of a query (see subqueries).

#### Inserting rows into a table: Examples

INSERT INTO employees
VALUES (4, 'RICARDO', 400000, 'BARCELONA',1,1);

**VALUES** (11, 3, 2, 'NURIA');

employees	s( <u>num_em</u> p	<u>l,</u> name_er	npl, sala	ary, city_empl,	num_dpt	num_proj )
	1	CARME	400000	MATARO	1	1
	2	EUGENIA	350000	TOLEDO	2	2
	3	JOSEP	250000	SITGES	3	1
	4	RICARDO	400000	BARCELONA	1	1
	11	NURIA	100000	NULL	3	2

# **Deleting rows from a table**

**DELETE FROM** 

**WHERE** <conditions>;

Rows that match the specified conditions in the clause WHERE are deleted from the table.

## **Deleting rows from a table: Examples**

**DELETE FROM** employees **WHERE** num\_dpt=2;

**DELETE FROM** employees **WHERE** salary <= 250000;

employees( <u>num_en</u>	<u>ıpl,</u> name_en	npl, sala	ry, city_empl,	num_d	pt num_p	roj )
1	CARME	400000	MATARO	1	1	
2	EUGENIA	350000	TOLEDO	2	2	
3	JOSEP	250000	SITGES	3	1	
4	RICARDO	400000	BARCELONA	1	1	
11	NURIA	100000	NULL	3	2	$\sim$

## **Updating rows from a table**

**UPDATE**

**SET** <col> = {expression/ NULL} [,<col> = {expression/ NULL}...]

**WHERE** <conditions>;

The rows columns of the table that meet the specified conditions in the WHERE clause are updated.

## **Updating rows from a table: Examples**

UPDATE employees
SET salary = salary +10000
WHERE num\_dpt = 1;

UPDATE employees
SET salary = salary + 50000, city\_empl = 'VIC'
WHERE num\_empl = 11;

employees	s( <u>num</u> _	<u>empl,</u> name_em	pl, sala	ry, city_emp	ol, (num_c	lpt num_proj)
	1	CARME	410000	MATARO	1	1
	2	EUGENIA	350000	TOLEDO	2	2
	3	JOSEP 2	250000	SITGES	3	1
	4	RICARDO 4	410000 I	BARCELONA	1	1
	11	NURIA	150000	VIC	3	2

Rows updated by the first sentence
Rows updated by the second sentence

#### Queries on a table: basic format

```
SELECT <columns_to_select> | *
FROM <table_to_query>
[ WHERE <conditions> ];
```

- The result of the query is the value of the columns\_to\_select from table\_to\_query only for the row or rows that match the conditions specified in the clause WHERE.
- In case we do not specify the clause WHERE, the result is the value of columns\_to\_select for all the rows of the table\_to\_query.
- If we write a \* instead of **columns\_to\_select** it denotes that we are interested in all the columns from the **table\_to\_query**.

# Queries on a table: basic format - Example 1

**SELECT** \* **FROM** employees;

employees	( <u>num_emp</u>	<u>l,</u> name_er	npl, sala	ry, city_empl,	num_dpt	num_proj	)
	1	CARME	410000	MATARO	1	1	
	2	EUGENIA	350000	TOLEDO	2	2	
	3	JOSEP	250000	SITGES	3	1	-(1
	4	RICARDO	410000	BARCELONA	1	1	
	11	NURIA	150000	VIC	3	2	

## Queries on a table: basic format - Example 2

**SELECT** num\_empl, name\_empl, salary **FROM** employees;

employees	( <u>num_en</u>	<u>npl,</u> name_er	npl, sala	ry, city_empl,	num_c	lpt num_proj )
	1	CARME	410000	MATARO	1	1
	2	EUGENIA	350000	TOLEDO	2	2
	3	JOSEP	250000	SITGES	3	1
	4	RICARDO	410000	BARCELONA	1	1
	11	NURIA	150000	VIC	3	2

## Queries on a table: basic format - Example 3

SELECT num\_empl, name\_empl,
 salary
FROM employees
WHERE num\_dpt = 3;

employees	( <u>num_emp</u>	o <u>l</u> , name_er	mpl, salar	y, city_empl,	num_d	pt (num_proj )
	1	CARME	410000	MATARO	1	1
	2	EUGENIA	350000	TOLEDO	2	2
	3	JOSEP	250000	SITGES	3	1
	4	RICARDO	410000	BARCELONA	1	1
	11	NURIA	150000	VIC	3	2

#### Operators in the conditions

#### operators

- aritmethic: \*, +, -, /
- comparators: =, <, >, <=, >=, <>
- logical: NOT, AND, OR
- other:
  - <column> BETWEEN limit<sub>1</sub>> AND limit<sub>2</sub>>
  - <column> IN (<value<sub>1</sub>>,<value<sub>2</sub>> [....,<value<sub>N</sub>>])
  - <column> LIKE <characteristic>
  - <column> IS [NOT] NULL

#### These operators may appear in the **conditions**:

- In the clause WHERE of the deleting sentences (DELETE), modification (UPDATE) and query (SELECT)
- In the clause CHECK of the sentences of table creation (CREATE TABLE).

#### **Operators in the conditions: Example**

```
FROM employees

WHERE NOT(num_dpt = 2) AND

( city_empl IN ( 'MATARO',  'SITGES',  'BARCELONA') OR
    city_empl LIKE 'V%') AND
    num_proj IS NOT NULL AND
    salary BETWEEN 400000 AND 500000;
```

employees	( <u>num_em</u> p	ol, name_er	npl, sala	ry, city_empl,	num_c	dpt num_proj
	1	CARME	410000	MATARO	1	1
	2	EUGENIA	350000	TOLEDO	2	2
	3	JOSEP	250000	SITGES	3	1
	4	RICARDO	410000	BARCELONA	1	1
	11	NURIA	150000	VIC	3	2

## Queries on a table: Sorting

```
SELECT <columns_to_select> | *
FROM <table_to_query>
[ WHERE <conditions> ]
ORDER BY <column> [DESC | ASC],....;
```

- In the resultin data is ordered by the columns indicated in the ORDER BY clause.
- If we do not specify **DESC** for a column, we assume that we want the values of the specified attributte ordered ascendingly. We can explicitally ask for this order with the word **ASC**.

## **Queries on a table: Sorting - Example**

SELECT num\_empl, name\_empl, salary

FROM employees

result

WHERE num\_dpt IN (1,2)

ORDER BY salary DESC, name\_empl;

nur	m_empl	name_en	npl salar	У
	1	CARME	410000	
	4	RICARDO	410000	
	2	EUGENIA	350000	
	12	NURIA	150000	

employees	(num_emp	ol, name_er	npl, sala	ry, city_empl,	num_d	pt num_	proj )
	1	CARME	410000	MATARO	1	1	1
	2	EUGENIA	350000	TOLEDO	2	2	
	3	JOSEP	250000	SITGES	3	1	
	4	RICARDO	410000	BARCELONA	1	1	
	11	NURIA	150000	VIC	3	2	
	12	NURIA	150000	MATARO	1	5	

Rows matching the condition of the WHERE clause
Ordered data as a result of the query

#### Queries on a table: results without repetitions

SELECT [ DISTINCT | ALL] <columns\_to\_select>
FROM <table\_to\_query>
[ WHERE <conditions> ];

- If we want the result of a query to be given without repetitions, we need to use the keyword **DISTINCT**.
- If we do not specify anything we will get the result with repetitions (if they exist). We can ask for this repetitions explicitally with the keyword ALL.

#### Queries on a table: results without repetitions

SELECT DISTINCT name\_empl, salary

salary

FROM employees

WHERE num\_dpt IN (1,3);

name\_empl salary

CARME 410000

JOSEP 250000

RICARDO 410000

NURIA 150000

employees	s( <u>num</u>	<u>empl</u> , name_en	npl, sala	ry, city_empl,	num_	dpt num_proj
	1	CARME	410000	MATARO	1	1
	2	EUGENIA	350000	TOLEDO	2	2
	3	JOSEP	250000	SITGES	3	1
	4	RICARDO	410000	BARCELONA	1	1
	11	NURIA	150000	VIC	3	2
	12	NURIA	150000	MATARO	1	5

Rows matching the condition of the WHERE clause
Data obtained as a result of the query

#### Queries on a table: Aggregate functions

SELECT <aggregate\_functions>
FROM <table\_to\_query>
[ WHERE <conditions> ];

- They are functions applied over the set of rows from table\_to\_query that match the conditions specified on the clause WHERE.
  - COUNT:
    - COUNT(\*) number of rows matching the conditions of the where clause
    - **COUNT(DISTINCT <column>)** number of different values of the colum, for the rows matching the conditions of the where clause.
    - COUNT(<column>) number of values of the column, without counting NULL values, for the rows matching the condition of the where clause.
  - SUM (expression), MIN(expression), MAX(expression), AVG(expression):
    - Expression may be simply a column, or maybe a calculation from different column values and constants.
    - SUM: it gives us the sum of the values resulting from computing the expression for the rows that match the condition of the WHERE clause
    - **MIN**: it gives us the minimum value resulting from computing the expression for the rows that match the condition for the WHERE
    - MAX: it gives us the maximum value resulting from computing the expression for the rows that match the condition for the WHERE
    - AVG: it gives us the average value resulting from computing the expression for the rows that match the condition for the WHERE

# **Queries on a table: Aggregate functions - Example**

SELECT COUNT(\*) AS quantEmpl ,

COUNT(DISTINCT name\_empl) AS quantNames,
quantNames,
SUM(salary\*0.1) AS partsalary

FROM employees

WHERE num\_dpt IN (1,3);

quantEmpl quantNames partsalary
5 4 137000

employees	( <u>num_</u> e	<u>empl,</u> name_en	npl, sala	ry, city_empl,	num_d	pt num_proj	)
	1	CARME	410000	MATARO	1	1	
	2	EUGENIA	350000	TOLEDO	2	2	
	3	JOSEP	250000	SITGES	3	1	
	4	RICARDO	410000	BARCELONA	1	1	
	11	NURIA	150000	VIC	3	2	
	12	NURIA	150000	MATARO	1	5	

Rows matching the condition of WHERE clause

Result of the query

#### Queries on a table: Group rows

SELECT <columns\_to\_select> [,<aggregate\_functions>]
FROM <table\_to\_query>

[ WHERE <conditions> ]

**GROUP BY** <columns\_to\_group\_by>;

- We organize in groups the rows of the table\_to\_query to match the conditions specified in the WHERE clause, depending on its value for the columns\_to\_group\_by
- The result of the query is the value of the **columns\_to\_select** for each of the row groups obtained.
- The columns\_to\_select must be columns containing the same value for all the rows inside a group.
- In the result we can also ask for the value of aggregate\_functions that are calculated for each of the row groups obtained.

#### Queries on a table: Group rows - Example

SELECT num\_dpt,

COUNT(\*) AS quantEmpl

FROM employees

WHERE num\_dpt IN (1,3)

GROUP BY num\_dpt;

num\_dpt quantEmpl

1 3
3 2

	employee	es( <u>num_</u> e	empl, name_en	npl, sala	ry, city_emp	I, num_d	pt num_pro	
grup1		<u> </u>	CARME	410000	MATARO	<u> </u>	/////	
		2	EUGENIA	350000	TOLEDO	2	2	
		3	JOSEP	250000	SITGES	3	1	
grup2		4///	RICARDO	410000	BARCELONA	<u> </u>	<u> </u>	
		11	NURIA	150000	VIC	3	2	
		12//	NURIA	150000	MATARO	<u> </u>	////5////	

Obtained rows matching the WHERE condition, grouped according to the GROUP BY clause
Obtained data as a result of the query. There is a result for each group

#### Queries on a table: conditions over groups

```
SELECT <columns_to_select> [,<aggregate_functions>]
FROM <table_to_query>
[WHERE <conditions>]
GROUP BY <columns_to_group_by>
HAVING <conditions_for_groups>;
```

- If we add the clause HAVING, the result will only appear for the groups matching the conditions\_for\_groups.
- The conditions\_for\_groups will be comparisons between constants, values from columns with groups defined for, and values of aggregate functions.
- It only makes sense to apply aggregate functions for the columns that are not in the set columns\_to\_group\_by.

## Queries on a table: conditions over groups - Example 1

SELECT num\_dpt, SUM(salary) AS
salariesSum

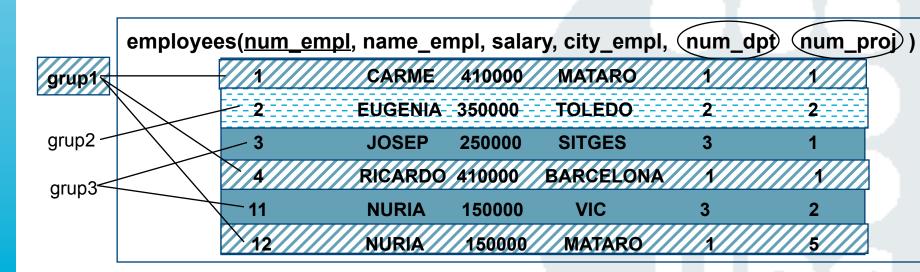
FROM employees

GROUP BY num dpt

num\_dpt salariesSum

1 970000

**HAVING** COUNT(\*) >= 3;



Since there is no WHERE, all the rows are grouped according to the GROUP BY clause

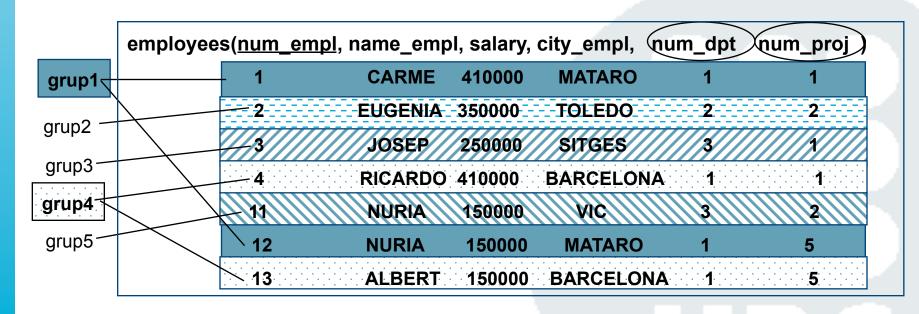
Obtained data as a result of the query, a result for each group matching the condition of HAVING.

## Queries on a table: conditions over groups - Example 2

FROM employees result num\_dpt

GROUP BY num\_dpt, city\_empl

HAVING COUNT(\*) >= 2;



Since there is no WHERE, all the rows are grouped according to the GROUP BY clause
Obtained data as a result of the query, a result for each group matching the condition of HAVING, without repeated results because of the DISTINCT clause.

#### Queries on more than one table: Basic format

```
SELECT <columns_to_select> | *
FROM <tables_to_query>
[ WHERE <conditions> ];
```

- The result of the query is the value of the **columns\_to\_select** from **tables\_to\_query** only for the row(s) matching the specified **conditions** in the **WHERE** clause.
- If we do not specify the WHERE clause, the result is the value of columns\_to\_select for the rows obtanied of the cartesian product of the rows in tables\_to\_query.
- If we write \* instead of **columns\_to\_select** we denote that we are interested in all the columns of **tables\_to\_query**.

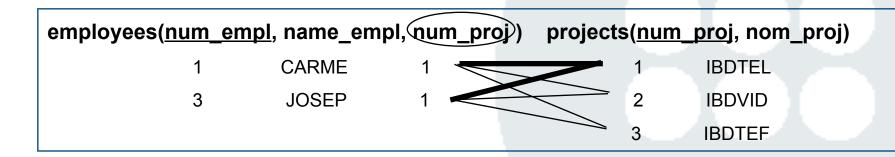
#### Queries on more than one table: Basic format – Example 1

e.num\_empl e.name\_empl e.num\_proj p.num\_proj p.nom\_proj **CARME IBDTEL CARME IBDVID** result **SELECT** CARME 3 **IBDTEF FROM** employees e, projects p; **JOSEP IBDTEL** 3 3 **JOSEP IBDVID JOSEP IBDTEF** 3 3

employees(num_er	npl, name_em	pl, num_proj))	projects( <u>num_proj</u> , nom_proj)
1	CARME	1	1 IBDTEL
3	JOSEP	1	2 IBDVID
			3 IBDTEF

Obtained data as a result of the query. Notice that, from all rows, the ones that we are probably interested on are marked in bold (they are the ones in which the project number where the employee is working is the same as the project number from the Projects table).

#### Queries on more than one table: Basic format – Exemple 2



Obtained data as a result of the query. Note that the only combinations appearing are the ones in which the project number where the employee works match with the project number from the Project table.

#### Queries on more than one table Alternative syntaxis – Inner Join clause - Example 3

The condition of the table combination can be written as:

- Either in the WHERE clause
- Or either using the JOIN clause in FROM
  - ➤ INNER JOIN requires the condition of combination explicitally
  - ➤ NATURAL INNER JOIN makes the combination por the columns with the same name on the implied tables.

```
SELECT e.num_empl, p.num_proj, p.nom_proj

FROM employees e, projects p

WHERE e.num_proj = p.num_proj;
```

**SELECT** e.num\_empl, p.num\_proj, p.nom\_proj **FROM** employees e **INNER JOIN** projects p **WHERE** e.num\_proj = p.num\_proj

SELECT e.num\_empl, p.num\_proj, p.nom\_proj FROM employees e NATURAL INNER JOIN projects p;

The three previous sentences are equivalent.

#### Queries on more than one table: General

- In the queries from more than one table we can apply all the variants shown in this chapter for the queries on one table. I.e.
  - Results without repetitions
  - Aggregate functions
  - Data sorting
  - Row grouping
  - **–** ...



#### Queries on more than one table: Example with row grouping

SELECT p.num\_proj, p.nom\_proj

FROM projects p, employees e

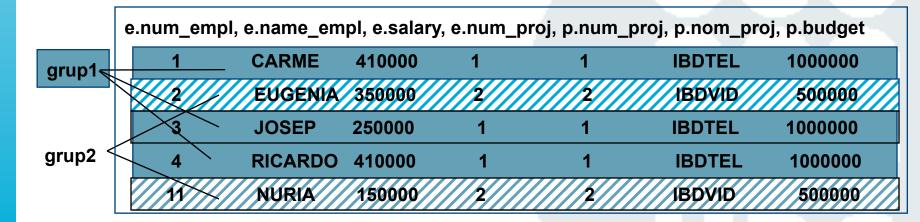
WHERE p.num\_proj = e.num\_proj

GROUP BY p.num\_proj, p.nom\_proj, p.budget

HAVING p.budget < SUM(e.salary);

p.num\_proj,p.nom\_proj

1 IBDTEL



Resulting rows from the combination of employees with the projects, grouped according to the GROUP BY clause
Obtained data as a result of the query. A result for each group matching the condition of HAVING.
There is only one project such that its budget is lower than the sum of salaries of the employees working on it.

#### **Queries: Union**

```
SELECT <columns_to_select> | *

FROM <tables_to_query>
[WHERE <conditions> ]

UNION

SELECT <columns_to_select> | *

FROM <tables_to_query>
[WHERE <conditions> ]

[ORDER BY <column> [DESC|ASC],...];
```

- El result is the union of the result obtained from the two SELECT sentences.
- The columns\_to\_select in the two sentences SELECT have to be semantically compatible
- All the results will be with no repetitions (in most SGBDR are already sorted).
- The columns appearing at the ORDER\_BY clause have to be a subset of the columns\_to\_select from the first SELECT.

# **Queries: Union - Example**

SELECT city\_empl

FROM employees

UNION

SELECT city\_dpt

FROM departments

ORDER BY city\_empl DESC;

city\_empl

SITGES

MATARO

MADRID

BARCELONA

employees( <u>num_empl</u> , name_empl, city_empl)			departments( <u>num_dpt</u> , city_dpt)		
1	CARME	MATARO	1	BARCELONA	
3	JOSEP	SITGES	2	MADRID	
			3	BARCELONA	

Obtained data as a result of the query.

#### **Queries: Difference**

```
SELECT < columns to select> | *
FROM <tables to query>
WHERE  NOT IN ( SELECT <column to select>
                                FROM < tables to query>
                                [ WHERE <conditions> ]);
SELECT < columns to select > | *
FROM <tables to query>
WHERE NOT EXISTS ( SELECT *
                     FROM <tables to query>
                     WHERE <conditions>);
```

- There are two alternative ways to make a difference: with a NOT IN or with NOT EXISTS.
- A NOT IN is true if the value of the column *table\_column* is not in the result of the subquery.
- A NOT EXISTS is true if the subquery does not return any result.
- There are other ways of making a difference (see Except operator in standard SQL), and there are also different operator names in different systems.

#### **Queries: Difference – Examples**

```
SELECT p.num_proj, p.nom_proj

FROM projects p

WHERE p.num_proj NOT IN (SELECT e.num_proj

FROM employees e);

p.num_proj p.nom_proj

result

2 IBDVID

3 IBDTEF

WHERE NOT EXISTS (SELECT * FROM employees e

WHERE p.num proj = e.num proj);
```

employees( <u>num_empl</u> , name_empl, <u>num_proj</u> )			oj) projects( <u>num_proj</u> , nom_proj)
1	CARME	1	1 IBDTEL
3	JOSEP	1	2 IBDVID
			3 IBDTEF
			4 IBDCOM

Obtained data as a result of the query, in any of the two alternatives.

In any case, the query gives those projects without any assigned employee.

NOT IN: A project is in the result of the query if its there is no employee assigned to it (num\_proj from employees). NOT EXISTS: A project is at the result of the query if it does not exist any employee with this project number.

# Subqueries in the sentences Delete, Update i Select

■ They can appear in those sentences where there is the clause **WHERE**:

Deleting rows from a table
 DELETE FROM 
 WHERE .... (SELECT ..... );

Modifying the rows of a table
 UPDATE

SET ....

WHERE ..... (SELECT ..... );

Queries to one or more tablesSELECT <columns\_to\_select>

FROM <tables\_to\_query>

WHERE ..... (SELECT ..... );

## Subqueries in delete sentences - Example

**DELETE FROM** projects

WHERE NOT EXISTS (SELECT \*

FROM employees e

WHERE e.num\_proj = projects.num\_proj);

employees( <u>nı</u>	um_emp	<u>l,</u> name_empl,	num_proj	) proje	ects( <u>num</u>	<u>_proj</u> , nom_p	roj)
	1	CARME	1		1	IBDTEL	
	3	JOSEP	1		2	IBDVID	
					3	IBDTEF	
					4	IBDCOM	

Deleted rows as a result of the query. The sentence deletes those projects without any employee assigned to. Note that the subquery gets those projects with at least one employee.

## Subqueries in update sentences - Example

employees( <u>num_empl</u> , name_empl, <u>num_proj</u> )			projects( <u>num_proj</u> , budget)			
1	CARME	1			1	1100000
3	JOSEP	1			2	500000
					3	4500000
			1		4	2000000

Deleted rows as a result of the query. The sentence increases the budget for those projects having too or more employees assigned to them. Note that the subquery is calculating the number of employees of each project to update

## Subqueries in select sentences – Example 1

**SELECT** e.num\_empl, e.name\_empl **FROM** employees e **WHERE** e.salary > ( **SELECT** AVG (e1.salary)

FROM employees e1);

employees( <u>num_empl</u> , name_empl, salary, city_empl, <u>num_dpt</u> <u>num_proj</u> )								
	1	CARME	410000	MATARO	1	1		
	2	EUGENIA	350000	TOLEDO	2	2		
	3	JOSEP	250000	SITGES	3	1		
	4	RICARDO	410000	BARCELONA	1	1 1		
	11	NURIA	150000	VIC	3	2		

Data obtained as a result of the sentence. The sentence returns the employees with bigger salary than the mean salary of all employees. Note that the subquery returns the mean salary of all employees in the employees table.

#### **Subqueries in select sentences – Example 2**

FROM projects p

WHERE p.budget < (SELECT SUM(e.salary)

FROM employees e

WHERE e.num proj=p.num proj);

P.num\_proj,p.nom\_proj

p.num\_proj,p.nom\_proj

- Data obtained as a result of the sentence. The sentence returns the projects with a budget lower than the sum of the salaries of the assigned employees. Note that the subquery returns the sum of the salaries of the employees assigned to the project that is being considered.
  - Selected rows in the subquery when considering the project number 1. The sum of the salaries is 1070000.
- Selected rows in the subquery when considering the project number 2. The sum of the salaries is 500000.

## Subqueries in insert sentences and Having clauses

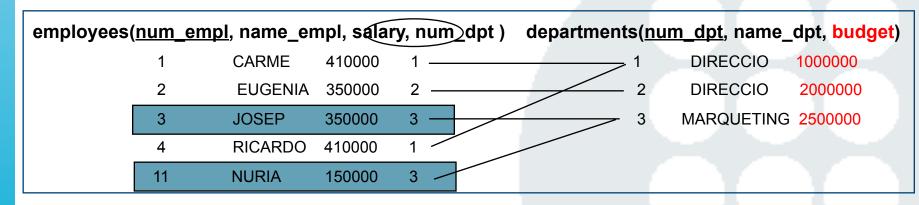
In those sentences where there is the HAVING clause:

```
FROM <tables_to_select>
FROM <tables_to_query>
WHERE <conditions>
GROUP BY <grouping_columns>
HAVING ..... (SELECT ..... );
```

■ Finally, in insert sentences of rows in a table (in this case the result of the subquery is a set of rows that must be compatible with the definition of the table in the **CREATE TABLE**)

```
INSERT INTO 
(SELECT .....);
```

# **Subqueries in Having clauses – Example**



- Data obtained as a result of the sentence. The sentence returns the departments in which the sum of the salary of their employees is bigger than the sum of the salaries of the employees on the department 3. Note that the subquery return the sum of the salary of the employees working on department number 3.
  - Selected rows in the subquery. The sum of the salaries is 500000.

## **Subqueries in Insert sentences – Example**

INSERT INTO clients
 (SELECT num\_empl,name\_empl,200000
 FROM employees
 WHERE num\_dpt IN (2,3));

