

LLENGUATGE DE CONSULTA SQL

Una consulta serveix per extreure informació d'una base de dades. També permet manipular dades com: afegir, eliminar i canviar. Pel que podem definir que una consulta és aquella sol·licitud que es dona a una base de dades per extreure o manipular dades. No obstant això, no només s'ha d'escriure una "sol·licitud" aleatòria, hem d'escriure una consulta basant-nos en un conjunt de codis predefinits, de manera que la base de dades pugui entendre la instrucció. Per dur-ho a terme utilitzarem un llenguatge de consulta, el llenguatge SQL (Structured Query Language). [Llenguatge SQL](#)

CONSULTES SENZILLES

La sol·licitud d'informació a una base de dades es realitza a través d'una sentència anomenada **SELECT**, que permet seleccionar les columnes que es mostraran i en l'ordre en què es farà. Simplement, és la comanda que la base de dades interpreta com que sol·licitarem informació. Aquesta comanda només obté informació, i tot i que la sortida es pot formatar de diferents formes, es poden utilitzar funcions, expressions, etc. **en cap cas es modifica la informació de la base de dades.**

Selecció

Taula 1

Projecció

Taula 1

Taula 1

Join



Taula 2

L'estructura d'una sentència SELECT bàsica és la següent:

```
SELECT    * | { [DISTINCT] column | expression [alias], ... }
[FROM    table];
```

A les "paraules reservades" del llenguatge que ens permeten construir les sentències SQL les anomenem **clàusules** o **paraules clau**.

Clàusules bàsiques:

- **SELECT:** Identifica quines columnes.
- **FROM:** Identifica quines taules.

[!Note]

En alguns SGBD com Postgresql la clàusula FROM no és obligatòria.
Llavors la sentència SQL no fa referència a cap taula.

Exemple: Selecciona totes les files de la taula *departments*

```
SELECT *  
FROM departments
```

department_id	department_name	location_id
1	Administration	1700
2	Marketing	1800
3	Purchasing	1700
4	Human Resources	2400
5	Shipping	1500
6	IT	1400
7	Public Relations	2700
8	Sales	2500
9	Executive	1700
10	Finance	1700
11	Accounting	1700
12	Nou dep	1500
13	Nou dep 1	1400
14	Nou dep	1500
15	Dep nou	1400

(15 rows)

Exemple:

Selecciona determinades columnes de la taula *departments*

```
SELECT DEPARTMENT_ID, LOCATION_ID  
FROM DEPARTMENTS;
```

department_id	location_id
1	1700
2	1800
3	1700
4	2400
5	1500
6	1400
7	2700
8	2500
9	1700
10	1700
11	1700
12	1500
13	1400
14	1500
15	1400

(15 rows)

SINTAXI SENTÈNCIES SQL

- Depèn de l'entorn són sensibles a majúscules/minúscules. Es recomana utilitzar les paraules clau SQL en majúscula.
- Poden ocupar una o diverses línies.
- Les paraules clau no poden ser abreujades ni dividides.

- Les clàusules solen estar col·locades en línies separades.
- S'utilitzen sagnies per millorar la lectura.

Exemple:

```
SELECT employee_id, first_name, salary
FROM employees
WHERE employee_id = 100;
```

EXPRESSIONS ARITMÈTIQUES

Podem crear expressions amb dades numèriques i dates mitjançant operadors aritmètics. No es poden utilitzar en la clàusula FROM.

Operador	Descripció
+	Suma
-	Resta
*	Multiplicació
/	Divisió

Exemple:

```
SELECT first_name, salary, salary + 100
FROM employees;
```

first_name	salary	?column?
Nancy	12000.00	12100.00
Daniel	9000.00	9100.00
John	8200.00	8300.00
Ismael	7700.00	7800.00
Jose Manuel	7800.00	7900.00
Luis	6900.00	7000.00
Alexander	3100.00	3200.00
Alexander	9210.00	9310.00
Steven	15000.00	15100.00
Lex	18000.00	18100.00
David	5010.00	5110.00
Neena	23000.00	23100.00
Valli	5010.00	5110.00
Bruce	12210.00	12310.00
Diana	4410.00	4510.00
Den	11000.00	11100.00
(16 rows)		

Prioritat dels operadors:

- La multiplicació i la divisió tenen prioritat sobre la suma i la resta.
- Els operadors de la mateixa prioritat s'avaluen d'esquerra a dreta.
- Els parèntesis poden ser usats per canviar la prioritat d'avaluació i per clarificar sentències.

Exemples:

```
SELECT first_name, salary, 12*salary+100
FROM employees;
```

first_name	salary	?column?
Nancy	12000.00	144100.00
Daniel	9000.00	108100.00
John	8200.00	98500.00
Ismael	7700.00	92500.00
Jose Manuel	7800.00	93700.00
Luis	6900.00	82900.00
Alexander	3100.00	37300.00
Alexander	9210.00	110620.00
Steven	15000.00	180100.00
Lex	18000.00	216100.00
David	5010.00	60220.00
Neena	23000.00	276100.00
Valli	5010.00	60220.00
Bruce	12210.00	146620.00
Diana	4410.00	53020.00
Den	11000.00	132100.00
(16 rows)		

```
SELECT first_name, salary, 12*(salary+100)
FROM employees;
```

first_name	salary	?column?
Nancy	12000.00	145200.00
Daniel	9000.00	109200.00
John	8200.00	99600.00
Ismael	7700.00	93600.00
Jose Manuel	7800.00	94800.00
Luis	6900.00	84000.00
Alexander	3100.00	38400.00
Alexander	9210.00	111720.00
Steven	15000.00	181200.00
Lex	18000.00	217200.00
David	5010.00	61320.00
Neena	23000.00	277200.00
Valli	5010.00	61320.00
Bruce	12210.00	147720.00
Diana	4410.00	54120.00
Den	11000.00	133200.00
(16 rows)		

EL VALOR NUL

- Valor **nul** és no disponible, no assignat, desconegut o no aplicable.
- Valor nul no és el mateix que zero ni que un espai en blanc.

Exemple:

```
SELECT first_name, salary, commision
FROM employees;
```

first_name	salary	commision
Nancy	12000.00	
Daniel	9000.00	
John	8200.00	
Ismael	7700.00	2.50
Jose Manuel	7800.00	
Luis	6900.00	0.10
Alexander	3100.00	
Alexander	9210.00	
Steven	15000.00	
Lex	18000.00	2.01
David	5010.00	
Neena	23000.00	1.01
Valli	5010.00	
Bruce	12210.00	
Diana	4410.00	1.20
Den	11000.00	0.21
(16 rows)		

- Depenent de l'editor (psql, pgAdmin...) el valor nul es representarà en blanc o bé amb l'anotació (null) o [null].
- Qualsevol expressió aritmètica que contingui valors nuls s'avaluarà com a nul·la.

Exemple: Operació aritmètica sobre la columna 'commision' que té valors nul.

```
SELECT first_name, 12*salary+commision
FROM employees;
```

first_name	salary	?column?
Nancy	12000.00	
Daniel	9000.00	
John	8200.00	
Ismael	7700.00	92430.00
Jose Manuel	7800.00	
Luis	6900.00	82801.20
Alexander	3100.00	
Alexander	9210.00	
Steven	15000.00	
Lex	18000.00	216024.12
David	5010.00	
Neena	23000.00	276012.12
Valli	5010.00	
Bruce	12210.00	
Diana	4410.00	52934.40
Den	11000.00	132002.52
(16 rows)		

DEFINICIÓ D'ÀLIES DE COLUMNA

- Com hem vist en els exemples anteriors quan utilitzem expressions el nom de la columna quan se'n presenta en el resultat és "?column?". Un àlies de columna canvia el nom d'una capçalera de columna.
- És útil especialment en càlculs.
- Se situa darrere del nom de la columna. (la paraula **AS** és opcional i s'utilitza entre la columna i l'àlies).
- Requereix tancar-lo entre cometes dobles si conté caràcters especials (espais, guions, accents...)

Exemple: Us d'àlies en diferents columnes, utilitzant diferent sintaxi.

```
SELECT first_name name, salary AS sal, 12*salary AS "Anual Salary"
FROM employees;
```

name	sal	Anual Salary
Nancy	12000.00	144000.00
Daniel	9000.00	108000.00
John	8200.00	98400.00
Ismael	7700.00	92400.00
Jose Manuel	7800.00	93600.00
Luis	6900.00	82800.00
Alexander	3100.00	37200.00
Alexander	9210.00	110520.00
Steven	15000.00	180000.00
Lex	18000.00	216000.00
David	5010.00	60120.00
Neena	23000.00	276000.00
Valli	5010.00	60120.00
Bruce	12210.00	146520.00
Diana	4410.00	52920.00
Den	11000.00	132000.00
(16 rows)		

OPERADOR DE CONCATENACIÓ

- Concatena columnes o cadenes de caràcters a altres columnes.
- Està representat per dues barres verticals ||
- Crea una columna resultat que és una expressió de caràcters.

Exemple: Consulta que concatena els valors dels camps first_name i last_name afegint un espai en blanc.

```
SELECT first_name || ' ' || last_name
FROM employees;
```

Nom complert
Nancy Greenberg
Daniel Faviet
John Chen
Ismael Sciarra
Jose Manuel Urman
Luis Popp
Alexander Khoo
Alexander Hunold
Steven King
Lex De Haan
David Austin
Neena Kochhar
Valli Pataballa
Bruce Ernst
Diana Lorentz
Den Raphaely
(16 rows)

CADENES DE CARÀCTERS LITERALS.

- Un literal és un caràcter, número o data inclòs a la SELECT
- Els valors literals de tipus data i caràcter han d'escriure's entre cometes simples.
- Cada cadena de caràcters té una sortida per a cada fila retornada.

[!IMPORTANT]

A diferència de la majoria de llenguatges de programació que representen les cadenes de caràcters o string amb cometes dobles " en el llenguatge SQL utilitzem cometes simples '

Exemple:

```
SELECT first_name || ' works in ' || department_name "Name and department"
FROM employees e JOIN departments d
ON e.department_id = d.department_id;
```

```
      Name and department
-----
Nancy works in Finance
Daniel works in Finance
John works in Finance
Ismael works in Finance
Jose Manuel works in Finance
Luis works in Finance
Alexander works in Purchasing
Alexander works in IT
Steven works in Executive
Lex works in Executive
David works in IT
Neena works in Executive
Valli works in IT
Bruce works in IT
Diana works in IT
Den works in Purchasing
(16 rows)
```

FILES DUPLICADES

Les consultes per defecte mostren totes les files, incloent-hi files duplicades.

Exemple:

```
SELECT department_id
FROM employees;
```

```
department_id
-----
10
10
10
10
10
10
10
3
6
9
9
6
9
6
6
6
3
(16 rows)
```

Les files duplicades es poden suprimir de la sortida utilitzant la clàusula **DISTINCT**.

[!IMPORTANT]
Dues files estan duplicades quan el valor de cada una de les columnes és el mateix.

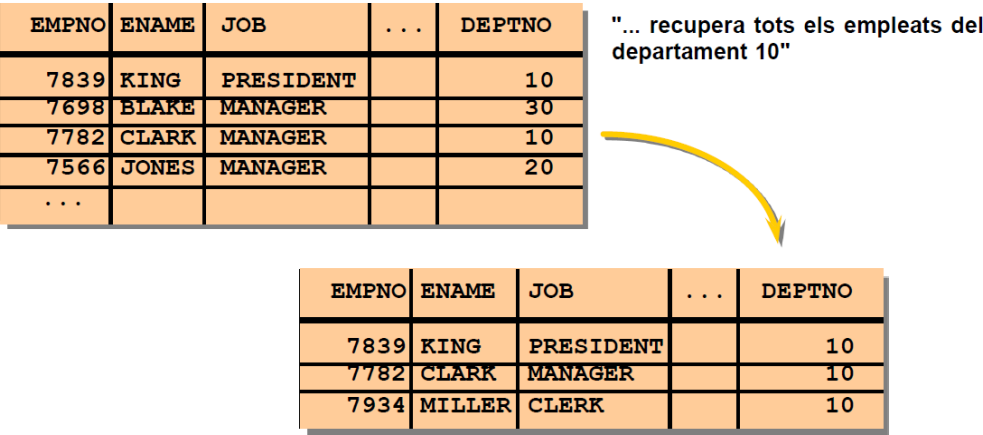
Exemple:

```
SELECT DISTINCT department_id
FROM employees;
```

```
department_id
-----
9
10
6
3
(4 rows)
```

RESTRICCIÓ I ORDENACIÓ DE DADES

Les consultes que hem vist fins ara recuperaven totes les files d'una taula, això no és pas el que ens interessa habitualment, sinó que ens interessa més limitar el nombre de files retornades.



Per limitar les files seleccionades mitjançant una o diverses condicions utilitzarem la clàusula **WHERE**.

```
SELECT *|{[DISTINCT] column|expression [alias],...}
FROM table;
[WHERE condition(s)];
```

La clàusula WHERE s'escriu després de la clàusula FROM.

Exemple: Seleccionem el nom, numero de departament i salari de tots els empleats que cobren més de 6000.

```
SELECT first_name, department_id, salary
FROM employees
WHERE salary > 6000;
```

first_name	department_id	salary
Nancy	10	12000.00
Daniel	10	9000.00
John	10	8200.00
Ismael	10	7700.00
Jose Manuel	10	7800.00
Luis	10	6900.00
Alexander	6	9210.00
Steven	9	15000.00
Lex	9	18000.00
Neena	9	23000.00
Bruce	6	12210.00
Den	3	11000.00

(12 rows)

- Les cadenes de caràcters o les dates es tanquen entre cometes simples.
- Les expressions de tipus caràcter són sensibles a majúscules/minúscules i els valors de tipus data i número són sensibles al format.
- El format data per defecte és 'yyyy-mm-dd' (on yyyy són els 4 dígits de l'any, mm són els 2 dígits del mes i dd són els 2 dígits del dia del mes).
- Els nombres s'escriuen sense "." i en el seu format el separador de decimals és el ','.

El següent exemple no retorna cap fila (tot i que hi ha un treballador amb cognom Chen):

```
SELECT *
FROM employees
WHERE last_name = 'chen';
```

Això és perquè en la base de dades la informació del camp last_name de la taula employees està desada amb el primer caràcter en majúscules.

Els operadors de comparació utilitzats en el llenguatge SQL són:

Operador	Descripció
=	Igual
>	Major que
<	Menor que
>=	Major o igual
<=	Menor o igual
<> or !=	No és igual

Exemple: Seleccionem els empleats, salaris i comissions dels empleats on el salari sigui inferior o igual a 10000

```
SELECT first_name, salary, commision
FROM employees
WHERE salary <= 10000;
```

first_name	salary	commision
Daniel	9000.00	
John	8200.00	
Ismael	7700.00	2.50
Jose Manuel	7800.00	
Luis	6900.00	0.10
Alexander	3100.00	
Alexander	9210.00	
David	5010.00	
Valli	5010.00	
Diana	4410.00	1.20
(10 rows)		

ALTRES OPERADORS DE COMPARACIÓ

Operador	Descripció
IN	Retorna cert si un valor coincideix amb algun valor d'una llista
BETWEEN ... AND ...	Retorna cert si un valor està entre un rang de valors
LIKE	Retorna cert si un valor coincideix amb un patró
IS NULL	Retorna cert si un valor és NUL

OPERADOR IN

Utilitzem la condició **IN** de pertinença per comprovar si hi ha valors en una llista . Si els valors de la llista son alfanumèrics hauran d'anar entre cometes.

Exemple: Seleccionem tots els empleats que tenen com a caps (manager) algun dels següents identificadors: 100, 102, 103

```
SELECT employee_id, first_name, salary, manager_id
FROM employees
WHERE manager_id IN (100, 102, 103);
```

employee_id	first_name	salary	manager_id
103	Alexander	9210.00	102
102	Lex	18000.00	100
105	David	5010.00	103
101	Neena	23000.00	100
106	Valli	5010.00	103
104	Bruce	12210.00	103
107	Diana	4410.00	103
114	Den	11000.00	100

(8 rows)

OPERADOR BETWEEN

Utilitzem la condició **BETWEEN** per visualitzar registres basats en un rang de valors. Els límits inferior i superior estan inclosos.

Exemple: Seleccionem tots els empleats que tenen un salari entre 10000 i 15000.

```
SELECT employee_id, first_name, salary
FROM employees
WHERE salary BETWEEN 10000 AND 15000;
```

employee_id	first_name	salary
108	Nancy	12000.00
100	Steven	15000.00
104	Bruce	12210.00
114	Den	11000.00

(4 rows)

OPERADOR LIKE

Utilitzarem l'operador **LIKE** per executar recerques en cadenes de caràcters que incloguin comodins. Les condicions de recerca poden contenir caràcters o nombres.

- "%" denota zero a diversos caràcters a partir de la posició indicada.
- "_" denota 1 sol caràcter en la posició indicada.

Exemple: Seleccionem tots els treballadors que el seu nom comença per la lletra S.

```
SELECT employee_id, first_name, salary
FROM employees
WHERE first_name LIKE 'S%';
```

```

employee_id | first_name | salary
-----+-----+-----
          100 | Steven    | 15000.00
(1 row)

```

Es poden combinar patrons de caràcters. Es pot fer servir l'identificador ESCAPE "" per cercar "%" i "_"

Exemple: Cercar tots els treballadors el nom dels quals comença per qualsevol lletra i a continuació conté la lletra o.

```

SELECT employee_id, first_name, salary
FROM employees
WHERE first_name LIKE '_o%';

```

```

employee_id | first_name | salary
-----+-----+-----
          110 | John      | 8200.00
          112 | Jose Manuel | 7800.00
(2 rows)

```

OPERADOR IS NULL

Quan necessitem verificar la presència del valor nul utilitzem l'operador **IS NULL**

Exemple: Seleccionem les dades dels treballadors que no tenen un cap (manager) assignat.

```

SELECT employee_id, first_name, manager_id
FROM employees
WHERE manager_id IS NULL;

```

```

employee_id | first_name | manager_id
-----+-----+-----
          100 | Steven    |
(1 row)

```

[!IMPORTANT]

Quan es comença a aprendre SQL un dels principals errors és utilitzar el NULL amb l'operador '=' com per exemple "WHERE mgr = NULL". Això es un error important, per què NULL no és un valor i per tant no es pot igualar a res.

OPERADORS LÒGICS

Quan necessitem establir més d'una condició en la clàusula WHERE haurem d'enllaçar-les mitjançant operadors lògics.

Operador	Descripció
AND	Operador lògic AND
OR	Operador lògic OR
NOT	Negar el resultat d'altres operadors

- **AND** requereix que totes dues condicions siguin certes (TRUE).

Exemple: Seleccionem tots aquells treballadors que tenen un salari superior a 7000 i la seva feina és "Accountant" (job_id = 6)

```
SELECT employee_id, first_name, job_id, salary
FROM employees
WHERE salary > 7000
AND job_id = 6;
```

employee_id	first_name	job_id	salary
109	Daniel	6	9000.00
110	John	6	8200.00
111	Ismael	6	7700.00
112	Jose Manuel	6	7800.00

(4 rows)

- **OR** requereix que alguna de les condicions sigui certa (TRUE).

Exemple: Seleccionem tots aquells treballadors que tenen un salari superior a 7000, o bé, que la seva feina sigui "Accountant" (job_id = 6)

```
SELECT employee_id, first_name, job_id, salary
FROM employees
WHERE salary > 7000
OR job_id = 6;
```

employee_id	first_name	job_id	salary
108	Nancy	7	12000.00
109	Daniel	6	9000.00
110	John	6	8200.00
111	Ismael	6	7700.00
112	Jose Manuel	6	7800.00
113	Luis	6	6900.00
103	Alexander	9	9210.00
100	Steven	4	15000.00
102	Lex	5	18000.00
101	Neena	5	23000.00
104	Bruce	9	12210.00
114	Den	14	11000.00

(12 rows)

- **NOT** nega una condició.

Exemple: Seleccionem tots aquells treballadors que no tenen com a feina ni "President", ni "Accounting manager", ni "Accountant".

```
SELECT employee_id, first_name, job_id, salary
FROM employees
WHERE NOT job_id IN (2, 4, 6);
```

employee_id	first_name	job_id	salary
108	Nancy	7	12000.00
115	Alexander	13	3100.00
103	Alexander	9	9210.00
102	Lex	5	18000.00
105	David	9	5010.00
101	Neena	5	23000.00
106	Valli	9	5010.00
104	Bruce	9	12210.00
107	Diana	9	4410.00
114	Den	14	11000.00

(10 rows)

REGLES DE PRIORITAT DELS OPERADORS

Alhora d'escriure sentències SQL és important saber com actua l'analitzador SQL alhora d'avaluar els diferents operadors ja que segueix un determinat ordre de prioritats. Recordem que sempre podem establir l'ordre utilitzant parèntesis.

Ordre Avaluat	Operador
1	Aritmètic
2	Concatenació
3	Comparació
4	[NOT] <atribut>IS NULL, LIKE, [NOT] atribut IN
5	[NOT] BETWEEN
6	NOT
7	AND
8	OR

Exemple 1:

```
SELECT employee_id, first_name, job_id
FROM employees
WHERE job_id = 9
OR job_id = 6
AND salary > 8000;
```

employee_id	first_name	job_id	salary
109	Daniel	6	9000.00
110	John	6	8200.00
103	Alexander	9	9210.00
105	David	9	5010.00
106	Valli	9	5010.00
104	Bruce	9	12210.00
107	Diana	9	4410.00

(7 rows)

Per l'ordre que hem vist abans primer s'avaluarà AND i seguidament OR. Per tant obtindrem tots els treballadors que treballen com a "Accountant" i a més guanyen més de 8000, i també, tots els treballadors que treballen com a "Programmer".

Exemple 2:

```
SELECT employee_id, first_name, job_id
FROM employees
WHERE (job_id = 9
OR job_id = 6)
AND salary > 8000;
```

employee_id	first_name	job_id	salary
109	Daniel	6	9000.00
110	John	6	8200.00
103	Alexander	9	9210.00
104	Bruce	9	12210.00

(4 rows)

En aquest cas hem forçat que s'avalui primer OR ja que està entre parèntesi i a continuació AND. Per tant obtindrem tots els treballadors que treballen com a "Accountant", o bé, com a "Programmer", i que guanyen més de 8000.

ORDENAR EL RESULTAT. CLÀUSULA ORDER BY

Per definició en el model relacional, els registres no es guarden ordenats. Si volem obtenir resultats ordenats els hem d'ordenar explícitament.

```
SELECT      *|{[DISTINCT] column|expression [alias],...}
FROM        table
[WHERE      condition(s)]
[ORDER BY  {column, expr, alias} [ASC|DESC]]
```

La clàusula ORDER BY sempre va al final de la sentència SELECT.

- **ASC** indica que l'ordre ha de ser ascendent. (És el valor per defecte).
 - Nombres. Ordenarà de menor a major.

- Dates. Ordenarà de més antic a més nou.
- Caràcters. Ordenarà en ordre ascendent segons l'ordre alfabètic A - Z
- **DESC** indica que l'ordre ha de ser descendent.
 - Nombres. Ordenarà de major a menor.
 - Dates. Ordenarà de més nou a més antic.
 - Caràcters. Ordenarà en ordre descendent segons l'ordre alfabètic Z - A

En la clàusula ORDER BY podem indicar el nom del camp, una expressió, l'aliàs de columna, i fins i tot, un nombre que representa la posició de la columna en la clàusula SELECT.

Exemple:

```
SELECT employee_id, first_name, 12*salary AS "Anual salari", department_id
FROM employees
ORDER BY department_id DESC, "Anual salari";
```

employee_id	first_name	Annual salary	department_id
113	Luis	82800.00	10
111	Ismael	92400.00	10
112	Jose Manuel	93600.00	10
110	John	98400.00	10
109	Daniel	108000.00	10
108	Nancy	144000.00	10
100	Steven	180000.00	9
102	Lex	216000.00	9
101	Neena	276000.00	9
107	Diana	52920.00	6
106	Valli	60120.00	6
105	David	60120.00	6
103	Alexander	110520.00	6
104	Bruce	146520.00	6
115	Alexander	37200.00	3
114	Den	132000.00	3

(16 rows)

CLAUSULES FETCH I LIMIT

D'avegades ens pot interessar limitar el nombre de files retornades per una consulta.

```
SELECT      *|{[DISTINCT] column|expression [alias],...}
FROM        table
[WHERE      condition(s)]
[ORDER BY  {column, expr, alias} [ASC|DESC]]
[LIMIT {row_count} | FETCH {FIRST|NEXT}
           [row_count] {ROW|ROWS} ONLY ]
```

- **LIMIT** no és estàndard, **FETCH** compleix amb l'estàndard SQL 2008.
- La clàusula LIMIT/FETCH va després d'ORDER BY i s'utilitza per obtenir les primeres/últimes N files retornades per una consulta.

Exemple: Seleccionem les dades de tots els empleats ordenades per data de contractació, però només volem les dues primeres files, que serien els dos treballadors més "antics".

```
SELECT employee_id, first_name, hire_date
FROM employees
ORDER BY hire_date
LIMIT 2;
```

employee_id	first_name	hire_date
100	Steven	1987-06-17
101	Neena	1989-09-21

(2 rows)

La clàusula FETCH té una sintaxi més complexa:

- FETCH { FIRST|NEXT } [row_count] { ROW|ROWS } ONLY

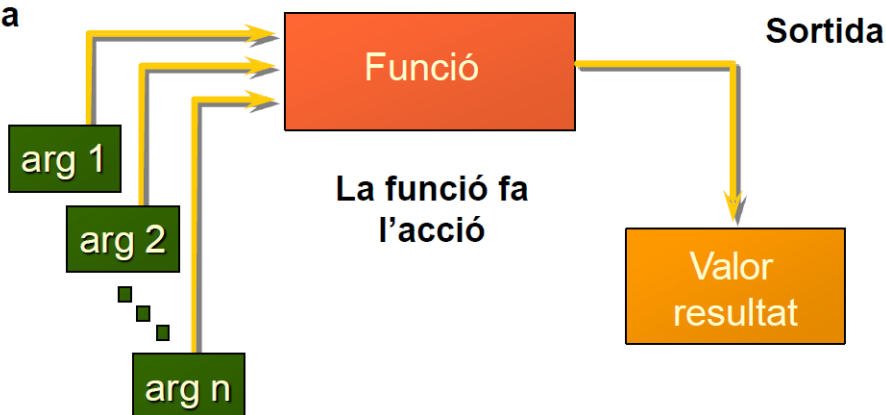
Exemple: Seleccionem les mateixes dades de l'exemple anterior utilitzant FETCH

```
SELECT employee_id, first_name, hire_date
FROM employees
ORDER BY hire_date
FETCH FIRST 2 ROW ONLY;
```

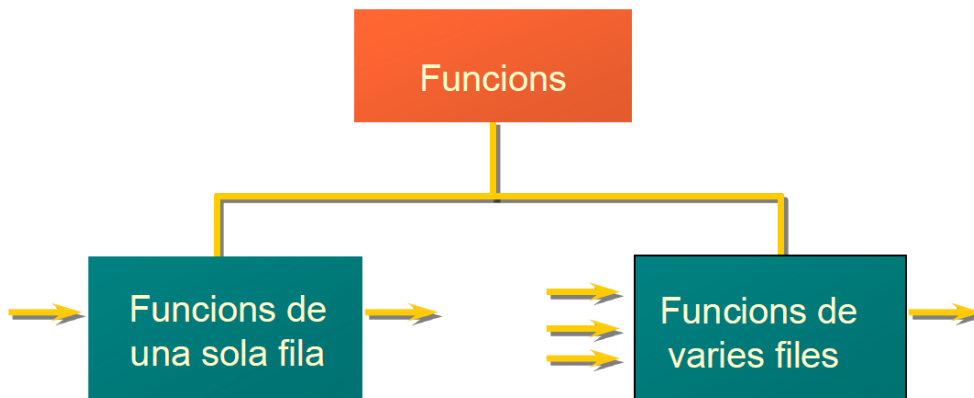
FUNCIONS DE FILA ÚNICA

Una funció és una porció de codi independent que realitza una tasca específica, que pot ser executada en qualsevol part del codi principal (mitjançant una crida) i que pot rebre valors d'entrada (arguments) i normalment retornarà un valor de sortida.

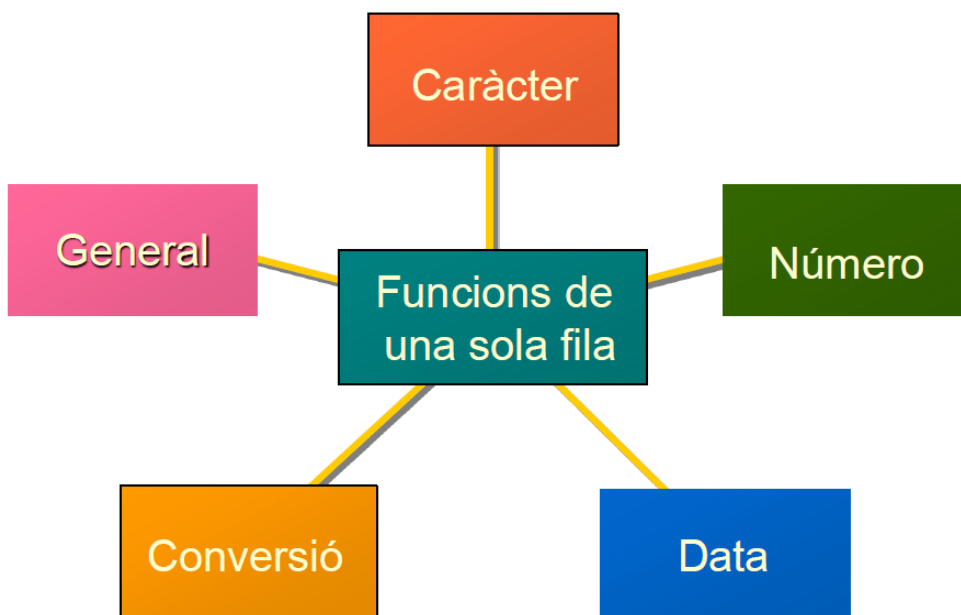
Entrada



En el llenguatge SQL tenim dos tipus de funcions, les anomenades funcions d'una sola fila (reben com entrada un valor d'una fila i retornen un valor en una única fila), i les anomenades funcions de diverses files (reben com entrada els valors de diverses files i retornen un valor en una única fila).



CARACTERÍSTIQUES DE LES FUNCIONS D'UNA SOLA FILA



- Manipulen ítems de dades.
- Accepten arguments i retornen un valor.
- Actuen sobre cada fila retornada.
- Retornen un resultat per fila.
- Poden modificar el tipus de dada.
- Es poden aniuar.
- Accepten arguments columna o expressió.
- Es poden utilitzar en totes les clàusules, excepte en el FROM.

```
function_name [(arg1, arg2,...)]
```

[!important]

Les funcions són pròpies del SGBD, no són estàndard, per tant poden ser diferents en funció del SGBD utilitzat. En aquest tema ens referim a les funcions PostgreSQL.

FUNCIONS DE CARÀCTERS

- Accepten com entrada una cadena de caràcters.

- Retornen com a sortida una cadena de caràcters o un nombre.

Postgresql String Functions

MANIPULACIÓ DE MAJÚSCULES/MINÚSCULES

Funció	Resultat
LOWER('Curs SQL')	curs sql
UPPER('Curs SQL')	CURS SQL
INITCAP('Curs SQL')	Curs Sql

Exemples:

```
SELECT first_name, last_name, salary
FROM employees
WHERE last_name = 'chen';
```

No retorna cap fila.

```
SELECT first_name, last_name, salary
FROM employees
WHERE LOWER(last_name) = 'chen';
```

```
first_name | last_name | salary
-----+-----+-----
John       | Chen      | 8200.00
(1 row)
```

Mateix exemple, però ara a més visualitzem el nom i cognom de l'empleat amb majúscules.

```
SELECT UPPER(first_name) Nom, UPPER(last_name) Cognom, salary
FROM employees
WHERE LOWER(last_name) = 'chen';
```

```
nom  | cognom | salary
-----+-----+-----
JOHN | CHEN   | 8200.00
(1 row)
```

MANIUPULACIÓ DE CARÀCTERS

Aquestes són les funcions més habituals per a la manipulació de cadenes de caràcters.

Funció	Resultat
CONCAT('Hello', 'World')	HelloWorld
SUBSTR('HelloWorld',1,5)	Hello
LENGTH('HelloWorld')	10
POSITION('W' in 'HelloWorld')	6
LPAD(ename,10,'*')	*****SCOTT
RPAD(ename,10,'*')	SCOTT*****
TRIM(LEADING 'H' FROM 'Hello')	Ello

Exemple: Seleccionem el nom, el nom i cognom concatenats, la mida del nom i la posició que ocupa la primera aparició de la lletra A en el nom de tots els empleats que el seu nom comença per Alex

```
SELECT first_name, CONCAT(first_name, job), LENGTH(first_name),
POSITION('A' in ename)
FROM employees
WHERE SUBSTR(first_name,1,4) = 'Alex';
```

```
first_name |      concat      | length | position
-----+-----+-----+-----
Alexander | AlexanderKhoo   |      9 |      1
Alexander | AlexanderHunold |      9 |      1
(2 rows)
```

FUNCIONS NUMÈRIQUES

- Accepten com entrada un valor numèric.
- Retornen com a sortida un valor numèric.

Postgresql Math Functions

FUNCIÓ ROUND

Obtenim un valor numèric arrodonit a n decimals. Si no indiquem l'argument numero de decimals arrodoneix a 0 decimals.

Exemple:

```
SELECT ROUND(45.923,2), ROUND(45.923,0), ROUND(45.923,-1);
```

```
round | round | round
-----+-----+-----
45.92 | 46    | 50
(1 row)
```

FUNCIO TRUNC

Obtenim un valor numèric truncat a partir del decimal n, si s'omet el nombre de decimals trunca a 0 decimals.

Exemple:

```
SELECT TRUNC(45.923,2), TRUNC(45.923,0), TRUNC(45.923,-1);
```

```
trunc | trunc | trunc
-----+-----+-----
45.92 | 45    | 40
(1 row)
```

FUNCIO MOD

Obtenim el residu de dividir el nombre argument 1 per el nombre argument 2.

Exemple:

```
SELECT employee_id, first_name, salary, MOD(salary, 1000)
FROM employees;
```

```
employee_id | first_name | salary | mod
-----+-----+-----+-----
108 | Nancy | 12000.00 | 0.00
109 | Daniel | 9000.00 | 0.00
110 | John | 8200.00 | 200.00
111 | Ismael | 7700.00 | 700.00
112 | Jose Manuel | 7800.00 | 800.00
113 | Luis | 6900.00 | 900.00
115 | Alexander | 3100.00 | 100.00
103 | Alexander | 9210.00 | 210.00
100 | Steven | 15000.00 | 0.00
102 | Lex | 18000.00 | 0.00
105 | David | 5010.00 | 10.00
101 | Neena | 23000.00 | 0.00
106 | Valli | 5010.00 | 10.00
104 | Bruce | 12210.00 | 210.00
107 | Diana | 4410.00 | 410.00
114 | Den | 11000.00 | 0.00
(16 rows)
```

TREBALL AMB DATES

PostgreSql utilitza diferents tipus per emmagatzemar dates.

- **DATE:** Emmagatzema només dates (sense temps) usant 4 bytes (des de 4713 A.C. fins a 5874897 D.C.). Per defecte en el format yyyy-mm-dd.
- **TIMESTAMP:** Emmagatzema dates i temps usant 8 bytes. Per defecte en el format yyyy-mm-dd hh:nn:ss.ds-e7

El format de data per defecte depèn de l'aplicació client que utilitzem.

Es pot modificar amb la comanda:

```
SET TIMEZONE = '<descripció zona>'
```

Podem veure la nostra zona amb la comanda:

```
SHOW TIMEZONE
```

FUNCIÓ NOW()

La funció **NOW()** retorna la data i hora del servidor on tenim instal·lat el SGBD. Amb la funció **NOW()::DATE** obtindrem només la data, sense la hora. Per saber la data i hora també podem fer servir els paràmetres interns **CURRENT_DATE** i **CURRENT_TIME**.

Exemple:

```
SELECT NOW(), NOW()::DATE, CURRENT_DATE, CURRENT_TIME;
```

now	now	current_date	current_time
2024-07-18 15:53:44.425189+00	2024-07-18	2024-07-18	15:53:44.425189+00

(1 row)

Podem sumar o restar dues dates per trobar el nombre de dies entre elles, per exemple `Now () - '1 year'` resta un any a la data actual. També podem sumar un interval de dies, mesos, anys a una data.

Exemple: Amb aquesta consulta podem saber quants dies/setmanes porten els treballadors a l'empresa

```
SELECT first_name, (now()::date - hire_date) / 7 weeks  
FROM employees;
```

```

first_name | weeks
-----+-----
Nancy      | 1561
Daniel     | 1561
John       | 1398
Ismael     | 1398
Jose Manuel | 1375
Luis       | 1284
Alexander  | 1522
Alexander  | 1802
Steven     | 1935
Lex        | 1644
David      | 1412
Neena      | 1817
Valli      | 1380
Bruce      | 1730
Diana      | 1327
Den        | 1545
(16 rows)

```



FUNCIONS AMB DATES

- Accepten com entrada un valor data.
- Retornen com a sortida un valor data, un interval o una nombre.

Postgresql Date Functions

Funció	Descripció
AGE	Retorna el número d'anys, mesos i dies entre dos dates
DATE_PART o EXTRACT	Extreu part de una data o temps(any, mes, dia, hora)
CURRENT_DATE	Retorna la data actual (sense hora)
CURRENT_TIME	Retorna la hora actual
DATE_TRUNC	Trunca una data

Alguns exemples:

- **AGE('2021-09-01','2021-01-11')**
 '1 year 7 mons 21 days'
- **EXTRACT (YEAR FROM timestamp '2021-09-01')**
 '2021'
- **DATE_TRUNC ('month', timestamp '2021-09-20')**
 '2021-09-01'

CONVERSIÓ DE DADES

En algunes ocasions serà necessari fer conversions de dades ja sigui perquè una funció necessita un argument alfanumèric i li hem de passar un nombre, o perquè volem convertir un valor alfanumèric que sabem que és un nombre per poder realitzar alguna operació aritmètica, etc.

Els SGBD permeten realitzar dos tipus de conversions de dades: implícites i explícites.

- Conversió implícita:
És aquella que duu a terme automàticament el SGBD per a realitzar una assignació o avaluació d'una expressió.
- Per assignació:

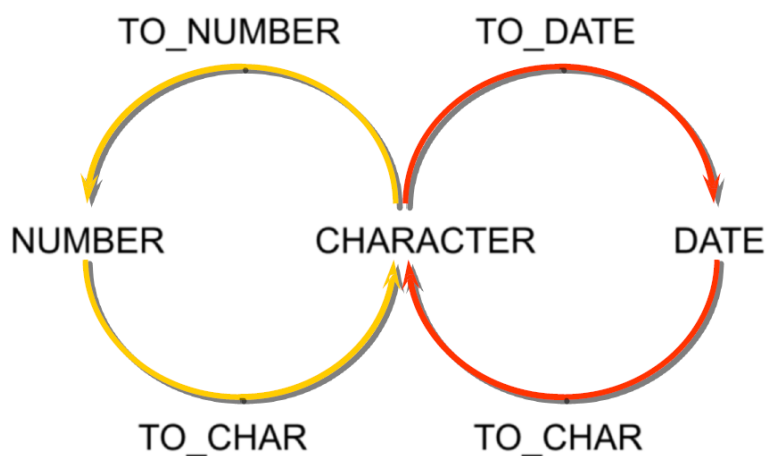
De	A
VARCHAR2 o CHAR	NUMBER
VARCHAR2 o CHAR	DATE
NUMBER	VARCHAR2
DATE	VARCHAR2

- Per avaluació:

De	A
VARCHAR2 o CHAR	NUMBER
VARCHAR2 o CHAR	DATE

- Conversió explícita:

És aquella que forcem explícitament mitjançant funcions del SGBD. Bàsicament utilitzarem les funcions: TO_NUMBER, TO_CHAR i TO_DATE.



FUNCIÓ TO_CHAR

Aquesta és una funció molt 'versàtil' que ens permet convertir valors numèrics i dates en cadenes de caràcters.

- Amb dates:

```
TO_CHAR(date, 'format_model')
```

Format:

- S'ha de tancar entre cometes simples i és sensible a majúscules/minúscules.
- Inclou un element de format de data vàlid.
- Té un element fm (fill mode) per eliminar espais en blanc de farciment o suprimir zeros a l'esquerra.
- Està separat de la data per una coma.

YYYY	Any complet en número
YY	Any en dos dígit
MM	Nº del mes amb dos dígit
MONTH	Nom complet del mes
DY	Abreviatura de tres lletres del dia de la setmana
DAY	Nom complet del dia

Postgresql TO_CHAR function

Exemple:

```
SELECT first_name, TO_CHAR(hire_date, 'DD MONTH YYYY') "Data contracte"  
FROM employees;
```

```

first_name | Data contracte
-----+-----
Nancy      | 17 AUGUST 1994
Daniel     | 16 AUGUST 1994
John       | 28 SEPTEMBER 1997
Ismael     | 30 SEPTEMBER 1997
Jose Manuel | 07 MARCH 1998
Luis       | 07 DECEMBER 1999
Alexander  | 18 MAY 1995
Alexander  | 03 JANUARY 1990
Steven     | 17 JUNE 1987
Lex        | 13 JANUARY 1993
David      | 25 JUNE 1997
Neena      | 21 SEPTEMBER 1989
Valli      | 05 FEBRUARY 1998
Bruce      | 21 MAY 1991
Diana      | 07 FEBRUARY 1999
Den        | 07 DECEMBER 1994
(16 rows)

```

- Amb nombres:

```
TO_CHAR(number, 'format_model')
```

Utilitzem aquests formats per mostrar un nombre en format alfanumèric:

9	Representa un nombre
0	Força a que es mostri el 0 (zero)
L	Utilitza el símbol de moneda local
.	Escriu el punt decimal
,	Escriu l'indicador de milers

Exemple:

```

SELECT TO_CHAR(salary, '99,999') SALARY
FROM employees
WHERE last_name = 'Chen';

```

```

salary
-----
8,200
(1 row)

```

FUNCIONS TO_NUMBER I TO_DATE

Convertirem una cadena de caràcters a un format numèric utilitzant la funció TO_NUMBER.

```
TO_NUMBER(char[, 'format_model'])
```

Convertirem una cadena de caràcters a un format data utilitzant la funció TO_DATE.

```
TO_DATE(char[, 'format_model'])
```

En aquestes funcions "Format model" indica el format que té la cadena de caràcters i que s'ha de donar perquè la conversió sigui possible.

FUNCIONS GENERALS

Funcions que treballen amb qualsevol tipus de dada i estan relacionades amb l'ús de valors nuls.

FUNCIÓ COALESCE

Si la primera expressió no és nul·la, retorna aquesta expressió; en cas contrari, fa el mateix amb les expressions restants.

Exemple:

```
SELECT FIRST_NAME, COMMISSION, SALARY, COALESCE(COMMISSION, SALARY, 10) COMM
FROM EMPLOYEES;
```

first_name	commision	salary	comm
Nancy		12000.00	12000.00
Daniel		9000.00	9000.00
John		8200.00	8200.00
Ismael	2.50	7700.00	2.50
Jose Manuel		7800.00	7800.00
Luis	0.10	6900.00	0.10
Alexander		3100.00	3100.00
Alexander		9210.00	9210.00
Steven		15000.00	15000.00
Lex	2.01	18000.00	2.01
David		5010.00	5010.00
Neena	1.01	23000.00	1.01
Valli		5010.00	5010.00
Bruce		12210.00	12210.00
Diana	1.20	4410.00	1.20
Den	0.21	11000.00	0.21
(16 rows)			

EXPRESSIONS CONDICIONALS. L'EXPRESSIÓ CASE

Facilita les consultes condicionals realitzant el treball d'una sentència IF-THEN-ELSE.

```

CASE expr
  WHEN comparison_expr1 THEN result_expr1
  WHEN comparison_expr2 THEN result_expr2
  WHEN comparison_exprn THEN result_exprn
  ELSE else_expr
END

```

Exemple: Sintaxi 1. Avaluem el valor d'una expressió o columna.

```

SELECT first_name, salary,
CASE salary
  WHEN 5000 THEN 'Ets mileurista'
  WHEN 15000 THEN 'Guanyes 15000'
  ELSE 'Misteri'
END
FROM employees;

```

first_name	salary	case
Nancy	12000.00	Misteri
Daniel	9000.00	Misteri
John	8200.00	Misteri
Ismael	7700.00	Misteri
Jose Manuel	7800.00	Misteri
Luis	6900.00	Misteri
Alexander	3100.00	Misteri
Alexander	9210.00	Misteri
Steven	15000.00	Guanyes 15000
Lex	18000.00	Misteri
David	5010.00	Misteri
Neena	23000.00	Misteri
Valli	5010.00	Misteri
Bruce	12210.00	Misteri
Diana	4410.00	Misteri
Den	11000.00	Misteri
(16 rows)		

Exemple: Sintaxi 2. Avaluem expressions condicionals.

```

SELECT first_name, salary,
CASE
  WHEN salary = 5000 THEN 'Ets mileurista'
  WHEN salary > 5000 THEN 'Guanyes més de 5000'
  ELSE 'Guanyes menys de 5000, ets becari'
END
FROM employees;

```

first_name	salary	case
Nancy	12000.00	Guanyes més de 5000
Daniel	9000.00	Guanyes més de 5000
John	8200.00	Guanyes més de 5000
Ismael	7700.00	Guanyes més de 5000
Jose Manuel	7800.00	Guanyes més de 5000
Luis	6900.00	Guanyes més de 5000
Alexander	3100.00	Guanyes menys de 5000, ets becari
Alexander	9210.00	Guanyes més de 5000
Steven	15000.00	Guanyes més de 5000
Lex	18000.00	Guanyes més de 5000
David	5010.00	Guanyes més de 5000
Neena	23000.00	Guanyes més de 5000
Valli	5010.00	Guanyes més de 5000
Bruce	12210.00	Guanyes més de 5000
Diana	4410.00	Guanyes menys de 5000, ets becari
Den	11000.00	Guanyes més de 5000

(16 rows)

CONSULTES AVANÇADES - CONSULTES SOBRE VÀRIES TAULES

Fins ara totes les consultes que hem vist obtenien la informació d'una sola taula. En el model relacional les taules (relacions) es relacionen entre elles mitjançant les claus foranes, que referencien una clau primària d'una altra taula amb les que tenen relació. Amb aquest model de dades el més habitual és tenir que obtenir dades de diferents taules que estan relacionades entre elles, per exemple, les dades de la capçalera d'una factura, que incloguin dades concretes del client, formes de pagament, etc. dades que estarien en altres taules.

Tornant a la base de dades HR que estem utilitzant en els exemples, si ens fixem amb les taules employees i departments, aquestes estan relacionades mitjançant la clau forana department_id de la taula employees que referencia a la clau primària department_id de la taula departments. Aquesta relació expressa el departament que té assignat cada treballador.

EMPLOYEES

EMPNO	ENAME	...	DEPTNO
7839	KING	...	10
7698	BLAKE	...	30
...			
7934	MILLER	...	10

DEPARTMENTS

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

EMPNO	DEPTNO	LOC
7839	10	NEW YORK
7698	30	CHICAGO
7782	10	NEW YORK
7566	20	DALLAS
7654	30	CHICAGO
...		

14 rows selected.

UNIÓ DE TAULES

La clàusula JOIN .. ON ens permet realitzar la unió de dues o més taules per a obtenir informació. En el llenguatge SQL tenim diverses variants d'unió 'JOIN' que veurem a continuació.

[!Important] El llenguatge SQL ens proporciona diverses formes d'unir dues o més taules per a obtenir informació. Cada SGBD té la seva pròpia forma però a partir de la revisió SQL-92 s'estandaritza mitjançant la clàusula **JOIN .. ON**.

```
SELECT table1.column, table2.column
FROM table1
[CROSS JOIN table2] |
[NATURAL JOIN table2] |
[JOIN table 2 USING (column_name) |
[JOIN table2
    ON (table1.column_name = table2.column_name)] |
[LEFT | RIGHT | FULL OUTER JOIN table2
    ON (table1.column_name = table2.column_name)];
```

Per especificar condicions arbitràries o especificar columnes per unir, s'utilitza la clàusula ON. La condició d'unió es separa d'altres condicions de cerca. La clàusula ON facilita la comprensió del codi.

[!Important] Per facilitar la comprensió del codi en la clàusula ON només posarem les condicions d'unió entre taules, no hi posarem cap altre condició, per a la resta farem servir la clàusula WHERE.

PRODUCTE CARTESIÀ

El producte cartesià és una operació no desitjable quan realitzem consultes sobre vàries taules. El producte cartesià uneix totes les files de la primera taula amb totes les files de la segona. El producte cartesià es forma quan:

- Una condició de join s'omet.
- Una condició de join no és vàlida.

Per evitar un producte cartesià, s'ha d'incloure una condició de join vàlida en la clàusula ON.

Exemple: La següent sentència produeix un producte cartesià.

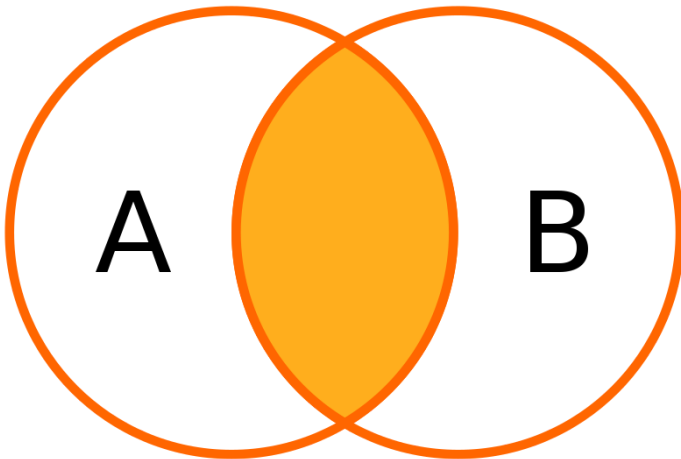
```
SELECT employee_id, first_name, department_name
FROM employees, departments;
```

La taula employees té 16 registres, la taula departments té 15 registres, la consulta generarà 16x15 = 240 files.

UNIÓ D'IGUALTAT

Aquest és el tipus d'unió més habitual, consisteix en enllaçar les taules mitjançant la igualtat dels valors de les columnes clau forana de la primera taula i clau primària de la segona.

Utilitzant la teoria de conjunts sobre els conjunts A (valors clau forana taula A) i B (valors clau primària taula B), podriem representar una unió d'igualtat mitjançant la **intersecció**, és a dir, aquells valors que són coincidents en ambdós conjunts:



Podem recuperar registres amb les següents unions d'igualtat:

- NATURAL JOIN
- Clàusula USING
- INNER JOIN (o simplement JOIN)

NATURAL JOIN

Natural Join s'utilitza quan els camps que enllacen les dues taules tenen el mateix nom.

Exemple:

```
SELECT employee_id, first_name, salary, department_name
FROM employees
NATURAL JOIN departments;
```

employee_id	first_name	salary	department_name
108	Nancy	12000.00	Finance
109	Daniel	9000.00	Finance
110	John	8200.00	Finance
111	Ismael	7700.00	Finance
112	Jose Manuel	7800.00	Finance
113	Luis	6900.00	Finance
115	Alexander	3100.00	Purchasing
103	Alexander	9210.00	IT
100	Steven	15000.00	Executive
102	Lex	18000.00	Executive
105	David	5010.00	IT
101	Neena	23000.00	Executive
106	Valli	5010.00	IT
104	Bruce	12210.00	IT
107	Diana	4410.00	IT
114	Den	11000.00	Purchasing

(16 rows)

CLÀUSULA USING

Using també s'utilitza quan el nom dels camps que enllacen les dues taules tenen el mateix nom, però en aquest cas hem d'especificar quin és el camp que actua de clau forana i clau primària a cada taula.

Exemple:

```
SELECT employee_id, first_name, salary, department_id, department_name
FROM employees
INNER JOIN departments USING(department_id);
```

employee_id	first_name	salary	department_id	department_name
108	Nancy	12000.00	10	Finance
109	Daniel	9000.00	10	Finance
110	John	8200.00	10	Finance
111	Ismael	7700.00	10	Finance
112	Jose Manuel	7800.00	10	Finance
113	Luis	6900.00	10	Finance
115	Alexander	3100.00	3	Purchasing
103	Alexander	9210.00	6	IT
100	Steven	15000.00	9	Executive
102	Lex	18000.00	9	Executive
105	David	5010.00	6	IT
101	Neena	23000.00	9	Executive
106	Valli	5010.00	6	IT
104	Bruce	12210.00	6	IT
107	Diana	4410.00	6	IT
114	Den	11000.00	3	Purchasing

(16 rows)

La diferència amb INNER JOIN que veurem a continuació és que només es mostra una de les dues columnes department_id i no tenim el problema de clau ambigua que veurem a continuació.

INNER JOIN

La clàusula INNER JOIN estableix quines taules unim i mitjançant la clàusula ON establim la condició d'igualtat amb els camps que enllacen ambdues taules, que no tenen perquè dir-se igual. La clàusula INNER és opcional, no cal posar-la.

Exemple:

```
SELECT employee_id, first_name, salary, department_name
FROM employees e
INNER JOIN departments d ON e.department_id = d.department_id;
```


employee_id	first_name	salary	department_name
108	Nancy	12000.00	Finance
109	Daniel	9000.00	Finance
110	John	8200.00	Finance
111	Ismael	7700.00	Finance
112	Jose Manuel	7800.00	Finance
113	Luis	6900.00	Finance
115	Alexander	3100.00	Purchasing
103	Alexander	9210.00	IT
100	Steven	15000.00	Executive
102	Lex	18000.00	Executive
105	David	5010.00	IT
101	Neena	23000.00	Executive
106	Valli	5010.00	IT
104	Bruce	12210.00	IT
107	Diana	4410.00	IT
114	Den	11000.00	Purchasing

(16 rows)

- Condicions addicionals: Qualsevol condició addicional de la consulta s'escriurà en la clàusula WHERE.
- Qualificació de noms de camps ambigus: Haurem d'utilitzar prefixos de taula per a qualificar noms de columnes que estiguin en ambdues taules, en cas contrari el SGBD no sap a quin camp ens referim i ens donarà un error. Els prefixos de taula també ens ajuden a millorar el rendiment. Podem distingir columnes d'idèntic nom però que pertanyen a diferents taules utilitzant àlies de columna.

També podem utilitzar **àlies de taula** per a simplificar la sintaxi de la sentència.

Exemple: Utilitzant prefixos de taula

```
SELECT employees.employee_id, employees.first_name,
employees.salary, departments.department_id, departments.department_name
FROM employees
INNER JOIN departments ON employees.department_id =
departments.department_id;
```

Exemple: Utilitzant àlies de taula

```
SELECT emp.employee_id, emp.first_name, emp.salary, dep.department_id,
dep.department_name
FROM employees emp
INNER JOIN departments dep ON emp.department_id = dep.department_id;
```

UNIÓ DE DIVERSES TAULES

CUSTOMER

NAME	CUSTID
-----	-----
JOCKSPORTS	100
TKB SPORT SHOP	101
VOLLYRITE	102
JUST TENNIS	103
K+T SPORTS	105
SHAPE UP	106
WOMENS SPORTS	107
...	...
9 rows selected.	

ORD

CUSTID	ORDID
-----	-----
101	610
102	611
104	612
106	601
102	602
106	604
...	...
21 rows selected.	

ITEM

ORDID	ITEMID
-----	-----
610	3
611	1
612	1
601	1
602	1
...	...
64 rows selected.	

Per unir n taules, necessitem un mínim de n-1 condicions d'unió.
Ex: tres taules, necessita mínim de dues unions.

Exemple:

```
SELECT employee_id, city, department_name
FROM employees e JOIN departments d
ON (e.department_id = d.department_id)
JOIN locations l
ON d.location_id = l.location_id;
```

employee_id	city	department_name
-----	-----	-----
108	Seattle	Finance
109	Seattle	Finance
110	Seattle	Finance
111	Seattle	Finance
112	Seattle	Finance
113	Seattle	Finance
115	Seattle	Purchasing
103	Southlake	IT
100	Seattle	Executive
102	Seattle	Executive
105	Southlake	IT
101	Seattle	Executive
106	Southlake	IT
104	Southlake	IT
107	Southlake	IT
114	Seattle	Purchasing
(16 rows)		

UNIONS DE NO IGUALTAT

EMPLOYEES

EMPNO	ENAME	SAL
7839	KING	5000
7698	BLAKE	2850
7782	CLARK	2450
7566	JONES	2975
7654	MARTIN	1250
7499	ALLEN	1600
7844	TURNER	1500
7900	JAMES	950
...		
14 rows selected.		

JOBS

JOB	MINSAL	MAXSAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

“el salari en la taula
EMPLOYEES, està entre qualsevol parell
de rangs de
salari mínim i màxim.”

Exemple:

```
SELECT e.first_name, e.salary, j.job_title
FROM employees e
JOIN jobs j ON e.salary BETWEEN j.min_salary AND j.max_salary
WHERE j.job_id = 1;
```

first_name	salary	job_title
Daniel	9000.00	Public Accountant
John	8200.00	Public Accountant
Ismael	7700.00	Public Accountant
Jose Manuel	7800.00	Public Accountant
Luis	6900.00	Public Accountant
David	5010.00	Public Accountant
Valli	5010.00	Public Accountant
Diana	4410.00	Public Accountant

(8 rows)

UNIONS EXTERNES

Les claus foranes referencien una clau primària o són nul·les, en aquest cas els registres que no tenen un valor assignat a la seva clau primària no es recuperaran mai mitjançant una unió d'igualtat. En aquests casos utilitzarem unions externes anomenades també **OUTER JOIN**.

EMPLOYEES		DEPARTMENTS	
ENAME	DEPTNO	DEPTNO	DNAME
-----	-----	-----	-----
KING	10	10	ACCOUNTING
BLAKE	30	30	SALES
CLARK	10	10	ACCOUNTING
JONES	20	20	RESEARCH
...		...	
		40	OPERATIONS

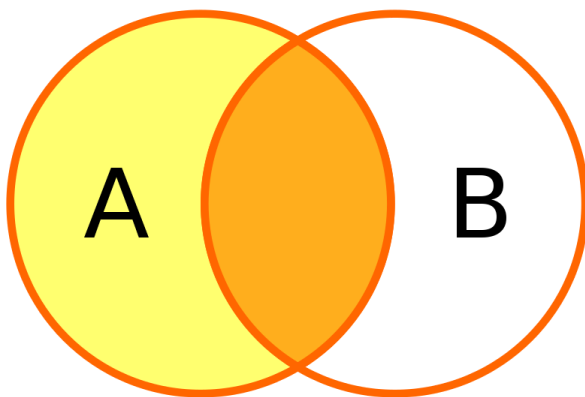
No hi ha empleats en el departament
OPERATIONS

- La unió que retorna només files coincidents és una unió interna.
- La unió que retorna files coincidents, així com les files no coincidents de la taula esquerra o dreta, es una unió externa esquerra o dreta.
- La unió que retorna files d'una unió interna i d'una unió esquerra i dreta és una unió externa completa.

LEFT JOIN

Retorna els valors coincidents d'ambdues taules, així com els no coincidents de la taula que està a l'esquerra de la clàusula join.

Utilitzant la teoria de conjunts sobre els conjunts A (valors clau forana taula A) i B (valors clau primària taula B), podriem representar el LEFT JOIN amb aquells valors que pertanyen a A (coincidentes i no coincidents):



Exemple: Volem les dades de TOTS els empleats amb el nom del seu departament.

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e
LEFT JOIN departments d
ON (e.department_id = d.department_id);
```

last_name	department_id	department_name
Greenberg	10	Finance
Faviet	10	Finance
Chen	10	Finance
Sciarra	10	Finance
Urman	10	Finance
Popp	10	Finance
Khoo	3	Purchasing
Hunold	6	IT
King	9	Executive
De Haan	9	Executive
Austin	6	IT
Kochhar	9	Executive
Pataballa	6	IT
Ernst	6	IT
Lorentz	6	IT
Raphaely	3	Purchasing

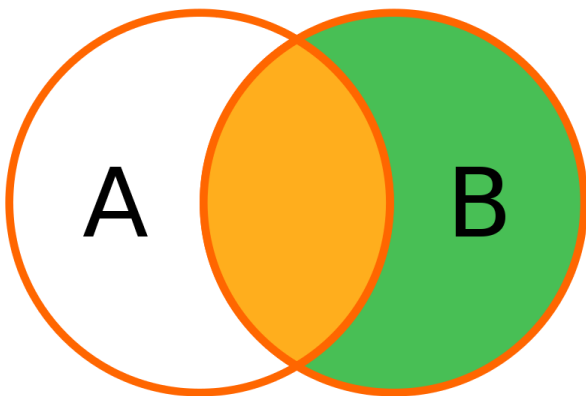
(16 rows)

Els valors dels camps de la taula dreta quan no hi ha correspondència seran NUL.

RIGHT JOIN

Retorna els valors coincidents d'ambdues taules, així com els no coincidents de la taula que està a la dreta de la clàusula join.

Utilitzant la teoria de conjunts sobre els conjunts A (valors clau forana taula A) i B (valors clau primària taula B), podríem representar el RIGHT JOIN amb aquells valors que pertanyen a B (coincidents i no coincidents):



Exemple: Volem les dades de TOTS els departaments amb algunes dades dels empleats que hi treballen.

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e
RIGHT JOIN departments d
ON (e.department_id = d.department_id);
```

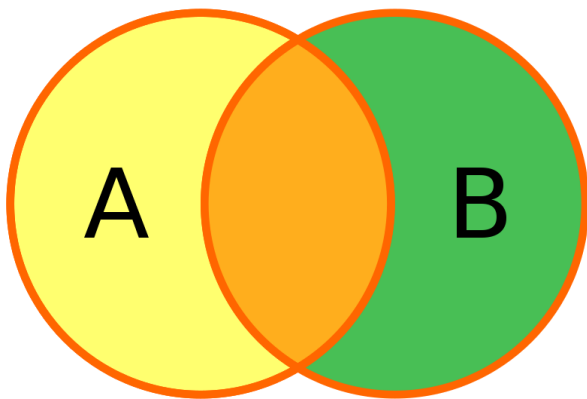
last_name	department_id	department_name
		Administration
		Marketing
Raphaely	3	Purchasing
Khoo	3	Purchasing
		Human Resources
		Shipping
Lorentz	6	IT
Ernst	6	IT
Pataballa	6	IT
Austin	6	IT
Hunold	6	IT
		Public Relations
		Sales
Kochhar	9	Executive
De Haan	9	Executive
King	9	Executive
Popp	10	Finance
Urman	10	Finance
Sciarra	10	Finance
Chen	10	Finance
Faviet	10	Finance
Greenberg	10	Finance
		Accounting
		Operations

(24 rows)

FULL JOIN

Retorna els valors coincidents d'ambdues taules, així com els no coincidents de les taules que estan a dreta i esquerra de la clàusula join.

Utilitzant la teoria de conjunts sobre els conjunts A (valors clau forana taula A) i B (valors clau primària taula B), podríem representar el FULL JOIN mitjançant la operació UNIÓ:



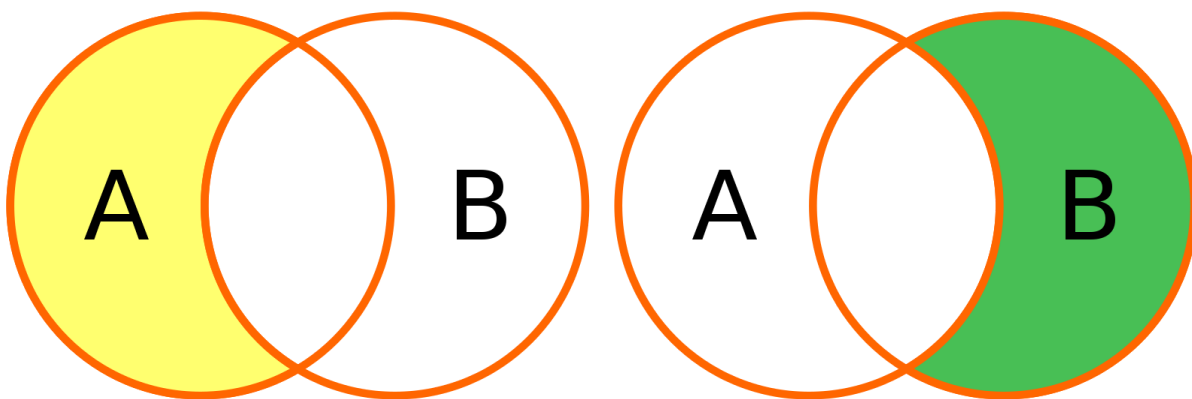
Exemple: Volem les dades de TOTS els departaments amb algunes dades dels empleats que hi treballen.

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e
FULL JOIN departments d
ON (e.department_id = d.department_id);
```

last_name	department_id	department_name
		Administration
		Marketing
Raphaely	3	Purchasing
Khoo	3	Purchasing
		Human Resources
		Shipping
Lorentz	6	IT
Ernst	6	IT
Pataballa	6	IT
Austin	6	IT
Hunold	6	IT
		Public Relations
		Sales
Kochhar	9	Executive
De Haan	9	Executive
King	9	Executive
Popp	10	Finance
Urman	10	Finance
Sciarra	10	Finance
Chen	10	Finance
Faviet	10	Finance
Greenberg	10	Finance
		Accounting
		Operations
Dere		
(25 rows)		

Amb les clàusules LEFT o RIGHT JOIN podem descartar també els elements coincidents (de la intersecció), per tant, que només ens retorni els elements no coincidents de la taula esquerra o dreta del join. Per fer això afegirem una condició de Null mitjançant una clàusula Where.

Utilitzant la representació de la teoria de conjunts estariem davant d'una operació de diferència: $A - B$ i $B - A$:



Exemple 1: Volem obtenir les dades de tots els empleats que no estan assignats a cap departament.

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e
LEFT JOIN departments d
ON (e.department_id = d.department_id)
WHERE d.department_id IS NULL;
```

last_name	department_id	department_name
Dere		

(1 row)

Exemple 2: Volem obtenir les dades de tots els departaments que no tenen assignats cap treballador.

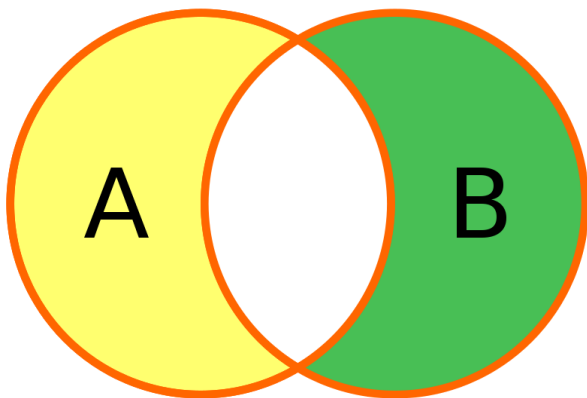
```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e
RIGHT JOIN departments d
ON (e.department_id = d.department_id)
WHERE e.employee_id IS NULL;
```

last_name	department_id	department_name
		Administration
		Marketing
		Human Resources
		Shipping
		Public Relations
		Sales
		Accounting
		Operations

(8 rows)

Amb la clàusula FULL JOIN podem descartar també els elements coincidents (de la intersecció), per tant, que només ens retorni els elements no coincidents de la taula esquerra i dreta del join. Per fer això afegirem una condició de Nul mitjançant una clàusula Where.

Utilitzant la representació de la teoria de conjunts estariem davant d'una operació de diferència simètrica entre A i B:



Exemple 2: Volem obtenir les dades de tots els departaments que no tenen assignats cap treballador i dels treballadors que no tenen assignat cap departament.

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e
FULL JOIN departments d
ON (e.department_id = d.department_id)
WHERE e.employee_id IS NULL OR d.department_id IS NULL;
```



```

last_name | department_id | department_name
-----+-----+-----
          |               | Administration
          |               | Marketing
          |               | Human Resources
          |               | Shipping
          |               | Public Relations
          |               | Sales
          |               | Accounting
          |               | Operations
Dere
(9 rows)

```

FUNCIONS D'AGREGAT DE DADES

Quan vàrem tractar les funcions vàrem veure que teníem funcions de fila única i funcions de files múltiples, aquestes són les funcions d'agregat de dades, és a dir, reben les dades de diverses files (agrupen la informació) i retornen un valor sobre una sola fila.

EMP

DEPTNO	SAL
10	2450
10	5000
10	1300
20	800
20	1100
20	3000
20	3000
20	2975
30	1600
30	2850
30	1250

“màxim salari en la taula EMP”

MAX (SAL)
5000

FUNCIONS AVG I SUM

Podem utilitzar les funcions **AVG** i **SUM** per obtenir la mitjana o la suma de dades numèriques.

Exemple: Obtenim la mitjana i la suma total dels salaris dels treballadors del departament 8

```

SELECT AVG(salary), SUM(salary)
FROM   employees
WHERE  department_id = 8;

```

```

      avg      |      sum
-----+-----
9616.666666666667 | 57700.00
(1 row)

```

FUNCIONS MIN I MAX

Posem utilitzar les funcions **MIN** i **MAX** per obtenir els valors mínim i màxim amb qualsevol tipus de dades (text, numèriques o dates).

Exemple: Obtenim la data de contractació dels treballadors més antic i més nou del departament 8

```
SELECT AVG(salary), SUM(salary)
FROM   employees
WHERE  department_id = 8;
```

```
min | max
-----+-----
1996-10-01 | 2000-01-04
(1 row)
```

FUNCIÓ COUNT

La funció **COUNT** ens serveix per contar registres.

La funció COUNT(*) ens serveix per contar el nombre de registres que té una taula.

Exemple: Obtenim el nombre de treballadors assignats al departament 8

```
SELECT COUNT(*)
FROM   employees
WHERE  department_id = 8;
```

```
count
-----
      6
(1 row)
```

La funció COUNT(expressió) retorna el nombre de files 'no nul·les' de expressió.

Exemple: Obtenim el nombre de treballadors que tenen assignada comissió

```
SELECT COUNT(comission)
FROM   employees;
```

```
count
-----
     15
(1 row)
```

La funció COUNT(DISTINCT expressió) retorna el nombre de valors **diferents** no nuls de expressió.

Exemple: Obtenim el nombre de comissions diferents de la taula empleats

```
SELECT COUNT(DISTINCT comission)
FROM employees;
```

```
count
-----
      6
(1 row)
```

Les funcions de grup ignoren els valors nuls de les columnes.

Exemple: Obtenim la mitjana de les comissions

```
SELECT AVG(comission)
FROM employees;
```

```

      avg
-----
0.177333333333333333
(1 row)

```

Mitjançant la funció COALESCE podem forçar un valor perquè les funcions de grup puguin incloure els valors nul.

Exemple: Obtenim la mitjana de les comissions, en cas que la comissió sigui nul la substituïm per 0.

```
SELECT AVG(COALESCE(comission,0))
FROM employees;
```

```

      avg
-----
0.0665000000000000000000000000000000
(1 row)

```

AGRUPACIÓ DE DADES

Moltes vegades no ens interessa aplicar una funció de grup sobre totes les files retornades per la consulta, sinó que ens interessa agrupar els resultats segons algun camp o expressió, creant subgrups de dades.

DEPTNO	SAL		DEPTNO	AVG (SAL)
10	2450		10	2916.6667
10	5000			
10	1300			
20	800		20	2175
20	1100			
20	3000			
20	3000			
20	2975			
30	1600		30	1566.6667
30	2850			
30	1250			
30	950			
30	1500			
30	1250			

2916.6667

“mitjana de
salaries
en EMP
per a cada
departament”

2175

1566.6667

Per agrupar dades utilitzarem la clàusula **GROUP BY**:

```
SELECT      column, group_function
FROM        table
[WHERE      condition]
[GROUP BY  group_by_expression]
[ORDER BY  column];
```

[!important] Totes les columnes esmentades a la SELECT que no són funcions de grup, han d'estar a la clàusula GROUP BY.

Exemple: Obtenim la mitjana de salaris de cada departament.

```
SELECT department_id, ROUND(AVG(salary),2) Mitjana
FROM   employees
GROUP BY department_id
ORDER BY department_id;
```

```
department_id | mitjana
-----+-----
1 | 4400.00
2 | 9500.00
3 | 4150.00
4 | 6500.00
5 | 5885.71
6 | 5760.00
7 | 10000.00
8 | 9616.67
9 | 19333.33
10 | 8600.00
11 | 10150.00
(11 rows)
```

La columna referència per GROUP BY no és obligatori que estigui a la clàusula SELECT.

Exemple: Consulta d'agrupació utilitzant més d'una columna SELECT department_id, job_id, ROUND(AVG(salary),2) Mitjana FROM employees GROUP BY department_id, job_id ORDER BY department_id;

Qualsevol columna o expressió en la SELECT que no sigui una funció agregada, ha de ser especificada en la clàusula GROUP BY.

```
SQL> SELECT deptno, COUNT(ename)
2 FROM emp;
```

```
SELECT deptno, COUNT(ename)
      *
ERROR at line 1:
ORA-00937: not a single-group group function
```

Columna NO especificada en la clàusula GROUP BY

No podem fer servir una clàusula WHERE per restringir grups. En aquest cas hem d'utilitzar la clàusula HAVING per restringir grups.

```
SQL> SELECT department_id, AVG(salary)
2 FROM employees
3 WHERE AVG(salary) > 2000
4 GROUP BY department_id;
```

```
WHERE AVG(salary) > 2000
      *
ERROR at line 3:
ORA-00934: group function is not allowed here
```

No puede usar la clàusula WHERE para restringir grupos

Utilitzem la clàusula **HAVING** per restringir grups:

- Els registres són agrupats.
- S'aplica la funció de grup.
- Els grups que es corresponen amb la clàusula HAVING es visualitzen

```

SELECT      column, group_function
FROM        table
[WHERE      condition]
[GROUP BY   group_by_expression]
[HAVING     group_condition]
[ORDER BY   column];

```

Exemple: Consulta que retorna el valor del salari màxim de cada departament, però només aquells departaments que el seu salari màxim sigui superior a 10000

```

SELECT department_id, MAX(salary)
FROM   employees
GROUP BY department_id
HAVING MAX(salary) > 10000;

```

department_id	max
11	12000.00
9	24000.00
3	11000.00
10	12000.00
2	13000.00
8	14000.00

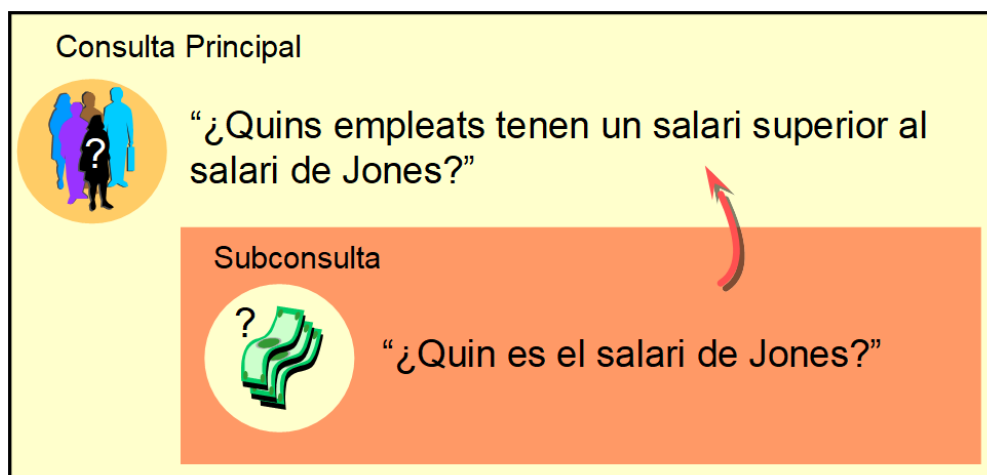
(6 rows)

SUBCONSULTES

Utilitzarem subconsultes quan volem aplicar un filtre sobre dades que no coneixem d'entrada, però que podem obtenir mitjançant una consulta.

- Davant del següent problema:

Qui té un salari superior al de l'empleat amb cognom 'Jones'?



- La subconsulta s'executa una vegada i abans de la consulta principal.
- El resultat de la subconsulta és usat per la consulta principal externa.

```
SELECT  select_list
FROM    table
WHERE   expr operator
        (SELECT  select_list
          FROM    table);
```

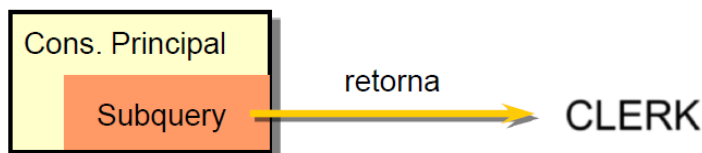
```
SELECT last_name
FROM   employees
WHERE  salary >
      (SELECT salary
       FROM   employees
       WHERE  employee_id = 7566);
```

INSTRUCCIONS PER LES SUBCONSULTES

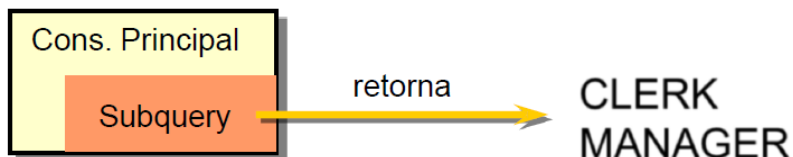
- Escriu les subconsultes entre parèntesis.
- Una subconsulta ha d'aparèixer a la dreta de l'operador.
- No afegeixis ORDER BY a una subconsulta.
- Utilitza operadors a nivell de fila per subconsultes que retornin només una fila.
- Utilitza operadors que actuen sobre diversos registres per subconsultes que retornin més d'una fila.

TIPUS DE SUBCONSULTES

- Subconsultes d'una sola fila:



- Subconsultes de diverses files:



SUBCONSULTES D'UNA SOLA FILA

- Retornen una sola fila.
- Utilitzem comparadors d'una sola fila.

Operador	Significat
=	Igual a
>	Major que
>=	Major que o igual a
<	Menor que
<=	Menor que o igual a
<>	No igual a

Exemple: Volem algunes dades dels empleats que treballen en la mateixa feina que l'empleat amb id = 102 i que cobren un salari superior al de l'empleat amb id = 110.

```
SELECT first_name, last_name, job_id
FROM employees
WHERE job_id = ( SELECT job_id
                  FROM employees
                  WHERE employee_id = 102 )
AND salary > ( SELECT salary
                FROM employees
                WHERE employee_id = 110 )
```

- Una subconsulta pot ser de qualsevol tipus, per tant, podem utilitzar funcions d'agregat de dades.

Exemple: Volem obtenir les dades de tots els empleats que cobren el salari més baix.

```
SELECT first_name, last_name, job_id, salary
FROM employees
WHERE salary = ( SELECT min(salary)
                  FROM employees )
```

- Quan la consulta principal conté funcions d'agregat de dades, de la mateixa manera que utilitzem la clàusula WHERE podem utilitzar la clàusula HAVING amb una subconsulta.

Exemple: Volem obtenir el salari del treballador que cobra menys de cada departament, però només d'aquells departaments on el treballador que cobra menys cobri més que el salari menor del departament 20.

```
SELECT department_id, MIN(salary)
FROM employees
GROUP BY department_id
HAVING MIN(salary) > ( SELECT MIN(salary)
                       FROM employees
                       WHERE department_id = 20 )
```

[!Important]

La següent subconsulta no funcionaria: (Proveu-la)


```
SELECT employee_id, first_name
FROM employees
WHERE salary = (SELECT MIN(salary)
                FROM employees
                GROUP BY department_id)
```

Hem d'anar molt amb compte amb el fet que les subconsultes retornin una sola fila quan utilitzem els operadors =, >, <, >=, <= i <> o !=

SUBCONSULTES DE DIVERSES FILES

- Retornen diverses files.
- Utilitzem comparadors de diverses files.

Operador	Significat
IN	Igual als valors d'una llista
ANY	Compara els valors amb qualsevol valor retornat per la subconsulta
ALL	Compara els valors amb tots els valors retornats per la subconsulta

- Utilitzant l'operador ANY, la condició s'ha de donar per a qualsevol valor retornat per la subconsulta.

```
SELECT employee_id, salary
FROM employees
WHERE salary < ANY
      (SELECT salary
       FROM employees
       WHERE job_id = 5)
AND job_id <> 5;
```

- Utilitzant l'operador ALL, la condició s'ha de donar per a tots els valors retornats per la subconsulta.

```
SELECT employee_id, first_name
FROM employees
WHERE salary > ALL
      (SELECT AVG(salary)
       FROM employees
       GROUP BY department_id)
```

- Utilitzant l'operador IN, la condició s'ha de donar si la dada a comparar pertany a la llista de valors retornats per la subconsulta.

Exemple: Volem obtenir el nom i cognom de tots els empleats que no son caps (manager).

```
SELECT emp.first_name, emp.last_name
FROM employees emp
```

```
WHERE emp.employee_id NOT IN (SELECT mgr.manager_id
                               FROM employees mgr)
```

- Si ens fixem bé: NOT IN és equivalent a <> ALL, i IN és equivalent a = ANY

SUBCONSULTES MULTI-COLUMNA

En tots els exemples que hem vist fins ara sempre comparem el valor d'una dada amb el valor o valors retornats per la subconsulta. En certes ocasions ens pot interessar comparar el conjunt de valors que formen les dades de diverses columnes amb els retornats per la subconsulta.

Exemple: Visualitzar el nom, número de departament, salari i comissió de qualsevol empleat el salari i la comissió del qual es corresponguin (ambdós) amb la comissió i salari de qualsevol empleat del departament 30.

```
SELECT first_name, department_id, salary, COALESCE(commision)
FROM employees
WHERE (salary, COALESCE(commision)) IN (SELECT salary, COALESCE(commision)
                                         FROM employees
                                         WHERE department_id = 30);
```

SUBCONSULTES EN LA CLÀUSULA FROM

Fins ara hem vist l'us més habitual de les subconsultes, que és en la clàusula WHERE per a resoldre els problemes que platejavem en l'inici d'aquest apartat. Una altre us, potser no tant habitual, és el d'executar una subconsulta en la clàusula FROM de manera que actui com una vista. En aquest cas s'executa la subconsulta i el resultat de la mateixa és una taula temporal o vista que podem utilitzar per combinar amb d'altres taules de la base de dades. Forçosament haurem d'utilitzar un àlies de taula per a la subconsulta.

Exemple: Volem obtenir el nom, salari, departament i salari mitjà del departament on treballa cada treballador, però només dels empleats el salari dels quals sigui superior a la mitjana del departament. D'entrada no tenim cap taula on registrem la mitjana del salari de cada departament, per això utilitzarem una subconsulta, que ens proporcionarà aquesta informació.

```
SELECT a.first_name, a.salary, a.department_id, b.mitjana
FROM employees a JOIN (SELECT department_id, AVG(salary) mitjana
                       FROM employees
                       GROUP BY department_id) b
ON a.department_id = b.department_id
WHERE a.salary > b.mitjana;
```