

Lenguajes de Bases de Datos Relacionales

Tema 8. SQL: Manipulación de Datos (DML)

B3

Lenguajes de Bases de Datos Relacionales

Tema 8. SQL: Manipulación de Datos (DML)

Tema 8. SQL: DML

1

Objetivos

- Aprender la **sintaxis** de la sentencia SELECT del **SQL** con el fin de escribir sentencias de *consulta y de modificación* de datos ya almacenados en una base de datos relacional
DML: Data Manipulation Language del SQL
 - Diferenciar entre el lenguaje ANSI SQL y los **dialectos SQL** implementados por sistemas de bases de datos comerciales
- Entender el *modelo de ejecución* de las sentencias SQL
- Conocer **buenas prácticas de programación** de sentencias SELECT

Tema 8. SQL: DML

2

Bibliografía

- [CB 2015] Connolly, T.M.; Begg C.E.: ***Database Systems: A Practical Approach to Design, Implementation, and Management***, 6th Edition. Pearson. (Capítulos 6 y 7)
- [EN 2016] Elmasri, R.; Navathe, S.B.: ***Fundamentals of Database Systems***, 7th Edition. Pearson. (Caps. 6 y 7)

Tema 8. SQL: DML

3

Contenidos

- 8.1 **Consulta**: recuperación o selección de datos
 - *Running Example*. Presentación del esquema y datos utilizados en los ejemplos
 - Sentencia **SELECT** del lenguaje estándar **ANSI SQL**
 - Con indicaciones acerca del lenguaje **Oracle SQL**
 - Ejemplos resueltos
- 8.2 **Buenas prácticas** de programación
- 8.3 **Modificación** de datos
 - Introducción, actualización y borrado de datos
- Anexo: Estructura el lenguaje **Oracle SQL**

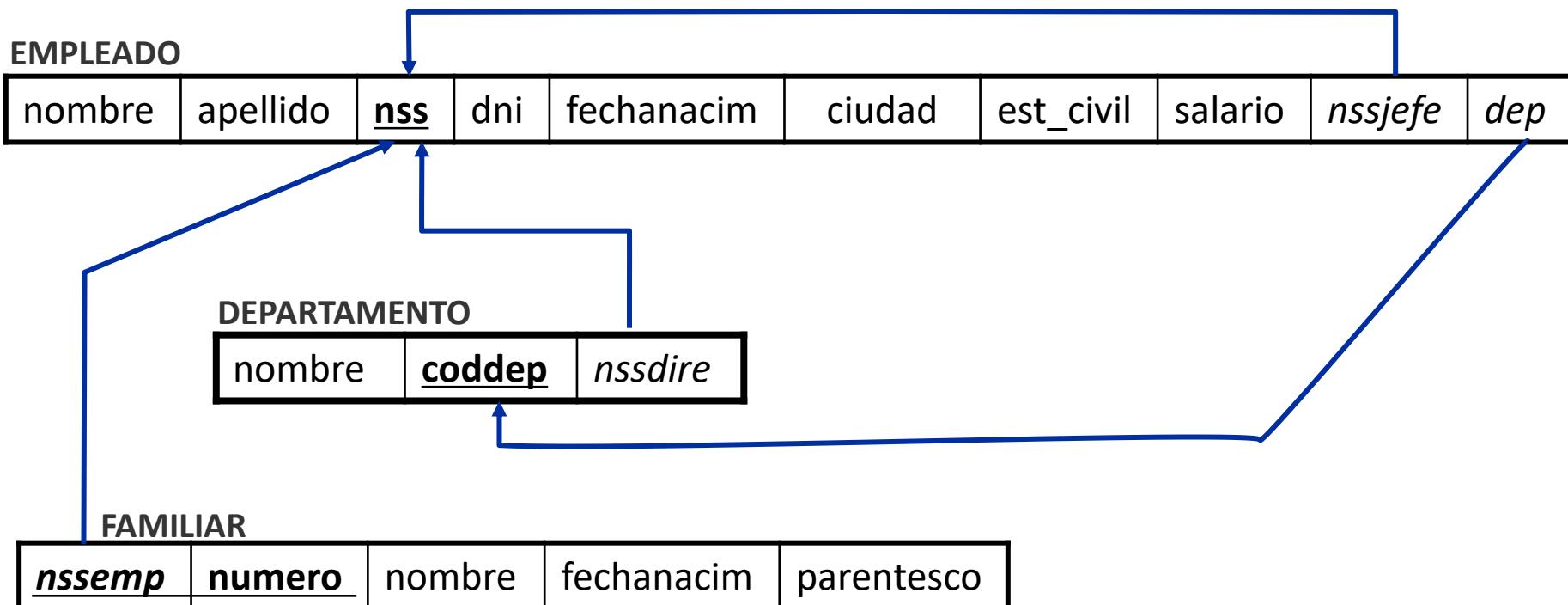
8.1 Consulta

4

- SQL permite **extraer información** almacenada en las tablas de una base de datos relacional
 - Es decir, dar respuesta a consultas como esta:
“Obtener un listado de nombres y salarios de los empleados del departamento INVESTIGACION”
- Es necesario **escribir** (programar) **las consultas** utilizando un conjunto de instrucciones que el SGBD pueda entender
- Hay que redactar cada consulta indicando **qué** datos queremos extraer: **qué condiciones deben cumplir**
 - “Mostrar nombres y apellidos de los empleados que viven en Yecla”
 - Deben ser filas de la **tabla EMPLEADO tales que el valor de la columna “ciudad” debe ser igual a ‘YECLA’**, y se desea mostrar los valores de las columnas “nombre” y “apellido”
- ...y el SGBD interpretará la consulta, buscará los datos en la base de datos, y los devolverá como resultado

SQL. Running Example

5



- Un esquema (simplificado) de base de datos “Empresa”

Esquema Lógico Específico (Oracle)

6

```

CREATE TABLE EMPLEADO (
    nombre          VARCHAR(25) NOT NULL,
    apellido        VARCHAR(15) NOT NULL,
    nss             NUMBER(12) NOT NULL,
    dni             CHAR(9)    NOT NULL,
    fechanacim     DATE       NULL,
    ciudad          VARCHAR(30),
    est_civil       CHAR(1),
    salario         NUMBER(6,2) DEFAULT 1000 NOT NULL,
    nssjefe         NUMBER(12),
    dep             CHAR(3)    NULL,
    cuantos_familiares NUMBER(2) DEFAULT 0 NOT NULL,
    CONSTRAINT emp_pk      PRIMARY KEY (nss),
    CONSTRAINT emp_ak      UNIQUE (dni),
    CONSTRAINT emp_fk_emp  FOREIGN KEY (nssjefe)
                           REFERENCES EMPLEADO(nss),
    CONSTRAINT emp_ec      CHECK (est_civil IN ('S','C','V','D','P')),
    CONSTRAINT emp_salario CHECK (salario > 0)
    CONSTRAINT emp_jefe    CHECK (nssjefe <> nss)
);

```

Importante: no es posible incluir la clave ajena para hacer referencia al departamento (columna **dep**), porque aún no se ha creado la tabla DEPARTAMENTO



Esquema Lógico Específico (Oracle)

7

```
CREATE TABLE DEPARTAMENTO (
    nombre          VARCHAR(20) NOT NULL,
    coddep         CHAR(3)     NOT NULL,
    nssdire        NUMBER(12)  NULL,
    CONSTRAINT dep_pk PRIMARY KEY (coddep),
    CONSTRAINT dep_ak UNIQUE(nssdire),
    CONSTRAINT dep_fk_emp FOREIGN KEY (nssdire)
                                REFERENCES EMPLEADO(nss);
);
/* Ciclo referencial: una vez que se ha creado la tabla
DEPARTAMENTO ya se puede definir la clave ajena
desde EMPLEADO a DEPARTAMENTO: */

ALTER TABLE EMPLEADO ADD
CONSTRAINT emp_fk_dep
FOREIGN KEY (dep) REFERENCES DEPARTAMENTO(coddep);
```

Esquema Lógico Específico (Oracle)

8

```
CREATE TABLE FAMILIAR (
    nssemp      NUMBER(12)      NOT NULL,
    numero       NUMBER(2)       NOT NULL,
    nombre       VARCHAR(25)     NOT NULL,
    fechanacim   DATE          NOT NULL,
    parentesco   VARCHAR(10)     NOT NULL
        CHECK (parentesco IN ('MADRE','PADRE',
                               'HIJO', HIJA', 'ABUELO', 'ABUELA',
                               'SOBRINO', 'SOBRINA', 'TIA', 'TIO')),
    CONSTRAINT fam_pk PRIMARY KEY (nssemp, numero),
    CONSTRAINT fam_fk_emp FOREIGN KEY (nssemp)
        REFERENCES EMPLEADO(nss),
    CONSTRAINT fam_numero_ok CHECK (numero > 0)
);
```

SQL. Un Estado del esquema

9

EMPLEADO										
nombre	apellido	nss	dni	fechanacim	ciudad	est_civil	salario	nssjefe	dep	
JONÁS	SOLANO	123	11A	10/10/1945	MURCIA	P	1100	111	D1	
RIGOBERTA	CALAVERA	321	21C	12/11/1974	YECLA	C	900	333	D3	
EUSEBIO	MULETAS	222	22B	01/01/1969	TOTANA	D	2100	123	D2	
MACARENO	SOSO	111	23D	06/04/1944	JUMILLA	S	1100	NULL	D1	
CASIANA	FABERGÉ	333	33B	15/06/1943	MURCIA	V	920	123	D3	
FILOMENA	RASCAS	234	34E	18/07/1970	MURCIA	C	1100	111	D1	
GUMERSINDA	MIMOS	543	45F	10/02/1980	PINOSO	P	850	NULL	NULL	

FAMILIAR

nssemp	numero	nombre	fechanacim	parentesco
123	1	JONÁS	17/05/1992	HIJO
321	2	RÓMULA	23/09/1923	ABUELA
222	1	ELEUTERIO	30/10/2002	HIJO
321	1	RENATA	10/03/2002	HIJA
111	2	JULIANA	10/10/1936	MADRE
321	3	TORCUATA	17/05/1938	ABUELA
111	3	SINFOROSA	23/09/1947	ABUELA

DEPARTAMENTO

nombre	coddep	nssdire
INVESTIGACION	D2	222
ADMINISTRACION	D1	111
PERSONAL	D3	333
TRAINING	D4	NULL

SQL. Sentencia **SELECT**

10

- Instrucción básica de **obtención de información**



SELECT lista_columnas

FROM lista_tablas

WHERE condición ; ← siempre termina con un punto y coma

- La consulta selecciona las filas de lista_tablas que satisfacen condición y proyecta el resultado sobre las columnas de lista_columnas
- Resultado: **Tabla** con las con las columnas indicadas y las filas seleccionadas

* *DNI y ciudad del empleado con apellido 'RASCAS'*

SELECT dni, ciudad

FROM empleado

WHERE apellido = 'RASCAS';

dni	ciudad
34E	MURCIA

SELECT. Consultas básicas

11

* Código y nombre del departamento dirigido por el empleado con NSS 111

```
SELECT coddep, nombre  
FROM departamento  
WHERE nssdire = 111;
```

* Nombres y apellidos de los empleados cuyo estado civil no sea 'soltero'

```
SELECT nombre, apellido  
FROM empleado  
WHERE est_civil <> 'S';
```

* Nombres de los familiares que tienen un parentesco distinto a 'HIJO'

```
SELECT nombre  
FROM familiar  
WHERE NOT (parentesco = 'HIJO');
```

* NSS y nombre de empleados que cobran entre 900€ y 1200€

```
SELECT nombre, apellido  
FROM empleado  
WHERE salario BETWEEN 900 AND 1200;
```

Buenas prácticas de redacción SQL
►Palabras reservadas en **MAYÚSCULA**
►Atributos y tablas en **minúscula**

SELECT. Consultas básicas

12

- Cualquier nº de **condiciones** (selección/reunión) en SELECT
 - * Para cada empleado del departamento *D1* o del *D3*, que vivan en Murcia y tengan un salario no superior a 1000€, obtener su dni, nombre y estado civil

```
SELECT dni, nombre, est_civil
FROM empleado
WHERE (dep = 'D1' OR dep = 'D3')
      AND ciudad = 'MURCIA'
      AND NOT (salario > 1000);
```

dni	nombre	est_civil
33B	CASIANA	V

- Una SELECT puede obtener **filas repetidas**

- No se eliminan las filas duplicadas de forma automática
 - * Salario de los empleados de los departamentos *D1*, *D2* y *D3*

```
SELECT salario
FROM empleado
WHERE dep = 'D1' OR dep = 'D2' OR dep = 'D3';
```

salario
1100
900
2100
1100
920
1100

SELECT. Eliminar filas duplicadas

13

- SQL, en general, **no elimina filas repetidas** del resultado de una consulta porque...
 - Eliminar duplicados es **costoso**
 - Ordenar + Recorrer + Eliminar duplicados
 - El usuario puede **desechar ver las filas repetidas** en el resultado
 - Si se desea, por ejemplo, calcular la suma o la media de los valores de cierta columna, es necesario mantener las filas duplicadas

* *Salarios de todos los empleados*

```
SELECT salario  
FROM empleado;
```

salario
1100
900
2100
1100
920
1100
850

SELECT. Eliminar filas duplicadas

14

□ Cláusula **DISTINCT** (o **UNIQUE**):

- Para **eliminar filas repetidas** del resultado de una consulta
- Así, el resultado es una Relación del Modelo Relacional Formal (un conjunto de filas)

* Salarios distintos de los empleados

```
SELECT DISTINCT salario
FROM empleado;
```

salario
1100
900
2100
920
850

* NSS de los empleados que tienen algún familiar que sea su hija o su abuela.

```
SELECT nssemp
FROM familiar
WHERE parentesco = 'HIJA'
OR parentesco = 'ABUELA';
```

nssemp
321
321
321
111

```
SELECT DISTINCT nssemp
FROM familiar
WHERE parentesco = 'HIJA'
OR parentesco = 'ABUELA';
```

nssemp
321
111

SELECT. Uso de * (asterisco o star)

15

- Obtención de los valores de **todas las columnas** de las filas seleccionadas

- Uso del símbolo ***** (que significa “todas las columnas”)
- Así **no** hay que **listar todos los nombres** tras la cláusula SELECT

```
SELECT *
FROM empleado
WHERE dep = 'D1';
```

nombre	apellido	<u>nss</u>	dni	fechanacim	ciudad	est_civil	salario	<i>nssjefe</i>	dep
JONÁS	SOLANO	123	11A	10/10/1945	MURCIA	P	1100	111	D1
MACARENO	SOSO	111	23D	06/04/1944	JUMILLA	S	1100	NULL	D1
FILOMENA	RASCAS	234	34E	18/07/1970	MURCIA	C	1100	111	D1

```
SELECT *
FROM departamento
WHERE nombre = 'INVESTIGACION';
```

nombre	<u>coddep</u>	<i>nssdire</i>
INVESTIGACION	D2	222

SELECT. Cadenas de caracteres

16

□ Operador **LIKE**

- Comparación de cadenas de caracteres (textos)
- Suele usar caracteres reservados (comodines):
 - % significa “una cantidad cualquiera de caracteres cualesquiera (letras y dígitos)”
 - _ significa **un solo carácter** (una letra o un dígito)

* Nombres y apellidos de empleados de Las Torres de Cotillas o Cabezo de Torres

```
SELECT nombre, apellido
FROM empleado
WHERE ciudad LIKE '%TORRES%';
```

* Nombres y fecha de nacimiento de los familiares padres o madres de empleados

```
SELECT nombre, fechanacim
FROM familiar
WHERE parentesco LIKE '_ADRE';
```

□ Operador ||

- Concatenación de cadenas de caracteres

* Nombres completos, en una sola columna, de los empleados de las ciudades de Aledo, Alguazas, Alhama y Alcantarilla

```
SELECT nombre || ' ' || apellido
FROM empleado
WHERE ciudad LIKE 'AL%';
```

SELECT. Aritmética

17

□ Operaciones aritméticas

- Aplicación de operadores aritméticos (+, -, *, /) sobre valores numéricos
- Se muestra el resultado de la operación, pero **no se modifica ningún dato almacenado en la tabla**

* ¿Cómo quedarían los salarios de los empleados del departamento D3 tras un aumento del 10% ?

```
SELECT apellido, nombre, 1.1*salario
FROM empleado
WHERE dep = 'D3';
```

apellido	nombre	1.1*salario
CALAVERA	RIGOBERTA	990
FABERGÉ	CASIANA	1012

* ¿Cómo quedaría el salario de los empleados del departamento D2 si le sumamos 200 euros?

```
SELECT apellido, nombre, salario+200
FROM empleado
WHERE dep = 'D2';
```

apellido	nombre	salario+200
MULETAS	EUSEBIO	2300

► el valor de los salarios en la tabla EMPLEADO no cambia



SELECT. Fechas (Oracle)

18

- **Funciones para convertir fechas (DATE) en cadenas de caracteres y al contrario**
 - **TO_CHAR(*fecha, formato*)**: convierte un valor de tipo DATE a una cadena de caracteres que expresa una fecha (o parte de ella) en el *formato* que se establece como segundo parámetro
 - **TO_DATE(*cadena_texto, formato*)**: convierte una cadena de caracteres, escrita en el *formato* indicado en el 2º parámetro, en un valor de tipo DATE

* Año de nacimiento y ciudad del empleado con apellido 'RASCAS'

```
SELECT TO_CHAR(fechanacim, 'yyyy'), ciudad
FROM empleado
WHERE apellido = 'RASCAS';
```

TO_CHAR(fechnacim, 'yyyy')	ciudad
1970	MURCIA

* Nombres y apellidos de los empleados nacidos antes de 1972

```
SELECT nombre, apellido
FROM empleado
WHERE fechanacim < TO_DATE('01/01/1972', 'dd/mm/yyyy');
```

SELECT. Fechas (Oracle)

19

- El “*formato*” es una cadena de caracteres que describe el formato en el que se desea visualizar/almacenar un valor de tipo DATE o de tipo CHAR
 - Ejemplos de formato:
 - '20/05/19' está en el formato 'dd/mm/yy'
 - '11-Nov-2019' está en el formato 'dd-Mon-yyyy'
 - '2021' está en el formato 'yyyy'
 - '12/abril 17:45:29' está en el formato 'dd/month hh24:mi:ss'
 - Función **EXTRACT (campo FROM fecha)**
 - Devuelve el valor de un campo concreto de una columna tipo DATE
 - “*campo*” puede ser YEAR, MONTH, DAY, HOUR, MINUTE, SECOND
- * *Nombre y mes de nacimiento de los familiares nacidos antes de 1990*
- ```
SELECT nombre, EXTRACT (MONTH FROM fechanacim)
FROM familiar
WHERE EXTRACT (YEAR FROM fechanacim) < 1990;
```

# SELECT. Conjuntos de valores y Operador IN

20

- Un **conjunto explícito de valores** es una lista de valores, separados por comas, encerrada entre paréntesis
- Puede aparecer en la cláusula WHERE

\* Nss de lxs empleadxs que tienen padres/madres o abuelxs a su cargo  
SELECT nssemp

FROM familiar

WHERE parentesco **IN ('MADRE', 'PADRE', 'ABUELO', 'ABUELA')**;

- **Operador IN (también NOT IN)**

∨ **IN V**

- Indica si el **valor v** pertenece al **conjunto de valores V**
- Devuelve TRUE si **algún** elemento e de V cumple que **v = e**

\* Nss de lxs empleadxs que tengan a cargo familiares que no sean hijxs ni sobrinxs  
SELECT nssemp

FROM familiar

WHERE parentesco **NOT IN ('HIJA', 'HIJO', 'SOBRINA', 'SOBRINO')**;

# SELECT. NULL

21

## □ NULL

- No es un valor, sino una marca que indica **desconocimiento o ausencia** de información
- **Comparar NULL** usando `=`, `<>`, `>`, `>=`, `<`, `<=`, **con cualquier cosa siempre da FALSE o UNKNOWN**
  - NULL es **distinto a cualquier otra cosa**, incluso a otro NULL

\* Nombres y apellidos de los empleados sin supervisores

```
SELECT nombre, apellido
FROM empleado
WHERE nssjefe = NULL;
```

| nombre  | apellido |
|---------|----------|
| (vacío) |          |

- Resultado: **tabla vacía**, porque ninguna fila cumple la condición de selección
- Para JONÁS SOLANO, por ejemplo, el valor de “nssjefe” es 111 y 111 no es igual a NULL. El resultado de la comparación es FALSE. Ok.
- Pero para MACARENO SOSO, el valor de “nssjefe” sí es NULL. Pero NULL no es igual a nada, ni siquiera a otro NULL. Así que el resultado también es FALSE. Ok.

# SELECT. NULL

22

- Es necesario un comparador específico para comprobar si una columna contiene o no el NULL
- **Operador IS NULL , IS NOT NULL**
  - ▼ **IS NULL**
    - Devuelve TRUE si ▼ es NULL
  - ▼ **IS NOT NULL**
    - Devuelve TRUE si ▼ no es un valor NULL

\* Nombres y apellidos de los empleados sin supervisores

```
SELECT nombre, apellido
FROM empleado
WHERE nssjefe IS NULL;
```

| nombre     | apellido |
|------------|----------|
| MACARENO   | SOSO     |
| GUMERSINDA | MIMOS    |

# SELECT. Orden de presentación

23

- SQL permite **presentar las filas** resultado de una consulta de forma **ordenada**: **Cláusula ORDER BY**
  - Ordenación según valores de una o varias columnas
  - Ascendente ASC (por defecto) o Descendente DESC
  - Suele ser una operación muy costosa
  - Las filas **no se ordenan en disco**: se ven ordenadas, pero no cambian dentro de la tabla

```
SELECT coddep, nombre
FROM departamento
ORDER BY nombre;
```

```
SELECT nssemp, nombre, fechanacim, parentesco
FROM familiar
ORDER BY nssemp, fechanacim DESC;
```

```
SELECT apellido, nombre, fechanacim
FROM empleado
WHERE ciudad = 'MURCIA'
ORDER BY apellido ASC, nombre DESC;
```

# SELECT. Reunión (JOIN)

24

- Hasta ahora hemos realizado SELECTs que extraen datos de una tabla: **sólo 1 tabla** en la cláusula FROM
- ¿Qué ocurre si necesitamos mostrar **datos que están almacenados en varias tablas?**
- Eso es muy habitual en nuestro día a día:
  - \* *Estante en el que está la copia número 2 del libro titulado ‘Yo, robot’ junto con el año de publicación de dicho libro.*
  - \* *Nombre y año de nacimiento de cada empleado junto con el nombre y parentesco de sus familiares.*
  - \* *Todos los datos del empleado ‘SOLANO’ junto con los datos de su departamento.*
  - \* *ISBN del libro con título ‘Mentira’ junto con el nombre y la ciudad de su editorial.*



Resolvamos esta última consulta...

# SELECT. Reunión (JOIN)

25

\* ISBN del libro con título ‘Mentira’ junto con el nombre y la ciudad de su editorial

- Si lo hacemos “a mano”, buscamos en la tabla LIBRO la fila que tiene como valor de “título” el texto ‘Mentira’
- Esa fila contiene los datos de dicho libro. Ya tenemos el ISBN y el nombre de la editorial (‘Edebé’)
- Pero en LIBRO no se almacenan las ciudades de las editoriales: eso está en la tabla EDITORIAL
- Hemos de buscar la fila de EDITORIAL cuyo valor de “nombre” es ‘Edebé’, es decir, aquella fila a la que está referenciando la fila del libro ‘Mentira’
- Y ahí encontramos la ciudad (‘Barcelona’)
- Resultado:

| ISBN          | nombre | ciudad    |
|---------------|--------|-----------|
| 9788468315775 | Edebé  | Barcelona |

# SELECT. Reunión (JOIN)

26

**EDITORIAL**

| nombre     | calle                    | numero | cod_post | ciudad              |
|------------|--------------------------|--------|----------|---------------------|
| Espasa     | Josefa Valcárcel         | 42     | 28027    | Madrid              |
| Santillana | Avenida de los Artesanos | 6      | 28760    | Tres Cantos, Madrid |
| Edebé      | Paseo San Juan Bosco     | 62     | 08017    | Barcelona           |

**LIBRO**

| ISBN          | título                           | año  | edicion | num_copias | editorial  |
|---------------|----------------------------------|------|---------|------------|------------|
| 9788408217251 | Un científico en el supermercado | 2019 | 1       | 14         | Espasa     |
| 9788491223542 | Malamandra                       | 2019 | 5       | 5          | Santillana |
| 9788468315775 | Mentira                          | 2015 | 4       | 12         | Edebé      |
| 9788467009101 | Divina comedia                   | 2010 | 50      | 25         | Espasa     |
| 9788468319612 | La nueva vida del señor Rutin    | 2014 | 23      | 16         | Edebé      |

# SELECT. Reunión (JOIN)

27

\* Todos los datos del empleado ‘SOLANO’ junto con los datos de su departamento

- Si lo hacemos “a mano”, buscamos en la tabla EMPLEADO la fila que tiene el valor ‘SOLANO’ en la columna “apellido”
- Esa fila contiene los datos de dicho empleado. Pero no tenemos todos los datos del departamento, sino sólo su código ‘D1’ (columna “dep”)
- Los datos de los departamentos están en DEPARTAMENTO
- Hemos de buscar aquella fila de DEPARTAMENTO cuyo valor de “coddep” es ‘D1’, es decir, aquella fila a la que está referenciando la fila del empleado ‘SOLANO’
- Y ahí encontramos los datos del departamento de ‘SOLANO’
- Resultado:

| nombre | apellido | nss | ... | dep | nombre         | coddep | nssdire |
|--------|----------|-----|-----|-----|----------------|--------|---------|
| JONÁS  | SOLANO   | 123 | ... | D1  | ADMINISTRACION | D1     | 111     |

# SELECT. Reunión (JOIN)

28

## EMPLEADO

| nombre     | apellido | nss | dni | añonacim | direccion   | ciudad  | salario | nssjefe | dep  |
|------------|----------|-----|-----|----------|-------------|---------|---------|---------|------|
| JONÁS      | SOLANO   | 123 | 11A | 1945     | C/PEZ, 10   | MURCIA  | 1100    | 111     | D1   |
| RIGOBERTA  | CALAVERA | 321 | 21C | 1974     | C/BOJ, 2    | YECLA   | 900     | 333     | D3   |
| EUSEBIO    | MULETAS  | 222 | 22B | 1969     | C/RIF, 6    | TOTANA  | 2100    | 123     | D2   |
| MACARENO   | SOSO     | 111 | 23D | 1944     | C/MAR, 4    | JUMILLA | 1100    | NULL    | D1   |
| CASIANA    | FABERGÉ  | 333 | 33B | 1943     | C/SOL, 8    | MURCIA  | 920     | 123     | D3   |
| FILOMENA   | RASCAS   | 234 | 34E | 1970     | C/NUEZ, 3   | MURCIA  | 1100    | 111     | D1   |
| GUMERSINDA | MIMOS    | 543 | 45F | 1980     | C/QUINTO, 5 | PINOSO  | 850     | NULL    | NULL |

## FAMILIAR

| nssemp | numero | nombre    | añonacim | parentesco |
|--------|--------|-----------|----------|------------|
| 123    | 1      | JONÁS     | 1992     | HIJO       |
| 321    | 2      | RÓMULA    | 1923     | ABUELA     |
| 222    | 1      | ELEUTERIO | 2002     | HIJO       |
| 321    | 1      | RENATA    | 2002     | HIJA       |
| 111    | 2      | JULIANA   | 1936     | MADRE      |
| 321    | 3      | TORCUATA  | 1938     | ABUELA     |
| 111    | 3      | SINFOROSA | 1947     | ABUELA     |

## DEPARTAMENTO

| nombre         | coddep | nssdire |
|----------------|--------|---------|
| INVESTIGACION  | D2     | 222     |
| ADMINISTRACION | D1     | 111     |
| PERSONAL       | D3     | 333     |
| TRAINING       | D4     | NULL    |

# SELECT. Reunión (JOIN)

29

- Lo que estamos haciendo es **sacar información de dos filas que están relacionadas entre sí**
- Y ese **vínculo** entre ellas está representado mediante la referencia que una fila (en una tabla ) tiene hacia la otra (en otra tabla)
  - ▣ **Clave Ajena → Clave Primaria**
- Bien ¿Y si en lugar de hacerlo solo para una fila (un solo empleado), lo hacemos **para todas las filas** (todos los empleados)?

# SELECT. Reunión (JOIN)

30

- \* *Para todo empleado, mostrar sus datos junto con los datos de su departamento*
- Es necesario combinar cada fila de la tabla EMPLEADO con aquella fila de DEPARTAMENTO cuyo valor en “coddep” sea igual al valor de “dep”
- Se consigue aplicando la **operación REUNIÓN (JOIN)** a las dos tablas

```
SELECT *
FROM empleado JOIN departamento ←
 ON dep = coddep;
```

# SELECT. Reunión (JOIN)

31

## EMPLEADO

| nombre     | apellido | nss | ... | dep  |
|------------|----------|-----|-----|------|
| JONÁS      | SOLANO   | 123 | ... | D1   |
| RIGOBERTA  | CALAVERA | 321 | ... | D3   |
| EUSEBIO    | MULETAS  | 222 | ... | D2   |
| MACARENO   | SOSO     | 111 | ... | D1   |
| CASIANA    | FABERGÉ  | 333 | ... | D3   |
| FILOMENA   | RASCAS   | 234 | ... | D1   |
| GUMERSINDA | MIMOS    | 543 | ... | NULL |

## DEPARTAMENTO

| nombre         | coddep | nssdire |
|----------------|--------|---------|
| INVESTIGACION  | D2     | 222     |
| ADMINISTRACION | D1     | 111     |
| PERSONAL       | D3     | 333     |
| TRAINING       | D4     | NULL    |

```
SELECT *
FROM empleado JOIN departamento
ON dep=coddep;
```

Resultado del JOIN: una tabla con una fila por cada dos filas vinculadas de las tablas origen

| nombre    | apellido | nss | ... | dep | nombre         | coddep | nssdire |
|-----------|----------|-----|-----|-----|----------------|--------|---------|
| JONÁS     | SOLANO   | 123 | ... | D1  | ADMINISTRACION | D1     | 111     |
| RIGOBERTA | CALAVERA | 321 | ... | D3  | PERSONAL       | D3     | 333     |
| EUSEBIO   | MULETAS  | 222 | ... | D2  | INVESTIGACION  | D2     | 222     |
| MACARENO  | SOSO     | 111 | ... | D1  | ADMINISTRACION | D1     | 111     |
| CASIANA   | FABERGÉ  | 333 | ... | D3  | PERSONAL       | D3     | 333     |
| FILOMENA  | RASCAS   | 234 | ... | D1  | ADMINISTRACION | D1     | 111     |

Importante: En el resultado del JOIN no está la empleada GUMERSINDA MIMOS , ¿Por qué? Tampoco aparece el departamento TRAINING. ¿Por qué?

# SELECT. Reunión (JOIN)

32

- \* *Datos de cada empleado junto con los datos de sus familiares*
- Es necesario combinar cada fila de EMPLEADO con cada fila de FAMILIAR tal que el valor de “nssemp” coincida con el de “nss”
- Se consigue aplicando la **operación REUNIÓN (JOIN)** a las dos tablas:

```
SELECT *
FROM empleado JOIN familiar
 ON nss = nssemp;
```

# SELECT. Reunión (JOIN)

33

## EMPLEADO

| nombre     | apellido | nss | ... | dep  |
|------------|----------|-----|-----|------|
| JONÁS      | SOLANO   | 123 | ... | D1   |
| RIGOBERTA  | CALAVERA | 321 | ... | D3   |
| EUSEBIO    | MULETAS  | 222 | ... | D2   |
| MACARENO   | SOSO     | 111 | ... | D1   |
| CASIANA    | FABERGÉ  | 333 | ... | D3   |
| FILOMENA   | RASCAS   | 234 | ... | D1   |
| GUMERSINDA | MIMOS    | 543 | ... | NULL |

## FAMILIAR

| nssemp | numero | nombre    | fechanacim | parentesco |
|--------|--------|-----------|------------|------------|
| 123    | 1      | JONÁS     | 17/05/1992 | HIJO       |
| 321    | 2      | RÓMULA    | 23/09/1923 | ABUELA     |
| 222    | 1      | ELEUTERIO | 30/10/2002 | HIJO       |
| 321    | 1      | RENATA    | 10/03/2002 | HIJA       |
| 111    | 1      | JULIANA   | 10/10/1936 | MADRE      |
| 321    | 3      | TORCUATA  | 17/05/1938 | ABUELA     |
| 111    | 2      | SINFOROSA | 23/09/1947 | ABUELA     |

```
SELECT *
FROM empleado JOIN familiar
ON nss = nssemp;
```

Si un empleado tiene varios familiares, aparece varias veces en el resultado, combinado una vez con cada familiar

| nombre    | apellido | nss | ... | dep | nssemp | numero | nombre    | ... | parentesco |
|-----------|----------|-----|-----|-----|--------|--------|-----------|-----|------------|
| JONAS     | SOLANO   | 123 | ... | D1  | 123    | 1      | JONÁS     | ... | HIJO       |
| RIGOBERTA | CALAVERA | 321 | ... | D3  | 321    | 2      | RÓMULA    | ... | ABUELA     |
| EUSEBIO   | MULETAS  | 222 | ... | D2  | 222    | 1      | ELEUTERIO | ... | HIJO       |
| RIGOBERTA | CALAVERA | 321 | ... | D3  | 321    | 1      | RENATA    | ... | HIJA       |
| MACARENO  | SOSO     | 111 | ... | D1  | 111    | 1      | JULIANA   | ... | MADRE      |
| RIGOBERTA | CALAVERA | 321 | ... | D3  | 321    | 3      | TORCUATA  | ... | ABUELA     |
| MACARENO  | SOSO     | 111 | ... | D1  | 111    | 2      | SINFOROSA | ... | ABUELA     |

Importante: En el resultado del JOIN **faltan** varios empleados (CASIANA por ejemplo) ¿Por qué? ¿Y faltan familiares? ¿Por qué?

# SELECT. Reunión (JOIN)

Recopilamos y formalizamos...

34

- La reunión permite **procesar vínculos entre las tablas**
- **Combina cada dos filas vinculadas**, una de una tabla y otra de la otra tabla, **obteniendo una sola fila** para el resultado
- Para especificar una **reunión** en la sentencia SELECT, hay que indicar los **nombres de las tablas como operandos del JOIN** en la **cláusula FROM**
- Y la **condición de reunión** se incluye en la **cláusula ON**

SELECT *lista\_columnas\_de\_R\_y\_S*

FROM **R JOIN S**

**ON condición\_de\_reunión;**

- Sinónimo: **INNER JOIN** (reunión interna)

# SELECT. Reunión (JOIN)

35

- Acerca de la **condición de reunión** ...
  - Es la que garantiza que se reúnan las filas que están relacionadas entre sí
  - Suele ser la **comparación por igualdad** entre una **clave ajena** y una **clave primaria**
  - No debe incluir otras condiciones de selección de filas de alguna de las tablas (deben estar en la cláusula WHERE)

\* Para cada familiar que sea hijx de empleadx, mostrar su parentesco, el nss del empleadx del cual es familiar, junto con el estado civil y ciudad de dichx empleadx

SELECT parentesco, nss, estado\_civil, ciudad

FROM empleado JOIN familiar → **reunión o join** de dos tablas

ON nss = nssemp → **condición de reunión** entre tablas ◀

WHERE parentesco LIKE 'HIJ\_'; → **condición de selección** de filas de FAMILIAR

# SELECT. Reunión (JOIN)

36

- \* **Para cada departamento, mostrar su código y el apellido de su director**
- Hemos de sacar información de la tabla DEPARTAMENTO, pues ahí están todos los departamentos.
  - Hemos de poner DEPARTAMENTO en la cláusula FROM
- Cada fila de DEPARTAMENTO incluye una referencia a su director: “nssdire” contiene un NSS de empleado, pero no su apellido
- Con ese valor de “nssdire” podemos ir a la tabla EMPLEADO, buscar la fila cuyo “nss” coincide, y ya tenemos los datos del empleado director del departamento (el apellido)
  - También hemos de poner EMPLEADO en el FROM
  - Así que hay que hacer un JOIN entre DEPARTAMENTO y EMPLEADO
  - Y la condición de reunión es nssdire=nss
- Por tanto, esta es la SELECT que hemos de redactar:

```
SELECT coddep, apellido
FROM departamento JOIN empleado
 ON nssdire = nss;
```



# SELECT. Reunión (JOIN)

37

- La condición de reunión puede incluir **más de una comparación por igualdad**, ligadas con AND, si es necesario usar dos o más columnas para combinar correctamente las filas de una tabla con las filas de la otra tabla
  - Esto ocurre cuando la clave ajena y la clave primaria (vínculo entre las tablas) son compuestas (tienen más de una columna)
  - Ejemplo: la clave primaria de EJEMPLAR es (ISBN, numero)
  - Por tanto, la clave ajena en PRESTAMO es (libro, ejemplar)
- \* Para cada préstamo de un ejemplar realizado por el socio 'S03', mostrar el ISBN y número del ejemplar, el estante donde estaba y la fecha del préstamo

SELECT ISBN, numero, estante, fecha

FROM prestamo JOIN ejemplar → **reunión de dos tablas**

ON libro = ISBN → **condición de reunión**  
AND ejemplar = numero **con 2 comparaciones**

WHERE socio = 'S03'; → **condición de selección de filas**

# SELECT. Reunión (JOIN)

38

EJEMPLAR

| ISBN          | numero | estante |
|---------------|--------|---------|
| 9788408217251 | 1      | H4      |
| 9788408217251 | 2      | H4      |
| 9788408217251 | 3      | H5      |
| 9788467009101 | 1      | M2      |
| 9788467009101 | 2      | M2      |
| 9788468319612 | 1      | S1      |
| 9788491223542 | 1      | S2      |
| 9788491223542 | 2      | A3      |
| 9788468315775 | 1      | S7      |

PRESTAMO

| libro         | ejemplar | socio | fecha      | ... |
|---------------|----------|-------|------------|-----|
| 9788408217251 | 1        | S02   | 14/09/2019 |     |
| 9788408217251 | 2        | S03   | 14/09/2019 |     |
| 9788408217251 | 3        | S06   | 15/10/2019 |     |
| 9788467009101 | 1        | S04   | 09/09/2019 |     |
| 9788467009101 | 1        | S04   | 19/10/2019 |     |
| 9788408217251 | 3        | S02   | 20/10/2019 |     |
| 9788491223542 | 2        | S03   | 09/10/2019 |     |
| 9788491223542 | 2        | S01   | 18/11/2019 |     |

```

SELECT libro, numero, estante, fecha
FROM prestamo JOIN ejemplar
ON libro = ISBN
AND ejemplar = numero
WHERE socio = 'S03';

```

Resultado de la consulta

| libro         | numero | estante | fecha      |
|---------------|--------|---------|------------|
| 9788408217251 | 2      | H4      | 14/09/2019 |
| 9788491223542 | 2      | A3      | 09/10/2019 |

# SELECT. Calificación

39

- En SQL los **nombres** de las **columnas** deben ser **únicos dentro de cada tabla**
  - Pero **distintas tablas pueden tener columnas denominadas igual**
    - DEPARTAMENTO y EMPLEADO tienen columnas llamadas “nombre”
    - EMPLEADO y FAMILIAR tienen columnas llamadas “nombre” y “fechanacim”
    - ...

\* *Nombre, apellido de los empleados junto con el nombre de su departamento*  
SELECT **nombre**, apellido, **nombre** -- Error: **columnas ambiguas**  
FROM empleado **JOIN** departamento  
                ON dep = coddep;

- Si en una consulta aparecen varias **columnas de igual nombre, pero de tablas distintas...** ▶ **AMBIGÜEDAD**
  - El SGBD no sabe a qué tabla pertenece cada columna (**ERROR!**)

# SELECT. Calificación

40

- Para resolver la ambigüedad hay que CALIFICAR cada columna ambigua con el nombre de su tabla

```
SELECT empleado.nombre, apellido, departamento.nombre
FROM empleado JOIN departamento
 ON dep = coddep;
```

\* Nombre, apellido y estado civil de los empleados del departamento INVESTIGACION

```
SELECT nombre, apellido, estado_civil
FROM empleado JOIN departamento
 ON dep = coddep
```

WHERE nombre = 'INVESTIGACION'; -- Error: columnas ambiguas

-- Sentencia correcta:

```
SELECT empleado.nombre, apellido, estado_civil
FROM empleado JOIN departamento
 ON dep = coddep
WHERE departamento.nombre = 'INVESTIGACION';
```

# SELECT. Calificación

41

- Se puede utilizar **pseudónimos** (alias) para **acortar los nombres de tabla que se usan para calificar**:

```
SELECT e.nombre, apellido, ciudad
FROM empleado e JOIN departamento d
 ON dep = coddep
WHERE d.nombre = 'INVESTIGACION';
```

El SGBD ejecuta primero el FROM, por lo que conoce los alias de cada tabla al principio de la ejecución

- \* *Nombre, apellido de los empleados junto con el nombre de su departamento*

```
SELECT e.nombre, apellido, d.nombre
FROM empleado e JOIN departamento d
 ON dep = coddep;
```

Sólo hace falta **calificar las columnas ambiguas**. El resto de columnas pueden dejarse sin calificar (aunque si se califican, no hay problema)

- \* *Obtener nombre de cada empleado y de sus familiares*

```
SELECT e.nombre, f.nombre
FROM empleado e JOIN familiar f
 ON e.nss = f.nssemp;
```

# SELECT. Calificación

42

- Si en una SELECT aparece **la misma tabla dos o más veces** seguro que hay AMBIGÜEDAD
  - Solución: CALIFICACIÓN con PSEUDÓNIMOS
- \* Obtener nombre de cada empleado junto con el nombre de su jefe
  - Para cada empleado, obtenemos su nombre y el código de su jefe (columna “nssjefe”)
  - Con el valor de “nssjefe” podemos acceder otra vez a EMPLEADO para buscar ese jefe: la fila que tiene ese código en su columna “nss”
  - Hay que acceder dos veces a EMPLEADO, una para recorrer los empleados, y otra para encontrar al jefe de cada empleado:
    - JOIN entre EMPLEADO (visto como empleado) y EMPLEADO (visto como jefe)

Lo vemos con un ejemplo, mejor  
→ →

# SELECT. Calificación

43

EMPLEADO

| nombre       | apellido | nss | dni | añonacim | direccion   | ciudad  | salario | nssjefe | dep  |
|--------------|----------|-----|-----|----------|-------------|---------|---------|---------|------|
| e → JONÁS    | SOLANO   | 123 | 11A | 1945     | C/PEZ, 10   | MURCIA  | 1100    | 111     | D1   |
| RIGOBERTA    | CALAVERA | 321 | 21C | 1974     | C/BOJ, 2    | YECLA   | 900     | 333     | D3   |
| EUSEBIO      | MULETAS  | 222 | 22B | 1969     | C/RIF, 6    | TOTANA  | 2100    | 123     | D2   |
| j → MACARENO | SOSO     | 111 | 23D | 1944     | C/MAR, 4    | JUMILLA | 1100    | NULL    | D1   |
| CASIANA      | FABERGÉ  | 333 | 33B | 1943     | C/SOL, 8    | MURCIA  | 920     | 123     | D3   |
| FILOMENA     | RASCAS   | 234 | 34E | 1970     | C/NUEZ, 3   | MURCIA  | 1100    | 111     | D1   |
| GUMERSINDA   | MIMOS    | 543 | 45F | 1980     | C/QUINTO, 5 | PINOSO  | 850     | NULL    | NULL |

SELECT e.nombre, j.nombre  
 FROM empleado e JOIN empleado j  
 ON e.nssjefe = j.nss;

| e.nombre  | j.nombre |
|-----------|----------|
| JONÁS     | MACARENO |
| RIGOBERTA | CASIANA  |
| EUSEBIO   | JONÁS    |
| CASIANA   | JONÁS    |
| FILOMENA  | MACARENO |

# SELECT. Renombrar columnas

44

- Es posible dar **nuevos nombres** para las **columnas** del **resultado** de la consulta
  - \* Nombres de cada empleado y su jefe, cambiando al mismo tiempo los nombres de las columnas resultantes a 'empleado' y 'jefe'

```
SELECT e.nombre empleado, j.nombre jefe
FROM empleado e JOIN empleado j
 ON e.nssjefe = j.nss;
```

Basta con **separar el nombre de la columna y el nuevo nombre** con un **espacio** en blanco

Resultado SIN los nuevos nombres

| e.nombre  | j.nombre |
|-----------|----------|
| JONÁS     | MACARENO |
| RIGOBERTA | CASIANA  |
| EUSEBIO   | JONÁS    |
| CASIANA   | JONÁS    |
| FILOMENA  | MACARENO |

Resultado CON los nuevos nombres

| empleado  | jefe     |
|-----------|----------|
| JONÁS     | MACARENO |
| RIGOBERTA | CASIANA  |
| EUSEBIO   | JONÁS    |
| CASIANA   | JONÁS    |
| FILOMENA  | MACARENO |

- Es una nueva cabecera para la tabla resultado

# SELECT. Varios JOIN ON

45

- Es posible incluir **varios operadores JOIN ON**, para realizar reuniones entre más de dos tablas

\* Para cada familiar 'ABUELA' o 'MADRE', mostrar tal parentesco, el apellido del empleado del que depende, y el nss del director del departamento al que pertenece el/la empleado.

```
SELECT parentesco, apellido, nssdire
FROM familiar JOIN empleado
 ON nssemp = nss
 JOIN departamento
 ON dep = coddep
```

```
WHERE parentesco IN ('ABUELA', 'MADRE');
```

El 1<sup>er</sup> JOIN obtiene una fila con los datos de cada familiar y los de su empleado

El 2º JOIN obtiene una fila para cada familiar con todos sus datos, los de su empleado y los del departamento al que pertenece el empleado

El WHERE selecciona sólo aquellas filas correspondientes a familiares con parentesco ABUELA o MADRE

# SELECT. Varios JOIN ON

46

\* Empleados (*dni* y *nombre*) del departamento dirigido por 'MACARENO SOSO'.

```
SELECT e.dni, e.nombre
FROM departamento d
 JOIN empleado j ON nssdire = j.nss
 JOIN empleado e ON e.dep = coddep
WHERE j.nombre = 'MACARENO'
 AND j.apellido = 'SOSO';
```

El WHERE selecciona sólo aquellas filas correspondientes a los empleados cuyo departamento tiene un director llamado MACARENO SOSO

El 1er JOIN obtiene una fila con los datos de cada departamento y los de su empleado director

El 2º JOIN obtiene una fila para cada empleado, su departamento y el empleado director del departamento

# SELECT. Notación CLÁSICA del JOIN

47

- Las primeras versiones del estándar ANSI SQL no incluían el operador JOIN ON
- La forma (clásica) de especificar una **reunión** era indicar los nombres de las **tablas separados por comas en la cláusula FROM**
- E incluir la **condición de reunión** en la **cláusula WHERE**

```
SELECT lista_columnas_de_R_y_S
FROM R, S
WHERE condición_de_reunión;
```

- Es posible encontrar muchos ejemplos de sentencias con JOIN redactadas con esta notación, así que debemos conocerla

# SELECT. Notación CLÁSICA del JOIN

48

## □ Especificación clásica, en las cláusulas FROM y WHERE

\* Nombre y ciudad de cada empleado del departamento de Investigación

SELECT e.nombre, apellido, ciudad

FROM empleado e, departamento d

← reunión de tablas

WHERE dep = coddep

← condición de reunión

AND d.nombre = 'INVESTIGACION';

← condición de selección

## □ Especificación mediante JOIN ON

\* Nombre, apellido y ciudad de cada empleado del departamento de Investigación

SELECT e.nombre, apellido, ciudad

FROM empleado e **JOIN** departamento d  
ON dep = coddep

WHERE d.nombre = 'INVESTIGACION'; ← condición de selección

## □ Como vemos, con JOIN ON las consultas son más comprensibles: condiciones de reunión de tablas separadas de las condiciones de selección de filas

# SELECT. Notación CLÁSICA del JOIN

49

- Con esta notación, para especificar una **reunión** entre **más de dos tablas**, es necesario incluir tantas condiciones de reunión como (número de tablas - 1) haya en la cláusula FROM
  
- \* Para cada familiar con parentesco 'ABUELA' o 'MADRE', mostrar dicho parentesco, el apellido del empleadx del que depende, y el nss del director del departamento al que pertenece el/la empleadx.

```
SELECT parentesco, apellido, nssdir
FROM FAMILIAR, EMPLEADO, DEPARTAMENTO → reunión de TRES tablas
WHERE nssemp = nss → condición de reunión entre FAMILIAR y EMPLEADO ◀
 AND dep = coddep → condición de reunión entre EMPLEADO y DEPARTAMENTO ◀
 AND parentesco IN ('ABUELA', 'MADRE'); → condición de selección
```

# SELECT. Omisión de WHERE

50

- Selección incondicional
  - ▣ **No incluir WHERE** equivale a una condición TRUE para todas las filas
- Si en la cláusula FROM aparece **sólo una tabla**, se obtienen todas las filas de dicha tabla
  - \* *Seleccionar los nss de todxs lxs empleadxs*

```
SELECT nss
FROM empleado;
```

- Si el FROM incluye **más de una tabla**, se obtiene el **producto cartesiano** entre dichas tablas

- ▣ Se combina **cada fila** de una tabla con **todas las filas** de la otra

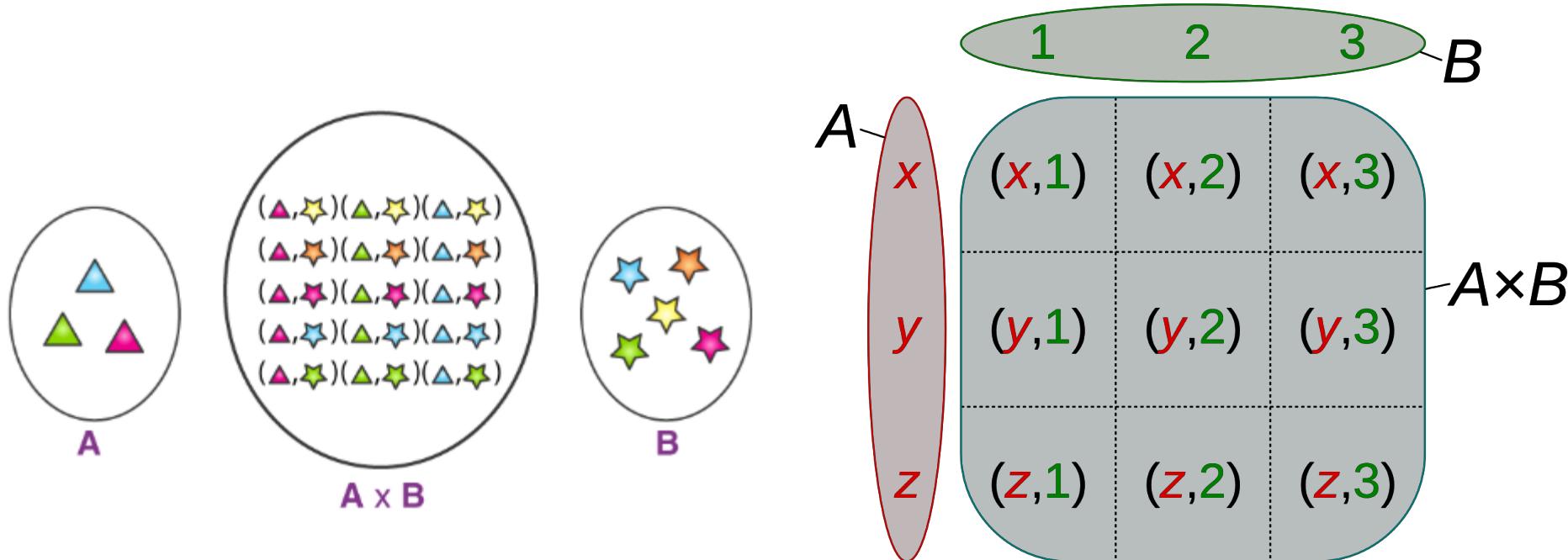
- \* *Obtener todas las combinaciones de los datos de empleadxs con los de departamentos*

```
SELECT *
FROM empleado, departamento;
```



# Inciso: Producto Cartesiano

51



Otro ejemplo, dados los conjuntos:

$$R = \{ 1, 2, 3, 4 \} \quad y \quad S = \{ a, b \}$$

El producto cartesiano de R por S es:

$$R \times S = \{ (1,a), (1,b), (2,a), (2,b), (3,a), (3,b), (4,a), (4,b) \}$$

# SELECT. Reunión sin WHERE

52

```
SELECT *
FROM empleado, departamento;
```

| nombre       | apellido | nss | dni | añonacim   | direccion   | ciudad  | salario | nssiefe | dep  | nombre         | coddep | nssdire |
|--------------|----------|-----|-----|------------|-------------|---------|---------|---------|------|----------------|--------|---------|
| JONÁS        | SOLANO   | 123 | 11A | 10/10/1945 | C/PEZ, 10   | MURCIA  | 1100    | 111     | D1   | ADMINISTRACION | D1     | 111     |
| RIGOBERTA    | CALAVERA | 321 | 21C | 12/11/1974 | C/BOJ, 2    | YECLA   | 900     | 333     | D3   | ADMINISTRACION | D1     | 111     |
| EUSEBIO      | MULETAS  | 222 | 22B | 01/01/1969 | C/RIF, 6    | TOTANA  | 2100    | 123     | D2   | ADMINISTRACION | D1     | 111     |
| MACARENO     | SOSO     | 111 | 23D | 06/04/1944 | C/MAR, 4    | JUMILLA | 1100    | NULL    | D1   | ADMINISTRACION | D1     | 111     |
| CASIANA      | FABERGÉ  | 333 | 33B | 15/06/1943 | C/SOL, 8    | MURCIA  | 920     | 123     | D3   | ADMINISTRACION | D1     | 111     |
| FILOMENA     | RASCAS   | 234 | 34E | 18/07/1970 | C/NUEZ, 3   | MURCIA  | 1100    | 111     | D1   | ADMINISTRACION | D1     | 111     |
| GUMERSINDA   | MIMOS    | 543 | 45F | 10/02/1980 | C/QUINTO, 5 | PINOSO  | 850     | NULL    | NULL | ADMINISTRACION | D1     | 111     |
| JONÁS        | SOLANO   | 123 | 11A | 10/10/1945 | C/PEZ, 10   | MURCIA  | 1100    | 111     | D1   | INVESTIGACION  | D2     | 222     |
| RIGOBERTA    | CALAVERA | 321 | 21C | 12/11/1974 | C/BOJ, 2    | YECLA   | 900     | 333     | D3   | INVESTIGACION  | D2     | 222     |
| EUSEBIO      | MULETAS  | 222 | 22B | 01/01/1969 | C/RIF, 6    | TOTANA  | 2100    | 123     | D2   | INVESTIGACION  | D2     | 222     |
| MACARENO     | SOSO     | 111 | 23D | 06/04/1944 | C/MAR, 4    | JUMILLA | 1100    | NULL    | D1   | INVESTIGACION  | D2     | 222     |
| CASIANA      | FABERGÉ  | 333 | 33B | 15/06/1943 | C/SOL, 8    | MURCIA  | 920     | 123     | D3   | INVESTIGACION  | D2     | 222     |
| FILOMENA     | RASCAS   | 234 | 34E | 18/07/1970 | C/NUEZ, 3   | MURCIA  | 1100    | 111     | D1   | INVESTIGACION  | D2     | 222     |
| GUMERSINDA   | MIMOS    | 543 | 45F | 10/02/1980 | C/QUINTO, 5 | PINOSO  | 850     | NULL    | NULL | INVESTIGACION  | D2     | 222     |
| ... sigue... |          |     |     |            |             |         |         |         |      |                |        |         |

ⓘ En el resultado SÍ ESTÁ la empleada GUMERSINDA MIMOS, y el departamento TRAINING

# SELECT. Reunión sin WHERE

53

```
SELECT *
FROM empleado, departamento;
```

| nombre       | apellido | nss | dni | añonacim   | direccion   | ciudad  | salario | nssjefe | dep  | nombre   | coddep | nssdire |
|--------------|----------|-----|-----|------------|-------------|---------|---------|---------|------|----------|--------|---------|
| ... sigue... |          |     |     |            |             |         |         |         |      |          |        |         |
| JONÁS        | SOLANO   | 123 | 11A | 10/10/1945 | C/PEZ, 10   | MURCIA  | 1100    | 111     | D1   | PERSONAL | D3     | 333     |
| RIGOBERTA    | CALAVERA | 321 | 21C | 12/11/1974 | C/BOJ, 2    | YECLA   | 900     | 333     | D3   | PERSONAL | D3     | 333     |
| EUSEBIO      | MULETAS  | 222 | 22B | 01/01/1969 | C/RIF, 6    | TOTANA  | 2100    | 123     | D2   | PERSONAL | D3     | 333     |
| MACARENO     | SOSO     | 111 | 23D | 06/04/1944 | C/MAR, 4    | JUMILLA | 1100    | NULL    | D1   | PERSONAL | D3     | 333     |
| CASIANA      | FABERGÉ  | 333 | 33B | 15/06/1943 | C/SOL, 8    | MURCIA  | 920     | 123     | D3   | PERSONAL | D3     | 333     |
| FILOMENA     | RASCAS   | 234 | 34E | 18/07/1970 | C/NUEZ, 3   | MURCIA  | 1100    | 111     | D1   | PERSONAL | D3     | 333     |
| GUMERSINDA   | MIMOS    | 543 | 45F | 10/02/1980 | C/QUINTO, 5 | PINOSO  | 850     | NULL    | NULL | PERSONAL | D3     | 333     |
| JONÁS        | SOLANO   | 123 | 11A | 10/10/1945 | C/PEZ, 10   | MURCIA  | 1100    | 111     | D1   | TRAINING | D4     | NULL    |
| RIGOBERTA    | CALAVERA | 321 | 21C | 12/11/1974 | C/BOJ, 2    | YECLA   | 900     | 333     | D3   | TRAINING | D4     | NULL    |
| EUSEBIO      | MULETAS  | 222 | 22B | 01/01/1969 | C/RIF, 6    | TOTANA  | 2100    | 123     | D2   | TRAINING | D4     | NULL    |
| MACARENO     | SOSO     | 111 | 23D | 06/04/1944 | C/MAR, 4    | JUMILLA | 1100    | NULL    | D1   | TRAINING | D4     | NULL    |
| CASIANA      | FABERGÉ  | 333 | 33B | 15/06/1943 | C/SOL, 8    | MURCIA  | 920     | 123     | D3   | TRAINING | D4     | NULL    |
| FILOMENA     | RASCAS   | 234 | 34E | 18/07/1970 | C/NUEZ, 3   | MURCIA  | 1100    | 111     | D1   | TRAINING | D4     | NULL    |
| GUMERSINDA   | MIMOS    | 543 | 45F | 10/02/1980 | C/QUINTO, 5 | PINOSO  | 850     | NULL    | NULL | TRAINING | D4     | NULL    |

ⓘ En el resultado SÍ ESTÁ la empleada GUMERSINDA MIMOS, y el departamento TRAINING

# SELECT. Reunión Natural y Externa

54

- El operador JOIN ON, además de permitir especificar una reunión “interna”, que ya hemos visto, también permite especificar reuniones de otros tipos:
  - **Reunión Natural**
  - **Reunión Externa**

# SELECT. Reunión Natural

55

- Formato

SELECT ...

FROM R1 **NATURAL JOIN** R2

WHERE ...

- NO necesita condición de reunión
  - No incluye la cláusula **ON condición**
- El SGBD asume una condición de reunión para cada par de **columnas con igual nombre** en una y otra tabla
- Y sólo se incluye **una** de estas columnas en el resultado
  - Elimina una de las columnas idénticas
- Veámoslo con un ejemplo...

# SELECT. Reunión Natural

56

- Sea una nueva tabla

**OFICINA(coddep, oficina, ubicacion)**

que almacena las distintas oficinas de cada departamento,  
 cuya columna **coddep** es una clave ajena  
 que referencia a **DEPARTAMENTO(coddep)**

**OFICINA**

| <u>coddep</u> | oficina | ubicacion |
|---------------|---------|-----------|
| D1            | O1      | MURCIA    |
| D3            | O1      | MADRID    |
| D2            | O1      | ALICANTE  |
| D1            | O3      | MAZARRON  |
| D3            | O2      | ALICANTE  |
| D1            | O2      | ALBACETE  |

**DEPARTAMENTO**

| nombre         | <u>coddep</u> | nssdire |
|----------------|---------------|---------|
| INVESTIGACION  | D2            | 222     |
| ADMINISTRACION | D1            | 111     |
| PERSONAL       | D3            | 333     |
| TRAINING       | D4            | NULL    |

# SELECT. Reunión Natural

57

- Si se desea obtener la información de cada oficina y de su departamento, hay que realizar un JOIN entre ambas tablas

- Puesto que las columnas de reunión (clave primaria de DEPARTAMENTO y clave ajena de OFICINA) se llaman igual (coddep), es posible realizar una reunión natural
  - Si ejecutamos esta sentencia:

```
SELECT *
FROM oficina NATURAL JOIN departamento;
```

- Lo que asume y ejecuta el SGBD es esto:

```
SELECT *
FROM oficina o JOIN departamento d
ON o.coddep = d.coddep;
```

# SELECT. Reunión Natural

58

**OFICINA**

| <u>coddep</u> | oficina | ubicacion |
|---------------|---------|-----------|
| D1            | O1      | MURCIA    |
| D3            | O1      | MADRID    |
| D2            | O1      | ALICANTE  |
| D1            | O3      | TOLEDO    |
| D3            | O2      | ALICANTE  |
| D1            | O2      | ALBACETE  |

**DEPARTAMENTO**

| nombre         | <u>coddep</u> | nssdire |
|----------------|---------------|---------|
| INVESTIGACION  | D2            | 222     |
| ADMINISTRACION | D1            | 111     |
| PERSONAL       | D3            | 333     |
| TRAINING       | D4            | NULL    |

oficina **NATURAL JOIN** departamento

| <u>coddep</u> | oficina | ubicacion | nombre         | nssdire |
|---------------|---------|-----------|----------------|---------|
| D1            | O1      | MURCIA    | ADMINISTRACION | 111     |
| D3            | O1      | MADRID    | PERSONAL       | 333     |
| D2            | O1      | ALICANTE  | INVESTIGACION  | 222     |
| D1            | O3      | TOLEDO    | ADMINISTRACION | 111     |
| D3            | O2      | ALICANTE  | PERSONAL       | 333     |
| D1            | O2      | ALBACETE  | ADMINISTRACION | 111     |

oficina o **JOIN** departamento d  
ON o.coddep = d.coddep

| <u>o.coddep</u> | oficina | ubicacion | <u>d.coddep</u> | nombre         | nssdire |
|-----------------|---------|-----------|-----------------|----------------|---------|
| D1              | O1      | MURCIA    | D1              | ADMINISTRACION | 111     |
| D3              | O1      | MADRID    | D3              | PERSONAL       | 333     |
| D2              | O1      | ALICANTE  | D2              | INVESTIGACION  | 222     |
| D1              | O3      | TOLEDO    | D1              | ADMINISTRACION | 111     |
| D3              | O2      | ALICANTE  | D3              | PERSONAL       | 333     |
| D1              | O2      | ALBACETE  | D1              | ADMINISTRACION | 111     |



Columnas idénticas

# SELECT. Reunión Natural

59

- La reunión natural sólo devuelve **una** de las dos columnas de reunión
  - Relación resultado: (**coddep**, oficina, ubicacion, nombre, nssdire)
- Si se usan alias para las tablas, **no** se puede calificar una *columna de reunión fuera del natural join*
  - Una vez realizado el natural join, sólo hay una columna en el resultado (no dos), ya no se puede anteponer el nombre de una de las tablas...
  - Es decir, esto daría ERROR:

```
SELECT *
```

```
FROM (departamento d NATURAL JOIN oficina o)
```

```
WHERE d.coddep <> 'D1';
```

► **No es necesario calificar** la columna “coddep” (isólo hay una!)

-- Se debe quitar el prefijo **d.** ... y ifuncionará!



# SELECT. Reunión Externa

60

- Necesaria si en una reunión se necesita obtener las filas...
  - con valor **NONE** en **las columnas de reunión**, o
  - **sin correspondencia** en la otra tabla
- Tipos de reunión externa:
  - **LEFT [OUTER] JOIN**      **Reunión externa izquierda**
  - **RIGHT [OUTER] JOIN**      **Reunión externa derecha**
  - **FULL [OUTER] JOIN**      **Reunión externa total** o completa
- Las cláusulas LEFT, RIGHT y FULL indican *dónde está situada la tabla cuyas filas deben aparecer, todas ellas, en el resultado*

# SELECT. Reunión Externa

61

- Veamos los tipos de reunión externa con un ejemplo

**EMPLEADO**

| nombre     | apellido | nss | ... | dep         |
|------------|----------|-----|-----|-------------|
| JONÁS      | SOLANO   | 123 | ... | D1          |
| RIGOBERTA  | CALAVERA | 321 | ... | D3          |
| EUSEBIO    | MULETAS  | 222 | ... | D2          |
| MACARENO   | SOSO     | 111 | ... | D1          |
| CASIANA    | FABERGÉ  | 333 | ... | D3          |
| FILOMENA   | RASCAS   | 234 | ... | D1          |
| GUMERSINDA | MIMOS    | 543 | ... | <b>NULL</b> |

→ Esta fila no tiene correspondencia en la otra tabla, porque tiene NULL en la columna clave ajena

**DEPARTAMENTO**

| nombre         | coddep    | nssdire |
|----------------|-----------|---------|
| INVESTIGACION  | D2        | 222     |
| ADMINISTRACION | D1        | 111     |
| PERSONAL       | D3        | 333     |
| TRAINING       | <b>D4</b> | NULL    |



← Esta fila no tiene correspondencia en la otra tabla, porque ninguna fila en EMPLEADO le hace referencia (ningún empleado tiene 'D4' en la columna "dep")

# SELECT. Reunión Externa

62

- Punto de partida:

\* *Datos de cada empleado y de su departamento*

`SELECT *`

`FROM (empleado e JOIN departamento d  
ON dep = coddep);`

- Es una reunión “interna”
- En el resultado no aparecen los empleados sin departamento, ni los departamentos a los que no pertenece ningún empleado
  - No están la empleada 543 ni el departamento D4

| e.nombre  | apellido | nss | ... | dep | d.nombre       | coddep | nssdire |
|-----------|----------|-----|-----|-----|----------------|--------|---------|
| JONÁS     | SOLANO   | 123 | ... | D1  | ADMINISTRACION | D1     | 111     |
| RIGOBERTA | CALAVERA | 321 | ... | D3  | PERSONAL       | D3     | 333     |
| EUSEBIO   | MULETAS  | 222 | ... | D2  | INVESTIGACION  | D2     | 222     |
| MACARENO  | SOSO     | 111 | ... | D1  | ADMINISTRACION | D1     | 111     |
| CASIANA   | FABERGÉ  | 333 | ... | D3  | PERSONAL       | D3     | 333     |
| FILOMENA  | RASCAS   | 234 | ... | D1  | ADMINISTRACION | D1     | 111     |

# SELECT. Reunión Externa

63

\* Datos de empleados y departamentos *incluyendo empleados sin departamento*

```
SELECT *
FROM (empleado e LEFT JOIN departamento d
 ON dep = coddep);
```

- Es una reunión externa **izquierda**
- EMPLEADO está a la izquierda del JOIN, por lo que en el resultado sí aparecen los empleados sin departamento
  - Con las columnas correspondientes a DEPARTAMENTO llenas con NULL
- Pero no están los departamentos a los que no pertenece ningún empleado (departamento D4)

| e.nombre   | apellido | nss | ... | dep  | d.nombre       | coddep | nssdire |
|------------|----------|-----|-----|------|----------------|--------|---------|
| JONÁS      | SOLANO   | 123 | ... | D1   | ADMINISTRACION | D1     | 111     |
| RIGOBERTA  | CALAVERA | 321 | ... | D3   | PERSONAL       | D3     | 333     |
| EUSEBIO    | MULETAS  | 222 | ... | D2   | INVESTIGACION  | D2     | 222     |
| MACARENO   | SOSO     | 111 | ... | D1   | ADMINISTRACION | D1     | 111     |
| CASIANA    | FABERGÉ  | 333 | ... | D3   | PERSONAL       | D3     | 333     |
| FILOMENA   | RASCAS   | 234 | ... | D1   | ADMINISTRACION | D1     | 111     |
| GUMERSINDA | MIMOS    | 543 | ... | NULL | NULL           | NULL   | NULL    |



# SELECT. Reunión Externa

64

\* Datos de empleados y departamentos *incluyendo departamentos sin empleados*

```
SELECT *
FROM (empleado e RIGHT JOIN departamento d
 ON dep = coddep);
```

- Es una reunión externa **derecha**
- DEPARTAMENTO está a la derecha del JOIN, por lo que en el resultado sí aparecen los departamentos sin empleados
  - Con las columnas correspondientes a EMPLEADO rellenas con NULL
- Pero no están los empleados sin departamento (empleada 543)

| e.nombre  | apellido | nss  | ... | dep  | d.nombre       | coddep | nssdire |
|-----------|----------|------|-----|------|----------------|--------|---------|
| JONÁS     | SOLANO   | 123  | ... | D1   | ADMINISTRACION | D1     | 111     |
| RIGOBERTA | CALAVERA | 321  | ... | D3   | PERSONAL       | D3     | 333     |
| EUSEBIO   | MULETAS  | 222  | ... | D2   | INVESTIGACION  | D2     | 222     |
| MACARENO  | SOSO     | 111  | ... | D1   | ADMINISTRACION | D1     | 111     |
| CASIANA   | FABERGÉ  | 333  | ... | D3   | PERSONAL       | D3     | 333     |
| FILOMENA  | RASCAS   | 234  | ... | D1   | ADMINISTRACION | D1     | 111     |
| NULL      | NULL     | NULL |     | NULL | TRAINING       | D4     | NULL    |



# SELECT. Reunión Externa

65

\* Datos de empleados y sus departamentos, *incluyendo los departamentos sin empleados y los empleados sin departamento*

```
SELECT *
FROM (empleado e FULL JOIN departamento d
 ON dep = coddep);
```

- Es una reunión externa **total**
- En el resultado aparecen tanto los departamentos sin empleados, como los empleados sin departamento
  - Con las columnas correspondientes a la otra tabla llenadas con NULL

| e.nombre   | apellido | nss  | ... | dep  | d.nombre       | coddep | nssdire |
|------------|----------|------|-----|------|----------------|--------|---------|
| JONÁS      | SOLANO   | 123  | ... | D1   | ADMINISTRACION | D1     | 111     |
| RIGOBERTA  | CALAVERA | 321  | ... | D3   | PERSONAL       | D3     | 333     |
| EUSEBIO    | MULETAS  | 222  | ... | D2   | INVESTIGACION  | D2     | 222     |
| MACARENO   | SOSO     | 111  | ... | D1   | ADMINISTRACION | D1     | 111     |
| CASIANA    | FABERGÉ  | 333  | ... | D3   | PERSONAL       | D3     | 333     |
| FILOMENA   | RASCAS   | 234  | ... | D1   | ADMINISTRACION | D1     | 111     |
| GUMERSINDA | MIMOS    | 543  | ... | NULL | NULL           | NULL   | NULL    |
| NULL       | NULL     | NULL |     | NULL | TRAINING       | D4     | NULL    |

# SELECT. Reunión Externa

66

## □ Función COALESCE()

- Permite mostrar el **valor** que se desee en cierta columna del resultado de una consulta, **en lugar de un NULL**
- Suele usarse con dos parámetros: **COALESCE(columna, valor)**
  - “columna” corresponde a una columna de alguna de las tablas del FROM
  - “valor” debe ser del mismo tipo de datos que la “columna” (1<sup>er</sup> parámetro)

SELECT nombre, apellido, **COALESCE(dep, 'SIN DEPARTAMENTO')**  
FROM empleado;

- Muestra para cada empleado, su nombre y apellido, y el código del departamento al que pertenece, pero para aquel empleado con NULL en la columna “dep” (que es CHAR), mostraría la cadena ‘SIN DEPARTAMENTO’

SELECT nss, nombre, **COALESCE(nssjefe, 0)**  
FROM empleado;

- Muestra para cada empleado, su nss, nombre y el nss de su jefe, pero para cada empleado con NULL en la columna “nssjefe” (que es NUMBER), mostraría un cero

# SELECT. Reunión Externa

67

## □ Función COALESCE()

- Si se desea mostrar una cadena de caracteres cuando hay NULL en una columna de tipo NUMBER, hay que pasarle el primer argumento convertido a caracteres, usando la función TO\_CHAR de Oracle SQL:

```
SELECT nss, nombre, COALESCE(nssjefe, 'SIN JEFE')
FROM empleado;
```

Da error “INVALID NUMBER”,  
porque “nssjefe” es  
NUMBER y el 2º argumento  
es CHAR

```
SELECT nss, nombre, COALESCE(TO_CHAR(nssjefe), 'SIN JEFE')
FROM empleado;
```



### ■ Se suele usar con las reuniones externas:

\* Nombre de cada empleado y de su departamento; deben aparecer todos los empleados; para los que no tienen departamento, mostrar ‘SIN DEPARTAMENTO’

```
SELECT e.nombre, COALESCE(d.nombre, 'SIN DEPARTAMENTO')
FROM (empleado e LEFT JOIN departamento d
 ON dep=coddep);
```

# SELECT. Tablas como conjuntos

68

## □ Operaciones de conjuntos: **UNION, INTERSECT, MINUS**

- Es posible realizar la unión, intersección o resta entre dos tablas (entre dos conjuntos de filas, en realidad)
- El resultado es un conjunto de filas (es una tabla)
- Importante: **las filas repetidas se eliminan, y se deja sólo una de ellas en la tabla resultado**
- Las tablas operando han de ser **compatibles en tipo:**
  - igual número de columnas, y
  - columnas “correspondientes” (en la misma posición) deben tener el mismo tipo de datos
  - Es imposible hacer la unión entre EMPLEADO y DEPARTAMENTO, o entre EMPLEADO y FAMILIAR, porque el *resultado no sería un conjunto de filas, todas ellas con la misma estructura* (pues tienen distintas columnas)

# SELECT. Tablas como conjuntos

69

## □ Operación UNION

- La **unión** entre dos tablas obtiene una tabla con todas las filas de las dos tablas de origen (y sin duplicados)
- Como el resultado de una SELECT es una tabla, es posible hacer la UNION entre dos consultas (entre dos SELECT)

\* Empleados veteranos (nacidos antes de 1965), o con familiares

abuelos a su cargo

(SELECT nss

FROM empleado

WHERE fechanacim < TO\_DATE('01/01/1965', 'dd/mm/yyyy'))

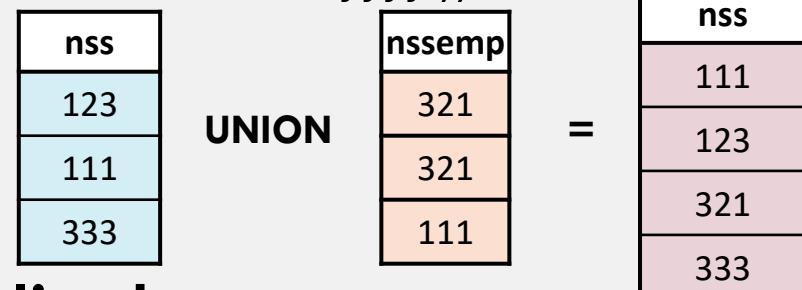
**UNION**

(SELECT nssemp

FROM familiar

WHERE parentesco LIKE 'ABUEL\_');

No hace falta que las columnas se  
llamen igual, pero sí deben tener un  
tipo de datos compatible



- Si no se desea eliminar duplicados...

- Usar la cláusula ALL: operación **UNION ALL**

# SELECT. Tablas como conjuntos

70

## □ Operación INTERSECT

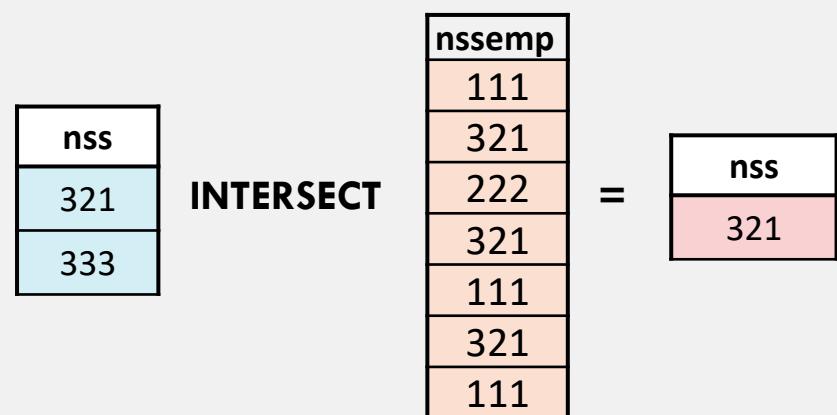
- La **intersección** entre dos tablas obtiene una tabla con las filas que están a la vez en las dos tablas de origen (y sin duplicados)
- Como el resultado de una SELECT es una tabla, es posible hacer la **INTERSECCIÓN** entre dos consultas (entre dos SELECT)

\* *Empleados del departamento 'D3' y que tienen familiares*

```
(SELECT nss
 FROM empleado
 WHERE dep = 'D3')
```

**INTERSECT**

```
(SELECT nssemp
 FROM familiar);
```



# SELECT. Tablas como conjuntos

71

## □ Operación MINUS

- La **resta** entre dos tablas obtiene una tabla con las filas que están en la tabla de la izquierda y que no están en la tabla de la derecha (y sin duplicados)
- Como el resultado de una SELECT es una tabla, es posible hacer la RESTA entre dos consultas (entre dos SELECT)

\* Empleados que cobran menos de 950€ y que no tienen hijos

(SELECT nss  
FROM empleado  
WHERE salario < 950)

**MINUS**

(SELECT nssemp  
FROM familiar  
WHERE parentesco LIKE 'HIJ\_');

| nss |
|-----|
| 321 |
| 333 |
| 543 |

MINUS

| nssemp |
|--------|
| 111    |
| 222    |
| 321    |

=

| nss |
|-----|
| 333 |
| 543 |

En ANSI SQL es EXCEPT y no MINUS

# SELECT. Consultas anidadas

72

- Algunas consultas requieren obtener o calcular valores de la base de datos para ser usados en una condición de comparación: se necesita realizar una consulta “antes” o “dentro” de otra
- **Una consulta anidada es una consulta SELECT completa, dentro de cláusula WHERE de otra consulta** (consulta exterior)
- Obtiene **valores de la BD que se usan en la condición de otra consulta**, para obtener otros datos

\* Familiares de los empleados que son directores de departamento  
SELECT nombre  
FROM familiar

WHERE nssemp **IN** (SELECT nssdire  
FROM departamento);

Otro uso del operador **IN**

Primero se obtienen los códigos de los empleados que son directores de los departamentos

Después se usa dicho código para seleccionar a sus familiares: los que tienen ese valor en su columna “nssemp”

# SELECT. Consultas anidadas

73

## □ Operador IN (también NOT IN) -- otro uso del operador t IN S

– Indica **si la fila t pertenece al conjunto de filas S** (subconsulta)

\* Nombre y estado civil de los empleados que tienen algún familiar.

```
SELECT nombre, est_civil
FROM empleado
```

```
WHERE nss IN (SELECT nssemp
 FROM familiar);
```

1º se obtienen los nss de los empleados que tienen familiares

2º se seleccionan los empleados cuyo nss está entre los obtenidos

\* Dni, nombre y ciudad de los empleados que son jefes de departamento.

```
SELECT dni, nombre, ciudad
FROM empleado
```

```
WHERE nss IN (SELECT nssdire
 FROM departamento);
```

1º se obtienen los nss de los empleados que dirigen los departamentos

2º se seleccionan los empleados cuyo nss está entre los obtenidos

\* Nombre y fecha de nacimiento de los familiares de empleados del departamento 'D1'.

```
SELECT nombre, fechanacim
FROM familiar
```

```
WHERE nssemp IN (SELECT nss
 FROM empleado
 WHERE dep='D1');
```

1º se obtienen los nss de los empleados del departamento 'D1'

2º se seleccionan los familiares de dichos empleados

# SELECT. Consultas anidadas

74

\* Familiares de empleadxs que son directores de departamento

```
SELECT nombre FROM familiar
WHERE nssemp IN (SELECT nssdire
 FROM departamento);
```

Primero se evalúa la subconsulta

| nssdire |
|---------|
| 222     |
| 111     |
| 333     |
| NULL    |

Después se seleccionan las filas de FAMILIAR cuyo valor de columna “nssemp” está entre los obtenidos en la subconsulta

Veamos con detalle el modelo de ejecución de este tipo de consultas...

| nssemp | numero | nombre    | fechanacim | parentesco |
|--------|--------|-----------|------------|------------|
| 123    | 1      | JONÁS     | 17/05/1992 | HIJO       |
| 321    | 2      | RÓMULA    | 23/09/1923 | ABUELA     |
| 222    | 1      | ELEUTERIO | 30/10/2002 | HIJO       |
| 321    | 1      | RENATA    | 10/03/2002 | HIJA       |
| 111    | 2      | JULIANA   | 10/10/1936 | MADRE      |
| 321    | 3      | TORCUATA  | 17/05/1938 | ABUELA     |
| 111    | 3      | SINFOROSA | 23/09/1947 | ABUELA     |

Por último, se proyecta en la columna indicada

| nombre    |
|-----------|
| ELEUTERIO |
| JULIANA   |
| SINFOROSA |

# SELECT. Consultas anidadas

75

\* Empleados (nss y nombre) del departamento de Investigación

```
SELECT nss, nombre
```

```
FROM empleado
```

```
WHERE dep IN (SELECT coddep FROM departamento
 WHERE nombre = 'INVESTIGACION');
```

Primero se evalúa la subconsulta

| coddep |
|--------|
| D2     |

| nombre     | apellido | nss | ... | nssjefe | dep  |
|------------|----------|-----|-----|---------|------|
| JONÁS      | SOLANO   | 123 |     | 111     | D1   |
| RIGOBERTA  | CALAVERA | 321 |     | 333     | D3   |
| EUSEBIO    | MULETAS  | 222 |     | 123     | D2   |
| MACARENO   | SOSO     | 111 |     |         | D1   |
| CASIANA    | FABERGÉ  | 333 |     | 123     | D3   |
| FILOMENA   | RASCAS   | 234 |     | 111     | D1   |
| GUMERSINDA | MIMOS    | 543 |     | NULL    | NULL |

| nss | nombre  |
|-----|---------|
| 222 | EUSEBIO |

Por último, se proyecta en las columnas indicadas

# SELECT. Consultas anidadas

76

\* Familiares de la empleada ‘RIGOBERTA CALAVERA’

SELECT nombre FROM familiar

WHERE nssemp IN (SELECT nss FROM empleado  
 WHERE nombre=‘RIGOBERTA’  
 AND apellido =‘CALAVERA’);

Primero se evalúa la subconsulta

| nss |
|-----|
| 321 |

Después se seleccionan las filas de FAMILIAR cuyo valor de columna “nssemp” está entre los obtenidos en la subconsulta

| nssemp | numero | nombre    | fechanacim | parentesco |
|--------|--------|-----------|------------|------------|
| 111    | 1      | JONÁS     | 17/05/1992 | HIJO       |
| 321    | 2      | RÓMULA    | 23/09/1923 | ABUELA     |
| 222    | 1      | ELEUTERIO | 30/10/2002 | HIJO       |
| 321    | 1      | RENATA    | 10/03/2002 | HIJA       |
| 111    | 2      | JULIANA   | 10/10/1936 | MADRE      |
| 321    | 3      | TORCUATA  | 17/05/1938 | ABUELA     |
| 111    | 3      | SINFOROSA | 23/09/1947 | ABUELA     |

Por último, se proyecta en las columnas indicadas

| nombre   |
|----------|
| ROMULA   |
| RENATA   |
| TORCUATA |

# SELECT. Consultas anidadas

77

- Es posible tener **varios niveles de consultas anidadas**

\* Empleadxs (dni y nombre) del departamento dirigido por 'MACARENO SOSO'.

# SELECT. Consultas anidadas

78

- El operador **IN** en realidad **compara filas (tuplas)**

\* Familiares con igual nombre que el empleado del que dependen

```
SELECT *
FROM familiar
WHERE (nombre, nssemp) IN (SELECT nombre, nss
 FROM empleado);
```

Primero, la subconsulta obtiene filas con el nombre y nss de todos los empleados

Después, para cada fila de FAMILIAR, compara las columnas “nombre” y “nssemp” con las filas obtenidas por la subconsulta

Compara por posición: “nombre” de FAMILIAR con “nombre” de EMPLEADO y “nssemp” con “nss” ... Y selecciona las filas de FAMILIAR que cumplen ambas comparaciones

\* NSS de empleadxs que tienen la misma ciudad y salario que ‘JONÁS SOLANO’.

```
SELECT nss
FROM empleado
WHERE (ciudad, salario) IN (SELECT ciudad, salario
 FROM empleado
 WHERE nombre = 'JONÁS'
 AND apellido = 'SOLANO');
```

| nss |
|-----|
| 123 |
| 234 |

# SELECT. Consultas anidadas

79

- Si la consulta anidada **devuelve una sola columna y una única fila**, el resultado es un valor escalar, y se permite usar **cualquier comparador** ( $=, <, >, <=, >=, <>$ ) en lugar de IN
  - El más usado es el  $=$

\* Empleados (nss y nombre) del departamento de Investigación  
 SELECT nss, nombre  
 FROM empleado

WHERE dep = (SELECT coddep  
 FROM departamento  
 WHERE nombre = 'INVESTIGACION');

Recomendamos usar IN  
 en lugar de  $=$  para  
 evitar errores en tiempo  
 de ejecución...

|     |
|-----|
| nss |
| 222 |

\* Familiares del empleado 'RIGOBERTA CALAVERA'

SELECT nombre  
 FROM familiar  
 WHERE nssemp = (SELECT nss  
 FROM empleado  
 WHERE nombre = 'RIGOBERTA'  
 AND apellido = 'CALAVERA');

|          |
|----------|
| nombre   |
| RENATA   |
| RÓMULA   |
| TORCUATA |

# SELECT. Consultas anidadas

80

- **Operador ANY o SOME** (otro uso del mismo operador)  
 $t \text{ op ANY } S \text{ o } t \text{ op SOME } S, \text{, op } \in \{ >, \geq, <, \leq, <>, = \}$ 
  - Compara una fila  $t$  con las filas resultado de una consulta anidada  $S$
  - Devuelve TRUE si **alguna** fila  $e$  de  $S$  cumple que  $t \text{ op } e$
  
- **Operador ALL** (otro uso del mismo operador)  
 $t \text{ op ALL } S, \text{, op } \in \{ >, \geq, <, \leq, <>, = \}$ 
  - Compara una fila  $t$  con filas resultado de una consulta anidada  $S$
  - Devuelve TRUE si **para toda** fila  $e$  de  $S$  se cumple que  $t \text{ op } e$

\* Nombres y apellidos de los empleados cuyo salario es menor que el de todos los empleados del departamento D3

```
SELECT nombre, apellido
FROM empleado
WHERE salario < ALL (SELECT salario
 FROM empleado
 WHERE dep = 'D3');
```

# SELECT. Consultas anidadas

81

- \* Nombres y apellidos de los empleados cuyo salario es menor que el de todos los empleados del departamento D3

SELECT nombre, apellido FROM empleado

WHERE salario < ALL (SELECT salario FROM empleado WHERE dep = 'D3');

Primero se evalúa la subconsulta

| salario |
|---------|
| 900     |
| 920     |

Después se seleccionan las filas de EMPLEADO cuyo valor de columna salario es inferior a todos los valores obtenidos en la subconsulta

| nombre     | apellido | nss | ... | salario | nssjefe | dep  |
|------------|----------|-----|-----|---------|---------|------|
| JONÁS      | SOLANO   | 123 |     | 1100    | 111     | D1   |
| RIGOBERTA  | CALAVERA | 321 |     | 900     | 333     | D3   |
| EUSEBIO    | MULETAS  | 222 |     | 2100    | 123     | D2   |
| MACARENO   | SOSO     | 111 |     | 1100    |         | D1   |
| CASIANA    | FABERGÉ  | 333 |     | 920     | 123     | D3   |
| FILOMENA   | RASCAS   | 234 |     | 1100    | 111     | D1   |
| GUMERSINDA | MIMOS    | 543 |     | 850     | NULL    | NULL |

Por último, se proyecta en las columnas indicadas

| nombre     | apellido |
|------------|----------|
| GUMERSINDA | MIMOS    |

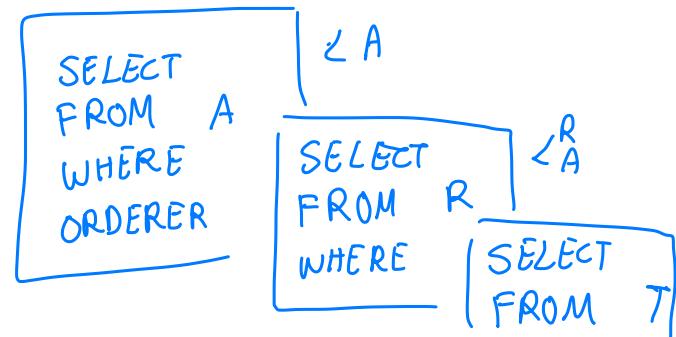
# SELECT. Correlación

82

- A veces, una consulta anidada necesita usar columnas de una tabla declarada en el FROM de una consulta exterior

\* Código de los departamentos cuyo director pertenece a otro departamento distinto  
**(detección de errores en los datos introducidos)**

```
SELECT coddep
FROM departamento
WHERE nssdire IN (SELECT nss
 FROM empleado
 WHERE dep <> coddep);
```



- La consulta contiene una **correlación**
- Una consulta exterior y otra anidada están **correlacionadas** si **una condición de la anidada contiene columnas de una tabla declarada en la consulta exterior**

# SELECT. Correlación

83

- Si las **columnas** en las consultas exterior y anidada se llaman igual, entonces existe **ambigüedad**

\* Nombre y apellidos de cada empleado con familiares de igual nombre que él

```
SELECT nombre, apellido
FROM empleado
WHERE nss IN (SELECT nssemp
 FROM familiar
 WHERE nombre = nombre); ▶ ¿cómo evitar esta ambigüedad?
```

- **Solución:** **CALIFICAR** la columna ambigua de la tabla exterior

```
SELECT nombre, apellido
FROM empleado e
WHERE nss IN (SELECT nssemp
 FROM familiar
 WHERE nombre = e.nombre);
```

**Regla:** una **columna no calificada** se refiere a la tabla del FROM de la SELECT anidada **más interior**

No hace falta calificar la columna de la tabla que está en el FROM de la consulta anidada (“nombre” de FAMILIAR): ya se asume que corresponde a dicha tabla. Pero, por supuesto, se pueden calificar ambas columnas.

# SELECT. Correlación. EXISTS

84

- Operador **EXISTS (S)**: comprobación de tablas vacías
  - Devuelve TRUE si S contiene **al menos una fila**
  - Devuelve FALSE si S está **vacía** (sin filas)

(i) S suele ser una **consulta anidada correlacionada**

\* Nombre y apellido de cada empleado con familiares de igual nombre que él

```
SELECT nombre, apellido
FROM empleado e
WHERE EXISTS (SELECT *
 FROM familiar
 WHERE nssemp=nss AND nombre=e.nombre);
```

Dentro de la consulta anidada aparecen columnas de las tablas que están en el FROM del SELECT externo (**correlación**)

- Se usa, sobre todo, **en sentido negativo: NOT EXISTS (S)**

\* Nombres y apellidos de los empleados sin familiares

```
SELECT nombre, apellido
FROM empleado
WHERE NOT EXISTS (SELECT *
 FROM familiar
 WHERE nssemp=nss);
```

No es importante qué columnas se seleccionan en la consulta anidada. EXISTS sólo comprueba si devuelve alguna fila (TRUE) o no (FALSE)

# SELECT. Funciones de agregados

85

- Es muy habitual necesitar realizar cálculos con los datos almacenados en una tabla:
  - ¿Cuántos empleados tenemos?
  - ¿Cuál es la media del salario de los empleados?
  - ¿A cuánto asciende la suma de todos los salarios de los empleados?
  - ¿Cuántos familiares hay?
  - ¿Cuál es el mínimo salario que cobran los empleados?  
¿Y cuál es el máximo?
- Es preciso poder calcular estos datos a partir de los valores almacenados en ciertas columnas (“salario”, por ejemplo)
- Esto se consigue con las *funciones de agregados*

# SELECT. Funciones de agregados

86

- **Funciones SUM( ), MAX( ), MIN( ), AVG( )...**
  - Suma, máximo, mínimo y media aritmética (promedio),...
  - Aplicadas a un multiconjunto (saco, bag) de valores numéricos
- Pueden aparecer en cláusula **SELECT**

\* Suma de los salarios y salario máximo, mínimo y medio de los empleados

SELECT **SUM(salario)**, **MAX(salario)**, **MIN(salario)**, **AVG(salario)**  
FROM empleado;

| <b>SUM(salario)</b> | <b>MAX(salario)</b> | <b>MIN(salario)</b> | <b>AVG(salario)</b> |
|---------------------|---------------------|---------------------|---------------------|
| 8070                | 2100                | 850                 | 1152,86             |

\* Suma de salarios y salario máximo, mínimo y medio de empleados del dep. Investigación

SELECT **SUM(salario)**, **MAX(salario)**, **MIN(salario)**, **AVG(salario)**  
FROM empleado  
WHERE dep IN (SELECT coddep  
FROM departamento  
WHERE nombre='INVESTIGACION');

| <b>SUM(salario)</b> | <b>MAX(salario)</b> | <b>MIN(salario)</b> | <b>AVG(salario)</b> |
|---------------------|---------------------|---------------------|---------------------|
| 3300                | 1100                | 1100                | 1100                |

- También pueden aparecer en cláusula **HAVING** (\*se verá\*)

# SELECT. Funciones de agregados

87

- **Función COUNT()**
  - Cuenta **número de filas** (usando **\***) o **valores no nulos** en una columna
- Puede aparecer en la cláusula **SELECT**

## ① Contar número de filas:

\* Número total de empleados de la empresa

```
SELECT COUNT(*)
FROM empleado;
```

|          |
|----------|
| COUNT(*) |
| 7        |

\* ¿Cuántos empleados hay en el departamento de Investigación?

```
SELECT COUNT(*)
FROM empleado JOIN departamento d
 ON dep = coddep
WHERE d.nombre = 'INVESTIGACION';
```

|          |
|----------|
| COUNT(*) |
| 1        |

\* ¿Cuántos familiares tiene la empleada 'RIGOBERTA CALAVERA'?

```
SELECT COUNT(*)
FROM familiar
WHERE nssemp IN (SELECT nss
 FROM empleado
 WHERE nombre = 'RIGOBERTA' AND apellido = 'CALAVERA');
```

|          |
|----------|
| COUNT(*) |
| 3        |

# SELECT. Funciones de agregados

88

## ② Contar valores no nulos en cierta columna

\* Contar el número de empleados de la empresa que tienen un jefe

SELECT **COUNT(nssjefe)** ▶ cuenta filas con nssjefe NOT NULL  
FROM empleado;

|                       |
|-----------------------|
| <b>COUNT(nssjefe)</b> |
| 5                     |

\* Contar el número de departamentos de la empresa que tienen un director

SELECT **COUNT(nssdire)** ▶ cuenta filas con nssdire NOT NULL  
FROM departamento;

|                       |
|-----------------------|
| <b>COUNT(nssdire)</b> |
| 3                     |

# SELECT. Funciones de agregados

89

## ③ Contar valores distintos en cierta columna: uso de DISTINCT

\* Contar el nº de valores distintos de salario que pueden cobrar los empleados

```
SELECT COUNT(salario)
FROM empleado;
```

|                |
|----------------|
| COUNT(salario) |
| 7              |

- ◀ Error: NO se eliminan duplicados y cuenta valores no nulos; como todos los empleados tienen salario, entonces COUNT(salario) = COUNT(\*)

□ La consulta correcta:

```
SELECT COUNT(DISTINCT salario)
FROM empleado; ◀ OK !!
```

|                         |
|-------------------------|
| COUNT(DISTINCT salario) |
| 5                       |

- COUNT( ) También pueden aparecer en cláusula HAVING



(\*se verá\*)

# SELECT. Funciones de agregados

90

## □ Ejemplo

\* ¿Cuántos empleados son jefes de algún otro?

-- Con correlación:

```
SELECT COUNT(*)
FROM empleado j
WHERE EXISTS (SELECT *
 FROM empleado e
 WHERE e.nssjefe=j.nss);
```

-- Sin correlación:

```
SELECT COUNT(*)
FROM empleado
WHERE nss IN (SELECT nssjefe
 FROM empleado);
```

-- Sin subconsulta (la mejor opción):

```
SELECT COUNT(DISTINCT nssjefe)
FROM empleado;
```

# SELECT. Funciones de agregados

91

- SUM, MAX, MIN, AVG admiten la cláusula...
  - ▣ **DISTINCT**
    - La función sólo considerará valores distintos (no las repeticiones)
    - Ejemplo: El valor de AVG(DISTINCT x) donde x={1, 1, 1 y 3} es **2**
  - ▣ **ALL**
    - Opción por omisión
    - La función considerará todos los valores, duplicados incluidos
    - Ejemplo: el valor de AVG(x) donde x={1, 1, 1 y 3} es **1.5**
- ¿Qué pasa con los **NULL**?
  - ▣ SUM, MAX, MIN, AVG ignoran el NULL en su argumento
  - ▣ COUNT(\*) no ignora los NULL
  - ▣ SUM, MAX, MIN, AVG devuelven NULL si el conjunto de datos no contiene filas, o contiene sólo filas con NULL en sus argumentos
  - ▣ COUNT devuelve un número o un cero, nunca NULL
- Se pueden anidar: **MAX(COUNT(\*))**, **AVG(MAX(salario))**

# SELECT. Funciones de agregados

92

- Es posible que una consulta anidada (y quizá correlacionada con otra exterior) incluya una función agregada

\* *Apellidos y nombres de los empleados con sólo un familiar*

SELECT apellido, nombre

FROM empleado

WHERE **1** = (SELECT COUNT(\*)

FROM familiar

WHERE **nssemp** = **nss**);

| apellido | nombre  |
|----------|---------|
| MULETAS  | EUSEBIO |
| SOLANO   | JONÁS   |

\* *Apellidos, nombres y salarios de empleados cuyo sueldo coincide con el sueldo medio*

SELECT nombre, apellido, salario

FROM empleado

WHERE salario = (SELECT AVG(salario)

FROM empleado);

| apellido | nombre |
|----------|--------|
| vacío    |        |

# SELECT. Funciones de agregados

93

- A veces se necesita realizar cálculos de agregados **agrupando los valores de cierta columna:**
  - ❑ ¿Cuántos empleados hay en cada departamento?
  - ❑ ¿Cuál es la media del salario de *los empleados* en cada departamento?
  - ❑ ¿A cuánto asciende la suma de los salarios de *los empleados* de cada departamento?
  - ❑ ¿Cuántos familiares tiene cada uno de *los empleados*?
  - ❑ ¿Cuál es el mínimo salario que cobran *los empleados* en cada departamento? ¿Y el máximo?
- Esto se consigue aplicando **funciones de agregados a grupos de filas**

# SELECT. Agrupación

94

## □ Cláusula **GROUP BY**

- Para formar **grupos de filas** dentro de una tabla
- Los grupos se forman **según el valor de ciertas columnas**
  - **Columnas de agrupación**
- Las **filas de cada grupo** tendrán el **mismo valor en las columnas de agrupación**

## □ Aplicación de funciones agregadas a grupos de filas

\* Para cada departamento, obtener su código, cuántos empleados tiene y el salario medio de los empleados del mismo

```
SELECT dep, COUNT(*), AVG(salario)
FROM empleado
GROUP BY dep ; ← una columna de agrupación
```

| dep  | COUNT(*) |
|------|----------|
| D1   | 3        |
| D2   | 1        |
| D3   | 2        |
| NULL | 1        |

👁 En la cláusula SELECT, las **columnas de agrupación deben aparecer junto con las funciones agregadas**, para que su valor (único para cada grupo) aparezca junto al resultado de aplicar las funciones al grupo

# SELECT. Agrupación

95

- \* Para cada departamento, obtener su código, cuántos empleados tiene dicho departamento y el salario medio de los empleados del mismo

```
SELECT dep, COUNT(*), AVG(salario)
 FROM empleado
GROUP BY dep ;
```

Primero se construyen los grupos en la tabla EMPLEADO, en función del valor de “dep”

| nombre     | apellido | salario | ... | dep  |
|------------|----------|---------|-----|------|
| JONÁS      | SOLANO   | 1100    | ... | D1   |
| MACARENO   | SOSO     | 1100    | ... | D1   |
| FILOMENA   | RASCAS   | 1100    | ... | D1   |
| EUSEBIO    | MULETAS  | 2100    | ... | D2   |
| RIGOBERTA  | CALAVERA | 900     | ... | D3   |
| CASIANA    | FABERGÉ  | 920     | ... | D3   |
| GUMERSINDA | MIMOS    | 850     | ... | NULL |

Grupo 1: D1

Grupo 2: D2

Grupo 3: D3

Grupo 4: NULL

| dep  | COUNT(*) | AVG(*) |
|------|----------|--------|
| D1   | 3        | 1100   |
| D2   | 1        | 2100   |
| D3   | 2        | 910    |
| NULL | 1        | 850    |

Después se aplica la cláusula SELECT a cada grupo



# SELECT. Agrupación y HAVING

97

```

SELECT nombre, nssdire
FROM departamento
WHERE coddep IN (SELECT dep
 FROM empleado
 GROUP BY dep
 HAVING AVG(salario) < 1200);

```

1º) Se ejecuta la **subconsulta**.

1.1 Se **particiona en grupos** la tabla EMPLEADO según el valor de la columna “dep”

| nombre     | apellido | salario | ... | dep  |
|------------|----------|---------|-----|------|
| JONÁS      | SOLANO   | 1100    | ... | D1   |
| MACARENO   | SOSO     | 1100    | ... | D1   |
| FILOMENA   | RASCAS   | 1100    | ... | D1   |
| EUSEBIO    | MULETAS  | 2100    | ... | D2   |
| RIGOBERTA  | CALAVERA | 900     | ... | D3   |
| CASIANA    | FABERGÉ  | 920     | ... | D3   |
| GUMERSINDA | MIMOS    | 850     | ... | NULL |

Grupo 1: D1
Grupo 2: D2
Grupo 3: D3
Grupo 4: NULL

# SELECT. Agrupación y HAVING

98

```

SELECT nombre, nssdire
FROM departamento
WHERE coddep IN (SELECT dep
 FROM empleado
 GROUP BY dep
 HAVING AVG(salario) < 1200);

```

1.2 Se **seleccionan** los grupos que cumplen lo indicado en el HAVING

| nombre     | apellido | salario | ... | dep  | AVG(salario)                             |
|------------|----------|---------|-----|------|------------------------------------------|
| JONÁS      | SOLANO   | 1100    | ... | D1   | 1100 <input checked="" type="checkbox"/> |
| MACARENO   | SOSO     | 1100    | ... | D1   |                                          |
| FILOMENA   | RASCAS   | 1100    | ... | D1   |                                          |
| EUSEBIO    | MULETAS  | 2100    | ... | D2   | 2100                                     |
| RIGOBERTA  | CALAVERA | 900     | ... | D3   |                                          |
| CASIANA    | FABERGÉ  | 920     | ... | D3   | 910 <input checked="" type="checkbox"/>  |
| GUMERSINDA | MIMOS    | 850     | ... | NULL |                                          |

| dep  |
|------|
| D1   |
| D3   |
| NULL |

1.3 Se aplica la cláusula SELECT de la subconsulta

# SELECT. Agrupación y HAVING

99

```
SELECT nombre, nssdire
FROM departamento
WHERE coddep IN (SELECT dep
 FROM empleado
 GROUP BY dep
 HAVING AVG(salario) < 1200);
```

2º) Se ejecuta la SELECT externa.

2.1 Selecciona las filas de DEPARTAMENTO con coddep IN ('D1', 'D3', NULL)

| nombre         | coddep | nssdire |
|----------------|--------|---------|
| INVESTIGACION  | D2     | 222     |
| ADMINISTRACION | D1     | 111     |
| PERSONAL       | D3     | 333     |
| TRAINING       | D4     | NULL    |

2.2 Se aplica la cláusula SELECT a las filas seleccionadas

| nombre         | nssdire |
|----------------|---------|
| ADMINISTRACION | 111     |
| PERSONAL       | 333     |

# SELECT. WHERE vs. HAVING

100

- **WHERE...** se aplica a **filas individuales**
- **HAVING...** se aplica a **grupos de filas**

\* Para cada departamento **en el que trabajen más de 10 empleados**, obtener el código y nombre del departamento, y el nº de empleados que trabajan en él

```
SELECT coddep, d.nombre, COUNT(*)
FROM departamento d
```

```
 JOIN empleado ON coddep = dep
WHERE COUNT(*) > 10
GROUP BY coddep, d.nombre;
```

```
SELECT coddep, d.nombre, COUNT(*)
FROM departamento d
 JOIN empleado ON coddep = dep
GROUP BY coddep, d.nombre
HAVING COUNT(*) > 10 ;
```



En el WHERE no pueden aplicarse funciones de agregación, pues WHERE se evalúa antes del GROUP BY

Además, se contaría sobre el resultado del join, antes de hacer los grupos --- ¡KO!

Hay que usar HAVING para aplicar las condiciones a los grupos que se forman como resultado del GROUP BY

Primero se hacen los grupos, y se cuentan las filas de cada grupo, descartando los que tienen menos de 10 filas

**SINTAX  
ERROR**

# SELECT. Oracle Online View

101

- Una **Online View** es una **consulta** (una SELECT completa) **dentro de la cláusula FROM** de otra SELECT
- Vista en línea
- No es una consulta anidada** ya que no está dentro de la cláusula WHERE de otra consulta, sino en el FROM

\* Para cada departamento, mostrar los datos de los empleados que cobran el máximo salario, así como ese salario máximo.

Columnas: (dep, nss, nombre, apellido, salariomax)

SELECT X.dep, e.nss, e.nombre, e.apellido, **X.max** salariomax

FROM empleado e JOIN (**SELECT dep, MAX(salario) max**

**FROM empleado  
GROUP BY dep**) X

ON e.dep = **X.dep**

2. El JOIN obtiene una fila para cada empleado, junto con el código de su departamento y el salario máximo

1. La **online view** obtiene, para cada departamento, el salario máximo que cobran sus empleados

WHERE e.salario = **X.max**;

3. El WHERE selecciona los empleados que cobran el salario máximo en su departamento.

# SELECT. Oracle Online View

102

\* Empleados (dni y nombre) del departamento dirigido por 'MACARENO SOSO'

-- con online view

```
SELECT dni, e.nombre
 FROM empleado e
 JOIN departamento d ON e.dep = d.coddep
 JOIN (SELECT nss
 FROM empleado
 WHERE nombre = 'MACARENO'
 AND apellido = 'SOSO') m
 ON d.nssdire = m.nss;
```

La **online view** permite que en los JOIN sólo se considere la fila del empleado MACARENO SOSO como director de departamento.

En la SELECT de la diapositiva 46, que no usa online view, en el JOIN se consideran todos los empleados directores

# Ejemplos resueltos

103

Los siguientes son enunciados de consultas que en las diapositivas que siguen han sido resueltas de diversas formas, para que se pueda observar que una misma consulta puede resolverse de varias maneras

1. Nombre, apellido y ciudad de los empleados del departamento llamado 'INVESTIGACION'.
2. NSS y nombre de los empleados que son jefes de otros y al mismo tiempo son directores de departamento.
3. NSS de los empleados cuyo salario es inferior a 950€ y no tienen familiares 'HIJO'.
4. DNI y nombre de los empleados del departamento dirigido por 'MACARENO SOSO'.
5. Nombre y fecha de nacimiento del empleado que más familiares tiene a su cargo (varios, si hay empate).
6. Nombre y fecha de nacimiento del empleado que más familiares tiene a su cargo (varios, si hay empate), indicando cuántos familiares tiene.

# Ejemplos resueltos

104

1. `SELECT e.nombre, e.apellido, e.ciudad  
FROM empleado e, departamento d  
WHERE e.dep = d.coddep  
AND d.nombre = 'INVESTIGACION';` -- JOIN clásico

---

`SELECT e.nombre, e.apellido, e.ciudad  
FROM (empleado e JOIN departamento d ON e.dep=d.coddep)  
WHERE d.nombre = 'INVESTIGACION';` -- Tabla reunida

---

`SELECT nombre, apellido, ciudad  
FROM empleado  
WHERE dep IN (SELECT coddep  
FROM departamento  
WHERE nombre = 'INVESTIGACION');` -- Subconsulta sin correlación

---

`SELECT nombre, apellido, ciudad  
FROM empleado e  
WHERE EXISTS (SELECT *  
FROM departamento d  
WHERE d.nombre = 'INVESTIGACION' AND e.dep = d.coddep);` -- Subconsulta con correlación

# Ejemplos resueltos

105

2.

```
SELECT nss, nombre
FROM empleado
WHERE nss IN
((SELECT nssjefe
 FROM empleado)
INTERSECT
(SELECT nssdire
 FROM departamento));
```

```
SELECT nss, nombre
FROM empleado
WHERE nss IN (SELECT nssjefe
 FROM empleado)
AND nss IN (SELECT nssdire
 FROM departamento);
```

3.

```
(SELECT nss
FROM empleado
WHERE salario<950)
MINUS
(SELECT nssemp
FROM familiar
WHERE parentesco = 'HIJO');
```

```
SELECT nss
FROM empleado
WHERE salario < 950
AND nss NOT IN
(SELECT nssemp
FROM familiar
WHERE parentesco = 'HIJO');
```

# Ejemplos resueltos

106

4. \* Empleados (*dni* y *nombre*) del departamento dirigido por ‘MACARENO SOSO’.

```
SELECT dni, nombre
FROM empleado
WHERE dep IN (SELECT coddep
 FROM departamento
 WHERE nssdire IN (SELECT nss
 FROM empleado
 WHERE nombre = 'MACARENO'
 AND apellido = 'SOSO'));
```

---

```
SELECT e.dni, e.nombre
FROM empleado e, departamento d , empleado m
WHERE e.dep = d.coddep AND d.nssdire = m.nss
 AND m.nombre = 'MACARENO' AND m.apellido = 'SOSO';
```

---

```
SELECT e.dni, e.nombre
FROM empleado e, departamento d ,
 (SELECT nss
 FROM empleado
 WHERE nombre = 'MACARENO' AND apellido = 'SOSO') m
WHERE e.dep = d.coddep AND d.nssdire = m.nss;
```

# Ejemplos resueltos

107

## 4. cont.

```
SELECT e.dni, e.nombre -- Tablas reunidas
```

```
FROM (empleado e
```

```
 JOIN (departamento d
```

```
 JOIN empleado m
```

```
 ON d.nssdire = m.nss)
```

```
 ON e.dep = d.coddep)
```

```
WHERE m.nombre = 'MACARENO' AND m.apellido = 'SOSO';
```

---

```
SELECT e.dni, e.nombre -- Tablas reunidas + online view
```

```
FROM empleado e
```

```
 JOIN (departamento d
```

```
 JOIN (SELECT nss FROM empleado
```

```
 WHERE nombre = 'MACARENO' AND apellido = 'SOSO') m
```

```
 ON d.nssdire = m.nss
```

```
 ON e.dep = d.coddep;
```

# Ejemplos resueltos

## MAX/MIN COUNT

108

### 5. \*Empleado (nombre, fechanacim) que más familiares tiene a su cargo

-- Serán los empleados tales que, si se cuenta sus familiares, dicho número coincide con el máximo número de familiares calculado para todos los empleados.

```
SELECT nombre, fechanacim
FROM empleado
WHERE nss IN
```

```
(SELECT nssemp
 FROM familiar
 GROUP BY nssemp
```

**HAVING COUNT(\*) = (SELECT MAX(COUNT(\*))**

```
FROM familiar
GROUP BY nssemp));
```

3) Una vez obtenidos los nss de los empleados, se obtienen sus datos de la tabla EMPLEADO

2) Selecciona empleados tales que, si se cuenta su nº de familiares, coincide con el máximo

1) La consulta anidada obtiene el máximo nº de familiares que tienen los empleados

# Ejemplos resueltos

109

6. \*Empleado (*nombre, fechanacim*) que más familiares tiene a su cargo, indicando **cuántos** tiene (*familiares*)

- Necesitamos acceder al número de familiares (el resultado del COUNT(\*))
- Para ello, podemos usar una *online view* porque una SELECT sólo puede mostrar columnas de las tablas que aparezcan en el FROM
- Obsérvese que la *online view* es igual a la subconsulta del ejercicio 5

SELECT nombre, fechanacim, **familiares**  
FROM empleado

**JOIN** (SELECT nssemp, **COUNT(\*) familiares**  
FROM familiar  
GROUP BY nssemp)

**HAVING COUNT(\*) = (SELECT MAX(COUNT\*))**

2) Selecciona empleados tales que su  
nº de familiares coincide con el máximo

**ON** nss = nssemp;

FROM familiar  
GROUP BY nssemp))

3) La *online view* calcula para cada empleado, cuántos familiares tiene. Al estar en el FROM, es posible sacar sus columnas como resultado final de la SELECT

1) La consulta anidada obtiene el máximo nº de familiares que tienen los empleados

# Ejemplos resueltos

110

6.cont. \*Empleado (*nombre, fechanacim*) que más *familiares* tiene a su cargo, indicando cuántos tiene (*familiares*)

-- Otra forma, algo menos “bonita” y eficiente que la anterior (es mejor usar HAVING)

```
SELECT nombre, fechanacim, familiares
FROM empleado e, (SELECT nssemp, COUNT(*) familiares
 FROM familiar
 GROUP BY nssemp) f
WHERE e.nss = f.nssemp
 AND familiares = (SELECT MAX(COUNT(*))
 FROM familiar
 GROUP BY nssemp);
```

---

```
SELECT nombre, fechanacim, familiares
FROM (empleado JOIN (SELECT nssemp, COUNT(*) familiares
 FROM familiar
 GROUP BY nssemp) ON nss=nssemp)
WHERE familiares = (SELECT MAX(COUNT(*))
 FROM familiar
 GROUP BY nssemp);
```

# División en SQL

111

- En ocasiones podemos encontrar un tipo de consultas que requieren la obtención de las **filas de una tabla** que están **relacionadas con todas** y cada una de **las filas de otra tabla**
- Ejemplos
  - \* Clientes que tienen cuentas en **todas** las sucursales
  - \* Nombres de las salas de lectura de la biblioteca que contienen libros de **todos** los géneros
  - \* Nombres de los actores/actrices que trabajan en **todas** las películas dirigidas por Tim Burton
  - \* Clientes que han hecho pedidos de **todos** los productos
- Este tipo de consultas se resuelven con una DIVISION ( $\div$ ) en el lenguaje formal Álgebra Relacional

# SELECT. División

112

- En SQL **no existe un operador** para realizar la **división** ( $\div$ ) del *Álgebra Relacional*
- Hay que redactar la consulta de forma que sea posible obtener el resultado utilizando las cláusulas y operadores que hemos visto para la sentencia SELECT
- El primer paso es *poner el enunciado de la consulta en negativo, utilizando dos o más negaciones...*
- ... Y después *expresarlo en SQL*
- Veamos cómo hacerlo con varios ejercicios

# SELECT. División

113

**EJERCICIO 1.**- Enuncia en **lenguaje natural** las siguientes consultas utilizando DOS (o más) negaciones.

0. Clientes que han hecho pedidos de todos los productos.
1. Palabras que contengan todas las vocales.
2. Alumnos que han aprobado todas las asignaturas.
3. Personas que hayan visto todas las películas de Harry Potter.
4. Actores que participan en todas las películas de Woody Allen.
5. Animales cuya piel incluye todos los colores básicos.

# SELECT. División

114

6. Niño que se ha comido todas las chuches de la piñata.
7. Socios de la biblioteca que han tomado prestado todos los libros de la saga "Canción de Hielo y Fuego".
8. Niño que ha pintarajeado con rotulador todas las paredes de su casa.
9. Paciente que ha sufrido gripe todos los años.
10. Productos que hayan sido comprados por todos los clientes.

# SELECT. División

115

**Solución** de una consulta del ejemplo:

0. Clientes que han hecho pedidos de todos los productos

- Clientes tales que **NO EXISTE un producto que NO hayan pedido.**

1º Lo que va detrás del “tod@s l@s”

2º La primera condición, lo que va antes del “tod@s l@s”

# SELECT. División

116

## SOLUCIONES del ejercicio 1.-

1. Palabras tales que NO EXISTE una vocal que NO contengan
2. Alumnos tales que NO EXISTE una asignatura que NO hayan aprobado
3. Personas tales que NO EXISTE una película de Harry Potter que NO hayan visto
4. Actores tales que NO EXISTE una película de Woody Allen en la que NO participan
5. Animales tales que NO EXISTE un color básico que NO esté en su piel

# SELECT. División

117

## SOLUCIONES del ejercicio 1.-

6. Niño tal que NO EXISTE una chuche de la piñata que NO se haya comido
7. Socios de la biblioteca tales que NO EXISTE un libro de la saga "Canción de Hielo y Fuego" que ellos NO hayan tomado prestado
8. Niño tal que NO EXISTE una pared de su casa que NO haya pintarrajeado
9. Paciente tal que NO EXISTE un año en el que NO haya sufrido gripe
10. Productos tales que NO EXISTE ningún cliente que NO los haya comprado

# SELECT. División

118

**EJERCICIO 2.**- Escribe distintas soluciones para una consulta SQL que implican una DIVISIÓN

## A) Esquema del **BANCO**

SUCURSAL (codigo, nombre, activo, ciudad)

CLIENTE (codigo, nombre, calle, ciudad)

CUENTA (cuenta\_id, *sucursal\_cod*, *cliente\_cod*, saldo)

Claves ajenas: *sucursal\_cod* → SUCURSAL(codigo)  
*cliente\_cod* → CLIENTE(codigo)

**Consulta:** Códigos de los clientes con cuenta en todas las sucursales que están en ‘MURCIA’

- Código de los clientes tales que  
NO EXISTE una sucursal de Murcia  
en la que NO tengan cuenta

# SELECT. División

119

## SOLUCIÓN 1:

-- Dos correlaciones

```
SELECT codigo
FROM cliente C
WHERE NOT EXISTS
 (SELECT *
 FROM sucursal S
 WHERE ciudad = 'MURCIA'
 AND NOT EXISTS
 (SELECT *
 FROM cuenta Q
 WHERE Q.cliente_cod = C.codigo
 AND Q.sucursal_cod= S.codigo));
```

1. NO EXISTE una sucursal de Murcia...

2. Esta SELECT saca las cuentas del cliente C en la sucursal S

3. ...en la que NO tenga una cuenta

# SELECT. División

120

## SOLUCIÓN 2:

-- Una correlación (un poquito mejor que la anterior)

```
SELECT codigo
FROM cliente C
WHERE NOT EXISTS
```

(SELECT \*

1. NO EXISTE una sucursal de Murcia...

FROM sucursal S

WHERE ciudad = 'MURCIA'

AND S.codigo NOT IN

NOT IN en vez de NOT EXISTS

(SELECT sucursal\_cod

FROM cuenta Q

2. ...en la que NO tenga una cuenta

WHERE Q.cliente\_cod = C.codigo));

La comparación que se hace en el  
NOT IN evita la segunda comparación  
dentro de la subconsulta

# SELECT. División

121

## SOLUCIÓN 3:

-- Restando

SELECT codigo  
FROM cliente C

WHERE NOT EXISTS -- clientes tales que la siguiente  
-- subconsulta (resta) no obtiene filas

( (SELECT codigo  
FROM sucursal  
WHERE ciudad = 'MURCIA')

**MINUS** -- EXCEPT en ANSI SQL

(SELECT sucursal\_cod  
FROM cuenta Q  
WHERE Q.cliente\_cod = C.codigo) );

1. Todas las sucursales de Murcia

2. Sucursales donde el cliente C tiene una cuenta

Si las sucursales de Murcia en las que un cliente tiene cuenta son las mismas que todas las sucursales de Murcia, la subconsulta queda vacía y el cliente es seleccionado para el resultado

# SELECT. División

122

## SOLUCIÓN 4:

-- Contando (es un "atajo" algo *chapucilla*)

```
SELECT codigo
FROM cliente C JOIN cuenta Q
 ON C.codigo = Q.cliente_cod
```

```
WHERE Q.sucursal_cod IN (SELECT codigo
 FROM sucursal
 WHERE ciudad = 'MURCIA')
```

GROUP BY codigo

HAVING COUNT(DISTINCT(Q.sucursal\_cod))=(SELECT COUNT(\*)
 FROM sucursal
 WHERE ciudad='MURCIA');

2. Clientes con cuenta en alguna sucursal de Murcia

1. Códigos de las sucursales de Murcia

3. Cuenta las sucursales de Murcia

4. Cuenta las sucursales diferentes de Murcia en las que cada cliente tiene cuentas

Si el número de sucursales de Murcia en las que un cliente tiene cuenta coincide con el número total de sucursales de Murcia, entonces el cliente es seleccionado para el resultado

# SELECT. División

123

## SOLUCIÓN 4bis:

-- Igual a la anterior, pero con notación JOIN clásica (*igual de “chapucilla”*)

SELECT codigo

FROM cliente C, cuenta Q

WHERE C.codigo = Q.cliente\_cod

    AND Q.sucursal\_cod IN (SELECT codigo

                FROM sucursal

                WHERE ciudad = 'MURCIA')

GROUP BY codigo

HAVING COUNT(DISTINCT(Q.sucursal\_cod)) = (SELECT COUNT(\*)

                FROM sucursal

                WHERE ciudad='MURCIA');

# SELECT. División

124

## B) Esquema PRODUCTORA

PELICULA (codp, titulo, año, genero, guion, *director*, ...)

ACTOR(coda, nombre, nomreal, nacionalidad, fechanacim,...)

ACTUA\_EN (actor, film, papel, paga)

Claves ajenas: actor → ACTOR(coda)

film → PELICULA(codp)

**Consulta.**- Nombre de actores/actrices que participan en todas las películas dirigidas por el director con código '103'.

- Seleccionar nombres de los actores/actrices tales que NO EXISTE una película dirigida por el director 103 en la que NO hayan participado

# SELECT. División

125

SOLUCIÓN 1:

-- Dos correlaciones

SELECT nombre

FROM actor a

WHERE NOT EXISTS

(SELECT \*

FROM pelicula p

WHERE director = '103'

AND NOT EXISTS (SELECT \*

FROM actua\_en

WHERE actor = a.coda

AND film = p.codp ));

# SELECT. División

126

## SOLUCIÓN 2:

-- Una correlación

SELECT nombre

FROM actor a

**WHERE NOT EXISTS**

(SELECT \*

FROM pelicula

WHERE director = '103'

AND codp **NOT IN**

(SELECT film

FROM actua\_en

WHERE **actor = a.coda**));

# SELECT. División

127

## SOLUCIÓN 3:

-- Restando; una correlación

SELECT nombre

FROM actor a

WHERE NOT EXISTS (

(SELECT codp

FROM pelicula

WHERE director = '103')

**MINUS** -- EXCEPT en ANSI SQL

(SELECT film

FROM actua\_en

WHERE actor = a.coda) );

# SELECT. División

128

## SOLUCIÓN 4:

--Contando (atajo algo *chapucilla*)

```
SELECT nombre
FROM actor a JOIN actua_en c
 ON a.coda = c.actor
WHERE c.film IN (SELECT codp
 FROM pelicula
 WHERE director = '103')
```

```
GROUP BY a.coda
HAVING COUNT(DISTINCT(c.film))=(SELECT COUNT(*)
 FROM pelicula
 WHERE director = '103');
```

3. Cuenta las películas diferentes del director 103 en las que participa cada actor/actriz

1. Actores/actrices que participan en películas del director 103

2. Cuenta las películas del director 103

# SELECT. Evaluación

129

- En una consulta SQL hay un máximo de 6 cláusulas
- Sólo son obligatorias SELECT y FROM
- Debe terminar con un ;

| Orden de ESCRITURA                 |                                             |
|------------------------------------|---------------------------------------------|
| SELECT lista columnas o funciones  | <i>Lista select:</i> lo que desea obtener   |
| FROM lista tablas                  | Tablas necesarias (incluso las reunidas)    |
| WHERE condición para filas         | Condiciones de selección de filas o reunión |
| GROUP BY lista columnas agrupación | Especificación del agrupamiento de filas    |
| HAVING condición para grupos       | Condición de selección de grupos de filas   |
| ORDER BY lista columnas ordenación | Orden de presentación del resultado         |

# SELECT. Evaluación

130

| ↓ Orden de EVALUACIÓN              |                                                                                                       |
|------------------------------------|-------------------------------------------------------------------------------------------------------|
| FROM lista tablas                  | Acceso a tablas, o reunión                                                                            |
| WHERE condición para filas         | Descarta filas que incumplen la condición                                                             |
| GROUP BY lista columnas agrupación | Crea grupos de filas                                                                                  |
| HAVING condición para grupos       | Descarta los grupos de filas que la incumplen                                                         |
| ORDER BY lista columnas ordenación | Ordena las filas del resultado                                                                        |
| SELECT lista columnas o funciones  | Para el conjunto de filas resultado, elige las columnas indicadas, calcula las funciones y lo muestra |

- Diversas formas de especificar una misma consulta
  - Ejemplo: es posible expresar una consulta utilizando...
    - Lista de tablas en el FROM y condiciones de reunión en el WHERE, o
    - Tablas reunidas en la cláusula FROM, o
    - Consultas anidadas (correlacionadas o no ) y comparación con IN ...
- ▶ Flexibilidad

## 8.2 Buenas Prácticas

131

- Objetivo: escribir consultas SQL correctas, más claras y legibles, lo que facilita la **comprensión**, su **eficiencia** y el **mantenimiento** del código SQL a lo largo del tiempo
- Se recomienda usar, en orden de elegancia (y claridad):
  - +  IN (subconsulta)
  - EXISTS (subconsulta)
  - JOIN
- Pero deberíamos aplicar los consejos indicados en las diapositivas siguientes, **siempre que sea posible** y tenga sentido

## 8.2 Buenas Prácticas

132

- **No usar \* en la cláusula SELECT, sino enumerar las columnas que se necesitan**

SELECT \*  
FROM empleado;

Con SELECT \*, antes de ejecutar la consulta el SGBD accederá a los metadatos para saber qué columnas contiene la tabla

SELECT nss, nombre, ciudad  
FROM empleado;

Más eficiente y más  
comprendible



- Usar **alias** o pseudónimos para las **columnas cuando se realiza un join**

SELECT nss, apellido, parentesco  
FROM empleado JOIN familiar  
ON nss = nssemp;

SELECT e.nss, e.apellido, f.parentesco  
FROM empleado e JOIN familiar f  
ON e.nss = f.nssemp;

Ahorra al SGBD acceder a los metadatos para localizar a qué tabla pertenece cada columna



## 8.2 Buenas Prácticas

133

- Utilizar el operador **JOIN ON** (tablas reunidas) **en lugar de la lista de tablas** en la cláusula FROM y la condición de reunión en la WHERE

```
SELECT e.nombre empleado, d.nombre departamento
FROM empleado e, departamento d
WHERE coddep <> 'D2'
 AND e.dep = d.coddep;
```

```
SELECT e.nombre empleado, d.nombre departamento
FROM empleado e JOIN departamento d
 ON e.dep = d.coddep
WHERE d.coddep <> 'D2';
```

Consulta más clara  
y comprensible  
Y menos propensa a  
errores de  
programación



## 8.2 Buenas Prácticas

134

- Formatear las consultas en múltiples líneas**
- Usar sangrías y espaciados**

```
SELECT d.coddep, COUNT(*) empleados FROM
departamento d JOIN empleado e ON d.coddep =
e.dep WHERE e.nss IN (SELECT nssemp FROM familiar
WHERE parentesco = 'HIJA') GROUP BY d.coddep
HAVING COUNT(*) > 5;
```

```
SELECT d.coddep, COUNT(*) empleados
FROM departamento d
JOIN empleado e
ON d.coddep = e.dep
WHERE e.nss IN (SELECT nssemp
FROM familiar
WHERE parentesco = 'HIJA')
GROUP BY d.coddep
HAVING COUNT(*) > 5;
```



## 8.2 Buenas Prácticas

135

- Escribir las consultas con el **mínimo ordenamiento implícito**: no usar innecesariamente cláusulas ORDER BY, DISTINCT, GROUP BY...

Consultas más eficientes y comprensibles

SELECT nombre, apellido, ciudad

FROM empleado

WHERE nss IN (SELECT DISTINCT nssemp

FROM familiar

WHERE añonacim BETWEEN 1990 AND 2005)

**GROUP BY nombre, apellido, ciudad;**

El GROUP BY en la SELECT exterior es erróneo porque no se aplican funciones de agregados. Es un error común usarlo (en vez de DISTINCT) para eliminar duplicados, lo que provoca un ordenamiento de los resultados.

En este caso, además, es innecesario porque es raro que se obtenga filas duplicadas

El DISTINCT en la SELECT anidada provoca un ordenamiento de su resultado, para eliminar duplicados. Es del todo innecesario porque la comparación IN del WHERE termina en cuanto encuentra un valor de nssemp que lo hace true

# 8.2 Buenas Prácticas

136

## □ Evitar subconsultas correlacionadas

- Una consulta anidada correlacionada **se evalúa una vez para cada fila** (o combinación de filas) **de la consulta exterior**

\* Nombre y apellido de los empleados que tienen algún familiar con igual nombre

```
SELECT nombre, apellido
FROM empleado e
WHERE nss IN (SELECT nssemp
 FROM familiar
 WHERE nombre = e.nombre);
```

Esta SELECT anidada se ejecuta tantas veces como filas hay en la tabla EMPLEADO

- Veámoslo con un ejemplo...

# Evitar subconsultas correlacionadas

137

| EMPLEADO   |          |     |     |
|------------|----------|-----|-----|
| nombre     | apellido | nss | ... |
| JONÁS      | SOLANO   | 123 | ... |
| RIGOBERTA  | CALAVERA | 321 | ... |
| EUSEBIO    | MULETAS  | 222 | ... |
| MACARENO   | SOSO     | 111 | ... |
| CASIANA    | FABERGÉ  | 333 | ... |
| FILOMENA   | RASCAS   | 234 | ... |
| GUMERSINDA | MIMOS    | 543 | ... |

## FAMILIAR

| nssemp | numero | nombre    | ... |
|--------|--------|-----------|-----|
| 123    | 1      | JONÁS     | ... |
| 321    | 2      | RÓMULA    | ... |
| 222    | 1      | ELEUTERIO | ... |
| 321    | 1      | RENATA    | ... |
| 111    | 2      | JULIANA   | ... |
| 321    | 3      | TORCUATA  | ... |
| 111    | 3      | SINFOROSA | ... |

Las distintas ejecuciones de la SELECT anidada

1

(SELECT nssemp  
FROM familiar  
WHERE nombre = 'JONÁS')

2

(SELECT nssemp  
FROM familiar  
WHERE nombre = 'RIGOBERTA')

3

(SELECT nssemp  
FROM familiar  
WHERE nombre = 'EUSEBIO')

4

(SELECT nssemp  
FROM familiar  
WHERE nombre = 'MACARENO')

5

(SELECT nssemp  
FROM familiar  
WHERE nombre = 'CASIANA')

6

(SELECT nssemp  
FROM familiar  
WHERE nombre = 'FILOMENA')

7

(SELECT nssemp  
FROM familiar  
WHERE nombre = 'GUMERSINDA')

# Evitar subconsultas correlacionadas

138

- Esta circunstancia suele implicar que este tipo de **consultas correlacionadas sean poco eficientes...** 
- ... y sea deseable **redactarlas de otra manera**
- La **mayoría** de las correlaciones (no todas) se puede evitar, expresando la consulta de otro modo

\* Nombre y apellido de los empleados que tienen algún familiar con igual nombre

```
SELECT e.nombre, apellido
FROM empleado e JOIN familiar f
 ON e.nss = f.nssemp
WHERE e.nombre = f.nombre;
```

Esta consulta es equivalente a la anterior y no contiene ninguna correlación: ya no hay subconsulta, sino que utiliza la operación REUNION (JOIN)



- Una SELECT con una **consulta anidada** que use el operador **=** o **IN** **siempre puede expresarse como una reunión (JOIN)**

# Evitar subconsultas correlacionadas

139

\* Código de los departamentos cuyo director pertenece a otro departamento distinto (detección de errores en los datos introducidos)

```
SELECT coddep
FROM departamento d
WHERE nssdire IN (SELECT nss
 FROM empleado
 WHERE dep <> d.coddep);
```

```
SELECT coddep
FROM departamento d
JOIN empleado e
ON d.nssdire = e.nss
WHERE e.dep <> d.coddep;
```

\* Nombre y departamento de los empleados jefes de otros que cobran más que él

```
SELECT nombre, dep
FROM empleado j
WHERE nss IN (SELECT nssjefe
 FROM empleado e
 WHERE e.salario > j.salario);
```

```
SELECT j.nombre, j.dep
FROM empleado j
JOIN empleado e
ON j.nss = e.nssjefe
WHERE e.salario > j.salario;
```

# Evitar subconsultas correlacionadas

140

- La **correlación** que provoca **EXISTS**, a veces se **puede evitar** usando el operador **IN** o **JOIN**

\* Nombre y apellido de lxs empleadxs que tienen algún familiar con igual nombre

```
SELECT nombre, apellido
FROM empleado e
WHERE EXISTS (SELECT *
 FROM familiar
 WHERE nssemp = e.nss AND nombre = e.nombre);
```

EXISTS y  
correlación



```
SELECT nombre, apellido
FROM empleado
WHERE (nss, nombre) IN (SELECT nssemp, nombre
 FROM familiar);
```

Uso de IN: no  
hay correlación  
y es equivalente



```
SELECT e.nombre, e.apellido
FROM empleado e JOIN familiar f
ON e.nss = f.nssemp
WHERE e.nombre = f.nombre;
```

Uso de JOIN: no  
hay correlación y  
es equivalente



# Evitar subconsultas correlacionadas

141

\* Nombres y apellidos de los empleados sin familiares

```
SELECT nombre, apellido
FROM empleado e
WHERE NOT EXISTS (SELECT *
 FROM familiar
 WHERE nssemp = e.nss);
```

EXISTS y  
correlación



```
SELECT nombre, apellido
FROM empleado
WHERE nss NOT IN (SELECT nssemp
 FROM familiar);
```

Uso de NOT IN: no hay  
correlación y es  
equivalente



```
SELECT e.nombre, e.apellido
FROM empleado e
JOIN familiar f
ON e.nss <> f.nssemp;
```



ANTI-JOIN  
DETECTED

¡¡Aaaargh!!

**Consulta INCORRECTA:** el JOIN sirve para encontrar filas conectadas en una y otra tabla, no para justo lo contrario

Expliquemos esto...

# Evitar subconsultas correlacionadas

142

- Para entenderlo bien, veamos lo que devuelve el “**anti-JOIN**” antes de seleccionar las dos columnas para el resultado

```
SELECT e.nombre, e.apellido
FROM empleado e JOIN familiar f
ON e.nss <> f.nssemp;
```

**Efectivamente, combina  
cada empleado con todo  
familiar de otro  
empleado distinto y  
nunca con los suyos!**

# Evitar subconsultas correlacionadas

143

- Vemos que **NO** se puede **usar JOIN para evitar consultas anidadas que usan NOT IN o NOT EXISTS**
- Y eso ocurre **en general, haya o no correlación**

\* Código y nombre de los departamentos que no tengan ningún empleado de Murcia

```
SELECT coddep, nombre
FROM departamento
```

2º Selecciona los departamentos que **no están** entre los devueltos por la subconsulta

```
WHERE coddep NOT IN (
```

Uso de NOT IN y subconsulta  
**SIN correlación**

```
(SELECT dep
FROM empleado
WHERE ciudad = 'MURCIA');
```

1º La subconsulta selecciona los departamentos de los empleados de Murcia

```
SELECT d.coddep, d.nombre
FROM departamento d
JOIN empleado e
ON d.coddep <> e.dep
WHERE e.ciudad = 'MURCIA';
```



**¡¡Aaaargh!!**



**ANTI-JOIN  
DETECTED**

Comprendámoslo con datos de ejemplo

# Evitar subconsultas correlacionadas

144

- El **resultado final correcto** es este:  
(SELECT con NOT IN y subconsulta):



| d.coddep | d.nombre      |
|----------|---------------|
| D2       | INVESTIGACION |
| D4       | TRAINING      |

- ¿Por qué la consulta con el “**anti-JOIN**” es incorrecta?
  - Tras ejecutar el “anti-JOIN” y aplicar la condición (ciudad = 'MURCIA')...
  - ... pero ANTES de seleccionar las dos columnas para el resultado, se obtiene esta tabla (bordes verdes)...

```
SELECT d.coddep, d.nombre
FROM departamento d
JOIN empleado e
ON d.coddep <> e.dep
WHERE e.ciudad = 'MURCIA';
```



Combina cada departamento con todos los empleados de Murcia **del resto de departamentos** con todos, salvo con los suyos!

Resultado final incorrecto

| d.coddep | d.nombre       | .. | e.nombre | dep | ... |
|----------|----------------|----|----------|-----|-----|
| D1       | ADMINISTRACION | .. | CASIANA  | D3  | ... |
| D2       | INVESTIGACIÓN  | .. | CASIANA  | D3  | ... |
| D2       | INVESTIGACIÓN  | .. | FILOMENA | D1  | ... |
| D2       | INVESTIGACIÓN  | .. | JONÁS    | D1  | ... |
| D3       | PERSONAL       | .. | FILOMENA | D1  | ... |
| D3       | PERSONAL       | .. | JONÁS    | D1  | ... |
| D4       | TRAINING       | .. | CASIANA  | D3  | ... |
| D4       | TRAINING       | .. | FILOMENA | D1  | ... |
| D4       | TRAINING       | .. | JONÁS    | D1  | ... |

# Evitar subconsultas correlacionadas

145

- Tampoco valdría reescribir la consulta, “corrigiendo” el “anti-JOIN” y cambiando la condición de selección de filas...

\* Código y nombre de los departamentos que no tengan ningún empleado de Murcia

```
SELECT d.coddep, d.nombre
FROM departamento d
JOIN empleado e
ON d.coddep = e.dep
WHERE e.ciudad <> 'MURCIA' ;
```

El JOIN es correcto: reúne cada departamento con cada uno de sus empleados.  
Eso está bien...

| d.coddep | d.nombre       |
|----------|----------------|
| D1       | ADMINISTRACION |
| D2       | INVESTIGACION  |
| D3       | PERSONAL       |



Resultado final incorrecto

Pero al aplicar la condición del WHERE, el **resultado** sería **incorrecto**: un departamento con varios empleados de Murcia y otros empleados de otras ciudades también saldría en el resultado (ADMINISTRACION)

- Veamos por qué es incorrecto, detallando cómo resuelve esta consulta el SGBD...

# Evitar subconsultas correlacionadas

146

```

SELECT d.coddep, d.nombre
FROM departamento d JOIN empleado e
 ON d.coddep = e.dep
WHERE e.ciudad <> 'MURCIA';

```

1º

2º

- 1º se ejecuta el JOIN

| d.nombre       | coddep | nssdire | e.nombre  | apellido | nss | dni | fechanacim | ciudad  | estado_civil | salario | dep | nssjefe |
|----------------|--------|---------|-----------|----------|-----|-----|------------|---------|--------------|---------|-----|---------|
| ADMINISTRACION | D1     | 111     | MACARENO  | SOSO     | 111 | 23D | 1944-04-06 | JUMILLA | S            | 1100    | D1  | NULL    |
| ADMINISTRACION | D1     | 111     | JONÁS     | SOLANO   | 123 | 11A | 1945-10-10 | MURCIA  | P            | 1100    | D1  | 111     |
| ADMINISTRACION | D1     | 111     | FILOMENA  | RASCAS   | 234 | 34E | 1970-07-18 | MURCIA  | C            | 1100    | D1  | 111     |
| INVESTIGACION  | D2     | 222     | EUSEBIO   | MULETAS  | 222 | 22B | 1969-01-01 | TOTANA  | D            | 2100    | D2  | 123     |
| PERSONAL       | D3     | 333     | RIGOBERTA | CALAVERA | 321 | 21C | 1974-11-12 | YECLA   | C            | 900     | D3  | 333     |
| PERSONAL       | D3     | 333     | CASIANA   | FABERGÉ  | 333 | 33B | 1943-06-15 | MURCIA  | V            | 920     | D3  | 123     |

- 2º se aplica la condición de selección de filas

| d.nombre       | coddep | nssdire | e.nombre  | apellido | nss | dni | fechanacim | ciudad  | estado_civil | salario | dep | nssjefe |
|----------------|--------|---------|-----------|----------|-----|-----|------------|---------|--------------|---------|-----|---------|
| ADMINISTRACION | D1     | 111     | MACARENO  | SOSO     | 111 | 23D | 1944-04-06 | JUMILLA | S            | 1100    | D1  | NULL    |
| INVESTIGACION  | D2     | 222     | EUSEBIO   | MULETAS  | 222 | 22B | 1969-01-01 | TOTANA  | D            | 2100    | D2  | 123     |
| PERSONAL       | D3     | 333     | RIGOBERTA | CALAVERA | 321 | 21C | 1974-11-12 | YECLA   | C            | 900     | D3  | 333     |

- 3º se seleccionan las columnas indicadas

Es incorrecta por seleccionar departamentos con ALGÚN empleado que NO sea de Murcia, en vez de los departamentos que no tienen NINGUN empleado de Murcia

| d.coddep | d.nombre       |
|----------|----------------|
| D1       | ADMINISTRACION |
| D2       | INVESTIGACION  |
| D3       | PERSONAL       |

# Rendimiento

147

- En cuanto a la **eficiencia**, de mejor a peor...
  - +  IN con subconsulta sin correlación
  - JOIN
  - IN y EXISTS con subconsulta con correlación
- Pero... a veces un JOIN es igual o más eficiente que un IN con subconsulta sin correlación
  - Depende del **esquema** de BD, de las estructuras físicas de **almacenamiento** y **acceso** utilizadas, y sobre todo del **optimizador** del SGBD
  - Más adelante estudiaremos los **índices** y cómo utilizarlos con propósito de aumentar el rendimiento
  - Consejo: estudiar los manuales del SGBD (Oracle, MySQL, etc.) y probar!

## 8.3. Modificación de datos

148

- Sentencias SQL que permiten **introducir** nuevas filas, **eliminar** filas completas y **modificar** el contenido de las **filas** de una tabla
- ∈ Lenguaje de Manipulación de Datos (LMD) del SQL
- Orden INSERT
  - INSERT .. SELECT
  - (LDD) CREATE AS SELECT
- Orden DELETE
- Orden UPDATE

# Inserción de datos en tablas

149

## □ Orden **INSERT**

- Añade una fila completa a una tabla
- Incluye nombre de tabla y lista de valores para columnas, escritos en igual orden al especificado en el CREATE TABLE

**INSERT INTO** empleado

```
VALUES ('CEFERINA', 'SALSIPUEDES', 444, '44C',
 TO_DATE('30-DIC-92', 'dd-MON-yy'),
 'YECLA', 'S', 3700, 'D4', NULL, 0);
```

**INSERT INTO** empleado

```
VALUES ('PRUDENCIO', 'SILENCIO', 456, '45S',
 TO_DATE('22/01/1995', 'dd/mm/yyyy'),
 'VILLENA', 'P', 2250, 'D4', 444, 0);
```

**INSERT INTO** empleado

```
VALUES ('FERDINANDA', 'FUERTES', 567, '56F',
 TO_DATE('12-AGO-88', 'dd-MON-yy'),
 'MAZARRÓN', 'C', 1560, 'D4', 456, 0);
```

# Valores en cualquier orden

150

- Para poder poner los **valores** de las columnas **en cualquier orden**, hay que especificar los **nombres de las columnas antes de la cláusula VALUES** (**recomendado**)
- El SGBD hará un *matching* nombre-valor por posición

**INSERT INTO empleado (nombre, apellido, nss, dni, dep, salario, nssjefe, ciudad, fechanacim, est\_civil, cuantos\_familiares)**

**VALUES ('CEFERINA', 'SALSIPUEDES', 444, '44C', 'D4', 3700, NULL, 'YECLA', TO\_DATE('30-DIC-1992', 'dd-MON-yyyy'), 'S', 0);**

**INSERT INTO empleado (apellido, nombre, nss, dni, cuantos\_familiares, fechanacim, ciudad, est\_civil, salario, nssjefe, dep)**

**VALUES ('SILENCIO', 'PRUDENCIO', 456, '45S', 0, TO\_DATE('22/01/1995', 'dd/mm/yyyy'), 'VILLENA', 'P', 2250, 444, 'D4');**

**INSERT INTO empleado (nombre, apellido, nss, dni, ciudad, fechanacim, est\_civil, salario, nssjefe, dep, cuantos\_familiares)**

**VALUES ('FERDINANDA', 'FUERTES', 567, '56F', 'MAZARRÓN', TO\_DATE('12-AGO-88', 'dd-MON-yy'), 'C', 1560, 456, 'D4', 0);**

# Omisión de valores

151

- Especificación de valores sólo para **algunas** columnas
    - Es posible **omitir** las **columnas cuyo valor se desconoce**
    - ... siempre que **admitan NULL o tengan valor por defecto**
  - Cada columna no especificada tomará el...
    - **valor por omisión:** valor tomado de su cláusula DEFAULT, o
    - **NULL:** si la columna admite nulos y no tiene definido DEFAULT

\* Inserción de empleado del que sólo se conoce nombre, apellido, NSS, DNI y dep  
INSERT INTO empleado (nombre, apellido, nss, dni, dep)  
VALUES ('GILDA', 'ROLLER', 678, '67R', 'D2');

# Control de la integridad

152

## □ Comprobación automática (SGBD) de restricciones

```
INSERT INTO EMPLEADO (nombre, apellido, dep)
VALUES ('CIRILO', 'HUM', 'D2');
```

### □ Inserción rechazada:

no incluye valor para la columna “nss”, definida NOT NULL

\* Supongamos que no existe ningún departamento con coddep='D8'

```
INSERT INTO EMPLEADO (nombre, apellido, nss, dni, dep)
VALUES ('CIRILO','HUM', 789, '78H', 'D8');
```

### □ Inserción rechazada :

no encuentra el valor D8 en la columna “coddep” de DEPARTAMENTO

### □ Se incumple la Integridad Referencial

- Restricción “emp\_fk\_dep”  
(clave ajena desde EMPLEADO a DEPARTAMENTO)

# INSERT INTO SELECT

153

## □ **Carga de una tabla con “información resumen” de la BD**

\* Sea una tabla INFO\_DEP vacía, creada mediante esta sentencia

```
CREATE TABLE info_dep (
 nombre_dep VARCHAR2(25) NOT NULL,
 num_empleados NUMBER(3) DEFAULT 0 NOT NULL,
 suma_salarios NUMBER(8) NOT NULL
);
```

► Almacenará el nombre de cada departamento, cuántos empleados tiene y la suma de los salarios de sus empleados

## □ **Es posible llenar una tabla extrayendo su contenido de la base de datos, mediante una SELECT**

```
INSERT INTO info_dep (nombre_dep, num_empleados, suma_salarios)
SELECT d.nombre, COUNT(*), SUM(salario)
FROM departamento d JOIN empleado e ON d.coddep = e.dep
GROUP BY d.nombre ;
```

# INSERT INTO SELECT

154

INFO\_DEP

| nombre_dep     | num_empleados | suma_salarios |
|----------------|---------------|---------------|
| ADMINISTRACION | 3             | 3300          |
| PERSONAL       | 2             | 1820          |
| INVESTIGACION  | 2             | 3100          |
| TRAINNING      | 3             | 7510          |

- Por supuesto, es posible manipular el contenido de la tabla mediante sentencias SELECT

```
SELECT nombre_dep, suma_salarios
FROM info_dep
WHERE num_empleados < 3;
```

| nombre_dep    | suma_salarios |
|---------------|---------------|
| PERSONAL      | 1820          |
| INVESTIGACION | 3100          |

INFO\_DEP puede contener información **no actualizada**

- Si se modifican filas o valores en EMPLEADO y/o en DEPARTAMENTO, esos cambios no se reflejarán en esta tabla INFO\_DEP
  - Esta tabla no se actualiza automáticamente
  - Si en vez de una tabla se definiera una VISTA con “la misma estructura”, sí mostraría siempre los datos actualizados
  - Veremos el concepto de VISTA más adelante



# Oracle SQL. Crear y Cargar una tabla

155

- El SQL de Oracle, además permite crear una tabla como “copia de estructura y contenido” de otra tabla  
**CREATE TABLE <tabla> AS SELECT ...;**

\* Crear una tabla para almacenar datos de empleados nacidos antes de 1960

**CREATE TABLE** emp\_veterano (nss, nombre, jefe, depto)  
**AS SELECT** nss, nombre || '' || apellido, nssjefe, dep  
FROM empleado  
WHERE EXTRACT(YEAR FROM fechanacim) < 1960;

EMP\_VETERANO

| nss | nombre          | jefe | depto |
|-----|-----------------|------|-------|
| 111 | MACARENO SOSO   | NULL | D1    |
| 123 | JONÁS SOLANO    | 111  | D1    |
| 333 | CASIANA FABERGÉ | 123  | D3    |

# Eliminación de filas

156

## □ Orden **DELETE**

- **Elimina filas completas** de una tabla (sólo una en el FROM)
- **Cláusula WHERE** para seleccionar las filas que se desea eliminar

```
DELETE FROM empleado
WHERE apellido = 'ROLLER';

DELETE FROM departamento
WHERE nssdire = 111;

DELETE FROM empleado
WHERE dep IN
(SELECT coddep
FROM departamento
WHERE nombre = 'INVESTIGACION');
```

1° se accede a la tabla  
2° se ejecuta el WHERE  
3° se ejecuta el borrado

## □ Propagación de eliminaciones vía claves ajena

- Según las acciones de mantenimiento de la Integridad Referencial (ON DELETE...) en los CREATE TABLE



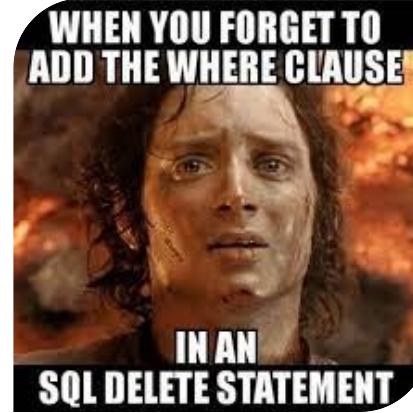
# Eliminación de filas

157

! ATTENTION

- Si no hay WHERE, se eliminan TODAS las filas: la tabla **permanece**, pero queda **vacía**

**DELETE  
FROM empleado;**



# Actualización de Datos

158

## □ Orden **UPDATE**

- Modifica valores de columnas en una o más filas de una tabla

\* Cambiar el nombre del departamento D4 por 'FORMACION' y asignarle un director  
**UPDATE** departamento

**SET** nombre = 'FORMACION', nssdire = 444  
    **WHERE** coddep = 'D4';

1° se accede a la tabla  
2° se ejecuta el WHERE  
3° se ejecuta el SET

- **Cláusula WHERE** para seleccionar las filas que actualizar
  - Si no hay WHERE, se aplica la modificación a TODAS las filas
- **Cláusula SET** con las columnas que modificar y los nuevos valores
- NULL o DEFAULT como nuevo valor de una columna

**UPDATE** empleado **SET** nssjefe = NULL  
WHERE apellido = 'FUERTES';

**UPDATE** empleado **SET** salario = DEFAULT  
WHERE nss = 333;

Estos ejemplos de UPDATE  
sólo modifican una fila  
de la tabla, pues sólo una  
de ellas hace TRUE lo  
indicado en el WHERE

# Modificación de varias filas a la vez

159

- Varias filas hacen TRUE la comparación incluida en la WHERE

**UPDATE empleado SET ciudad = 'Murcia'  
WHERE ciudad = 'MURCIA';**

Modificados todos los empleados cuya ciudad era 'MURCIA'

- Inclusión de una **subconsulta** en la cláusula WHERE para seleccionar las filas que modificar

\* Subir un 10% el salario de todo empleado del departamento de Investigación

**UPDATE empleado SET salario = salario\*1.1  
WHERE dep IN ( SELECT coddep  
FROM departamento  
WHERE nombre = 'INVESTIGACION' );**

Modificados todos los empleados del departamento de investigación

- Inclusión de una **consulta SELECT** en la cláusula SET

\* Actualizar el número de familiares que tiene cada empleado

**UPDATE empleado e SET cuantos\_familiares = ( SELECT COUNT(\*)**

Modificados todos los empleados (no hay WHERE)

**FROM familiar  
WHERE nssemp = e.nss );**

# Modificación de varias filas a la vez

160

- Inclusión de una **subconsulta** en la cláusula WHERE y una **consulta SELECT** en la cláusula SET para calcular el valor

\* Asignar a todo director de departamento un salario igual a la media de los salarios de su departamento más el 25% de dicha media

**UPDATE empleado d SET salario = ( SELECT AVG(salario) \* 1.25**

**FROM empleado e  
WHERE e.dep = d.dep )**

1º selecciona sólo las filas de EMPLEADO que son directores

**WHERE nss IN ( SELECT nssdire  
FROM departamento );**

2º para cada una de esas filas,  
calcula la media de su departamento  
y le asigna el nuevo salario

- **Propagación de modificaciones vía claves ajena**
  - **SGBD Oracle NO** permite cambiar un valor de clave candidata si existen claves ajena que le estén haciendo referencia
  - **ANSI SQL sí** permite propagar el cambio a los valores de clave ajena en otras tablas, si se especificó ON UPDATE CASCADE en el CREATE TABLE



# Modificación de varias filas a la vez

160

- Inclusión de una **subconsulta** en la cláusula WHERE y una **consulta SELECT** en la cláusula SET para calcular el valor

\* Asignar a todo director de departamento un salario igual a la media de los salarios de su departamento más el 25% de dicha media

**UPDATE empleado d SET salario = ( SELECT AVG(salario) \* 1.25**

**FROM empleado e  
WHERE e.dep = d.dep )**

1º selecciona sólo las filas de EMPLEADO que son directores

**WHERE nss IN ( SELECT nssdire  
FROM departamento );**

2º para cada una de esas filas,  
calcula la media de su departamento  
y le asigna el nuevo salario

- **Propagación de modificaciones vía claves ajena**
  - **SGBD Oracle NO** permite cambiar un valor de clave candidata si existen claves ajena que le estén haciendo referencia
  - **ANSI SQL sí** permite propagar el cambio a los valores de clave ajena en otras tablas, si se especificó ON UPDATE CASCADE en el CREATE TABLE



# Modificación de varias filas a la vez

160

- Inclusión de una **subconsulta** en la cláusula WHERE y una **consulta SELECT** en la cláusula SET para calcular el valor

\* Asignar a todo director de departamento un salario igual a la media de los salarios de su departamento más el 25% de dicha media

**UPDATE empleado d SET salario = ( SELECT AVG(salario) \* 1.25**

**FROM empleado e  
WHERE e.dep = d.dep )**

1º selecciona sólo las filas de EMPLEADO que son directores

**WHERE nss IN ( SELECT nssdire  
FROM departamento );**

2º para cada una de esas filas,  
calcula la media de su departamento  
y le asigna el nuevo salario

- **Propagación de modificaciones vía claves ajena**
  - **SGBD Oracle NO** permite cambiar un valor de clave candidata si existen claves ajena que le estén haciendo referencia
  - **ANSI SQL sí** permite propagar el cambio a los valores de clave ajena en otras tablas, si se especificó ON UPDATE CASCADE en el CREATE TABLE

