2023

GRUPO STUDIUM ciclos

Tema 5: Temario

CLAÚSULAS AVANZADAS DE SELECCIÓN

STUDIUM

5.1 Introducción

En el presente tema nos centraremos en las instrucciones de **selección de datos**. Una base de datos que solamente guarde información es útil, pero lo es mucho más si podemos sacar dicha información de forma estructurada, ordenada y que nos ayude a tomar decisiones, a valorar situaciones, contrastar resultados, hacer inventarios, etc.

5.2 Instrucción SELECT

Estudiemos en profundidad esta sentencia tan simple, pero a la vez, tan potente.

Estructura

La instrucción con la que vamos a trabajar es **SELECT**. Se expresa como:

```
SELECT Campos FROM Tabla WHERE Condición;
```

Viene a responder a la siguiente estructura:

```
SELECT qué FROM de dónde WHERE cómo
```

Esta instrucción realizará una selección de todos los registros de la tabla indicada, que cumplan la condición que aparece tras el WHERE. Si no encuentra ninguno devolverá Vacío.

De todos los registros que encuentre, solamente cogerá los campos indicados tras la cláusula SELECT. Si queremos que obtenga TODOS los campos, utilizaremos la partícula * para tal fin.

NOTA: *Usaremos la base de datos **Empresa** presentada en el Tema 3*.

Tabla de los **Centros de Trabajo (centrostrabajo)**:

idCentroTrabajo	nombreCentroTrabajo	direccionCentroTrabajo
10	Sede Central	C/Alcalá, 820, Madrid
20	Relación con Clientes	C/Atocha, 405, Madrid

Tabla de los **Departamentos (departamentos)**:

idDepartamento	idCentroTrabajoFK	idEmpleadoFK	tipoDirectorDepartamento	presupuestoDepartamento	idDepartamentoFK	nombreDepartamento
100	10	260	P	12	NULL	DIRECCION GENERAL
110	20	180	P	15	100	DIRECCION COMERCIAL
111	20	180	F	11	110	SECTOR INDUSTRIAL
112	20	270	P	9	110	SECTOR SERVICIOS
120	10	150	F	3	100	ORGANIZACION
121	10	150	P	2	120	PERSONAL
122	10	350	P	6	120	PROCESO DE DATOS
130	10	310	P	2	100	FINANZAS

Tabla **Empleados**:

idEmpleado	idDepartamentoFK	extensionEmpleado	fechaNacimientoEmpleado	fechaIngresoEmpleado	salarioEmpleado	comisionEmpleado	hijosEmpleado	nombreEmpleado
110	121	350	1969-02-05	1990-02-05	2100.00	0.00	3	PONS, CESAR
120	112	840	1975-05-30	2008-09-30	2500.00	1100.00	1	LASA, MARIO
130	112	810	1985-08-30	2009-01-22	1900.00	1100.00	2	TEROL, LUCIANO
150	112	340	1970-07-31	1988-01-05	3400.00	0.00	0	PEREZ, JULIO
160	111	740	1979-06-29	2008-11-01	2100.00	1100.00	2	AGUIRRE, AUREO
180	110	508	1974-10-08	1996-03-08	3800.00	500.00	2	PEREZ, MARCOS
190	121	350	1972-05-02	2002-02-01	2000.00	0.00	4	VEIGA, JULIANA
210	100	740	1980-09-18	1999-01-12	2800.00	0.00	2	GALVEZ, PILAR
240	111	508	1982-02-16	2006-02-14	1800.00	1000.00	3	SANZ, LAVINIA
250	100	200	1986-10-17	2007-02-19	3500.00	0.00	0	ALBA, ADRIANA
260	100	760	1983-09-23	2008-07-02	5200.00	0.00	6	LOPEZ, ANTONIO
270	112	250	1985-05-11	2006-08-31	2800.00	800.00	3	GARCIA, OCTAVIO
280	130	220	1948-01-01	1968-09-28	1900.00	0.00	5	FLOR, DOROTEA
285	122	800	1989-10-15	2009-02-05	2800.00	0.00	0	POLO, OTILIA
290	120	410	1947-11-20	1968-02-04	1700.00	0.00	3	GIL, GLORIA
310	130	620	1986-11-11	2011-02-05	3200.00	0.00	0	GARCIA, AUGUSTO
320	122	910	1947-12-18	1968-01-29	3050.00	0.00	2	SANZ, CORNELIO
330	112	480	1948-08-09	1982-02-20	1800.00	900.00	3	DIEZ, AMELIA
350	112	620	1969-04-08	2004-09-05	3500.00	0.00	1	CAMPS, AURELIO
360	111	850	1998-10-19	2008-09-30	1500.00	100.00	2	LARA, DORINA
370	121	610	1987-06-17	2007-01-15	900.00	0.00	1	RUIZ, FABIOLA
380	112	750	1948-03-25	1967-12-27	800.00	0.00	0	MARTIN, MICAELA
390	110	360	1946-02-14	1966-10-03	1120.00	0.00	1	MORAN, CARMEN
400	111	880	1989-08-13	2007-10-27	850.00	0.00	0	LARA, LUCRECIA
410	122	500	1988-07-09	2008-10-08	750.00	0.00	0	MUÑOZ, AZUCENA
420	130	780	1986-10-17	2008-11-11	3000.00	0.00	0	FIERRO, CLAUDIA
430	122	660	1947-02-21	2020-10-28	1100.00	0.00	1	MORA, VALERIANA
440	111	450	1986-09-21	2006-02-03	1100.00	1000.00	0	DURAN, LIVIA
450	112	650	1986-10-16	2006-02-23	1100.00	1000.00	0	PEREZ, SABINA
480	111	760	1985-03-30	2006-02-23	1100.00	1000.00	1	PINO, DIANA
490	112	880	1984-06-01	2007-12-27	800.00	1000.00	0	TORRES, HORACIO
500	111	750	1985-10-03	2006-12-27	1000.00	1000.00	0	VAZQUEZ, HONO...
510	110	550	1986-04-26	2006-10-27	1000.00	0.00	1	CAMPOS, ROMULO
550	111	780	1990-01-05	2008-01-16	900.00	1200.00	0	SANTOS, SANCHO

Vamos a seleccionar TODOS los registros de la tabla Departamentos, mostrando TODOS los campos:

```
SELECT
        *
FROM
        departamentos;
```

idDepartamento	idCentroTrabajoFK	idEmpleadoFK	tipoDirectorDepartamento	presupuestoDepartamento	idDepartamentoFK	nombreDepartamento
100	10	260	P	12	NULL	DIRECCION GENERAL
110	20	180	P	15	100	DIRECCION COMERCIAL
111	20	180	F	11	110	SECTOR INDUSTRIAL
112	20	270	P	9	110	SECTOR SERVICIOS
120	10	150	F	3	100	ORGANIZACION
121	10	150	P	2	120	PERSONAL
122	10	350	P	6	120	PROCESO DE DATOS
130	10	310	P	2	100	FINANZAS

Haremos la misma consulta pero que muestre solamente los campos idDepartamento y nombreDepartamento:

```
SELECT
        idDepartamento, nombreDepartamento
FROM
        departamentos;
```

idDepartamento	nombreDepartamento
100	DIRECCION GENERAL
110	DIRECCION COMERCIAL
111	SECTOR INDUSTRIAL
112	SECTOR SERVICIOS
120	ORGANIZACION
121	PERSONAL
122	PROCESO DE DATOS
130	FINANZAS

Alias

Cuando se visualiza el resultado de la consulta, normalmente las columnas toman el nombre que tiene la columna en la tabla; si queremos cambiar ese nombre lo podemos hacer definiendo un **alias** de columna mediante la cláusula AS indicando el nombre que queremos que aparezca como título de la columna de la consulta.

Vamos a repetir la última consulta, pero dando un nombre más "humano" a las columnas:

```
SELECT
        idDepartamento AS 'Nº departamento',
        nombreDepartamento AS 'Nombre del departamento'
FROM
        departamentos;
```

Nº departamento	Nombre del departamento
100	DIRECCION GENERAL
110	DIRECCION COMERCIAL
111	SECTOR INDUSTRIAL
112	SECTOR SERVICIOS
120	ORGANIZACION
121	PERSONAL
122	PROCESO DE DATOS
130	FINANZAS

Varias puntualizaciones:

1. Si el alias es una única palabra, no son necesarias las comitas; pero si tenemos varias palabras, las comitas son **obligatorias**. Pero se recomienda usar SIEMPRE las comitas.
2. En el alias podemos usar tildes, eñes, espacios en blanco y todos los caracteres NO permitidos en los nombres de tablas o de campos.
3. También podemos dar nombre con el alias al resultado de operaciones, como veremos más adelante.
4. Los alias también se pueden aplicar a Tablas. También se estudiará en su momento.
5. La partícula AS se puede omitir.

Condiciones

Las **condiciones** sirven para delimitar los resultados obtenidos. Hasta ahora hemos visto consultas en las que se obtienen todos los registros; pero esto no es lo habitual. Lo normal es querer obtener un subconjunto del total. Y esto se consigue con las condiciones. Para ello usaremos la cláusula WHERE en los siguientes ejemplos ilustrativos.

Seleccionar el nombre de los empleados que trabajan en el departamento 121:

```
SELECT
        nombreEmpleado AS 'Empleado'
FROM
        empleados
WHERE
        idDepartamentoFK = 121;
```

Empleado
PONS, CESAR
VEIGA, JULIANA
RUIZ, FABIOLA

Seleccionar los empleados cuya comisión sea de 1000 €:

```
SELECT
        nombreEmpleado AS 'Empleado'
FROM
        empleados
WHERE
        comisionEmpleado = 1000;
```

Empleado
SANZ, LAVINIA
DURAN, LIVIA
PEREZ, SABINA
PINO, DIANA
TORRES, HORACIO
VAZQUEZ, HONORIA

Filtro LIKE

En ocasiones, puede ser interesante usar una condición más genérica. En estos casos podemos hacer uso de los filtros con la partícula LIKE. Se aplica sobre campos de tipo cadena o fecha. En el siguiente ejemplo seleccionamos TODOS los registros de la tabla empleados cuyos nombres acaben a la letra O:

```
SELECT
        nombreEmpleado AS 'Empleado'
FROM
        empleados
WHERE
        nombreEmpleado LIKE '%O';
```

Empleado
LASA, MARIO
TEROL, LUCIANO
PEREZ, JULIO
AGUIRRE, AUREO
LOPEZ, ANTONIO
GARCIA, OCTAVIO
GARCIA, AUGUSTO
SANZ, CORNELIO
CAMPS, AURELIO
TORRES, HORACIO
CAMPOS, ROMULO
SANTOS, SANCHO

El signo % equivale a cualquier cantidad de cualquier carácter.
Ahora, obtendremos un listado de los empleados cuyo apellido acabe por ‘ez’:

```
SELECT
        nombreEmpleado AS 'Empleado'
FROM
        empleados
WHERE
        nombreEmpleado LIKE '%EZ,%';
```

Empleado
PEREZ, JULIO
PEREZ, MARCOS
GALVEZ, PILAR
LOPEZ, ANTONIO
DIEZ, AMELIA
PEREZ, SABINA
VAZQUEZ, HONORIA

NOTA: El filtro LIKE diferencia entre minúsculas y MAYÚSCULAS.

También tenemos el carácter subrayado (_) que equivale a un único carácter. Para ver cómo funciona, obtendremos un listado de los empleados cuyo nombre acabe en A y que tengan exactamente 6 caracteres incluyendo la letra A:

```
SELECT
        nombreEmpleado AS 'Empleado'
FROM
        empleados
WHERE
        nombreEmpleado LIKE '%,_____A';
```

Observar que se han incluido 5 subrayados más la letra A.
Aquí hay algunos ejemplos más:

- 'A_Z': Toda línea que comience con 'A', otro carácter y termine con 'Z'. Por ejemplo, 'ABZ' y 'A2Z' deberían satisfacer la condición, mientras 'AKKZ' no debería (debido a que hay dos caracteres entre A y Z en vez de uno).
- 'ABC%': Todas las líneas que comienzan con 'ABC'. Por ejemplo, 'ABCD' y 'ABCABC' ambas deberían satisfacer la condición.
- '%XYZ': Todas las líneas que terminan con 'XYZ'. Por ejemplo, 'WXYZ' y 'ZZXYZ' ambas deberían satisfacer la condición.
- '%AN%': Todas las líneas que contienen el patrón 'AN' en cualquier lado. Por ejemplo, 'LOS ANGELES' y 'SAN FRANCISCO' ambos deberían satisfacer la condición.

Cláusula DISTINCT

Al incluir la cláusula **DISTINCT** en una sentencia SELECT, se eliminan del resultado las repeticiones de filas. Si por el contrario queremos que aparezcan todas las filas incluidas las duplicadas, podemos incluir la cláusula **ALL** o nada, ya que ALL es el valor que SQL asume por defecto.
Con la sentencia siguiente se nos muestran todas las extensiones telefónicas de todos los empleados:

```
SELECT
        extensionEmpleado AS ‘Teléfono’
FROM
        empleados;
```

Se obtiene el listado que aparece en el lateral. Si somos observadores, veremos que hay extensiones telefónicas repetidas. Para evitarlas, usaremos la cláusula DISTINCT

```
SELECT
        DISTINCT extensionEmpleado AS ‘Teléfono’
FROM
        empleados;
```

Teléfono
350
840
810
340
740
508
350
740
508
200
760
250
220
800
410
620
910
480
620
850
610
750
360
880
500
780
660
450
650
760
880
750
550
780

Teléfono
350
840
810
340
740
508
200
760
250
220
800
410
620
910
480
850
610
750
360
880
500
780
660
450
650
550

Ya no aparecen los valores duplicados.

Ordenación

Con la cláusula **ORDER BY** podemos establecer la forma en que queremos que nos aparezca ordenada la información que obtenemos con los SELECT. En ningún caso se altera la propia tabla, solamente los resultados obtenidos con el SELECT.

Si no incluimos esta instrucción, los resultados mostrados se ordenarán por los **índices** creados en la tabla y en su defecto, por el **campo** o **campos clave**.

Con la instrucción ORDER BY podemos indicar el nombre de un campo o el orden en que se encuentra dicho campo dentro de la lista de selección (Tras el SELECT):

```
SELECT
        nombreEmpleado AS 'Empleado',
        salarioEmpleado AS 'Salario'
FROM
        empleados
ORDER BY salarioEmpleado;
```

Empleado	Salario
MUÑOZ, AZUCENA	750.00
TORRES, HORACIO	800.00
MARTIN, MICAELA	800.00
LARA, LUCRECIA	850.00
SANTOS, SANCHO	900.00
RUIZ, FABIOLA	900.00
CAMPOS, ROMULO	1000.00

Pero también:

```
SELECT
        nombreEmpleado AS 'Empleado',
        salarioEmpleado AS 'Salario'
FROM
        empleados
ORDER BY 2;
```

Por defecto el orden será **ascendente** (ASC) (de menor a mayor si el campo es numérico, por orden alfabético si el campo es de tipo texto, de anterior a posterior si el campo es de tipo fecha/hora, etc....) Pero también podemos especificar el orden contrario con **DESC**.

La misma sentencia, pero en orden descendente:

```
SELECT
        nombreEmpleado AS 'Empleado',
        salarioEmpleado AS 'Salario'
FROM
        empleados
ORDER BY salarioEmpleado DESC;
```

Empleado	Salario
LOPEZ, ANTONIO	5200.00
PEREZ, MARCOS	3800.00
ALBA, ADRIANA	3500.00
CAMPS, AURELIO	3500.00
PEREZ, JULIO	3400.00
GARCIA, AUGUSTO	3200.00

También podemos ordenar por varias columnas, en este caso se indican las columnas separadas por comas. Se ordenan las filas por la primera columna de

ordenación, pero para un mismo valor de la primera columna, se ordenan por la segunda columna, y así sucesivamente.

Las cláusulas DESC o ASC se pueden indicar para cada columna y así utilizar una ordenación distinta para cada columna. Por ejemplo, ascendente por la primera columna y para valores iguales de la primera columna, descendente por la segunda columna.

Cláusula LIMIT

La cláusula **LIMIT** permite sacar las n primeras filas de la tabla sobre la que estamos trabajando. No elige entre valores iguales, si pido los 25 primeros valores, pero el que hace 26 es el mismo valor que el 25, entonces devolverá 26 registros en vez de 25 (o los que sea). Siempre se guía por la columna de ordenación, la que aparece en la cláusula ORDER BY o en su defecto la clave principal de la tabla.

Por ejemplo, queremos saber los dos empleados más antiguos de la empresa:

```
SELECT
        idEmpleado AS 'Nº Empleado',
        nombreEmpleado AS 'Nombre'
FROM
        empleados
ORDER BY fechaIngresoEmpleado
LIMIT 2;
```

Nº Empleado	Nombre
390	MORAN, CARMEN
380	MARTIN, MICAELA

5.3 Operadores

Los operadores son signos que nos permiten realizar operaciones sencillas con los valores de los campos o incluso establecer condiciones en las cláusulas WHERE de forma que podamos delimitar correctamente las búsquedas.

Operadores aritméticos

Los operadores **aritméticos** sirven para realizar las operaciones básicas siguientes:

+	Suma
-	Resta
*	Multiplicación
/	División

Obtengamos los nombres de los empleados junto al salario total, es decir, salario base más comisiones:

```
SELECT
        idEmpleado AS 'Nº Empleado',
        nombreEmpleado AS 'Nombre',
        (salarioEmpleado+comisionEmpleado) AS 'Salario Total'
FROM
        empleados;
```

Nº Empleado	Nombre	Salario Total
110	PONS, CESAR	2100.00
120	LASA, MARIO	3600.00
130	TEROL, LUCIANO	3000.00
150	PEREZ, JULIO	3400.00
160	AGUIRRE, AUREO	3200.00
180	PEREZ, MARCOS	4300.00
190	VEIGA, JULIANA	2000.00
210	GALVEZ, PILAR	2800.00

Operadores de comparación

Los operadores de **comparación** vinculan un campo con un valor para que SQL compare cada registro (el campo especificado) con el valor dado.

Los operadores de comparación son los siguientes:

=	Igual
<> !=	Distinto
>	Mayor
<	Menor
>=	Mayor o igual
<=	Menor o igual

Listar los empleados cuyo salario supere los 2500 €:

```
SELECT
        idEmpleado AS 'Nº Empleado',
        nombreEmpleado AS 'Nombre'
FROM
        empleados
WHERE
        salarioEmpleado>2500;
```

Nº Empleado	Nombre
150	PEREZ, JULIO
180	PEREZ, MARCOS
210	GALVEZ, PILAR
250	ALBA, ADRIANA
260	LOPEZ, ANTONIO
270	GARCIA, OCTAVIO
285	POLO, OTILIA
310	GARCIA, AUGUSTO
320	SANZ, CORNELIO
350	CAMPS, AURELIO
420	FIERRO, CLAUDIA

Operadores lógicos

Los operadores lógicos sirven para aplicar más de una condición en la cláusula WHERE. También existe uno para negar la condición al completo.

AND	Y lógico	Se deben cumplir AMBAS condiciones
OR	O lógico	Se debe cumplir al MENOS una condición
NOT	No lógico	Se niega la condición
!	No lógico	Se niega la condición

Las condiciones pueden ser más de una, relacionadas mediante los operadores lógicos OR (O), AND (Y) o NO (NOT).

Seleccionaremos ahora los empleados cuya comisión sea 1000€ o 500 €:

```
SELECT
        nombreEmpleado AS 'Empleado'
FROM
        empleados
WHERE
        comisionEmpleado = 1000
OR
        comisionEmpleado = 500;
```

Empleado
PEREZ, MARCOS
SANZ, LAVINIA
DURAN, LIVIA
PEREZ, SABINA
PINO, DIANA
TORRES, HORACIO
VAZQUEZ, HONORIA

Seleccionar los empleados que tengan comisión de 1000€ y pertenezcan al departamento 111:

```
SELECT
        nombreEmpleado AS 'Empleado'
FROM
        empleados
WHERE
        comisionEmpleado = 1000
AND
        idDepartamentoFK = 111;
```

Empleado
SANZ, LAVINIA
DURAN, LIVIA
PINO, DIANA
VAZQUEZ, HONORIA

Utilizando el AND para concatenar condiciones, la condición completa será verdadera si TODAS lo son.

Sin embargo, con el OR, en cuanto una condición de las que forman la concatenación sea verdadera, la expresión completa lo será también.

Por último, tenemos el NOT que consiste en la negación lógica. Devuelve el valor contrario de la expresión evaluada.

Si aparecen combinados los operadores AND, NOT y OR, la prioridad de evaluación es NOT, luego AND y por último OR. Para evitar errores, haced uso intensivo de paréntesis y así forzamos nosotros el orden de evaluación. No es lo mismo *condicion1 AND condicion2 OR condicion3* que *condicion1 AND (condicion2 OR condicion3)*. Por ejemplo, si tenemos por un lado *Hombres* y *Mujeres* y por otro lado *Altos* y *Bajos*, estas dos comparaciones no obtienen el mismo resultado:

- Hombres O Mujeres Y Altos → Obtiene todos los Hombres (altos y bajitos) o las Mujeres altas.
- (Hombres O Mujeres) Y Altos → Obtiene tanto los Hombres altos o todas las Mujeres altas. Los Hombres bajos ahora no se obtienen

5.4 Comprobaciones con conjunto de valores

En SQL, hay dos usos de la palabra clave **IN**, y esta sección introduce el relacionado con la cláusula WHERE. Cuando se lo utiliza en este contexto, sabemos exactamente el valor de los valores devueltos que deseamos ver para al menos una de las columnas. La sintaxis para el uso de la palabra clave IN es la siguiente:

```
SELECT
        "nombre_columna"
FROM
        "nombre_tabla"
WHERE
        "nombre_columna" IN ('valor1', 'valor2',...)
```

El número de valores en los paréntesis pueden ser uno o más, con cada valor separado por comas. Los valores pueden ser números o caracteres. Si hay sólo un valor dentro del paréntesis, este comando es equivalente a

```
WHERE
        "nombre_columna" = 'valor1'
```

Por ejemplo, la siguiente sentencia mostrará los empleados de los departamentos 100, 112 y 120:

```
SELECT
        idEmpleado AS 'Nº Empleado',
        nombreEmpleado AS 'Nombre'
FROM
        empleados
WHERE
        idDepartamentoFK IN (100, 112, 120);
```

Nº Empleado	Nombre
210	GALVEZ, PILAR
250	ALBA, ADRIANA
260	LOPEZ, ANTONIO
120	LASA, MARIO
130	TEROL, LUCIANO
150	PEREZ, JULIO
270	GARCIA, OCTAVIO
330	DIEZ, AMELIA
350	CAMPS, AURELIO
380	MARTIN, MICAELA
450	PEREZ, SABINA
490	TORRES, HORACIO
290	GIL, GLORIA

La palabra clave **BETWEEN** permite la selección de un rango. La sintaxis para la cláusula BETWEEN es la siguiente:

```
SELECT
        "nombre_columna"
FROM
        "nombre_tabla"
WHERE
        "nombre_columna" BETWEEN 'valor1' AND 'valor2'
```

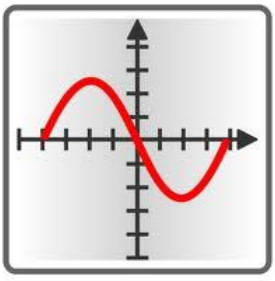
Esto seleccionará todas las filas cuya columna tenga un valor entre 'valor1' y 'valor2'. Los valores pueden ser números, textos o fechas.

Seleccionar TODOS los registros de la tabla empleados cuyas fechas de ingreso sean en el año 2006:

```
SELECT
        idEmpleado AS 'Nº Empleado',
        nombreEmpleado AS 'Nombre'
FROM
        empleados
WHERE
        fechaIngresoEmpleado BETWEEN '2006/01/01' AND '2006/12/31';
```

Nº Empleado	Nombre
240	SANZ, LAVINIA
270	GARCIA, OCTAVIO
440	DURAN, LIVIA
450	PEREZ, SABINA
480	PINO, DIANA
500	VAZQUEZ, HONORIA
510	CAMPOS, ROMULO

5.5 Funciones



En este apartado veremos una serie de funciones que se pueden utilizar para el tratamiento de los datos obtenidos de una base de datos en una consulta SELECT y realizar consultas más precisas y con valor añadido con respecto al dato simple guardado en la base de datos.

La sintaxis para el uso de las funciones es:

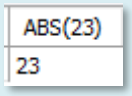
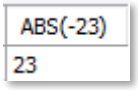
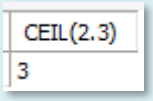
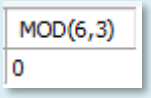
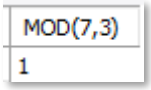
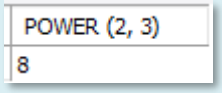
```
SELECT nombre_función(nombre_columna) FROM nombre_tabla WHERE condición;
```

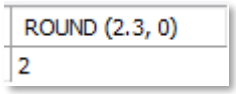
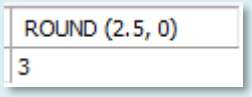
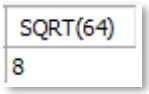
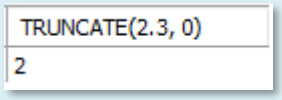
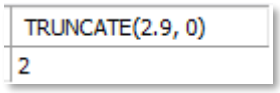
donde el nombre de la función será una de las siguientes.

Funciones de valores simples

Función	Efecto
ABS(n)	Devuelve el valor absoluto de (n)
CEIL(n)	Obtiene el valor entero inmediatamente superior o igual a "n"
FLOOR(n)	Devuelve el valor entero inmediatamente inferior o igual a "n"
MOD (m, n)	Devuelve el resto resultante de dividir "m" entre "n"
NVL (valor, expresión)	Sustituye un valor nulo por otro valor. Nos permite trabajar con operaciones matemáticas, aunque haya valores NULL
POWER (m, exponente)	Calcula la potencia de un número.
ROUND (numero [, m])	Redondea números con el número de dígitos de precisión indicados.
SIGN (valor)	Indica el signo del "valor"
SQRT(n)	Devuelve la raíz cuadrada de "n"
TRUNCATE(numero, [m])	Trunca números para que tengan una cierta cantidad de dígitos de precisión

Estas funciones se suelen usar con datos de tablas; pero también se pueden probar “sueltas”. Veremos, de momento, de este tipo.

Sentencia	Resultado
SELECT ABS(23);	
SELECT ABS(-23);	
SELECT CEIL(2.3);	
SELECT FLOOR(2.3);	
SELECT MOD (6, 3);	
SELECT MOD (7, 3);	
SELECT POWER (2, 3);	

SELECT ROUND (2.3, 0);	
SELECT ROUND (2.5, 0);	
SELECT SIGN (2);	
SELECT SIGN (-2);	
SELECT SQRT(64);	
SELECT TRUNCATE(2.3, 0);	
SELECT TRUNCATE(2.9, 0);	

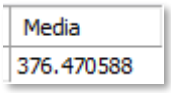
Funciones de grupos de valores

Función	Efecto
AVG(n)	Calcula el valor medio de "n" ignorando los valores nulos
COUNT (* | Expresión)	Cuenta el número de veces que la expresión evalúa algún dato con valor no nulo. La opción "*" cuenta todas las filas seleccionadas
MAX (expresión)	Calcula el máximo
MIN (expresión)	Calcula el mínimo
STDDEV (valor1, valor 2, ...)	Calcula de desviación típica de los valores sin tener en cuenta los nulos
SUM (expresión)	Obtiene la suma de los valores de la expresión
VARIANCE (valor1, valor2...)	Devuelve la varianza de un conjunto de valores
GREATEST (valor1, valor2...)	Obtiene el mayor valor de la lista. Dentro del () no funciona con subconsulta
LEAST (valor1, valor2...)	Obtiene el menor valor de la lista. Dentro del () no funciona con subconsulta

Ejemplos:

AVG: Calcular la media de las comisiones:

```
SELECT AVG(comisionEmpleado) AS 'Media' FROM empleados;
```



COUNT: Contar el número de Empleados del departamento 100:

```
SELECT COUNT(*) AS 'Contador' FROM empleados WHERE idDepartamentoFK = 100;
```

Contador
3

COUNT y DISTINCT pueden utilizarse juntos en una instrucción para determinar el número de las distintas entradas en una tabla. Por ejemplo, si deseamos saber el número de las distintas extensiones, ingresaríamos:

Nº Extensiones
26

```
SELECT COUNT(DISTINCT extensionEmpleado) AS 'Nº Extensiones' FROM Empleados;
```

MAX: Obtener la comisión máxima:

```
SELECT MAX(comisionEmpleado) AS 'Máximo' FROM empleados;
```

Máximo
1200.00

MIN: Obtener la comisión mínima:

```
SELECT MIN(comisionEmpleado) AS 'Mínimo' FROM empleados;
```

Mínimo
0.00

SUM: Dar el total de salarios del departamento 130:

```
SELECT SUM(salarioEmpleado) AS 'Suma' FROM empleados WHERE idDepartamentoFK = 130;
```

Suma
8100.00

Funciones que devuelven cadenas de caracteres

Función	Efecto
CONCAT (cad1, cad2)	Devuelve "cad1" concatenada con "cad2"
LOWER (cad)	Devuelve la cadena "cad" en minúsculas
UPPER (cad)	Devuelve la cadena "cad" en mayúsculas
LPAD (campo, n, car)	Añade los caracteres indicados por *car* a la izquierda del campo hasta que tiene una longitud de n
RPAD (campo, n, car)	Añade los caracteres indicados por *car* a la derecha del campo hasta que tiene una longitud de n
REPLACE (campo, cadena_busqueda, cadena_sustitucion)	Sustituye en el campo los caracteres de cadena_busqueda por cadena_sustitucion
SUBSTR (cad, m, n)	Obtiene de la cadena cad a partir de la posición m, n caracteres

Ejemplos:

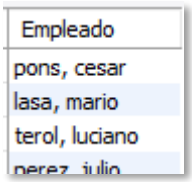
CONCAT: Mostrar los salarios de los empleados junto a símbolo del Euro:

```
SELECT CONCAT(salarioEmpleado, ' €') AS 'Salario' FROM empleados;
```

Salario
2100.00 €
2500.00 €
1900.00 €
3400.00 €

LOWER: Mostrar todos los empleados en minúsculas:

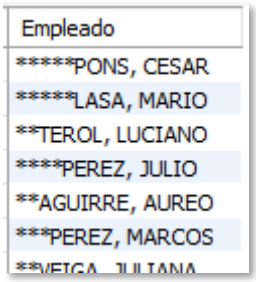
```
SELECT LOWER(nombreEmpleado) AS 'Empleado' FROM empleados;
```



Empleado
pons, cesar
lasa, mario
terol, luciano
perez, julio

LPAD: Completar con asteriscos por la izquierda, los nombres de los empleados hasta una longitud de 16 caracteres:

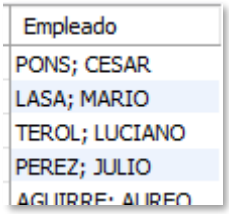
```
SELECT LPAD(nombreEmpleado, 16, '*') AS 'Empleado' FROM empleados;
```



Empleado
*****PONS, CESAR
*****LASA, MARIO
**TEROL, LUCIANO
****PEREZ, JULIO
**AGUIRRE, AUREO
***PEREZ, MARCOS
**VEIGA, JULIANA

REPLACE: Reemplazar la coma del nombre de los empleados que separa el nombre del apellido por un signo de punto y coma:

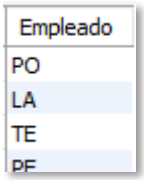
```
SELECT REPLACE(nombreEmpleado, ',', ';') AS 'Empleado' FROM empleados;
```



Empleado
PONS; CESAR
LASA; MARIO
TEROL; LUCIANO
PEREZ; JULIO
AGUIRRE; AUREO

SUBSTR: Obtener los dos primeros caracteres de los apellidos de los empleados:

```
SELECT SUBSTR(nombreEmpleado, 1, 2) AS 'Empleado' FROM empleados;
```



Empleado
PO
LA
TE
PE

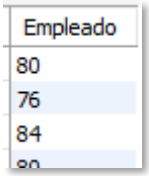
Funciones que devuelven valores numéricos

Función	Efecto
ASCII(cad)	Devuelve el valor ASCII de la primera letra del campo indicado
LENGTH (campo)	Devuelve el número de caracteres del campo indicado

Ejemplos:

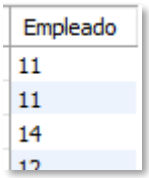
ASCII: Obtener los valores ASCII de la primera letra del apellido de cada empleado:

```
SELECT ASCII(nombreEmpleado) AS 'Empleado' FROM empleados;
```



LENGTH: Obtener la longitud del nombre completo de cada empleado, incluyendo la coma y el espacio que separa el apellido del nombre:

```
SELECT LENGTH(nombreEmpleado) AS 'Empleado' FROM empleados;
```



Funciones para el manejo de fechas y horas

Función	Efecto
CURDATE()	Devuelve la fecha actual
CURTIME()	Devuelve la hora actual
NOW()	Devuelve el timestamp (fecha+hora) actual
YEAR(fecha)	Devuelve el año de la fecha indicada
MONTH(fecha)	Devuelve el mes de la fecha indicada
DAY(fecha)	Devuelve el día de la fecha indicada
LAST_DAY(fecha)	Devuelve el último día del mes de la fecha indicada
DATE_ADD(fecha, intervalo)	A la fecha indicada se le suma el intervalo correspondiente para obtener otra fecha. En el *intervalo* podemos especificar: • MINUTE • HOUR • DAY • MONTH • YEAR
DATE_SUB(fecha, intervalo)	A la fecha indicada se le resta el intervalo correspondiente para obtener otra fecha. El intervalo es el mismo que DATE_ADD
DATEDIFF(fecha1, fecha2)	Obtiene los días entre las dos fechas indicadas
DATE_FORMAT(fecha, formato)	Devuelve la fecha con el formato indicado según los siguientes valores: • %d #Día del mes numérico (00...31) • %H #Hora (00...23) • %h #Hora (01...12) • %i #Minutos, numérico (00...59) • %M #Nombre mes (January...December) • %m #Mes, numérico (00...12) • %p #AM o PM • %W #Día semana (Sunday...Saturday) • %Y #Año, numérico, cuatro dígitos • %y #Año, numérico (dos dígitos) • %s #Segundos (00...59)

Ejemplos:

CURDATE: Obtener la fecha actual:

```
SELECT CURDATE();
```

CURDATE()
2021-11-12

CURTIME: Obtener la hora actual:

```
SELECT CURTIME();
```

CURTIME()
10:46:42

NOW: Obtener el timestamp actual:

```
SELECT NOW();
```

NOW()
2021-11-12 10:47:54

YEAR: Obtener el año actual:

```
SELECT YEAR(CURDATE());
```

YEAR(CURDATE())
2021

MONTH: Obtener el mes actual:

```
SELECT MONTH(CURDATE());
```

MONTH(CURDATE())
11

DAY: Obtener el día de hoy:

```
SELECT DAY(CURDATE());
```

DAY(CURDATE())
12

LAST_DAY: Obtener el último día del mes actual:

```
SELECT LAST_DAY(CURDATE());
```

LAST_DAY(CURDATE())
2021-11-30

DATE_ADD: Obtener qué fecha será dentro de 20 días:

```
SELECT DATE_ADD(NOW(),INTERVAL 20 DAY);
```

DATE_ADD(NOW(),INTERVAL 20 DAY)
2021-12-02 11:33:01

DATE_SUB: Obtener qué fecha fue hace 25 años:

```
SELECT DATE_SUB(NOW(),INTERVAL 25 YEAR);
```

DATE_SUB(NOW(),INTERVAL 25 YEAR)
1996-11-12 11:38:04

DATEDIFF: Obtener los días que han pasado desde primero de año:

```
SELECT DATEDIFF(NOW(),'2021-01-01');
```

DATEDIFF(NOW(),'2021-01-01')
315

DATE_FORMAT: Obtener la fecha de hoy en formato europeo:

```
SELECT DATE_FORMAT(NOW(),'%d/%m/%Y') AS 'Hoy';
```

Hoy
12/11/2021

5.6 Agrupaciones

Ahora seguimos avanzando con las **funciones de agregados**. ¿Recuerdas que utilizamos la palabra clave SUM para calcular el total de los salarios del departamento 130? ¿Y si quisiéramos calcular el total de salarios por cada departamento? Entonces, necesitamos hacer dos cosas: Primero, necesitamos asegurarnos de que hayamos seleccionado el nombre del departamento, así como también los salarios totales. Segundo, debemos asegurarnos de que todas las sumas de los salarios estén en la cláusula **GROUP BY**. La sintaxis SQL correspondiente es,

```
SELECT
        "nombre1_columna",
        SUM("nombre2_columna")
FROM
        “nombre_tabla"
GROUP BY "nombre1-columna";
```

Calculemos el salario total que hay que pagar cada mes en la empresa:

```
SELECT
        SUM(salarioEmpleado+comisionEmpleado) AS 'Salario Total Mensual'
FROM
        empleados;
```

Salario Total Mensual
81670.00

Ahora, calcularemos lo mismo, pero por departamento:

```
SELECT
        idDepartamentoFK AS 'Departamento',
        SUM(salarioEmpleado+comisionEmpleado) AS 'Salario Total Mensual'
FROM
        empleados
GROUP BY idDepartamentoFK;
```

Departamento	Salario Total Mensual
100	11500.00
110	6420.00
111	16750.00
112	24500.00
120	1700.00
121	5000.00
122	7700.00
130	8100.00

La palabra clave GROUP BY se utiliza cuando estamos seleccionando columnas múltiples desde una tabla (o tablas) y aparece al menos un operador aritmético en la instrucción SELECT. Cuando esto sucede, necesitamos usar GROUP BY con todas las otras columnas seleccionadas, es decir, todas las columnas excepto aquella(s) que se operan por un operador aritmético. En nuestro caso, si hacemos SUM(salarioEmpleado+comisionEmpleado) no podemos hacer también GROUP BY (salarioEmpleado+comisionEmpleado). Tiene que ser otro campo o campos.

Otra cosa que la gente puede querer hacer es limitar el resultado según la suma correspondiente (o cualquier otra función de agregado). Por ejemplo, podríamos desear ver sólo los departamentos con salarios totales mayores a 10000 €. En vez de utilizar la cláusula WHERE, usaremos la cláusula HAVING, que se reserva para funciones de agregados. La cláusula HAVING se coloca generalmente cerca del final de la instrucción SQL, y la instrucción SQL con la cláusula HAVING puede o no incluir la cláusula GROUP BY.

La sintaxis para HAVING es,

```
SELECT
        "nombre1_columna",
        SUM("nombre2_columna")
FROM
        "nombre_tabla"
GROUP BY
        "nombre1_columna"
HAVING
        (condición de función aritmética);
```

Nota: La cláusula **GROUP BY** es opcional.

Para nuestro ejemplo:

```
SELECT
        idDepartamentoFK AS 'Departamento',
        SUM(salarioEmpleado+comisionEmpleado) AS 'Salario Total Mensual'
FROM
        empleados
GROUP BY
        idDepartamentoFK
HAVING
        SUM(salarioEmpleado+comisionEmpleado) > 10000;
```

Departamento	Salario Total Mensual
100	11500.00
111	16750.00
112	24500.00

5.7 Subconsultas

Ahora veremos cómo realizar consultas de varias tablas a la vez. Las **subconsultas** consisten en anidar una consulta SELECT dentro de otra sentencia SELECT o bien sacar datos de dos tablas relacionadas. En este apartado veremos las primeras y en el apartado siguiente veremos el segundo caso.

Si usamos la subconsulta para una **comparación**, dicha subconsulta debe devolver una **única columna**, si no, se produce un error. Si la subconsulta no produce **ninguna fila** o devuelve el valor **nulo**, el test devuelve el **valor nulo**, si la subconsulta produce **varias filas**, SQL devuelve una **condición de error**. Pero se puede trabajar con un SELECT subordinado que obtenga más de un valor haciendo uso de ALL, ANY y SOME. El SELECT subordinado devolverá una sola columna con cero o más registros, y dependiendo del ALL, ANY o SOME obtendremos un resultado y otro.

Ejemplo: Obtener un listado con los compañeros de departamento de Aurelio Camps:

```
SELECT
        nombreEmpleado AS 'Compañeros'
FROM
        empleados
WHERE
        idDepartamentoFK = (SELECT
                                idDepartamentoFK
                            FROM
                                empleados
                            WHERE
                                nombreEmpleado = 'CAMPS, AURELIO'
                           );
```

Compañeros
LASA, MARIO
TEROL, LUCIANO
PEREZ, JULIO
GARCIA, OCTAVIO
DIEZ, AMELIA
CAMPS, AURELIO
MARTIN, MICAELA
PEREZ, SABINA
TORRES, HORACIO

¿Cómo se hace para que el propio Aurelio no aparezca?

Ejemplo: Obtener todos los empleados con salarios superiores al de Aurelio Camps:

```
SELECT
        nombreEmpleado AS 'Compañeros',
        salarioEmpleado AS 'Salario'
FROM
        empleados
WHERE
        salarioEmpleado > (SELECT
                                salarioEmpleado
                        FROM
                                empleados
                        WHERE
                                nombreEmpleado = 'CAMPS, AURELIO'
                        );
```

Compañeros	Salario
PEREZ, MARCOS	3800.00
LOPEZ, ANTONIO	5200.00

También podemos usar la subconsulta para **comparar** el valor de la expresión con cada uno de los valores producidos por la subconsulta. La subconsulta debe devolver una única columna si no, se produce un error. Tenemos el test ANY (algún, alguno en inglés) y el test ALL (todos en inglés).

ANY: Se usa en comparaciones (>, <, =, >=, ...). La expresión = ANY equivale a IN, es lo mismo. La subconsulta debe devolver una única columna, si no, se produce un error. Se evalúa la comparación con cada valor devuelto por la subconsulta. Si alguna de las comparaciones individuales produce el resultado verdadero, el test ANY devuelve el resultado **verdadero**. Si la subconsulta no devuelve ningún valor, el test ANY devuelve **falso**. Si el test de comparación es falso para todos los valores de la columna, ANY devuelve **falso**. Si el test de comparación no es verdadero para ningún valor de la columna, y es nulo para al menos alguno de los valores, ANY devuelve **nulo**.

Por ejemplo, vamos a obtener los empleados del departamento 112 si en él hay algún empleado que cobre más de 3000 €:

```
SELECT
        nombreEmpleado AS 'Empleado',
        salarioEmpleado AS 'Salario'
FROM
        empleados
WHERE
        salarioEmpleado >= ANY (SELECT
                                        salarioEmpleado
                                FROM
                                        empleados
                                WHERE
                                        idDepartamentoFK = 112
                                AND
                                        salarioEmpleado > 3000
                                );
```

Empleado	Salario
PEREZ, JULIO	3400.00
PEREZ, MARCOS	3800.00
ALBA, ADRIANA	3500.00
LOPEZ, ANTONIO	5200.00
CAMPS, AURELIO	3500.00

En este caso la subconsulta devuelve varias columnas con los salarios de los empleados del departamento 112 con un salario mayor a 3000€. Es por ello, por lo que lista los empleados con salario igual o superior a **alguno** de los salarios obtenidos.

La misma sentencia sobre el departamento 111 no ofrece ningún resultado pues el SELECT interior no encuentra ningún valor válido (todos los empleados de dicho departamento cobran menos de 3000€).

ALL: Se usa en comparaciones (>, <, =, >=, ...). La expresión != ALL es lo mismo que NOT IN. La subconsulta debe devolver una única columna, si no se produce un error. Se evalúa la comparación con cada valor devuelto por la subconsulta. Si TODAS las comparaciones individuales, producen un resultado verdadero, el test devuelve el valor **verdadero**. Si la subconsulta no devuelve ningún valor, el test ALL devuelve el valor **verdadero** (¡Ojo con esto!). Si el test de comparación es falso para algún valor de la columna, el resultado es **falso**. Si el test de

comparación no es falso para ningún valor de la columna, pero es nulo para alguno de esos valores, el test ALL devuelve valor **nulo**.

Si la consulta anterior la realizamos ahora con ALL en vez de ANY, obtenemos algo ligeramente diferente:

```
SELECT
        nombreEmpleado AS 'Empleado',
        salarioEmpleado AS 'Salario'
FROM
        empleados
WHERE
        salarioEmpleado >= ALL (SELECT
                                        salarioEmpleado
                                FROM
                                        empleados
                                WHERE
                                        idDepartamentoFK = 112
                                AND
                                        salarioEmpleado > 3000
                                );
```

Empleado	Salario
PEREZ, MARCOS	3800.00
ALBA, ADRIANA	3500.00
LOPEZ, ANTONIO	5200.00
CAMPS, AURELIO	3500.00

Si comparamos los resultados, observamos que Julio Pérez ahora no aparece. No aparece pues su salario (3400) no es superior a TODOS los empleados que se obtienen del SELECT interior, cuyos resultados son 3400€ y 3500€.

EXISTS: Examina si la subconsulta produce alguna fila de resultados. Si la subconsulta contiene filas, el test adopta el valor **verdadero**. Si la subconsulta no obtiene ninguna fila, el test toma el valor **falso**, nunca puede tomar el valor nulo. Con este test la subconsulta puede tener varias columnas, no importa, ya que el test se fija, no en los valores devueltos sino en si hay o no fila en la tabla resultado de la subconsulta. Cuando se utiliza el test de existencia en la mayoría de los casos habrá que utilizar una referencia externa. Si no se utiliza una referencia externa la subconsulta devuelta siempre será la misma para todas las filas de la consulta principal y en este caso se seleccionan todas las filas de la consulta principal (si la subconsulta genera filas) o ninguna (si la subconsulta no devuelve ninguna fila). Es similar a ANY, pero ahora podemos obtener más de una columna en el SELECT interior pues no comparamos con nada.
Mostrar los empleados del departamento 111 si hay alguien en el departamento 112:

```
SELECT
        nombreEmpleado AS 'Empleado',
        salarioEmpleado AS 'Salario'
FROM
        empleados
WHERE
        idDepartamentoFK = 111
AND
        EXISTS (SELECT
                        *
                FROM
                        empleados
                WHERE
                        idDepartamentoFK = 112
                );
```

Empleado	Salario
AGUIRRE, AUREO	2100.00
SANZ, LAVINIA	1800.00
LARA, DORINA	1500.00
LARA, LUCRECIA	850.00
DURAN, LIVIA	1100.00
PINO, DIANA	1100.00
VAZQUEZ, HONORIA	1000.00
SANTOS, SANCHO	900.00

Si esa misma sentencia se aplica sobre el departamento 115, en vez del 112, al no obtenerse ningún resultado en el SELECT interior, tampoco se obtiene ningún resultado en el SELECT exterior. Igualmente existe el **NOT EXISTS**.

5.8 JOIN

Ahora veremos las subconsultas que trabajan con los campos de **varias tablas** a la vez. Se pueden incluir tantas tablas como necesitemos, al igual que los campos tras el SELECT. Si existen campos con el mismo nombre, pero en diferentes tablas, podemos diferenciarlos con el formato Tabla.Campo. El criterio para combinar las tablas se especifica con la cláusula WHERE. Si no se hace así, obtendremos el PRODUCTO CARTESIANO de las tablas, es decir, un emparejamiento de cada registro de una tabla con cada registro de la otra tabla. Si tenemos 3 registros en una tabla y 4 en la otra, obtendremos 12 (3 x 4) resultados, que, además, no tiene por qué tener sentido.

Para los ejemplos usaremos seguiremos con las tablas de Empleados, Departamentos y Centros de Trabajo de la base de datos Empresa.

Ejemplo 1: Obtener el nombre de empleado y el nombre de departamento de todos los empleados:

```
SELECT
        nombreEmpleado AS 'Nombre Empleado',
        nombreDepartamento AS 'Nombre Departamento'
FROM
        empleados, departamentos
WHERE
        empleados.idDepartamentoFK = departamentos.idDepartamento;
```

Nombre Empleado	Nombre Departamento
GALVEZ, PILAR	DIRECCION GENERAL
ALBA, ADRIANA	DIRECCION GENERAL
LOPEZ, ANTONIO	DIRECCION GENERAL
PEREZ, MARCOS	DIRECCION COMERCIAL
MORAN, CARMEN	DIRECCION COMERCIAL
CAMPOS, ROMULO	DIRECCION COMERCIAL
AGUIRRE, AUREO	SECTOR INDUSTRIAL
SANZ, LAVINIA	SECTOR INDUSTRIAL
LARA, DORINA	SECTOR INDUSTRIAL
LARA, LUCRECIA	SECTOR INDUSTRIAL
DURAN, LIVIA	SECTOR INDUSTRIAL
PINO, DIANA	SECTOR INDUSTRIAL
VAZQUEZ, HONORIA	SECTOR INDUSTRIAL
SANTOS, SANCHO	SECTOR INDUSTRIAL

Ejemplo 2: Obtener el nombre de los departamentos junto al nombre del centro de trabajo al que pertenecen.

```
SELECT
        nombreDepartamento AS 'Departamento',
        nombreCentroTrabajo AS 'Centro de Trabajo'
FROM
        departamentos
JOIN
        centrostrabajo ON idCentroTrabajoFK = idCentroTrabajo
ORDER BY 1;
```

Ejemplo 3: Obtener el nombre de empleado y el nombre de departamento de los empleados de los departamentos 100 o 120:

```
SELECT
        nombreEmpleado AS 'Nombre Empleado',
        nombreDepartamento AS 'Nombre Departamento'
FROM
        empleados, departamentos
WHERE
        empleados.idDepartamentoFK = departamentos.idDepartamento
AND
        empleados. idDepartamentoFK IN(100, 120);
```

Nombre Empleado	Nombre Departamento
GALVEZ, PILAR	DIRECCION GENERAL
ALBA, ADRIANA	DIRECCION GENERAL
LOPEZ, ANTONIO	DIRECCION GENERAL
GIL, GLORIA	ORGANIZACION

Ejemplo 4: Obtener el nombre de empleado, el nombre del departamento al que pertenece y el nombre del centro de trabajo en el que está enmarcado el departamento de los empleados del departamento 110:

```
SELECT
        nombreEmpleado AS 'Nombre Empleado',
        nombreDepartamento AS 'Nombre Departamento',
        nombreCentroTrabajo AS 'Centro Trabajo'
FROM
        empleados, departamentos, centrostrabajo
WHERE
        empleados.idDepartamentoFK = departamentos.idDepartamento
AND
        departamentos.idCentroTrabajoFK = centrostrabajo.idCentroTrabajo
AND
        empleados.idDepartamentoFK = 110;
```

Nombre Empleado	Nombre Departamento	Centro Trabajo
PEREZ, MARCOS	DIRECCION COMERCIAL	Relación con Clientes
MORAN, CARMEN	DIRECCION COMERCIAL	Relación con Clientes
CAMPOS, ROMULO	DIRECCION COMERCIAL	Relación con Clientes

La sentencia **JOIN** en SQL permite combinar registros de dos o más tablas en una base de datos relacional. En el Lenguaje de Consultas Estructurado (SQL), hay tres tipos de JOIN: interno, externo y cruzado.

En casos especiales una tabla puede unirse a sí misma, produciendo una auto-combinación o SELF-JOIN.

Todas las explicaciones que están a continuación utilizan las siguientes dos tablas para ilustrar el efecto de diferentes clases de uniones JOIN. Para probar las sentencias SELECT con los diferentes JOINS, crear una base de datos llamada “EmpresaJoin”, dentro crear las dos tablas siguientes y poblar con los datos que aquí se indican. Para que los ejemplos salgan correctamente, las tablas deben tener exactamente estos datos “preparados”.

Tabla **empleados**

idEmpleado	ApellidoEmpleado	idDepartamentoFK
1	Andrade	31
2	Jordán	33
3	Steinberg	33
4	Róbinson	34
5	Zolano	34
6	Gaspar	36

Tabla **departamentos**	
idDepartamento	**nombreDepartamento**
31	Ventas
33	Ingeniería
34	Producción
35	Marketing

La tabla **empleados** contiene a los empleados con el número del departamento al que pertenecen; mientras que la tabla **departamentos**, contiene el nombre de los departamentos de la empresa, se puede notar que existe un empleado que tiene asignado un número de departamento que no se encuentra en la tabla departamentos (Gaspar), igualmente, en la tabla departamentos existe un departamento al cual no pertenece empleado alguno (Marketing). Esto servirá para presentar algunos ejemplos más adelante.

Otros Ejemplos

```
-- Primeros Ejemplos JOIN
-- Sacar información de más de UNA tabla (relacionadas)
-- Ejemplo 5: Mostrar la información de los Departamentos, incluyendo
-- el NOMBRE del Centro de trabajo al que pertenece.
-- Producto Cartesiano o
-- JOIN Natural
SELECT
idDepartamento AS 'Nº Departamento',
nombreDepartamento AS 'Departamento',
nombreCentroTrabajo AS 'Centro Trabajo'
FROM
departamentos,
centrostrabajo;
-- Ejemplo 6: Mostrar la información de los Departamentos, incluyendo
-- el NOMBRE del Centro de trabajo al que pertenece.
-- INNER JOIN o JOIN interno
SELECT
idDepartamento AS 'Nº Departamento',
nombreDepartamento AS 'Departamento',
nombreCentroTrabajo AS 'Centro Trabajo'
FROM
departamentos,
centrostrabajo
WHERE
```

```
departamentos.idCentroTrabajoFK = centrostrabajo.idCentroTrabajo;
-- Ejemplo 7: Obtener id de empleado, nombre de empleado
-- y NOMBRE del departamento al que pertenece
-- 110 PONS, CESAR Personal
-- ...
SELECT
idEmpleado AS 'Código empleado',
nombreEmpleado AS 'Empleado',
nombreDepartamento AS 'Departamento'
FROM
empleados,
departamentos
WHERE
empleados.idDepartamentoFK = departamentos.idDepartamento
ORDER BY nombreEmpleado;
-- Ejemplo 8: Obtener id de empleado, nombre de empleado
-- y NOMBRE del departamento al que pertenece los empleados
-- que cobran menos de 1500 € al mes
SELECT
idEmpleado AS 'Código empleado',
nombreEmpleado AS 'Empleado',
nombreDepartamento AS 'Departamento'
FROM
empleados,
departamentos
WHERE
empleados.idDepartamentoFK = departamentos.idDepartamento
AND salarioEmpleado < 1500
ORDER BY nombreEmpleado;
-- Ejemplo 9: Obtener id de empleado, nombre de empleado
-- y NOMBRE del departamento al que pertenece
-- Usar preferiblemente
-- INNER JOIN
SELECT
idEmpleado AS 'Código empleado',
nombreEmpleado AS 'Empleado',
nombreDepartamento AS 'Departamento'
FROM
empleados
INNER JOIN
```

```
departamentos ON empleados.idDepartamentoFK =
departamentos.idDepartamento
ORDER BY nombreEmpleado;
-- Propuesta: Obtener id de empleado, nombre de empleado
-- y NOMBRE del departamento al que pertenece los empleados
-- que cobran menos de 1500 € al mes
```

Combinación interna (*INNER JOIN*)

Con esta operación se calcula el **producto cruzado** de todos los registros; así cada registro en la tabla A es combinado con cada registro de la tabla B; pero sólo permanecen aquellos registros en la tabla combinada que satisfacen las condiciones que se especifiquen. Este es el tipo de JOIN más utilizado por lo que es considerado el tipo de combinación predeterminado.

SQL especifica dos formas diferentes para expresar estas combinaciones. La primera, conocida como **explícita** usa la palabra JOIN, mientras que la segunda es **implícita** y usa ',' para separar las tablas a combinar en la sentencia FROM de la declaración SELECT. Entonces siempre se genera el producto cruzado del cual se seleccionan las combinaciones que cumplan lo que indica la sentencia WHERE.

Es necesario tener especial cuidado cuando se combinan columnas con valores nulos NULL ya que el valor nulo no se combina con otro valor o con otro nulo, excepto cuando se le agregan predicados tales como IS NULL o IS NOT NULL.

Como ejemplo, la siguiente consulta toma todos los registros de la tabla Empleados y encuentra todas las combinaciones en la tabla Departamentos. La sentencia JOIN compara los valores en la columna idDepartamento en ambas tablas. Cuando no existe esta correspondencia entre algunas combinaciones, éstas no se muestran; es decir que, si el número de departamento de un empleado no coincide con los números de departamento de la tabla Departamentos, no se mostrará el empleado con su respectivo departamento en la tabla resultante.

Las dos consultas siguientes son similares, y se realizan de manera explícita (A) e implícita (B):
A) Ejemplo de la sentencia INNER JOIN explícita:

```
SELECT *
FROM
        empleados
INNER JOIN
        departamentos
ON
        empleados.idDepartamentoFK = departamentos.idDepartamento;
```

B) Ejemplo de la sentencia INNER JOIN implícita:

```
SELECT *
FROM
        empleados, departamentos
WHERE
        empleados.idDepartamentoFK = departamentos.idDepartamento;
```

Resultados:

Empleados.ApellidoEmpleado	Empleados.idDepartamentoFK	Departamentos.NombreDepartamento	Departamentos.idDepartamento
Zolano	34	Producción	34
Jordán	33	Ingeniería	33
Róbinson	34	Producción	34
Steinberg	33	Ingeniería	33
Andrade	31	Ventas	31

El empleado Gaspar y el departamento de Marketing no son presentados en los resultados ya que ninguno de éstos tiene registros correspondientes en la otra tabla. No existe un departamento con número 36 ni existe un empleado con número de departamento 35.

A la combinación que utiliza comparaciones dentro del predicado JOIN se le llama **Theta-join**.

C) Ejemplo de combinación tipo theta:

```
SELECT *
FROM
        empleados
INNER JOIN
        departamentos
ON
        empleados.idDepartamentoFK < departamentos.idDepartamento;
```

Las operaciones INNER JOIN pueden ser clasificadas como de equivalencia, naturales, y cruzadas.

De equivalencia (equi-join)

Es una especie de theta-join que usa comparaciones de igualdad en el predicado JOIN. Cuando se usan operadores, tales como < o > no se pueden clasificar en este rango.

D) Ejemplo de combinación de equivalencia:

```
SELECT *
FROM
        empleados
INNER JOIN
        departamentos
ON
        empleados.idDepartamentoFK = departamentos. idDepartamento;
```

La tabla resultante presenta dos columnas idDepartamento e idDepartmentoFK, una proveniente de la tabla Empleado y otra de la tabla Departamento.

Natural (Natural join)

Es una especialización de la combinación de equivalencia, anteriormente mencionada. En este caso se comparan todas las columnas que tengan el mismo nombre en ambas tablas. La tabla resultante contiene sólo una columna por cada par de columnas con el mismo nombre.

E) Ejemplo de combinación natural:

```
SELECT *
FROM
        empleados
NATURAL JOIN
        departamentos;
```

El resultado es un poco diferente al del ejemplo D, ya que esta vez la columna idDepartamento se muestra sola una vez en la tabla resultante.

Empleados.ApellidoEmpleado	idDepartamento	Departamentos.NombreDepartamento
Zolano	34	Producción
Jordán	33	Ingeniería
Róbinson	34	Producción
Steinberg	33	Ingeniería
Andrade	31	Ventas

El uso de esta sentencia **NATURAL** puede producir resultados ambiguos y generar problemas si la base de datos cambia, porque al añadir, quitar, o renombrar las columnas, puede perder el sentido la sentencia; por esta razón es preferible expresar el predicado usando las otras expresiones nombradas anteriormente (ejemplos A y B).

Cruzada (Cross join)

Presenta el **producto cartesiano** de todos los registros de las dos tablas.

El código SQL para realizar este producto cartesiano enuncia las tablas que serán combinadas, pero no incluye algún predicado que filtre el resultado.

F) Ejemplo de combinación cruzada explícita:

```
SELECT *
FROM
        empleados
CROSS JOIN
        departamentos;
```

G) Ejemplo de combinación cruzada implícita:

```
SELECT * FROM  empleados, departamentos;
```

Empleados.ApellidoEmpleado	Empleados.idDepartamentoFK	Departamentos.NombreDepartamento	Departamentos.idDepartamento
Andrade	31	Ventas	31
Jordán	33	Ventas	31
Steinberg	33	Ventas	31
Zolano	34	Ventas	31
Róbinson	34	Ventas	31
Gaspar	36	Ventas	31
Andrade	31	Ingeniería	33
Jordán	33	Ingeniería	33
Steinberg	33	Ingeniería	33
Solano	34	Ingeniería	33
Róbinson	34	Ingeniería	33
Gaspar	36	Ingeniería	33
Andrade	31	Producción	34
Jordán	33	Producción	34
Steinberg	33	Producción	34
Solano	34	Producción	34
Róbinson	34	Producción	34
Gaspar	36	Producción	34
Andrade	31	Marketing	35
Jordán	33	Marketing	35
Steinberg	33	Marketing	35
Solano	34	Marketing	35
Róbinson	34	Marketing	35
Gaspar	36	Marketing	35

Esta clase de combinaciones son usadas pocas veces, generalmente se les agregan condiciones de filtrado con la sentencia WHERE para hallar resultados específicos.

Combinación externa (OUTER JOIN)

Mediante esta operación no se requiere que cada registro en las tablas a tratar tenga un registro equivalente en la otra tabla. El registro es mantenido en la tabla combinada si no existe otro registro que le corresponda.

Este tipo de operación se subdivide dependiendo de la tabla a la cual se le admitirán los registros que no tienen correspondencia, ya sean de tabla izquierda, de tabla derecha, o combinación completa.

De tabla izquierda (LEFT OUTER JOIN o LEFT JOIN)

El resultado de esta operación siempre contiene todos los registros de la tabla de la izquierda (la primera tabla que se menciona en la consulta), aun cuando no exista un registro correspondiente en la tabla de la derecha, para uno de la izquierda.

La sentencia LEFT OUTER JOIN retorna la pareja de todos los valores de la tabla izquierda con los valores de la tabla de la derecha correspondientes, o retorna un valor nulo NULL en caso de no correspondencia.

A diferencia del resultado presentado en los ejemplos A y B (de combinación interna) donde no se mostraba el empleado cuyo departamento no existía; en el siguiente ejemplo se presentarán los empleados con su respectivo departamento, e inclusive se presentará el empleado, cuyo departamento no existe.

H) Ejemplo de tabla izquierda para la combinación externa:

```
SELECT
        *
FROM
        empleados
LEFT OUTER JOIN
        departamentos
ON
        empleados.idDepartamentoFK = departamentos.idDepartamento;
```

Empleados.ApellidoEmpleado	Empleados.idDepartamentoFK	Departamentos.NombreDepartamento	Departamentos.idDepartamento
Jordán	33	Ingeniería	33
Andrade	31	Ventas	31
Róbinson	34	Producción	34
Zolano	34	Producción	34
Gaspar	36	NULL	NULL
Steinberg	33	Ingeniería	33

De tabla derecha (RIGHT OUTER JOIN o RIGHT JOIN)

Esta operación es inversa a la anterior; el resultado de esta operación siempre contiene todos los registros de la tabla de la derecha (la segunda tabla que se menciona en la consulta), aun cuando no exista un registro correspondiente en la tabla de la izquierda, para uno de la derecha.

La sentencia RIGHT OUTER JOIN retorna la pareja de todos los valores de la tabla derecha con los valores de la tabla de la izquierda correspondientes, o retorna un valor nulo NULL en caso de no correspondencia.

I) Ejemplo de tabla derecha para la combinación externa:

```
SELECT
        *
FROM
        empleados
RIGHT OUTER JOIN
        departamentos
ON
        empleados.idDepartamentoFK = departamentos.idDepartamento;
```

Empleados.ApellidoEmpleado	Empleados.idDepartamentoFK	Departamentos.NombreDepartamento	Departamentos.idDepartamento
Zolano	34	Producción	34
Jordán	33	Ingeniería	33
Róbinson	34	Producción	34
Steinberg	33	Ingeniería	33
Andrade	31	Ventas	31
NULL	NULL	Marketing	35

En este caso el área de Marketing fue presentada en los resultados, aunque aún no hay empleados registrados en dicha área.

Combinación completa (FULL OUTER JOIN)

Esta operación presenta los resultados de tabla izquierda y tabla derecha, aunque no tengan correspondencia en la otra tabla. La tabla combinada contendrá, entonces, todos los registros de ambas tablas y presentará valores nulos NULL para registros sin pareja.

J) Ejemplo de combinación externa completa:

```
SELECT
        *
FROM
        empleados
FULL OUTER JOIN
        departamentos
ON
        empleados.idDepartamentoFK = departamentos.idDepartamento;
```

Empleado.ApellidoEmpleado	Empleado.IDDepartamentoFK	Departamento.NombreDepartamento	Departamento.IDDepartamento
Zolano	34	Producción	34
Jordán	33	Ingeniería	33
Róbinson	34	Producción	34
Gaspar	36	NULL	NULL
Steinberg	33	Ingeniería	33
Andrade	31	Ventas	31
NULL	NULL	Marketing	35

Como se puede notar, en este caso se encuentra el empleado Gaspar con valor nulo en su área correspondiente, y se muestra además el departamento de Marketing con valor nulo en los empleados de esa área.

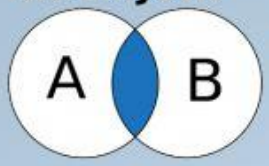
Algunos sistemas de bases de datos no soportan esta funcionalidad, pero puede ser emulada a través de las combinaciones de tabla izquierda, tabla derecha y de la sentencia de unión UNION.

K) El mismo ejemplo puede expresarse así:

```
SELECT
	*
FROM
	empleados
LEFT JOIN
	departamentos
ON
	empleados.idDepartamentoFK = departamentos.idDepartamento
UNION
SELECT
	*
FROM
	empleados
RIGHT JOIN
	departamentos
ON
	empleados.idDepartamentoFK = departamentos.idDepartamento
WHERE
	empleados.idDepartamentoFK IS NULL;
```

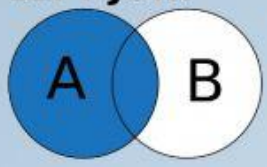
SQL JOINS

INNER JOIN



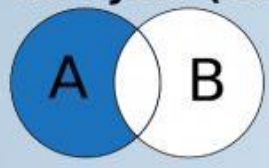

```
SELECT *
FROM A
INNER JOIN B ON A.key = B.key
```

LEFT JOIN



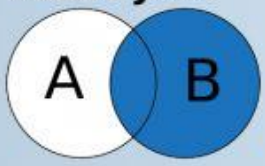

```
SELECT *
FROM A
LEFT JOIN B ON A.key = B.key
```

LEFT JOIN (sans l'intersection de B)



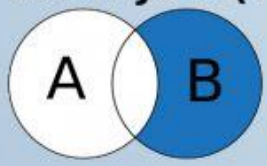

```
SELECT *
FROM A
LEFT JOIN B ON A.key = B.key
WHERE B.key IS NULL
```

RIGHT JOIN



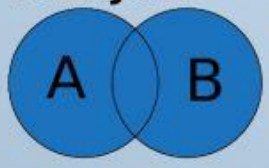

```
SELECT *
FROM A
RIGHT JOIN B ON A.key = B.key
```

RIGHT JOIN (sans l'intersection de A)



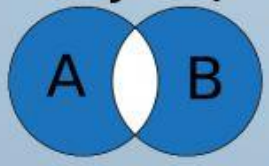

```
SELECT *
FROM A
RIGHT JOIN B ON A.key = B.key
WHERE B.key IS NULL
```

FULL JOIN




```
SELECT *
FROM A
FULL JOIN B ON A.key = B.key
```

FULL JOIN (sans intersection)



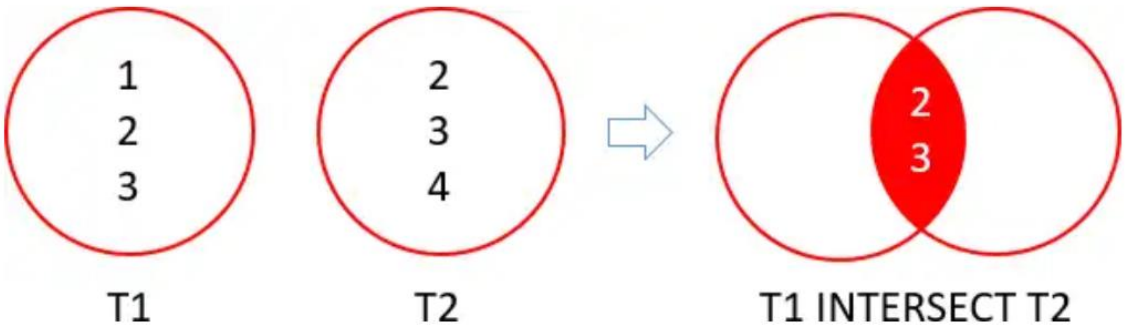

```
SELECT *
FROM A
FULL JOIN B ON A.key = B.key
WHERE A.key IS NULL
OR B.key IS NULL
```

sql.sh

Por último, hay que comentar que existe la posibilidad de obtener información relacionada a los resultados de dos o más consultas. Estamos hablando de los operadores **UNION** (visto anteriormente), **INTERSECT** y **MINUS**.

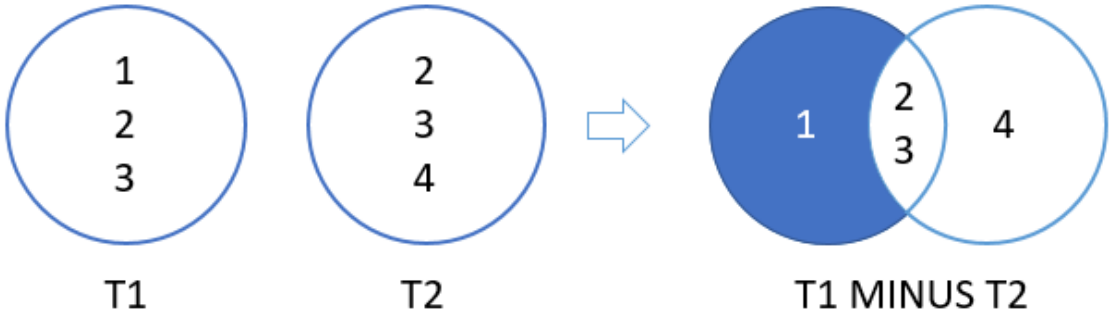
Con INTERSECT tratamos de averiguar qué resultados se repiten en los resultados de dos consultas:

Consulta T1 INTERSECT Consulta T2



De otro modo, con MINUS tratamos de localizar los elementos de una consulta que no son de la otra ni de la intersección:

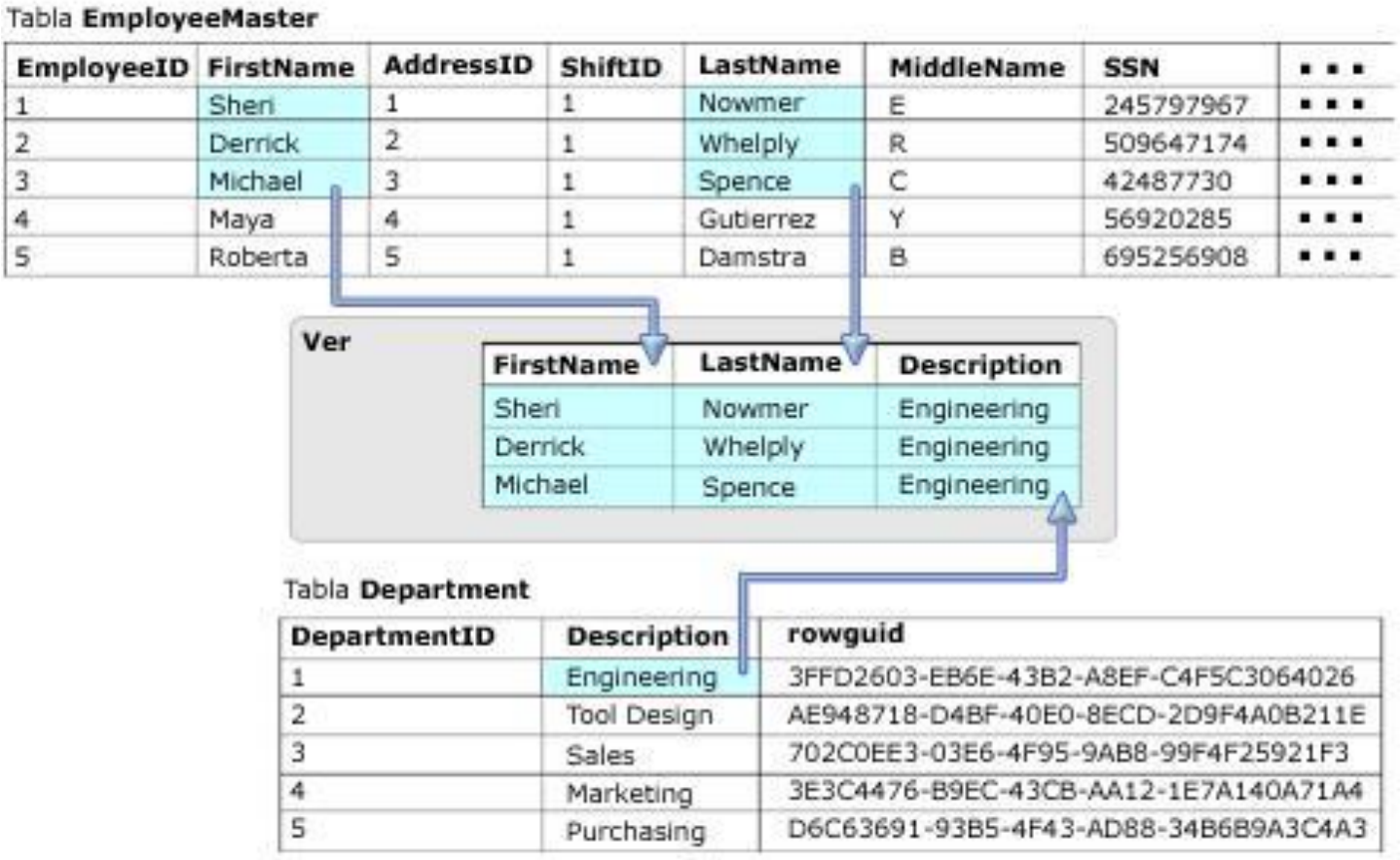
Consulta T1 MINUS Consulta T2



5.9 Consultas sobre Vistas

También podemos realizar consultas sobre Vistas en vez de sobre Tablas directamente. Pero la forma de trabajar es exactamente la misma.

Las **vistas** pueden considerarse como tablas virtuales. Generalmente hablando, una tabla tiene un conjunto de definiciones, y almacena datos físicamente. Una vista también tiene un conjunto de definiciones, que se construye en la parte superior de la(s) tabla(s) u otra(s) vista(s), pero no almacena datos físicamente.



Una vista es una tabla sin contenido, totalmente virtual, que devuelve las filas resultado de ejecutar una consulta SQL. La diferencia con una consulta directamente es que, mientras cada sentencia SQL enviada al SGBD tiene que pasar por un proceso de compilación, la vista es una consulta cuya definición ha sido almacenada previamente y que ya ha sido compilada, siendo por tanto el tiempo de ejecución bastante menor.

También tienen una implicación importante en el hecho de que un usuario podría no tener acceso a la información de varias tablas y, sin embargo, sí tener acceso a la vista que consulta esas tablas, proporcionando de esta manera un acceso controlado solo a determinadas filas y columnas de las tablas.

La sintaxis para la creación de una vista es la siguiente:

```
CREATE VIEW "NOMBRE_VISTA" AS "Instrucción SQL"
```

La *"Instrucción SQL"* puede ser cualquiera de las instrucciones SQL que hemos descripto en este temario.

Vamos a crear una vista llamada vistaEmpleados con su id, su nombre y el nombre del departamento al que pertenecen:

```
CREATE VIEW vistaEmpleados AS
SELECT
        idEmpleado AS 'Código',
        nombreEmpleado AS 'Nombre',
        nombreDepartamento AS 'Departamento'
FROM
        empleados
JOIN
        departamentos
ON
        empleados.idDepartamentoFK = departamentos.idDepartamento
ORDER BY 1;
```

Ahora, simplemente haciendo una consulta sobre la vista, obtendremos el mismo resultado:

```
SELECT * FROM vistaEmpleados;
```

Código	Nombre	Departamento
110	PONS, CESAR	PERSONAL
120	LASA, MARIO	SECTOR SERVICIOS
130	TEROL, LUCIANO	SECTOR SERVICIOS
150	PEREZ, JULIO	SECTOR SERVICIOS
160	AGUIRRE, AUREO	SECTOR INDUSTRIAL

Las vistas se pueden modificar con:

```
ALTER VIEW nombre_vista AS "Instrucción SQL";
```

5.10 Otros usos de SELECT

Como hemos podido comprobar, existen multitud de uso para la sentencia SELECT. Pero aún vamos a comentar un par más: la posibilidad de insertar en una tabla datos a partir del resultado de una sentencia SELECT.

Tan sencillo como hacer un INSERT INTO junto a un SELECT de esta forma:

```
INSERT INTO table2

SELECT * FROM table1

WHERE condición;
```

05/08/2023