

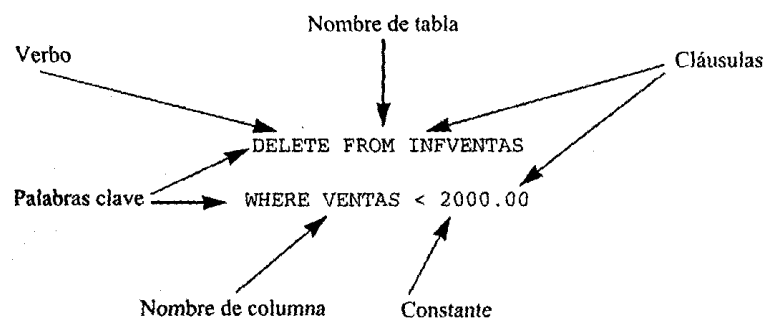
1. CONCEPTOS BÁSICOS DE SQL.....	2
2. LA SENTENCIA SELECT: SELECCIÓN DE DATOS.....	5
3. FUNCIONES DE AGREGACIÓN	14
4. CONSULTAS AGRUPADAS (CLÁUSULA GROUP BY)	16
5. CONSULTAS MULTITABLAS. COMPOSICIONES (JOIN).....	20
6. SUBCONSULTAS.....	30

1. CONCEPTOS BÁSICOS DE SQL.

SENTENCIAS

El lenguaje SQL consta de un conjunto de sentencias. Cada **sentencia demanda una acción específica para el SGBD**, tal como la creación de una nueva tabla, la recuperación de datos o la inserción de nuevos datos en la base de datos.

- ⇒ Todas las sentencias SQL comienzan con un **verbo**, una palabra clave que describe lo que la sentencia lo que la sentencia hace. INSERT, DELETE, UPDATE son verbos típicos.
- ⇒ La sentencia continúa con una o más **cláusulas**. Una cláusula puede especificar los datos sobre los que debe actuar la sentencia, o proporcionar más detalles acerca de lo que la sentencia se supone que hace. Todas las cláusulas comienzan con una palabra clave, tal como **WHERE, FROM, INTO Y HAVING**. Algunas cláusulas son opcionales; otras son necesarias. La estructura y contenido específicos varían de una cláusula a otra.
- ⇒ Las palabras claves que forman parte de un sentencia SQL no pueden ser utilizadas para designar objetos de la base de datos, tales como tablas o campos.



Estructura de una sentencia SQL

CLÁUSULAS

Las cláusulas son condiciones de modificación utilizadas para definir los datos que desea seleccionar o manipular.

CLÁUSULA	DESCRIPCIÓN
FROM	Utilizada para especificar la tabla de la cual se van a seleccionar los registros
WHERE	Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar
GROUP BY	Utilizada para separar los registros seleccionados en grupos específicos
HAVING	Utilizada para expresar la condición que debe satisfacer cada grupo
ORDER BY	Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico

OPERADORES LÓGICOS

Operador	Uso
AND	Es el "y" lógico. Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.
OR	Es el "o" lógico. Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos es cierta.
NOT	Negación lógica. Devuelve el valor contrario de la expresión.

OPERADORES DE COMPARACIÓN

Operador	Uso
<	Menor que
>	Mayor que
<>	Distinto de
<=	Menor ó Igual que
>=	Mayor ó Igual que
=	Igual que
BETWEEN	Utilizado para especificar un intervalo de valores.
LIKE	Utilizado en la comparación de un modelo
In	Utilizado para especificar registros de una base de datos.

FUNCIONES DE AGREGADO

Las funciones de agregado se usan dentro de una cláusula SELECT en grupos de registros para devolver un único valor que se aplica a un grupo de registros.

Función	DESCRIPCIÓN
AVG	Utilizada para calcular el promedio de los valores de un campo determinado
COUNT	Utilizada para devolver el número de registros de la selección
SUM	Utilizada para devolver la suma de todos los valores de un campo determinado
MAX	Utilizada para devolver el valor más alto de un campo especificado
MIN	Utilizada para devolver el valor más bajo de un campo especificado

DESCRIPCIÓN DE LAS TABLAS

Antes de realizar ninguna consulta, modificación o borrado de datos, veamos las tablas con las que vamos a trabajar. Se utilizan las siguientes tablas de la base de datos compañía

EMPLEADO

NSS	Nombre	Apel1	Apel2	Sexo	Dirección	Fechanac	Salario	Numdept	NSSsup
11111	José	Nieto	Silva	M	C/Sol,4	12/03/63	420000	1	
11122	Federico	Fuertes	Serantes	M	C/Espiga,7	07/10/70	140000	2	88888
11133	Carmen	Prieto	Gonzalez	F	C/Fresnos,7	07/10/70	175000	1	11111
22222	Federico	García	López	M	C/Luna,8	12/04/70	150000	3	11111
33322	Maria	Ramos	Perez	F	C/Sol,22	03/03/65	120000	1	88888
33333	Ana	Sanchez	Valdés	F	C/Bravo,8	17/03/59	120000	1	22222
44444	Jaime	Silva	Gonzalez	M	C/Dalias,7	12/04/70	200000	1	22222
55555	Luis	Gonzalez	Mouro	M	C/Peru, 9	19/08/58	150000	2	11111
66666	José	Vilan	Silva	M	C/Panama, 8	17/12/59	200000	1	11111
77777	Ana	López	Villa	F	C/Rosas,7	11/04/71	200000	2	22222
88888	Pedro	Gonzalez	Jimenez	M	C/Valle, 6	17/03/53	300000	3	
99999	Ana	Silva	López	F	C/Bravo,8	07/10/70	150000	1	88888

DEPARTAMENTO

Numdep	Nombredp	Numempdep	NSSGerente	fechainicGerente
1	Investigacion	16	11111	12/03/89
2	Administracion	14	22222	13/02/89
3	Dirección	2	88888	14/08/99

PROYECTO

Numproy	Nombreproy	Lugarproy	Numdep
1	ProductoX	Vigo	1
2	ProductoY	Santiago	1
3	ProductoZ	Vigo	1
4	Reorganización	Vigo	3
5	Automatización	Vigo	2
6	Nuevaspresta	Lugo	2
7	ProductoW	Coruña	1

EMPLEADO-PROYECTO

NSS	NUMPROY	NUMHORAS
11111	1	30
11111	2	24
11111	4	13
22222	1	25
22222	3	30
22222	6	14
44444	5	17
44444	7	35
44444	1	25
66666	2	25
77777	7	35
88888	7	40

2. LA SENTENCIA SELECT: SELECCIÓN DE DATOS

La sentencia `SELECT` es la única sentencia de SQL para obtener datos a partir de las tablas de una base de datos. El formato general de esta sentencia es el siguiente:

```
SELECT columnas
FROM tablas
[WHERE condiciones_where]
[GROUP BY columnas_group]
[HAVING condiciones_having]
[ORDER BY columnas_orden]
```

Se empezará por consultas a una sola tabla y luego se incluirán varias.

⇒ SELECCIÓN DE ATRIBUTOS

La consulta más sencilla es seleccionar todos los campos de una tabla: "Selecciona todos los datos de todos los empleados":

```
SELECT * FROM EMPLEADO;
```

Se utiliza el asterisco (*) como comodín para seleccionar todos los campos. La sentencia anterior es equivalente a :

```
SELECT NSS, NOMBRE, APEL1, APEL2, SEXO, DIRECCION, FECHANAC,
        SALARIO, NUMDEPT, NSSSUP
FROM EMPLEADO;
```

Se pueden seleccionar columnas individuales: "Lista todos los salarios":

```
SELECT SALARIO FROM EMPLEADO
```

Lista el nombre, apellidos y fecha de nacimiento de todos los empleados:

```
SELECT NOMBRE, APEL1, APEL2, FECHANAC FROM EMPLEADO
```

⇒ ORDENAR LOS REGISTROS

Adicionalmente se puede especificar el orden en que se desean recuperar los registros de las tablas mediante la cláusula **ORDER BY** Lista de Campos. En donde Lista de campos representa los campos a ordenar.

Ejemplo: Lista el nombre, apellidos y fecha de nacimiento de todos los empleados ordenados por la fecha de nacimiento:

```
SELECT NOMBRE, APEL1, APEL2, FECHANAC FROM EMPLEADO
ORDER BY FECHANAC
```

Se pueden ordenar los registros por mas de un campo, como por ejemplo:

```
SELECT NOMBRE, APEL1, APEL2, FECHANAC FROM EMPLEADO
ORDER BY NOMBRE, APEL1, APEL2
```

Incluso se puede especificar el orden de los registros: ascendente mediante la cláusula (**ASC** -se toma este valor por defecto) ó descendente (**DESC**)

```
SELECT NOMBRE, APEL1, APEL2, FECHANAC FROM EMPLEADO
ORDER BY SALARIO ASC, FECHANAC DESC
```

⇒ COLUMNAS CALCULADAS

Además de las columnas cuyos valores provienen directamente de la base de datos, una consulta SQL puede incluir columnas calculadas cuyos valores se calculan a partir de los valores almacenados. Para solicitar una columna calculada, se especifica una expresión SQL en la lista de selección.

Expresión: Es una combinación de operadores, operandos y paréntesis. Cuando el SGBD ejecuta las operaciones indicadas en ella, produce como **resultado un único valor**.

- Los **operadores**: +, -, /, *
- **Operandos**: nombres de campos, constantes o otras expresiones. Puede haber otros tipos de operandos como las funciones, las variables de programación etc..
- **AS**: sirve para poner un nombre a la expresión. Si el nombre tiene espacios en blanco, en TRANSACT SQL se pone entre []. Ej: [Salario Mínimo]

También se pueden hacer operaciones con los campos seleccionados:

```
SELECT NOMBRE, APEL1, APEL2, SALARIO, SALARIO * 2 AS DOBLESALARIO
FROM EMPLEADO
```

Lo que nos daría el listado de empleados con el doble del salario que tienen.

⇒ FILAS DUPLICADAS (DISTINCT)

Omite los registros que contienen datos duplicados en los campos seleccionados. Para que los valores de cada campo listado en la instrucción SELECT se incluyan en la consulta deben ser únicos.

Ej: "Lista todos los salarios diferentes"

```
SELECT DISTINCT SALARIO FROM EMPLEADO;
```

Si se omite la palabra DISTINCT, SQL no elimina las filas duplicadas. También se puede especificar la palabra reservada **ALL** para indicar explícitamente que las filas duplicadas sean incluidas, pero es innecesario por que este es el comportamiento por omisión. Las siguientes sentencias devuelven los mismos resultados.

```
SELECT ALL SALARIO FROM EMPLEADO;
```

```
SELECT SALARIO FROM EMPLEADO;
```

⇒ DEVOLVER LAS N PRIMERAS FILAS

Para indicar las primeras n filas que se devolverán. La sintaxis es:

```
SELECT LIMIT inicio, numLineas
```

Para indicar los nombres de los cinco primeros empleados:

```
SELECT NOMBRE FROM EMPLEADO LIMIT 5
```

Si queremos que se devuelvan los cinco primeros nombres por orden alfabético:

```
SELECT NOMBRE FROM EMPLEADO ORDER BY NOMBRE LIMIT 5;
```

Si queremos que se devuelvan dos filas a partir de la tercera:

```
SELECT NOMBRE FROM EMPLEADO LIMIT 3,2
```

⇒ SELECCIÓN DE FILAS (CLÁUSULA WHERE)

Esta cláusula permite filtrar los registros con el fin de recuperar solamente aquellos que cumplan unas condiciones preestablecidas.

Detalles a tener en cuenta: En ANSI SQL las cadenas de texto y las fechas van encerradas entre comillas. En ACCESS la única diferencia es que la fecha va encerrada ente almohadillas (#)

Ej: Listar los empleados con sueldo superior a 130000 pts.

```
SELECT NOMBRE, APEL1, APEL2, SALARIO FROM EMPLEADO  
WHERE SALARIO > 130000
```

CONDICIONES PARA RESTRINGIR LA CONSULTA

La selección de filas se especifica en la cláusula WHERE mediante predicados. Un predicado expresa una condición y su resultado puede ser verdadero o falso.

La condición expresada por el predicado se considera satisfecha únicamente cuando toma el valor verdadero y sólo las filas que satisfagan el predicado serán seleccionadas

Se distinguen dos tipos: simples y compuestos:

❑ PREDICADOS SIMPLES:

Dentro de estos predicados distinguimos los siguientes:

- **Operadores de comparación:** expresan condiciones de comparación entre los valores. Se expresan con los signos que se detallan a continuación:

>, >=, <, <=, =, <>: Estos operadores se pueden utilizar con todos los tipos de datos, para compararlos según una relación de orden. Si los datos son números sigue la ordenación normal, si son cadenas de caracteres sigue la ordenación ASCII, si son fechas la ordenación temporal, etc.

En SQL-TRANSACT (el dialecto del SQL para SQL SERVER 7.0) utiliza también los siguiente operadores de comparación: **!=** (distinto a), **!>** (No mayor que) y **!<** (no menor que)

- Predicado **BETWEEN** : Sirve para determinar si un valor está comprendido entre un intervalo de valores:

Sintaxis:

```
WHERE campo [NOT] BETWEEN limite_inferior AND limite_superior.
```

Es equivalente a

```
(campo>=limite_inferior and campo <=limite_superior)
```

Ej: Lista los empleados cuyo NSS este entre el 22222 y el 44444

```
SELECT * FROM EMPLEADO
WHERE NSS BETWEEN 22224 AND 44444;
```

- Predicado **IN**: Sirve para averiguar si el resultado de una expresión está incluido en la lista de valores especificada detrás de la palabra IN.

Sintaxis:

```
WHERE Expresión [NOT] IN (valor1, valor2, ..., valor n)
```

Ej: Lista los empleados de salarios 1200000, 20000 o 300000


```
SELECT * FROM EMPLEADO
WHERE SALARIO IN (120000,200000,300000)
```

- Comparación de caracteres. Predicado **LIKE**:

Se puede utilizar para comparar una expresión de cadena con un patrón en una expresión. LIKE devuelve los valores de los campos que coincidan con el patrón especificado.

Sintaxis:

Expresión LIKE patrón

El patrón puede ser una cadena completa como ('JOSE') o puede ser una cadena que contiene caracteres comodines:

El ANSI SQL especifica los siguientes caracteres comodines:

Comodín	BUSCA
%	Cualquier cadena de cero o más caracteres. En Access, el equivalente es *
_	Cualquier carácter único. En Access, el equivalente es ? y en SQL-Transact ~

Ejemplo: Buscar los productos que en la descripción indiquen que son obras de Michael Fende.

```
SELECT *
FROM EMPLEADO
WHERE dirección LIKE '%Panama%';
```

Ejemplo . Seleccionar los datos de los empleados cuyo apellido empiece por A.y conste de 3 caracteres más.

```
SELECT *
FROM empleado
WHERE apell LIKE 'A___'
```

- Para realizar una comparación exacta es más rápido emplear un operador de comparación en lugar de LIKE.
- Hay que tener en cuenta los espacios en blanco, ya que estos son significativos.
- Si deseamos buscar los caracteres de los comodines deberemos ponerles antes el carácter de escape \. Es decir, deberemos poner \% y _

- Comparación de caracteres. Predicado **REGEXP**:

Se puede utilizar para comparar una expresión de cadena con una expresión regular. Las expresiones regulares permiten especificar un patrón para una búsqueda compleja. REGEXP devuelve los valores de los campos que coincidan con la expresión regular.

Sintaxis:

Expresión REGEXP <i>expresion_regular</i>

A continuación mostramos algunos de los caracteres especiales y constructores que pueden emplearse para en MySQL con el operador REGEXP. Pueden verse más en la siguiente URL:

<http://dev.mysql.com/doc/refman/5.0/es/regexp.html>

Una expresión regular no trivial usa ciertos constructores especiales, que pueden concordar más de en una cadena. Por ejemplo, la expresión regular `hello|word` concuerda tanto con la cadena `hello` como con la cadena `word`.

Un ejemplo más complicado: la expresión regular `B[an]*s` concuerda con las cadenas `Bananas`, `Baaaaas`, `Bs`, y cualquier otra cadena que comience con una `B`, termine con unas y contenga cualquier número de caracteres `a` o `n` entre ellas.

Una expresión regular para el operador REGEXP puede utilizar cualquiera de los siguientes caracteres especiales y constructores:

- `^`

que concuerda con el inicio de la cadena.

```
mysql> SELECT 'fo\nfo' REGEXP '^fo$';
```

-> 0

Es decir, no concuerda

```
mysql> SELECT 'fofo' REGEXP '^fo';
```

-> 1

Sí concuerda

Ejemplo . Seleccionar nombre y apellidos de los empleados cuyo nombre empiece por 'An', empleando REGEXP (pues también podríamos hacerlo con LIKE)

```
SELECT nombre, apell1, apell2 FROM empleado
WHERE nombre regexp '^An';
```

- `$`

que concuerda con el final de la cadena.

```
mysql> SELECT 'fo\no' REGEXP '^fo\no$';
```

-> 1

```
mysql> SELECT 'fo\no' REGEXP '^fo$';
```

-> 0

Ejemplo . Seleccionar nombre y apellidos de los empleados cuyo primer apellido termine en 'ez', empleando REGEXP (pues también podríamos hacerlo con LIKE)

```
select nombre, apell1, apell2 from empleado
where apell1 regexp 'ez$';
```

- .

que concuerda con cualquier carácter (incluyendo el retorno de carro y la nueva línea).

Ejemplo . Seleccionar nombre y apellidos de los empleados que contengan 'nz' en el primer apellido, empleando REGEXP (pues también podríamos hacerlo con LIKE)

```
select nombre, apell1, apell2 from empleado
where apell1 regexp '.nz';
```

Ejemplo . Seleccionar nombre y apellidos de los empleados cuyo primer apellido empiece por 'Go' y termine por 'ez', empleando REGEXP (pues también podríamos hacerlo con LIKE)

```
select nombre, apell1, apell2 from empleado
where apell1 regexp '^go.*ez$';
```

Para indicar rangos, algo que en MySQL no se puede hacer con LIKE, emplearíamos expresiones como las siguientes:

- [a-dX], [^a-dX]

Concuerda con cualquier carácter que sea (o no sea, si se utiliza ^) uno de los siguientes: a, b, c, d o x. Un carácter - entre dos caracteres distintos forma un rango que concuerda con cualquier carácter entre el primero y el segundo. Por ejemplo, [0-9] concuerda con cualquier dígito decimal. Para incluir un carácter] literal, debe estar precedido inmediatamente por un corchete abierto [. Para incluir un carácter - literal , debe estar al principio o al final. Cualquier carácter que no tenga un significado especial, concuerda sólo consigo mismo si está dentro de un par de [] .

```
mysql> SELECT 'aXbc' REGEXP '[a-dXYZ]';          -> 1
mysql> SELECT 'aXbc' REGEXP '^[a-dXYZ]$';        -> 0
mysql> SELECT 'aXbc' REGEXP '^a-dXYZ]+$';        -> 1
mysql> SELECT 'aXbc' REGEXP '^[^a-dXYZ]+$';      -> 0
mysql> SELECT 'gheis' REGEXP '^a-dXYZ]+$';      -> 1
mysql> SELECT 'gheisa' REGEXP '^a-dXYZ]+$';      -> 0
```

Ejemplo . Seleccionar nombre y apellidos de los empleados cuyo primer apellido empiece por 'G' y tenga a continuación una de las tres primeras vocales.

```
select nombre, apell1, apell2 from empleado
where apell1 regexp 'G[aei]';
```

- Valores **NULOS**:

Un campo tiene un valor nulo si no se han introducido valores durante la introducción de datos y si no hay valores predeterminados definidos para esa columna. Un valor nulo no es lo mismo que cero (un valor numérico) o un espacio en blanco (un valor de carácter):

Columna IS [NOT] NULL

Se utiliza el operador **IS NULL** para recuperar filas para las que falta información para la columna especificada y **IS NOT NULL** para recuperar filas que tienen valores conocidos.

Ej: Lista los empleados que no tengan un supervisor directo.

```
SELECT * FROM empleado WHERE nsssup IS NULL
```

❑ PREDICADOS COMPUESTOS:

Son combinaciones de otros predicados con los operadores lógicos: AND, NOT y OR

Operador	Uso
AND	Es el "y" lógico. Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.
OR	Es el "o" lógico. Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos es cierta.
NOT	Negación lógica. Devuelve el valor contrario de la expresión.

Ej: Lista los empleados con nss mayor de 33333, sólo los que pertenezcan al departamento 1 o 2.

```
SELECT * FROM empleado
WHERE NUMDEPT IN (1,2) AND NSS>33333
```

INTERVALOS. BETWEEN.

El operador **BETWEEN** sirve para recuperar campos que están dentro de un rango de valores entre dos límites especificados.

Sintaxis:

```
BETWEEN limite1 AND limite2
```

Consideraciones:

- Utiliza el operador BETWEEN con la palabra reservada NOT (NOT BETWEEN) para recuperar campos que no están dentro del intervalo.
- Utiliza el operador BETWEEN en vez de un predicado con un operador lógico AND con dos operadores de comparación.

En realidad, BETWEEN no hace nada que no se puede hacer igual con operadores de comparación, pero resulta más corto y legible.

Ejemplo

Seleccionar los datos de los empleados cuyo salario esté entre 200000 y 300000 maravedíes.

Solución.

```
SELECT nss, nombre, salario
FROM empleado
WHERE salario BETWEEN 200000 AND 300000
```

También se puede hacer de esta otra forma:

```
SELECT nss, nombre, salario
FROM empleado
WHERE salario >= 200000 AND salario <= 300000;
```

3. FUNCIONES DE AGREGACIÓN

Son funciones que su resultado se obtiene a partir de varios valores. También se llaman funciones de columnas. Acepta una columna entera de datos y produce un único valor que resume la columna.

Algunas de las más importantes son:

- **AVG(Expresión):** Calcula la media aritmética de una columna
 - **MIN(Expresión):** Selecciona el valor mínimo de una columna
 - **MAX(Expresión):** Selecciona el valor máximo de una columna
 - **SUM(Expresión):** Calcula la suma de los valores de una columna
 - **COUNT (Expresión):** Cuenta el número de valores de una columna
 - **COUNT (*):** Cuenta las filas de una consulta
- ❖ En donde **expresión** contiene el nombre del campo que desea contar. Los operandos de expresión pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL). Puede contar cualquier tipo de datos incluso texto.
- ❖ Aunque expresión puede realizar un cálculo sobre un campo, Count simplemente cuenta el número de registros sin tener en cuenta qué valores se almacenan en los registros. La función Count no cuenta los registros que tienen campos NULL a menos que expresión sea el carácter comodín asterisco (*). Si utiliza un asterisco, Count calcula el número total de registros, incluyendo aquellos que contienen campos NULL. Count(*) es considerablemente más rápida que Count(Campo).
- ❖ No se debe poner el asterisco entre comillas ('*').

“Lista el sueldo mínimo, máximo y medio de los empleados, y el número total de estos”

```
SELECT MIN (SALARIO) , MAX (SALARIO) , AVG (SALARIO) , COUNT (*)
FROM EMPLEADO
```

¿Cuántos nombres diferentes hay y cuantos empleados hay en la tabla empleado?

```
SELECT COUNT (DISTINCT NOMBRE) AS NOMBRESDIFERENTES ,
COUNT (*) AS TOTALPERSONAS
FROM EMPLEADO
```

¿Cuántas mujeres nacieron a partir de la fecha (01-01-1970)?

```
SELECT COUNT (*) FROM EMPLEADO
WHERE SEXO='F' AND FECHANAC>= ('01-01-1970')
```

¿Cuántas personas comienzan alguno de sus apellidos por alguna letra comprendida entre C y la M?

```
SELECT COUNT(*) FROM EMPLEADO
WHERE APEL1 LIKE '[C-M]%' OR APEL2 LIKE '[C-M]%'
```

¿Cuánto tendrá que pagar la empresa a los empleados del departamento 2, si a cada empleado le paga un plus de 5000 pts?

```
SELECT SUM (SALARIO+5000) FROM EMPLEADO
WHERE NUMDEPT=2
```

¿Cuántos empleados no tienen un supervisor directo?

```
SELECT COUNT(*) FROM EMPLEADO
WHERE NSSUP IS NULL
```

- ❖ Cuando se utiliza funciones agregadas en el SELECT, no se pueden utilizar nombres de columnas en la misma instrucción, ya que una **función agregada devuelve un único valor** y una **referencia a una columna devuelve un valor para cada fila**.

ERROR



```
SELECT MIN(SALARIO), MAX(SALARIO), NOMBRE FROM EMPLEADO
WHERE SEXO='F'
```

NOTA: los valores nulos pueden causar que las funciones agregadas devuelvan resultados inesperados. Por ejemplo, si ejecuta una función SELECT que incluye una función COUNT en una columna que tiene 18 filas, dos de las cuales continene valores nulos, la función COUNT devuelve el resultado de 16. SQL SERVER 7.0 ignora las dos filas que contienen valores nulos. Por tanto, si se decide utilizar funciones de agregados con valores nulos, se debe considerar los siguientes hechos:

Todas las funciones de agregadas de SQL SERVER, con la excepción de la función COUNT(*), ignoran los valores nulos en la columna.

EJ:

```
SELECT count(nssup) from empleado
```

Resultado: 10

```
SELECT count(*) from empleado
```

Resultado:12

4. CONSULTAS AGRUPADAS (CLÁUSULA GROUP BY)

Permite dividir la tabla en diferentes conjuntos de datos. En orden a realizar este agrupamiento añadiremos dos nuevas cláusulas a la sentencia SELECT

- ❑ La cláusula **GROUP BY** forma grupos con filas que tengan algunos valores comunes, de manera que la sentencia SELECT producirá un solo resultado para cada grupo de filas
- ❑ La cláusula **HAVING** establece condiciones para filtrar los grupos una vez que han sido formados.

⇒ La cláusula GROUP BY:

Suele utilizarse en combinación con funciones de agregación con el objetivo de obtener un resultado de la consulta para cada grupo, en lugar de un único resultado para la totalidad de la tabla. Se escribe detrás de la cláusula WHERE, si esta existe. Veamos su formato de utilización:

GROUP BY columna1 [, columna2]

donde columna1, ... son las llamadas columnas de agrupamiento.

Esta sentencia indica que se han de agrupar las filas de la tabla de tal manera que todas las que posean iguales valores en las columnas de agrupamiento forman un grupo. Puede haber grupos que sólo tengan una fila.

Una vez formados los grupos, para cada uno de ellos se evalúan las expresiones de la cláusula SELECT. Por tanto, cada uno produce una fila en la tabla final resultante. Veamos un ejemplo:

```
SELECT MIN (SALARIO) , MAX (SALARIO) , AVG (SALARIO) , COUNT (*) FROM EMPLEADO
```

Obtenemos el siguiente resulta: un único valor por cada función de agregado:

salario mínimo	Salario Máximo	Salario medio	Numero de empleados
120000	420000	193750	12

```
SELECT MIN (SALARIO) , MAX (SALARIO) , AVG (SALARIO) , COUNT (*) , NUMDEPT  
FROM EMPLEADO  
GROUP BY NUMDEPT
```

Obtenemos los siguientes resultados:

salario mínimo	Salario Máximo	Salario medio	Numero de empleados	Numdept
----------------	----------------	---------------	---------------------	---------

120000	420000	197857	7	1
140000	200000	163333	3	2
150000	300000	225000	2	3

Cuando se utiliza la cláusula **GROUP BY** se deben considerar los siguientes hechos:

- Se **produce una fila de valores para cada grupo definido**. Sólo se devuelven filas únicas para cada grupo, no devuelve información detallada.
- Todas las **columnas que estén especificadas en la lista de selección deben estar incluidas en la cláusula GROUP BY**. Las columnas incluidas en la cláusula GROUP BY no necesitan estar especificadas en la lista de selección.
- Si se incluye una cláusula WHERE, sólo se agrupan las filas que satisfagan las condiciones de la cláusula WHERE.
- Si se utiliza la cláusula GROUP BY en columnas que contienen valores nulos, los valores nulos son procesados como un grupo. Para excluir las filas con valores nulos se utiliza el operador IS [NOT] NULL en la cláusula WHERE.

⇒ **La cláusula HAVING**

Es una cláusula opcional que permite descartar grupos formados con una cláusula GROUP By aplicando a los grupos una o más condiciones de manera análoga a como WHERE filtra las filas individualmente.

Una importante característica de HAVING que la hace a veces preferible a WHERE es que en el predicado de la primera es **posible incluir funciones de agregado**, cosa que no podemos hacer en la segunda.

HAVING condición

Indica que después de haber separado las filas de la tabla en uno o varios grupos, se descarten aquellos que no satisfagan la condición.

La cláusula **GROUP BY antecede a la HAVING**

Se utiliza la cláusula WHERE para excluir aquellas filas que no se desea agrupar y la cláusula HAVING para filtrar los registros una vez agrupados.

EJ: Lista cuanto hay que pagar a los empleados de cada departamento excepto los del departamento 2.

```
SELECT SUM(SALARIO) , NUMDEPT FROM EMPLEADO
GROUP BY NUMDEPT
HAVING NUMDEPT <>2
```

Ej: Para los departamentos que tengan más de 2 empleados, lista el número de empleados y cuanto hay que pagar a sus empleados.

```
SELECT COUNT(*) AS [TOTAL EMPLEADOS], SUM(SALARIO) AS [TOTAL SALARIO],
NUMDEPT FROM EMPLEADO
GROUP BY NUMDEPT
HAVING COUNT(*) > 2
```

Resultado:

Total empleados	Total salario	numdept
7	1385000	1
3	490000	2
(2 filas afectadas)		

Sentencia SELECT con agrupamiento

Los pasos que se siguen en la ejecución de una sentencia SELECT con agrupamiento son los siguientes:

- ☐ Se selecciona la tabla especificada en **FROM**.
- ☐ Se eliminan de la tabla resultante las que no satisfagan la condición de la cláusula **WHERE**.
- ☐ Se forman grupos según las columnas de agrupamiento en la cláusula GROUP BY.
- ☐ Se descartan los grupos que no satisfagan la condición especificada en **HAVING**.
- ☐ Se evalúan las expresiones de la Cláusula **SELECT** para cada grupo, obteniendo una fila diferente para cada uno de ellos. Se eliminan las filas repetidas si se especifica la palabra DISTINCT.
- ☐ Se presenta el resultado ordenado según la cláusula **ORDER BY**.

Ej: Halla el salario mínimo, el salario máximo, el promedio del salario, el número de empleados de cada departamento sólo de los empleados que son chicos.

```
SELECT MIN(SALARIO), MAX(SALARIO), AVG(SALARIO), COUNT(*), NUMDEPT
FROM EMPLEADO
WHERE SEXO='M'
GROUP BY NUMDEPT
```

Ej: Halla el salario mínimo, el salario máximo, el promedio del salario, el número de empleados de cada departamento sólo de los empleados que son chicos y no pertenezcan al departamento 2

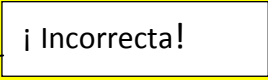
```
SELECT MIN(SALARIO), MAX(SALARIO), AVG(SALARIO), COUNT(*), NUMDEPT
FROM EMPLEADO
WHERE SEXO='M'
GROUP BY NUMDEPT
HAVING NUMDEPT <> 2
```

Ej: Para los departamentos que tengan un número de empleados varones mayor que 2, halla el salario mínimo, el salario máximo, el promedio del salario, el número de empleados de cada departamento sólo de los empleados que son chicos.

```
SELECT MIN(SALARIO) , MAX(SALARIO) , AVG(SALARIO) , COUNT(*) , NUMDEPT
FROM EMPLEADO
WHERE SEXO='M'
GROUP BY NUMDEPT
HAVING COUNT(*)>2
```

Las siguiente instrucción sería incorrecta:

```
SELECT MIN(SALARIO) , MAX(SALARIO) , AVG(SALARIO) , COUNT(*) , NUMDEPT
FROM EMPLEADO
GROUP BY NUMDEPT
HAVING COUNT(*)>2 and SEXO='M'
```



5. CONSULTAS MULTITABLAS. COMPOSICIONES (JOIN).

SQL permite recuperar datos procedentes de dos o más tablas de la base de datos.

Como dentro de una tabla, los nombres de las columnas son todos diferentes, cuando una sentencia SELECT opera sobre una sola tabla basta el nombre de la columna para designarla sin ambigüedad. Por el contrario, si interviene más de una tabla, puede ocurrir que algún nombre de columna se repita en más de una de ellas. En este caso, para referirnos sin ambigüedad a una columna determinada, hay que indicar a que tabla pertenece. Para ello, se precede al nombre del campo con la tabla que pertenece separado por un punto.

EJ: EMPLEADO.NUMDEPT DEPARTAMENTO.NUMDEPT

Consultas Padre/Hijo:

Las consultas multitablas más comunes implica a dos tablas que tienen una relación padre/hijo (1 a N). La tabla padre contiene la clave primaria y la tabla hijo la clave foránea.

1) Composición interna:

Una de las maneras más fáciles de consulta multitabla es especificar la lista de tablas en la **cláusula FROM** y una condición en la cláusula WHERE que compare la **clave primaria y la clave foránea**. En este caso, se emparejan las filas de las tablas que tienen igual valor.

Ej: Listar los nombres, apellidos y nombre del departamento donde trabaja cada empleado.

```
SELECT NSS,NOMBRE,NOMBREDEP
FROM EMPLEADO,DEPARTAMENTO
WHERE EMPLEADO.NUMDEPT=DEPARTAMENTO.NUMDEP
```

NSS	NOMBRE	NOMBREDEP

11111	José	Investigacion
11133	Carmen	Investigacion
33322	Maria	Investigacion
33333	Ana	Investigacion
44444	Jaime	Investigacion
66666	José	Investigacion
99999	Ana	Investigacion

11122	Federico	Administracion
55555	Luis	Administracion
77777	Ana	Administracion
22222	Federico	Dirección
88888	Pedro	Dirección
(12 filas afectadas)		

ESTRUCTURA DE UNA COMPOSICIÓN

Sql es un lenguaje de manipulación de datos de los SGBR basado en el lenguaje algebraico, que contiene una serie de operadores –operadores algebraicos- que operan sobre las tablas de un esquema relacional. Los operadores algebraicos básicos son: unión, diferencia, selección, proyección y producto cartesiano.

Cuando se realiza una consulta multitabla se realiza una composición que consiste en una nueva tabla donde se realiza el **producto cartesiano**. Esta nueva tabla estará formada por las filas que resultan de todas las combinaciones posibles de una fila de la primera tabla adherida a una fila de la segunda tabla. En general, el resultado del **producto cartesiano** de $R1(A1, A2, \dots, An)$ y de $R2(B1, B2, \dots, Bm)$ es una relación $R3$ con $n+m$ atributos $R3(A1, A2, \dots, An, B1, B2, \dots, Bm)$. La relación resultante $R3$ tiene una tupla por cada combinación de tuplas: una de $R1$ y una de $R2$, por tanto si $R1$ tiene X tuplas y $R2$ tiene Y tuplas, $R3$ tendrá $X*Y$ tuplas.

Las siguientes figuras muestra un ejemplo:

SELECT * FROM EMPLEADO, DEPARTAMENTO

EMPLEADO

NSS	Nombre	Apel1	Apel2	Salario	Numdept
1111	José	Nieto	Silva	420000	1
1112	Federic	Fuertes	Serantes	140000	2
1113	Carmen	Prieto	Gonzalez	175000	1
2222	Federic	García	López	150000	3
3332	Maria	Ramos	Perez	120000	1
3333	Ana	Sanchez	Valdés	120000	1
4444	Jaime	Silva	Gonzalez	200000	1
5555	Luis	Gonzalez	Mouro	150000	2
6666	José	Vilan	Silva	200000	1
7777	Ana	López	Villa	200000	2
8888	Pedro	Gonzalez	Jimenez	300000	3
9999	Ana	Silva	López	150000	1

DEPARTAMENTO

Numdep	Nombredp	Numempdep
1	Investigacion	16
2	Administracion	14
3	Dirección	2

Se realiza una composición con todas las combinaciones posibles entre las filas de ambas tablas dando lugar a una tabla de 9 campos (6 –Empleado + 3-Departamento) y 36 filas (12-Empleado X 3-Departamento)

El resultado sería:

NSS	Nombre	Apel1	Apel2	Salario	Numdep	Nombredp	Numempdep
1111	José	Nieto	Silva	420000	1	Investigacion	16
1111	José	Nieto	Silva	420000	2	Administracion	14
1111	José	Nieto	Silva	420000	3	Direccion	2
1112	Federic	Fuertes	Serantes	140000	1	Investigacion	16
1112	Federic	Fuertes	Serantes	140000	2	Administracion	14
1112	Federic	Fuertes	Serantes	140000	3	Direccion	2
1113	Carmen	Prieto	Gonzalez	175000	1	Investigacion	16
1113	Carmen	Prieto	Gonzalez	175000	2	Administracion	14
1113	Carmen	Prieto	Gonzalez	175000	3	Dirección	2
2222	Federic	García	López	150000	1	Investigacion	16
2222	Federic	García	López	150000	2	Administracion	14
2222	Federic	García	López	150000	3	Dirección	2
.....
7777	Ana	López	Villa	200000	1	Investigacion	16
7777	Ana	López	Villa	200000	2	Administracion	14
7777	Ana	López	Villa	200000	3	Dirección	2
8888	Pedro	Gonzalez	Jimenez	300000	1	Investigacion	16
8888	Pedro	Gonzalez	Jimenez	300000	2	Administracion	14
8888	Pedro	Gonzalez	Jimenez	300000	3	Dirección	2
9999	Ana	Silva	López	150000	1	Investigacion	16
9999	Ana	Silva	López	150000	2	Administracion	14
9999	Ana	Silva	López	150000	3	Dirección	2

El producto cartesiano nunca se utiliza como una operación significativa por sí sola, sino que se utiliza junto con una condición en la **cláusula WHERE** para emparejar cada fila de la primera tabla con su correspondiente en la segunda tabla y así restringir las filas que aparecen en los resultados de la consulta.

```
SELECT * FROM EMPLEADO, DEPARTAMENTO
WHERE EMPLEADO.NUMDEPT=DEPARTAMENTO.NUMDEP
```

El resultado sería:

nss	nombre	apel1	apel2	salario	numdept	nombredp	numempdep
11111	José	Nieto	Silva	420000	1	Investigacion	16
11122	Federico	Fuertes	Serantes	140000	2	Administracion	14
11133	Carmen	Prieto	Gonzalez	75000	1	Investigacion	16
22222	Federico	Garcia	López	150000	3	Dirección	2
33322	Maria	Ramos	Perez	120000	1	Investigacion	16
33333	Ana	Sanchez	Valdés	120000	1	Investigacion	16
44444	Jaime	Silva	Gonzalez	200000	1	Investigacion	16
55555	Luis	Gonzalez	Mouro	150000	2	Administracion	14
66666	José	Vilan	Silva	200000	1	Investigacion	16
77777	Ana	López	Villa	200000	2	Administracion	14
88888	Pedro	Gonzalez	Jimenez	300000	3	Dirección	2
99999	Ana	Silva	López	150000	1	Investigacion	16

(12 filas afectadas)

Composiciones con criterios de selección de fila:

La condición que se especifica en las columnas de emparejamiento en una consulta multitabla puede combinarse con otras condiciones de búsqueda para restringir aún más los contenidos de los resultados.

Ej: Listar los campos nss, nombre, apellidos, salario, departamento y nombre del departamento de las mujeres con salario mayores que 1300000

```
SELECT NSS,NOMBRE,APEL1,APEL2,SALARIO,NUMDEPT,NOMBREDEP
FROM EMPLEADO, DEPARTAMENTO
WHERE EMPLEADO.NUMDEPT=DEPARTAMENTO.NUMDEP AND
SEXO='F' AND SALARIO>1300000
```

El resultado sería:

nss	nombre	apel1	apel2	salario	numdept	nombredep
11133	Carmen	Prieto	Gonzalez	175000	1	Investigacion
77777	Ana	López	Villa	200000	2	Administracion
99999	Ana	Silva	López	150000	1	Investigacion

MÚLTIPLES COLUMNAS DE EMPAREJAMIENTO

SQL puede **combinar datos de tres o más tablas** utilizando las mismas técnicas básicas utilizadas en la consulta de dos tablas

Ej: listar el nombre y apellidos de cada empleado junto con el nombre de cada proyecto que tiene el empleado asignado y el nombre del departamento que está asignado cada proyecto, ordenados por el nss

```
SELECT NOMBRE,APEL1,APEL2,NOMBREPROY,NOMBREDEP
FROM EMPLEADO,EMPLEADOPROYECTO,PROYECTO,DEPARTAMENTO
WHERE EMPLEADO.NSS=EMPLEADOPROYECTO.NSS AND
EMPLEADOPROYECTO.NUMPROY=PROYECTO.NUMPROY AND
PROYECTO.NUMDEP=DEPARTAMENTO.NUMDEP
ORDER BY EMPLEADO.NSS
```

El resultado sería:

NOMBRE	APEL1	APEL2	NOMBREPROY	NOMBREDEP
José	Nieto	Silva	ProductoX	Investigacion
José	Nieto	Silva	ProductoY	Investigacion
José	Nieto	Silva	Reorganización	Dirección
Federico	García	López	ProductoX	Investigacion
Federico	García	López	ProductoZ	Investigacion
Federico	García	López	Nuevaspresta	Administracion
Jaime	Silva	Gonzalez	ProductoX	Investigacion
Jaime	Silva	Gonzalez	Automatización	Administracion
Jaime	Silva	Gonzalez	ProductoW	Investigacion
José	Vilan	Silva	ProductoY	Investigacion
Ana	López	Villa	ProductoW	Investigacion
Pedro	Gonzalez	Jimenez	ProductoW	Investigacion
Ana	Silva	López	ProductoX	Investigacion
Ana	Silva	López	ProductoY	Investigacion

Ej: listar los nombres de los proyectos con el nombre del departamento que está asignado cada proyecto del empleado con número de seguridad social 11111

```
SELECT NOMBREPROY, NOMBREDEP
FROM EMPLEADO, EMPLEADOPROYECTO, PROYECTO, DEPARTAMENTO
WHERE EMPLEADO.NSS=EMPLEADOPROYECTO.NSS AND
EMPLEADOPROYECTO.NUMPROY=PROYECTO.NUMPROY AND
PROYECTO.NUMDEP=DEPARTAMENTO.NUMDEP AND EMPLEADO.NSS=11111
```

El resultado sería:

NOMBREPROY	NOMBREDEP
-----	-----
ProductoX	Investigacion
ProductoY	Investigacion
Reorganización	Dirección

Ej: Lista para cada empleado, el número de proyectos que tiene asignados:

```
SELECT EMPLEADO.NSS, NOMBRE, APEL1, APEL2, COUNT (*) AS [N. DE PROYECTOS]
FROM EMPLEADO, EMPLEADOPROYECTO
WHERE EMPLEADO.NSS=EMPLEADOPROYECTO.NSS
GROUP BY EMPLEADO.NSS, NOMBRE, APEL1, APEL2
```

ALIAS

En la cláusula FROM se puede simplificar el uso de los nombres de las tablas mediante la asignación de nombres locales o alias (también llamados de correlación), que deberán ser usados en lo sucesivo en la sentencia. Dichos alias se definen escribiendo detrás de cada nombre de tabla el alias en cuestión:

Sintaxis: FROM tabla alias

FROM tabla AS alias

Ej: Para cada departamento, lista el salario máximo de sus empleados y el número de empleados:

```
SELECT MAX (SALARIO) , NUMDEPT, count (*)
FROM EMPLEADO
GROUP BY NUMDEPT
```

Ej.: Para cada departamento, listar el salario máximo de sus empleados, el nombre del departamento y numero de empleados.

```
SELECT MAX (SALARIO) , P.NUMDEPT, NOMBREDEPT, count (*)
FROM DEPARTAMENTO P, EMPLEADO E
WHERE E.NUMDEPT=P.NUMDEPT
GROUP BY P.NUMDEPT, NOMBREDEPT
```

Ej: Para departamento que tenga más de 2 empleados varones trabajando, listar su salario máximo y el número de empleados chicos.

```
SELECT MAX (SALARIO) , NUMDEPT, count (*)
FROM EMPLEADO
WHERE SEXO='M'
GROUP BY NUMDEPT
HAVING COUNT (*)>2
```

Ej: Para cada departamento que trabajen más de dos empleados, listar el salario máximo de sus empleados, el nombre del departamento y el numero de empleados..

```
SELECT MAX (SALARIO) , P.NUMDEPT, NOMBREDEPT, COUNT (*)
FROM DEPARTAMENTO P, EMPLEADO E
WHERE E.NUMDEPT=P.NUMDEPT
GROUP BY P.NUMDEPT, NOMBREDEPT
HAVING COUNT (*)>2
```

EJ: Hallar la media de los salarios de los departamentos cuyo salario mínimo supera a 130000 pts., visualizando también el nombre del departamento

```
SELECT AVG (E.SALARIO) , E. NUMDEPT, NOMBREDEPT
FROM EMPLEADO E, DEPARTAMENTO D
WHERE E.NUMDEPT=D.NUMDEPT
GROUP BY E.NUMDEPT, D.NOMBREDEPT
HAVING MIN(SALARIO)>130000
```

Ej: Mostrar los nombres y apellidos de los empleados que trabajan en algún proyecto que el nombre del proyecto tenga los caracteres PROD sin mostrar filas duplicadas.

```
SELECT DISTINCT NOMBRE, APEL1, APEL2
FROM EMPLEADO E, EMPLEADOPROYECTO EP, PROYECTO P
WHERE E.NSS=EP.NSS AND EP.NUMPROY=P.NUMPROY AND NOMBREPROY LIKE '%PROD%'
```

2.-Otra forma de composición interna: **INNER JOIN**

Las composiciones internas unen tablas comparando los valores en las columnas que son comunes a ambas tablas, devolviendo solo las filas que coinciden con las condiciones de combinación.

- Hasta ahora especificábamos las tablas de la combinación en la cláusula FROM y las condiciones de combinación en la cláusula WHERE.

```
SELECT NSS,NOMBREDEPT FROM EMPLEADO, DEPARTAMENTO
WHERE EMPLEADO.NUMDEPT=DEPARTAMENTO.NUMDEP
```

- Otra forma de composición interna: **INNER JOIN**.

formato:

```
SELECT columnas
FROM tabla1 INNER JOIN tabla2 ON condiciones de combinación
```

Se agregan dos palabras claves:

- La palabra **INNER** que especifica que la combinación sea interna, es decir, emparejando valores en las tablas que coinciden.
- **JOIN** que especifica las tablas.
- **ON PARA ESPECIFICAR LOS CAMPOS QUE SE UTILIZAN PARA COMBINAR.**

Ej:

```
SELECT NSS,NOMBREDEP FROM EMPLEADO INNER JOIN DEPARTAMENTO
ON EMPLEADO.NUMDEPT=DEPARTAMENTO.NUMDEP
```

3.- COMPOSICIONES EXTERNAS:

Al igual que las composiciones internas, las combinaciones externas devuelven filas combinadas que coinciden con la condición de combinación. No obstante, además de las filas que coinciden con la condición de combinación, las composiciones externas izquierda y derechas también devuelven filas no coincidentes de una de las tablas y las composiciones externas totales devuelven filas no coincidentes de ambas filas. Las columnas de la tabla primera, en las filas no coincidentes tiene datos, pero en las columnas de la otra tabla, contiene valores NULL (desconocidos)

El estándar SQL1 no da soporte a las composiciones externas; solo especifica la composición interna utilizando WHERE para las condiciones de combinación.

El estándar SQL2 da soporte tanto a las combinaciones externas como internas, y muchos productos como SQL SERVER, ACCESS, ORACLE, SQL BASE han implementado este tipo de composiciones.

Las composiciones externas pueden ser:

- A LA **IZQUIERDA**: LEFT [OUTER] JOIN
- A LA **DERECHA**: RIGHT [OUTER] JOIN
- **TOTAL**: FULL [OUTER] JOIN

Composición externa izquierda:

Realiza una composición interna incluyendo además incluyendo valores NULL a las columnas de las filas de la primera tabla (Izquierda) que no se emparejen con los valores de la segunda tabla (Derecha). No se incluyen las filas no emparejadas de la segunda tabla(derecha).

```
SELECT EMPLEADO.NSS, NUMPROY FROM EMPLEADO LEFT JOIN EMPLEADOPROYECTO  
ON EMPLEADO.NSS=EMPLEADOPROYECTO.NSS
```

EMPLEADO

NSS	Nombre	Apel1	Apel2	Salario	Numdept
1111	José	Nieto	Silva	420000	1
1112	Federic	Fuertes	Serantes	140000	2
1113	Carmen	Prieto	Gonzalez	175000	1
2222	Federic	García	López	150000	3
3332	María	Ramos	Perez	120000	1
3333	Ana	Sanchez	Valdés	120000	1
4444	Jaime	Silva	Gonzalez	200000	1
5555	Luis	Gonzalez	Mouro	150000	2
6666	José	Vilan	Silva	200000	1
7777	Ana	López	Villa	200000	2
8888	Pedro	Gonzalez	Jimenez	300000	3
9999	Ana	Silva	López	150000	1

EMPLEADO-PROYECTO

NSS	NUMPROY	NUMHORAS
11111	1	30
11111	2	24
11111	4	13
22222	1	25
22222	3	30
22222	6	14
44444	5	17
44444	7	35
44444	1	25
66666	2	25
77777	7	35
88888	7	40
99999	1	20
99999	2	20

El resulta de la ejecución de la anterior sentencia SQL sería:

NSS	NUMPROY
11111	1
11111	2
11111	4
11122	NULL
11133	NULL
22222	1
22222	3

Se realiza una composición interna con los valores del atributo NSS coincidente en las tablas EMPLEADO y EMPLEADOPROYECTO, más las filas que no se emparejan de la tabla EMPLEADO tomando en este caso los valores NULL

```

22222 6
33322 NULL
33333 NULL
44444 1
44444 5
44444 7
55555 NULL
66666 2
77777 7
88888 7
99999 1
99999 2

```

Composición externa derecha:

Realiza una composición interna incluyendo además incluyendo valores NULL a las columnas de las filas de la segunda tabla (derecha) que no se emparejen con los valores de la primera tabla (izquierda). No se incluyen las filas no emparejadas de la primera tabla (izquierda).

```

SELECT EMPLEADO.NSS, NUMPROY FROM EMPLEADOPROYECTO RIGHT JOIN EMPLEADO
ON EMPLEADOPROYECTO.NSS = EMPLEADO.NSS

```

Para mostrar qué empleados no han participado en ningún proyecto, podría emplear entonces la siguiente sentencia (no haría falta mostrar la columna empleadoproyecto.nss, sólo emplearla para la condición WHERE:

```

SELECT EMPLEADO.NSS, empleadoproyecto.nss
FROM EMPLEADOPROYECTO RIGHT JOIN EMPLEADO
ON EMPLEADOPROYECTO.NSS = EMPLEADO.NSS
where empleadoproyecto.nss is null;

```

Este mismo resultado podría obtenerlo mediante una subconsulta:

```

select nss from empleado where nss not in
(select nss from empleadoproyecto);

```

Ej. Visualizar el número de proyectos en el que participa cada empleado, y que aparezca 0 para aquellos que no participan en ninguno. En este caso, el argumento de la función count es importante que sea la columna que se indica. Esta columna tomará valor null en la combinación para aquellos empleados que no participen en ningún proyecto. Si aplicamos count a otra columna no mostrará el mismo resultado. También

es importante la columna de agrupación, debe ser la que se indica, para que aparezcan todos los empleados, y no sólo aquellos que están en la tabla EmpleadoProyecto.

```
SELECT EMPLEADO.NSS, count(empleadoproyecto.nss) as NumProyectos
FROM EMPLEADOPROYECTO RIGHT JOIN EMPLEADO
ON EMPLEADOPROYECTO.NSS = EMPLEADO.NSS
group by empleado.nss;
```

Composición externa total:

Realiza una composición interna incluyendo además valores NULL a las columnas de las filas de la primera tabla (Izquierda) que no se emparejen con los valores de la segunda tabla (Derecha) y los valores NULL de las columnas de las filas de la segunda tabla (derecha) que no se emparejen con los valores de la primera tabla (Izquierda).

En la práctica, las composiciones externas izquierda y derecha son más útiles que las composiciones totales, sobre todo en las composiciones Clave primaria/clave foránea.

Composición de mas de dos tablas

Cuando se combina más de dos tablas, es más fácil considerar cada combinación independiente. Por ejemplo, si se combinan las tablas A,B,C, la primera composición combina A y B para producir un conjunto de resultados, este conjunto de resultados se combina con la tercera tabla C para producir el resultado final.

Ej: Para cada empleado, listar nss, nombre, número y nombre del proyecto asignado.

```
SELECT EMPLEADO.NSS, NOMBRE, APEL1, APEL2, EMPLEADOPROYECTO.NUMPROY,
NOMBREPROY
FROM EMPLEADO INNER JOIN EMPLEADOPROYECTO
ON EMPLEADO.NSS=EMPLEADOPROYECTO.NSS INNER JOIN PROYECTO ON
EMPLEADOPROYECTO.NUMPROY=PROYECTO.NUMPROY
```

6. SUBCONSULTAS

Una subconsulta es una instrucción **SELECT** anidada dentro de una instrucción **SELECT**, **INSERT**, **UPDATE** o **DELETE**. Permite utilizar los resultados de una columna como parte de otra.

La consulta que contiene la subconsulta se denomina normalmente **Externa** y la subconsulta se denomina **Interna**.

Una subconsulta es una consulta que aparece dentro de la cláusula **WHERE** o **HAVING**.

Las subconsultas proporcionan un método eficaz de gestionar peticiones que se expresan en términos de los resultados de otras consultas.

Ej: Por ejemplo si queremos saber los datos de los empleados del departamento 1

```
SELECT * FROM EMPLEADO
WHERE NUMDEPT=1
```

Ej: Por ejemplo si queremos saber los datos de los empleados del departamento de investigación

```
SELECT * FROM EMPLEADO
WHERE NUMDEPT= ?? ----->
```

No sabemos el número de departamento de investigación. Los ?? deberán ser sustituidos y deberá ser igual "al número del departamento de investigación"

```
SELECT * FROM EMPLEADO
WHERE NUMDEPT= (SELECT NUMDEP FROM DEPARTAMENTO
                WHERE NOMBREDEP= 'INVESTIGACION')
```

EMPLEADO

NSS	Nombre	Apel1	Apel2	Salario	Numdept
1111	José	Nieto	Silva	420000	1
1112	Federic	Fuertes	Serantes	140000	2
1113	Carmen	Prieto	Gonzalez	175000	1
2222	Federic	García	López	150000	3
3332	María	Ramos	Perez	120000	1
3333	Ana	Sanchez	Valdés	120000	1
4444	Jaime	Silva	Gonzalez	200000	1
5555	Luis	Gonzalez	Mouro	150000	2
6666	José	Vilan	Silva	200000	1
7777	Ana	López	Villa	200000	2
8888	Pedro	Gonzalez	Jimenez	300000	3
9999	Ana	Silva	López	150000	1

DEPARTAMENTO

Numdep	Nombredep	Numempdep
1	Investigacion	16
2	Administracion	14
3	Dirección	2

Se ejecuta la subconsulta y se obtiene un único valor, en este ejemplo es 1 y utiliza el valor devuelto en la consulta externa

Con frecuencia, una consulta que contiene subconsultas puede ser escrita como una consulta multitable.

```
SELECT * FROM EMPLEADO E, DEPARTAMENTO D
WHERE E.NUMDEPT=D.NUMDEP AND NOMBREDEP='Investigacion'
```

TIPOS DE SUBCONSULTAS:

Las subconsultas pueden ser:

- **ANIDADAS**
- **CORRELACIONADAS**

SUBCONSULTAS ANIDADAS

Las **subconsultas anidadas** pueden devolver un **único valor** o una **lista de valores**. En estas consultas, la subconsulta se ejecuta una única vez y entonces utiliza el valor o la lista de valores de la subconsulta en la consulta externa.

➤ SUBCONSULTAS ANIDADAS QUE DEVUELVEN UN VALOR ÚNICO

Compara el valor de una expresión con el **valor único** producido por una subconsulta y devuelve un resultado TRUE si la comparación es cierta. Los operadores utilizados son (=, <, >, <=, >=)

Se puede especificar una función de agregado o especificar una columna o expresión para devolver un único valor. Cada consulta anidada se ejecuta una sola vez.

Si la subconsulta anidada se devuelve más de un valor se produce un error.

Puede tener tantos niveles de subconsulta como necesite, no hay límite.

Ej: Lista los nombres de empleados cuyo salario supera a la media del salario de todos los empleados.

```
SELECT NOMBRE, APEL1, APEL2 FROM EMPLEADO
WHERE SALARIO > (SELECT AVG (SALARIO) FROM EMPLEADO)
```

Lista todos los proyectos que dirige el departamento cuyo gerente es el 11111

```
SELECT * FROM PROYECTO
WHERE NUMDEP = (SELECT NUMDEP FROM DEPARTAMENTO
                WHERE NSSGERENTE='11111')
```

Lista todos los proyectos que dirige el departamento cuyo gerente es Jose Nieto Silva

```
SELECT * FROM PROYECTO
WHERE NUMDEP = (SELECT NUMDEP FROM DEPARTAMENTO
                WHERE NSSGERENTE= (SELECT NSS FROM EMPLEADO
                                   WHERE NOMBRE='José' AND
                                   APEL1='Nieto' AND APEL2='Silva'))
```

➤ SUBCONSULTAS ANIDADAS QUE DEVUELVEN UN CONJUNTO DE VALORES

Compara el valor de una expresión con un conjunto de valores producidos por una subconsulta y devuelve un resultado TRUE si el valor coincide con uno del conjunto de valores producidos. Cuando se utiliza subconsultas anidadas que devuelven una lista de valores, se debe considerar lo siguiente:

- Se utiliza el operador **IN**
- La consulta anidada se evalúa una única vez

Ej: Lista los empleados de los departamentos de investigación y administración.

```
SELECT * FROM EMPLEADO
WHERE NUMDEPT IN (SELECT NUMDEP FROM DEPARTAMENTO
                  WHERE NOMBREDEP='Investigacion' OR
                  NOMBREDEP='Administracion' );
```

NOT IN → Devuelve el valor TRUE cuando el valor no coincide con algún valor del conjunto de valores generados por la subconsulta.

Ej: Listar el nombre de las mujeres que no trabajen en el departamento de investigación

```
SELECT * FROM EMPLEADO
WHERE NUMDEPT NOT IN (SELECT NUMDEP FROM DEPARTAMENTO
                      WHERE NOMBREDEP = 'Investigacion') AND SEXO='F';
```

TEST CUANTIFICATIVOS (ANY, ALL)

La subconsulta del test IN comprueba si el valor de una columna **es igual** alguno de los valores de la lista devuelta por la subconsulta.

SQL proporciona los test ANY y ALL para extender la anterior noción al resto de los operadores de comparación (>,>=,<,<=,<>=).

TEST ANY (SOME):

Se utiliza conjuntamente con uno de los seis operadores de comparación (>,>=,<,<=,<>=) para comparar un valor único con una lista de valores producidos por una subconsulta. Para efectuar el test, SQL utiliza el operador de comparación especificado para comparar el valor de test con cada valor de la columna, uno cada vez. Si alguna de las comparaciones individuales producen un resultado verdadero, el test ANY devuelve el resultado verdadero.

Listar los datos de los empleados cuyo salario es menor que algún salario de los empleados del departamento 2 sin incluir a los de este departamento.

```
SELECT NSS,NOMBRE,APEL1,APEL2,SALARIO FROM EMPLEADO
WHERE SALARIO < ANY (SELECT SALARIO FROM EMPLEADO
                     WHERE NUMDEPT=2) AND NUMDEPT<>2
```


RESULTADO:

nss	nombre	apel1	apel2	salario
11133	Carmen	Prieto	Gonzalez	175000
22222	Federico	García	López	150000
33322	Maria	Ramos	Perez	120000
33333	Ana	Sanchez	Valdés	120000
99999	Ana	Silva	López	150000

La subconsulta devuelve una lista con los salarios de los empleados del departamento 2. La cláusula Where compara el salario de cada empleado con todos los valores devueltos por la subconsulta. Si el salario menor que alguno de la lista devuelta, el test <ANY devuelve true y el empleado queda incluido en los resultados de la consulta.

El operador **IN** sería equivalente a **=ANY**.

La palabra **SOME** es una alternativa para **ANY** especificada por el estándar SQL ANSI/ISO. Generalmente, cualquiera de las dos palabras clave puede ser utilizada pero algunos SGBR no admiten **SOME**.

TEST ALL:

Se utiliza conjuntamente con uno de los seis operadores de comparación (>,>=,<,<=,<>=) para comparar un valor único con una lista de valores producidos por una subconsulta. Para efectuar el test, SQL utiliza el operador de comparación especificado para comparar el valor de test con cada valor de la columna, uno cada vez. Si **todas** comparaciones individuales producen un resultado verdadero, el test **ALL** devuelve el resultado verdadero.

Listar los datos de los empleados cuyo salario es mayor que todos los salarios de los empleados del departamento 2 sin incluir a los de este departamento.

```
SELECT NSS,NOMBRE,APEL1,APEL2,SALARIO FROM EMPLEADO
WHERE SALARIO > ALL (SELECT DISTINCT SALARIO FROM EMPLEADO
                        WHERE NUMDEPT=2) AND NUMDEPT<>2
```

RESULTADO:

NSS	NOMBRE	APEL1	APEL2	SALARIO
11111	José	Nieto	Silva	420000
88888	Pedro	Gonzalez	Jimenez	300000

La subconsulta devuelve una lista con los salarios de los empleados del departamento 2. La cláusula Where compara el salario de cada empleado con todos los valores devueltos por la subconsulta. Si el salario es mayor que todos los valores de la lista devuelta, el test >ALL devuelve true y el empleado queda incluido en los resultados de la consulta.