IES CHAN DO MONTE
C.S. de Desarrollo de Aplicaciones Multiplataforma
Módulo Base de datos

UNIDAD 5: COMO REALIZAR CONSULTAS

Índice

1. Consultas básicas

1.1 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.

Obtiene filas de la base de datos y permite realizar la selección de una o varias filas o columnas de una o varias tablas.

La sintaxis completa de la instrucción SELECT es compleja, aunque las cláusulas principales se pueden resumir como sigue:

```
SELECT listaSelección
[INTO nuevaTabla_]
FROM   origenTabla
[WHERE condiciónBúsqueda]
[GROUP BY expresiónAgruparPor]
[HAVING condiciónBúsqueda]
[ORDER BY expresiónOrden [ASC | DESC] ]
```

La sentencia SELECT permite:

- Obtener datos de ciertas columnas de una tabla (***proyección***)
- Obtener registros (filas) de una tabla de acuerdo con ciertos criterios (***selección***)
- Mezclar datos de tablas diferentes (***asociación, join***)
- Realizar cálculos sobre los datos
- Agrupar datos

El **orden de las cláusulas en la instrucción SELECT es importante.** Se puede omitir cualquiera de las cláusulas opcionales, pero, cuando se utilizan, deben aparecer en el orden apropiado.

Debido a la complejidad de la instrucción SELECT, se muestran elementos y argumentos detallados de la sintaxis de cada cláusula.

1.2 Selección de atributos

```
SELECT * | {[DISTINCT] columna | expresión [[AS] alias], ...}
FROM tabla;
```

Donde:

- *******. El asterisco significa que se seleccionan todas las columnas
- **DISTINCT**. Hace que no se muestren los valores duplicados.
- ***columna***. Es el nombre de una columna de la tabla que se desea mostrar
- ***expresión***. Una expresión válida SQL
- ***alias***. Es un nombre que se le da a la cabecera de la columna en el resultado de esta instrucción.

Ejemplos:

```
/* Selección de todos los registros de la tabla clientes */
SELECT * FROM Clientes;
/* Selección de algunos campos*/
SELECT nombre, apellido1, apellido2 FROM Clientes;
/* Selección de las distintas localidades de los clientes*/
SELECT DISTINCT localidad FROM Clientes;
```

Cláusula FROM

Especifica **las tablas de las que se van a obtener filas**.

La cláusula FROM es necesaria excepto cuando la lista de selección sólo contiene constantes, variables y expresiones aritméticas (no nombres de columna).

```
/* Obtiene la versión, fecha de compilación y sistema operativo de la instalación actual
de SQL Server */
SELECT @@version
```

Si aparece **el mismo nombre de columna en más de una de las tablas** que se incluyen en la consulta, el **nombre debe ser especificado por completo con el nombre de tabla o el alias**, como por ejemplo en **TablaA.Columna1.**

Si una tabla o consulta posee un **alias**, **se deberá utilizar el nombre del alias**, no el nombre actual de la tabla o consulta

```
/* Listar los NSS, nombres y nombre del departamento donde trabaja cada empleado*/
SELECT NSS,NOMBRE,NOMBREDEP
FROM EMPLEADOS as E, DEPARTAMENTOS as D --E y D son alias
WHERE E.NUMDEP=D.NUMDEP
```

1.3 Filas duplicadas (DISTINCT)

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.

```
/*Lista todos los salarios diferentes */
SELECT DISTINCT Salario FROM Empleados;
```

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 porque este es el comportamiento predeterminado**.

```
SELECT ALL Salario  FROM Empleados;
--Devuelve los mismos resultados
SELECT Salario FROM Empleados;
```

1.4 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.

Cuando se utilizan como expresión en una consulta SELECT, no modifican los datos originales sino que como resultado de la vista generada por SELECT, aparece una nueva columna.

AS: sirve para poner un **nombre a la expresión**. Si el nombre tiene espacios en blanco o no sigue las reglas para identificadores normales, se pone entre [].

Ej: "Listado de los empleados con el doble del salario que tienen."

```
/*Listado de los empleados con el doble del salario que tienen*/
SELECT Nombre, Apel1, Apel2, Salario, Salario * 2 AS DobleSalario
FROM Empleados
```

También podemos usar la sintaxis:

```
alias_columna  = <expresion>
```

Por ejemplo:

```
SELECT nombre, precio, precio_rebajado=precio-precio*0.40
FROM producto;
```

OPERADORES ARITMÉTICOS

Los operadores aritméticos ejecutan cálculos con columnas numéricas o constantes.
Transact-SQL soporta los operadores aritméticos de **multiplicación (*), división (/), suma (+), resta (-).** El operador **(%)**, que devuelve el **resto entero** de una división también se puede utilizar.
La prioridad de esos operadores es la normal: tienen **más prioridad la multiplicación (*) y división(/) y resto (%),** después la suma(+) y la resta(-).

En caso de **igualdad de prioridad**, se realiza primero la operación que esté más a la **izquierda**. Como es lógico se puede evitar cumplir esa prioridad usando **paréntesis**; el interior de los paréntesis es lo que se ejecuta primero.

Cuando una expresión aritmética se calcula sobre valores NULL, el resultado es el propio valor NULL.

1.5 Concatenación de textos

Todas las bases de datos incluyen algún operador para encadenar textos. En **SQLSERVER** es el signo **+** en Oracle son los signos **||.** Ejemplo (Oracle):

```
SELECT nombre, apellido1, nombre + ” “ + apellido1 AS "Nombre completo"
FROM personal;
```

El resultado sería como el siguiente:

nombre	apellido1	Nombre completo
Alejandro	López	Alejandro López
Rebeca	Cuerva	Rebeca Cuerva
María Teresa	García	María Teresa García
Hugo	Calvo	Hugo Calvo

1.6 Devolver las N primeras Filas (TOP)

Podemos **limitar el número de registros** que devuelve una consulta a través de la cláusula **TOP**. La cláusula TOP admite como parámetros un valor numérico entero o un porcentaje. Su sintaxis es la siguiente:

```
[TOP <expresion> [PERCENT ][WITH TIES]]
```

- Si **no se incluye la cláusula ORDER BY**, la consulta devolverá un conjunto arbitrario de registros. El predicado TOP no elige entre valores iguales.

```
SELECT TOP 5 Nombre FROM Empleados ORDER BY Nombre;
```

Ejemplos:

```
/* Sólo los 10 primeros clientes*/
SELECT TOP 10 *
FROM cliente
/* El 10% de los clientes*/
SELECT TOP 10 PERCENT *
FROM cliente
/* Los 3 productos más baratos*/
SELECT TOP 3 nombre, precio
FROM producto
ORDER BY precio
```

Si existen más productos con el mismo precio que el último valor de la lista, éstos no saldrán en el resultado de la consulta. Para que salgan añadiremos la cláusula **WITH TIES**.

La cláusula **WITH TIES** **sólo se puede emplear** si **SELECT incluye un ORDER BY,** de lo contrario dará error. Así quedaría:

```
/* Los 3 productos más baratos*/
SELECT TOP 3 WITH TIES nombre, precio FROM producto
ORDER BY precio
```

1.7 Las palabras clave $IDENTITY y $ROWGUID

La palabra clave **$IDENTITY** se interpreta como **la columna de la tabla que tiene la propiedad Identity** (columna identidad). Por ejemplo, si en la tabla Producto la columna codigo se definió como columna identidad

Ejemplo:

```
SELECT $IDENTITY, nombre, descripcion
FROM Producto
```

sería equivalente a:

```
SELECT codigo, nombre, descripcion
FROM Producto
```

La palabra clave **$ROWGUID** se interpreta como la columna de la tabla que tiene la propiedad ROWGUIDCOL y se usa del mismo modo que **$IDENTITY**. Para que una sea un identificador global deber ser del tipo de datos **uniqueidentifier**

```
SELECT  codigoProyecto, NombreProyecto,$ROWGUID
FROM PROYECTOS
```

```
codigoProyecto NombreProyecto      GuidProyecto
-------------- ------------------------------------------------------------ ------------------------------------
5              A                   3AD5B0B1-D035-47FE-B771-A08F9265C5E0
6              B                    BB691A76-8FEC-4854-89E0-639D8BDB4C93
10             C                   +x22E8401E-9A06-48EE-A72E-819976704846
(3 filas afectadas)
```

1.8 Cláusula INTO

Crea una nueva tabla e inserta en ella las filas resultantes de la consulta.

Sintaxis

```
SELECT ……..[ INTO nuevaTabla ]  FROM ……..
```

- **nuevaTabla**: Especifica el **nombre de una nueva tabla** que se va a crear, basada en las columnas de la lista de selección y las filas seleccionadas por la cláusula WHERE.
  - ✓ Las columnas de nuevaTabla **se crean en el orden que especifica la lista de selección**.
  - ✓ Cada columna de nuevaTabla tiene el **mismo nombre, tipo de datos y valor que la expresión correspondiente de la lista de selección**,
  - ✓ **no se transfiere ninguna otra propiedad** del campo o de la tabla ni ninguna restricción (ej: clave primaria, claves foráneas, valores por defecto etc.)
  - ✓ Cuando se incluye una columna calculada en la lista de selección, la columna correspondiente de la nueva tabla no es una columna calculada.

```
/* Crea una tabla llamada CopiaEmpleado con todos los datos de la tabla Empleado.*/
SELECT * INTO CopiaEmpleado  FROM Empleado
/*  Crea una tabla llamada EmpProy2 con los campos NSS, nombre,apellidos de la tabla
Empledos */
SELECT NSS, Nombre, Apel1, Apel2 INTO EmpProy2  FROM Empleados
```

1.9 Cláusula WHERE

Especifica una condición de búsqueda para restringir las filas que se van a devolver.

Sintaxis

```
WHERE <condiciónBúsqueda>
```

<condiciónBúsqueda> Permite **filtrar los registros con el fin de recuperar solamente aquellos que cumplan unas condiciones preestablecidas**.

```
/*Listar los empleados con sueldo superior a 1000 €*/
SELECT Nombre, Apel1, Apel2, Salario  FROM Empleados
WHERE Salario > 1000
```

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**

OPERADORES DE COMPARACIÓN

Los operadores de comparación **comparan dos expresiones**. Se pueden utilizar en todas las expresiones excepto en las de los tipos de datos text, ntext o image

Se pueden utilizar tanto para comparar números como para comparar textos y fechas. En el caso de los textos, las comparaciones se hacen en orden alfabético. Sólo que es un orden alfabético estricto. Es decir el orden de los caracteres en la tabla de códigos.

Operador	Significado
<	Mayor que
>	Menor que
>=	Mayor o igual que
<=	Menor o igual que
=	Igual
<>	Distinto
!=	Distinto
!<	No es menor que
!>	No es mayor que

OPERADORES LÓGICOS

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

Operador	Significado
AND	Devuelve verdadero si las expresiones a su izquierda y derecha son ambas verdaderas
OR	Devuelve verdadero si cualquiera de las dos expresiones a izquierda y derecha del OR, son verdaderas
NOT	Invierte la lógica de la expresión que está a su derecha. Si era verdadera, mediante NOT pasa a ser falso.

Ejemplos:

```
/* Viviendas con superficie entre 60 y 80 metros cuadrados */
SELECT codigo, calle, numero, superficie FROM vivienda
WHERE superficie>=60 AND superficie<=80;
/* Viviendas de 1 dormitorio o con más de 3 dormitorios*/
SELECT codigo, calle, numero, dormitorios FROM vivienda
WHERE dormitorios=1 OR dormitorios>3;
```

```
/* Viviendas que tienen un nombre de calle que empieza por letras entre la P y la Z */
SELECT codigo, calle, numero FROM vivienda
WHERE calle>='P' AND calle<'Z';
/* Viviendas de 1 dormitorio con más de 50m2 ó con más de 3 dormitorios que tienen
más de 120 m2 */
SELECT codigo, calle, numero, dormitorios, superficie FROM vivienda
WHERE (dormitorios=1 AND superficie>50) OR (dormitorios>3 AND superficie>120);
```

BETWEEN

El operador **BETWEEN** nos permite **obtener datos que se encuentren en un rango**.

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

Es equivalente a

```
campo>=limite_inferior AND campo <=limite_superior
```

Por ejemplo:

```
/* Viviendas con más de 1 y menos de 6 dormitorios*/
SELECT codigo, calle, numero, dormitorios FROM vivienda
WHERE dormitorios BETWEEN 2 AND 5;
/* Viviendas con menos de 50 m2 o más de 80 m2*/
SELECT codigo, calle, numero, superficie FROM vivienda
WHERE superficie NOT BETWEEN 50 AND 80;
```

IN

El operador **IN** nos permite **obtener datos que coinciden con una lista de valores**.

Sintaxis:

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

Es equivalente a

```
(campo=valor1 OR campo =valor2 OR campo=valor3 ……)
```

Por ejemplo:

```
/* Viviendas con 1 ,2 o 6 dormitorios*/
SELECT codigo, calle, numero, dormitorios FROM vivienda
WHERE dormitorios IN (1,2,6);
/* Viviendas que no tengan 1 ,2 o 6 dormitorios*/
SELECT codigo, calle, numero, dormitorios FROM vivienda
WHERE dormitorios NOT IN (1,2,6);
```

LIKE

El operador **LIKE** se usa sobre todo con textos. Permite obtener registros **cuyo valor cumple un patrón**.

LIKE devuelve los valores de los campos que coincidan con el patrón especificado.

Sintaxis:

```
Expresión [NOT] LIKE patrón [ ESCAPE carácter de escape ]
```

Este patrón puede contener los siguientes comodines válidos:

Comodín	Significado
%	Cualquier cadena de cero o más caracteres
_ (carácter subrayado)	Cualquier carácter
[ ]	Cualquier carácter individual del intervalo ([a-f]) o del conjunto ([abcdef]) que se haya especificado
[^]	Cualquier carácter individual que no se encuentra en el intervalo ([^a-f]) o del conjunto ([^abcdef]) que se haya especificado

Ejemplos:

```
/* Clientes cuyo nombre comience por A */
SELECT nombre, apellido1, apellido2 FROM cliente
WHERE nombre LIKE 'A%';
/* Clientes cuyo nombre tenga 4 letras y la primera y la tercera sean una L como (Lola, Lalo, Luli, ...) */
SELECT nombre, apellido1, apellido2 FROM cliente
WHERE nombre LIKE 'L_l_';
/* Clientes cuyo código comience por A o B, vaya seguido de un guión y acabe en 4*/
SELECT código, nombre, apellido1, apellido2 FROM cliente
WHERE codigo LIKE '[AB]-%4';
/* Clientes cuyo código comience por una letra entre la A y la G o que empiece por P o por T*/
SELECT código, nombre, apellido1, apellido2 FROM cliente
WHERE codigo LIKE '[A-GPT]%';
/* Clientes cuyo nombre no comience por P ni por T */
SELECT nombre, apellido1, apellido2 FROM cliente
WHERE nombre NOT LIKE '[PT]%';
```

ESCAPE: especificando esta cláusula se pueden buscar cadenas de caracteres que **incluyan uno o más caracteres comodines especiales** (**%,_,[**).

Por ejemplo, la tabla PRECIO puede almacenar valores de descuento que incluyan un signo de porcentaje (**%)**.

Para buscar el **signo de porcentaje como carácter en lugar de como carácter comodín**, debe suministrarse la palabra clave **ESCAPE** y el **carácter de escape**.

```
/* Buscar los precios con un valor de descuento del 30% */
SELECT articulo, precio FROM Articulo
WHERE descuento LIKE '%30e%%' ESCAPE 'e'
/* Buscar los libros con el carácter _ en el título */
SELECT titulo FROM libro
WHERE descuento LIKE  '%s_%' ESCAPE 's'
```

VALORES NULOS

Devuelve **verdadero** si el valor que examina **es nulo**.:

Sintaxis:

```
Columna  IS [NOT]  NULL
```

Por ejemplo

```
/* Clientes que no tienen email*/
SELECT nombre, apellido1, apellido2 FROM cliente
WHERE email IS NULL;
```

```
/* Clientes que tienen email*/
SELECT nombre, apellido1, apellido2 FROM cliente
WHERE email IS NOT NULL;
```

PRECEDENCIA DE OPERADORES

A veces las expresiones que se producen en los SELECT son muy extensas y es difícil saber que parte de la expresión se evalúa primero, por ello se indica la siguiente tabla de precedencia:

Orden	Operador
1	~ (operador bit a bit NOT)
2	* (multiplicar), / (dividir) , % (módulo)
3	+ (positivo), - (negativo), + (sumar), (+ concatenar), - (restar), & (AND bit a bit) , ^ (OR exclusiva bit a bit), | (OR bit a bit)
4	=, >, <, >=, <=, <>, !=, !>, !< (operadores de comparación)
5	NOT
6	AND
7	ALL, ANY, BETWEEN, IN, LIKE, OR, SOME
8	= (asignación)

Cuando en una expresión dos operadores tengan el mismo nivel de prioridad de operador, se evalúan de izquierda a derecha en función de su prioridad.

1.10 Cláusula ORDER BY

Especifica el **orden del conjunto de resultados**. Las definiciones de subconsultas y vistas no pueden contener una cláusula ORDER BY.

Sintaxis

```
[ORDER BY {expresión    [ ASC | DESC ] }   [,...n] ]
```

Expresión: Especifica **la columna según la que se ordenará**. Se puede especificar:

- ✓ Una **columna** de orden como un nombre
- ✓ **Alias de columna** (que puede estar calificado con el nombre de una tabla o vista)
- ✓ Una **expresión**
- ✓ un **entero no negativo** que representa la posición del nombre (ORDER BY 3),

Se puede **especificar varias columnas de orden**. La secuencia de columnas de orden en la cláusula ORDER BY define la estructura del conjunto de resultados ordenado.

- **ASC**: Indica que los valores de la columna especificada se deben ordenar de manera ascendente, desde el valor más bajo al más alto. **Es el valor por defecto.**
- **DESC**: Indica que los valores de la columna especificada se deben ordenar de manera descendente, desde el valor más alto al valor más bajo.

Los **valores NULL se tratan como los valores más bajos posibles**.

Ejemplo:

```
/* Clientes ordenados por apellidos y nombre descendentemente*/
SELECT nombre, apellido1, apellido2 FROM cliente
ORDER BY apellido1, apellido2, nombre DESC
/*También se puede especificar el número de columna*/
SELECT nombre, apellido1, apellido2 FROM cliente
ORDER BY 2,3,1 DESC
/*También se puede especificar un alias */
SELECT nombre, precio, precio_rebajado=precio-precio*0.40  FROM producto;
ORDER BY precio_rebajado
```

1.11 Funciones

Todos los SGBD implementan funciones para facilitar la creación de consultas complejas. Estas funciones dependen del SGBD que utilicemos. Aquí se comentan algunas de las que se utilizan en **SQL SERVER**.

Todas las funciones devuelven un resultado que procede de un determinado cálculo. La mayoría de las funciones precisan que se les envíe datos de entrada (parámetros o argumentos) que son necesarios para realizar el cálculo de la función. Este resultado, lógicamente depende de los parámetros enviados. Dichos parámetros se pasan entre paréntesis. De tal manera que la forma de invocar a una función es:

nombre_función [(***parámetro1***[,***parámetro2***,...])]

Si una función no precisa parámetros (como CURRENT_USER) no hace falta colocar los paréntesis.

FUNCIONES NUMÉRICAS

Función	Descripción
ABS(expresión_numérica)	Devuelve el valor absoluto de una expresión numérica
CEILING(expresión)	Devuelve el entero más pequeño igual o mayor al de la expresión especificada.
FLOOR(expresión)	Devuelve el entero más alto igual o menor que el valor especificado en la expresión
ROUND(expresión,longitud)	Redondea el número al número de decimales indicado más cercano. `round(28.5392, 2) devuelve 28.54` Un valor entero positivo determina el número de dígitos significativos a la derecha del punto decimal; un entero negativo, el número de dígitos significativos a la izquierda del punto decimal. Así, `round(2109.576,-2) devuelve 2100.000` ROUND siempre devuelve un valor. Si *longitud* es un valor negativo y mayor que el número de dígitos anteriores al separador decimal, ROUND devuelve 0.
POWER(expresión, potencia)	Eleva el valor de la expresión especificada a la potencia especificada.
RAND(valor de inicialización)	Devuelve un valor aleatorio entre 0 y 1.

FUNCIONES DE FECHA

Las fechas se utilizan muchísimo en todas las bases de datos. Hay que tener en cuenta que a los valores de tipo fecha se les pueden sumar o restar números y se entendería que esta suma (o resta) es de días. Algunas funciones de fecha trabajan con una parte de la fecha para operar en días, semanas, meses,.. Los posibles valores que puede tomar este parámetro **parteFecha** se muestran en la siguiente tabla:

Unidad	Abreviaturas	Significado
year	yy, yyyy	año
quarter	qq, q	cuatrimestre
month	mm, m	mes
dayofyear	dy, y	nº de día del año
day	dd, d	día
week	wk, ww	semana
weekday	dw, w	día de la semana
hour	hh	hora
minute	mi, n	minuto
second	ss, s	segundo
millisecond	ms	milisegundo

Weekday (dw) devuelve un número que indica el *día de la semana*, por ejemplo: Lunes = 1, Sábado = 6. El número generado con la unidad weekday depende del valor establecido por **SET DATEFIRST.**

En la siguiente tabla podemos ver las funciones de fecha más utilizadas:

Función	Descripción
CURRENT_TIMESTAMP	Fecha y hora actuales
GETDATE()	Fecha y hora actuales

DATEPART(parteFecha, fecha)	Devuelve un entero que representa la parte de fecha especificada.
	`datepart(dw,'29/02/2012') devuelve 3`
	`datepart(dd,'29/02/2012') devuelve 29`
	`datepart(dy,'29/02/2012') devuelve 60`

DAY(fecha)	Devuelve un entero que representa la parte del día de la fecha especificada. Es equivalente a **DATEPART**(dd,fecha)
MONTH(fecha)	Devuelve un entero que representa la parte del mes de la fecha especificada. Es equivalente a **DATEPART** (mm,fecha)
YEAR(fecha)	Devuelve un entero que representa la parte del año de la fecha especificada. Es equivalente a **DATEPART** (yy,fecha)

DATEADD(parteFecha, número, fecha)	Añade un intervalo a una fecha y devuelve una nueva fecha. Cuando parteFecha es *month*, el número de días del mes afecta al resultado. Así:
	`dateadd(month,1,'30/03/2012')`
	`dateadd(month,1,'31/03/2012')`
	`devuelve en ambos casos 2012-04-30 00:00:00.000.`
	Marzo tiene 31 días y abril 30 días. Al agregar un mes al final de marzo, se devuelve el último día de abril.

DATEDIFF(parteFecha, fecha_inicio, echa_fin)	Devuelve la diferencia entre dos fechas especificadas en la unidad de tiempo especificada en parteFecha (p. e. horas, días, semanas)

DATENAME(parteFecha, fecha)	Devuelve una cadena que representa el nombre de la parte de fecha especificada. Cuando la unidad corresponde a mes o día de la semana, devuelve el valor en letras. Por ejemplo:
	`datename(month,'30/03/2012') devuelve Marzo`
	`datename(yy,'30/03/2012') devuelve 2012`
	`datename(dw,'30/03/2012') devuelve Viernes`

@@DATEFIRST	Devuelve el primer día de la semana establecido con SET DATEFIRST
SET DATEFIRST número	Establece el primer día de la semana en un número del 1 al 7 `SET DATEFIRST 7 establece como primer día de la semana el Domingo`

FUNCIONES DE CADENA

Funciones de conversión de mayúsculas a minúsculas

Función	Descripción
LOWER(cadena)	Convierte una cadena de tipo varchar a minúsculas
UPPER(Cadena)	Convierte una cadena de tipo varchar a mayúsculas

Funciones de transformación

Función	Descripción
LEN(cadena)	Devuelve la longitud de la cadena
REPLACE(cadena_original, cadena_a_quitar, cadena_a_poner)	Reemplaza todas las apariciones de un valor de cadena especificado por otro valor de cadena. `replace ('mi mami me mima,'mi','lo') devuelve lo malo me loma`
LTRIM(cadena)	Elimina cualquier espacio en blanco previo de una expresión de caracteres
RTRIM(cadena)	Elimina los espacios finales de una expresión de caracteres
LEFT(cadena,lon)	Devuelve la parte izquierda de una cadena de caracteres con el número de caracteres especificado por lon `left ('mi mami me mima',5) devuelve mi ma`
RIGHT(cadena,lon)	Devuelve la parte derecha de una cadena de caracteres con el número

	de caracteres especificado por lon `right ('mi mami me mima',6) devuelve e mima`
SUBSTRING(cadena_origen, n, lon)	Extrae de la cadena origen una subcadena, comenzando en el carácter n-ésimo, con una longitud lon `substring ('mi mami me mima',4,3) devuelve mam`
CHARINDEX(cadena_a_buscar, cadena_origen [, inicio])	Devuelve la posición inicial de la expresión especificada (cadena_a_buscar) en una cadena de caracteres. inicio indica la posición a partir de la cual se empieza a buscar. Si no se especifica o si es cero o negativo, la búsqueda empieza en el primer carácter de cadena_origen `charindex ('ma', 'mi mami me mima') devuelve 4`
REPLICATE(cadena, entero)	Repite una expresión de caracteres un número especificado de veces. `replicate ('mami',3) devuelve mamimamimami`
REVERSE(cadena)	Invierte el texto especificado en cadena (le da la vuelta). `reverse ('mami') devuelve imam`
SPACE(entero)	Devuelve una cadena de espacios repetidos, cuya longitud la indica un entero.

Otras funciones de caracteres

Función	Descripción
ASCII(cadena)	Devuelve el valor de código ASCII del carácter situado más a la izquierda de una expresión de caracteres `ascii('ala') devuelve 97, lo mismo que ascii('a')`
CHAR(numero)	Devuelve el carácter según el código ASCII correspondiente a un número entero. Se suele utilizar para insertar caracteres de control en cadenas alfanuméricas. (tabulación → char(9) ; return → char(13); salto de línea → char(10)

OTRAS FUNCIONES

Funciones de conversión

Función	Descripción

CAST(expresión as tipo_dato)	Devuelve la expresión convertida al tipo de dato especificado (por ejemplo, una fecha convertida a cadena de caracteres)
	`cast(dato AS int)`
	`cast(nombre AS char(30))`

CONVERT(tipo_dato, expresión [, estilo])	Cadena convertida según especifique el tipo de dato. *estilo* es el parámetro opcional que especifica el formato. Por ejemplo, si queremos convertir un varchar a datetime deberíamos indicar el formato de la fecha. `convert (varchar(20), fecha, 3) donde 3 es el formato dd/mm/aa`

Funciones de trabajo con nulos

Función	Descripción
ISNULL(expresion, valor_si_nulo)	Si expresion es nulo mostrará el valor. `isnull(precio, 0) Si precio tiene un valor no nulo la función devuelve ese valor, si precio contiene NULL entonces la función devuelve 0.`
COALESCE(expr, [...n])	Devuelve la primera expresión distinta de NULL en un conjunto de argumentos. `coalesce(fijo, movil) Devuelve el teléfono fijo del cliente si lo tiene, en caso contrario devuelve el móvil.`

FUNCIÓN CASE

Evalúa una lista de condiciones y devuelve una de las varias expresiones de resultado posibles. Existen dos variantes:

- La función **CASE sencilla** compara una expresión con un conjunto de expresiones sencillas para determinar el resultado.
- La función **CASE buscada** evalúa un conjunto de expresiones booleanas para determinar el resultado.

Sintaxis de la función CASE sencilla:

```
CASE expresion_entrada
        WHEN valor THEN resultado
        [, ...n]
        [ ELSE  resultado_sino ]
END
```

Ejemplo:

```
CASE estado_civil
      WHEN 'C' THEN 'casado'
      WHEN 'S' THEN 'soltero'
      WHEN 'V' THEN 'viudo'
      ELSE 'otro'
END
```

Si queremos evaluar si el campo es nulo no podemos añadir un WHEN NULL THEN 'desconocido' porque la condición estado_civil = NULL no es TRUE cuando el campo es nulo, en este caso habría que utilizar la segunda forma de la función CASE y utilizar el operador is null

Sintaxis de la función CASE buscada:

```
CASE
        WHEN condicion THEN resultado
        [, ...n]
        [ ELSE  resultado_sino ]
END
```

Ejemplo:

```
CASE
      WHEN estado_civill IS NULL THEN 'desconocido'
      WHEN estado_civill='C' THEN 'casado'
      WHEN estado_civil='S' THEN 'soltero'
      WHEN estado_civil= 'V' THEN 'viudo'
      ELSE 'otro'
END
```

2. Obtener datos de multitablas.

2.1 Introducción a las combinaciones

Habitualmente cuando necesitamos recuperar la información de una base de datos nos encontramos con que dicha **información se encuentra repartida en varias tablas**.

SQL nos proporciona una forma fácil de mostrar la información repartida en varias tablas: las **consultas combinadas(multitablas) o JOINS**.

Las tablas **se combinan para producir un conjunto único de resultados que incorpora filas y columnas de dos o más tablas**.

- ✓ Cuando combina tablas, SQL Server **utiliza los valores de la(s) columna(s) combinadas** de una tabla para seleccionar filas de otra tabla.
- ✓ Las filas con valores de columna coincidentes se combinan en nuevas filas, que forman el grupo de resultados.

Por ejemplo, si se necesita recuperar una lista de los proyectos de nuestra base de datos que incluya nombre y número del departamento al que está asignado cada proyecto, se tendrá que combinará la tabla Proyectos y la tabla Departamentos en el número de departamento. El nombre del departamento y las otras columnas de la tabla Departamentos podrán entonces ser utilizadas en la lista de selección de una consulta.

Una combinación le permite consultar dos o más tablas para producir un conjunto único de resultados. Cuando se implementan combinaciones, debemos tener en cuenta:

- La mayoría de las condiciones de combinaciones se basan en la **clave principal de una tabla** y en la **clave externa de otra tabla.** Se utiliza columnas comunes de las tablas especificadas para combinar las tablas. Estas columnas deberían **tener los mismos tipos de datos o parecidos**, pero no deben tener los mismos nombres.
- Se **debe calificar** cada nombre de columna utilizando el formato **nombre_tabla.nombre_columna** si los **nombres de las columnas son iguales.** Ejemplo:

```
SELECT NSS,NOMBRE, APEL1 APEL2, SALARIO, DEPARTAMENTOS.NUMDEP, NOMBREDEP
FROM EMPLEADOS, DEPARTAMENTOS
WHERE EMPLEADOS.NUMDEP=DEPARTAMENTOS.NUMDEP AND SEXO='H' AND SALARIO>1500
```

2.2 Estructura de una composición

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,....,An) y de R2(B1,B2,..,Bm)** es una relación **R3 con n+m atributos R3(A1,A2,..,An,B1,B2,...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. Ejemplo:

```
SELECT * FROM EMPLEADOS, DEPARTAMENTOS
```

NSS	Nombre	Apel1	Apel2	Resto campos	NumDep	NombreDep	Resto campos
10000111	Luisa	SamPedro	Silva	.....	1	Finanzas	.....
11111111	Juan	Fernández	García	.....	1	Finanzas	.....
22111111	Manuel	Lopez	Ramos	.....	1	Finanzas	.....
22222222	Ana	Rodríguez	SanTomé	.....	1	Finanzas	.....
.....	.....	.....	.....	.....	.....	.....	.....
.....	.....	.....	.....	.....	.....	.....	.....
10000111	Luisa	SamPedro	Silva	.....	2	Marketing	.....
11111111	Juan	Fernández	García	.....	2	Marketing	.....
22111111	Manuel	Lopez	Ramos	.....	2	Marketing	.....
.....	.....	.....	.....	.....	.....	.....	.....
.....	.....	.....	.....	.....	.....	.....	.....
10000111	Luisa	SamPedro	Silva	.....	10	Almacenaje	.....
11111111	Juan	Fernández	García	.....	10	Almacenaje	.....
22111111	Manuel	Lopez	Ramos	.....	10	Almacenaje	.....
22222222	Ana	Rodríguez	SanTomé	.....	10	Almacenaje	.....
33111111	Lucia	Gimenez	Galán	.....	10	Almacenaje	.....
.....	.....	.....	.....	.....	.....	.....	.....
.....	.....	.....	.....	.....	.....	.....	.....

2.3 Combinaciones internas con emparejamiento en la cláusula where

Las **combinaciones internas** unen tablas comparando **los valores en las columnas que son comunes a ambas tablas.** SQL Server devuelve sólo las filas que coinciden con las condiciones de combinación.

Una de las maneras más fáciles de consulta multitabla es:

- Especificar la **lista de tablas** en la cláusula **FROM**
- 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.NUMDEP=DEPARTAMENTO.NUMDEP
```

NSS	Nombre	NombreDep
10000111	Luisa	Nóminas
11111111	Juan	Finanzas
22111111	Manuel	Procesos
22222222	Ana	Marketing
33111111	Lucia	Personal
.....		

Nótese que se utiliza la notación **tabla.columna** para **evitar la ambigüedad,** ya que podríamos tener el mismo nombre de campo en ambas tablas. Para evitar repetir continuamente el nombre de la tabla, se puede utilizar un **alias de tabla:**

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

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 los hombres con salario mayores que 1500

```
SELECT NSS,NOMBRE, APEL1 APEL2, SALARIO, DEPARTAMENTOS.NUMDEP, NOMBREDEP
FROM EMPLEADOS, DEPARTAMENTOS
WHERE EMPLEADOS.NUMDEP=DEPARTAMENTOS.NUMDEP AND SEXO='H' AND SALARIO>1500
```

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 EMPLEADOS,PARTICIPACION,PROYECTOS,DEPARTAMENTOS
WHERE EMPLEADOS.NSS=PARTICIPACION.NSS AND
        PARTICIPACION.NUMPROY=PROYECTOS.NUMPROY AND
         PROYECTOS.NUMDEP=DEPARTAMENTOS.NUMDEP
ORDER BY EMPLEADOS.NSS
```

Juan	Fernández	García	A	Finanzas
Juan	Fernández	García	B	Finanzas
Ana	Rodríguez	SanTomé	A	Finanzas
Ana	Rodríguez	SanTomé	B	Finanzas
Ana	Rodríguez	SanTomé	C	Finanzas
Ana	Rodríguez	SanTomé	D	Marketing

EJ: listar los nombres de los proyectos con el nombre del departamento que está asignado cada proyecto del empleado con número de seguridad social 11111111.

```
SELECT NOMBREPROY,NOMBREDEP
FROM PARTICIPACION,PROYECTOS,DEPARTAMENTOS
WHERE PARTICIPACION.NUMPROY=PROYECTOS.NUMPROY AND
PROYECTOS.NUMDEP=DEPARTAMENTOS.NUMDEP AND PARTICIPACION.NSS=11111111
ORDER BY PARTICIPACION.NSS
```

2.4 Combinar Tablas: Sintaxis SQL 1999

En la versión SQL de 1999 se ideó una nueva sintaxis para consultar varias tablas.

La razón fue separar **las condiciones de asociación respecto de las condiciones de selección de registros**.

La sintaxis completa para la combinación de tablas en Transact SQL es la siguiente:

```
SELECT {nombre_columna} [,...n]
FROM {nombre_tabla }
[ [ INNER | { {LEFT | RIGHT | FULL } [OUTER] } ]  JOIN
nombre_tabla  ON condiciones de combinación  [, ...n]
[WHERE <condicion> [{ AND|OR <condicion>}]]
```

Una combinación le permite seleccionar columnas de tablas múltiples expandiendo en la cláusula FROM de la instrucción SELECT. Se agregan dos palabras clave adicionales a la cláusula FROM: **JOIN y ON**:

- La palabra clave **JOIN y sus opciones especifican qué tablas** deben ser combinadas y cómo combinarlas.
- La palabra clave **ON especifica las columnas que las tablas tienen en común**.
- Si una tabla tiene una **clave principal compuesta**, se deberá hacer referencia a **toda la clave** en la cláusula ON cuando se combinan las tablas.

2.5 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.

Formato:

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

Se agregan **tres 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.

Las **combinaciones internas son las predeterminadas de SQL Server**. Puede abreviar la cláusula INNER JOIN a JOIN.

Puede utilizar **múltiples cláusulas JOIN y combinar más de dos tablas en la misma consulta**.

```
/*Listar los nombres, apellidos y nombre del departamento donde trabaja cada empleado*/
SELECT NSS, Nombre, NombreDep
FROM Empleados INNER JOIN Departamentos  ON Empleados.NumDep = Departamentos.NumDep
/*Listar los campos nss, nombre, apellidos, salario, departamento y nombre del departamento de las
mujeres con salario mayores que 1000 (es decir con criterios de selección de fila) */
SELECT NSS, Nombre, Apel1, Apel2, Salario, NumDep, NombreDep
FROM Empleados E INNER JOIN Departamentos D  ON E.NumDep=D.NumDep
WHERE Sexo='F' AND  Salario>1000
```

SQL **no exige que las columnas de emparejamiento tengan relación clave primaria / clave foránea**. De hecho, dos columnas cualesquiera de dos tablas pueden utilizarse como columnas de emparejamiento siempre y cuando sus **tipos de datos sean comparables**.

```
SELECT Nombre, Apel1, Apel2, Salario
FROM Empleados INNER JOIN Departamentos ON FechaIngreso = FechaInicGerente
```

Hasta aquí sólo hemos utilizado JOINS en los que en la condición de búsqueda sólo aparece **la igualdad**, sin embargo, en la condición de **búsqueda del JOIN puede especificarse un operador lógico cualquiera (=, <>, <,>, etc)** para usarlo en los valores de comparación de las columnas. El JOIN forma parejas de filas haciendo cumplirse las condiciones del JOIN en los contenidos de las columnas relacionadas.

```
SELECT Nombre, Apel1, Apel2, Salario
FROM Empleados INNER JOIN Departamentos ON FechaIngreso > FechaInicGerente
```

2.6 Combinaciones externas: LEFT, RIGHT y FULL OUTER JOIN

La **composición externa** se escribe de manera similar al INNER JOIN indicando una condición de combinación.

En el resultado se añaden filas que no cumplen la condición de combinación.

- Al igual que las combinaciones 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 combinaciones externas **también devuelven filas no coincidentes de una de las tablas**.
- Las columnas de la tabla de origen de las filas no coincidentes contienen datos, pero las **columnas de la otra tabla** contienen **valores nulos**.

Las palabras **LEFT, RIGHT y FULL** indican **la tabla de la cual se van a añadir las filas sin correspondencia**

2.6.1 Composición externa izquierda: LEFT JOIN

Las combinaciones externas izquierda **LEFT JOIN** incluyen **todos los registros de la primera** (parte izquierda) de dos tablas, **aunque no haya valores coincidentes para los registros de la segunda tabla** (parte derecha) además de los valores que coinciden.

EJ: Muestra el nss del EMPLEADO con el numero proyecto y en el caso de que no tenga ningún proyecto asignado (tabla a la IZQUIERDA del JOIN) mostraría también el nss del empleado, poniendo a NULL los campos correspondientes de la tabla PARTICIPACION.

```
SELECT Empleados.NSS, NumProy FROM Empleados LEFT JOIN Participación
ON Empleados.NSS=Participación.NSS
```

NSS	NumProy
10000111	**NULL**
11111111	1
11111111	2
22111111	**NULL**
22222222	1
22222222	2
22222222	5

2.6.2 Composición externa derecha: RIGHT JOIN

Las combinaciones externas derecha **RIGHT JOIN** **incluyen todos los registros de la segunda** (parte derecha) de dos tablas, **aunque no haya valores coincidentes para los registros de la primera tabla** (parte izquierda) además de los registros coincidentes.

EJ: Muestra cada EMPLEADO con los datos del DEPARTAMENTO al que pertenece y en el caso de que haya algún departamento (tabla a la DERECHA del JOIN) que no tenga asignado ningún empleado lo mostraría también, poniendo a NULL los campos correspondientes de la tabla empleado.

```
SELECT E.Nombre, E.Apel1, E.apel2, D.NumDep, D.NombreDep
FROM Empleado E RIGHT JOIN Departamento D ON D.NumDep=E.NumDep
```

2.6.3 Composición externa completa:

Las combinaciones externas derecha **FULL JOIN** incluyen **todos los registros de la primera y segunda tablas**, aunque **no haya valores coincidentes para los registros de la primera tabla (parte izquierda) además de los registros coincidentes**

Ej: Además de mostrar cada empleado con los datos del departamento al que pertenece, mostrará tanto **los departamentos sin empleados como los empleados sin departamento.**

La utilización de una combinación cruzada presenta todas las combinaciones de filas posibles entre estas dos tablas. La tabla Empleados tiene cuarenta filas y la tabla Departamentos tiene cinco filas, el conjunto de resultados contiene, por tanto, 200 filas.

```
SELECT E.Nombre, E.Apel1, E.apel2, D.NumDep, D.NombreDep
FROM Empleado E FULL JOIN Departamento D ON D.NumDep=E.NumDep
```

2.6.4 Composición cruzada:

Las combinaciones cruzadas no utilizan columnas comunes para combinar filas, **devuelven una fila por cada combinación de filas de las dos tablas. Realiza un producto cartesiano**

```
SELECT Nombre, Apel1, Apel2, NombreDep
FROM Empleados CROSS JOIN Departamentos
--Es equivalente
SELECT Nombre, Apel1, Apel2, NombreDep
FROM Empleados, Departamentos
```

2.7 Consultas de no coincidentes

Un caso típico de uso de las **relaciones externas es el uso de consultas de no coincidentes**.

Estas consultas sustituyen de forma más eficiente al operador NOT IN (se explica más adelante) ya que permiten comprobar filas de una tabla que no están presentes en una segunda tabla que se relaciona con la primera a través de alguna clave secundaria.

Volviendo al ejemplo anterior supongamos que sólo queremos **mostrar los empleados que no están asignados a un departamento**.

```
SELECT E.Nombre, E.Apel1, E.apel2
FROM Empleado E LEFT JOIN Departamento D ON D.NumDep=E.NumDep
WHERE E.NumDep IS NULL;
```

En esta consulta, mediante el operador LEFT JOIN obteneos la lista completa de empleados (estén o no asignados a un departamento). Si sólo cogemos aquellas filas que no tienen ningún valor en el campo NumDep de la tabla empleado, mostrará los empleados que no pertenecen a ningún departamento.

2.8 Cómo combinar más de dos tablas.

Cuando se utilizan múltiples combinaciones es más fácil considerar cada combinación independientemente.

Por ejemplo, si **se combinan las tablas A, B y C:**

- La **primera combinación combina las tablas A y B** para producir un **conjunto de resultados, este conjunto de resultados se combina con la tabla C** en la segunda combinación para producir el **conjunto de resultados final.**
- Se pueden **especificar columnas de cualquiera de las tablas** en las otras cláusulas de la instrucción SELECT.

Una **operación LEFT JOIN o RIGHT JOIN se puede anidar dentro de una operación INNER JOIN**, pero una operación INNER JOIN no se puede anidar dentro de LEFT JOIN o RIGHT JOIN

Ej: Para cada empleado, listar NSS, nombre, número y nombre del proyecto asignado y num de horas de participación de cada empleado en cada proyecto.

```
SELECT Empleados.NSS, Nombre, Apel1, Apel2, NumHoras, Proyectos.NumProy, NombreProy
FROM (Empleados INNER JOIN Participación ON Empleados.NSS=Participación.NSS)
                INNER JOIN Proyectos ON Participación.NumProy=Proyectos.NumProy
```

En este último ejemplo para recuperar información de las dos tablas Empleados y Proyectos necesita utilizar una tabla intermedia y dos combinaciones. Aunque no necesita recuperar ninguna columna de la tabla Participacion, debe incluir esta tabla en la consulta para relacionar la tabla Empleados a la tabla Proyectos.

Es recomendable utilizar paréntesis para delimitar las composiciones, aunque no es obligatorio salvo que queramos cambiar el orden de ejecución de éstas. Por ejemplo:

```
SELECT DNI, e.nombre, e.apellido1, e.apellido2, d.codigo, d.nombre, p.*
FROM (departamento d LEFT JOIN Empleado e
        ON d.codigo=e.cod_departamento)
          INNER JOIN pedido p ON p.cod_empleado=e.codigo;
```

no es igual que:

```
SELECT DNI, e.nombre, e.apellido1, e.apellido2, d.codigo, d.nombre, p.*
FROM  departamento d LEFT JOIN (Empleado e INNER JOIN pedido p
                                ON p.cod_empleado=e.codigo)
                     ON d.codigo=e.cod_departamento;
```

ya que en la segunda consulta mostrará información de todos los departamentos, aunque no haya ningún empleado que haya realizado un pedido.

2.9 Cómo autocombinar una tabla

Si quiere encontrar filas que **tengan valores en común con otras filas en la misma tabla**, puede **combinar una tabla con otra instancia de sí misma**; esto se conoce como **autocombinación**

Se **necesitan alias de tablas para hacer referencia a dos copias de una tabla**.

Ej: Este ejemplo lista los nombres de los empleados y de sus jefes.

```
SELECT E.Nombre, E.Apel1, E.Apel2, J.Nombre, J.Apel1, J.Apel2
FROM Empleados E LEFT JOIN Empleados J
ON  E.NSSSup = J.NSS
```

3. Consultas de resumen

Una de las funcionalidades de la sentencia **SELECT** es el permitir obtener **resúmenes de los datos** contenidos en las columnas de las tablas.

Las funciones que calculan valores de resumen (como un promedio o una suma) se denominan **funciones agregadas**.

Cuando se ejecuta una función agregada, se resume los valores de una columna en concreto para una tabla entera o para un grupo de filas dentro de la tabla.

Se produce un único valor para la tabla entera o para cada grupo de filas.

- ✓ Las funciones agregadas con la instrucción **SELECT** para **resumir los valores de una tabla entera**.
- ✓ Se agrega una cláusula **GROUP BY** a la instrucción SELECT para **resumir valores por grupos de filas**.
- ✓ Y se añade una cláusula **HAVING** a la instrucción GROUP BY para **seleccionar aquellos grupos que cumplan una condición.**

Para poder llevarlo a cabo la sentencia **SELECT** consta de una serie de cláusulas específicas (**GROUP BY**, **HAVING**), y Transact-SQL tiene definidas unas funciones para poder realizar estos cálculos, las *funciones de agregado* (también llamadas funciones de columna). Así la sintaxis del SELECT queda:

```
SELECT lista_de_expresiones..
FROM lista_de_tablas
[JOIN] tablasRelacionadas Y condiciones_de_relacion]
[WHERE condiciones]
[GROUP BY grupos]
[HAVING condiciones_de_grupo]
[ORDER BY columnas]
```

En el apartado **GROUP BY**, se indican **las columnas por las que se agrupa**.
La función de este apartado es crear **un único registro** por **cada valor distinto en las columnas del grupo**.

Si por ejemplo, agrupamos en base a la *localidad* y el *código postal* en la tabla de *empleados*, nos devolverá un único registro por cada *localidad* y *código postal* distintos:

```
SELECT localidad, codigo_postal
FROM Empleado
GROUP BY localidad, codigo_postal;
```

De este modo si tenemos 3 empleados de Pontevedra cuyo código postal es 36002, la consulta sólo nos devolverá un registro con estos dos valores, y así para todas las localidades y códigos postales distintos de los empleados:

TABLA EMPLEADO

Nombre	Localidad	Codigo_postal
Elena	Vigo	36203
Manolo	Pontevedra	36002
Guillermo	Pontevedra	36001
Anxos	Lugo	27004
Mateo	Pontevedra	36002
Julia	Pontevedra	36001
Antonia	Vigo	36202
Juan Antonio	Vigo	36203
Sara	Pontevedra	36002

RESULTADO CONSULTA

Localidad	Codigo_postal
Vigo	**36203**
Pontevedra	**36002**
Pontevedra	**36001**
Lugo	**27004**
Vigo	**36202**

Si se utiliza **GROUP BY** pero no existe una función SQL agregada en la instrucción **SELECT** se obtiene el mismo resultado que con una consulta **SELECT DISTINCT**.

Los valores Null en los campos **GROUP BY** se agrupan y no se omiten. No obstante, los valores Null no se evalúan en ninguna de las funciones SQL agregadas.

IMPORTANTE: **Todos los campos de la lista de campos de SELECT deben incluirse** en la cláusula **GROUP BY** o como argumentos de una función SQL agregada.

3.1 Funciones de agregado

- Las funciones de agregado **realizan un cálculo sobre un conjunto de valores y devuelven un solo valor**.
- **Si exceptuamos la función COUNT**, todas las funciones de agregado **ignoran los valores NULL**.
- **Sólo se pueden utilizar** en:
  - ✓ La lista de selección de una **instrucción SELECT.**
  - ✓ Cláusulas **HAVING**.

Una función de agregado SQL acepta un grupo de datos (normalmente una columna de datos) como argumento, y produce un único dato que resume el grupo.
*Por ejemplo la función **AVG()** acepta una columna de datos numéricos y devuelve la media aritmética (average) de los valores contenidos en la columna.*

Todas las funciones de agregado tienen una estructura muy parecida:

FUNCION ([ALL | DISTINCT] *expresion*)

- ✓ La palabra **ALL** indica que se tiene que tomar en cuenta todos los valores de la columna. Es el valor por defecto.
- ✓ La palabra **DISTINCT** hace que se consideren todas las repeticiones del mismo valor como uno sólo (considera únicamente valores distintos).
- ✓ Todas las funciones de agregado se aplican a las filas del origen de datos una vez ejecutada la cláusula **WHERE** (si la hubiera).

Transact SQL pone a nuestra disposición múltiples funciones agregadas. Las más comunes son:

Función agregada	Descripción
AVG(expresión)	El promedio de todos los valores.
COUNT(expresión)	El número de valores no nulos. Devuelve un valor de tipo de datos int
COUNT(*)	El número de filas en la tabla o grupo incluyendo valores nulos y duplicados.
MAX(expresión)	El valor máximo de todos los valores.
MIN(expresión)	El valor mínimo de todos los valores.
SUM(expresión)	La suma de todos los valores.
STDEV(expresión)	La desviación estadística de todos los valores.
VAR(expresión)	La discrepancia estadística de todos los valores.

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

¡ERROR!

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

Sintaxis:

```
SELECT {{AVG | COUNT  | MAX | MIN | SUM| STDEV | VAR }
            (expresión | *) } [,..]
FROM lista_tabla
[ WHERE condiciones_búsqueda]
```

```
/* Indica cuantos departamentos existen*/
SELECT COUNT (*) FROM departamento;
/* Indica en cuantas localidades distintas viven los empleados*/
SELECT COUNT (DISTINCT localidad)  FROM empleado;
/* Indica cuanto pagamos a los empleados del departamento de PERSONAL*/
SELECT SUM (sueldo)
FROM empleado INNER JOIN departamento
 ON cod_departamento=departamento.codigo
WHERE departamento.nombre='PERSONAL';
/* El sueldo más alto y el más bajo de los empleados*/
SELECT MAX (sueldo), MIN (sueldo) FROM empleado
```

FUNCIONES DE AGREGADO CON VALORES NULOS

Hay que tener cuidado al utilizar funciones agregadas en columnas que contienen valores nulos, ya que el conjunto de resultados puede no ser representativo de sus datos. Hay que tener en cuenta:

- **Todas las funciones de agregado**, con la **excepción** de la función *COUNT(*):*
  - **Ignoran los valores nulos en las columnas.**
  - **Devuelven NULL si ninguna fila satisface la cláusula WHERE de la instrucción SELECT**.
- La función **COUNT(*) :**
  - **Cuenta todas las filas, incluso si todas las columnas contienen un valor nulo**.
  - **Devuelve un valor de cero si ninguna fila satisface la cláusula WHERE de la instrucción SELECT**..

Ej: Hay dos filas que en la columna nsssup contiene 2 valores NULL

```
SELECT count(nsssup) from empleados
```

12 filas afectadas

```
SELECT count(*) from empleados
```

14 filas afectadas

Para que un script funcione correctamente se debería utilizar **IS NULL** e **IS NOT NULL** en las comparaciones que puedan contener valores NULL.

Ejemplo: Se lista el número de empleados que tienen un valor nulo en la columna *Comision* en la tabla Empleados, indicando que no tienen Comisión.

```
SELECT count(*) FROM Empleados   WHERE Comision IS NULL
```

3.2 Cláusula GROUP BY

Se utiliza la cláusula **GROUP BY** en columnas o expresiones para **agrupar filas** y para **hacer operaciones de resumen en estos grupos**.

Sintaxis parcial:

```
SELECT seleccionar_lista
FROM origen_tabla
[WHERE condición_búsqueda]
[GROUP BY  agrupar_por_expresión [,...n]
[HAVING condición_búsqueda]
```

- 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**
- 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.

  *Cuando se incluye una cláusula **GROUP BY** en una sentencia **SELECT**, la **lista de selección** **debe estar formada por funciones de agregado o por los nombres de columnas** **especificadas en la cláusula GROUP BY**.*
- Si **incluye una cláusula WHERE**, se **agrupa sólo las filas que satisfagan las condiciones de la cláusula WHERE.**
- Si utiliza la cláusula **GROUP BY en columnas que contienen valores NULL, estas forman un grupo**.

  Para excluir las filas con valores nulos se utiliza el operador comparativo I**S [NOT] NULL** en la cláusula **WHERE** de la instrucción SELECT.
- En una cláusula **GROUP BY**, debe **especificar el nombre de una columna y no el alias de la columna**
- Si se utiliza **GROUP BY pero no existe una función de agregado** en la instrucción SELECT se obtiene el mismo resultado que con una consulta **SELECT DISTINCT**

```
/* El sueldo más alto y el más bajo de los empleados para cada departamento*/
SELECT cod_departamento, MAX (sueldo), MIN (sueldo) FROM empleado
GROUP BY cod_departamento
/*Lista cuanto hay que pagar a los empleados de cada departamento*/
SELECT NumDep, SUM(Salario) AS SalarioTotal FROM Empleados
GROUP BY NumDep
```

Cuando queremos realizar **una agrupación múltiple,** por varias columnas, éstas se indican en la cláusula **GROUP BY** en el orden de mayor a menor agrupación igual que con la cláusula **ORDER BY**.

```
/* El número de empleados para cada departamento y localidad*/
SELECT cod_departamento, localidad COUNT (*) FROM empleado
GROUP BY cod_departamento, localidad
```

3.3 La cláusula GROUP BY con la cláusula HAVING

La cláusula **HAVING** se utiliza en columnas o expresiones para **establecer condiciones en los grupos incluidos en un conjunto de resultados**.

- La cláusula **HAVING establece condiciones** en la cláusula **GROUP BY**
  (*parecido a como la cláusula WHERE establece condiciones para las filas devueltas por la instrución SELECT)* .

  ✓ ***La cláusula WHERE determina qué filas son agrupadas.***
  ✓ ***La cláusula HAVING determina qué grupos son devueltos.***

*La **condición de búsqueda de la cláusula WHERE se aplica a las filas individuales antes de que sean agrupadas,** mientras que **la condición de búsqueda de la cláusula HAVING se aplica a los grupos de filas**.*

- La cláusula **HAVING se utiliza sólo con la cláusula GROUP BY** para restringir las agrupaciones.
- En la cláusula **HAVING se puede hacer referencia a cualquiera de las columnas** que aparecen en el **SELECT**, incluyendo las funciones agregadas.

  No se puede hacer referencia a funciones agregadas en la cláusula WHERE.

  Una **importante característica de HAVING** es que es posible incluir funciones de agregado, cosa que no podemos hacer con WHERE.

```
/* Número de empleados por departamento, sólo para aquellos que tienen más de 5 empleados*/
SELECT cod_departamento, COUNT (*) FROM empleado
GROUP BY cod_departamento
HAVING COUNT(*) >5;
/* Para los departamentos que tengan un número de empleados varones mayor que 2, halla salario mínimo, salario máximo, promedio del salario, número de empleados d
SELECT NumDep, MIN(Salario), MAX(Salario), AVG(Salario), COUNT(*)
FROM Empleados
WHERE Sexo="M"
GROUP BY NumDep
HAVING COUNT(*)>2
```

La siguiente **instrucción sería incorrecta:**

```
SELECT NumDep, MIN(Salario), MAX(Salario), AVG(Salario), COUNT(*) FROM Empleados
GROUP BY NumDep
HAVING COUNT(*)>2 AND Sexo = "M"
```

ORDEN DE EJECUCIÓN DE LAS CONSULTAS

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

1.- Se **selecciona la tabla especificada** en **FROM**.

2.- Se **eliminan** de la tabla resultante **las filas que no satisfagan la condición** de la cláusula **WHERE**.

3.- Se **forman grupos** según las columnas de agrupamiento en la cláusula **GROUP BY**.

4.- Se **descartan los grupos** que **no satisfagan la condición** especificada en **HAVING**.

5.- 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**.

6.- Se presenta el resultado **ordenado** según la cláusula **ORDER BY**.

4. Subconsultas

Una **subconsulta** es **una instrucción SELECT anidada dentro de una instrucción SELECT, INSERT, UPDATE o DELETE** o dentro de **otra subconsulta**.

- La cláusula SELECT de una subconsulta se incluye **siempre entre paréntesis.**
- **La consulta que contiene la subconsulta** se denomina **Externa** y la **subconsulta** se denomina **Interna**.
- Una **subconsulta puede anidarse dentro de la cláusula WHERE o HAVING** de una instrucción externa **SELECT, INSERT, UPDATE o DELETE** o dentro de otra **subconsulta.**
- Una subconsulta puede **aparecer en cualquier parte en la que se pueda usar una expresión**, si **devuelve un valor individual**, excepto en una lista ORDER BY.
- Las subconsultas se utilizan para **dividir una consulta compleja en una serie de pasos lógicos y solucione un problema con una sola instrucción.**
- Tiene la misma **sintaxis que una sentencia SELECT normal** con alguna limitación: Por ejemplo, sólo puede incluir una cláusula ORDER BY cuando se especifica también una cláusula TOP

Ejemplo: *La siguiente consulta muestra para cada empleado, su salario y la **diferencia entre el salario** y el **salario promedio de todos los empleados***

```
SELECT NSS, Nombre, Apel1, Apel2, Salario, Salario – (SELECT AVG(Salario) FROM Empleados) AS [DIFERENCIA]
FROM Empleados
```

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

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

```
SELECT * FROM Empleados
WHERE NumDep=1
```

Pero si queremos saber los datos de los empleados del ***departamento de investigación***:

```
SELECT *
FROM Empleados
WHERE NumDep= ??   ----------> No sabemos el número de departamento de investigación.
                               Los  ?? deberán ser  sustituidos y deberá ser igual  "al número
                                    del departamento de investigación"
SELECT *
FROM Empleados
WHERE NumDep IN
                (SELECT NumDep  FROM Departamentos  WHERE NombreDep="INVESTIGACION")
```

Existen los siguientes **tipos de subconsultas**:

✓ **ANIDADAS**:

- Devolver un **único valor o una lista de valores**.
- La **subconsulta se ejecuta una única vez** y entonces utiliza el valor o la lista de valores de la subconsulta en la consulta externa.
- Podemos tener los siguientes tipos de consultas anidadas:
  - Las que **devuelven un único valor**, aparecen en la lista de selección de la consulta externa o con un **operador de comparación sin modificar**.
  - Las que **generan una lista de valores**, aparecen con el operador **IN** o con un operador de comparación modificado **con ANY, SOME** o **ALL**.
  - Las que pueden **generar cualquier número de columnas y filas**, son utilizadas en **pruebas de existencia** especificadas con **EXISTS.**

✓ **CORRELACIONADAS:**

- Las que en la **consulta interna utiliza información de la consulta externa.**
- Se **ejecuta una vez por cada fila en la consulta externa**.

4.1 Subconsultas anidadas que devuelven un valor único

- Existen subconsultas que deben obligatoriamente **devolver un único valor,**
- 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 (=,<>, <,>,<=,>= ,!=, !>, !<)
- Son las que aparecen en el **SELECT d**e la consulta externa o las que aparecen en **WHERE o HAVING**.
- Se puede especificar una **función de agregado** o **especificar una columna** o **expresión** para **devolver un único valor.**
- Solo se pueden **comparar los mismos tipos de datos entre la consulta y la subconsulta.**
- No puede ocurrir que la subconsulta tenga dos columnas y ese resultado se compare usando una sola columna en la consulta general.

Por ejemplo:

```
/* Empleados que ganan menos que el empleado con DNI 34343434A */
SELECT DNI, nombre, apellido1 e.apellido2 , sueldo FROM empleado
WHERE sueldo < (SELECT sueldo     FROM empleado
                WHERE DNI='34343434A')
```

En realidad lo primero que hace la base de datos es calcular el resultado de la subconsulta (imaginemos que su sueldo es de 24000) y a continuación mostrará los empleados cuyo sueldo es menor que este.

Una **subconsulta que utilice los valores >,<,>=,...** tiene que **devolver un *único* valor**, ya que **si no se producirá un error**. Si la subconsulta no devuelve ninguna fila, la comparación opera como si la segunda expresión fuese nula.

```
¡CONSULTA CON ERROR!
/* Empleados que ganan menos que los empleados con DNI 34343434A y 54545487Z */
SELECT DNI, nombre, apellido1 e.apellido2 , sueldo FROM empleado
WHERE sueldo < (SELECT sueldo   FROM empleado
                WHERE DNI='34343434A' OR DNI=' 54545487Z')
```

¡MAL! La subconsulta devuelve más de un valor, es decir, 2 sueldos, los de los empleados con *DNI 34343434A, y 54545487Z y* si una subconsulta devuelve más de un valor y la utilizamos comparándola con los operadores de comparación (=,<>, <,>,<=,>= ,**!=, !>, !<**) **nos da un error**.

CONSULTAS CON VARIOS NIVELES DE ANIDAMIENTO

Las **subconsultas pueden anidarse** de forma que una subconsulta aparezca en la cláusula WHERE (por ejemplo) de otra subconsulta que a su vez forma parte de otra consulta, y ésta a su vez forma parte de otra consulta y así sucesivamente. SQL Server permite 32 niveles de anidamiento.

```
/* Empleados que trabajan en el proyecto Investigación */
SELECT DNI, nombre, apellido1 e.apellido2  FROM Empleado
WHERE DNI IN (SELECT DNI FROM EmpleadoProyecto
              WHERE numproyecto= (SELECT numproyecto FROM Proyecto
                                  WHERE nombreproyecto='Investigación'))
```

NOTA: En la anterior consulta en la primera subconsulta se utiliza el operador IN por que la consulta puede devolver varios empleados que trabajen en el proyecto y en la segunda subconsulta se utiliza = porque sólo devuelve un resultado.

SUBCONSULTAS VS COMBINACIONES

Con frecuencia, una consulta que contiene subconsultas puede ser escrita como una combinación, cuando en las columnas del select sólo se referencia a una tabla. Por ejemplo, la anterior consulta se puede formular con combinaciones:

```
/* Empleados que trabajan en el proyecto Investigación */
SELECT DNI, nombre, apellido1 e.apellido2
FROM Empleado E INNER JOIN  EmpleadoProyecto EP ON E.DNI=EP.DNI
                INNER JOIN Proyecto P ON EP.numproyecto=P.numproyecto
WHERE nombreproyecto='Investigación'
```

Otras veces, una consulta se puede formular sólo con subconsultas, por ejemplo:

```
/* empleados cuyo salario supera a la media del salario de todos los empleados */
SELECT DNI, nombre, apellido1 e.apellido2  FROM Empleado E
WHERE SALARIO >( SELECT AVG(Salario) FROM Empleado)
```

En Transact-SQL, normalmente no hay diferencias de rendimiento entre una instrucción que incluya una subconsulta y una versión semánticamente equivalente que no la incluya. El rendimiento de una consulta

puede ser similar con una combinación y una subconsulta. La diferencia es que la subconsulta puede requerir que el optimizador de consultas realice pasos adicionales, como por ejemplo la organización, lo cual puede influir en la estrategia de procesamiento. Normalmente, la utilización de combinaciones permite al optimizador de consultas recuperar datos de la manera más eficaz.

4.2 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.

No podremos utilizar un operador de comparación directamente ya que esa subconsulta devuelve más de una fila. La solución a esto es utilizar operadores **especiales entre el operador y la consulta,** que permiten el uso de subconsultas de varias filas.

Hay que utilizar los siguientes operadores;

Instrucción	Significado
ANY/SOME	Compara con cualquier valor de la subconsulta. La **instrucción es válida si hay algún valor en la subconsulta que permite que la comparación sea cierta**. Se **utiliza conjuntamente con los operadores de comparación** ((=,<>, <,>,<=,>= ,**!=, !>, !<**)
ALL	Compara con todos los valores de la consulta. La **instrucción resulta cierta si es cierta para todos los valores de la subconsulta.** Se **utiliza conjuntamente con los operadores de comparación** ((=,<>, <,>,<=,>= ,**!=, !>, !<**)
IN	Examina si el valor que **se compara es uno de los valores incluidos en la lista de valores** generados por la subconsulta. Es equivalente **=ANY**
NOT IN	Comprueba si **un valor no se encuentra en la lista de valores generados** por la subconsulta

```
/* Empleados cuyo departamento se encuentra situado en Vigo */
SELECT DNI, nombre, apellido1 e.apellido2  FROM empleado
WHERE cod_departamento IN (SELECT codigo FROM departamento
                           WHERE lugar='Vigo')
/* Empleados cuyo departamento no se encuentra situado en Vigo */
SELECT DNI, nombre, apellido1 e.apellido2 FROM empleado
WHERE cod_departamento NOT IN (SELECT codigo FROM departamento
                               WHERE lugar='Vigo')
/* Empleadas cuya edad es mayor a la de cualquier empleado varón */
SELECT DNI, nombre, apellido1 e.apellido2 FROM empleado
WHERE sexo ='M'
AND edad > ANY (SELECT edad  FROM empleado
                WHERE sexo='H')
/* Empleados cuyo sueldo es mayor al de alguno de sus compañeros de departamento,
es decir, los que no tienen el sueldo más bajo de su departamento  */
SELECT DNI, nombre, apellido1 e.apellido2 , sueldo FROM empleado
WHERE sueldo > ANY (SELECT sueldo  FROM empleado e2
                WHERE empleado.cod_departamento=e.cod_departamento)

/* empleados cuyo salario es mayor que todos los salario de los empleados del
departamento 2 sin incluir a los de este departamento*/
SELECT NSS, Nombre, Apel1, Apel2, Salario FROM Empleados
WHERE Salario > ALL (SELECT Salario FROM Empleados  WHERE NumDep=2)
 AND NumDep<>2
```

La siguiente tabla muestra algún ejemplo del operador **ANY** y **ALL**

Valor 1	Operador	Valor 2	Resultado
4	> **ANY**	(2,3,6,9)	Cierto
4	= **ANY**	(2,3,6,9)	Falso
4	= **ANY**	(2,4,6,9)	Cierto
4	> **ALL**	(2,3,6,9)	Falso
4	< **ALL**	(5,6,9,11)	Cierto
11	>= **ALL**	(5,6,9,11)	Cierto

4.3 Test de existencia (EXISTS)

- El test de existencia **EXISTS** comprueba **si una subconsulta produce alguna fila de resultados**.

[NOT] EXISTS subconsulta

- Si la **subconsulta devuelve al menos una fila** el **resultado es verdadero**; en cualquier otro caso el resultado es **falso**.
- La subconsulta **no necesita devolver valores para este predicado; por tanto se utiliza *** (asterisco) en la **lista de selección de la subconsulta**, ya que no hay ninguna razón para utilizar nombres de columnas.
- Si se utiliza **NOT EXISTS** el **resultado será el contrario**.

```
/* Contar el número de empleados si hay algún empleado que gane más 1500 €*/
SELECT COUNT(*) FROM Empleados
WHERE  EXISTS (SELECT *  FROM Empleados  WHERE Salario> 1500)
```

4.4 Subconsultas correlacionadas

- Para las **subconsultas correlacionadas (o de referencias externas)** la **consulta interna utiliza información de la consulta externa**
- **Se ejecuta una vez por cada fila en la consulta externa.**
- Si en la subconsulta se utiliza la misma tabla que en la consulta externa, se **deben de utilizar alias** para **diferenciar los nombres de los campos en las tablas que se hace referencia**.

Las subconsultas anidadas solo hacen referencias a columnas de las tablas que están en su propia cláusula FROM. Esto significa que el resultado de la subconsulta puede evaluarse independientemente de sus sentencias antecedentes en todos los niveles. Por tanto, el SGBR la evalúa una sola vez y reemplaza los valores resultantes en el predicado donde se encuentre.

En cambio, en las **subconsultas correlacionadas, se especifica alguna columna de una tabla mencionada en la cláusula FROM de alguna consulta antecedente**. No se evalúan independientes de las consultas antecedentes, y por tanto, se evalúan múltiples veces

Normalmente las **subconsultas correlacionadas pueden ser rescritas como combinaciones.**

```
/*NSS, nombre y apellidos de los empleados que tengan asignado un proyecto*/
SELECT NSS, Nombre, Apel1, Apel2 FROM Empleado
WHERE  EXISTS (SELECT * FROM Participación
                    WHERE Partipacion.NSS= Empleados.NSS )
```

```
/* Empleados que tengan familiares  */
SELECT DNI, nombre, apellido1 e.apellido2 FROM Empleado
WHERE EXISTS (SELECT *   FROM familiar
                    WHERE empleado.codigo=familiar.cod_empleado)
```

4.5 Subconsulta como tabla derivada

- Es un **conjunto de registros dentro de una consulta** que **funciona como una tabla**, es decir, ocupa el lugar de la tabla en la cláusula FROM.

*Una tabla derivada se crea al utilizar una subconsulta **en lugar** de una tabla en una cláusula **FROM***

- Una **tabla derivada** es un uso especial de una subconsulta en una cláusula FROM a la que hace referencia un alias o nombre especificado por el usuario.
- **El conjunto de resultados de la subconsulta en la cláusula FROM crea una tabla que utiliza la instrucción SELECT externa**

En este ejemplo se utiliza una subconsulta para crear una tabla derivada en la parte interna de la consulta que consulta la parte externa. La tabla derivada propiamente dicha es funcionalmente equivalente a la consulta entera, pero aparece separada con propósitos ilustrativos.

```
SELECT *
FROM (SELECT NSS,SALARIO*1.20 AS SALARIOAUMENTO
      FROM EMPLEADOS)
                        AS TABLARESULTADO  --obligatorio poner un alias a la tabla derivada
```

Cuando se utilice como **tabla derivada**, hay que tener en cuenta que una subconsulta:

- Es **un conjunto de registros** dentro de una consulta que funciona como una tabla.
- **Ocupa el lugar de la tabla en la cláusula FROM.**
- Hay que poner un **nombre de alias a la tabla derivada**, igual que los nuevos campos generados que se vayan a utilizar en la consulta externa.

```
/* media del número de empleados por departamento. Se crea una tabla derivada con
los numdep y los empleados por numdep*/

SELECT AVG(CAST(NUMEMPLEADOS AS FLOAT))
FROM (SELECT NUMDEP, COUNT(*) AS NUMEMPLEADOS   FROM EMPLEADOS
      GROUP BY NUMDEP ) AS TABLARESULTADO
```

5. Operaciones de conjuntos

SQL Server permite **tres tipos de operaciones de conjuntos**: **UNION, EXCEPT** e **INTERSECT**. Para utilizar operaciones de conjuntos debemos cumplir una serie de normas:

- Las **consultas para unir deben tener el mismo número de campos**, y además **los campos** deben **tener tipos de datos compatibles**.
- Sólo puede haber **una única cláusula ORDER BY** al final de la sentencia SELECT.

5.1 Operador UNION

- El operador **UNION** permite **combinar los resultados de varias instrucciones SELECT en un único conjunto de resultados**.
- La palabra UNION permite añadir el resultado de un SELECT a otro SELECT.
- Para ello ambas instrucciones tienen que utilizar el **mismo número y tipo de columnas**.

Sintaxis

```
Instrucción_select
UNION [ALL] [,... n]
Instrucción_select
```

- SQL Server **elimina las filas duplicadas del conjunto de resultados**.

  Sin embargo, si se utiliza la **opción ALL, se incluyen todas las filas** (incluyendo los duplicados) en el conjunto de resultados.

- Los **nombres de las columnas** del conjunto de resultados se cogen de la **primera instrucción SELECT.** Por tanto, si quiere definir nuevos encabezados de columnas para el conjunto de resultados, debe crear los alias de las columnas en la primera instrucción SELECT.
- Si **quiere que el conjunto de resultados entero sea devuelto en un orden específico** debe especificar un orden de ordenación incluyendo una cláusula **ORDER BY después de la última instrucción SELECT**. Si no, puede que el conjunto de resultados no sea devuelto en el orden que usted quiere.
  - ✓ **SQL Server genera un error si se especifica una cláusula ORDER BY en cualquier otra instrucción SELECT de la consulta**.
  - ✓ Una cláusula **ORDER BY** utilizada con el operador UNION **sólo puede utilizar elementos** que aparecen en **la lista de selección de la primera instrucción SELECT**.
- Puede que tenga un mejor rendimiento si divide una consulta complicada en varias instrucciones SELECT y después utiliza el operador UNION para combinarlas.

Ejemplo: Se combina dos conjuntos de resultados. El primer conjunto de resultados devuelve el nombre, ciudad y código postal de cada empleado en la tabla Empleados. El segundo conjunto de resultados devuelve el nombre, ciudad y código postal de cada cliente de la tabla Clientes.

No se garantiza el orden de las filas devueltas a menos que especifique una cláusula ORDER BY, puede que los clientes sean listados antes que los empleados, aunque se haga referencia a la tabla Empleados antes en la instrucción.

```
SELECT Nombre +' '+ Apellidos AS Nombre, Ciudad, CodigoPostal  FROM Empleados
UNION
SELECT NombreCompleto, Ciudad, CodigoPostal FROM Clientes
```

Ejemplo: Se agrega una cláusula ORDER BY a la instrucción del Ejemplo 1.

```
SELECT Nombre +' '+ Apellidos AS Nombre, Ciudad, CodigoPostal FROM Empleados
UNION
SELECT NombreCompleto, Ciudad, CodigoPostal FROM Clientes
ORDER BY Nombre
```

Ejemplo: Hallar el salario medio de los empleados que tienen comisión y los que no.

```
SELECT 'CON COMISIÓN', avg(Salario) FROM  Empleados  WHERE Comision IS NOT NULL
UNION
SELECT 'SIN COMISIÓN', avg(Salario) FROM  Empleados WHERE comision IS NULL;
```

Se pueden combinar varias tablas con el operador **UNION.** Por ejemplo:

```
SELECT DNI, nombre, apellido1, apellido2  FROM Empleado
UNION
SELECT DNI, nombre, apellido1, apellido2  FROM Cliente
UNION
SELECT DNI, nombre, apellido1, apellido2  FROM Familiar;
```

5.2 Operadores EXCEPT e INTERSECT

EXCEPT:

- Devuelve los **valores distintos de la consulta izquierda que no se encuentran en la consulta derecha.**
- Este operador permite poner en práctica el operador de diferencia definido en el álgebra relacional. Permite localizar **las filas presentes en un conjunto de resultados y en el otro no.**

INTERSECT:

- Devuelve los **valores distintos que se encuentran tanto la consulta de la parte izquierda como la consulta de la parte derecha** del operador INTERSECT.
- Este operador corresponde al operador intersección del álgebra relacional y permitir identificar en un asola consulta, **las filas que están presentes en los dos conjuntos distintos de resultados**.

Las reglas básicas para combinar los conjuntos de resultados de dos consultas que utilizan EXCEPT o INTERSECT son los mismo que en el operador UNION.

Sintaxis:

```
Instrucción_select
EXCEPT| INTERSECT
Instrucción_select   [,... n]
```

Ejemplo: Si tenemos la tabla empleados y la tabla empleados_proyectos. Para ver **los empleados que no tienen asignado ningún proyecto** podríamos utilizar el operador EXCEPT

```
SELECT NSS FROM EMPLEADO
EXCEPT
SELECT NSS_EMPLEADO FROM EMPLEADO_PROYECTO
```

Ejemplo: Si tenemos la tabla empleados y la tabla empleados_proyectos. Para ver **los empleados que tienen asignado algún proyecto** podríamos utilizar el operador INTERSECT

```
SELECT NSS FROM EMPLEADO
INTERSECT
SELECT NSS_EMPLEADO FROM EMPLEADO_PROYECTO
```

Si **UNION, EXCEPT o INTERSECT** se utilizan con otros operadores en una expresión, ésta **se evalúa** en el contexto de la siguiente prioridad:

1. **Expresiones entre paréntesis**
2. El operando **INTERSECT**
3. **EXCEPT y UNION** se evalúan de **izquierda a derecha** según su posición en la expresión

Ej: Listar los que tienen asignados proyectos junto con el texto 'asignado proyecto' y los que no tienen proyectos con el texto sin 'asignar proyectos'

```
SELECT 'SIN ASIGNAR PROYECTO',NSS FROM EMPLEADO
EXCEPT
SELECT  'SIN ASIGNAR PROYECTO',NSS_EMPLEADO FROM EMPLEADO_PROYECTO
UNION
(SELECT 'ASIGNADO PROYECTO',NSS  FROM EMPLEADO
INTERSECT
SELECT  'ASIGNADO PROYECTO',NSS_EMPLEADO FROM EMPLEADO_PROYECTO)
ORDER BY 2
```