UNIDAD 7 CONSULTAS AVANZADAS

1. FUNCIONES

1.1. Introducción

Todos los SGBD implementan funciones para facilitar la creación de consultas complejas. Esas funciones dependen del SGBD que utilicemos, las que aquí se comentan son algunas de las que se utilizan con Oracle y MySQL.

Las funciones se usan dentro de expresiones y actúan con los valores de las columnas, variables o constantes.

Se utilizan en las cláusulas SELECT, WHERE y ORDER BY.

Todas las funciones reciben datos para poder operar (parámetros) y devuelven un resultado (que depende de los parámetros enviados a la función. Los argumentos se pasan entre paréntesis:

```
nombre_función(parámetro1, parámetro2, …)
```

Si la función no precisa parámetros no hace falta colocar los paréntesis. Como SYSDATE (en Oracle)(que devuelve la fecha del sistema)

Veremos que hay dos tipos de funciones:

- Funciones que operan con datos de la misma fila
- Funciones que operan con datos de varias filas diferentes (funciones de agrupación).

ORACLE dispone de una tabla para poder hacer pruebas, es la tabla DUAL que solo dispone de una fila y una columna.

```
SELECT SQRT(25) AS Raiz FROM DUAL
```

1.2. Funciones aritméticas

Redondeos

ORACLE	Descripción	MySQL
ROUND(n1,n2)	n1 redondeados a n2 posiciones decimales	ROUND
TRUNC(n1,n2)	n1 truncados a n2 posiciones decimales	TRUNCATE

Matemáticas

ORACLE	Descripción	MySQL
ABS(n)	Valor absoluto de n	ABS
CEIL(n)	Primer entero mayor o igual que n	CEIL
COS(n)	Coseno	COS
EXP(n)	e elevado a la potencia n (e = 2,71828183...)	EXP
FLOOR(n)	Primer entero menor o igual que n	FLOOR
LN(n)	Logaritmo neperiano de n	LN
LOG(m,n)	Logaritmo de n en base m	LOG
MOD(n1,n2)	Resto del cociente de n1/n2	MOD
POWER(n1,n2)	n1 elevado a n2	POW o POWER
SIGN(n)	-1 si n<0; si n = 0; +1 si n>0	SIGN
SIN(n)	Seno de n	SIN
SQRT(n)	Raíz cuadrada de n	SQRT
TAN(n)	Tangente de n	TAN

La mayor parte de las funciones matemáticas devuelven un valor con 38 cifras significativas, las funciones COS, EXP, LN, LOG, SIN, SQRT, TAN devuelven un valor con sólo 36 cifras significativas.

1.3. De grupos de valores (en Oracle y MySQL no hay diferencias)

AVG(n)	Calcula el valor medio de 'n' ignorando los valores nulos.
COUNT(* | expresión)	Cuenta el número de veces que la expresión evalúa algún dato con valor no nulo. La opción '*' cuenta todas las filas seleccionadas.
MAX(expresión)	Calcula el valor máximo de la expresión.
MIN(expresión)	Calcula el mínimo valor de la expresión.
SUM(expresión)	Obtiene la suma de valores de la expresión' distintos de nulos
VARIANCE(expresión)	Obtiene la varianza de los valores de éxpresión' distintos de nulos.

Nota: En todas las funciones de grupo, al indicar los argumentos se puede emplear las cláusulas **DISTINCT** y **ALL,** aunque no se suelen utilizar en las funciones AVG, SUM, MAX ni MIN, pero sí es más normal su uso en COUNT.

DISTINCT realiza una selección de filas cuyos valores en la columna especificada no están duplicados. La cláusula ALL recoge todas las filas aunque sus valores estén duplicados.

El formato de COUNT incluyendo DISTINCT y ALL es éste:

```
COUNT (* | [DISTINCT | ALL] expresión)
```

1.4. Funciones de cadenas de caracteres

Conversión de texto a mayúsculas a minúsculas

ORACLE	Descripción	MySQL
LOWER(c)	Devuelve c en minúsculas	LOWER o LCASE
UPPER(c)	Devuelve c en mayúsculas	UPPER o UCASE
INITCAP(c)	Devuelve c con la primera letra de cada palabra en mayúscula y las restantes en minúsculas.	

Funciones que devuelven valores carácter

ORACLE	Descripción	MySQL
CHR(c)	Código ASCII del primer carácter de c	CHAR
CONCAT(cad1, cad2)	Concatena cad1 con cad2	CONCAT, pero puede haber más de 2 argumentos
LPAD(cad1, n[,c2])	Devuelve cad1 con una longitud total n que se obtiene añadiendo a la izquierda cad2 (que por omisión es un espacio, ' ')	LPAD
RPAD(cad1, n[,cad2])	Devuelve cad1 con una longitud total n que se obtiene añadiendo a la derecha cad2 (que por omisión es un espacio, ' ')	RPAD
LTRIM(c1,[c2])	Devuelve c1 suprimiendo por la izquierda c2 (por omisión, ' ')	LTRIM
RTRIM(c1,[c2])	Devuelve c1 suprimiendo por la derecha c2 (por omisión, ' ')	RTRIM

REPLACE(c1,c2,c3)	Reemplaza en c1 todas las apariciones de c2 por c3 (c3 puede ser una cadena nula)	REPLACE
SUBSTR(c,n1[,n2])	Devuelve la cadena formada por n2 caracteres de c a partir de la posición 1	SUBSTRING
TRANSLATE(c1,c2,c3)	Reemplaza en c1 todas las apariciones de c2 por c3 (c3 no puede ser una cadena de caracteres nula)	

Funciones que devuelven valores numéricos

ORACLE	Descripción	MYSQL
ASCII(c)	Código ASCII del primer carácter c	ASCII
INSTR(c1,c2, [,n1[n2]])	Posición de la posición de la n-ésima aparición de c2 en c1 (la primera aparición es la opción predeterminada) a partir de la posición n1 (1 es el valor predeterminado)	INSTR
LENGTH(cad)	Devuelve el número de caracteres de cad.	LENGTH

1.5. Funciones de trabajos con nulos

Permiten definir valores a utilizar en el caso de que las expresiones tomen el valor nulo

ORACLE	Descripción	MYSQL
NVL(valor, sustituto)	Si el valor es NULL devielve el valor sustituto, de otro modo devuelve valor.	IFNULL
NVL2(valor, sustituto1, sustituto2)	Variante de la anterior, devuelve el valor sustituto1 si valor no es nulo. Si el valor es nulo devuelve el sustituto2	
NULLIF(valor1, valor2)	Devuelve nulo si valor1 es iguala a valor . De otro modo devuelve valor1	

1.6. Funciones para el manejo de fechas

Oracle puede almacenar datos de tipo fecha (DATE) y posee una interesante utilidad para formatear las fechas de cualquier manera que se pueda concebir. Tiene un formato por omisión 'DD/MM/YY', pero con la función TO_CHAR es posible mostrar las fechas de cualquier modo. Los literales de fecha deben encerrarse entre comillas simples.

Ejemplo: '12/08/08'.

Las funciones para el manejo de fechas son:

ORACLE	Descripción	MySQL
SYSDATE	Devuelve la fecha del sistema.	NOW, SYSDATE
ADD_MONTHS(fecha, n)	Devuelve la fecha incrementada en n meses.	PERIOD_ADD(P,n)
LAST_DAY(fecha)	Devuelve la fecha del último día del mes que contiene fecha	LAST_DAY
MONTHS_BETWEEN (fecha1, fecha2)	Devuelve la diferencia en meses entre las fechas 'fecha1' y 'fecha2' Devuelve el número de meses entre los periodos P1 y P2. P1 y	PERIOD_DIIF(P1, P2)

	P2 deben estar en el formato YYMM o YYYYMM. Los argumentos periodo P1 y P2 no son valores de fecha:	
NEXT_DAY(fecha, cad)	Devuelve la fecha del primer día de la semana indicando por ´cad' después de la fecha indicada por 'fecha'. El día de la semana se indica con su nombre, es decir, lunes (monday), martes (tuesday), miércoles (wednesday), jueves(thursday), viernes (friday), sabado (saturday), domingo (sunday)	
EXTRACT(*valor* FROM *fecha*)	Extrae un valor de una fecha concreta. El valor puede ser ***day*** (día), ***month*** (mes), ***year*** (año), etc.	EXTRACT
GREATEST(*fecha1, fecha2,..*)	Devuelve la fecha más moderna la lista	
LEAST(*fecha1, fecha2,..*)	Devuelve la fecha más antigua la lista	
ROUND(*fecha* [,*'formato'*]	Redondea la fecha al valor de aplicar el formato a la fecha. El formato puede ser: • 'YEAR' Hace que la fecha refleje el año completo • 'MONTH' Hace que la fecha refleje el mes completo más cercano a la fecha • 'HH24' Redondea la hora a las 00:00 más cercanas • 'DAY' Redondea al día más cercano	
TRUNC(*fecha* [*formato*])	Igual que el anterior pero trunca la fecha en lugar de redondearla.	

1.7. Funciones de conversión

La mayoría de las funciones que hemos visto hasta ahora son funciones de transformación, esto es, cambian los objetos. Hay otras funciones que cambian los objetos de una manera especial, pues transforman un tipo de de dato en otro. Las funciones de conversión elementales se muestran en la siguiente tabla:

ORACLE	**Descripción**	**MySQL**
TO_CHAR(fecha, 'formato')	Convierte una fecha (de tipo DATE) a tipo VARCHAR2 en el formato especificado. El formato es una cadena de caracteres que puede incluir las máscaras d formato definidas en la Tabla de control de formato de fechas, y donde es posible incluir literales definidos por el usuario encerrados entre comillas dobles.	
TO_CHAR(numero, 'formato')	Esta función convierte un número (de tipo NUMBER) a tipo VARCHAR2 en el formato especificado. Los formatos numéricos se muestran en otra tabla.	
TO_DATE(cad, 'formato')	Convierte 'cad', de tipo VARCHAR2 o CHAR, a un valor de tipo DATE según el 'formato' especificado.	STR_TO_DATE(str,format)
TO_NUMBER(cadena [,'formato'])	Convierte la 'cadena' a tipo NUMBER según el formato especificado. La cadena ha de contener números, el carácter decimal o el signo menos a la izquierda. No puede haber espacios entre los números, ni otros caracteres.	

Máscaras de formato numéricas

cc o scc	Valor del siglo
YY	Año en formato de dos cifras
YYYY	Año en formato de cuatro cifras
MM	Mes en formato de dos cifras
MON	Las tres primeras letras del mes
MONTH	Nombre completo del mes
DY	Día de la semana en tres letras
DAY	Día completo de la semana
DD	Día en formato de dos cifras
D	Día de la semana (del 1 al 7)
Q	Semestre
WW	Semana del año
D	Día de la semana (del 1 al 7)
DDD	Día del año
AM	Indicador AM
PM	Indicador PM
HH12	Hora de 1 a 12
HH24	Hora de 0 a 23
MI	Minutos (0 a 59)
SS	Segundos (0 a 59)
SSSS	Segundos desde medianoche
/ . ,:;'	Posición de los separadores, donde se pongan estos símbolos aparecerán en el resultado

Máscara de formato de caracteres

syear ó **year**	Año en inglés (ej: nineteen-eighty-two)
month	Nombre del mes (ENERO)
mon	Abreviatura de tres letras del nombre del mes (ENE)
day	Nombre del día de la semana (LUNES)
dy	Abreviatura de tres letras del nombre del día (LUN)
a.m ó **p.m**	Muestra a.m o p.m dependiendo del momento del día.
b.c ó **a.d**	Indicador para el año (antes o después de Cristo)

En el caso de números se utilizan estos formatos

9	Posición del número
0	Posición del número (muestra ceros)
$	Formato dólar
L	Símbolo local de la moneda
S	Hace que aparezca el símbolo del signo
D	Posición del símbolo decimal (en español, la coma)
G	Posición del separador de grupo (en español el punto)

1.8. CAST

Función muy versátil que permite convertir el resultado a un tipo concreto.

```
CAST(expresión AS tipoDatos)
```

Ejemplo:

```
SELECT CAST(2.34567 AS NUMBER(7,6)) FROM DUAL;
```

Lo interesante es que puede convertir de un tipo a otro. Por ejemplo imaginemos que tenemos una columna en una tabla mal planteada en la que el precio de las cosas se ha escrito en Euros. Los datos son (se muestra sólo la columna precio):

precio
25.2 €
2.8 €
123.65 €
.78 €
.123 €
20 €

Imaginemos que queremos doblar el precio, no podremos porque la columna es de tipo texto, por ello debemos tomar sólo la parte numérica y convertirla a número, después podremos mostrar los precios multiplicados por dos:

```
SELECT 2 * CAST(SUBSTR(precio,1,INSTR(precio,'€')-2) AS NUMBER)
FROM precios;
```

La combinación de SUBSTR e INSTR es para obtener sólo los números. Incluso es posible que haya que utilizar REPLACE para cambiar los puntos por comas (para utilizar el separador decimal del idioma español).

1.9. Otras funciones

DECODE (en Oracle)

Función que permite realizar condiciones en una consulta Se evalúa una expresión y se colocan a continuación pares valor, resultado de forma que si se la expresión equivale al valor, se obtiene el resultado indicado. Se puede indicar un último parámetro con el resultado a efectuar en caso de no encontrar ninguno de los valores indicados.

```
DECODE(expresión, valor1, resultado1
[,valor2, resultado2,...]
```

```
[,valorPordefecto])
```

Ejemplo:

```
SELECT
DECODE(cotizacion,1, salario*0.85,
       2,salario * 0.93,
       3,salario * 0.96,
       salario)
FROM empleados;
```

En el ejemplo dependiendo de la cotización se muestra rebajado el salario: un 85% si la cotización es uno, un 93 si es dos y un 96 si es tres. Si la cotización no es ni uno ni dos ni tres, sencillamente se muestra el salario sin más.

Ésta es una función IF-THEN-ELSE

VSIZE (expresión) (En Oracle)

Devuelve el número de bytes que ocupa una expresión. Si expresión es nulo, la función devuelve nulo.

USER (En Oracle y MySQL)

Esta función devuelve el nombre del usuario actual.

UID

Devuelve el identificador del usuario actual.

Para ver más funciones de MySQL podéis consultar la página Web:

http://mysql.conclase.net/curso/index.php?cap=011#

2. CONSULTAS DE RESUMEN O AGRUPACIONES

2.1. Introducción

SQL permite agrupar las filas de una tabla, seleccionadas en una consulta, y obtener salidas, calculadas a partir de los grupos formados.

El **criterio para agrupar** suele ser una de las columnas de la tabla. Si no se especifica ninguno, las filas de la tabla, seleccionadas en la consulta, forman un grupo único.

Las salidas obtenidas son los resultados de aplicar **funciones de agregado** a los grupos de filas.

Funciones de columna o funciones de grupo

Son las que hemos visto en el apartado de funciones

Realizan un cálculo sobre un conjunto de valores y devuelven un solo valor. Con excepción de COUNT, las funciones de agregado omiten los valores NULL.

FUNCIONES	DESCRIPCIÓN DE LA FUNCIÓN
SUM([ALL | DISTINCT] *expresión*)	Calcula la suma de valores de la expresión o de la columna indicada dentro del paréntesis, teniendo en cuenta que la cláusula DISTINCT omite valores repetidos. Sólo suele utilizarse en columnas numéricas.
AVG([ALL | DISTINCT] *expresión*)	Calcula el valor medio de la expresión que se indique dentro del paréntesis, teniendo en cuenta que los valores NULL no son incluidos.
MIN([ALL | DISTINCT] *expresión*)	Devuelve el valor mínimo de la expresión que le acompaña.
MAX([ALL | DISTINCT] *expresión*)	Devuelve el valor máximo de la expresión que le acompaña.
COUNT([ALL | DISTINCT] *expresión)*	Cuenta el número de valores de datos que hay en una expresión, sin incluir los valores NULL
COUNT(*)	Cuenta todas las filas de la tabla, sin considerar que en algunas columnas existan valores NULL.
STDEV(*expresión*)	Calcula la desviación típica para los valores de la expresión.
VAR(*expresión*)	Calcula la varianza para los valores de la expresión.

Formato de consulta con funciones de agregado (sin criterio de agrupación)

```
SELECT funciones_de_agregado
FROM tabla
[WHERE condición]
```

2.2. Cláusula GROUP BY

Por lo general, las funciones de columna se utilizan sobre más de un grupo de filas. La cláusula GROUP BY establece el criterio de agrupación.

Cuando se especifica GROUP BY 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.

Formato de consulta con funciones de agregado (con criterio de agrupación)

```
SELECT lista de campos, funciones_de_agregado
FROM lista_de_tablas
WHERE condición_de_selección
GROUP BY lista_de_columnas_para_agrupar
ORDER BY especificaciones_para_ordenar
```

Se utiliza la cláusula WHERE para excluir aquellas filas que no se desea agrupar.

En la cláusula GROUP BY, se indican las columnas por las que se agrupa. Se crea un único registro por cada valor distinto en las columnas del grupo.

Por ejemplo, si agrupamos en la siguiente tabla por las columnas expediente, nombre, se creará un único registro por cada expediente y nombre distintos.

EXPEDIENTE	CURSO	ASIGNATURA	NOTA
1111	1	Lengua	5
1111	2	Matemáticas	8
1111	1	Francés	5
2222	1	Lengua	6
2222	3	Matemáticas	9
2222	1	Física	1
3333	1	Inglés	7
3333	3	Matemáticas	5

La consulta siguiente

```
SELECT expediente, curso
FROM alumnos
GROUP BY expediente, curso;
```

creará esta salida:

EXPEDIENTE	CURSO
1111	1
1111	2
2222	1
2222	3
3333	1
3333	3

Es decir, es un resumen de los datos anteriores. Los datos de asignatura y nota no están disponibles directamente ya que son distintos en los registros del mismo grupo. Sólo se pueden utilizar desde funciones. Es decir, esta consulta es errónea:

```
SELECT expediente, curso, nota
FROM alumnos
GROUP BY expediente, curso;
```

La siguiente consulta si que tiene sentido:

```
SELECT expediente, curso, SUM(nota)
FROM alumnos
GROUP BY expediente, curso;
```

y nos saldría como resultado, nos suma las notas para cada grupo.

EXPEDIENTE	CURSO	SUM(NOTA)
1111	1	10
1111	2	8
2222	1	7
2222	3	9
3333	1	7
3333	3	5

2.3. Selección de grupos

Del mismo modo que la cláusula WHERE permite la selección de filas en una sentencia SELECT, la cláusula **HAVING** permite realizar una selección sobre los grupos obtenidos por la cláusula **GROUP BY**.

Formato de consulta con selección de grupos

```
SELECT campos, funciones_de_columna
FROM lista_de_tablas
WHERE condición_de_selección
GROUP BY lista_de_columnas_para_agrupar
HAVING condición_de_selección
ORDER BY especificaciones_para_ordenar
```

La **condición de selección** de la cláusula HAVING podrá estar formada por constantes, columnas de agrupación y funciones de agregado.

La cláusula **HAVING actúa como un filtro sobre filas agrupadas**, a diferencia de la cláusula WHERE que actúa sobre las filas antes de la agrupación.

Ejemplo: Seleccionar los oficios que tengan dos o más empleados, cuyo salario supere las 140000 euros.

```
SSELECT oficio, COUNT (*)
FROM empleados
WHERE salario > 140000
GROUP BY oficio
HAVING COUNT (*) >= 2
```

```
OFICIO         COUNT(*)
---------- ----------
DIRECTOR    2
VENDEDOR    2
```

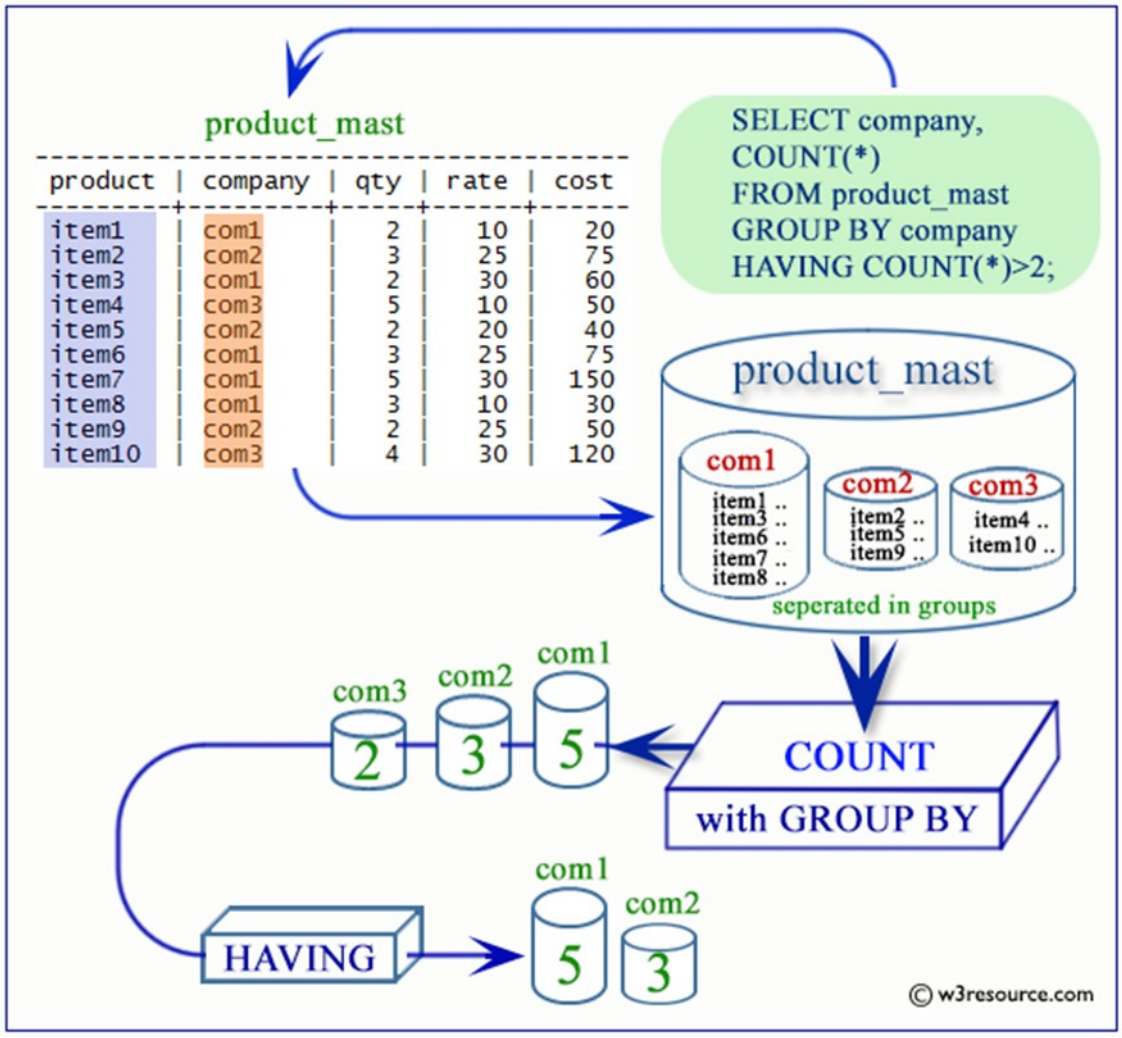
SQL realiza la selección de grupos en el proceso siguiente:

- A partir de la tabla sobre la que se realiza la consulta, la cláusula WHERE actúa como un primer filtro que da como resultado una tabla interna cuyas filas cumplen la condición especificada en el WHERE.
- La cláusula GROUP BY produce la agrupación de las filas de la segunda tabla, dando como resultado una tercera tabla.
- Se calculan los valores de la funciones de totales (COUNT, SUM; AVG, ...)
- La cláusula HAVING actúa filtrando las filas de la tercera tabla, según la condición de selección especificada,

dando como resultado la salida de la consulta.

- El resultado se ordena en base a ORDER BY.

Veamos un ejemplo gráfico:



2.4. Subconsultas en la selección de grupos.

Aunque las subconsultas suelen encontrarse sobre todo en la cláusula WHERE, también pueden usarse en la HAVING formando parte de la selección del grupo de filas efectuada por dicha cláusula.

2.5. Algunas cuestiones importantes

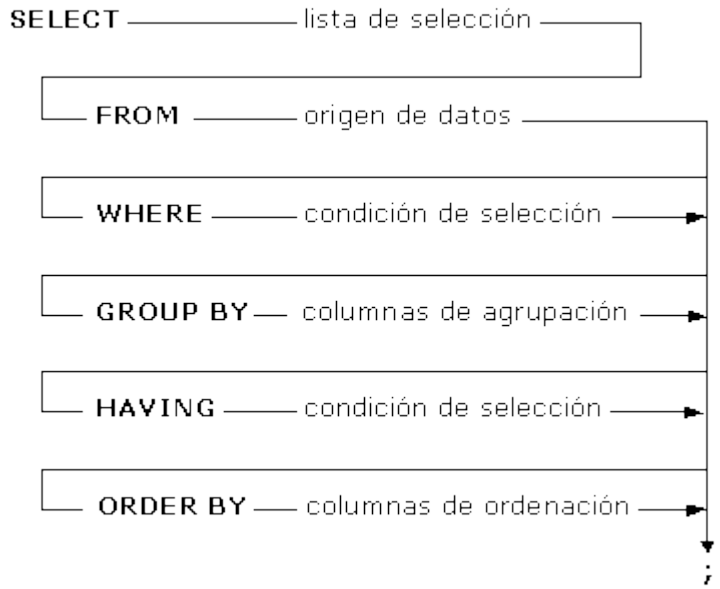
A continuación se plantean algunas cuestiones que es importante tener en cuenta cuando se realizan agrupaciones:

- Cuando se utilizan funciones de grupo en la cláusula SELECT sin que haya GROUP BY, el resultado de ejecutar la consulta obtiene una sola fila.
- A diferencia del resto de funciones que proporciona SQL, las funciones de grupo sólo se utilizan en las cláusulas SELECT y HAVING, nunca en la cláusula WHERE.
- La sentencia SELECT tiene dos cláusulas para realizar restricciones: WHERE y HAVING. Es muy importante saber situar cada restricción en su lugar: las restricciones que se deben realizar a nivel de filas, se sitúan en la cláusula WHERE; las restricciones que se deben realizar sobre grupos (normalmente involucran funciones de grupo), se sitúan en la cláusula HAVING.

- Una vez formados los grupos mediante la cláusula GROUP BY (son grupos de filas, no hay que olvidarlo), del contenido de cada grupo sólo es posible conocer el valor de las columnas por las que se ha agrupado (ya que dentro del grupo, todas las filas tienen dichos valores en común), por lo que sólo estas columnas son las que pueden aparecer, directamente, en las cláusulas SELECT y HAVING. Además, en estas cláusulas, se pueden incluir funciones de grupo que actúen sobre las columnas que no aparecen en la cláusula GROUP BY.

- Evidentemente, en la condición de la cláusula HAVING sólo pueden aparecer restricciones sobre columnas por las que se ha agrupado y también funciones de grupo sobre cualquier otra columna de la tabla. Lo mismo sucede en la cláusula SELECT: sólo es posible especificar de manera directa columnas que aparecen en la cláusula GROUP BY y también funciones de grupo sobre cualquier otra columna. Cuando en las cláusulas SELECT o HAVING aparecen columnas que no se han especificado en la cláusula GROUP BY y que tampoco están afectadas por una función de grupo, se produce un error.

- Recordar que el orden de las cláusulas debe ser el que se muestra en el siguiente esquema:



- La cláusula GROUP BY forma grupos con las filas que tienen en común los valores de una o varias columnas. Sobre cada grupo se pueden aplicar las funciones de columna (SUM, MAX, MIN, AVG, COUNT), también denominadas funciones de grupo si se usan con GROUP BY. Estas funciones, utilizadas en la cláusula SELECT, se aplican una vez para cada grupo.

Veamos con el siguiente ejemplo el orden de ejecución de la cláusula GROUP BY. La siguiente consulta muestra el código de los clientes y número de facturas realizadas por cada uno, que las ha cursado el empleado con código '705'.

```
SELECT id_cliente, COUNT(*)
FROM pedidos
WHERE id_empleado=705
GROUP BY id_cliente;
```

El modo en que se ejecuta la sentencia anterior es: Se toma la tabla especificada, pedidos, y se seleccionan las filas que cumplen la restricción (WHERE). A continuación, las facturas se separan en grupos, de modo que en un mismo grupo sólo hay facturas de un mismo cliente (GROUP BY cid_cliente), con lo cual hay tantos grupos como clientes

hay con facturas realizadas por el empleado con código 705. Finalmente, de cada grupo se muestra el código del cliente y el número de facturas que hay en el grupo (son las facturas de ese cliente): COUNT(*).

3. COMBINACIONES ESPECIALES: UNIÓN, INTERSECCIÓN, DIFERENCIA

Los operadores relacionales UNION, INTERSECT y MINUS son operadores de conjuntos. Los conjuntos son las filas resultantes de cualquier sentencia SELECT válida que permiten combinar los resultados de varias SELECT para obtener un único resultado.

Supongamos que tenemos dos listas de centros de enseñanza de una ciudad y que queremos enviar a esos centros una serie de paquetes de libros.

Dependiendo de ciertas características de los centros, podemos enviar libros a todos los centros de ambas listas (UNION), a los centros que están en las dos listas (INTERSECT) o a los que están en una lista y no están en la otra (MINUS). El formato de SELECT con estos operadores es el siguiente:

```
SELECT ... FROM ... WHERE
Operador de conjunto
SELECT ... FROM ... WHERE
```

3.1. Unión

El operador UNION combina los resultados de dos consultas. Permite añadir el resultado de un SELECT a otro SELECT. Para ello ambas instrucciones tienen que utilizar el mismo número y tipo de columnas, y además en el mismo orden.

Ejemplo: Supongamos que tenemos las siguientes tablas de alumnos:

- La tabla alumnos, contiene los datos de los alumnos que se han matriculado este curso en el centro.
- La tabla nuevos, contiene los datos de los alumnos que han reservado plaza para el curso que viene.
- La tabla antiguos contiene los datos de los antiguos alumnos del centro.

Cada tabla es de la forma:

```
Nombre          Tipo
-----------     -------
NOMBRE          VARCHAR(20)
EDAD            NUMERIC(2)
LOCALIDAD       VARCHAR(15)
```

Si queremos visualizar los nombres de los alumnos actuales y de los futuros alumnos, ejecutaríamos la siguiente sentencia:

```
SELECT nombre FROM alumnos UNION SELECT nombre from NUEVOS;
```

UNION crea una sola tabla con registros que están presentes en ambas consultas. Si están repetidos sólo aparecen una vez, para mostrar los duplicados se utiliza **UNION ALL** en lugar de UNION

3.2. Intersección

El operador INTERSECT devuelve las filas que son iguales en ambas consultas. Todas las filas serán eliminadas antes de la generación del resultado final.

Ejemplo: Obtener los nombre de los alumnos que están actualmente matriculados en el centro y que estuvieron en el centro hace ya algún tiempo.

```
SELECT nombre FROM alumnos INTERSECT SELECT nombre FROM antiguos;
```

Esta consulta también se puede realizar utilizando el operador IN:

```
SELECT nombre FROM alumnos
```

```
WHERE nombre IN (SELECT nombre FROM antiguos);
```

3.3. Diferencia

El operador MINUS devuelve aquellas filas que están en la primera SELECT y no están en la segunda. Las filas duplicadas del primer conjunto se reducirán a una fila única antes de que empiece la comparación con el otro conjunto.

Ejemplo: Obtener los nombres y localidades de los alumnos que están actualmente en el centro y que nunca estuvieron anteriormente en él, ordenados por localidad

```
SELECT nombre, localidad FROM alumnos

MINUS

SELECT nombre, localidad FROM antiguos ORDER BY localidad;
```

Esta consulta se podría haber hecho con el operador NOT IN

```
SELECT nombre, localidad FROM alumnos

WHERE nombre NOT IN (SELECT nombre FROM antiguos)

ORDER BY localidad;
```

Ejemplo: Seleccionar los nombres de los alumnos de la tabla alumnos que estén en nuevos y no en antiguos

```
SELECT nombre FROM alumnos INTERSECT SELECT nombre FROM nuevos MINUS SELECT nombre FROM antiguos;
```

Esta misma consulta se puede obtener con el operador IN

```
SELECT nombre FROM alumnos WHERE nombre IN(SELECT nombre FROM nuevos MINUS SELECT nombre FROM antiguos);
```

Ejemplo: Seleccionar los nombres de la tabla alumnos que estén en nuevos o en antiguos:

```
SELECT nombre FROM alumnos WHERE nombre IN (SELECT nombre FROM nuevos UNION SELECT nombre FROM antiguos);
```

3.4. Reglas para utilizar los operadores de conjuntos.

Los operadores de conjuntos pueden encadenarse. Los conjuntos se evalúan de izquierda a derecha; para forzar precedencia se pueden utilizar paréntesis.

```
(SELECT ...
...
UNION
SELECT ....
...)
MINUS
SELECT ...
/*Primero se hace la unión y luego la diferencia*/
```

Estos operadores se pueden manejar con consultas de diferentes tablas, siempre que se apliquen las siguientes reglas:

- Las columnas de las dos consultas se relacionan en orden, de izquierda a derecha.
- Los nombres de columna de columna de la primera SELECT no tienen porqué ser los mismos que los nombres de columna de la segunda.
- Las SELECT necesitan tener el mismo número de columnas.

Los tipos de datos deben coincidir, aunque la longitud no tiene porque ser la misma.