Unidad 6: Manipulación de BBDD Relacionales II - Consulas, Rutinas y Vistas

6.1 Introducción.

En este tema vamos a continuar con el DISEÑO FÍSICO DE DATOS e IMPLEMENTACIÓN. Esto es, después de haber creado la estructura de los datos, como vimos en la Unidad 5, y haber aprendido a introducir, modificar y eliminar datos en dicha estructura, como vimos en la Unidad 6, ha llegado el momento de aprender a hacer consultas sobre dichos datos, esto es, obtener INFORMACIÓN, que en definitiva es nuestro objetivo.

Para ello veremos las sentencias SQL que nos permitirán realizar esta tarea.

Como también vimos en la Unidad 5, las sentencias SQL básicas, independientemente del SGBD con el que trabajemos, son bastante estándar y, salvo particularidades de cada sistema, se dividen según el siguiente esquema:

A.- Sentencias de definición de datos (DDL):

1. Creación de estructuras de datos ==> Create
2. Modificación de estructuras de datos ==> Alter
3. Eliminación de estructuras de datos ==> Drop

B.- Sentencias de modificación de datos (DML):

1. Inserción de datos => Insert
2. Modificación de datos ==> Update
3. Eliminación de datos ==> Delete
4. Consulta de datos ==> Select

C.- Sentencias de control de datos

En este tema nos centraremos en las cuarta sentencias DML, que es la que se utiliza para consultar los datos de una base de datos.

En realidad, La sentencia SELECT, es la que nos permite hacer CONSULTAS, QUERIES, que es lo que le da nombre al lenguaje SQL (Sentences Query Language).

Debemos recordar que cuando manipulamos el contenido de la base de datos, tendremos que tener muy presentes las restricciones de clave, de integridad referencial y de usuario.

El alumnado debe tener presente que debe repasar el apartado 3 de la Unidad 5, ya que en el se explica de forma genérica los elementos de SQL, los tipos de datos, etc. Además mientras trabajemos con BD's siempre es posible que necesitemos utilizar alguna de las sentencias vistas en dicha Unidad.

6.1.1 Consultar datos. Sentencia SELECT. Cláusulas básicas de la sentencia SELECT.

La sentencia SELECT es la sentencia más utilizada de SQL. Esta sentencia es la que nos proporcionará la información que necesitamos obtener en cada caso.

Es decir, una vez creada la estructura en la que se almacenarán los datos, y guardados los datos en dicha estructura, mediante el uso de la sentencia SELECT proporcionaremos la INFORMACIÓN que necesitamos utilizando esos datos.

Dado el amplio abanico de posibilidades que pueden surgirnos a la hora de operar con los datos almacenados para obtener la información deseada, la sentencia SELECT puede ser muy sencilla o muy compleja.

Está organizada en CLAÚSULAS, algunas de ellas son de obligado uso y otras no.

La sentencia SELECT unifica, en una sola, varias de las sentencias utilizadas en álgebra (PROYECT, SELECT, JOIN, GROUP BY) pero la filosofía es la misma.

Debido a la complejidad de la sentencia, la veremos en dos fases. En este apartado veremos el uso y la sintaxis de una sentencia SELECT simplificada y más adelante estudiaremos la sintaxis completa y el significado de todas sus cláusulas.

Sintaxis básica de Consulta/Query

< expresion_query> ::=

<sentencia_query>
[UNION [ALL] -- *(ver unir más de una query)*
<sentencia_query> [*...n*]]

< sentencia_query> ::=

<CLAUSULA_SELECT> □	Indicamos los campos y/o resultado de expresiones que queremos mostrar en el resultado separados por comas. Se obtendrá una columna por cada campo o expresión especificados en la cláusula. Si se especifica DISTINCT se eliminarán los resultados repetidos.
<CLAUSULA_FROM> □	En esta cláusula indicamos el origen de los datos con los que vamos a trabajar. Es decir, la tabla/s y/o vista/s donde se encuentran los datos. Usaremos JOIN para tratar datos relacionados.
[<CLAUSULA_WHERE>] □	Filtramos las filas que cumplan una condición (predicado) descartando las que no satisfagan dicho predicado.
[<CLAUSULA_GROUP_BY>] □	-- la dejamos para ver más adelante (en esta unidad)
[<CLAUSULA_HAVING>] □	-- la dejamos para ver más adelante (en esta unidad)
[<CLAUSULA_ORDER_BY>] □	Se ordenarán las filas resultantes por el campo/s y/o expresión/es que se especifique en esta cláusula separados por comas. Por cada atributo o expresión se podrá ordenar de forma ascendente (opción por defecto) o descendente.

<CLAUSULA_SELECT>::=
SELECT [ALL | DISTINCT]
<listaSel >

<listaSel >::= {*
|{nom_tabla | nom_vista | alias_tabla}.*
|[{nom_tabla | nom_vista | alias_tabla}].nomcolumna [AS alias_colum]
|[alias_colum =]< expresion >
} [,..n]

<CLAUSULA_FROM>::=
FROM { <nom_tabla > |< nom_vista> [AS alias_tabla]} [**,**...*n*] | <expresion_join>

<expresion_join> ::=
{{ <nom_tabla1 > |< nom_vista1> [AS alias_tabla1]} JOIN | INNER_JOIN | LEFT_JOIN | RIGHT_JOIN | STRAIGHT_JOIN
{ <nom_tabla2 > |< nom_vista2> [AS alias_tabla2]}
ON { <nom_tabla1 > |< nom_vista1>|alias_tabla1}.nom_colum = { <nom_tabla2 >
|< nom_vista2> |alias_tabla2}.nom_colum} [JOIN …n]

<CLAUSULA_WHERE>::=
[WHERE < expresion_predicado>]

< **expresion_predicado**>::= predicado simple o compuesto (predicados simples unidos mediante AND/OR) con el que filtraremos las filas que queremos.

<CLAUSULA_ORDER_BY>::=
ORDER BY { <expresión_*order_by*> | posición_*columna* [ASC | DESC] } [**,**...*n*]

<expresión_*order_by*> *::= columna o expresión por la que se va a ordenar*

Nota.- Las expresiones que podremos utilizar se harán siguiendo el apartado 5.3.1 (Elementos de las órdenes SQL) de la Unidad 5.

Orden de ejecución de las sentencias SELECT

Es muy importante, también, entender cual es el orden en el que el servidor de datos ejecutará cada una de las cláusulas:

1. CLÁUSULA FROM
2. CLÁUSULA WHERE (opcional)
3. CLÁUSULA GROUP BY (opcional)
4. CLÁUSULA HAVING (opcional)
5. CLÁUSULA SELECT
6. CLÁUSULA ORDER BY (opcional)

Cuando en la cláusula FROM se especifica una sola relación se buscan los valores en ella. Cuando se pone más de una relación (separada por comas) se realiza el **producto cartesiano** de estas y sobre ese resultado se buscan los datos. Para evitar que se produzca el producto cartesiano, lo que puede hacer que las consultas sean muy lentas, **la solución es introducir en la cláusula FROM expresiones con "JOIN"**.

Como ya se ha comentado, las cláusulas GROUP BY y HAVING se verán más adelante en esta unidad.

Unir más de una sentencia QUERY

A veces nos puede interesar mostrar los resultados de más de una sentencia SELECT en un solo resultado, esto lo haremos mediante uno de los siguientes operadores:

1. UNION [ALL] sentencia_select_1 UNION [ALL] sentencia_select_2

Obtendremos un solo resultado en el que se mostrarán las filas de la primera sentencia y después las de la segunda sentencia.
Ambas sentencias deben tener el mismo número de columnas y cada una de ellas debe tener el mismo dominio.
Si utilizamos ALL, las filas comunes se mostrarán repetidas, si no utilizamos ALL, las filas comunes se mostrarán una sola vez.

2. INTERSECTED sentencia_select_1 INTERSECTED sentencia_select_2

Obtendremos un solo resultado en el que se mostrarán las filas comunes a la primera y a la segunda sentencia.
Ambas sentencias deben tener el mismo número de columnas y cada una de ellas debe tener el mismo dominio.

3. EXCEPT sentencia_select_1 EXCEPT sentencia_select_2

Obtendremos un solo resultado en el que se mostrarán las filas de la primera sentencia que no estén en la segunda sentencia.
Ambas sentencias deben tener el mismo número de columnas y cada una de ellas debe tener el mismo dominio.
Nota.- Los operadores INTERSECTED y EXCEPT no están implementados en MySQL. Se ha decidido mantener por que en otras herramientas si existen. En MySQL tendremos que emularlas con otras operaciones.

6.1.2 Concepto de predicado. Predicados básicos.

Los predicados son expresiones contenidas en sentencias u órdenes SQL y que devuelven un valor booleano.

El resultado de la evaluación de las condiciones del predicado para cada fila, devolverá un valor:

TRUE (verdadero) para las que cumplan la condición.

FALSE (falso) para las que no cumplan la condición.

UNKNOWN (desconocido) cuando al evaluar no se obtenga un resultado true o false.

Los predicados se aplican a filas de tablas relacionales, de forma que nos quedaremos con aquellas filas en las que la evaluación del predicado sea VERDADERO. O dicho de otra forma filtraremos por el predicado y descartamos aquellas que no devuelven un resultado VERDADERO para el predicado.

Las filas obtenidas después de aplicar el filtro se emplearán para la sentencia en la que se encuentra contenida dicho predicado.

Para formar predicados simples utilizaremos los operadores lógicos mientras que para formar predicados compuestos utilizaremos AND, OR o NOT.

Como ya veremos más adelante, los predicados se utilizan en la condición de búsqueda de las cláusulas WHERE y HAVING, y en las condiciones de combinación de las cláusulas FROM.

Clasificamos los predicados por el operador lógico que utilizan:

a) Predicados de comparación: Predicados que uilizan los operadores lógicos básicos (=, <, <=, <, >=, <>).
b) BETWEEN
c) IN
d) LIKE, RLIKE, REGEXP, REGEXP BINARY
e) IS NULL
f) Predicados cuantificados (SOME, ANY, ALL)

En este apartado veremos los predicados de comparación básicos. A medida que avancemos en la Unidad, estudiaremos el resto.

Por su simplicidad, la mejor forma de ver los predicados de comparación es mediante ejemplos:

a < b.

8<=10

‘1/2/2009’ > ‘31/12/2009’

edad_empleado >= 18 and edad_empleado <=65

fec_entrega <= fec_prev_entrega and (director=101 or director =102)

fec_entrega <> fec_prev_entrega

6.1.3 Modos de utilización de MySQL Server. SQL-Cliente. Los procedimientos almacenados.

Dependiendo de la forma en que ejecutemos las sentencias SQL existen distintos modos de utilización:

A. SQL Interactivo o ejecución directa

Las órdenes se escriben desde una línea de comandos (o desde un editor en entorno gráfico). Vemos los resultados inmediatamente, pero los comandos no quedan almacenados.

Ejemplos de este modo de utilización de SQL son el **"Editor SQL" de "MySQL WorkBench"**, el cliente en línea-comando de MySql o el editor de consultas de cualquier otra herramienta.

B. SQL Incrustado o embebido

Las sentencias forman parte del código de un lenguaje de programación, a este lenguaje se le llama "Anfitrión".

Las sentencias SQL tendrán que ir delimitadas por caracteres que sirvan al compilador para detectar que no son sentencias propias. Aunque existen algunos lenguajes (de 4ª generación) que no necesitan que las sentencias SQL se escriban delimitadas, sino que quedan completamente embebidas en el resto del código.

Los lenguajes anfitriones tendrán que incorporar los precompiladores SQL propios del SGBD que se esté utilizando o las instrucciones de acceso a dicho SGBD.

C. Módulos SQL-Cliente o procedimientos almacenados

Un módulo SQL-cliente es un objeto SQL que se hace para ser utilizado en un lenguaje anfitrión, pero que tiene la característica de que se compila a parte del código de dicho lenguaje.

Los módulos SQL-Cliente se asocian a una unidad de compilación en tiempo de ejecución, de modo que un único módulo puede estar asociado a varias unidades de compilación.

El mecanismo de asociación entre módulos, los lenguajes a los que puede asociarse un módulo y la transferencia de control de unos y otros son definidos por el fabricante.

En MySQL Server estos módulos cliente son los Procedimientos Almacenados y las Funciones.

Ambos se pueden utilizar como medio de guardar consultas (aunque no es su función).

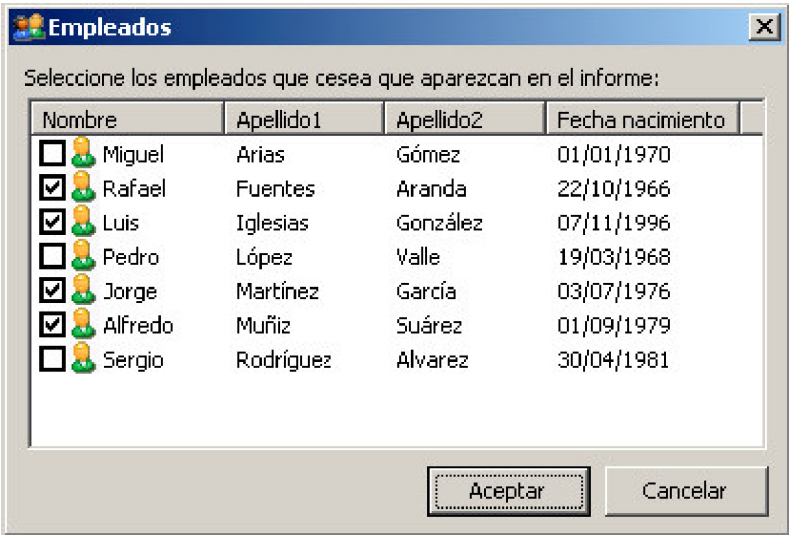
Tanto los procedimientos almacenados como las funciones se pueden ejecutar dentro del entorno de desarrollo de BD o bien mediante llamadas a los mismos desde una aplicación externa que acceda a nuestra base de datos o también desde la línea de comando del SO.

D. SQL Dinámico

En algunos casos, cuando se desarrolla el SI, no se sabe que sentencia SQL se va a ejecutar, sino que dicha sentencia se elabora en tiempo de ejecución, a esta forma de SQL se conoce como SQL dinámico.

Un ejemplo sería el siguiente:

Supongamos que tenemos una tabla de "Empleados" con cientos varios de registros, y una aplicación de gestión donde el usuario puede hacer altas, bajas y modificaciones de estos datos, así como generar informes basados en dichos datos.



En un formulario el usuario elige los empleados que desea que aparezca en un determinado informe, e incluso podría elegir los datos que quiere que aparezcan sobre cada empleado.

Será en tiempo de ejecución cuando se pueda elaborar la sentencia que determine que columnas y que registros se van a seleccionar para dicho informe.

El SQL dinámico es una utilidad que da mucha potencia a las aplicaciones, pero no hay que abusar de él, ya que puede resultar poco eficiente.

6.1.3.1 **Los procedimientos almacenados.**

Un **procedimiento almacenado** es un objeto perteneciente a una base de datos, que contiene un conjunto de instrucciones SQL, tanto de manipulación de datos, como de control de la secuencia de programa, asociadas a un nombre, y que son ejecutadas en conjunto. Puede contener parámetros tanto de entrada como de salida o de entrada/salida(parámetros pasados por referencia), así como (en algunas herramientas como MS SQL Server) devolver un valor de retorno.

Son precompilados al ejecutarse por primera vez, y no vuelven a compilarse cada vez que se ejecutan, lo que proporciona una mejora en el rendimiento. De todas formas, si se desea, se pueden volver a compilar, por ejemplo, si se ha producido una modificación en los datos que trata que afecte al procedimiento.

Una de las principales ventajas de este tipo de objetos, es que al residir en la propia base de datos son compartibles con todos los usuarios con permisos sobre ella. Además, al ser código externo a la aplicación (al programa), puede ser alterado sin que exista siempre la necesidad de modificar el código de la misma.

Al ser objetos de la base de datos, se hallan sujetos a los esquemas de seguridad determinados por el administrador de dicha base de datos.

Existen varios tipos de procedimientos almacenados, entre ellos se encuentran los **procedimientos almacenados de sistema**, que son herramientas para la realización de distintas tareas de administración. (Veremos algunos de ellos en la unidad de Administración).

Sintaxis básica de Procedimiento almacenado en MySQL:

CREATE PROC EDURE *nombre_procedimiento*
(<lista_de_parametros>)
[<caracteristicas_procedimiento>]
BEGIN
(<Sentencias SQL>)
END

Donde:

nombre_de_procedimiento:
Nombre que identifica al objeto, con el que después podremos ejecutarlo y que debe ser único para la misma base de datos.

<lista_de_parametros>::=
Lista de parámetros separados por comas definidos en el procedimiento con la siguiente sisntaxis:
[IN | OUT | INOUT] nom_parametro <tipo_dato> [=valor_def]
Valor_def es un posible valor de la variable antes de que le sea asignado otro valor durante la ejecución del procedimiento.
[IN | OUT | INOUT] determina que el parámetro se sea de entrada, salida o de entrada-salida.

<tipo_dato>::= Cualquiera de los tipos de datos soportados por MySQL

<Sentencias SQL>::=
El cuerpo del procedimiento puede estar formado por cualquier tipo de sentencia SQL.
Se puede anidar procedimientos almacenados, es decir, podemos llamar a un procedimiento dentro de otro.

Nota.- En MySQL es conveniente cambiar el delimitador de sentencia antes de crear un procedimiento para que el sistema entienda que el procedimiento termina cuando aparezca nuestro delimitador nuevo, mientras que el ";" delimitará cada sentencia que se encuentre dentro del procedimiento. No podemos olvidar volver a cambiar el delimitador al terminar el procedimiento. Veamos un ejemplo:

```
DELIMITER $$
CREATE PROCEDURE lee_clientes_de_ciudad(IN codpostal char(4))

BEGIN
    SELECT 'Los clientes que viven en la ciudad con código postal: ');
    SELECT codpostal;
    SELECT *
    FROM clientes
    WHERE cod_postal = codpostal;
END $$

DELIMITER ;
```

Para ejecutar el procedimiento anterior:

```
call lee_clientes_de_ciudad('29680');
```

Veamos un ahora un ejemplo con parámetros de salida:

```
DELIMITER $$
CREATE PROCEDURE num_clientes_de_ciudad(IN codpostal char(4),
                                        OUT num_clientes int)

BEGIN
    SELECT num_clientes = count(*)
    FROM clientes
    WHERE cod_postal = codpostal;
END $$

DELIMITER ;
```

Para ejecutar el procedimiento anterior:

```
call num_clientes_de_ciudad('29680', @nclientes);
SELECT 'El número de clientes de Estepona es: ';
SELECT @nclientes;
```

En el siguiente apartado estudiaremos las funciones, entre las que se encuentran las funciones que pueden crear los usuarios.

6.1.4 Funciones. Funciones de usuario. Funciones de agregado y cláusula GROUP BY de la sentencia SELECT.

Las funciones son elementos que toman cero, uno o más valores de entrada, realizan una o varias operaciones (sobre estos parámetros si los hay) y devuelven un resultado. En MySQL, como en la mayoría de las herramientas de Gestión de Bases de Datos, existen varios tipos de funciones. que iremos viendo a lo largo de la unidad:

Funciones de usuario.

Funciones del Sistema Gestor de Bases de Datos,

Funciones de agregado.

6.1.4.1 **Funciones de usuario**

Se trata de un grupo de funciones que son creadas por los usuarios informáticos, esto es, rutinas precompiladas, parecidas a los procedimientos almacenados pero que devuelven un valor.

Como todas las funciones se trata de rutinas que toman cero, uno o más valores de entrada (parámetros), realizan una serie de operaciones y devuelven un resultado.

Sintaxis básica de Funciones en MySQL:

```
CREATE FUNCTION nombre_funcion
    (<lista_de_parametros>)
    RETURNS <tipo_dato>
    [<caracteristicas_funcion>]
    BEGIN
        (<Sentencias SQL>)
    END
```

Donde:
nombre_de_funcion:
Nombre que identifica al objeto, con el que después podremos ejecutarlo y que debe ser único para la misma base de datos.
<lista_de_parametros>::=
Lista de parámetros separados por comas definidos en la función con la siguiente sisntaxis:
nom_parametro <tipo_dato> [=valor_def]
Valor_def es un posible valor de la variable antes de que le sea asignado otro valor durante la ejecución del procedimiento.
En las funciones todos los parámetros son de entrada.
<tipo_dato>::= Cualquiera de los tipos de datos soportados por MySQL
<Sentencias SQL>::=
El cuerpo de la función puede estar formado por cualquier tipo de sentencia SQL
Se pueden anidar funciones, es decir, podemos llamar a una función dentro de otra.
El cuerpo de la función debe incluir una sentencia "RETURN expresion", donde "expresion" será del tipo devuelto por la función.
Igual que en los procedimientos es recomendable cambiar el delimitador.

Veamos un ejemplo:

```
DELIMITER $$
CREATE FUNCTION num_clientes_de_ciudad(codpostal char(4))
RETURNS int
BEGIN
        DECLARE contador int;
        SELECT contador = count(*)
        FROM clientes
        WHERE cod_postal = codpostal;
        RETURN contador;
END $$

DELIMITER ;
```

Para obtener el número de clientes de una ciudad:

```
SELECT 'El número de clientes de Estepona es:';
SELECT num_clientes_de_ciudad('29680');
```

5.6.2 **Funciones de agregado o de agregación**

Las funciones de agregado realizan cálculos sobre los valores de un campo para un conjunto de registros y devuelven un solo valor.

Las funciones de agregado **sólo se aceptan** como expresiones en la sentencia SELECT en:

La cláusula SELECT.

La cláusula HAVING.

La cláusula GROUP BY

A diferencia del resto de funciones MySQL, las funciones de agregado **no se pueden anidar**.

Las funciones de agregación en MySQL son:

AVG, **COUNT**, GROUP_CONCAT, **MAX**, **MIN**, **SUM**, STD, STDDEV, ..., VAR_POP, VARSMAP, VARIANCE

Con la excepción de COUNT, las funciones de agregado omiten los valores NULL, es decir, no los tienen en cuenta para hacer los cálculos.

COUNT es la única función de agregado que se puede utilizar con el símbolo "*", ya que COUNT cuenta filas, nos daría igual que columna contar.

Veamos un ejemplo:

EMPLEADOS				
numem	**nombre**	**tlfcont**	**depto**	**salario**
1	Isabel Fernández	…….	1	1200
2	José Sánchez	…….	2	1435
3	Mª López	…...	2	1200
4	Jorge Quirós	…...	1	2000
5	Justo Gómez	…...	1	1200

Para el conjunto de registros "EMPLEADOS" si quisiéramos saber cuantos empleados hay, el salario máximo, el salario mínimo y lo que nos cuesta los salarios de nuestros empleados podríamos ejecutar la siguiente sentencia SELECT:

```
SELECT count(*) AS num_empleados, max(salario) AS salario_max, min(salario) AS salario_min,
       sum(salario) AS total_salarios
FROM EMPLEADOS
```

El resultado sería:

num_empleados	salario_max	salario_min	total_salarios
5	2000	1200	7035

Para nuestro ejemplo

count(*) = count(numem)

Con las funciones de agregado podemos utilizar el operador DISTINCT, de forma que se tengan en cuenta solo valores diferentes o no. Para el ejemplo anterior:

COUNT (DISTINCT depto) = 2
COUNT(depto) = 5

Las funciones de agregado se suelen utilizar junto a la cláusula GROUP BY de la sentencia SELECT. Vamos a verla a continuación.

Cláusula GROUP BY de la sentencia SELECT.

La cláusula GROUP BY se utiliza para formar grupos con las filas de la tabla por el valor de una o varias columnas, de forma que todas las filas que coincidan en el valor de la columna o columnas de la cláusula GROUP BY estarán en el mismo grupo.
Una vez hechos los grupos, las operaciones que hagamos mediante funciones agregado, devolverán un valor para cada grupo.

Para el ejemplo anterior, si quisiéramos obtener los mismos resultados pero, en lugar de para toda la empresa, para cada departamento podríamos ejecutar las siguiente sentencia SELECT:

```
SELECT depto, count(*) AS num_empleados, max(salario) AS salario_max, min(salario) AS
       salario_min, sum(salario) AS total_salarios
FROM EMPLEADOS
GROUP BY depto
```

El resultado será:

depto	num_empleados	salario_max	salario_min	total_salarios
1	3	2000	1200	4400
2	2	1435	1200	2635

Nota.- Hemos hecho un grupo para cada departamento y después hemos hecho los cálculos para cada grupo

En el orden de ejecución, la cláusula GROUP BY se ejecuta después de la cláusula WHERE. Esto es, una vez filtradas las filas y descartadas las que no interesan, se hacen los grupos para las que cumplen las condiciones de WHERE.

Sintaxis

<CLAUSULA_GROUP_BY> ::= GROUP BY <expresion_*group_by*> *[ASC | DESC]*[**,***...n*]
<expresion_*group_by*> *::= campo o expresión por los que queremos hacer los grupos*

6.1.5 Predicados LIKE, BETWEEN, IS NULL. Uso de comodines.

Como veíamos anteriormente, los predicados son expresiones contenidas en sentencias u órdenes SQL y que devuelven un valor booleano (verdadero, falso o desconocido).

Recordemos también que cuando aplicamos un predicado a un conjunto de registros nos quedaremos solo con aquellos para los que el resultado del predicado sea verdadero.

En este apartado vamos a estudiar otro grupo de predicados:

Predicado BETWEEN

A [NOT] BETWEEN B AND C Será cierto si A está entre el rango de valores comprendido entre B y C.

Predicado LIKE | RLIKE | REGEXP | REGEXP BINARY

El significado de estos predicados es el mismo, solo que dependiendo del predicado que utilicemos podremos usar unos comodines u otros.

A [NOT] {LIKE | RLIKE | REGEXP | REGEXP BINARY} B Será cierto cuando A sea igual al patrón representado por B, en B podremos utilizar comodines. En la siguiente tabla se representan los comodines que podemos usar en MySQL y el operador con el que podemos usarlo:

REGEXP BINARY es como REGEXP pero distinguiendo entre mayúsculas y minúsculas.

Carácter comodín	Descripción	USO
%	Cualquier cadena de cero o más caracteres.	LIKE
_ (subrayado)	Cualquier carácter individual	LIKE
[ ]	Cualquier carácter individual de intervalo ([a-f]) o del conjunto ([abcdef]) especificado.	RLIKE, REGEXP
[^]	Cualquier carácter individual que no se encuentre en el intervalo ([^a-f]) o el conjunto ([^abcdef]) especificado.	RLIKE, REGEXP
.	Cualquier carácter individual	RLIKE, REGEXP

*	Cualquier cadena de cero o más caracteres.	RLIKE, REGEXP
{n}	Varias repeticiones	RLIKE, REGEXP
^	La cadena comienza por lo que sigue al símbolo ^	RLIKE, REGEXP
$	La cadena termina por lo que antecede al símbolo $	RLIKE, REGEXP

Nota.- Para utilizar los caracteres comodín como literales se escriben detrás del carácter de escape de MySQL (\). Ej// 'La cadena contiene \%' ==> % no se interpreta como comodín.

Predicado IS NULL

A IS [NOT] NULL Será cierto cuando A sea nulo o, en el caso de utilizar NOT, no lo sea.

6.1.6 Uso de "SUBSELECT" en sentencias de manipulación de datos .(SELECT, INSERT, UPDATE y DELETE)

A veces, para trabajar con funciones y también con algunos tipos de predicados necesitamos ejecutar una subconsulta.

Una subconsulta es una consulta SELECT que está anidada en una instrucción SELECT, INSERT, UPDATE o DELETE, o dentro de otra subconsulta. Una subconsulta se puede utilizar en cualquier parte en la que se permita una expresión.

Una **subconsulta** puede **devolver un valor único o un conjunto de resultados**. Es de gran importancia tener esto en cuenta, ya que **de esto va a depender donde la podremos usar**.

Veamos un ejemplo de consulta SELECT con una subconsulta (SUBSELECT):

Ejemplo: Seleccionamos los productos cuyo precio coincida con el precio del pack de bolígrafos

```
SELECT DesProduc
FROM productos
WHERE precio =
    (SELECT precio
     FROM productos
         WHERE DesProduc = 'Pack 10 bolígrafos bic color azul')
```

Nota.- En este ejemplo el subselect debe devolver un solo resultado, en caso contrario la consulta daría un error.

6.1.7 Predicados IN, EXITS, SOME, ANY, ALL.

Recordemos que un predicado es una expresión que devuelve un valor verdadero, falso o desconocido. En los apartados 5.3.3 y 5.7 vimos los predicados básicos, pero existen algunos predicados más complejos en los que obtenemos el resultado comparando con el resultado de una subconsulta que, en este caso, nos devolverá un conjunto de resultados, pero de una sola columna.

a) IN

A [NOT] IN subselect Será cierto cuando A coincide (o no, si se utiliza NOT) con uno de los valores de una lista o bien de los resultados de un subselect. (Es decir A está en lista_subselect ó A no está en lista_subselect)

Ejemplo.- Averiguar que empleados cuyo salario es igual a la media de los salarios de algún departamento:

```
SELECT nomem, ape1em, ape2em, salarem
FROM  empleados
WHERE salarem IN (SELECT avg(salarem)
                  FROM empleados
                  GROUP BY numde)
```

b) EXISTS

Un predicado EXISTS está formado por EXISTS y una subconsulta, y será cierto cuando el resultado de la subconsulta contiene alguna fila, es decir existe alguna fila para la consulta.

Ejemplo: Busca el nombre de los empleados que son directores, si es que existe algún centro de trabajo en Málaga):
(Devolverá el nombre de todos los directores si existe algún centro de trabajo. Si no existen centros de trabajo en Málaga, no devolverá ningún director).

```
SELECT nomemp
FROM empleados JOIN deptos  ON empleados.numem=deptos.coddir
WHERE EXISTS
  (SELECT *
  FROM centros
  WHERE poblac_centro = 'Málaga')
```

c) Predicados cuantificados (SOME, ANY, ALL)

En estos predicados no se obtiene un único valor booleano sino un conjunto de valores de un atributo (campo de una tabla).

Sintaxis:

A {=|<|<=|>|>=|<>} {SOME| ANY | ALL} subselect

Los predicados cuantificados están formados por una expresión en la que se compara, mediante un operador lógico (= ó < ó <= …) y una expresión (A) con todos y cada uno de los resultados de una subconsulta (que puede devolver varios valores pero de una sola columna).

El resultado de dicha comparación dependerá del un cuantificador (some, any, all) que usemos:

Si utilizamos ALL, el resultado tiene que ser cierto para todos los valores obtenidos en la subconsulta.

Para SOME y ANY es suficiente con que se cumpla para alguno de ellos.

Ejemplo.- Obtener los nombres de los empleados cuyo salario supere al salario de todos los empleados del departamento 121

```
SELECT nomem
FROM empleados
WHERE salarem >= ALL (SELECT salarem
                      FROM empleados
                      WHERE numde = 121)
```

Ejemplo.- Obtener los nombres de los empleados cuyo salario supere al salario a alguno de los empleados del departamento 121

```
SELECT nomem
FROM empleados
WHERE salarem >= SOME (SELECT salarem
                       FROM empleados
                       WHERE numde = 121)
```

(Siempre podemos sustituir SOME por ANY).

6.1.8 Cláusula HAVING de la sentencia SELECT.

Para comprender mejor esta cláusula vamos a repasar todas las cláusulas de una sentencia SELECT:

< sentencia_select> ::=

<CLAUSULA_SELECT> se seleccionan los campos y se evalúan las expresiones sobre los campos o los grupos que queremos mostrar en el resultado (PROYECT de ALGEBRA) obtendremos una columna por cada campo o expresión de la cláusula. Si se especifica DISTINCT se eliminarán los resultados repetidos.

<CLAUSULA_FROM> se seleccionan las tablas relacionales que se muestran en esta cláusula (tablas relacionales y/o JOIN de ALGEBRA)

[<CLAUSULA_WHERE>] filtramos las filas que cumplan una condición (predicado) descartando las que no satisfagan dicho predicado.

[<CLAUSULA_GROUP_BY>] indicamos el campo o conjunto de campos por los que haremos grupos para obtener resultados sobre dichos grupos.

[<CLAUSULA_HAVING>] Filtramos los grupos que no cumplan una condición.

[<CLAUSULA_ORDER_BY>] Se ordenarán las filas resultantes por el atributo/s y/o expresión/es que se especifique en esta cláusula, de forma ascendente o descendentemente. Los atributos y expresiones de esta cláusula deben estar en la cláusula SELECT.

Recordemos también el orden de ejecución de las cláusulas de la sentencia SELECT, ya que es bastante importante para entender la necesidad de la cláusula HAVING:

1. CLÁUSULA FROM
2. CLÁUSULA WHERE (opcional)
3. CLÁUSULA GROUP BY (opcional)
4. CLÁUSULA HAVING (opcional)
5. CLÁUSULA SELECT
6. CLÁUSULA ORDER BY (opcional)

Recordemos también que en la cláusula WHERE no se podían utilizar funciones de agregado. Esto es porque cuando se ejecuta esta cláusula, todavía no se han hecho los grupos, por lo tanto no podemos aplicar funciones que obtienen datos sobre grupos.
Es por esta razón por la que cuando necesitamos incluir en la consulta alguna condición que afecte a los grupos, y que incluya funciones de agregado, tendremos que utilizar la cláusula HAVING.
En esta cláusula solo podremos incluir predicados relacionados con los grupos, ya que cuando se ejecuta ya están formados los grupos y, por tanto, no tiene sentido poner condiciones sobre filas simples.

La sintaxis de HAVING es:

<clausula_having>::= HAVING <predicado_grupo>

Ejemplo: Obtener el número de empleados de los departamentos del centro de trabajo 10, solo para aquellos departamentos cuyo salario medio sea 1000 € o más:

```
SELECT deptos.nomde, count(*)
FROM empleados join deptos on empleados.numde = deptos.numde
WHERE deptos.numce = 10
GROUP BY deptos.nomde
HAVING avg(empleados.salarem) >= 1000
```

6.1.9 Funciones MySQL. Búsqueda de funciones en el manual de referencia MySQL

Una función es un elemento que toma cero, uno o más valores de entrada y devuelven un valor escalar o un conjunto de valores en forma de tabla que es el resultado de realizar alguna o algunas operaciones sobre los datos de entrada.

Existen distintos tipos de funciones en MySQL: Funciones de sistema, funciones de agregado, funciones escalares, etc.

Algunas de ellas ya las hemos visto (funciones de agregado).

Recordemos que las funciones de agregado tienen la peculiaridad de que no se pueden utilizar en la cláusula WHERE de la sentencia SELECT, esto es lógico ya que cuando se ejecuta la cláusula WHERE todavía no se ha aplicado los grupos, por lo que no es posible averiguar valores relacionados con éstos. Esta característica no es aplicable al resto tipos de funciones.

Esto es, todas las funciones de MySQL, excepto de las funciones de agregado, pueden ir en cualquier expresión, incluidas las expresiones de la cláusula WHERE.

De la misma manera, todas las funciones, con la excepción de las funciones de agregado, pueden anidarse.

En este apartado veremos las funciones escalares y veremos como obtener información sobre cualquiera de ellas, debido a la gran cantidad de estas que existen.

Funciones escalares (o simplemente funciones):

Se trata de un tipo de función en el que el valor devuelto es escalar (no un conjunto de resultados)

Se clasifican por el tipo de datos de los valores con los que trabajan. Según esto, podremos encontrar:

Funciones numéricas:

ABS(expr) Devuelve valor absoluto de expr.

MOD(expr1, expr2) Devuelve el resto de dividir expr1 entre expr2.

SQRT(expr) Devuelve la raíz cuadrada de expr.

POW(X,Y) Devuelve X a la potencia de Y.

RAND() Devuelve un número aleatorio en coma flotante entre 0 y 1.0

ROUND(N, D) Devuelve el número N redondeando a D decimales (si D es negativo, redondeará a D posiciones a la izquierda de la coma)

…..

Funciones de cadena de caracteres:

SUBSTRING (expr, start, length) Devuelve la subcadena de expr, empezando en el carácter start y con el tamaño length.

UPPER (expr) o UCASE(expr) Convierte expr a mayúsculas.

LOWER (expr) o LCASE(expr) Convierte expr a minúsculas.

LENGTH (expr) Devuelve el tamaño de expr.

…..

Existen otros muchos tipos de funciones: de fecha y hora, de bit, de cifrado…

Uno de los tipos de funciones son las **"Funciones de información"** que nos devuelve información sobre el servidor, el estado del sistema, etc.

Entre estas funciones se encuentran funciones como USER(), CURRENT_USER(), DATABASE(), ROW_COUNT(), etc.

Función IFNULL(expr1, expr2)

Esta función es muy útil cuando queremos trabajar con datos que pueden tomar valores nulos y queremos evitarlos. Devolverá el valor de expr2 cuando expr1 sea nula, en otro caso devolverá expr1.

En el "Manual de referencia de MySQL" podremos encontrar todas las funciones que existen.

6.1.10 Diseño de los Esquemas Externos o Subesquemas. Las Vistas

Como ya hemos comentado en temas anteriores, en el modelo de referencia ANSI/SPARC, se definía un esquema externo en el que distintos usuarios y/o grupos de usuarios podían tener visiones diferentes de la misma base de datos.

Una de las herramientas con las que se consigue este objetivo es mediante el uso de Vistas.

Las VISTAS son objetos que dan una visión externa de los datos, esto es, nos sirven para ocultar la parte de los datos que no queremos que se muestre. Los usuarios que tienen acceso a una vista, pueden tratarla como si fuera una tabla, es decir, pueden ejecutar sentencias SELECT sobre estos objetos e incluso, siempre que se cumplan unas condiciones que lo permitan, también se podrán ejecutar sentencias de inserción, actualización y borrado de datos.

Los datos no están duplicados, las vistas acceden a las tablas en las que realmente se encuentran dichos datos.

La sintaxis de esta sentencia es la siguiente:

CREATE VIEW nom_vista

[(<lista campos>)]

AS

<expresión_query>

Opcionalmente, si queremos que los nombres de las columnas de las vistas sean otros que los de las tablas desde las que vamos a obtener los datos, en <lista_campos> pondremos los nombres de las columnas de nuestra vista separados por el símbolo ','.

Nota.- Ver la sintaxis de <expresión_query> en la sentencia SELECT.

Una vista será actualizable si:

El usuario que accede a ella tiene privilegios para actualizar las tablas.
La tabla o tablas a las que accede la vista no tiene/n columnas que no puedan contener valores nulos y/o que no tengan definido un valor por defecto inaccesibles desde la vista.
No contiene valores de agrupación.
En general, cuando actualizar los datos en la vista no provoca inconsistencias en los datos.

6.1.11 Bibliografía

Libros en pantalla de Microsoft SQL Server 2008.

Manual de referencia MySQL 5.0 (http://dev.mysql.com/doc/refman/5.0/es/index.html).

References Manual MySQL 5.1 (http://dev.mysql.com/doc/refman/5.1/es/index.html).

References Manual MySQL 5.5 (http://dev.mysql.com/doc/refman/5.5/es/index.html).

References Manual MySQL 5.6 (http://dev.mysql.com/doc/refman/5.6/es/index.html).