INTRODUCCIÓN A MYSQL

1ºDAW
David González Sanz

MySQL

Primeros pasos

1) Conectar/desconectar del servidor

MySQL [-h nombre_host] -u nombre_usuario -p

MySQL -u root -p

exit o quit

dentro del servidor todas las órdenes terminan en ;

2) Podemos ejecutar comandos del SO anfitrión con

system comando_so

3) El Diccionario de Datos

Son los metadatos del SGBD. En MySQL el diccionario está compuesto de 4 Bases de Datos, que son:

- **informattion_schema**: Información sobre las BBDD, las tablas, las vistas, etc.
- **perfomance_schema**: Contiene información sobre el rendimiento del servidor.
- **MySQL**: Contiene información sobre parámetros del SGBD, usuarios y privilegios.
- **sys**: Contiene información sobre procedimientos y funciones almacenadas.

4) Cada usuario tendrá una visión del SGBD en función de los permisos que tenga. El usuario root tiene todos los permisos y, por lo tanto, acceso a toda la información.

5) Comando para ver las BBDD a las que tiene acceso el usuario.

show databases;

6) Abrir una BD

use nombre_bd;

No existe un comando para cerrar una BD, simplemente al abrir otra, cambia.

7) Ver todas las tablas o relaciones de una BD

show tables;

8) Ver la descripción y algunas propiedades de una tabla o relación

describe nombre_tabla;

9) Podemos obtener información genérica como:

usuario conectado, versión, fecha y hora actuales, etc.

- **select user();**
- **version();**
- **current_date;**
- **current_time;**

Lenguajes SGBD

10) El lenguaje SQL contiene los tres lenguajes del SGBD

-DDL (Lenguaje de definición de Datos): Es el lenguaje que se utiliza para la creación para la creación y gestión de objetos en la BD.

Sus tres comandos o verbos principales son:

a) **CREATE**: Para la creación de objetos.

b) **DROP**: Para el borrado de objetos.

c) **ALTER**: Para modificar objetos

-DCL: Es el lenguaje para la gestión de permisos sobre objetos del sistema

Sus comandos principales son:

a) **GRANT**: Para otorgar permisos o privilegios

b) **REVOKE**: Para denegarlos.

-DML: Es el lenguaje que se utiliza para la gestión de los datos.

Sus comandos principales son:

a) **INSERT**: Para añadir datos.

b) **DELETE**: Para borrar datos.

c) **UPDATE**: Para modificar datos.

d) **SELECT**: Para seleccionar datos.

Lenguaje de definición de datos (DDL)

Creación de usuarios y BD

Crear y borrar una BD

```
CREATE DATABASE nombre_bd;
```

```
DROP DATABASE nombre_bd;
```

Crear un usuario. Puede hacerse con un usuario que tenga permiso para ello (root)

```
CREATE USER nombre_usuario IDENTIFIED BY 'contraseña';
```

```
CREATE USER david IDENTIFIED BY 'david';
```

Borrar un usuario.

```
DROP USER nombre_usuario;
```

Creación de tablas o relaciones.

En primer lugar, debemos conocer los tipos de datos disponibles en MySQL. Podemos consultar el manual o webs como w3schools.

a) Tipos de datos caracteres

- **CHAR**: Para un solo carácter.
- **VARCHAR (longitud)**: Son cadenas de caracteres de hasta 65535 caracteres.
- **ENUM** ('valor1', 'valor2', ..., ' valorN'): Solo permite introducir uno de los valores que están entre paréntesis.

b) Tipos de datos numéricos

- **INTEGER**: Números enteros entre -2147483648 a 2147483647.
- **DECIMAL (tamaño, decimales)**: Tamaño indica el nº total de dígitos del número y decimales, el número de decimales.

c)Tipos de datos tiempo

- **DATE**: Para las fechas con formato YYYY-MM-DD.
- **TIME**: Para horas con el formato hh:mm:ss
- **TIMESTAMP**: Indica un instante y está compuesto por la fecha y la hora con el formato YYYY-MM-DD hh:mm:ss

d)Datos de tipo lógico o booleano

- **BOOLEAN**: En los que 0 equivale a FALSO y 1 equivale a VERDADERO

Crear una nueva relación o tabla. El comando genérico seria:

```
(
 nombre_atributo1 tipo_dato [restricción semántica],
 nombre_atributo2 tipo_dato [restricción semántica],
 .....,
 nombre_atributoN tipo_dato [restricción semántica]
);
```

Restricciones semánticas (en SQL se llaman CONSTRAINTS)

- PRIMARY KEY

- NOT NULL

- UNIQUE

- CHECK

- FOREIGN KEY

Modificar una relación o tabla existente

a) Añadir un nuevo atributo a la tabla

ALTER TABLE nombre_tabla **ADD** nombre_atributo tipo_dato [restricción semántica]

b) Borrar un atributo existente

ALTER TABLE nombre_tabla **DROP** nombre_atributo;

c) Modificar un atributo existente

ALTER TABLE nombre_tabla **MODIFY** nombre_atributo tipo_dato [restricción semántica];

Borrar una tabla o relación:

DROP TABLE nombre_tabla;

Existen restricciones que pueden indicarse en la misma línea de declaración del atributo y otras que requieren su definición mediante un CONSTRAINT.

Todo CONSTRAINT tiene el mismo esquema:

CONSTRAINT nombre_del_constraint cuerpo_del_constraint

(NO PUEDE HABER DOS CONSTRAINT CON EL MISMO NOMBRE EN LA MISMA BD)

Situaciones en las que debemos usar CONSTRAINTS:

a) Cuando la PK de una relación está formada por más de un atributo

CONSTRAINT pk_nombretabla

PRIMARY KEY (at1, at2...atN)

b) Para la realización de chequeos o validaciones

CONSTRAINT ck_nombreatributo_tabla

CHECK (condición o predicado)

c) Para declarar una FK

CONSTRAINT fk_tablaactual_tablareferencia

FOREIGN KEY (at1, at2...atN)

REFERENCES nombretablareferenciada (at1,at2...atN)

[ON DELETE NO ACTION | CASCADE | SET NULL | SET DEFAULT]

[ON UPDATE NO ACTION | CASCADE | SET NULL | SET DEFAULT]

Ejemplo 2: Creación de la tabla vivienda

```
CREATE TABLE vivienda (
 calle VARCHAR(30),
 numero INTEGER,
 extensión INTEGER NOT NULL,
 dormitorios INTEGER NOT NULL,
 CONSTRAINT pk_vivienda
 PRIMARY KEY (calle, numero)
);
```

Ejemplo 3: Vamos a modificar la tabla 'vivienda' añadiendo 2 chequeos (que la extensión sea mayor que 0 y que el número de dormitorios superior a 1)

```
ALTER TABLE vivienda ADD CONSTRAINT ck_extension CHECK(extensión>0);
ALTER TABLE vivienda ADD CONSTRAINT ck_dormitorios CHECK(dormitorios>1);
```

Ejemplo 4: Crear una tabla 'nomina' relacionada con la tabla 'empleado'

```
CREATE TABLE nomina (
 dni_empleado VARCHAR(9)
 mes ENUM('ENE','FEB','MAR','ABR','MAY','JUN','JUL','AGO','SEP','OCT','NOV','DIC'),
 agno INTEGER,
 total DECIMAL(7,2) NOT NULL,
 CONSTRAINT pk_nomina
 PRIMARY KEY(dni_empleado, mes, agno),
 CONSTRAINT fk_nomina_empleado
 FOREIGN KEY(dni_empleado)
 REFERENCES empleado(dni)
 ON DELETE CASCADE
 ON UPDATE CASCADE
);
```

*****UN CASO ESPECIAL: La declaración de una clave alterna o alternativa*****

Las claves alternas son claves candidatas que no han sido elegidas como PK.

Llevan asociada las restricciones semánticas UNIQUE y NOT NULL

Ejemplo 5: añadir una clave alterna a la tabla 'vivienda' llamada 'ref_catastral'

```
ALTER TABLE vivienda ADD ref_catastral VARCHAR(15) UNIQUE NOT NULL;
```

SCRIPT SQL

Un script SQL es un fichero de texto plano (codificación UTF-8) que guardamos con las extensión .SQL

Se pueden crear con editores sencillos como:

-En Linux: Gedit, Nano, Vi

-En Windows: Notepad, Bloc de notas, Notepad++

Una vez creado el script el comando para su ejecución es:

```
SOURCE ruta script.sql;
```

Ejemplo 6: A partir del modelo lógico relacional de perfumería, crear un script SQL con:

-Una base de datos que se llame 'perfumeria'

-Abrir la base de datos perfumeria

-Crear las tablas que aparecen en el esquema

(Hecho en MySQL ya)

Clausula IF NOT EXISTS

Se utiliza para evitar la creación de objetos que ya hayan sido previamente creados.

La sintaxis es:

```
CREATE tipo_objeto IF NOT EXISTS nombre_objeto resto_de_opciones_de_objeto
```

Para una tabla:

```
CREATE TABLE IF NOT EXISTS nomina (
 dni_empleado VARCHAR(9),
 mes INTEGER,
 agno INTEGER,
 total DECIMAL(7,2) NOT NULL,
 CONSTRAINT pk_nomina
 PRIMARY KEY(dni_empleado,mes,agno),
 CONSTRAINT fk_nomina_empleado
 FOREIGN KEY(dni_empleado)
```

```
REFERENCES empleado(dni)
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT ck_mes
CHECK(mes>=1 and mes <=12)
);
```

Para una base de datos:

```
CREATE DATABASE IF NOT EXISTS nombre_base
```

USUARIOS DE MYSQL

Los nombres de los usuarios de MySQL se componen de dos partes:

- nombre_usuario@nombre_o_ip_del_host

El nombre o ip del host indica desde donde se puede conectar el usuario, si no se indica nada, el usuario puede conectarse desde cualquier ip.

En nuestro caso hemos deshabilitado el acceso remoto de manera que solo nos conectamos desde modo local,

por lo tanto, no indicaremos host a la hora de crear un usuario, lo haremos:

```
CREATE USER IF NOT EXISTS nombre usuario IDENTIFIED BY 'contraseña'
```

LENGUAJE DE CONTROL DE DATOS (DCL)

Es el lenguaje que se utiliza para la gestión de permisos y sus dos verbos principales son:

- **GRANT** para otorgar permisos

- **REMOVE** para retirar permisos

En primer lugar, debemos distinguir entre:

- Privilegios: Se trata de un permiso en un ámbito concreto

- Roles: Contienen un conjunto de permisos asociados a un tipo de usuario. Existen roles predefinidos pero también podemos crear nuestros propios roles.

PERMISOS O PRIVILEGIOS

-Permisos para crear usuarios

-Permisos a nivel global

-Permisos a nivel de BD

-Permisos a nivel de tabla

-Permisos a nivel de columna

-Permisos a nivel de procedimientos y menciones almacenados

ROLES

Tenemos una serie de roles predefinidos:

-**DBA**: equivaldría a convertirlo en root.

-**UserAdmin**: Permite la gestión completa de usuarios.

-**DBManager**: Otorga todos los privilegios sobre todas las BD del sistema.

También es posible la creación de roles personalizados

VERBO GRANT

- **GRANT** privilegios_o_roles **ON** objetos **TO** usuarios [**WITH GRANT OPTION**]

La cláusula **WITH GRANT OPTION**, si se indica, significa que el usuario que recibe los roles o privilegios puede, a su vez, otorgarlos a otros usuarios.

Nosotros vamos a trabajar exclusivamente con:

-Privilegios DML: **INSERT**, **DELETE**, **UPDATE**, y **SELECT**

-Todos los privilegios: **ALL PRIVILEGES**

Ejemplo 7: Supongamos que tenemos una BD llamada 'mibd' con tres tablas {tabla1 tabla2 tabla3}

Supongamos que tenemos dos usuarios llamados "miusuario" y "usuario2"

- Otorgar todos los privilegios al usuario "miusuario" sobre todas las tablas de la BD "mibd"

```
GRANT ALL PRIVILEGES ON midb.* TO miusuario;
```

- Otorgar todos los privilegios al usuario "miusuario" sobre todas las tablas de todas las BD del servidor

```
GRANT ALL PRIVILEGES ON *.* TO miusuario;
```

- Otorgar al usuario "usuario2" privilegio de consulta (solo consulta) sobre la tabla1, y sobre la tabla2 y tabla3 los de consulta e inserción de datos (no puedes ni borrar ni modificar)

```
GRANT SELECT ON mibd.tabla1 TO usuario2;
GRANT SELECT, INSERT ON mibd.tabla2 TO usuario2;
GRANT SELECT, INSERT ON mibd.tabla3 TO usuario2;
```

Cada vez que otorguemos o revoquemos privilegios tenemos que confirmarlo con la orden:

```
FLUSH PRIVILEGES;
```

VERBO REVOKE

REVOKE privilegios o roles, [GRANT OPTION] **ON** objetos **FROM** usuario;

PROCESO PARA LA CREACION DE UNA BD

1.- Creamos una BD

2.- Creamos un usuario para la BD

3.- Otorgamos a este usuario todos los privilegios sobre esa BD

4.- Grabamos los privilegios (FLUSH PRIVILEGES)

5.- Ejecutamos el script de creación de las tablas de BD

Ejercicio 8: A partir del modelo relacional ciclismo crear script SQL con:

-Una BD llamada ciclismo

-Un usuario llamado "userciclismo", "operaciclismo" con contraseña igual

-userciclismo tendrá todos los privilegios sobre todas las tablas

-operaciclismo solo tendrá privilegios de consulta sobre todas las tablas.

LENGUAJE DE MANIPULACION DE DATOS (DML)

Es el lenguaje que se utiliza para la gestión de datos

Verbo INSERT (añadir datos)

INSERT INTO nombre_tabla **VALUES**(lista_de_valores_ordenada);

La secuencia anterior inserta una fila en la tabla. La lista de valores debe ir en el mismo orden que los atributos definidos en la tabla.

Los valores se introducen de la siguiente forma:

-**VARCHAR**: entre comillas simples (‘ ‘)

-**INTEGER**: sin comillas y usaremos . para los decimales

-**DATE**: entre comillas simples, siguiendo el formato de cada uno. (‘YYYY’ ‘MM’ ‘DD’)

También es posible la inserción masiva de datos, es decir, insertar varias filas en una única sentencia INSERT

INSERT nombre_tabla (at1, at2,… atN) **VALUES**

(valor1, valor2,…, valorN),

(valor1, valor2,…, valorN),

(valor1, valor2,…, valorN),

……

(valor1, valor2,…, valorN);

La inserción de datos, como otras operaciones DML, debe tener en cuenta una serie de consideraciones:

- Los valores deben corresponder en tipo a los definidos en la tabla.

- El orden de la inserción debe respetar el modelo lógico, como ocurría en la creación.

- Además los valores introducidos en las tablas tienen que satisfacer obligatoriamente las restricciones de integridad del modelo.

Ejercicio 9: Entrar al servidor con userciclismo y ejecutar script de inserción de datos para que haya al menos 2 filas por tabla.

Verbo DELETE (Borrar datos)

DELETE FROM nombre_tabla [**WHERE** condición p predicado];

Si no se indica la cláusula **WHERE**, se borrarán todos los datos.

DELETE FROM tabla1; (borraría todos los datos de la tabla1)

TRUNCATE TABLE tabla1; (borraría todos los datos de la tabla1)

La clausula WHERE en general es:

WHERE predicado simple o complejo.

Verbo UPDATE (Modificar datos)

UPDATE nombre_tabla **SET** at1=valor1 [,at2=valor2…] **WHERE** condición;

UPDATE nombre_tabla **SET** at1=valor1; (Modifica el valor del at1 en todas las filas de la tabla)

UPDATE nombre_tabla **SET** at1 =valor1 **WHERE** condición; (añadir, borrar o modificar)

Verbo SELECT (Consulta de datos)

1.-Consultas básicas sobre una única tabla

SELECT operador_proyección

FROM nombre_tabla

[**WHERE** condición] (operador selección)

[**ORDER BY** nombre atributo];

Operadores

- Operador proyección: Sirve para indicar las columnas de las tablas que queremos obtener.

- *: Obtenemos todas las columnas de la tabla.

- at1 [,at2,…,atN]

- FROM: Indica la tabla desde la que queremos consultar.

Ej1: Obtener todos los datos de una tabla

SELECT * **FROM** temple;

Ej2: Obtener algunos datos de una tabla

SELECT nomem, fecna **FROM** temple;

Ej3: Uso de alias. Podemos renombrar las columnas de la tabla para la salida

- Si el alias está formado por una única palabra:

SELECT nomem AS Nombre, fecna AS Nacimiento **FROM** temple;

- Si el alias está formado por más de una palabra:

SELECT nomem AS 'Nombre del empleado', fecna AS 'Fecha de nacimiento' **FROM** temple;

- Operador selección: Devuelve las filas que cumplen la condición.

- Predicado simple: Expresión con un operador relacional o de comparación.

SELECT * **FROM** temple **WHERE** numde=100;

- Predicado compuesto: Unen más de una condición con operadores lógicos (AND, OR).

SELECT * **FROM** temple **WHERE** numde=111 **AND** numhi>1;

- ORDER BY: Ordena el resultado.

- Orden ascendente.

SELECT * **FROM** temple **WHERE** numde=111 **ORDER BY** numhi;

- Orden descendente.

SELECT * **FROM** temple **WHERE** numde=111 **ORDER BY** numhi **DESC**;

- Clausula DISTINCT: Se utiliza en la proyección para eliminar filas repetidas.

SELECT numde, numhi **FROM** temple **WHERE** numhi=0 **ORDER BY** numde;

SELECT DISTINCT numde FROM temple WHERE numhi=0;

- Expresiones: Podemos proyectar atributos de una tabla o expresiones basadas en esa tabla. Normalmente cuando vamos a expresar una expresión, usaremos un alias.

SELECT numde, numem, nomem, salar, salar*0.1 as 'Paga extra' FROM temple WHERE numde=111 ORDER BY numem;

SELECT numde, numem, nomem, salar, (numhi -1) *salar*0.1 as 'Paga extra' FROM temple WHERE numde=111 and numhi>0 ORDER BY numem;

- Predicado NULL:

- Expresión IS (NOT) NULL

SELECT * FROM temple WHERE comis IS NOT NULL;

- Predicado BETWEEN:

expresión1(NOT) BETWEEN expresion2 AND expresión 3;

SELECT * FROM temple WHERE salar BETWEEN 1500 AND 2000 ORDER BY salar;

- Predicado IN: Comprueba si la expresión es igual, o no, a alguno de los valores de la lista.

expresión [NOT] IN (lista de valores);

SELECT * FROM temple WHERE numde IN (100,110,111) ORDER BY numde;

- Predicado LIKE: Permite el uso de metacaracteres

- Metacaracter (_): Equivale a un único carácter.

- Metacaracter (%): Equivale a una cadena de caracteres.

SELECT * FROM temple WHERE nomem LIKE '%,A%;

Expresión_alfanumérica [NOT] LIKE cadena_caracteres;

SELECT * FROM temple WHERE numem='LOPEZ, ANTONIO';

SELECT * FROM temple WHERE nomem LIKE 'LOPEZ, ANTONIO';

****

Tratamiento de valores nulos

La construcción de predicados debe tener en cuenta la posibilidad de que alguno de los operadores o predicados simples, sean valores nulos.

En el pdf del aula se detallan todos los casos que hay para cada uno.

Predicado BETWEEN: V1 [NOT] BETWEEN V2 AND V3

1) V1 BETWEEN V2 AND V3

Si alguno de los valores V1, V2-V3 es NULL → NULL

2) V1 NOT BETWEEN V2 AND V3

a) Si V1 es NULL → NULL

b) SI V1 no es NULL

- Si V2 y V3 es NULL → NULL

- Si V2 no es NULL y V3 es NULL → Si V1 + V2 → V

- Si V2 no es NULL y V3 no es NULL → Si V1 + V3 → V

SQL estándar (versión del 92)

Incorpora dos funciones escalares que podemos usar para el manejo de cadenas:

- **LENGTH** (expresión): Me devuelve la longitud de la expresión. Si la expresión es una cadena de caracteres, me devuelve el número de caracteres. Si es un número, me devuelve el número de bytes que internamente se utilizan para almacenar.

SELECT nomem, LENGTH (nomem) FROM temple WHERE numde=111 ORDER BY 1;

- **SUBSTR** (cadena,posición,longitud): Devuelve una subcadena de una cadena. Donde:

- cadena: Es la cadena de la que partimos.

- posición: Es la posición en la que la cadena en la que comienza la subcadena a extraer.

- longitud: Es el numero de caracteres o tamaño de la subcadena a extraer.

Funciones de cadena de MySQL (no están el en estándar)

- **CONCAT** (cadena1, cadena2 [cadena3,…,cadena]): Une las cadenas que se le pasan por argumento.

- **LOCATE** (subcadena, cadena, [posición]): Busca la primera ocurrencia de la subcadena , dentro de la cadena, y a partir de la posición (si no se indica desde el principio)

Ej. Extraer los nombres de pila de los empleados

SELECT nomem, SUBSTR(nomem, LOCATE(',',nomem)+1, (LENGTH(nomem)-LOCATE(',',nomem)))as 'Nombre' from temple order by 1;

Predicados y subconsultas

Una subconsulta es una consulta dentro de otra, concretamente en la proyección [WHERE]. Además las subconsultas pueden anidarse.

Subconsultas que devuelven un único valor: Podemos construir predicados de la siguiente forma genérica:

SELECT proyección FROM tabla WHERE expresión (operador relacional <,>,…) (SELECT proyección FROM tabla [WHERE condición]) ORDER BY at;

Donde las subconsultas devuelven un único valor

Ej. Obtener todos los datos de los empleados que son más jóvenes que GARCIA,AUGUSTO.

SELECT * FROM temple WHERE fecna > (SELECT fecna FROM temple WHERE nomem LIKE 'GARCIA,AUGUSTO') ORDER BY fecna;

Subconsultas que devuelven más de un valor:

- Predicados cuantificados: Utilizan las palabras reservadas

- ALL: Comprueba si el predicado se cumple para todos los valores que devuelve la subconsulta.

- SOME o ANY: Comprueba si el predicado se cumple para alguno de los valores que devuelven la subconsulta.

EJ. Obtener los datos de los empleados que tienen un salario superior a cualquier empleado del departamento 110

SELECT * FROM temple WHERE salar > ALL (SELECT salar FROM temple WHERE numde=110) ORDER BY salar;

************************************CASOS ESPECIALES**

Si la subconsulta devuelve una tabla vacía: EL predicado es verdadero tanto con SOME o ANY como con ALL.

Si la subconsulta devuelve valores NULL mezclados con valores validos:

- Con ALL:

- Sera F si es falso alguno de ellos.
- Si es V para todos los valores válidos, será NULL.

- Con SOME:

- Sera V si es V para alguno de ellos.
- Si es F para todos los valores válidos, será NULL.

Si la subconsulta devuelve solo valores NULL entonces es NULL (similar a F) en todos los casos.

****

- Predicado IN: Comprueba si la expresión es igual a alguno de los valores de la subconsulta.

SELECT proyección FROM tabla WHERE expresión IN (SELECT proyección FROM tabla [WHERE condición]);

EJ. Obtener por orden alfabético los nombres de los empleados que trabajan en el mismo dep FLOR,DOROTEA y GALVEZ,PILAR

SELECT nomem FROM temple WHERE numde IN (SELECT numde FROM temple WHERE nomem IN ('GALVEZ,PILAR', 'FLOR,DOROTEA')) order by 1;

- Predicado EXISTS: Comprueba si la subconsulta devuelve algún valor, en cuyo caso es V, y si no devulve ningún valor, entonces es F. Nunca devuelve NULL.

SELECT proyección FROM tabla WHERE EXISTS (SELECT proyección FROM table [WHERE condición]);

Ej. Obtener el nombre de los centros donde existen departamentos con tipo de dirección en funciones.

SELECT nomce FROM tcentr WHERE EXISTS (SELECT numce FROM tdepto WHERE tidir LIKE ‘F’);

Datos de Tipo Tiempo

Para trabajar con datos de tipo tiempo, en realidad usamos una cadena de caracteres que se ajusta a unos determinados formatos o patrones preestablecidos.

Cuando introducimos datos de tipo tiempo de manera explícita, en realidad lo que hacemos es introducir una cadena de caracteres cuyo formato lo admite el SGBD como un dato de tipo tiempo. Ello supone, además, que puedo realizar operaciones de comparación (<,<,=…) entre un dato de tipo tiempo almacenado en la BD y una cadena de caracteres que cumpla con el formato.

- Fecha (DATE) En mysql su formato general es 'YYYY-MM-DD'

- Hora (TIME) EN mysql su formato general es 'HH:MM:SS'

- Instante (TIMESTAMP) En MySQL su formato general es 'YYYY-MM-DD HH:MM:SS'

1) Registros especiales

Existen dos registros especiales de tiempo, que también se pueden tratar como funciones.

CURRENT DATE (): Devuelve la fecha actual del sistema.

CURRENT TIME(): Devuelve la hora.

En mysql, además, tenemos:

CURRENT TIMESTAMP(): Devuelve el instante actual.

2) Funciones que operan con datos de tipo tiempo

- Funciones que permiten descomponer un dato tipo tiempo:
    - YEAR (expresión)
    - MONTH (expresión)
    - DAY (expresión)
    - HOUR (expresión)
    - MINUTE (expresión)
    - SECOND (expresión)
- Funciones que permiten formatear datos tiempo en mysql:
    - DATE_FORMAT (fecha, formato)
        -Fecha es un dato fecha almacenado o una cadena con formato valido
        -Formato que se va a usar
    - TIME_FORMAT (hora, formato)

<u>EJ. Pasar las fechas a nuestro formato habitual</u>

SELECT nomem, date_format(fecna, '%d/%m/%Y') as Nacimiento from temple order by 2;

3) Aritmética de fechas

Un valor de tipo tiempo puede restarse de otro o puede incrementarse o decrementarse con una duración PERO EL RESULTADO OBTENIDO ES UNA DURACION o un número de días en otros sistemas.

- Incrementar o decrementar

DATE_ADD (fecha, INTERVAL valor intervalo): Suma a una fecha una duración indicado por un intervalo.

-fecha. Es la fecha

-INTERVAL es una palabra reservada del lenguaje que hay que poner obligatoriamente.

-valor es el número de los intervalos que se le suma a la fecha

-intervalo indica el tipo de intervalo que se le va a sumar (year,month,week,day,…).

DATE_SUB (fecha, INTERVAL valor intervalo): Resta a una fecha una duración indicado por un intervalo.

-fecha. Es la fecha

-INTERVAL es una palabra reservada del lenguaje que hay que poner obligatoriamente.

-valor es el número de los intervalos que se le suma a la fecha

-intervalo indica el tipo de intervalo que se le va a sumar (year,month,week,day,…).

Ej. Actualizar los datos de los empleados incrementando las fechas de nacimiento y e ingreso en 20 años.

UPDATE temple SET fecna = DATE_ADD(fecna, INTERVAL 20 year), fecin = DATE_ADD(fecin, INTERVAL 20 year);

- Diferencia entre datos de tipo tiempo:

- DATEDIFF (fecha1,fecha2): Devuelve el número de días entre dos valores de fecha (fecha1-fecha2)

- fecha1 es obligatorio y debe ser la más reciente.

- fecha2 es obligatorio.

- TIMESTAMPDIFF (intervalo, dato_tiempo1, dato_tiempo2): Devuelve la diferencia en intervalos entre dato_tiempo1 y dato_tiempo2.

- intervalo es uno de los intervalos usados en las funciones DATE_ADD y DATE_SUB.

- dato_tiempo1 es obligatorio y representa el más antiguo.

- dato_tiempo2 es obligatorio y representa el más actual.

Ej. Calcular la edad y antigüedad en la empresa de los empleados.

SELECT nomem, TIMESTAMPDIFF(year, fecna, current_date) as Edad, TIMESTAMPDIFF(year, fecin, current_date) as Antigüedad from temple;

Ej2. Mostrar los datos de los empleados con los siguientes criterios: Separar nombres y apellidos, las fechas de nacimiento e ingreso mostrarlas en formato europeo, añadir edad y antigüedad.

SELECT SUBSTR(nomem, LOCATE(',',nomem)+1, (LENGTH(nomem)-LOCATE(',',nomem)))as Nombre, SUBSTR(nomem,1,LOCATE(',',nomem)-1) as Apellido, date_format(fecna,'%d/%m/%Y') as 'Fecha de Nacimiento', timestampdiff(year,fecna,current_date) as Edad, date_format(fecin,'%d/%m/%Y') as 'Fecha de ingreso', timestampdiff(year,fecin,current_date) as Antigüedad from temple;

CONSULTAS CON AGRUPAMIENTOS DE FILAS

1) Agrupamiento implícito. Funciones colectivas o de columna.

Se crea un grupo, sin necesidad de indicarlo, con todas las filas que satisfacen la consulta. A ese grupo se le aplica una función que realiza un determinado calculo.

Estas funciones se llaman colectivas ya que operan sobre un consulto de valores. Se llaman también de columna porque operan sobre una de las columnas de la consulta o bien sobre una expresión basada en una de las columnas.

Son las siguientes:

- AVG(valores): Devuelve el promedio de los valores.

- MAX(valores): Devuelve el máximo de los valores.

- MIN(valores): Devuelve el mínimo de los valores.

- SUM(valores): Devuelve la suma de los valores.

- COUNT(valores): Devuelve el número de los valores.

Reglas de uso:

- Antes de aplicar cualquiera de estas funciones a los valores, si estos incluyen valores NULOS, se eliminan antes de la operación.

- Las funciones MAX, MIN y SUM devuelven el mismo tipo de dato que sus valores.

- La función AVG siempre devuelve un valor decimal.

Formatos de uso:

- Formato 1: (para todas las funciones)

Nombre_funcion ([DISTINCT] nombre_columna);

Select Count (distinct salar);

Si se indica DISTINCT, antes de la operación se eliminan los valores duplicados.

- Formato 2: (para todas las funciones excepto COUNT)

Nombre_funcion (expresion)

Select Max(timestampdiff(year,fecna,current_date)) as 'Más veterano' from temple;´

En este caso la expresión debe incluir un nombre de columna y no puede contener otra función de columna.

- Formato 3: (solo aplicable a la función COUNT):

Nombre_funcion(*)

Select count(*) from temple where numde=111;

Devuelve el número de filas que satisfacen la consulta.

La proyección de una función colectiva es incompatible con la proyección de un dato individual, ya que la función colectiva devuelve un valor y el dato individual devuelve tantos valores como filas satisfagan la consulta.

2) Agrupamiento explícito

Podemos crear grupos con aquellos valores de una consulta que se repitan. Este se realiza mediante la clausula GROUP BY, cuya sintaxis es la siguiente:

SELECT proyección (*1)

FROM tabla

[WHERE selección]

[GROUP BY at1 [at2,…atn]]

[HAVING condición] (*2)

[ORDER BY at1 [at2,…atn]];

(*1) En la proyección implícita hemos visto que es incompatible la proyección de una función colectiva con una columna individual. Cuando creamos grupos de manera explícita (GROUP BY) el concepto es similar, es decir, solo se podrá proyectar el atributo por el que se crean los grupos junto con funciones de columna

Ej. Obtener cuantos empleados trabajan en cada departamento

SELECT numde, count(*) FROM temple GROUP BY numde;

Podemos hacer agrupamientos basados en expresiones en vez de en atributos.

Ej. Obtener cuantos empleados hay con la misma edad

SELECT timestampdiff(year, fecna, current_date) as Edad, count(*) FROM temple GROUP BY 1;

Podemos hacer agrupamientos por más de un atributo.

Ej. Obtener por departamento cuantos empleados tienen el mismo numero de hijos

SELECT numde, numhi, count(*) as Empleados FROM temple GROUP BY 1,2 ORDER BY 1;

(*2) Clausula HAVING: Permite discriminar o descartar grupos previamente creados con GORUP BY, lleva una condición que se establece sobre el grupo, no sobre los elementos individuales. Por lo tanto, esa condición debe ser una función colectiva sobre el grupo.

Ej. Obtener por departamento cuantos empleados trabajan en cada departamento, para los departamentos en los que trabajen más de 3 empleados.

SELECT numde, count(*) as Empleados FROM temple GROUP BY 1 HAVING (count(*) >3) ORDER BY 1;

No se debe confundir la condición sobre el grupo (HAVING) con la condición sobre la consulta (WHERE). Recuerda que el HAVING, al ser sobre grupo, es una condición basada en una función de columna, mientras que WHERE es una condición que puede estar basada en cualquier atributo o expresión.

Ej. Para los empleados que tengan mas de 40 años, obtener cuantos empleados hay con la misma edad, siempre que haya mas de un empleado con la misma edad.

SELECT timestampdiff(year,fecna,current_date) as Edad, count(*) FROM temple WHERE timestampdiff(year,fecna,current_date) > 40 GROUP BY 1 HAVING(count(*)>1) ORDER BY 1;

Álgebra relacional

Codd enuncio el algebra relacional como la base matemática para operar con el Modelo Relacional.

Se trata de un conjunto de operadores que tratan a las relaciones (tablas) como conjuntos y a una fila de una tabla como un elemento de un conjunto.

Estos operadores pueden clasificarse:

- Atendiendo a su naturaleza:
    - Operadores clásicos de conjuntos (unión ,diferencia, producto cartesiano, intersección)
    - Operadores definidos para el Modelo Relacional (proyección, selección, reunión natural(join), división)
- Atendiendo al número de operandos (nº de relaciones sobre el que operan):
    - Unarios: Solo operan sobre una única relación (proyección, selección)
    - Binario: Necesitan al menos 2 relaciones para operar (unión, diferencia ,intersección, producto cartesiano, reunión, división)

Operadores algebraicos: La aplicación de algunos de estos operadores requiere una condición previa y es la compatibilidad entre relaciones. Diremos que dos relaciones son compatibles si tienen el mismo número de atributos o columnas y además esas columnas están definidas en el mismo tipo de dato y en el mismo orden.

- Unión: Es un operador binario que requiere que las dos relaciones sean compatibles. Por ejemplo sean A y B dos relaciones compatibles

    SELECT * FROM A UNION SELECT * FROM B; → Devuelve todas las filas de A y de B sin repetición.

    SELECT * FROM A UNION ALL SELECT * FROM B; → Devuelve todas las filas de A y de B aunque haya repetidas.

- Diferencia: Es un operador binario que requiere que las dos relaciones sean compatibles. Por ejemplo sean A y B dos relaciones compatibles.

    A MINUS B → Devuelve las filas de A que no están en B

    B MINUS A → Devuelve las filas de B que no están en A

- Intersección: Es un operador binario que requiere que las dos relaciones sean compatibles. Por ejemplo sean A y B dos relaciones compatibles.

    A INTERSEC B → Devuelve las filas comunes a ambas tablas.

- Proyección: Es un operador unario que requiere solo una relación para operar. Obtiene un subconjunto vertical de una relación.

    SELECT {*|at1[,at2…atn]|expresión} FROM tabla;

- Selección: Es un operador unario que requiere solo una relación para operar. Obtiene un subconjunto horizontal de la tabla estableciendo una condición que deben cumplir las filas de la tabla.

    SELECCT proyección FROM tabla WHERE (condición);

- Producto cartesiano: Es un operador binario pero que no requiere que las relaciones sean compatibles. Sean A y B dos relaciones no necesariamente compatibles, el resultado cartesiano de A*B es:
    - Una tabla que tiene las columnas de A mas las columnas de B.
    - Tiene tantas filas como resulte de combinar cada fila de A con todas las de B. Ej. tcentr*tdepto

    SELECT * FROM tcentre,tedepto;

CUALIFICACION DE NOMBRES

Cuando trabajamos con mas de un a relación puede ocurrir que haya atributos que se llamen igual en mas de una ellas. Esto ocurre con casi todas las claves foráneas.

Para poder referirnos con exactitud a estos atributos, se utiliza la cualificación de nombres, que consiste en indicar el nombre de la tabla antes del atributo.

```
SELECT * FROM tcentr, tdepto ORDER BY tcentr.numce;
```

Además del nombre de la tabla también podemos usar un alias de la tabla que se crea en el FROM. Por ejemplo la consulta anterior es similar a:

```
SELECT * FROM tcentr c, tdepto d order by c.numce;
```

- Reunión: Es el operador más importante del algebra relacional ya que nos permite acceder a las tuplas de una tabla que están relacionadas con las de otra.

Podemos decir que la reunión es un operador compuesto ya que resulta de la combinación de dos de los operadores anteriores

Reunión = producto cartesiano + selección de los atributos

EJ. Reunión entre centros y departamentos

```
SELECT * FROM tcentr c, tdepto d WHERE c.numce =d.numde ORDER BY c.numce;
```

EJ. Reunión entre departamentos y empleados

```
SELECT * FROM tdepto d, temple e WHERE d.numde=e.numde ORDER BY d.numde;
```

La reunión puede realizarse algebraicamente, como acabamos de ver o también puede realizarse mediante el operador especifico JOIN

```
SELECT * FROM tdepto d JOIN temple e ON d.numde=e.numde ORDER BY d.numde;
```

La reunión es el operador mágico que nos permite obtener información de una tabla a partir de datos de cualquier tabla de mi esquema.

Ej. Obtener todos los datos de los empleados que trabajan en departamentos cuyo presupuesto es superior a 50000€

```
SELECT e.* FROM tdepto d, temple e WHERE d.numde=e.numde AND presu>50000;
```

EJ. Obtener todos los datos de los departamentos en los que trabajen empleados que tengan más de 3 hijos.

```
SELECT d* FROM tdepto d, temple e WHERE d.numde=e.numde AND numhi>3;
```

METODO PARA LA REALIZACION DE CONSULTAS DE TODO TIPO

Se trata de seguir unos pocos pasos en un orden determinado.

1. A partir de los datos que me piden y de los datos que me dan, determinar en el Modelo Lógico las relaciones implicadas y el camino más corto para llegar a ellas. Realizar la reunión entre esas tablas. (FROM y WHERE)
2. Añadir en la selección (WHERE) la condición o condiciones específicas de la consulta.
3. Determinar si hay grupos. En caso de que si (me piden como salida una función de columna), si es explicito lleva GROUP BY. Si hay descartes de grupos lleva HAVING.
4. ¿Qué me piden? Son los datos de salida en el SELECT.
5. Si se pueden repetir las filas lleva DISTINCT.