MySQL

Antonio Espín Herranz

Herramientas MySQL

- Herramienta de Gestión de MySQL.
  - phpMyAdmin (escrita en PHP).
  - Desplegar la aplicación en el directorio:
    - htdocs (de Apache).
  - Una buena opción → “EasyPHP”.
    - Instala Apache, MySQL, PHP y PHPMyAdmin.
  - Necesario arrancar el Servidor de Apache y MySQL.

PHPMyAdmin

- Archivo de configuración: config.inc.php
  - Debe existir en el directorio raíz de PHPMyAdmin.
  - Por defecto aparece: config.sample.inc.php.
  - Podemos configurarlo en:
  - localhost/phpmyadmin/scripts/setup.php

PHPMyAdmin

- Servers → Add
  - Server hostname: localhost.
  - Conection type: tcp.
  - PHP extension to use: mysqli.
  - Authentication type: http.
  - User for config auth: root.
- Botón Add.
- Configuration, botón save.
- Habrá creado una carpeta llamada config y dentro estará el fichero: config.inc.php
- Copiarlo a la carpeta raíz de php.
- Actualizar el navegador.
- Ahora pedirá la contraseña.

Herramientas MySQL

- Servidor de MySQL. Conectar:
  - Tener el Servicio arrancado en Windows.
  - Para conectar, en la carpeta bin (desde una consola):
  - Mysql –u root –p
    - Así conectamos con el usuario root y al entrar nos pedirá la password.

- Para el resto de herramientas lo mismo pero desde una ventana.
  - MySQL Administrator.
  - MySQL Query.
  - Toad.
  - HeidiSQL.

Tipos de datos en MySQL

- **Bit[M]**: Tipo bit. M indica el número de bits. El límite son 64.
- **Tinyint[Unsigned]:** Entero de 0 a 255 o de -128 a 128 con signo.
- **Bool / Boolean**: <> 0 → true, 0 → false.
- **Smallint[unsigned]:** entre 0 y 65535, -32… al 32 … con signo.
- **Int / integer**: entero normal.
- **Bigint[M][unsigned]:** entero largo.
- **Float[(M,D)]:** Números en coma flotante M dígitos y D los decimales.
- **Double[(M,D)]:** idem pero mayor precisión.
- **Decimal[M [, D])]:** M numero total de digitos y D decimales.

Tipos de datos en MySQL

- **Date**: Fechas, se puede pasar como cadenas. Formatos: YYYY-MM-DD, YY-MM-DD o YYMMDD.
- **Time**: Horas, HH:MM:SS, HHMMSS, HHMM.
- **Char(longitud)**: cadena de caracteres, reserva espacio aunque no se utilicen. De 0 a 255.
- **Varchar(longitud)**: cadena de caracteres, el espacio no se reserva. 65… tope de chars.
- **Blob / Text**: Bits sin interpretar. Texto mas largo de 255, blob diferencia entre comparaciones de mayúsculas y minúsculas, en cambio Text no.

Lenguaje SQL

Componentes del lenguaje SQL:

* **DML** (Data Manipulation Language)
  Manipulación de los datos
  Seleccionar / Añadir / Modificar / Borrar

* **DDL** (Data Definition Language)
  Creación de objetos de la BD
  Creación de tablas, ...

Lenguaje SQL

DML = Data Manipulation Language

(Manipulación de los Datos)

* **Select**

  Permite hacer **consultas** en la BD, obteniendo una nueva “tabla” con los resultados de la consulta.

* **Insert**

  Permite **añadir** registros en la BD (filas de una tabla).

* **Update**

  Permite **modificar** registros en la BD (filas de una tabla).

* **Delete**

  Permite **borrar** registros en la BD (filas de una tabla).

* **Truncate**

  Permite **borrar** registros en la BD y los campos que sean autoincrement se vuelven a inicializar.

Lenguaje SQL

- Clasificación de las Consultas:
  - De **Selección**: Select
    - No modifican la base de datos, se utilizan para extraer información de la Base de datos.
    - Los resultados se muestran en forma de Tabla.
  - De **Acción**: Insert, Update, Delete, Truncate.
    - Si modifican la base de datos.
    - Devuelve el número de registros afectados.

Lenguaje SQL

- Selección **Select**: Selecciona las filas que cumplen los criterios, el resultado es otra tabla.

- Sintaxis Select: (***los [ ] indican opcional***)
  - **Select** campos [**into** VARIABLE] **from** tabla [**where** criterios] [**group by** campos_agrupacion [**having** criterios]] [**order by** campos]

- Ejemplos:
  - Select nombre, apellidos from clientes where departamento = ‘compras’
  - Select departamento, count(*) from ventas group by departamento having count(*) > 12000
  - Select **distinct** departamento from compras;

Lenguaje SQL

- Apartados del comando select:
- **Campos**:
  - Son los campos que queremos mostrar en el resultado de la consulta.
  - Cuando tengamos varias tablas en la consulta los campos tendrán que ir precedidos del nombre de la tabla.
    - Select tabla1.campo1, tabla2.campo3 …
  - Dentro de esta sección también pueden aparecer valores constantes o funciones.
  - Si utilizamos un ***** estaremos seleccionando todos los campos de la tabla o tablas que formen parte de la consulta.

Lenguaje SQL

- Apartados del comando select:
- **into Variable**:
  - Lo vamos a utilizar para redirigir el resultado de la consulta a una variable.
  - Se utilizará dentro de las funciones definidas por el usuario.

Lenguaje SQL

- **From** Tabla:
  - Tabla de donde se va a realizar la consulta. También puede ser una vista o puede haber varias tablas (lo veremos mas adelante).
  - Es conveniente utilizar alias para los nombres de la tablas. Esto lo utilizaremos cuando tengamos varias tablas.
  - El alias los podemos utilizar siempre que referenciemos a un campo de una tabla.

  - Ejemplo: Sobre una sola tabla. En este caso el alias sería ‘d’.
    - Select **d**.id, **d**.departamento from departamentos **d**

Lenguaje SQL

- Apartados del comando select:
- **Where** (Criterios): Nos permite utilizar:
  - Atributos, constantes o expresiones.
  - Operadores de comparación (>, <, >=, <=, =, <>).
  - Operadores lógicos: (and, or, not).
  - Texto entre comillas ‘’, números tal cual.
  - Fechas entre comillas ‘’ y en formato ‘2010-06-30’

- Between … and …: where sueldo between 20000 and 40000.
- Like: where nombre like ‘j%’. → Todos los que empiezan por j.
- Campo IS [NOT] NULL → Si tiene valor o no el campo.

Establecer Criterios en la consulta

- Cuando la claúsula Where no está, no establecemos ningún criterio y se volcarán todos los resultados de la tabla.
- **Where** campo operador valor_cte / campo
- Los operadores pueden ser:
  - Los relacionales: <, >, <=, >=, =, <>
  - Like
  - Between … and …, not between … and …
  - Is, is not
  - Regexp
  - isnull

Establecer Criterios en la consulta

- Ejemplos:

  Select nombre from alumnos Where edad > 34;

  Select nombre from alumnos Where fechaNacimiento between ‘1980-1-1’ and ‘1980-3-31’;

  Select nombre from alumnos Where nombre like ‘a%’;

- Si necesitamos establecer mas de un criterio tenemos que utilizar los operadores lógicos: **and, or**.
  - Podemos utilizar paréntesis para forzar prioridades y agrupar las condiciones.

  - **And**: será verdadero cuando se cumplan todos los criterios:

  Select nombre from alumnos

  Where (edad > 34) **and** (fechaNacimiento between ‘1980-1-1’ and ‘1980-3-31’);

  Solo mostrará los registros que cumplan ambas condiciones.

  - **Or**: será verdadero cuando se cumpla algunos de los criterios:

  Select nombre from alumnos Where (edad > 34) **or** (nombre like ‘a%’)

  Muestra los registros que cumplan una u otra condición.

Establecer Criterios en la consulta

- Dentro de Like podemos utilizar metacaracteres:

  % Representa cualquier carácter. Incluso la cadena vacía.

  _ Guión de subrayado representa 1 solo carácter. “___” → (3 guiones son 3 caracteres).

Establecer Criterios en la consulta

- **Regexp** podemos utilizar expresiones regulares, para establecer criterios de búsqueda de una forma más potente.

- Se muestran las distintas posibilidades que tiene en la siguiente tabla:

Expresiones Regulares

expresión regular	significado
“.”	Cualquier carácter, pero sólo uno
“[xyz]”	El carácter **x**, el **y** o el **z**
“[x-z]	Igual que el anterior
“[0-9]”	Cualquier número
“x*”	Una o más equis
“.*”	Cualquier número de caracteres
“^b”	Que empiece por **b**
“b$”	Que termine por **b**
“[69].*”	Que empiece por 6 o por 9
“^[69]”	Que empiece por 6 o por 9
“^.....$”	Que tenga exactamente cinco caracteres
“^.{5}$”	Que tenga exactamente cinco caracteres

Establecer Criterios en la consulta

- Localizar atributos con valores nulos.
- Disponemos de una tabla de productos (descripción, precio, fechaAlta, fechaBaja)

- Por ejemplo, consultar los productos que no tienen fecha de Baja:

```
Select descripcion from productos
Where fechaBaja is null;
```

- Los que si tienen fecha de baja:

```
Select descripcion from productos
Where fechaBaja is NOT null;
```

Establecer Criterios en la consulta

- **ISNULL(*expr*)**
  - Si *expr* es NULL,
    - ISNULL() retorna 1, sino retorna 0.

- mysql> SELECT ISNULL(1+1); → 0
  mysql> SELECT ISNULL(1/0); → 1

Establecer Criterios en la consulta

- **In / Not In**
  - Si el campo o el valor se encuentra o no dentro de un subconjunto.

- mysql> SELECT 2 IN (0,3,5,'wefwf'); → 0
- mysql> SELECT 'wefwf' IN (0,3,5,'wefwf'); → 1

Subconsultas: Any, In, Some, All

- **operand comparison_operator ANY (subquery)**
  - La palabra clave **ANY** , que debe seguir a un operador de comparación, significa “return TRUE si la comparación es TRUE para ANY (cualquiera) de los valores en la columna que retorna la subconsulta.
  - SELECT s1 FROM t1 WHERE s1 > ANY (SELECT s1 FROM t2);

Subconsultas: Any, In, Some, All

- **operand IN (subquery) :**

  Si el valor está dentro del subconjunto.

  Select * From empleados

  Where codDpto in (select codDpto

  From Departamentos where (situacion = ‘3 planta’)

Subconsultas: Any, In, Some, All

- **operand comparison_operator SOME (subquery)**
- La palabra **SOME** es un alias para ANY. Por lo tanto, estos dos comandos son el mismo:
  - SELECT s1 FROM t1 WHERE s1 <> ANY (SELECT s1 FROM t2);
  - SELECT s1 FROM t1 WHERE s1 <> SOME (SELECT s1 FROM t2);

Subconsultas: Any, In, Some, All

- *operand comparison_operator* ALL (*subquery*)

- La palabra **ALL**, que debe seguir a un operador de comparación, significa “return TRUE si la comparación es TRUE para ALL todos los valores en la columna que retorna la subconsulta.”

- Por ejemplo:
  - SELECT s1 FROM t1 WHERE s1 > ALL (SELECT s1 FROM t2);

Lenguaje SQL

- Eliminación **Delete**: Elimina los registros que cumplen los criterios.

- Sintaxis:
  - DELETE FROM Tabla WHERE criterios

- Ejemplo:
  - DELETE FROM Empleados WHERE Cargo = 'Vendedor';

Lenguaje SQL

- Cuando utilizamos delete para eliminar registros de una tabla y esa tabla tiene un campo autoincrement si después del borrado damos de alta algún registro los valores de este campo no se reutilizan para los nuevos registros.

- Ver ejemplo.

Ejemplo

- Disponemos de los siguientes registros:
- Y realizamos las siguientes operaciones:
  - Delete from Departamentos where id = 2;
  - Insert into Departamentos (departamento) values (‘Exportación’);

Id	Departamento
1	Compras
2	Ventas
3	Contabilidad

Id	Departamento
1	Compras
3	Contabilidad
4	Exportación

Cuando realizamos una operación de inserción los campos autoincrement se generan de forma automática.

Lenguaje SQL

- Eliminación **Truncate**: Elimina todas las filas y los campos autoincrement los vuelve a inicializar.

- Sintaxis:
  - TRUNCATE tabla

- Ejemplo:
  - TRUNCATE despartamentos

Lenguaje SQL

- Inserción de datos: **Insert into**

- Sintaxis a)
  - **INSERT INTO** Tabla (campo1, campo2, .., campoN) **VALUES** (valor1, valor2, ..., valorN)

- Sintaxis b)
  - **INSERT INTO** Tabla (campo1, campo2, ..., campoN) **SELECT** TablaOrigen.campo1, TablaOrigen.campo2, ..., TablaOrigen.campoN **FROM** TablaOrigen

Lenguaje SQL

- Ejemplos:
  - INSERT INTO Clientes.Cliente_Nuevos SELECT Clientes_Viejos.* FROM Clientes_Viejos;

  - INSERT INTO Empleados (Nombre, Apellido, Cargo) VALUES ('Luis', 'Sánchez', 'Becario');

- También soporta esta sintaxis cuando queremos insertar mas de un registro con una sentencia.

```
INSERT INTO `gruposcursos` (`id`, `descripcion`) VALUES
(1, 'Java'), (2, 'Seminarios Prácticos'), (3, 'Visual Basic'),
(4, 'Bases de Datos'));
```

Lenguaje SQL

- Actualización Update:

- Sintaxis:
  - **UPDATE** Tabla **SET** Campo1=Valor1, Campo2=Valor2, ... CampoN=ValorN **WHERE** Criterio;

- **EJEMPLOS:**
  - UPDATE Pedidos SET Pedido = Pedidos * 1.1, Transporte = Transporte * 1.03 WHERE PaisEnvío = 'ES';

Lenguaje SQL

- Funciones de agregado (con **Group by**):
  - Count(* | distinct atributo) → cuenta.
  - Sum(Atributo) → Suma los valores.
  - Avg(Atributo) → Media.
  - Max(Atributo) → Máximo.
  - Min(Atributo) → Mínimo.

- Select codDpto, count(*) from Empleados group by CodDpto.

Lenguaje SQL

- Otro uso típico de Group By es para quitar los registros repetidos de una tabla.
  - Select nombre from alumnos group by nombre.

- En este caso si en la tabla alumnos hubiera repetidos sólo los mostraría una vez.

Lenguaje SQL

- **Having**: Establecer condiciones para los registros de un grupo.
- Where lo utilizamos para filtrar filas de una tabla.
- En caso de tener grupos para establecer alguna condición tenemos que utilizar Having.
- Por ejemplo:
  - Obtener los registros duplicados de una tabla.

```
Select nombre from alumnos
group by nombre
having count(*) > 1
```

Lenguaje SQL

- Varias tablas:
  - **Select** Empleados.CodDpto, Descripcion, sum(sueldo) from **Empleados, Departamentos**

    Where Empleados.CodDpto = Departamentos.CodDpto

    a**nd** Empleados.CodDpto = Departamentos.CodDpto

    Group by Empleados, Descripcion.

Join

- SELECT table1.* FROM table1 **LEFT JOIN** table2 ON table1.id=table2.id WHERE table2.id IS NULL;
- Left join, right join, inner join.
- Más cómodo utilizando alias:
  - Select a.nombre, b.telefono from Tabla1 a left join Tabla2 b on a.id = b.idnombre
  - Los alias también se puede utilizar a nivel de campos:
    - select nombre as nombreAlumno from alumnos;

Join

- **cross join**: Producto cruzado. Combina cada registro de la primera tabla con cada registro de la tabla relacionada.
- **inner join:** Unión normal. Muestra sólo registros de ambas tablas que estén relacionados.
- **left join:** Muestra todos los registros de la primera tabla y sólo los registros relacionados en la segunda.
- **right join:** Muestra todos los registros de la segunda tabla y sólo los registros relacionados en la primera.

Join

- Resolver preguntas de este estilo:
- Dos equipos (de futbol y baloncesto) y queremos saber
  - Quien se ha apuntado sólo a futbol,
  - Quien se ha apuntado sólo a baloncesto.
  - Quien se ha apuntado a ambos equipos.

Hacer el ejemplo.

Union / Union All

- Representa la unión de conjuntos de datos.

- Podemos juntar varios campos de diversas tablas en el resultado de la consulta, teniendo en cuenta que de cada tabla tenemos que tomar el mismo número de campos y deben de ser del mismo tipo.

- **Union**: juntaría los datos quitando repetidos.
- **Union All**: Hace lo mismo pero sin quitar los repetidos.

Union / Union All

- Si disponemos de dos tablas:
  - Clientes: id, nombre, teléfono.
  - Proveedores: id, nombre, teléfono.

- Y queremos mostrar toda la información en una consulta:

  Select nombre, teléfono from clientes

  Union

  Select nombre, teléfono from proveedores

  - Podríamos juntar mas consultas (también se pueden juntar tablas), pero habría que seguir intercalando tablas.

Paginación en MySQL

- A la hora de ejecutar una consulta, podemos realizar una paginación.
- Al final del comando sql, añadimos “limit” inicio, tamaño_pagina;
- Ejemplo:
  - Select * from empleados limit 0,5;
  - Muestra los 5 primeros empleados.

Ordenación

Ordenación de los datos presentados (ORDER BY)

Notación:

SELECT <nombre_cols>

FROM <nombre_tablas>

[WHERE <condiciones_booleanas>]

ORDER BY <atributo_1>, ..., <atributo_N>;

Ejemplo:

SELECT nombre, edad

FROM alumnos

ORDER BY edad **DESC**; {por defecto es **ASC**}

Funciones de MySQL

- Todos los gestores de base de datos disponen de un conjunto de funciones del lenguaje que se pueden utilizar dentro de las consultas de SQL.

- MySQL también dispone de estas funciones y las podemos utilizar en los criterios y la selección de campos.

- Se suelen agrupar por la funcionalidad que realizan: de fecha / hora, de texto, etc.

Funciones de MySQL

- Grupos:
  - Matemáticas.
  - Fecha / Hora.
  - De Texto.

- Para probar las funciones de una forma rápida:
  - Select now()
  - Imprime la fecha y la hora del Sistema.

Funciones de MySQL: Matemáticas

- Round(valor, num_decimales)
  - Redondea la valor según el número de decimales indicado.
  - Select round(123.778, 2); → 123.78
- Ceiling(valor)
  - Redondea al alza.
  - Select ceiling(8.4); → 9
- Floor(valor)
  - Redondea a la baja.
  - Select floor(5.6); → 5

Funciones de MySQL: Matemáticas

- Abs(valor)
  - Devuelve el valor absoluto.
  - Select abs(-89); → 89

- Truncate(valor, num_decimales)
  - Trunca los decimales del número, según el número indicado.
  - Select truncate(8.97776, 1); → 8.9

- Sqrt(valor)
  - Devuelve la raíz cuadrada de un número.

Funciones de MySQL: Matemáticas

- Sign(valor)
  - Devuelve el signo del número.
  - 1 si es positivo.
  - (-1) si es negativo.

- Rand()
  - Genera números aleatorios entre 0 y 1.

- Pow(numero, exponente)
  - Devuelve el numero elevado al exponente.
  - select pow(2,3); → 8

Funciones de MySQL: Matemáticas

- Mod(numero, divisor)
  - Devuelve el resto de la división entera: numero / divisor
  - Select mod(8,3); → 2

- También soporta las funciones trigonométricas: sin, cos, tan, pi, etc.

Funciones de MySQL: Fecha / Hora

- Curdate()
  - Devuelve la fecha actual.
  - Select curdate(); → en formato: yyyy-mm-dd.

- CurTime()
  - Devuelve la hora actual.
  - Select curtime(); → 14:45:56

- Now()
  - La fecha y la hora actual.

Funciones de MySQL: Fecha / Hora

- DateDiff(fecha1, fecha2)
  - Devuelve la diferencia en días de ambas fechas. Resta fecha1 – fecha2
  - Select datediff(‘2010-12-31’, ‘2010-4-20’);

- Date(expresión)
  - Extrae la fecha de la expresión.
  - SELECT DATE('2003-12-31 01:02:03');
    → '2003-12-31'
  - Select date(now());

- Time(expresión)
  - Igual que la anterior pero lo hace con la hora.

Funciones de MySQL: Fecha / Hora

- Date_add(date, interval expr-type)
  - Incrementa una fecha el intervalo se lo podemos dar según la tabla adjunta.
    - Select date_add('2010-5-30', interval 1 day);
    - 2010-5-31
  - También es válido la siguiente sintaxis:
    - Select '2010-5-31' + interval 1 day;

MICROSECOND
SECOND
MINUTE
HOUR
DAY
WEEK
MONTH
QUARTER
YEAR
SECOND_MICROSECOND
MINUTE_MICROSECOND
MINUTE_SECOND
HOUR_MICROSECOND
HOUR_SECOND
HOUR_MINUTE
DAY_MICROSECOND
DAY_SECOND
DAY_MINUTE
DAY_HOUR
YEAR_MONTH

Funciones de MySQL: Fecha / Hora

- date_format(fecha, formato)
  - Devuelve la fecha según el formato aplicado.
  - select date_format('1997-10-04 22:23:00', '%w %m %y'); → 'saturday october 1997‘
  - Los formatos se especifican según la siguiente tabla:

Funciones de MySQL: Fecha / Hora

Especificador	Descripción
%a	Día de semana abreviado (Sun..Sat)
%b	Mes abreviado (Jan..Dec)
%c	Mes, numérico (0..12)
%D	Día del mes con sufijo inglés (0th, 1st, 2nd, 3rd, ...)
%d	Día del mes numérico (00..31)
%e	Día del mes numérico (0..31)
%f	Microsegundos (000000..999999)
%H	Hora (00..23)
%h	Hora (01..12)
%I	Hora (01..12)
%i	Minutos, numérico (00..59)
%j	Día del año (001..366)
%k	Hora (0..23)
%l	Hora (1..12)
%M	Nombre mes (January..December)
%m	Mes, numérico (00..12)
%p	AM o PM
%r	Hora, 12 horas (hh:mm:ss seguido de AM o PM)
%S	Segundos (00..59)
%s	Segundos (00..59)
%T	Hora, 24 horas (hh:mm:ss)
%U	Semana (00..53), donde domingo es el primer día de la semana
%u	Semana (00..53), donde lunes es el primer día de la semana
%V	Semana (01..53), donde domingo es el primer día de la semana; usado con %X
%v	Semana (01..53), donde lunes es el primer día de la semana; usado con %x
%W	Nombre día semana (Sunday..Saturday)
%w	Día de la semana (0=Sunday..6=Saturday)
%X	Año para la semana donde domingo es el primer día de la semana, numérico, cuatro dígitos; usado con %V
%x	Año para la semana, donde lunes es el primer día de la semana, numérico, cuatro dígitos; usado con %v
%Y	Año, numérico, cuatro dígitos
%y	Año, numérico (dos dígitos)
%%	Carácter '%' literal

Funciones de MySQL: Fecha / Hora

- Funciones para extraer partes de fecha:
  - Day(fecha): El día de la fecha
  - DayOfWeek(fecha): El nombre del día. (lunes, etc).
  - Month(fecha): El mes de la fecha.
  - NonthName(fecha): El nombre del mes de la fecha.
  - Year(fecha): El año de la fecha.

- Extract(type from date);
  - Extraer partes de una fecha.
  - SELECT EXTRACT(YEAR FROM '1999-07-02'); → 1999

Funciones de MySQL: Fecha / Hora

- Last_day(fecha)
  - select last_day('2003-02-05'); → '2003-02-28'
  - Toma una fecha o fecha/hora y retorna el valor correspondiente para el último día del mes. retorna null si el argumento es inválido.

- Parte de Hora:
  - Second(time), minute(time), hour(time)

Funciones de MySQL: De Texto

- Concat(cadena1, cadena2, …):
  - Concatena las n cadenas.
  - Select concat(‘Cadena1’, ‘ ‘, ‘cadena2’)
    - Cadena1 cadena2

- length(‘cadena’)
  - Devuelve la longitud de la cadena.

- Find_in_set(string, lista de string)
  - Retorna un valor en el rango de 1 a *N* si la cadena *string* está en la lista de cadenas *strlist* consistente de *N* subcadenas.
  - select find_in_set('b','a,b,c,d'); → 2

Funciones de MySQL: De Texto

- instr(*str*,*substr*)
  - Localizar una cadena dentro de otra.
  - Retorna la posición de la primera ocurrencia de la subcadena *substr* en la cadena *str*.
  - select instr('foobarbar', 'bar'); → 4

Funciones de MySQL: De Texto

- Lcase(cadena)
  - Transformar a minúsculas la cadena.
- Ucase(cadena)
  - Transformar a mayúsculas.
- Left(cadena, numero)
  - Extraer n caracteres de la cadena, por la izquierda.
- Right(cadena, numero)
  - Extraer n caracteres de la cadena, por la derecha.

Funciones de MySQL: De Texto

- Substr(cadena, pos, len)
  - Extraer n caracteres (de longitud len) de la cadena, a partir de la posición pos.

- Space(n):
  - Genera n espacios en blanco.

- Trim(cadena), LTrim(cadena), Rtrim(cadena):
  - Recorta los blancos, de la cadena, por los dos lados, por la izquierda y por la derecha respectivamente.

Funciones de MySQL: De Control de Flujo

- **Case**: Permite una evaluación múltiple.
- Tiene dos posibles formatos:
- CASE *case_value*
  - WHEN *when_value* THEN *statement_list*
  - [WHEN *when_value* THEN *statement_list*] ...
  - [ELSE *statement_list*]
- END

- CASE
  - WHEN *search_condition* THEN *statement_list*
  - [WHEN *search_condition* THEN *statement_list*] ...
  - [ELSE *statement_list*]
- END

Funciones de MySQL: De Control de Flujo

- Ejemplo de Case:
- SELECT CASE 1
  - WHEN 1 THEN 'one‘
  - WHEN 2 THEN 'two'
  - ELSE 'more'
- END

- SELECT CASE
  - WHEN 1>0 THEN 'true'
  - ELSE 'false'
- END

Funciones de MySQL: De Control de Flujo

- **If (Condicional):**
- IF(*expr1*,*expr2*,*expr3*)
- Si *expr1* es TRUE (*expr1* <> 0 and *expr1* <> NULL) entonces:
- IF() retorna *expr2*; de otro modo retorna *expr3*.
- IF() retorna un valor numérico o cadena de caracteres, en función del contexto en que se usa.

- SELECT IF(1>2,2,3); -> 3
- SELECT IF(1<2,'yes','no'); -> 'yes‘
- SELECT IF(STRCMP('test','test1'),'no','yes'); -> 'no'

Funciones de MySQL: De Control de Flujo

- **IFNULL(*expr1*,*expr2*)**
- Si *expr1* no es NULL, IFNULL() retorna *expr1*, de otro modo retorna *expr2*.
- IFNULL() retorna un valor numérico o de cadena de caracteres, en función del contexto en que se usa.

- mysql> SELECT IFNULL(1,0); -> 1
- mysql> SELECT IFNULL(NULL,10); -> 10
- mysql> SELECT IFNULL(1/0,10); -> 10
- mysql> SELECT IFNULL(1/0,'yes'); -> 'yes'

Funciones de MySQL: De Control de Flujo

- **Exists / Not Exists**
  - Se puede aplicar a subconsultas y con condicionales:
- Si una subconsulta retorna algún registro, entonces EXISTS *subquery* es TRUE, y NOT EXISTS *subquery* es FALSE.
- Ejemplo:
  - SELECT column1 FROM t1 WHERE EXISTS (SELECT * FROM t2);

Funciones de MySQL: De Control de Flujo

- **¿Qué clase de tienda hay en una o más ciudades?**
  - SELECT DISTINCT store_type FROM Stores WHERE EXISTS (SELECT * FROM Cities_Stores WHERE Cities_Stores.store_type = Stores.store_type);
- **¿Qué clase de tienda no hay en ninguna ciudad?**
  - SELECT DISTINCT store_type FROM Stores WHERE NOT EXISTS (SELECT * FROM Cities_Stores WHERE Cities_Stores.store_type = Stores.store_type);
- **Borra la tabla SI existe:**
  - DROP TABLE IF EXISTS `ambitosgeograficos`;

Lenguaje DDL

- Esta otra parte del lenguaje SQL, la vamos a utilizar para crear objetos de la base de datos: Tablas, Vistas, Funciones, etc.
- Hay tres tipos de instrucciones:
  - **Create**: Para crear un nuevo objeto.
  - **Alter**: Modificar un objeto existente.
  - **Drop**: Para eliminar un objeto existente.

Lenguaje DDL

- Create database nombre_base_datos;
- Use nombre_base_datos;
- Create table nombre_tabla
  (Atributo1 tipo(longitud) [not null],
  (Atributo2 tipo(longitud) [not null],
  (AtributoN tipo(longitud) [not null],
  [Primary key (lista de atributos),]
  [Foreign key (Atributo) references nombretabla
  on delete cascade | restrict | set null | no action
  on update cascade | restrict | set null | no action
  ])

Ejemplo

- Creación de Tablas:
- Antes debemos crear la base de datos:
  - Create Database ClientesBD;
  - Use Database ClientesBD;
  - Create Table clientes

    (id_cliente int not null primary key auto_increment,
    Nombre Varchar(20) not null,
    Apellidos Varchar(20) not null,
    Direccion Varchar(50),
    Edad int,
    Sexo Enum('Hombre', 'Mujer') Default 'Mujer');

Lenguaje DDL

- Si se pone como nombre del índice la palabra primary key, entonces el índice crea la clave principal de la tabla:
  - alter table clientes add primary key (campos);

- Si se desea eliminar el índice:
  - drop index (índice) ON tabla;

- También (recomendada desde la versión 3.23):
  - alter table tabla drop index (índice);
  - alter table tabla drop primary key;

- En la creación de una tabla se pueden crear también los índices. Ejemplo:

```
create table personas (dni char(10), nombre varchar(25) not null,
apellidos varchar(50) not null,dirección varchar(50),
primary key (dni),
key datosCompletos (apellidos,nombre));
```

Lenguaje DDL

- **Claves externas:**
  - Sólo funcionan correctamente si las tablas son **innoDB. También se pueden crear** claves secundarias. Las claves secundarias se crean para campos de una tabla relacionados con campos que forman índices en otras tablas (normalmente forman claves principales, es decir son los campos que permiten relacionar tablas).
  - La creación de estas claves se puede hacer desde **create table.**
- **Sintaxis:**
  - **create table *tabla (***
  - *lista y propiedades de campo e índices,*
  - **constraint nombreDeClave**
  - **foreign key (camposQueFormaClave)**
  - **references *tabla (camposClaveDeLatabla));***
  - La palabra **constraint es opcional y permite indicar un nombre para la clave externa.**
  - **foreign key indica los campos de esta tabla relacionados con campos de otra.**
  - **references indica el nombre de la tabla relacionada y el nombre de los campos** Relacionados.

Lenguaje DDL

- Se pueden indicar:
  - **Cascade**: Borra o actualiza en la tabla padre y en las tablas relacionadas.
  - **Restrict**: Rechaza la operación de actualización / borrado.
  - **Set null**: Borra o actualiza en la tabla padre y establece a null en las hijas.
  - **No action**: No hace nada.

Lenguaje DDL

- Ejemplo:

```
CREATE TABLE `provincias` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `provincia` varchar(50) NOT NULL default '',
  `idComunidad` int(10) unsigned NOT NULL default '0',
  PRIMARY KEY  (`id`),
  KEY `FK_provincias_1` (`idComunidad`), → CREA EL INDICE
  CONSTRAINT `FK_provincias_1` FOREIGN KEY (`idComunidad`)
    REFERENCES `comunidades` (`id`)) → CREA LA RESTRICCIÓN
```

- Establece la ligadura entre el campo de idComunidad de la tabla de provincias con el campo id de la tabla de comunidades.

- **Otro Ejemplo: Añadiendo la clave externa después de crear la tabla.**

```
ALTER TABLE `tareas` ADD FOREIGN KEY ( `idcategoria` )
REFERENCES `agenda`.`categorias` (`id`)
ON DELETE CASCADE ON UPDATE CASCADE ;
```

Tipos de Tablas en MySQL

- **Tablas orientadas a transacciones (TST)**
  - InnoDB
  - BDB
- **Tablas no orientadas a transacciones (NTST)**
  - HEAP
  - ISAM
  - MERGE
  - myISAM
- **El tipo se especifica opcionalmente al final de la definición**
- **CREATE TABLE cursos (**
- **...**
- **) TYPE=InnoDB; → UTILIZAREMOS ESTE TIPO, de todas formas lo pone por defecto.**

Tablas en MySQL con acentos en los datos

- Cuando creamos las tablas en MySQL por defecto el conjunto de caracteres permitido lo pone a latin1 (permite acentos, utf-8 también).
  - Create database prueba;
  - Use prueba;
  - Create tabla p(a int, b char(20)) engine=InnoDB character set=uft-8;
  - Insert into p(a, b) values (1, ‘camión’);

Indices

- Create index [unique] nombre_indice on nombre_Tabla (lista_nombre_columnas)

- Añadir un índice modificando la estructura de la tabla:
  - Alter table Clientes add index (Apellido);
  - Alter table Clientes add primary key (id_cliente);

- Drop index nombre_indice; → Elimina el index.

Borrar Tablas

- Drop Table nombre_tabla;

- Para eliminar una base de datos:

  Drop database nombre_BD;

Modificar Tablas

- ALTER TABLE *nombreTabla [*
- ADD [COLUMN] *columna [FIRST | AFTER columna]*
- ADD PRIMARY KEY (*columna1, …)*
- ADD FOREIGN KEY [*reference_definition]*
- ALTER [COLUMN] *columna {SET DEFAULT literal | DROP*
- DEFAULT}
- DROP [COLUMN] *columna*
- DROP PRIMARY KEY
- RENAME [TO] *nombreTabla]*

- **Añade una columna a la tabla de Cursos:**

  ALTER TABLE CURSOS ADD DESCRIPCION TEXT;

- **Añade una columna en una lugar concreto y con valor por defecto:**

  ALTER TABLE CURSOS ALTER COLUMN LUGARPRACTICAS SET

  DEFAULT ‘AULA 7 – CITE III’;

Ejemplos

- Añadir una clave primaria a la tabla:
  - Alter table prueba add primary key (id);
    - Suponiendo que la tabla y la columna id existen.
- Eliminar la clave primaria:
  - Alter table prueba drop primary key;
- Renombrar una tabla:
  - Alter table prueba rename to pruebas;

Comandos MySQL

- **Show Databases**; → Muestra las bases de datos, se almacenan en el directorio data de mysql, un fichero por tabla.
- **Show Tables;** → Muestra las tablas de una base de datos.
- **Describe** nombre_tabla; → muestra la información de una tabla, la estructura.
- **Use** nombre_bd; → Selecciona una base de datos.
- **Select Database**(); → Muestra la Base de datos activa.
- **Select user()** → Con que user estamos conectados.
- **Select version**(); → La Versión instalada de MySQL;
- **Show Create table nombre_tabla** → Muestra como se ha creado la tabla.
- **Show Create database nombre_bd** → Muestra comose creo la BD.

Comandos MySQL

- Exportar datos de una consulta a un fichero de texto:
  - select * from clientes into outfile ‘fichero.txt’;
  - select * from city into outfile 'ciudades.txt' fields terminated by ',' enclosed by "" lines terminated by '\r\n';
    - Exporta al fichero, los campos separados por comas, encerrados entre comillas y terminan con un salto de línea.
  - select * from clientes into outfile ‘fichero.txt’ lines terminated by ‘\r\n’;
    - Mete el salto de línea y por defecto como separador un Tab.

Comandos MySQL

- Importar datos de un fichero de texto a una tabla:
  - load data local infile “c:\\archivos de programa\\ejemplo\\mitabla.txt” into table mi_tabla;
    - Carga el fichero, en la tabla, se espera que la tabla ya exista, los campos los leerá separados por un tab. Y cada registro vendrá en una línea.
  - load data local infile 'ciudades.txt' into table city fields terminated by ',' enclosed by "" lines terminated by '\r\n';
    - Se esperan los campos separados por comas, encerrados entre “ y terminan con un salto de línea.

Backup

- **Crear un backup**:
  mysqldump –uUsuario –p nombre_BD > fichero.sql

  – Ejemplo:
  mysqldump –u**root** –p Mi_Base_Datos > Mi_fichero.sql

- **Recuperar backup:**
  mysql –uUsuario –p nombre_BD < fichero.sql

  – Ejemplo:
  mysql –u**root** –p Mi_Base_Datos < Mi_Fichero.sql

Crear una BD con un script

- La extensión del fichero será: **sql**.
- Dentro de este fichero metemos todas las sentencias para crear las tablas, incluso las sentencias de creación de la base de datos y la selección.
- Desde phpmyadmin: seleccionar importar e indicar la ruta del fichero sql.
- Desde el modo comando:

  source nombre_fichero.

Gestión de Usuarios

- Cambiar el password de root / otros usuarios:

  UPDATE mysql.user

  SET Password=PASSWORD('*newpwd*') WHERE User='root';

  FLUSH PRIVILEGES; → **Refrescar los privilegios.**

- La función password() cifra la palabra que le hemos mandado.
- Prueba:
  - Select PASSWORD(‘hola’);

Gestión de Usuarios

- MySQL tiene la BD ‘MySQL’ para almacenar información de los usuarios.
  - Use MySQL.
  - Show Tables;
  - Describe user;
- Consultar los usuarios que tenemos:
  - Use mysql;
  - Select user from user;

Gestión de Usuarios

- Con la sentencia **GRANT** podemos crear un usuario a la par que otorgarle uno o varios privilegios sobre los objetos de una base de datos, o la base de datos completa.

- **Ejemplo**:
  - Crea un usuario sin permisos:
    - Grant usage on *.* to anonimo identified by ‘anonimo’;
  - Asigna permiso de Select a todas las tablas de la BD: si3_67:
    - Grant select on si3_67.* to anonimo;
  - Otorgar privilegios a todas las BDs y todas las operaciones:
    - Grant **ALL** on *.* to anonimo;
  - Crear un user y darle permisos de selección y actualización, en todas las tablas de la base de datos: nombre_Bd
    - Grant select, update on nombre_Bd.* to anonimo2 identified by ‘anonimo2’

Gestión de Usuarios

- Se pueden otorgar principalmente estos permisos:
  - **ALL:** para conceder todos los privilegios.
  - **CREATE:** permite crear nuevas tablas.
  - **DELETE:** permite usar la sentencia .
  - **DROP:** permite borrar tablas.
  - **INSERT:** permite insertar datos en tablas.
  - **UPDATE:** permite usar la sentencia .

Gestión de Usuarios

- Para consultar que permisos tiene un user:
  - Show Grants for **user**;

- **Quitar permisos:**
  - **REVOKE** SELECT ON prueba.gente FROM anonimo;
    - Elimina el permiso de select en la tabla gente de la BD prueba para el usuario anónimo.

- **Eliminar un user:**
  - **Drop user** nombre_user;

procedimientos

```
delimiter //

CREATE PROCEDURE proc1(OUT param1 INT)
    BEGIN
        SELECT COUNT(*) INTO param1 FROM prueba;
    END;
//

delimiter ;

-- Definir una variable:
Set @a = 0;

-- La llamada al procedure: Si el parámetro que recibe el procedimiento es de salida,
    tenemos que definir una variable.
-- Si el parámetro es de entrada, se puede enviar un valor.
call proc1(@a);

-- Ver el contenido del parámetro:
select @a;
```

Modo del parámetro:
IN, OUT, INOUT

Procedimientos

```
CREATE PROCEDURE sp_name ([proc_parameter[,...]])
[characteristic ...] routine_body

proc_parameter:      [ IN | OUT | INOUT ] param_name type

type:      Any valid MySQL data type

characteristic:
    LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA
  | SQL SECURITY { DEFINER | INVOKER }
  | COMMENT 'string'

routine_body:     Valid SQL procedure statement
```

Ejemplo de Procedimiento

```
delimiter $$

CREATE PROCEDURE upd_precio (IN ipc DECIMAL(3,1))
BEGIN
UPDATE ARTICULOS
   SET PREUNART= PREUNART + (PREUNART * ipc/100)
   WHERE PREUNART is not null;
END $$

delimiter ;


call upd_precio (10.0);
```

Ejemplo Procedimiento

```
DELIMITER $$

DROP PROCEDURE IF EXISTS `academia`.`oo` $$
CREATE PROCEDURE `academia`.`oo` (out num int)
BEGIN

  select 5 into num; -- Dar un valor a la variable.
END $$

DELIMITER ;

-- La forma de utilizarlo cuando son parámetros de salida:
Set @var=1;

Call oo(@var);

Select @var;
```

Funciones de Usuario

CREATE FUNCTION sp_name ([func_parameter[,...]])
RETURNS type
[characteristic ...] routine_body

func_parameter: param_name type

type: Any valid MySQL data type

characteristic: idem procedures

routine_body: Valid SQL procedure statement

Funciones de Usuario

- Dentro de la funciones podemos definir variables para almacenar datos que se obtengan de una consulta.
  - OJO, serán consultas que devuelvan una sola fila y una sola columna.
    - Ejemplo: select precio from asignatura where id=1;

- Se definen con declare. Tenemos que dar un nombre y un tipo.
  - Declare miVariable float;
  - Para meter un dato dentro de la variable:
    - select precio **into miVariable** from asignatura where id=1;

Ejemplo de Función

```
CREATE function calculo_pedido(codigo char(6)) returns float
READS SQL DATA
begin
DECLARE v_total float;
DECLARE v_iva float;

  SELECT  SUM((preunlin*unilin)*(1-desculin/100)) into v_total
        from lineas
        WHERE numped=codigo
        group by numped;

  SELECT ivaped into v_iva from pedidos
        WHERE numped=codigo;

        RETURN v_total + (v_total * (v_iva/100));
END $$
```

Select calculo_pedido(‘000001’);

Vistas

```
CREATE|ALTER [OR REPLACE] VIEW view_name [(column_list)]
AS select_statement
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

- **WITH CHECK OPTION**: Verifica que cumple la condición select al insertar o modificar, en la vista exclusivamente (local) o también en las vistas que depende (cascada).

Ejemplo Vista

```
CREATE VIEW pedidos_totales AS
SELECT numped, SUM((preunlin*unilin)*(1-desculin/100))
        from lineas
        group by numped;


-- su uso

SELECT * FROM pedidos_totales
```

Triggers

```
CREATE [DEFINER = { user | CURRENT_USER }]
TRIGGER trigger_name trigger_time trigger_event ON tbl_name
FOR EACH ROW trigger_stmt
```

- Trigger_time: Se activa BEFORE o AFTER de la instrucción que lo activó.
- Trigger_event: Evento que activa el Trigger: INSERT, UPDATE, DELETE.
- OLD.col_name y NEW.col_name se refieren a los valores de antes y después de la acción.
- Permite actuar sobre tablas y sobre valores de las tablas afectada, pero no las reglas de negocio. Permite también llamar a procedimientos almacenados.

Ejemplo de Trigger

```
-- CONTROL DE LOG. Si se modifica el precio de un artículo apunto
cuándo y quién lo ha hecho en la tabla auditoria_art.

CREATE TRIGGER trigger_auditoria_art AFTER UPDATE ON articulos
   FOR EACH ROW
   BEGIN
   INSERT INTO auditoria_art (codart, precio_anterior, precio_actual,
          usuario, fecha_modificacion)
   VALUES (OLD.codart, OLD.preunart, NEW.preunart,CURRENT_USER(),
          NOW() );
   END;
```

Transacciones

- Una transacción la podemos definir como un conjunto de operaciones que realizamos sobre una base de datos y que se ejecutan como si fueran una sola.

- Las transacciones nos permite mantener la información almacenada en la BD de forma coherente.

- ¿Qué ocurre en un almacén si emitimos un pedido para sacar material del almacén y no actualizamos las existencias?
  - Podríamos tener:
    - Insert into pedidos(…)
    - Update existencias …
    - ESTAS OPERACIONES DEBEN SER TOMADAS COMO SI FUERAN UNA SOLA. → **atomicidad**.

Transacciones

- Esquema general para trabajar con Transacciones:
  - Inicio de Transacción.
    - Acción 1.
    - Acción 2.
    - …
    - Acción n.
  - Confirmar Transacción.
- Las N acciones se tomarán como una sola.
- Si por cualquier motivo la transacción NO se confirma, los cambios realizados no se efectúan y los datos se quedarán como estuvieran al principio.
- También vamos a poder echar abajo la transacción si se produce alguna circunstancia.

Transacciones

- Normalmente por defecto todos los gestores de BD (incluido MySQL), confirma cada sentencia SQL cada vez que se ejecuta. Están modo autocommit = 1.
- Si queremos trabajar con transacciones tenemos deshabilitar el modo autocommit.
- **Set autocommit = 0**

Transacciones

- Instrucciones para trabajar con transacciones.
  - **Begin**: Inicio de Transacción.
  - **Commit**: Confirmar la transacción. Los cambios se hacen efectivos en la BD.
  - **Rollback**: Cancelar la transacción. No se graba ningún dato en la Bd.

Transacciones

```
mysql> CREATE TABLE CUSTOMER (A INT, B CHAR (20), INDEX (A))
    -> ENGINE=InnoDB;
Query OK, 0 rows affected (0.00 sec)
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO CUSTOMER VALUES (10, 'Heikki');
Query OK, 1 row affected (0.00 sec)
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
mysql> SET AUTOCOMMIT=0;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO CUSTOMER VALUES (15, 'John');
Query OK, 1 row affected (0.00 sec)
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT * FROM CUSTOMER;
+------+--------+
| A    | B      |
+------+--------+
|   10 | Heikki |
+------+--------+
1 row in set (0.00 sec)
mysql>
```