LMD MYSQL

1 Introducción

En este capitulo vamos a analizar la primera categoría de sentencias de SQL , lenguajes de modificación de datos.

Mediante las sentencias que vamos a ver en este tema podemos realizar sobre una base de datos las siguientes operaciones :

- seleccionar registros que cumplan una serie de criterios (operación de selección)
- seleccionar una serie de campos de una tabla (operación de proyección)
- Listar registros de varias tablas (multiplicación, combinación)
- Actualizar los valores de ciertos registros
- Eliminar algunos registros
- Insertar registros en una tabla
- Crear tablas a partir de datos de una tabla existente
- …

Los comandos de SQL característicos de esta parte son :

- SELECT
- INSERT
- UPDATE
- DELETE

Después en este mismo tema veremos otros comandos que también podemos incluir en esta categoría como :

- LOAD DATA INFILE
- REPLACE
- TRUNCATE
- HANDLER

2 Sentencia Select

Mediante esta sentencia podemos realizar consultas :

- selección simples
- selección de varias tablas
    - combinación natural
    - combinación
    - multiplicación

La instrucción select es la fundamental del lenguaje SQL y por tanto de MySQL. Esta instrucción permite realizar consultas sobre la base de datos.

SELECT se utiliza para recuperar las filas seleccionados desde una o más tablas.

La sintaxis de la sentencia SELECT es la siguiente :

***SELECT** [**ALL**|**DISTINCT**|**DISTINCTROW**] ListaDecampos [**INTO** {**OUTFILE**|**DUMPFILE**} 'nombre_fichero' opciones_exportar] **FROM** tablas **WHERE** condición **GROUP BY** camposagrupar **HAVING** condicion1 **ORDER BY** campos **LIMIT** [offset,]columnas] ;*

La ListaDeCampos indica las columnas que quieres recuperar.

SELECT puede utilizarse también para recuperar filas procesadas sin referencia a ninguna tabla. Esto seria similar a utilizar el comando como una calculadora.

Ejemplo

```
SELECT 45,2,3;
```

45	2	3
45	2	3

```
SELECT pow(4,2);
```

pow(4,2)
16

```
SELECT sqrt(64),249/2,249 div 2,truncate(249/2,2);
```

sqrt(64)	249/2	249 div 2	truncate(249/2,2)
8	124.5000	124	124.50

```
SELECT @v1:=1432, 249 mod 2 as resto, truncate(@v1,-1), truncate(@v1,-2);
```

@v1:=1432	resto	truncate(@v1,-1)	truncate(@v1,-2)
1432	1	1430	1400

```
SELECT least(4,5,6,87,9,-2) as menor,greatest(4,5,6,87,9,-2) as mayor
```

menor	mayor
-2	87

```
SELECT @v:='esto es un ejemplo',left(@v,5),right(@v,5)
```

@v:='esto es un ejemplo'	left(@v,5)	right(@v,5)
esto es un ejemplo	esto	emplo

En ListaDeCampos : aqui podemos colocar un * para indicar que saque todos los campos de la tabla, o los campos separados por comas que deseemos ver.

En tablas colocaremos todas las tablas que deseamos utilizar separadas por ",".

Mediante estos elementos podemos conseguir implementar la operación proyección de algebra relacional (sin valores unicos).

$\Pi_{campo1,campo2,...}$(tabla1) ➔ SELECT campo1,campo2 FROM tabla1

Ejemplo

De qué tabla/ s se selecciona la información

SELECT *
FROM alumno;

Todos los atributos de la tabla en el orden de su creación

⇨ *Todos* los atributos de la tabla ALUMNO (todas las tuplas).

num_mat	nombre	ciudad	cod_grupo	precio
1	Juan	Madrid	I11	25000
3	Ana	Leganés	I21	80000
8	María	Leganés	I22	30000
2	Pedro	Getafe	I21	20000
5	Salomé	Madrid	I22	25000

Ejemplo

Proyección

SELECT **nombre, ciudad**
FROM alumno

⇨ Selección de los atributos *nombre* y *ciudad* de la tabla ALUMNO.

nombre	ciudad
Juan	Madrid
Ana	Leganés

Ejemplo

¿Qué realizan las siguientes consultas?

- SELECT nombre FROM t1
- SELECT * FROM t1

Ejemplo

SQL Query Area

```
1 SELECT * FROM ejemplo
```

codigo	nombre	edad	fecha_entrada
1	ramon	30	2005-02-01
2	jose	25	2004-04-08
3	ana	45	2005-01-01
4	luis	26	2004-05-07

SQL Query Area

```
1 SELECT * FROM ejemplo as unica
2 SELECT * FROM ejemplo unica
```

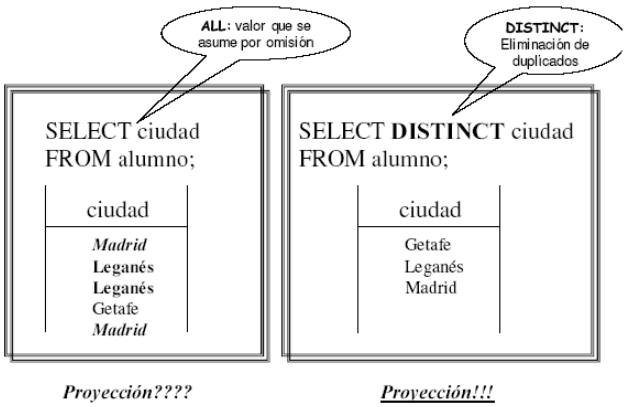
codigo	nombre	edad	fecha_entrada
1	ramon	30	2005-02-01
2	jose	25	2004-04-08
3	ana	45	2005-01-01
4	luis	26	2004-05-07

¿Realmente estas operaciones son idénticas a la proyección?

La respuesta es no, debido a los valores duplicados. Para ello podemos utilizar las cláusulas :

- ALL : no es necesario utilizarla ya que es la que se utiliza por defecto. Muestra todos los registros
- DISTINCT : Mediante esta opción eliminaremos los valores duplicados
- DISTINCTROW : Mediante esta opción eliminamos los registros duplicados.

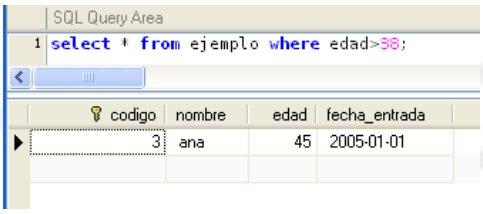
Ejemplo



Todas las palabras claves utilizadas deben ser dadas en el orden exacto presentado más arriba. Por ejemplo, una cláusula HAVING debe hallarse después de cualquier cláusula GROUP BY y antes de cualquier cláusula ORDER BY.

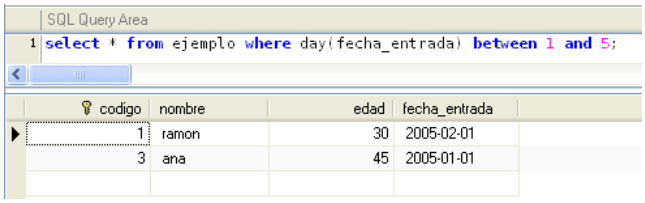
Otra de las operaciones del álgebra relacional era la selección, que consistía en seleccionar filas de una relación que cumplieran determinadas condiciones. Para implementar esta operación en mysql tenemos la cláusula WHERE. Detrás de la cláusula where podéis colocar cualquier expresión valida en SQL.

Ejemplo



Recordamos que en las expresiones podéis utilizar campos calculados, operadores lógicos, funciones, …

Ejemplo



```
SQL Query Area
1 select * from ejemplo where nombre like 'r%' and edad<35;
```

codigo	nombre	edad	fecha_entrada
1	ramon	20	2005-02-01

Recordando los comodines del capitulo anterior

expresión *like*	significado
"g%"	Que empiece por **g**
"%g"	Que termine por **g**
"%g%"	Que tenga una **g**
"_____"	Que tenga cinco caracteres

Los operadores y funciones podéis verlos en el tema anterior.

Una expresión SELECT puede venir dada a través de un alias utilizando AS. El alias se utiliza como el nombre de columna de la expresión y puede ser utilizado con las cláusulas ORDER BY y HAVING (nunca en la cláusula WHERE). Estos alias de columnas serán utilizados como encabezados de columna.

Ejemplo

```
SQL Query Area
1 SELECT nombre, nombre as los_nombres from ejemplo;
```

nombre	los_nombres
ramon	ramon
jose	jose
ana	ana
luis	luis

Ejemplo

No se permite utilizar los alias de columnas en cláusulas WHERE, porque el valor de la columna puede no estar determinado cuando se ejecute la cláusula WHERE.

```
SQL Query Area
1 SELECT nombre, nombre as los_nombres from ejemplo order by los_nombres;
2 SELECT nombre, nombre as los_nombres from ejemplo where los_nombres like 'a%';
```

La palabra clave AS es opcional podemos omitirla.

La cláusula FROM table_references indica las tablas desde las que se recuperan las filas. Si nombras más de una tabla, estás realizando una consulta de varias tablas.

Ejemplo

SQL Query Area

```
1 SELECT ejemplo.nombre, a.nombre FROM ejemplo, ejemplo as a
```

nombre	nombre
ramon	ramon
jose	ramon
ana	ramon
luis	ramon
ramon	jose
jose	jose
ana	jose
luis	jose
ramon	ana
jose	ana
ana	ana
luis	ana
ramon	luis
jose	luis
ana	luis
luis	luis

Producto

Hablaremos mas de las consultas de varias tablas cuando nos ocupemos de las combinaciones.

Para cada tabla especificada, puedes especificar opcionalmente un alias al igual que con los campos. Al igual que con los campos la clausula AS es completamente opcional.

Puedes referirte a una columna como :

- col_name
- tbl_name.col_name : si el campo esta en varias tables de las seleccionadas
- db_name.tbl_name.col_name : si la tabla no se encuentra en la base de datos activa

No precisas especificar prefijos de base de datos o tabla en una sentencia SELECT a no ser que sea una referencia ambigua.

Ejemplo

SQL Query Area

```
1 SELECT unica.nombre NOMBRES,edad EDADES from ejemplo unica;
```

NOMBRES	EDADES
ramon	30
jose	25
ana	45
luis	26

La clausula ORDER BY podéis utilizarla para realizar las ordenaciones. El orden inicial de los registros obtenidos por un SELECT no guarda más que una relación respecto al orden en el que fueron introducidos. Para ordenar en base a criterios más interesantes, se utiliza la cláusula ORDER BY. En esa cláusula se coloca una lista de campos que indica la forma de ordenar. Se ordena primero por el primer campo de la lista, si hay coincidencias por el segundo, si ahí también las hay por el tercero, y así sucesivamente.
Se puede colocar las palabras ASC O DESC (por defecto se toma ASC) detrás de los campos. Esas palabras significan en ascendente (de la A a la Z, de los números pequeños a los grandes) o en descendente (de la Z a la a, de los números grandes a los pequeños) respectivamente.

Ejemplo

SQL Query Area

```
select * from ejemplo order by edad
```

codigo	nombre	edad	fecha_entrada
2	jose	25	2004-04-08
4	luis	26	2004-05-07
1	ramon	30	2005-02-01
3	ana	45	2005-01-01

SQL Query Area

```
select * from ejemplo order by edad DESC
```

codigo	nombre	edad	fecha_entrada
3	ana	45	2005-01-01
1	ramon	30	2005-02-01
4	luis	26	2004-05-07
2	jose	25	2004-04-08

SQL Query Area

```
select * from ejemplo where year(fecha_entrada)=year(current_date()) order by fecha_entrada DESC;
```

codigo	nombre	edad	fecha_entrada
1	ramon	30	2005-02-01
3	ana	45	2005-01-01

La cláusula LIMIT puede ser utilizada para restringir el número de filas retornados por la sentencia SELECT. LIMIT toma uno o dos argumentos numéricos. Los argumentos deben ser constantes enteras. Si se dan dos argumentos, el primero especifica el número de orden (offset) de la primera fila a retornar, el segundo especifica el número máximo de filas a retornar. El número de la fila inicial es 0 (no 1). Si se da un solo argumento indica el numero de filas a retornar desde la primera.

Ejemplo

SQL Query Area

```
select * from ejemplo limit 1;
```

codigo	nombre	edad	fecha_entrada
1	ramon	30	2005-02-01

SQL Query Area

```
select * from ejemplo limit 1,3;
```

codigo	nombre	edad	fecha_entrada
2	jose	25	2004-04-08
3	ana	45	2005-01-01
4	luis	26	2004-05-07

SQL Query Area

```
select * from ejemplo limit 2;
```

codigo	nombre	edad	fecha_entrada
1	ramon	30	2005-02-01
2	jose	25	2004-04-08

La forma SELECT...INTO OUTFILE 'file-name' de SELECT escribe las filas seleccionadas en un archivo. El archivo se crea en el host del servidor y no puede existir anteriormente (entre otras cosas, esto previene a las tablas de la base de datos y archivos tales como 'etc/passwd' de ser destruidas). Lógicamente debes tener privilegios en el servidor para crear ficheros.

Ejemplo

```
SQL Query Area
1 select * into outfile 'a.txt' from ejemplo;
2
```

El directorio de salida por defecto es data\nombre_vuestro_esquema.

La forma SELECT...INTO OUTFILE es utilizado principalmente para permitirte volcar rápidamente una base de datos en una máquina de servidor. Si quieres crear el archivo en cualquier otro host que el del servidor, no puedes utilizar SELECT...INTO OUTFILE. En este caso deberías utilizar un programa cliente.

Para realizarlo a través de nuestro cliente se utilizaría la opción de File->Export Resultset

SELECT ... INTO OUTFILE es el complemento a LOAD DATA INFILE.

Las opciones de exportación son las mismas que para este comando:

```
[FIELDS
    [TERMINATED BY '\t']
    [[OPTIONALLY] ENCLOSED BY '']
    [ESCAPED BY '\\' ]
]
  [LINES
   [TERMINATED BY '\n']
   ]
```

Ambas cláusulas son opcionales, pero FIELDS debe preceder LINES si se especifican ambas.

Si especificas una cláusula FIELDS, cada una de las subcláusulas (TERMINATED BY, [OPTIONALLY] ENCLOSED BY, y ESCAPED BY) son también opcionales, excepto cuando debas especificar al menos una de ellas.

Si no especificas FIELDS, el valor por defecto sería el mismo que si hubieras escrito esto:

FIELDS TERMINATED BY ‘\t’ ENCLOSED BY ‘’ ESCAPED BY ‘\\’

Si no especificas LINES, el valor por defecto es el mismo que si hubieras escrito esto:

LINES TERMINATED BY ‘\n’

Los valores por defecto de SELECT...INTO OUTFILE actúen como sigue en el momento de escribir una salida:

- Escribe tabulaciones entre campos.
- No encierra los campos entre caracteres.
- Utiliza '\' para escapar casos de tabulaciones, saltos de línea o '\' que tengan lugar en los valores de los campos, es decir si dentro de un campo ocurre uno de estos casos se sustituye por una \.
- Escribe saltos de línea al final de la línea.

Para escribir FIELDS ESCAPED BY '\\', debes especificar dos contrabarras para que el valor se lea como una contrabarra simple.

Cuando utilizas SELECT ... INTO OUTFILE con LOAD DATA INFILE para leer y escribir posteriormente datos de una base de datos, las opciones de manejo de campo y línea para ambos comandos deben coincidir. En cualquier otro caso, LOAD DATA INFILE no interpretará propiamente los contenidos del archivo.

Supón que utilizas SELECT...INTO OUTFILE para escribir un archivo con los campos delimitados por comas:

```
SQL Query Area
1 select * into outfile 'c:\\salida.txt' fields terminated by ',' from ejemplo;
```

El archivo de salida seria el siguiente :

```
1,ramon,30,2005-02-01
2,jose,25,2004-04-08
3,ana,45,2005-01-01
4,luis,26,2004-05-07
```

Al leer el archivo delimitado por coma, la sentencia correcta sería:

```
SQL Query Area
1 load data infile 'c:\\salida.txt' into table ejemplo1 fields terminated by ',';
```

Si en cambio trataras de leer el archivo con la siguiente sentencia, no trabajaría debido a que instruye LOAD DATA INFILE para localizar tabulaciones entre campos:

```
SQL Query Area
1 load data infile 'c:\\salida.txt' into table ejemplo1 fields terminated by '\t';
```

El resultado que parecería obtenerse es que cada línea se interpretaría como un solo campo.

Podemos concluir indicando :

- La cláusula FIELDS se refiere a las opciones de cada columna:
  - TERMINATED BY 'carácter': nos permite elegir el carácter delimitador que se usará para separar cada columna. Por defecto, el valor que se usa es el tabulador, '\t'

- [OPTIONALLY] ENCLOSED BY 'carácter': sirve para elegir el carácter usado para entrecomillar cada columna. Por defecto no se entrecomilla ninguna columna, pero podemos elegir cualquier carácter. Si se añade la palabra OPTIONALLY sólo se entrecomillarán las columnas de char y varchar.
- ESCAPED BY 'carácter': sirve para indicar el carácter que se usará para escapar aquellos caracteres que pueden dificultar la lectura posterior del fichero. Por ejemplo, si teminamos las columnas con ',' y no las entrecomillamos, un carácter ',' dentro de una columna de texto se interpretará como un separador de columnas. Para evitar esto se puede escapar esa coma con otro carácter. Por defecto se usa el carácter '\'.

➢ La cláusula LINES se refiere a las opciones para cada fila:

- STARTING BY 'carácter': permite seleccionar el carácter para comenzar cada línea. Por defecto no se usa ningún carácter para ello.
- TERMINATED BY 'carácter': permite elegir el carácter para terminar cada línea. Por defecto es el retorno de línea

Ejemplo

FIELDS [OPTIONALLY] ENCLOSED BY controla los campos enmarcados. Para las salidas (SELECT...INTO OUTFILE), si omites la palabra OPTIONALLY, todos los campos serán encerrados por el carácter indicado. Un ejemplo de tal salida (utilizando una coma como delimitador de campo) , se puede ver aquí:

“1”, “a string”,”100.20”
“2”, “a string containing a , comma”,”102.20”
“3”, “a string containing a \” quote”,”102.20”
“4”, “a string containing a \”, quote and comma”,”102.20”

Si especificas OPTIONALLY, el carácter ENCLOSED BY se utilizará sólo para encerrar campos del tipo CHAR y VARCHAR:

1, “a string”,100.20
2, “a string containing a , comma”,102.20
3, “a string containing a \” quote”,102.20
4, “a string containing a \”, quote and comma”,102.20

Recordemos los literales :

Secuencia	Significado
\0	Carácter nulo de ASCII (NULL)
\’	Carácter de comilla simple.
\”	Caracter de comillas dobles.
\b	Un carácter de borrado (backspace)
\n	Carácter de nueva línea

Secuencia	Significado
\r	Carácter de retorno de carro
\t	Carácter de tabulación
\z	ASCII 26 (Control-Z). Este carácter es particularmente importante, porque indica el final de archivo (END-OF-FILE) en Windows., de modo que podría causar problemas en caso de utilizar comandos del tipo
\\	Carácter de contrabarra

Ejemplo

```
SQL Query Area
1 select * into outfile 'c:\\salida.txt' fields terminated by '\n' lines terminated by '\n\n\n' from ejemplo;
```

```
1
ramon
30
2005-02-01


2
jose
25
2004-04-08


3
ana
45
2005-01-01


4
luis
26
2004-05-07
```

Ejemplo

```
SQL Query Area
1 select * into outfile 'c:\\salida.txt'
2 fields terminated by '\t' enclosed by '"'
3 lines terminated by '\n\n\n' from ejemplo;
```

```
"1"	"ramon"	"30"	"2005-02-01"


"2"	"jose"	"25"	"2004-04-08"


"3"	"ana"	"45"	"2005-01-01"


"4"	"luis"	"26"	"2004-05-07"
```

Ejemplo

SQL Query Area

```
1 select * into outfile 'c:\\salida.txt'
2 fields terminated by 'c' enclosed by '"'escaped by '\\'
3 lines terminated by '\n\n\n' from ejemplo;
```

```
"1"   "ramon"     "30"  "2005-02-01"

"2"   "jose"      "25"  "2004-04-08"

"3"   "ana" "45"  "2005-01-01"

"4"   "luis"      "26"  "2004-05-07"
```

Ejemplo

SQL Query Area

```
1 select * into outfile 'c:\\salida.txt'
2 fields terminated by 'a' escaped by '\\'
3 lines terminated by '\n\n\n' from ejemplo;
```

```
1ar\amona30a2005-02-01

2ajosea25a2004-04-08

3a\an\aa45a2005-01-01

4aluisa26a2004-05-07
```

2.1 Consultas de agrupamiento o de totales

Vamos a ver como utilizando mysql podemos realizar consultas donde vamos a realizar agrupaciones por uno o mas campos. Ademas si a esto unimos el poder utilizar las funciones de agregado conseguimos calcular un valor por cada grupo.

SQL permite la posibilidad de calcular funciones en grupos de tuplas utilizando la cláusula GROUP BY. Para formar estos grupos, se utilizan todos los atributos que aparecen en la cláusula GROUP BY, y cada una de las tuplas que tienen el mismo valor en cada uno de los atributos que se especifican en la cláusula GROUP BY se colocan en el mismo grupo.

SQL incluye algunas funciones para calcular totales. Los ejemplos mas claros serian :

- Promedio: AVG()
- Mínimo: MIN ()
- Máximo: MAX()
- Total: SUM ()
- Cuenta: COUNT ()

A estas operaciones se les denomina funciones de agregación, ya que operan sobre grupos de tuplas, y el resultado de estas funciones es un valor único.

Para realizar grupos es muy sencillo, simplemente los campos para formar los grupos se colocan en la clausula GROUP BY separados por comas.

Ejemplo

Suponer que tengo la tabla ejemplo con los siguientes valores :

codigo	nombre	edad	fecha_entrada
1	ramon	30	2005-02-01
2	jose	25	2004-04-08
3	ana	45	2005-01-01
4	luis	26	2004-05-07
5	jose	25	2005-01-01
6	ramon	25	2004-05-01
7	jose	45	2004-04-08

Realicemos agrupaciones:

SQL Query Area

```
select * from ejemplo group by nombre;
```

codigo	nombre	edad	fecha_entrada
3	ana	45	2005-01-01
2	jose	25	2004-04-08
4	luis	26	2004-05-07
1	ramon	30	2005-02-01

SQL Query Area

```
select * from ejemplo group by edad, nombre;
```

codigo	nombre	edad	fecha_entrada
2	jose	25	2004-04-08
6	ramon	25	2004-05-01
4	luis	26	2004-05-07
1	ramon	30	2005-02-01
3	ana	45	2005-01-01
7	jose	45	2004-04-08

SQL Query Area

```
select * from ejemplo group by nombre, edad;
```

codigo	nombre	edad	fecha_entrada
3	ana	45	2005-01-01
2	jose	25	2004-04-08
7	jose	45	2004-04-08
4	luis	26	2004-05-07
6	ramon	25	2004-05-01
1	ramon	30	2005-02-01

SQL Query Area

```
select * from ejemplo group by year(fecha_entrada) order by nombre;
```

codigo	nombre	edad	fecha_entrada
2	jose	25	2004-04-08
1	ramon	30	2005-02-01

Cuando realizamos este tipo de consultas debemos observar que campos colocamos:

```
SQL Query Area
1 select year(fecha_entrada) from ejemplo group by year(fecha_entrada);
```

year(fecha_entrada)
2004
2005

Ejemplo

Supongamos que tenemos la siguiente tabla :

```
SQL Query Area
1 SELECT * FROM unica;
```

id	nombre	edad	casado	fecha_entrada
1	ramon	28	1	2005-01-14
2	ana	25	0	NULL
3	rosa	40	1	1998-01-05
4	ana	25	1	2000-04-01
5	federico	18	0	NULL
6	ramon	30	1	2000-12-04

Podemos realizar una consulta donde necesitemos realizar una restricción de filas, para ello lo mas cómodo es utilizar la cláusula WHERE.

```
SQL Query Area
1 SELECT edad FROM unica where casado is true group by edad
```

edad
25
28
30
40

Las funciones de totales podemos utilizarlas tanto si hemos utilizado la clausula group by como si no. Si utilizas una función de agrupación en una sentencia que no contenga una cláusula GROUP BY, equivale a agrupar todas las filas.

Ejemplo

En esta consulta tenemos el numero total de elementos de la tabla unica.

```
SQL Query Area
1 SELECT count(*) as numero_elementos_totales from unica
```

numero_elementos_totales
6

En la siguiente consulta contamos los elementos del campo fecha_entrada, y los nulos no se contarian. Por eso el resultado difiere.

```
SQL Query Area
1 SELECT count(fecha_entrada) as numero_elementos_totales from unica
```

numero_elementos_totales
4

Ejemplo

```
SELECT count(fecha_entrada) as numero_elementos_totales, nombre from unica group by nombre
```

numero_elementos_...	nombre
1	ana
0	federico
2	ramon
1	rosa

Las funciones de agregación son funciones que toman una colección (un conjunto o multiconjunto) de valores como entrada y producen un único valor como salida.

Hay casos en los que se deben eliminar los duplicados antes de calcular una función de agregación. Para eliminar duplicados se utiliza la palabra clave distinct en la expresión de agregación.

COUNT(DISTINCT 'NOMBRE_CAMPO')

```
SELECT * FROM pruebas.ejemplo e
```

codigo	nombre	edad	fecha_entrada
1	ramon	30	2005-02-01
2	jose	25	2004-04-08
3	ana	45	2005-01-01
4	luis	26	2004-05-07
5	jose	25	2005-01-01
6	ramon	25	2004-05-01
7	jose	45	2004-04-08

```
select COUNT(nombre) from ejemplo;
```

COUNT(nombre)
7

```
select COUNT(DISTINCT nombre) from ejemplo;
```

COUNT(DISTINC...
4

Con la palabra DSTINCT no podemos colocar el asterisco, es necesario colocar el nombre de un campo, o podemos colocar una serie de campos separados con comas o incluso podemos colocar expresiones.

Ejemplo

```
select COUNT(DISTINCT nombre,edad) from ejemplo;
```

COUNT(DISTINC...
6

```
SQL Query Area
1 select COUNT(DISTINCT day(fecha_entrada)) as 'mismo dia' from ejemplo;
```

mismo dia
3

```
SQL Query Area
1 select COUNT(DISTINCT day(fecha_entrada),year(fecha_entrada)) as 'mismo dia' from ejemplo;
```

mismo dia
4

A veces es más útil establecer una condición que se aplique a los grupos que una que se aplique a las tuplas. Para expresar este tipo de consultas se utiliza la cláusula having de SQL. Los predicados de la cláusula having se aplican después de la formación de grupos, de modo que se pueden usar las funciones de agregación.

La cláusula HAVING se ejecuta después de formar los grupos luego solamente puede hacer referencia a estos o a los campos utilizados.

Ejemplo

```
SQL Query Area
1 SELECT nombre FROM ejemplo where edad<40;
```

nombre
ramon
jose
luis
jose
ramon

Restriccion de filas

```
SQL Query Area
1 SELECT nombre,count(edad<30) FROM ejemplo group by nombre;
```

nombre	count(edad<30)
ana	1
jose	3
luis	1
ramon	2

Restriccion de grupos

```
SQL Query Area
1 SELECT nombre,count(edad<30) FROM ejemplo group by nombre having count(edad<30)>2;
```

nombre	count(edad<30)
jose	3

Si en una misma consulta aparece una cláusula where y una cláusula having, se aplica primero el predicado de la cláusula where. Las tuplas que satisfagan el predicado de la cláusula where se colocan en grupos según la cláusula group by. La cláusula having, si existe, se aplica entonces a cada grupo; los grupos que no satisfagan el predicado de la cláusula having se eliminan. La cláusula select utiliza los grupos restantes para generar las tuplas resultado de la consulta.

Ejemplo

```
SQL Query Area
1 SELECT nombre,count(edad<30) FROM ejemplo where length(nombre)>4 group by nombre having count(edad<30)>1;
```

nombre	count(edad<30)
ramon	2

Los grupos que podemos utilizar en HAVING pueden ser distintos a los utilizados en la clausula SELECT o a los utilizados en la clausula GROUP BY.

Otras de las funciones de totales que podemos utilizar es la del promedio. Cuidado con esta función por que lo que calcula es el promedio del grupo, no el promedio la filas. La sintasis es :

AVG(expr)

Ejemplo

```
SQL Query Area
1 SELECT avg(edad) from ejemplo;
```

avg(edad)
31.5714

```
SQL Query Area
1 SELECT avg(edad>35) from ejemplo;
```

avg(edad>35)
0.2857

También podemos utilizar el operador DISTINCT.

Ejemplo

```
SQL Query Area
1 SELECT avg(DISTINCT edad) from ejemplo;
```

avg(DISTINCT edad)
31.5000

Esta consulta la podemos colocar como una consulta de una consulta (consultas anidadas, las veremos mas adelante).

```
SQL Query Area
1 SELECT avg(edad) from (select distinct edad from ejemplo) p;
```

avg(edad)
31.5000

Otras funciones que podemos utilizar son las de calcular los valores maximos y minimos:

MIN(expr)

MAX(expr)

Pueden tomar un argumento de cadena: en tales casos retornan el mínimo o máximo valor de cadena. Las cadenas ***seran mas grandes en funcion de su posición en el codigo ASCII no de su longitud***.

Ejemplo

```
SQL Query Area
1 SELECT max(edad) from ejemplo;
```

max(edad)
45

```
SQL Query Area
1 SELECT max(nombre) from ejemplo;
```

max(nombre)
ramon

```
SQL Query Area
1 SELECT min(nombre) from ejemplo;
```

min(nombre)
ana

Ejemplo

En este ejemplo vamos a utilizar un concepto nuevo que no hemos visto, introducir una consulta de totales como parámetro de restricción.

```
SQL Query Area
1 select nombre from ejemplo where length(nombre)=(SELECT min(length(nombre)) from ejemplo);
```

nombre
ana

En algunas ocasiones podemos utilizar varias funciones de totales juntas.

Ejemplo

```
SQL Query Area
1 select nombre,min(edad),max(edad) from ejemplo group by nombre;
```

nombre	min(edad)	max(edad)
ana	45	45
jose	25	45
luis	26	26
ramon	25	30

Otra de las funciones de totales es la de suma.

SUM(expr)

Retorna la suma de expr. Si el resultado de retorno no tiene filas, se retorna NULL.

Ejemplo

```
select nombre,sum(edad) from ejemplo group by nombre;
```

nombre	sum(edad)
ana	45
jose	95
luis	26
ramon	55

```
select nombre,sum(edad),count(edad),avg(edad) from ejemplo group by nombre;
```

nombre	sum(edad)	count(edad)	avg(edad)
ana	45	1	45.0000
jose	95	3	31.6667
luis	26	1	26.0000
ramon	55	2	27.5000

```
select nombre, least(sum(edad),avg(edad)) from ejemplo group by nombre;
```

nombre	least(sum(edad),avg(edad))
ana	45
jose	31.666666666
luis	26
ramon	27.500000000

Tenemos otra función que nos permite calcular la desviación estándar. Esta función no esta definida en el ANSI de SQL.

STD(expr)

STDDEV(expr)

Retorna la desviación estándar de expr.

La forma STDDEV() de esta proporcionada por compatibilidad con Oracle.

Ejemplo

```
select std(edad) from ejemplo;
```

std(edad)
8.6496

Después existen una serie de funciones de totales que son menos utilizadas :

- BIT_OR(expr): Retorna el OR bit a bit de todos los bits en expr. El cálculo se realiza con precisión de BIGINT (64 bits)
- BIT_AND(expr) : Retorna el AND bit a bit de todos los bits en expr. El cálculo se realiza con precisión BIGINT (64 bits).
- BIT_XOR(expresión): Retorna el XOR bit a bit de todos los bits en expr. El cálculo se realiza con precisión BIGINT (64 bits).
- VARIANCE(exp.) : Calcula la varianza.
- GROUP_CONCAT([DISTINCT] expresion1, expresion2,.. [ORDER BY columnas][ASC|DESC][SEPARATOR str_val]): Esta funcion devuelve un string con los valores concatenados del grupo. El separador que utiliza por defecto es la ",".

Ejemplo

```
select group_concat(nombre) from ejemplo;
```

group_concat(nombre)
ramon,jose,ana,luis,jose,ramon,jose

```
select group_concat(DISTINCT nombre) from ejemplo;
```

group_concat(DISTINCT nombre)
ramon,jose,ana,luis

```
select group_concat(DISTINCT nombre SEPARATOR ';') as nombres from ejemplo;
```

nombres
ramon;jose;ana;luis

Hay que tener en cuenta características de Mysql que son diferentes al estándar :

- MySQL ha extendido el uso de GROUP BY. Puedes utilizar columnas o cálculos en las expresiones SELECT que no aparecen en la parte GROUP BY. Esto se mantiene para cualquier valor posible del grupo. Puedes utilizar esto para conseguir una mayor realización evitando la ordenación y agrupación de elementos innecesarios. En ANSI SQL, deberías añadir en la cláusula GROUP BY todos los campos utilizados en SELECT. No utilices esta funcionalidad si las columnas que omites en el GROUP BY no son únicas en el grupo! Los resultados son absurdos

Ejemplo

Esta operación podemos realizarla en MySQL y en SQL deberiamos colocar el campo month(fecha_entrada) en la clausula select.

```
SELECT avg(edad) FROM ejemplo group by month(fecha_entrada);
```

avg(edad)
35.0000
30.0000
35.0000
25.5000

El poder realizar esto, puede producir alguna consulta incorrecta.

```
SELECT edad FROM ejemplo group by month(fecha_entrada);
```

edad
45
30
25
26

- Si utilizas MySQL v3.22 (o anterior) o si estás tratando de seguir el ANSI SQL, no puedes utilizar las expresiones en las cláusulas GROUP BY o ORDER BY. Puedes trabajar sobre esta limitación utilizando un alias para la expresión.

Ejemplo

En el siguiente ejemplo podemos ver como utilizamos una expresión en HAVING.

```
SELECT nombre nombres from ejemplo group by nombres having avg(edad)>40;
```

nombres
ana

En SQL deberíamos realizar lo siguiente.

```
SELECT nombre nombres,avg(edad) media from ejemplo group by nombres having media>40;
```

nombres	media
ana	45.0000

- En el estándar de sql no se permiten colocar ciertos operadores dentro de las sentencias de agregación. Por ejemplo una sentencia que hemos utilizado en los ejemplos.

```
SELECT avg(edad>35) from ejemplo;
```

avg(edad>35)
0.2857

Esta consulta daría error por ejemplo en SQL Server, porque no te permite introducir el operador > dentro de la sentencia count. Nosotros podemos colocarlo pero no realiza lo que suponemos ¿Por qué? Queremos calcular la media aritmética de los registros cuya edad es superior a 35. El resultado es incorrecto. La sentencia SQL correctamente escrita seria la siguiente:

```
select avg(edad) from ejemplo where edad>35;
```

avg(edad)
45.0000

¿De donde sale el resultado 0.2857?. La explicación la tenemos en la siguiente figura:

```
select sum(edad>35),count(edad>35),sum(edad>35)/count(edad>35) from ejemplo
```

sum(edad>35)	count(edad>35)	sum(edad>35)/count(edad>35)
2	7	0.2857

Para evitar estos problemas llegamos a la siguiente conclusión :

Dentro de las funciones de agregado no debéis colocar expresiones de restricción de filas

```
select sum(edad>35) from ejemplo;
sustituirla por
select sum(edad) from ejemplo where edad>35;
```

Podemos colocar otras expresiones sin ningún tipo de problema.

Ejemplo

Vamos a realizar algunos ejemplos sobre la siguiente tabla :

```
SELECT * FROM coches c
```

id	nombre	nuevo	comprado	precio
1	uno	1	2005-01-01	5000
2	dos	0	2004-08-07	8000
3	tres	1	2000-12-04	12000
4	cuatro	1	1987-12-10	24000

Explicar cada una de las siguientes consultas :

```
select sum(length(nombre)) from coches
```

sum(length(nombre))
16

```
select variance(precio) varianza ,std(precio) desviacion_tipica ,pow(std(precio),2) from coches;
```

varianza	desviacion_tipica	pow(std(precio),2)
52187500	7224.0916383999	52187500

```
select sum(if(year(comprado)=2005,1,2)) from coches
```

sum(if(year(comprado)=2005,1,2))
7

```
select day(comprado),sum(precio) from coches group by day(comprado);
```

day(comprado)	sum(precio)
1	5000
4	12000
7	8000
10	24000

2.2 Combinación de tablas

Cuando coloqueis mas de una tabla en una consulta podemos encontrarnos en los siguientes casos:

- producto de las tablas : cuando coloquemos las tablas sin realizar ninguna restricción o combinación
- combinación natural : cuando coloquemos las tablas y utilicemos el operador de comparación para relacionar los campos.
- Combinación : cuando coloquemos las tablas y utilicemos una expresión para relacionar los campos.

Cuidado cuando tengamos mas de una tabla en las consultas porque debéis colocar los nombres de los campos correctamente, es decir, si existe campos iguales en las tablas, tendréis que colocar el nombre de la tabla precediendo al del campo :

Nombre_tabla.Nombre_campo

Ejemplo

Para los siguientes ejemplos vamos a trabajar con la siguiente base de datos:

```
SELECT * FROM padres;
```

dni	nombre	direccion	telefono	movil
45789456	padre1	direccion12	999-89-89-89	698-78-45-54
56895699	padre2	direccion1	NULL	669-89-89-77

```
SELECT * FROM madres;
```

dni	nombre	direccion	telefono	movil
10101010	madre1	direccion1	942-20-23-24	666-45-45-78
20202020	madre2	direccion2	942-22-22-22	NULL
30303030	madre3	direccion3	978-45-45-45	687-98-89-78

```
SELECT * FROM niños;
```

id	nombre	edad	nacimiento	padre	madre
1	niño1	10	1995-01-01	56895699	10101010
2	niño2	8	1997-01-05	45789456	NULL

Tenemos realizada una relación con integridad referencial en cascada en los siguientes campos :

Niños_padre ➔ dni_padres

Niños_madre ➔ dni_madres

Si realizamos la siguiente consulta

```
SELECT niños.nombre,padres.nombre from niños,padres
```

nombre	nombre
niño1	padre1
niño2	padre1
niño1	padre2
niño2	padre2

Debemos colocar los nombres de las tablas para indicar a que tabla pertenece cada campo.

2.2.1 Producto tablas

En el ejemplo anterior el resultado es el producto de la tabla niños por la tabla padres. Lo mismo pasa con tres tablas :

```
SQL Query Area
1 SELECT niños.nombre,padres.nombre,madres.* from niños,padres,madres
```

nombre	nombre	dni	nombre	direccion	telefono	movil
niño1	padre1	10101010	madre1	direccion1	942-20-23-24	666-45-45-78
niño2	padre1	10101010	madre1	direccion1	942-20-23-24	666-45-45-78
niño1	padre2	10101010	madre1	direccion1	942-20-23-24	666-45-45-78
niño2	padre2	10101010	madre1	direccion1	942-20-23-24	666-45-45-78
niño1	padre1	20202020	madre2	direccion2	942-22-22-22	NULL
niño2	padre1	20202020	madre2	direccion2	942-22-22-22	NULL
niño1	padre2	20202020	madre2	direccion2	942-22-22-22	NULL
niño2	padre2	20202020	madre2	direccion2	942-22-22-22	NULL
niño1	padre1	30303030	madre3	direccion3	978-45-45-45	687-98-89-78
niño2	padre1	30303030	madre3	direccion3	978-45-45-45	687-98-89-78
niño1	padre2	30303030	madre3	direccion3	978-45-45-45	687-98-89-78
niño2	padre2	30303030	madre3	direccion3	978-45-45-45	687-98-89-78

Ahora estamos realizando la combinación de todos los registros de padres con cada uno de los de niños y después cada combinación de estas dos tablas se combina con cada registro de madres.

Padres X madres X niños

Luego cuando queramos realizar el producto de algunas tablas simplemente debeis colocarlas en la sentencia from separadas por comas.

2.2.2 Combinaciones naturales

Una combinación natural de dos tablas es una composición entre las mismas basada en la coincidencia exacta de dos columnas, una de cada tabla. La combinación forma parejas de filas haciendo coincidir los contenidos de las columnas relacionadas.

Es decir una combinación lo que va a realizar es la selección de algunos registros del producto cartesiano en funciona de una operación de restricción.

Si la operación de restricción es una comparación entre los campos relacionados de las dos tablas se denomina ***combinación natural***. Si la restricción se realiza con una expresión mas compleja se denomina ***combinación***.

Ejemplo

```
SQL Query Area
1 select niños.nombre,padres.nombre from niños,padres where padre=dni
```

nombre	nombre
niño2	padre1
niño1	padre2

```
SQL Query Area
1 select niños.nombre,padres.nombre from niños,padres;
```

nombre	nombre
niño1	padre1
niño2	padre1
niño1	padre2
niño2	padre2

combinacion natural

```
SQL Query Area
1 select niños.nombre,padres.nombre from niños,padres where padre<>dni;
```

nombre	nombre
niño1	padre1
niño2	padre2

combinacion

En este punto vamos a tratar solamente las combinaciones naturales.

Las combinaciones podemos realizarlas utilizando la cláusula WHERE o utilizando la clausula JOIN.

Ejemplo

Combinación de dos tablas :

```
SQL Query Area
1 select n.nombre,p.nombre from niños n,padres p where dni=padre;
```

nombre	nombre
niño2	padre1
niño1	padre2

Combinación de tres tablas :

```
SQL Query Area
1 select n.nombre,p.nombre,m.nombre from niños n,padres p, madres m where p.dni=padre and m.dni=madre;
```

nombre	nombre	nombre
niño1	padre2	madre1

Cuando tenemos combinaciones naturales de tres tablas utilizamos el operador AND para unir en la restricción los campos.

Para realizar las combinaciones Mysql nos proporciona un método mas optimo que la cláusula WHERE, la cláusula JOIN.

El comando JOIN para una combinación tiene la siguiente sintaxis :

Tabla1 [NATURAL] [LEFT OUTER|RIGHT OUTER|INNER] [JOIN|STRAIGHT_JOIN] tabla2 [ON tabla1.campo1 operadorComparacion tabla2.campo2] | [USING (campo1,campo2,..)]

Para realizar la operación de producto cartesiano simplemente debéis colocar la cláusula JOIN.

```
SQL Query Area
1 select n.nombre,p.nombre from niños n join padres p;
```

nombre	nombre
niño1	padre1
niño2	padre1
niño1	padre2
niño2	padre2

Para realizar las combinaciones tenemos dos opciones :

- no colocar nada o colocar la palabra clave INNER : Esta es una combinación interna. Es decir las que estamos analizando en este punto.
- Colocar la palabra clave LEFT OUTER (o solamente LEFT) o RIGHT OUTER (o solamente RIGHT): Esta es una combinación externa por la derecha o por la izquierda.

Ejemplo

```
select n.nombre,m.nombre from niños n join madres m on dni=madre;
```

nombre	nombre
niño1	madre1

```
select n.nombre,m.nombre from niños n left outer join madres m on dni=madre;
```

nombre	nombre
niño1	madre1
niño2	NULL

```
select n.nombre,m.nombre from niños n right join madres m on dni=madre;
```

nombre	nombre
niño1	madre1
NULL	madre2
NULL	madre3

La palabra clave STRAIGHT_JOIN es idéntico a JOIN, excepto que la tabla izquierda se lee siempre antes que la tabla derecha. Esto solamente tiene sentido en la optimización de consultas.

Para colocar las condiciones de combinación podemos utilizar las cláusulas ON o USING :

- Con la cláusula ON colocamos a continuación las condiciones de combinación
- Con la cláusula USING colocamos únicamente los campos a utilizar para combinaciones naturales cuando los campos en ambas tablas se llaman igual.

Ejemplo

```
select * from uno,dos join using(campo1);
```

```
SELECT * FROM coches c join coches using (id);
```

id	nombre	nuevo	comprado	precio	nombre	nuevo	comprado	precio
1	uno	1	2005-01-01	5000	uno	1	2005-01-01	5000
2	dos	0	2004-08-07	8000	dos	0	2004-08-07	8000
3	tres	1	2000-12-04	12000	tres	1	2000-12-04	12000
4	cuatro	1	1987-12-10	24000	cuatro	1	1987-12-10	24000

Cuando queremos combinar mas de dos tablas simplemente colocaremos varios on consecutivos.

```
SQL Query Area
1 SELECT p.dni padre,m.dni madre,n.nombre
2 from niños n join padres p on p.dni=padre
3 join madres m on m.dni=madre;
```

padre	madre	nombre
56895699	10101010	niño1

```
SQL Query Area
1 SELECT p.dni padre,m.dni madre,n.nombre
2 from niños n join padres p join madres m
3 on p.dni=padre and m.dni=madre;
4
```

padre	madre	nombre
56895699	10101010	niño1

Luego para las combinaciones naturales internas utilizamos la palabra INNER delante de JOIN o solamente JOIN. A continuación colocamos las condiciones utilizando ON y si los campos se llaman igual se puede utilizar el USING.

Las combinaciones externas nos permiten seleccionar registros de las tablas que no cumplan las condiciones de combinación :

- combinación externa por la izquierda (LEFT o LEFT OUTER) : Se toma cada fila del primer operando que no se haya podido emparejar en la unión interna, y se concatena con otra formada por tantos Nulos como columnas tenga el segundo operando. Las filas así construidas se añaden al resultado.
- Combinación externa por la derecha (RIGHT o RIGHT OUTER) : Se construye una fila formada por tantos nulos como columnas tenga el primer operando, y se concatena con cada fila del segundo operando que no se haya emparejado. Las filas asi construidas se añaden al resultado.

Ejemplo

```
SQL Query Area
1 SELECT p.dni padre,n.nombre
2 from niños n left join padres p
3 on p.dni=padre;
4
```

padre	nombre
56895699	niño1
45789456	niño2

Una combinación externa por la izquierda puede formularse como una combinación por la derecha cambiando el orden de colocación de las tablas. La mayoría de los sistemas gestores realizan las combinaciones por la derecha, cambiándolas internamente a combinaciones por la izquierda.

La combinación por los dos lados externa no existe en mysql pero podemos generarla como vemos en el siguiente ejemplo.

Ejemplo

```
SQL Query Area
1 (select m.nombre,n.nombre from madres m left join niños n on m.dni=n.madre)
2 union
3 (select m.nombre,n.nombre from madres m right join niños n on m.dni=n.madre)
```

nombre	nombre
madre1	niño1
madre2	NULL
madre3	NULL
NULL	niño2

Veamos un ejemplo completo sobre las combinaciones.

Ejemplo

```
SQL Query Area
1 SELECT * FROM editorial;
```

id	nombre	localidad
1	editorial1	l1
2	editorial3	l2
3	editorial2	l3
4	editorial4	l4
5	editorial5	l1
6	editorial6	l2
7	editorial7	l3

```
SQL Query Area
1 SELECT * FROM libros;
```

idlibros	nombre	paginas	editorial
1	libro1	450	1
2	libro2	900	2
3	libro3	2000	3
4	libro4	1000	4
5	libro5	100	1
6	libro6	400	5
7	libro7	1500	2
8	libro8	240	1

La siguiente consulta realiza el producto cartesiano y se queda con aquellos registros que coinciden los campos id de la tabla editorial con el campo editorial de la tabla libros.

```
SQL Query Area
1 select * from libros join editorial on id=editorial;
```

idlibros	nombre	paginas	ed...	id	nombre	localidad
1	libro1	0000000450	1	1	editorial1	l1
5	libro5	0000000100	1	1	editorial1	l1
8	libro8	0000000240	1	1	editorial1	l1
2	libro2	0000000900	2	2	editorial3	l2
7	libro7	0000001500	2	2	editorial3	l2
3	libro3	0000002000	3	3	editorial2	l3
4	libro4	0000001000	4	4	editorial4	l4
6	libro6	0000000400	5	5	editorial5	l1

En la siguiente consulta sale lo mismo por la integridad referencial.

SQL Query Area

```
1 select * from libros left join editorial on id=editorial;
```

idlibros	nombre	paginas	edi...	id	nombre	localidad
1	libro1	0000000450	1	1	editorial1	11
2	libro2	0000000900	2	2	editorial3	12
3	libro3	0000002000	3	3	editorial2	13
4	libro4	0000001000	4	4	editorial4	14
5	libro5	0000000100	1	1	editorial1	11
6	libro6	0000000400	5	5	editorial5	11
7	libro7	0000001500	2	2	editorial3	12
8	libro8	0000000240	1	1	editorial1	11

Si damos la vuelta al orden de las tablas es como si tuviéramos una consulta externa de derechas.

SQL Query Area

```
1 select * from editorial left join libros on id=editorial;
```

id	nombre	localidad	idli...	nombre	paginas	editorial
1	editorial1	11	1	libro1	0000000450	1
1	editorial1	11	5	libro5	0000000100	1
1	editorial1	11	8	libro8	0000000240	1
2	editorial3	12	2	libro2	0000000900	2
2	editorial3	12	7	libro7	0000001500	2
3	editorial2	13	3	libro3	0000002000	3
4	editorial4	14	4	libro4	0000001000	4
5	editorial5	11	6	libro6	0000000400	5
6	editorial6	12	NULL	NULL	NULL	NULL
7	editorial7	13	NULL	NULL	NULL	NULL

SQL Query Area

```
1 select * from libros right join editorial on id=editorial;
```

idlibros	nombre	paginas	editor...	id	nombre	localidad
1	libro1	0000000450	1	1	editorial1	11
5	libro5	0000000100	1	1	editorial1	11
8	libro8	0000000240	1	1	editorial1	11
2	libro2	0000000900	2	2	editorial3	12
7	libro7	0000001500	2	2	editorial3	12
3	libro3	0000002000	3	3	editorial2	13
4	libro4	0000001000	4	4	editorial4	14
6	libro6	0000000400	5	5	editorial5	11
NULL	NULL	NULL	NULL	6	editorial6	12
NULL	NULL	NULL	NULL	7	editorial7	13

Hay que tener cuidado por que en ocasiones se intenta realizar una combinación FULL (por ambos lados) utilizando una combinación (no una combinación interna) mediante el operador <>. Como podeis observar el resultado no es el esperado :

SQL Query Area

```
select * from libros join editorial on id<>editorial;
```

idlibros	nombre	paginas	editorial	id	nombre	localidad
1	libro1	0000000450	1	2	editorial3	12
1	libro1	0000000450	1	3	editorial2	13
1	libro1	0000000450	1	4	editorial4	14
1	libro1	0000000450	1	5	editorial5	11
1	libro1	0000000450	1	6	editorial6	12
1	libro1	0000000450	1	7	editorial7	13
2	libro2	0000000900	2	1	editorial1	11
2	libro2	0000000900	2	3	editorial2	13
2	libro2	0000000900	2	4	editorial4	14
2	libro2	0000000900	2	5	editorial5	11
2	libro2	0000000900	2	6	editorial6	12
2	libro2	0000000900	2	7	editorial7	13
3	libro3	0000002000	3	1	editorial1	11
3	libro3	0000002000	3	2	editorial3	12
3	libro3	0000002000	3	4	editorial4	14
3	libro3	0000002000	3	5	editorial5	11
3	libro3	0000002000	3	6	editorial6	12
3	libro3	0000002000	3	7	editorial7	13
4	libro4	0000001000	4	1	editorial1	11
4	libro4	0000001000	4	2	editorial3	12

La consulta por ambos lados seria la siguiente:

SQL Query Area

```
(select * from libros left join editorial on id=editorial)
union
(select * from libros right join editorial on id=editorial);
```

idlibros	nombre	paginas	editorial	id	nombre	localidad
1	libro1	450	1	1	editorial1	11
2	libro2	900	2	2	editorial3	12
3	libro3	2000	3	3	editorial2	13
4	libro4	1000	4	4	editorial4	14
5	libro5	100	1	1	editorial1	11
6	libro6	400	5	5	editorial5	11
7	libro7	1500	2	2	editorial3	12
8	libro8	240	1	1	editorial1	11
NULL	NULL	NULL	NULL	6	editorial6	12
NULL	NULL	NULL	NULL	7	editorial7	13

2.3 Consultas anidadas

MySQL proporciona un mecanismo para las subconsultas anidadas. Una subconsulta es una expresión select-from where que se anida dentro de otra consulta.

Una subconsulta la podemos colocar en prácticamente cualquier lado de otra consulta. Veamos algunos ejemplos.

Ejemplo

```
select (select count(*) from madres) numero_madres,(select count(*) from padres) as numero_padres;
```

numero_madres	numero_padres
3	2

```
select count(*) from (select distinct nombre from niños where edad>1) as t1;
```

count(*)
2

```
select * from padres where nombre in (select distinct nombre from niños);
```

Como podemos observar dependiendo de donde coloques la subconsulta su objetivo es diferente.

2.3.1 Utilización como nuevo campo

Podemos colocar la subconsulta detrás del select como campo calculado. No es necesario colocar un alias, solo para el titulo del campo. Es importante que la subconsulta devuelva un único valor por que de lo contrario daría un error.

Ejemplo

```
select (select * from niños);
```

```
select (select sum(edad*length(nombre)) from niños) as niños_calculo,
(select sum(greatest(edad,length(nombre))) from niños) as niños_calculo1;
```

niños_calculo	niños_calculo1
90	18

Ejemplo

```
select
(select count(*) from editorial) as numero_editoriales,
(select count(*) from libros) as numero_libros,
(select count(distinct editorial) from libros) as editoriales_publicado
```

numero_editoriales	numero_libros	editoriales_publicado
7	8	5

Recordemos que en el select podemos realizar operaciones.

```
select *,numero_editoriales-editoriales_publicado as editoriales_no_publicado
from (
select
(select count(*) from editorial) as numero_editoriales,
(select count(*) from libros) as numero_libros,
(select count(distinct editorial) from libros) as editoriales_publicado
) tabla;
```

numero_editoriales	numero_libros	editoriales_publicado	editoriales_no_publicado
7	8	5	2

No se pueden utilizar los alias de los campos en la misma sentencia select

```
select
(select count(*) from editorial) as numero_editoriales,
(select count(*) from libros) as numero_libros,
(select count(distinct editorial) from libros) as editoriales_publicado,
numero_editoriales;
```

Error

2.3.2 Utilización como una nueva tabla

Podemos colocar la subconsulta como si fuera una tabla. Para ello colocamos la subconsulta detrás de la clausula FROM. Para evitar que de error ***es necesario colocar un alias a la subconsulta***.

Ejemplo

```
select * from (select * from niños);
```

```
select * from (select * from niños) alias;
```

id	nombre	edad	nacimiento	padre
1	niño1	10	1995-01-01	56895699
2	niño2	8	1997-01-05	45789456

Podemos colocar las consultas con el nivel de anidamiento que queráis.

Ejemplo

```
select * from (select * from (select * from (select * from niños) uno ) alias) otro_alias;
```

id	nombre	edad	nacimiento	padre
1	niño1	10	1995-01-01	56895699
2	niño2	8	1997-01-05	45789456

```
SQL Query Area
1 select * from
2 (
3 (select * from niños where nombre like 'a%')
4 union
5 (select * from niños where nombre like 'n%')
6 ) alias;
```

id	nombre	edad	nacimiento	padre
1	niño1	10	1995-01-01	56895699
2	niño2	8	1997-01-05	45789456

2.3.3 Utilización como criterio de la consulta

Para utilizar una subconsulta como criterio lo colocaremos en la clausula where.

Las cláusulas que permiten enlazar la consulta principal y la subconsulta son las siguientes:

- Cualquier comparador (>, <, =, etc...). En este caso, la subconsulta debe proporcionar un resultado único con el que realizar la comparación (test de comparación con subconsulta)
- Cualquier comparador seguido de ALL, ANY o SOME. En este caso, la subconsulta puede proporcionar múltiples registros como resultados.
    - ALL: se seleccionan en la consulta principal sólo los registros que verifiquen la comparación con todas las tuplas seleccionadas en la subconsulta.
    - ANY: Para recuperar registros de la consulta principal, que satisfagan la comparación con algún registro recuperado en la subconsulta.
    - SOME es idéntico a ANY.
    - IN. En este caso la subconsulta puede proporcionar múltiples tuplas como resultados, y se seleccionan en la consulta principal los registros para los que el valor del campo aparezca también en le resultado de la subconsulta. Es equivalente a utilizar "= ANY". Se puede utilizar NOT IN para conseguir el efecto contrario.
    - [not] Exists : para comprobar si existen relaciones vacias o con algo.

Empecemos con el primer caso, la comprobación sobre pertenencia a conjuntos. La clausula ***in*** comprueba la pertenencia a un conjunto, donde el conjunto es la colección de valores resultado de una cláusula select.

La conectiva ***not in***, lógicamente, comprueba la no pertenencia a un conjunto.

Ejemplo

```
SQL Query Area
1 select nombre from niños where nombre in (select nombre from padres);
```

```
SQL Query Area
1 select nombre from niños where nombre not in (select nombre from padres);
```

nombre
niño1
niño2

Podemos utilizar la clausula in con varios campos. Para ello debeis colocar el nombre de los campos entre paréntesis.

Ejemplo

```
SQL Query Area
1 select nombre,edad from niños where (nombre,edad)
2 not in
3 (select nombre,edad from niños where nacimiento between '1997/1/1' and '2005/1/1');
```

nombre	edad
niño1	10

El segundo caso es la comparación de conjuntos. Para ello podemos utilizar las clausulas some, any o all.

MySQL permite realizar las comparaciones <some, <= some, >= some, = some y <> some para comprobar si existe algun valor que cumpla el criterio.

Se puede verificar que = some es idéntico a in, mientras que <= some no es lo mismo que not in.

En SQL, la palabra clave any es sinónimo de some. Las versiones más antiguas de SQL sólo admitían any. Sin embargo, versiones posteriores añadieron la alternativa some para evitar la ambigüedad lingüística de la palabra inglesa any.

Ejemplo

```
SQL Query Area
1 select nombre,
2 case
3 when paginas<100 then 'pocas'
4 when paginas between 100 and 1000 then 'medias'
5 when paginas>1000 then 'altas'
6 else 'no tengo ni idea'
7 end as nuevo_campo
8 from libros
9
```

nombre	nuevo_campo
libro1	medias
libro2	medias
libro3	altas
libro4	medias
libro5	medias
libro6	medias
libro7	altas
libro8	medias

```
SQL Query Area
1  select *
2  from
3  (select nombre,
4  case
5  when paginas<100 then 'pocas'
6  when paginas between 100 and 1000 then 'medias'
7  when paginas>1000 then 'altas'
8  else 'no tengo ni idea'
9  end as nuevo_campo
10 from libros) tabla
11 where nombre =any
12 (select libros.nombre from libros join editorial on id=editorial where localidad='l1');
13
```

nombre	nuevo_campo
libro1	medias
libro5	medias
libro6	medias
libro8	medias

La constructora > all corresponde a la expresión «superior a todas» y <all seria inferior a todas.

Utilizamos la clausula Exists para comprobar los conjuntos vacios o con algun valor.

Ejemplo

```
SQL Query Area
1  select nombre from niños where exists (select nombre from niños);
```

nombre
niño1
niño2

2.4 Operaciones sobre conjuntos de tuplas

Las operaciones de SQL-92 union, intersect y except operan sobre relaciones y corresponden a las operaciones del álgebra relacional ∪, ∩ y –.

Al igual que la unión, intersección y diferencia de conjuntos en el álgebra relacional, las relaciones que participan en las operaciones han de ser compatibles; esto es, deben tener el mismo conjunto de atributos y el mismo nombre.

En mysql :

- la operación union existe y funciona correctamente
- la operación intersect no funciona.
- la operación except no funciona.

Ejemplo

```
SQL Query Area
1 (select nombre from libros where paginas>1000)
2 union
3 (select nombre from editorial where nombre like 'e%')
```

nombre
libro3
libro7
editorial1
editorial3
editorial2
editorial4
editorial5
editorial6
editorial7

La operación unión como ya sabemos (del algebra) une todos los registros de la primera consulta con todos los registros de la segunda. Por defecto la cláusula unión elimina los duplicados. Para colocar los duplicados se debe utilizar la directiva ALL.

Ejemplo

```
SQL Query Area
1 (select nombre from libros where paginas>1000)
2 union
3 (select nombre from libros where paginas>1200);
4
```

nombre
libro3
libro7

```
SQL Query Area
1 (select nombre from libros where paginas>1000)
2 union all
3 (select nombre from libros where paginas>1200);
4
```

nombre
libro3
libro7
libro3
libro7

Los operadores except e intersect los podemos sacar mediante operaciones derivadas de los operadores fundamentales.

Ejemplo

Por ejemplo suponeros que de los libros que tienen mas de 200 paginas quiero quitar los que tienen mas de 1200.

```
SQL Query Area
1 select nombre from libros where paginas>200 and nombre not in
2 (select nombre from libros where paginas>1200);
3
```

nombre
libro1
libro2
libro4
libro6
libro8

```
SQL Query Area
1 select nombre from libros where paginas>200 and paginas<1200;
2
```

nombre
libro1
libro2
libro4
libro6
libro8

Ejemplo

Supongamos las siguientes tablas.

SQL Query Area

```
SELECT * FROM peliculas;
```

id	nombre	director	genero	año
1	peli1	1	2	2005
2	los carraces	2	4	1985
3	unica	4	1	1998
4	los de mas alla	1	4	2001
5	esto es asi	5	3	2000

SQL Query Area

```
SELECT * FROM generos;
```

id	nombre
1	terror
2	comedia
3	familiar
4	suspense
5	policiaca
6	espacio
7	deporte
8	accion

SQL Query Area

```
SELECT * FROM director;
```

id	nombre
1	terror
2	comedia
3	familiar
4	suspense
5	policiaca
6	espacio
7	deporte
8	accion

Vamos a restar de las películas las que no son de suspense.

SQL Query Area

```
SELECT * FROM peliculas p join generos g on g.id=genero;
```

id	nombre	director	genero	año	id	nombre
1	peli1	1	2	2005	2	comedia
2	los carraces	2	4	1985	4	suspense
3	unica	4	1	1998	1	terror
4	los de mas alla	1	4	2001	4	suspense
5	esto es asi	5	3	2000	3	familiar

SQL Query Area

```
SELECT * FROM peliculas p join generos g on g.id=genero where g.nombre='suspense';
```

id	nombre	director	genero	año	id	nombre
2	los carraces	2	4	1985	4	suspense
4	los de mas alla	1	4	2001	4	suspense

SQL Query Area

```
SELECT * FROM peliculas p join generos g on g.id=genero where g.nombre<>'suspense';
```

id	nombre	director	genero	año	id	nombre
1	peli1	1	2	2005	2	comedia
3	unica	4	1	1998	1	terror
5	esto es asi	5	3	2000	3	familiar

3 Sentencia insert

Hasta ahora hemos visto todas las formas que podemos utilizar para seleccionar diferentes registros de nuestras tablas. Ahora nos vamos a empezar a ocupar de una serie de clausulas que nos permiten realizar operaciones de accion, es decir operaciones que modifican el contenido de las tablas.

Comenzamos con una clausula que nos permite introducir datos en una tabla, INSERT. Las sintaxis de esta clausula:

INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
[INTO] table_name [(col_name,...)]
VALUES ((expression | DEFAULT),...),(...),...

INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
[INTO] table_name [(col_name,...)]
SELECT ...

INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
[INTO] table_name
SET col_name=(expression | DEFAULT),...

Para insertar datos en una relación:

- o bien se especifica la tupla que se desea insertar. La forma INSERT...VALUES de la sentencia inserta filas basada en valores explícitamente especificados. Podemos insertar varias tuplas a la vez. La forma INSERT...VALUES con múltiples listas de valores se soporta en MySQL 3.22.5 o posterior.
- o se formula una consulta cuyo resultado sea el conjunto de tuplas que se desean insertar. La forma INSERT...SELECT inserta las filas seleccionadas desde otra tabla o tablas.
- O podemos introduciendo valores indicando los campos exactamente que queremos utilizar. Para realizar esto tenemos la opcion SELECT … SET. La sintaxis col_name=expression se soporta en MySQL 3.22.10 o posterior.

Obviamente, los valores de los atributos de la tuplas que se inserten deben pertenecer al dominio de los atributos. De igual modo, las tuplas insertadas deberán tener el numero de campos correcto.

La instrucción insert más sencilla corresponde a la de inserción de una tupla.

Ejemplo

SQL Query Area

```
1 insert into madres values ('12121212','madre4','direccion4','944-44-55-89','66-666-555');
```

La directiva into es opcional.

```
SQL Query Area
1 insert madres values ('11121212','madre5','direccion5','944-45-55-89','665-666-555');
```

Podemos introducir mas de un registro a la vez. Simplemente debeis colocar todos los registros entre paréntesis, separados por comas.

```
SQL Query Area
1 insert madres values
2 ('41121212','madre6','direccion6','914-45-55-89','667-676-555'),
3 ('51121212','madre7','direccion7','924-45-55-89','767-668-525'),
4 ('61121212','madre8','direccion8','934-45-55-89','677-666-511');
```

En este ejemplo los valores se especifican en el mismo orden en que los atributos se listan en el esquema de relación. Para beneficio de los usuarios, que pueden no recordar el orden de los atributos, SQL permite que los atributos se especifiquen en la cláusula insert.

Ejemplo

```
SQL Query Area
1 insert padres (dni,nombre,direccion)
2 values ('44155212','padre8','direccion5');
```

Los campos que no se utilizan se colocan con el valor por defecto. Si no tienen valor se colocarían a NULL. Hay que tener cuidado por que si el campo es requerido este hecho daría error.

También puedes utilizar la palabra clave DEFAULT para dar el valor por defecto a las columnas. (Nuevo en MySQL 4.0.3.) Esto hace mucho más sencillo escribir sentencias INSERT que asignen valores a todas las columnas excepto unas pocas, porque ello te permite evitar de escribir una lista VALUES incompleta (una lista que no incluye un valor para cada columna en la tabla). De cualquier otro modo, deberías escribir la lista de nombres de columna correspondiente para cada valor de la lista VALUES().

Ejemplo

Suponeros que yo no quiero introducir datos en todos los campos, como ocurre en la siguiente instrucción.

```
SQL Query Area
1 insert cpadres values
2 ('11121212','padre5','direccion5','944-45-55-89','665-666-555');
```

Lo que podemos realizar es utilizar la opción :

- colocar el nombre de los campos que quiero rellenar.
- Colocarlos todos y utilizar la palabra default en aquellos que no quiero rellenar.

```
SQL Query Area
1 insert cpadres (dni,nombre,direccion) values
2 ('11121212','padre5','direccion5');
```

```
SQL Query Area
1 insert cpadres values
2 ('11121212','padre5','direccion5',default,default);
```

Generalmente se desea insertar las tuplas que resultan de una consulta. En lugar de especificar una tupla, como se hizo en los primeros ejemplos de este apartado, se utiliza una instrucción select para especificar un conjunto de tuplas. La instrucción select se evalúa primero, produciendo un conjunto de tuplas que a continuación se insertan en la tabla que le indiques.

Ejemplo

```
SQL Query Area
1 insert into cpadres select * from padres;
```

Puedes utilizar los paréntesis para una mayor compresión a la hora de realizar expresiones SQL.

```
SQL Query Area
1 insert into cpadres (select * from padres);
```

Ejemplo

```
SQL Query Area
1 insert cpadres(dni,nombre) select dni,nombre from madres;
```

Una expresión puede referirse a cualquier columna que fue asignada antes en una lista de valores, pero no a una columna que todavía no hemos utilizado.

Ejemplo

Esta consulta es correcta.

```
SQL Query Area
1 insert coches (nombre,precio)
2 values
3 ('primero',length(nombre)*1000);
```

Esta consulta si se ejecutara pero de forma incorrecta.

```
SQL Query Area
1 insert coches (precio,nombre)
2 values
3 (length(nombre)*1000,'segundo');
```

El campo precio estaria a NULL.

Si especificas la palabra clave LOW_PRIORITY, la ejecución del INSERT se retrasa hasta que no haya otros clientes leyendo desde la tabla. En este caso, el cliente ha de esperar hasta que la inserción se completa, lo que puede tomar un largo rato si la tabla tiene un fuerte uso.

Esto contrasta con INSERT DELAYED, que permite al cliente continuar al mismo tiempo (esta es la opcion que utiliza por defecto).

Si especificas la palabra clave IGNORE y un INSERT con varios valores de filas, cualquier fila que duplique una clave PRIMARY o UNIQUE en la tabla se ignora y no se inserta. Si no especificas IGNORE, la inserción aborta si hay cualquier fila que duplica un valor clave existente.

La ultima forma de realizar un insert nos permite introducir valores de una forma distinta a la clasica.

Ejemplo

```
SQL Query Area
1 insert coches
2 set precio=5000,nombre='unomas';
```

4 Sentencia Update

En determinadas situaciones puede ser deseable cambiar un valor dentro de una tupla, sin cambiar todos los valores de la misma. Para este tipo de situaciones se utiliza la instrucción update. Se pueden elegir las tuplas que van a ser actualizadas mediante una consulta.

Para realizar esta tarea mysql incorpora la orden UPDATE. La sintaxis de la clausula es:

UPDATE [LOW_PRIORITY] [IGNORE] table_name
SET col_name1=expr1 [,col_name2=expr2, ...]
[WHERE where_definition]
[LIMIT #]

UPDATE actualiza las columnas en filas existentes de una tabla con nuevos valores. La cláusula SET indica qué columnas modificar y qué valores deberían darse. La cláusula WHERE, si se da, especifica qué filas deberían ser actualizadas. En cualquier otro caso, todas las filas se actualizan.

Ejemplo

Suponer que tenemos la siguiente tabla.

```
SQL Query Area
1 SELECT * FROM cpadres c
```

dni	nombre	direccion	telefono	movil
44155212	padre8	direccion5	NULL	NULL
45789456	padre1	direccion12	999-89-89-89	698-78-45-54
56895699	padre2	direccion1	NULL	669-89-89-77

Quiero rellenar los campos vacíos del primer registro :

```
SQL Query Area
update cpadres
set telefono='111-111-111',movil='549-459-888'
where nombre='padre8';
```

dni	nombre	direccion	telefono	movil
44155212	padre8	direccion5	111-111-111	549-459-888
45789456	padre1	direccion12	999-89-89-89	698-78-45-54
56895699	padre2	direccion1	NULL	669-89-89-77

Si deseamos rellenar todos los teléfonos con el mismo valor:

```
SQL Query Area
update cpadres
set telefono='111-111-111';
```

dni	nombre	direccion	telefono	movil
44155212	padre8	direccion5	111-111-111	549-459-888
45789456	padre1	direccion12	111-111-111	698-78-45-54
56895699	padre2	direccion1	111-111-111	669-89-89-77

Si especificas la palabra clave LOW_PRIORITY, la ejecución del UPDATE se retrasa hasta que no existan otros clientes leyendo la tabla.

Si especificas la palabra clave IGNORE, la sentencia de actualización no abortará aunque se den errores de claves duplicadas durante la actualización. Las filas que causasen conflictos no se actualizarían.

Si accedes a una columna de la tabla table_name en una expresión, UPDATE utiliza en valor actual de la columna.

Ejemplo

```
SQL Query Area
update niños
set edad=edad+1;
```

Las asignaciones de UPDATE se evalúan de izquierda a derecha.

Ejemplo

Tenemos la siguiente tabla:

```
SELECT * FROM `niños`;
```

id	nombre	edad	nacimiento
1	niño1	27	1995-01-01
2	niño2	25	1997-01-05

Observar como se evalua la operación:

```
update niños
set edad=edad+1,edad=edad*2;
```

id	nombre	edad	nac
1	niño1	56	19
2	niño2	52	19

En MySQL versión 3.23 o superior, puedes utilizar LIMIT # para asegurarte que sólo un número dado de registros son actualizados. Con la cláusula ORDER BY puedes asegurarte como estan ordenados.

Ejemplo

```
SELECT * FROM coches1;
```

id	nombre	nuevo	fecha_compra	precio
1	uno	1	2005-01-01	18000
2	uno	1	2005-01-01	18000
3	uno	1	2005-01-01	18000
4	uno	1	2005-01-01	18000
5	uno	1	2005-01-01	18000
6	uno	1	2005-01-01	18000
7	uno	1	2005-01-01	18000
8	uno	1	2005-01-01	18000
9	uno	1	2005-01-01	18000
10	uno	1	2005-01-01	18000
11	uno	1	2005-01-01	18000
12	uno	1	2005-01-01	18000
13	uno	1	2005-01-01	18000

```
update coches1 set nuevo=0;
```

id	nombre	nuevo
1	uno	0
2	uno	0
3	uno	0
4	uno	0
5	uno	0
6	uno	0
7	uno	0
8	uno	0
9	uno	0
10	uno	0
11	uno	0
12	uno	0
13	uno	0

Si solo hubiéramos querido cambiar un numero determinado de registros.

```
update coches1 set nuevo=0 order by precio limit 4;
```

id	nombre	nuevo	fecha_
1	uno	0	2005-0
2	uno	0	2005-0
3	uno	0	2005-0
4	uno	0	2005-0
5	uno	1	2005-0
6	uno	1	2005-0
7	uno	1	2005-0
8	uno	1	2005-0
9	uno	1	2005-0
10	uno	1	2005-0
11	uno	1	2005-0
12	uno	1	2005-0
13	uno	1	2005-0

5 Sentencia DELETE

Las consultas que vamos a analizar a continuación nos permiten borrar registros de nuestras bases de datos :

- solamente algunos registros que cumplen una condición determinada
- todos los registros de una tabla

Se pueden borrar sólo tuplas completas, es decir, no se pueden borrar valores de atributos concretos. Un borrado se expresa en SQL del modo siguiente:

DELETE [LOW_PRIORITY |QUICK] FROM table_name
[WHERE where_definition]
[ORDER BY...]
[LIMIT rows]

o

DELETE [LOW_PRIORITY |QUICK] table_name[.*] [, table_name[.*],...]
FROM table-reference
[WHERE where_definition]

o

DELETE [LOW_PRIORITY |QUICK]
FROM table_name[.*] [, table_name[.*],...]
[USING table_reference]
[WHERE where_definition]

El .* después del nombre de la tabla se utiliza por compatibilidad con Access.

Por defecto mediante este comando se crea una consulta de eliminación que elimina los registros de una o más de las tablas listadas en la cláusula FROM que satisfagan la cláusula WHERE. Esta consulta elimina los registros completos, no es posible eliminar el contenido de algún campo en concreto. Si desea eliminar valores en un campo especificado, crear una consulta de actualización que cambie los valores a Null.

Una vez que se han eliminado los registros utilizando una consulta de borrado, no puede deshacer la operación. Si desea saber qué registros se eliminarán, primero examine los resultados de una consulta de selección que utilice el mismo criterio y después ejecute la consulta de borrado. Mantenga copias de seguridad de sus datos en todo momento.

Ejemplo

Realizo la siguiente consulta de selección:

```
SELECT * FROM pruebas.`as` a
```

id	edad	fecha
1	23.00	NULL
2	45.40	NULL
3	0.12	NULL
4	21.24	2005-02-03
5	23.00	2005-01-01

Quiero borrar los tres primeros registros. Primero realizo una consulta de selección para evitar posibles errores.

```
SELECT * FROM pruebas.`as` where id<=3
```

id	edad	fecha
1	23.00	NULL
2	45.40	NULL
3	0.12	NULL

Ahora cambio el SELECT por un DELETE.

```
DELETE FROM pruebas.`as` where id<=3;
```

Si queremos borrar todos los registros de la tabla:

```
DELETE FROM pruebas.`as`;
```

El siguiente comando es idéntico.

```
DELETE `as`.* FROM `as`;
```

```
DELETE `as` FROM `as`;
```

DELETE borra las filas de la tabla table_name que satisfacen la condición dada por where_definition y retorna el número de registros borrados.

La sentencia DELETE soporta los siguientes modificadores:

- Si se especifica la palabra LOW_PRIORITY, la ejecución de DELETE se retrasa hasta que no existan clientes leyendo de la tabla.
- Para tablas MyISAM, si se especifica la palabra QUICK en algunos tipos de borrado se acelera la operación.
- La opción IGNORE hace que MySQL ignore todos los errores durante el proceso de borrado.

El primer borrado multi-tabla se soporta desde MySQL 4.0.0. El segundo formato de borrado multi-tabla se soporta desde MySQL 4.0.2.

La idea es que sólo los registros coincidentes de las tablas listadas antes del FROM o de USING se borran. El efecto es que puedes borrar filas desde varias filas de una vez y además tienes tablas adicionales que se usan para búsqueda.

Ejemplo

Borramos las filas coincidentes sólo de las tablas t1 y t2.

DELETE t1,t2 FROM t1,t2,t3 WHERE t1.id=t2.id AND t2.id=t3.id

o

DELETE t1,t2 USING t1,t2,t3 WHERE t1.id=t2.id AND t2.id=t3.id

ORDER BY utilizando múltiples tablas en la sentencia DELETE se soporta en MySQL 4.0.

Si se utiliza una cláusula ORDER BY, las filas se borrarán en tal orden. Esto sólo es realmente útil en combinación con LIMIT.

Ejemplo

Tenemos la siguiente tabla.

SQL Query Area

```
1 SELECT * FROM coches1;
```

id	nombre	nuevo	comprado	precio
1	uno	1	2005-01-01	5000
2	dos	0	2004-08-07	8000
3	tres	1	2000-12-04	12000
4	cuatro	1	1987-12-10	24000
5	primero	NULL	NULL	7000
6	segundo	NULL	NULL	NULL
7	unomas	NULL	NULL	5000

Borramos los tres primeros registros.

SQL Query Area

```
1 DELETE FROM coches1 order by id asc limit 3;
```

6 Comando TRUNCATE

En el capitulo siguiente veremos un comando que nos permite borrar una tabla completamente (registros y estructura)

DROP TABLE nombre_tabla.

Para eliminar únicamente los registros de una tabla podemos utilizar el comando TRUNCATE. La sintaxis de este comando es :

TRUNCATE TABLE nombre_tabla;

Esta es una extensión de SQL de Oracle.

TRUNCATE TABLE difiere de DELETE FROM... en los siguientes sentidos:

- Las operaciones de truncado vacían y crean de nuevo la tabla, lo que es mucho más rápido que borrar las filas una por una.
- No está a salvo de transacciones; obtendrás un error si tienes una transacción en activo o un bloqueo activo de la tabla.
- No retorna el número de registros borrados.
- Los campos autonuméricos comenzaran a contar desde 1.

Ejemplo

La siguiente instrucción borra todos los registros de la tabla coches1.

```
SQL Query Area
1 truncate coches1;
```

7 Comando REPLACE

REPLACE trabaja exactamente igual que INSERT, excepto que si existe algún registro viejo en la tabla que tenga el mismo valor que uno nuevo para un índice PRIMARY KEY o UNIQUE, el viejo se borra antes de que el nuevo sea insertado.

Hay que tener en cuenta que salvo que la tabla tenga una PRIMARY KEY o un índice UNIQUE, usar una sentencia REPLACE no tiene sentido. En ese caso es equivalente usar una sentencia INSERT, ya que no hay ningún índice que se pueda usar para determinar si una nueva fila duplica a otra.

Por lo tanto la sintaxis de este comando es:

REPLACE [LOW_PRIORITY |DELAYED]
[INTO] table_name [(col_name,...)]
VALUES (expression,...), (...),...
o
REPLACE [LOW_PRIORITY |DELAYED]
[INTO] table_name [(col_name,...)]
SELECT ...
o
REPLACE [LOW_PRIORITY |DELAYED]
[INTO] table_name
SET col_name=expression, col_name=expression,...

La sentencias REPLACE devuelve un contador para indicar el número de filas afectadas. Ese número es la suma de filas borradas e insertadas. Si el contador es 1 para un REPLACE de una única fila, la fila fue insertada y no se borró ninguna fila. Si el contador es mayor de 1, una o más de las viejas filas fue borrada antes de que la nueva fila fuese insertada. Es posible que una única fila

reemplace a más de una fila vieja si la tabla contiene varios índices únicos y la nueva fila duplica valores de diferentes filas viejas en diferentes índices únicos.

Ejemplo

Supongamos que tenemos el siguiente registro dentro de la tabla.

```
SQL Query Area
1 SELECT * FROM coches1 c
```

id	nombre	nuevo	comprado	precio
1	ramon	1	1999-12-12	4000

Vamos a insertar dos valores.

```
SQL Query Area
1 REPLACE coches1 values(default,'jose',0,'2000/7/8',3256),(default,'luisa',1,'2001/12/7',2000);
```

id	nombre	nuevo	comprado	precio
1	ramon	1	1999-12-12	4000
2	jose	0	2000-07-08	3256
3	luisa	1	2001-12-07	2000

Este comando podemos sustituirlo por INSERT y el resultado es el mismo.

Suponeros que tenemos la siguiente tabla:

```
SQL Query Area
1 SELECT * FROM uno;
```

id	nombre	coche	años
1	ramon	1	0

Al realizar este comando nos devuelve el error que se puede ver en la figura. Esto es debido a que el campo nombre es indexado sin duplicados.

```
SQL Query Area
1 insert uno values(default,'ramon',1,12);
```

!	Description
!	Duplicate entry 'ramon' for key 2

Para evitar este error podemos ejecutar el siguiente comando:

```
SQL Query Area
1 REPLACE uno values(default,'ramon',1,12);
```

El resultado podemos observarlo en la figura.

```
SQL Query Area
1 SELECT * FROM uno;
```

id	nombre	coche	años
2	ramon	1	12

Se ha borrado el registro existente y se ha introducido el nuevo. Esto nos lo ha indicado mysql de la siguiente forma:

```
2 rows affected by the last command, no resultset returned.
```

8 *Comando LOAD DATA INFILE*

La sentencia LOAD DATA INFILE lee registros desde un archivo de texto para ponerlos en una tabla a una velocidad muy alta.

La sintaxis y la forma de trabajar es muy similar al comando que hemos visto anteriormente denominado SELECT …. INTO OUTILE …

LOAD DATA [LOW_PRIORITY|CONCURRENT] [LOCAL] INFILE 'filename.txt'
[REPLACE|IGNORE]
INTO TABLE tbl_name
[opciones de importación]

Ejemplo

Borramos el contenido de la tabla coches

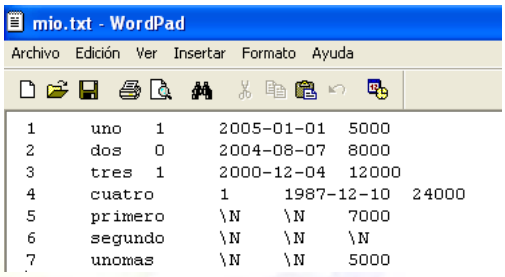
```
SQL Query Area
1 delete from coches;
```

Ahora insertamos una serie de registros a la tabla desde un fichero.

```
SQL Query Area
1 load data infile 'c:\\mio.txt' into table coches;
```

id	nombre	nuevo	comprado	precio
1	uno	1	2005-01-01	5000
2	dos	0	2004-08-07	8000
3	tres	1	2000-12-04	12000
4	cuatro	1	1987-12-10	24000
5	primero	NULL	NULL	7000
6	segundo	NULL	NULL	NULL
7	unomas	NULL	NULL	5000

El fichero es el siguiente:



Si se especifica LOCAL, el archivo es leído desde el ordenador cliente. Si LOCAL no se especifica, el archivo debe estar ubicado en el servidor. Utilizando LOCAL será algo más lento que dejar que el servidor acceda al archivo directamente, debido a que los contenidos del archivo deben viajar desde el cliente hasta el servidor.

Por razones de seguridad, al leer archivos de texto localizados en el servidor, los archivos deben residir o bien en el directorio de la base de datos o ser leíbles por todos.

Si especificas LOW_PRIORITY, la ejecución de LOAD DATA se retrasa hasta que no haya clientes leyendo de la tabla. Si especificas CONCURRENT con una tabla MyISAM, otras llamadas pueden recuperar datos desde la tabla mientras se ejecuta LOAD DATA.

Al ubicar archivos en el servidor, éste sigue las siguientes reglas:

- Si se da una ruta absoluta, el servidor lo utiliza tal cual.
- Si se da una ruta relativa con uno o más componentes, el servidor busca el archivo partiendo del directorio del servidor de la base de datos.
- Si se da un archivo sin componentes, el servidor busca el archivo en el directorio de la base de datos actual.

Las palabras clave REPLACE e IGNORE controlan el manejo de la introducción de registros que duplican los registros existentes en valores clave únicos. Si especificas IGNORE, las filas entradas que dupliquen claves existentes serán saltadas. Si no especificas ninguna de las opciones, se da un error cuando se encuentra una duplicación de valores clave, y el resto del archivo de texto se ignora.

Las opciones de importación son las mismas que vimos para el comando anterior.