Clase 2

CLAVE PRIMARIA

Una clave primaria es un campo (o varios) que identifica 1 solo registro (fila) en una tabla.

Para un valor del campo clave existe solamente 1 registro. Los valores no se repiten ni pueden ser nulos.

Veamos un ejemplo, si tenemos una tabla con datos de personas, el número de documento puede establecerse como clave primaria, es un valor que no se repite; puede haber personas con igual apellido y nombre, incluso el mismo domicilio (padre e hijo por ejemplo), pero su documento será siempre distinto.

Si tenemos la tabla "usuarios", el nombre de cada usuario puede establecerse como clave primaria, es un valor que no se repite; puede haber usuarios con igual clave, pero su nombre de usuario será siempre distinto.

Establecemos que un campo sea clave primaria al momento de creación de la tabla:

```
create table usuarios (
 nombre varchar(20),
 clave varchar(10),
 primary key(nombre)
);
```

Para definir un campo como clave primaria agregamos "primary key" luego de la definición de todos los campos y entre paréntesis colocamos el nombre del campo que queremos como clave.

Si visualizamos la estructura de la tabla con "describe" vemos que el campo "nombre" es clave primaria y no acepta valores nulos(más adelante explicaremos esto detalladamente).

Ingresamos algunos registros:

```
insert into usuarios (nombre, clave)
 values ('Leonardo','payaso');
insert into usuarios (nombre, clave)
 values ('MarioPerez','Marito');
insert into usuarios (nombre, clave)
 values ('Marcelo','River');
insert into usuarios (nombre, clave)
 values ('Gustavo','River');
```

Si intentamos ingresar un valor para el campo clave que ya existe, aparece un mensaje de error indicando que el registro no se cargó pues el dato clave existe. Esto sucede porque los campos definidos como clave primaria no pueden repetirse.

Ingresamos un registro con un nombre de usuario repetido, por ejemplo:

```
insert into usuarios (nombre, clave)
 values ('Gustavo','Boca');
```

Una tabla sólo puede tener una clave primaria. Cualquier campo (de cualquier tipo) puede ser clave primaria, debe cumplir como requisito, que sus valores no se repitan.

Al establecer una clave primaria estamos indexando la tabla, es decir, creando un índice para dicha tabla; a este tema lo veremos más adelante.

CLAVES PRIMARIAS COMPUESTAS

Las claves primarias pueden ser simples, formadas por un solo campo o compuestas, más de un campo.

Recordemos que una clave primaria identifica 1 solo registro en una tabla. Para un valor del campo clave existe solamente 1 registro. Los valores no se repiten ni pueden ser nulos.

Retomemos el ejemplo de la playa de estacionamiento que almacena cada día los datos de los vehículos que ingresan en la tabla llamada "vehiculos" con los siguientes campos:

- patente char(6) not null,
- tipo char (4),
- horallegada time not null,
- horasalida time,

Necesitamos definir una clave primaria para una tabla con los datos descriptos arriba. No podemos usar la patente porque un mismo auto puede ingresar más de una vez en el día a la playa; tampoco podemos usar la hora de entrada porque varios autos pueden ingresar a una misma hora. Tampoco sirven los otros campos.

Como ningún campo, por si solo cumple con la condición para ser clave, es decir, debe identificar un solo registro, el valor no puede repetirse, debemos usar 2 campos.

Definimos una clave compuesta cuando ningún campo por si solo cumple con la condición para ser clave.

En este ejemplo, un auto puede ingresar varias veces en un día a la playa, pero siempre será a distinta hora.

Usamos 2 campos como clave, la patente junto con la hora de llegada, así identificamos unívocamente cada registro.

Para establecer más de un campo como clave primaria usamos la siguiente sintaxis:

```
create table vehiculos(
 patente char(6) not null,
 tipo char(4),
 horallegada time not null
 horasalida time,
 primary key(patente,horallegada)
);
```

Nombramos los campos que formarán parte de la clave separados por comas.

Si vemos la estructura de la tabla con "describe" vemos que en la columna "key", en ambos campos aparece "PRI", porque ambos son clave primaria.

Un campo que es parte de una clave primaria puede ser autoincrementable sólo si es el primer campo que compone la clave, si es secundario no se permite.

Es posible eliminar un campo que es parte de una clave primaria, la clave queda con los campos restantes. Esto, siempre que no queden registros con clave repetida. Por ejemplo, podemos eliminar el campo "horallegada":

```
alter table vehiculos drop horallegada;
```

siempre que no haya registros con "patente" duplicada, en ese caso aparece un mensaje de error y la eliminación del campo no se realiza.

En caso de ejecutarse la sentencia anterior, la clave queda formada sólo por el campo "patente".

AUTO-INCREMENTO

Un campo de tipo entero puede tener otro atributo extra 'auto_increment'. Los valores de un campo 'auto_increment', se inician en 1 y se incrementan en 1 automáticamente.

Se utiliza generalmente en campos correspondientes a códigos de identificación para generar valores únicos para cada nuevo registro que se inserta.

Sólo puede haber un campo "auto_increment" y debe ser clave primaria (o estar indexado).

Para establecer que un campo autoincremente sus valores automáticamente, éste debe ser entero (integer) y debe ser clave primaria:

```
create table libros(
 codigo int auto_increment,
 titulo varchar(20),
 autor varchar(30),
 editorial varchar(15),
 primary key (codigo)
);
```

Para definir un campo autoincrementable colocamos "auto_increment" luego de la definición del campo al crear la tabla.

Hasta ahora, al ingresar registros, colocamos el nombre de todos los campos antes de los valores; es posible ingresar valores para algunos de los campos de la tabla, pero recuerde que al ingresar los valores debemos tener en cuenta los campos que detallamos y el orden en que lo hacemos.

Cuando un campo tiene el atributo "auto_increment" no es necesario ingresar valor para él, porque se inserta automáticamente tomando el último valor como referencia, o 1 si es el primero.

Para ingresar registros omitimos el campo definido como "auto_increment", por ejemplo:

```
insert into libros (titulo,autor,editorial)
 values('El aleph','Borges','Planeta');
```

Este primer registro ingresado guardará el valor 1 en el campo correspondiente al código.

Si continuamos ingresando registros, el código (dato que no ingresamos) se cargará automáticamente siguiendo la secuencia de autoincremento.

Un campo "auto_increment" funciona correctamente sólo cuando contiene únicamente valores positivos. Más adelante explicaremos cómo definir un campo con sólo valores positivos.

Está permitido ingresar el valor correspondiente al campo "auto_increment", por ejemplo:

```
insert into libros (codigo,titulo,autor,editorial)
 values(6,'Martin Fierro','Jose Hernandez','Paidos');
```

Pero debemos tener cuidado con la inserción de un dato en campos "auto_increment". Debemos tener en cuenta que:

- si el valor está repetido aparecerá un mensaje de error y el registro no se ingresará.
- si el valor dado saltea la secuencia, lo toma igualmente y en las siguientes inserciones, continuará la secuencia tomando el valor más alto.
- si el valor ingresado es 0, no lo toma y guarda el registro continuando la secuencia.
- si el valor ingresado es negativo (y el campo no está definido para aceptar sólo valores positivos), lo ingresa.

Para que este atributo funcione correctamente, el campo debe contener solamente valores positivos; más adelante trataremos este tema.

NULL

Analizaremos la estructura de una tabla que vemos al utilizar el comando "describe". Tomamos como ejemplo la tabla "libros":

Field	Type		Null	Key	Default	Extra
codigo	int(11)	7 b..	NO	PRI	auto_increment	
titulo	varchar(20)	11 b..	YES		(NULL)	
autor	varchar(30)	11 b..	YES		(NULL)	
editorial	varchar(15)	11 b..	YES		(NULL)	
precio	float	5 b..	YES		(NULL)	

La primera columna indica el tipo de dato de cada campo.

La segunda columna "**Null**" especifica si el campo permite valores nulos; vemos que en el campo "codigo", aparece "NO" y en las demás "YES", esto significa que el primer campo no acepta valores nulos (porque es clave primaria) y los otros si los permiten.

La tercera columna "Key", muestra los campos que son clave primaria; en el campo "codigo" aparece "PRI" (es clave primaria) y los otros están vacíos, porque no son clave primaria.

La cuarta columna "Default", muestra los valores por defecto, esto es, los valores que MySQL ingresa cuando omitimos un dato o colocamos un valor inválido; para todos los campos, excepto para el que es clave primaria, el valor por defecto es "null".

La quinta columna "Extra", muestra algunos atributos extra de los campos; el campo "codigo" es "auto_increment".

Vamos a explicar los valores nulos.

"null' significa "dato desconocido" o "valor inexistente". No es lo mismo que un valor 0, una cadena vacía o una cadena literal "null".

A veces, puede desconocerse o no existir el dato correspondiente a algún campo de un registro. En estos casos decimos que el campo puede contener valores nulos. Por ejemplo, en nuestra tabla de libros, podemos tener valores nulos en el campo "precio" porque es posible que para algunos libros no le hayamos establecido el precio para la venta.

En contraposición, tenemos campos que no pueden estar vacíos jamás, por ejemplo, los campos que identifican cada registro, como los códigos de identificación, que son clave primaria.

Por defecto, es decir, si no lo aclaramos en la creación de la tabla, los campos permiten valores nulos.

Imaginemos que ingresamos los datos de un libro, para el cual aún no hemos definido el precio:

```
insert into libros (titulo,autor,editorial,precio)
  values ('El aleph','Borges','Planeta',null);
```

Note que el valor "null" no es una cadena de caracteres, no se coloca entre comillas.

Si un campo acepta valores nulos, podemos ingresar "null" cuando no conocemos el valor.

Los campos establecidos como clave primaria no aceptan valores nulos. Nuestro campo clave primaria, está definido "auto_increment"; si intentamos ingresar el valor "null" para este campo, no lo tomará y seguirá la secuencia de incremento.

El campo "titulo", no debería aceptar valores nulos, para establecer este atributo debemos crear la tabla con la siguiente sentencia:

```
create table libros(
 codigo int auto_increment,
 titulo varchar(20) not null
 autor varchar(30),
 editorial varchar(15),
 precio float,
 primary key (codigo)
);
```

Entonces, para que un campo no permita valores nulos debemos especificarlo luego de definir el campo, agregando "not null". Por defecto, los campos permiten valores nulos, pero podemos especificarlo igualmente agregando "null".

Explicamos que "null" no es lo mismo que una cadena vacía o un valor 0 (cero).

Para recuperar los registros que contengan el valor "null" en el campo "precio" no podemos utilizar los operadores relacionales vistos anteriormente: = (igual) y <> (distinto); debemos utilizar los operadores "is null" (es igual a null) y "is not null" (no es null):

```
select * from libros
 where precio is null;
```

La sentencia anterior tendrá una salida diferente a la siguiente:

```
select * from libros
 where precio=0;
```

Con la primera sentencia veremos los libros cuyo precio es igual a "null" (desconocido); con la segunda, los libros cuyo precio es 0.

Igualmente para campos de tipo cadena, las siguientes sentencias "select" no retornan los mismos registros:

```
select * from libros where editorial is null;
select * from libros where editorial='';
```

Con la primera sentencia veremos los libros cuya editorial es igual a "null", con la segunda, los libros cuya editorial guarda una cadena vacía

ORDER BY

Podemos ordenar el resultado de un "select" para que los registros se muestren ordenados por algún campo, para ello usamos la cláusula "order by".

Por ejemplo, recuperamos los registros de la tabla "libros" ordenados por el título:

```
select codigo,titulo,autor,editorial,precio from libros order by titulo;
```

Aparecen los registros ordenados alfabéticamente por el campo especificado.

También podemos colocar el número de orden del campo por el que queremos que se ordene en lugar de su nombre. Por ejemplo, queremos el resultado del "select" ordenado por "precio":

```
select codigo,titulo,autor,editorial,precio from libros order by 5;
```

Por defecto, si no aclaramos en la sentencia, los ordena de manera ascendente (de menor a mayor). Podemos ordenarlos de mayor a menor, para ello agregamos la palabra clave "desc":

```
select codigo,titulo,autor,editorial,precio from libros order by editorial desc;
```

También podemos ordenar por varios campos, por ejemplo, por "titulo" y "editorial":

```
select codigo,titulo,autor,editorial,precio from libros order by titulo, editorial;
```

Incluso, podemos ordenar en distintos sentidos, por ejemplo, por "titulo" en sentido ascendente y "editorial" en sentido descendente:

```
select codigo,titulo,autor,editorial,precio
  from libros order by titulo asc, editorial desc;
```

Debe aclararse al lado de cada campo, pues estas palabras claves afectan al campo inmediatamente anterior.

Los operadores "**regexp**" y "not regexp" busca patrones de modo similar a "like" y "not like".

Para buscar libros que contengan la cadena "Ma" usamos:

```
select titulo from libros
 where titulo regexp 'Ma';
```

Para buscar los autores que tienen al menos una "h" o una "k" o una "w" tipeamos:

```
select autor from libros
 where autor regexp '[hkw]';
```

Para buscar los autores que no tienen ni "h" o una "k" o una "w" tipeamos:

```
select autor from libros
 where autor not regexp '[hkw]';
```

Para buscar los autores que tienen por lo menos una de las letras de la "a" hasta la "d", es decir, "a,b,c,d", usamos:

```
select autor from libros
 where autor regexp '[a-d]';
```

Para ver los títulos que comienzan con "A" tipeamos:

```
select titulo from libros
 where titulo regexp '^A';
```

Para ver los títulos que terminan en "HP" usamos:

```
select titulo from libros
 where titulo regexp 'HP$';
```

Para buscar títulos que contengan una "a" luego un caracter cualquiera y luego una "e" utilizamos la siguiente sentencia:

```
select titulo from libros
 where titulo regexp 'a.e';
```

El punto (.) identifica cualquier caracter.

Podemos mostrar los títulos que contienen una "a" seguida de 2 caracteres y luego una "e":

```
select titulo from libros
 where titulo regexp 'a..e';
```

Para buscar autores que tengan 6 caracteres exactamente usamos:

```
select autor from libros
 where autor regexp '^......$';
```

Para buscar autores que tengan al menos 6 caracteres usamos:

```
select autor from libros
 where autor regexp '......';
```

Para buscar títulos que contengan 2 letras "a" usamos:

```
select titulo from libros
 where titulo regexp 'a.*a';
```

El asterisco indica que busque el caracter inmediatamente anterior, en este caso cualquiera porque hay un punto.

REGEXP

Los operadores "regexp" y "not regexp" busca patrones de modo similar a "like" y "not like".

Para buscar libros que contengan la cadena "Ma" usamos:

```
select titulo from libros
 where titulo regexp 'Ma';
```

Para buscar los autores que tienen al menos una "h" o una "k" o una "w" tipeamos:

```
select autor from libros
 where autor regexp '[hkw]';
```

Para buscar los autores que no tienen ni "h" o una "k" o una "w" tipeamos:

```
select autor from libros
 where autor not regexp '[hkw]';
```

Para buscar los autores que tienen por lo menos una de las letras de la "a" hasta la "d", es decir, "a,b,c,d", usamos:

```
select autor from libros
 where autor regexp '[a-d]';
```

Para ver los títulos que comienzan con "A" tipeamos:

```
select titulo from libros
 where titulo regexp '^A';
```

Para ver los títulos que terminan en "HP" usamos:

```
select titulo from libros
 where titulo regexp 'HP$';
```

Para buscar títulos que contengan una "a" luego un caracter cualquiera y luego una "e" utilizamos la siguiente sentencia:

```
select titulo from libros
 where titulo regexp 'a.e';
```

El punto (.) identifica cualquier caracter.

Podemos mostrar los títulos que contienen una "a" seguida de 2 caracteres y luego una "e":

```
select titulo from libros
 where titulo regexp 'a..e';
```

Para buscar autores que tengan 6 caracteres exactamente usamos:

```
select autor from libros
 where autor regexp '^......$';
```

Para buscar autores que tengan al menos 6 caracteres usamos:

```
select autor from libros
 where autor regexp '......';
```

Para buscar títulos que contengan 2 letras "a" usamos:

```
select titulo from libros
 where titulo regexp 'a.*a';
```

El asterisco indica que busque el caracter inmediatamente anterior, en este caso cualquiera porque hay un punto.

Funciones de agrupamiento (count - max - min - sum - avg)

Existen en MySQL funciones que nos permiten contar registros, calcular sumas, promedios, obtener valores máximos y mínimos. Ya hemos aprendido "count()", veamos otras.

La función "sum()" retorna la suma de los valores que contiene el campo especificado. Por ejemplo, queremos saber la cantidad de libros que tenemos disponibles para la venta:

```
select sum(cantidad) from libros;
```

También podemos combinarla con "where". Por ejemplo, queremos saber cuántos libros tenemos de la editorial "Planeta":

```
select sum(cantidad) from libros
 where editorial ='Planeta';
```

Para averiguar el valor máximo o mínimo de un campo usamos las funciones "max()" y "min()" respectivamente. Ejemplo, queremos saber cuál es el mayor precio de todos los libros:

```
select max(precio) from libros;
```

Queremos saber cuál es el valor mínimo de los libros de "Rowling":

```
select min(precio) from libros
 where autor like '%Rowling%';
```

La función avg() retorna el valor promedio de los valores del campo especificado. Por ejemplo, queremos saber el promedio del precio de los libros referentes a "PHP":

```
select avg(precio) from libros
 where titulo like '%PHP%';
```

Estas funciones se denominan "funciones de agrupamiento" porque operan sobre conjuntos de registros, no con datos individuales.

Tenga en cuenta que no debe haber espacio entre el nombre de la función y el paréntesis, porque puede confundirse con una referencia a una tabla o campo. Las siguientes sentencias son distintas:

```
select count(*) from libros;
select count (*) from libros;
```

La primera es correcta, la segunda incorrecta.

COUNT

Existen en MySQL funciones que nos permiten contar registros, calcular sumas, promedios, obtener valores máximos y mínimos. Veamos algunas de ellas.

Imaginemos que nuestra tabla "libros" contiene muchos registros. Para averiguar la cantidad sin necesidad de contarlos manualmente usamos la función "count()":

```
select count(*) from libros;
```

La función "count()" cuenta la cantidad de registros de una tabla, incluyendo los que tienen valor nulo.

Para saber la cantidad de libros de la editorial "Planeta" tipeamos:

```
select count(*) from libros
 where editorial='Planeta';
```

También podemos utilizar esta función junto con la clausula "where" para una consulta más específica. Por ejemplo, solicitamos la cantidad de libros que contienen la cadena "Borges":

```
select count(*) from libros
 where autor like '%Borges%';
```

Para contar los registros que tienen precio (sin tener en cuenta los que tienen valor nulo), usamos la función "count()" y en los paréntesis colocamos el nombre del campo que necesitamos contar:

```
select count(precio) from libros;
```

Note que "count(*)" retorna la cantidad de registros de una tabla (incluyendo los que tienen valor "null") mientras que "count(precio)" retorna la cantidad de registros en los cuales el campo "precio" no es nulo. No es lo mismo. "count(*)" cuenta registros, si en lugar de un asterisco colocamos como argumento el nombre de un campo, se contabilizan los registros cuyo valor en ese campo no es nulo.

Tenga en cuenta que no debe haber espacio entre el nombre de la función y el paréntesis, porque puede confundirse con una referencia a una tabla o campo. Las siguientes sentencias son distintas:

```
select count(*) from libros;
select count (*) from libros;
```

La primera es correcta, la segunda incorrecta.

GROUP BY

Hemos aprendido que las funciones de agrupamiento permiten contar registros, calcular sumas y promedios, obtener valores máximos y mínimos. También dijimos que dichas funciones operan sobre conjuntos de registros, no con datos individuales.

Generalmente esta funciones se combinan con la sentencia "group by", que agrupa registros para consultas detalladas.

Queremos saber la cantidad de visitantes de cada ciudad, podemos tipear la siguiente sentencia:

```
select count(*) from visitantes
 where ciudad='Cordoba';
```

y repetirla con cada valor de "ciudad":

```
select count(*) from visitantes
 where ciudad='Alta Gracia';
select count(*) from visitantes
 where ciudad='Villa Dolores';
...
```

Pero hay otra manera, utilizando la cláusula "group by":

```
select ciudad, count(*)
 from visitantes
 group by ciudad;
```

Entonces, para saber la cantidad de visitantes que tenemos en cada ciudad utilizamos la función "count()", agregamos "group by" y el campo por el que deseamos que se realice el agrupamiento, también colocamos el nombre del campo a recuperar.

La instrucción anterior solicita que muestre el nombre de la ciudad y cuente la cantidad agrupando los registros por el campo "ciudad". Como resultado aparecen los nombres de las ciudades y la cantidad de registros para cada valor del campo.

Para obtener la cantidad visitantes con teléfono no nulo, de cada ciudad utilizamos la función "count()" enviándole como argumento el campo "telefono", agregamos "group by" y el campo por el que deseamos que se realice el agrupamiento (ciudad):

```
select ciudad, count(telefono)
 from visitantes
 group by ciudad;
```

Como resultado aparecen los nombres de las ciudades y la cantidad de registros de cada una, sin contar los que tienen teléfono nulo. Recuerde la diferencia de los valores que retorna la función "count()" cuando enviamos como argumento un asterisco o el nombre de un campo: en el primer caso cuenta todos los registros incluyendo los que tienen valor nulo, en el segundo, los registros en los cuales el campo especificado es no nulo.

Para conocer el total de las compras agrupadas por sexo:

```
select sexo, sum(montocompra)
 from visitantes
 group by sexo;
```

Para saber el máximo y mínimo valor de compra agrupados por sexo:

```
select sexo, max(montocompra) from visitantes
 group by sexo;
select sexo, min(montocompra) from visitantes
 group by sexo;
```

Se pueden simplificar las 2 sentencias anteriores en una sola sentencia, ya que usan el mismo "group by":

```
select sexo, max(montocompra),
 min(montocompra)
 from visitantes
 group by sexo;
```

Para calcular el promedio del valor de compra agrupados por ciudad:

```
select ciudad, avg(montocompra) from visitantes
 group by ciudad;
```

Podemos agrupar por más de un campo, por ejemplo, vamos a hacerlo por "ciudad" y "sexo":

```
select ciudad, sexo, count(*) from visitantes
 group by ciudad,sexo;
```

También es posible limitar la consulta con "where".

Vamos a contar y agrupar por ciudad sin tener en cuenta "Cordoba":

```
select ciudad, count(*) from visitantes
 where ciudad<>'Cordoba'
 group by ciudad;
```

Podemos usar las palabras claves "asc" y "desc" para una salida ordenada:

```
select ciudad, count(*) from visitantes
 group by ciudad desc;
```

HAVING

Así como la cláusula "where" permite seleccionar (o rechazar) registros individuales; la cláusula "having" permite seleccionar (o rechazar) un grupo de registros.

Si queremos saber la cantidad de libros agrupados por editorial usamos la siguiente instrucción ya aprendida:

```
select editorial, count(*) from libros
 group by editorial;
```

Si queremos saber la cantidad de libros agrupados por editorial pero considerando sólo algunos grupos, por ejemplo, los que devuelvan un valor mayor a 2, usamos la siguiente instrucción:

```
select editorial, count(*) from libros
 group by editorial
 having count(*)>2;
```

Se utiliza "having", seguido de la condición de búsqueda, para seleccionar ciertas filas retornadas por la cláusula "group by".

Veamos otros ejemplos. Queremos el promedio de los precios de los libros agrupados por editorial:

```
select editorial, avg(precio) from libros
 group by editorial;
```

Ahora, sólo queremos aquellos cuyo promedio supere los 25 pesos:

```
select editorial, avg(precio) from libros
 group by editorial
 having avg(precio)>25;
```

En algunos casos es posible confundir las cláusulas "where" y "having". Queremos contar los registros agrupados por editorial sin tener en cuenta a la editorial "Planeta".

Analicemos las siguientes sentencias:

```
select editorial, count(*) from libros
 where editorial<>'Planeta'
 group by editorial;
select editorial, count(*) from libros
 group by editorial
 having editorial<>'Planeta';
```

Ambas devuelven el mismo resultado, pero son diferentes.

La primera, selecciona todos los registros rechazando los de editorial "Planeta" y luego los agrupa para contarlos. La segunda, selecciona todos los registros, los agrupa para contarlos y finalmente rechaza la cuenta correspondiente a la editorial "Planeta".

No debemos confundir la cláusula "where" con la cláusula "having"; la primera establece condiciones para la selección de registros de un "select"; la segunda establece condiciones para la selección de registros de una salida "group by".

Veamos otros ejemplos combinando "where" y "having".

Queremos la cantidad de libros, sin considerar los que tienen precio nulo, agrupados por editorial, sin considerar la editorial "Planeta":

```
select editorial, count(*) from libros
 where precio is not null
 group by editorial
 having editorial<>'Planeta';
```

Aquí, selecciona los registros rechazando los que no cumplan con la condición dada en "where", luego los agrupa por "editorial" y finalmente rechaza los grupos que no cumplan con la condición dada en el "having".

Generalmente se usa la cláusula "having" con funciones de agrupamiento, esto no puede hacerlo la cláusula "where". Por ejemplo queremos el promedio de los precios agrupados por editorial, de aquellas editoriales que tienen más de 2 libros:

```
select editorial, avg(precio) from libros
 group by editorial
 having count(*) > 2;
```

Podemos encontrar el mayor valor de los libros agrupados por editorial y luego seleccionar las filas que tengan un valor mayor o igual a 30:

```
select editorial, max(precio) from libros
 group by editorial
 having max(precio)>=30;
```

Esta misma sentencia puede usarse empleando un "alias", para hacer referencia a la columna de la expresión:

```
select editorial, max(precio) as 'mayor' from libros
 group by editorial
 having mayor>=30;
```

DISTINCT

Con la cláusula "distinct" se especifica que los registros con ciertos datos duplicados sean obviadas en el resultado. Por ejemplo, queremos conocer todos los autores de los cuales tenemos libros, si utilizamos esta sentencia:

```
select autor from libros;
```

Aparecen repetidos. Para obtener la lista de autores sin repetición usamos:

```
select distinct autor from libros;
```

También podemos tipear:

```
select autor from libros
 group by autor;
```

Note que en los tres casos anteriores aparece "null" como un valor para "autor"· Si sólo queremos la lista de autores conocidos, es decir, no queremos incluir "null" en la lista, podemos utilizar la sentencia siguiente:

```
select distinct autor from libros
 where autor is not null;
```

Para contar los distintos autores, sin considerar el valor "null" usamos:

```
select count(distinct autor)
 from libros;
```

Note que si contamos los autores sin "distinct", no incluirá los valores "null" pero si los repetidos:

```
select count(autor)
 from libros;
```

Esta sentencia cuenta los registros que tienen autor.

Para obtener los nombres de las editoriales usamos:

```
select editoriales from libros;
```

Para una consulta en la cual los nombres no se repitan tipeamos:

```
select distinct editorial from libros;
```

Podemos saber la cantidad de editoriales distintas usamos:

```
select count(distinct editoriales) from libros;
```

Podemos combinarla con "where". Por ejemplo, queremos conocer los distintos autores de la editorial "Planeta":

```
select distinct autor from libros
 where editorial='Planeta';
```

También puede utilizarse con "group by":

```
select editorial, count(distinct autor)
 from libros
 group by editorial;
```

Para mostrar los títulos de los libros sin repetir títulos, usamos:

```
select distinct titulo from libros
 order by titulo;
```

La cláusula "distinct" afecta a todos los campos presentados. Para mostrar los títulos y editoriales de los libros sin repetir títulos ni editoriales, usamos:

```
select distinct titulo,editorial
 from libros
 order by titulo;
```

Note que los registros no están duplicados, aparecen títulos iguales pero con editorial diferente, cada registro es diferente.

LIMIT

La cláusula "limit" se usa para restringir los registros que se retornan en una consulta "select".

Recibe 1 ó 2 argumentos numéricos enteros positivos; el primero indica el número del primer registro a retornar, el segundo, el número máximo de registros a retornar. El número de registro inicial es 0 (no 1).

Si el segundo argumento supera la cantidad de registros de la tabla, se limita hasta el último registro.

Ejemplo:

```
select * from libros limit 0,4;
```

Muestra los primeros 4 registros, 0,1,2 y 3.

Si tipeamos:

```
select * from libros limit 5,4;
```

recuperamos 4 registros, desde el 5 al 8.

Si se coloca un solo argumento, indica el máximo número de registros a retornar, comenzando desde 0. Ejemplo:

```
select * from libros limit 8;
```

Muestra los primeros 8 registros.

Para recuperar los registros desde cierto número hasta el final, se puede colocar un número grande para el segundo argumento:

```
select * from libros limit 6,10000;
```

recupera los registros 7 al último.

"limit" puede combinarse con el comando "delete". Si queremos eliminar 2 registros de la tabla "libros" Usamos:

```
delete from libros
 limit 2;
```

Podemos ordenar los regitros por precio (por ejemplo) y borrar 2:

```
delete from libros
 order by precio
 limit 2;
```

esta sentencia borrará los 2 primeros registros, es decir, los de precio más bajo.

Podemos emplear la cláusula "limit" para eliminar registros duplicados. Por ejemplo, continuamos con la tabla "libros" de una librería, ya hemos almacenado el libro "El aleph" de "Borges" de la editorial "Planeta", pero nos equivocamos y volvemos a ingresar el mismo libro, del mismo autor y editorial 2 veces más, es un error que no controla MySQL. Para eliminar el libro duplicado y que sólo quede un registro de él vemos cuántos tenemos:

```
select * from libros
 where titulo='El aleph' and
 autor='Borges' and
 editorial='Planeta';
```

Luego eliminamos con "limit" la cantidad sobrante (tenemos 3 y queremos solo 1):

```
delete from libros
 where titulo='El aleph" and
 autor='Borges' and
 editorial='Planeta'
 limit 2;
```

Un mensaje nos muestra la cantidad de registros eliminados.

Es decir, con "limit" indicamos la cantidad a eliminar.

Veamos cuántos hay ahora:

```
select * from libros
 where titulo='El aleph" and
 autor='Borges' and
 editorial='Planeta';
```

Sólo queda 1.