Curso de MySQL y Java

Recuperación de información: consultas en MySQL

1.Algunos detalles sobre consultas

MySQL utiliza como lenguaje para la recuperación y la definición de datos el lenguaje SQL con alguna peculiaridad.

- **Literales**
  - numéricos: 8, 0.0, 9.1 son números constantes
  - Cadenas de caracteres: 'Hola me llamo Esther' , "Hola me llamo Esther"
  - Tipo fecha: '2004- 07-12'-> 'año-mes-dia'
- **Valor NULL**:
- **Tipos de datos para las columnas**. Se pueden clasificar en tres tipos: numéricos, cadenas de caracteres y fecha/hora.
  - Tipos numéricos: `tinyint, smallint, mediumint, int, bigint, flota, double, decimal.`
  - Tipo fecha: `DATE, TIME, DATETIME, TIMESTAMP`
  - Tipo cadena: `char(n), varchar(n), tinyblob, tinytext,blob, text, mediumblob, mediumtext, longblob,longtext.`
  - Tipo enumerado: `enum('tipo1','tipo2','tipo3'…);`
  - Tipo set: set ('uno','dos', 'tres') (tipo conjunto)
- **Operadores:** a continuación se muestran los operadores que se pueden utilizar en MySQL en orden de precedencia:

```
:=
||, OR, XOR
&&, AND
BETWEEN,
=, <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN
|
&
-, +
*, /, DIV, %, MOD
^
- (unary minus), ~ (unary bit inversion)
NOT, !
```

2. Funciones útiles para realizar consultas

Funciones sobre cadenas

`ASCII(str)`

Returns the numeric value of the leftmost character of the string `str`. Returns `0` if `str` is the empty string. Returns `NULL` if `str` is `NULL`. `ASCII()` works for characters with numeric values from `0` to `255`.

```
mysql> SELECT ASCII('2');
        -> 50
```

`CHAR_LENGTH(str)`

Returns the length of the string `str`, measured in characters. A multi-byte character counts as a single character. This means that for a string containing five two-byte characters, `LENGTH()` returns `10`, whereas `CHAR_LENGTH()` returns `5`.

`CONCAT(str1,str2,...)`

Returns the string that results from concatenating the arguments. Returns `NULL` if any argument is `NULL`. May have one or more arguments. A numeric argument is converted to its equivalent string form.

```
mysql> SELECT CONCAT('My', 'S', 'QL');
        -> 'MySQL'
mysql> SELECT CONCAT('My', NULL, 'QL');
        -> NULL
```

`INSERT(str,pos,len,newstr)`

Returns the string `str`, with the substring beginning at position `pos` and `len` characters long replaced by the string `newstr`.

```
mysql> SELECT INSERT('Quadratic', 3, 4, 'What');
        -> 'QuWhattic'
```

This function is multi-byte safe.

`INSTR(str,substr)`

Returns the position of the first occurrence of substring `substr` in string `str`. This is the same as the two-argument form of `LOCATE()`, except that the arguments are swapped.

```
mysql> SELECT INSTR('foobarbar', 'bar');
        -> 4
mysql> SELECT INSTR('xbar', 'foobar');
        -> 0
```

`LOAD_FILE(file_name)`

Reads the file and returns the file contents as a string. The file must be located on the server, you must specify the full pathname to the file, and you must have the `FILE` privilege. The file must be readable by all and be smaller than `max_allowed_packet` bytes. If the file doesn't exist or cannot be read because one of the preceding conditions is not satisfied, the function returns `NULL`.

```
mysql> UPDATE tbl_name
           SET blob_column=LOAD_FILE('/tmp/picture')
           WHERE id=1;
```

`LOWER(str)`

Returns the string `str` with all characters changed to lowercase according to the current character set mapping (the default is ISO-8859-1 Latin1).

```
mysql> SELECT LOWER('QUADRATICALLY');
        -> 'quadratically'
```

`LOCATE(substr,str,pos)`

The first syntax returns the position of the first occurrence of substring `substr` in string `str`. The second syntax returns the position of the first occurrence of substring `substr` in string `str`, starting at position `pos`. Returns `0` if `substr` is not in `str`.

```
mysql> SELECT LOCATE('bar', 'foobarbar');
        -> 4
mysql> SELECT LOCATE('xbar', 'foobar');
        -> 0
mysql> SELECT LOCATE('bar', 'foobarbar',5);
        -> 7
```

This function is multi-byte safe. In MySQL 3.23, this function is case sensitive. For 4.0 on, it is case sensitive only if either argument is a binary string.

`REPLACE(str,from_str,to_str)`

Returns the string `str` with all occurrences of the string `from_str` replaced by the string `to_str`.

```
mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
        -> 'WwWwWw.mysql.com'
```

`REVERSE(str)`

Returns the string `str` with the order of the characters reversed.

```
mysql> SELECT REVERSE('abc');
        -> 'cba'
```

`RIGHT(str,len)`

Returns the rightmost `len` characters from the string `str`.

```
mysql> SELECT RIGHT('foobarbar', 4);
```

`UPPER(str)`

Returns the string `str` with all characters changed to uppercase according to the current character set mapping (the default is ISO-8859-1 Latin1).

```
mysql> SELECT UPPER('Hej');
        -> 'HEJ'
```

This function is multi-byte safe.

```
        -> 'rbar'
```

Comparación entre cadenas

Para comparar con patrones de cadenas se utiliza el operador LIKE. Existen dos caracteres comodines:

%	**Encaja cualquier número de carácteres**
_	**Corresponde con un único carácter**

```
mysql> SELECT 'David!' LIKE 'David_';
        -> 1
```

Nota: Si el patrón o la expresión es NULL `el resultado es NULL`.

```
mysql> SELECT * FROM pet WHERE name LIKE 'b%';
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
```

mysql> SELECT * FROM pet WHERE name LIKE '_____';

```
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

Para realizar búsquedas sobre texto existen dos operadores muy útiles en MySQL: `REGEXP y NOT REGEXP:`

Algunas características sobre las expresiones regulares:

- `.' , encaja con cualquier carácter.
- Un grupo de carácteres `[...] encaja cualquier carácter dentro del corchete. Por ejemplo [abc] encaja con los caracteres 'a', 'b' y 'c'. Para definir un intervalo: [a-z]
- '*' encaja con cero o más apariciones del carácter que le precede. Por ejemplo: 'x*' encaja con el character 'x', [0-9] encaja cualquier número de dígitos, y '.*' encaja cualquier número de repeticiones de cualquier cosa.
- El operador `REGEXP` tiene éxito si el patrón encaja en cualquier lugar del texto que está siendo analizado (al contrario que el operador LIKE)
- Para obligar a que el patrón encaje con el principio o final del texto buscado se utiliza '^' al principio o '$' al final del patrón.

Veamos algunos ejemplos del uso de las expresiones regulares:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^b';
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+

mysql> SELECT * FROM pet WHERE name REGEXP 'fy$';
+--------+--------+---------+------+------------+--------+
| name   | owner  | species | sex  | birth      | death  |
+--------+--------+---------+------+------------+--------+
| Fluffy | Harold | cat     | f    | 1993-02-04 | NULL   |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL   |
+--------+--------+---------+------+------------+--------+

mysql> SELECT * FROM pet WHERE name REGEXP 'w';
+----------+-------+---------+------+------------+------------+
| name     | owner | species | sex  | birth      | death      |
+----------+-------+---------+------+------------+------------+
| Claws    | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Bowser   | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
+----------+-------+---------+------+------------+------------+

mysql> SELECT * FROM pet WHERE name REGEXP '^.....$';
+-------+--------+---------+------+------------+--------+
| name  | owner  | species | sex  | birth      | death  |
+-------+--------+---------+------+------------+--------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL   |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL   |
+-------+--------+---------+------+------------+--------+
```

Búsquedas de texto completo

MySQL soporta una técnica llamada búsqueda de texto completo. En este tipo de búsquedas se pueden localizar apariciones de palabras entre los valores almacenados en columnas. La sintaxis para realizar una búsqueda de texto completo es la siguiente:

```
mysql>select * from tabla where MATCH (columna) AGAINST
('cadena');
```

El resultado de la consulta es la devolución de las filas en orden de relevancia, de mayor a menor. En otras palabras, se devolverán en primer lugar las filas en las cuales cadena presenta una coincidencia mayor con columna.

```
mysql> CREATE TABLE articles (
    ->   id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
    ->   title VARCHAR(200),
    ->   body TEXT,
    ->   FULLTEXT (title,body)
    -> );
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO articles (title,body) VALUES
    -> ('MySQL Tutorial','DBMS stands for DataBase ...'),
    -> ('How To Use MySQL Well','After you went through a ...'),
    -> ('Optimizing MySQL','In this tutorial we will show ...'),
    -> ('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
    ->  ('MySQL vs. YourSQL','In the following database comparison
...'),
    -> ('MySQL Security','When configured properly, MySQL ...');
Query OK, 6 rows affected (0.00 sec)
Records: 6  Duplicates: 0  Warnings: 0
mysql> SELECT * FROM articles
    -> WHERE MATCH (title,body) AGAINST ('database');
+----+-------------------+------------------------------------------+
| id | title             | body                                     |
+----+-------------------+------------------------------------------+
|  5 | MySQL vs. YourSQL | In the following database comparison ... |
|  1 | MySQL Tutorial    | DBMS stands for DataBase ...             |
+----+-------------------+------------------------------------------+
2 rows in set (0.00 sec)
```

Nota: Este tipo de búsqueda solo se puede utilizar con tablas tipo MyIsam.

Funciones de tipo fecha:

`NOW()`: Fecha y hora.

`CURDATE()`

Returns the current date as a value in `'YYYY-MM-DD'` or `YYYYMMDD` format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT CURDATE();
        -> '1997-12-15'
mysql> SELECT CURDATE() + 0;
        -> 19971215
```

`DATEDIFF(expr,expr2)`

`DATEDIFF()` returns the number of days between the start date `expr` and the end date `expr2`. `expr` and `expr2` are date or date-and-time expressions. Only the date parts of the values are used in the calculation.

```
mysql> SELECT DATEDIFF('1997-12-31 23:59:59','1997-12-30');
```

```
        -> 1
mysql> SELECT DATEDIFF('1997-11-30 23:59:59','1997-12-31');
        -> -31
```

`DATEDIFF()` was added in MySQL 4.1.1.

```
DATE_ADD(date,INTERVAL expr type)
DATE_SUB(date,INTERVAL expr type)
```

These functions perform date arithmetic. `date` is a `DATETIME` or `DATE` value specifying the starting date. `expr` is an expression specifying the interval value to be added or subtracted from the starting date. `expr` is a string; it may start with a `-' for negative intervals. `type` is a keyword indicating how the expression should be interpreted. The `INTERVAL` keyword and the `type` specifier are not case sensitive. The following table shows how the `type` and `expr` arguments are related:

type **Valor**	**La expresión debe ser**
SECOND	SECONDS
MINUTE	MINUTES
HOUR	HOURS
DAY	DAYS
WEEK	WEEKS
MONTH	MONTHS
YEAR	YEARS

```
mysql> SELECT INTERVAL 1 DAY + '1997-12-31';
        -> '1998-01-01'
mysql> SELECT '1998-01-01' - INTERVAL 1 SECOND;
        -> '1997-12-31 23:59:59'
mysql> SELECT DATE_ADD('1997-12-31 23:59:59',
    ->                 INTERVAL 1 DAY);
        -> '1998-01-01 23:59:59'
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
        -> '1997-12-02'
```

3. Más cosas específicos de mysql para realizar consultas.

En MySQL se pueden utilizar variables de usuario para recordar resultados sin tener que almacenarlos temporalmente en el cliente. Por ejemplo, si se desea encontrar los artículos más caros y más baratos se podría realizar lo siguiente (equivalente a una subconsulta):

```
mysql> SELECT @min_precio:=MIN(precio),@max_precio:=MAX(precio) FROM
tienda;
mysql> SELECT * FROM tienda WHERE precio=@min_precio OR
precio=@max_precio;
+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0003 | D      |  1.25 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

Administración de mysql

1. Realización de copias de seguridad

Realizar una copia de seguridad implica, en mysql copiar el directorio donde mysql almacena todos los datos (normalmente en '**pathmysql\data**'. Ahí las BD y las tablas se almacenan con una estructura de directorios. Por ejemplo si ha creado una BD librería, ese directorio contendrá un subdirectorio con ese nombre y dentro de él se encuentran unos ficheros para cada una de las tablas creadas.
Si se utiliza como tipo de tablas el myIsam, habrá 3 ficheros por tabla (uno almena el diseño de la tabla, otra los datos y otra los índices creados para esa tabla):

- `*.frm` : definición de la tabla.
- `*.MYD`: fichero de datos.
- `*.MYI`: fichero de índices.

Si por el contrario utiliza las tablas tipo `InnoDB,` el diseño de la tabla se guarda en un fichero con extensión `.frm`, mientras que el resto de información se guarda en un fichero ibdata1 (se puede modificar este nombre en el fichero de configuración).

Una copia de seguridad puede ser algo tan simple como copiar el directorio de datos (con la base de datos parada, obviamente).

No obstante, Mysql proporciona una herramienta muy útil conocida como mysqldump que permite copiar la información. Lo que hace es generar las sentencias SQL necesarias para reconstruir la BD.
Ejemplo de utilización:

```
mysql> mysqldump –u root –p --opt biblioteca > mifichero.sql
```

La orden anterior creará un fichero mifichero.sql con todas las sentencias sql necesarias para crear las tablas e insertar los datos.

Nota para poder utilizar este programa es necesario que la BD esté en marcha.

Opciones interesantes de mysqldump:

- **`--add-drop-table:`** añade código a la salida de maysqldump para que borrar una tabla antes de crearla.
- **`-d`** que especifica que deben volcarse los datos, pero no la estructura.
- **`-t`** que permite indicar que se desea volcar la estructura pero no los datos.
- **`-T`** que permite indicar un directorio donde debe almacenarse la salida como una serie de archivos.
- **`--opt`**

Para exportar los datos de una determinada table a un fichero externo se puede utilizar la sentencia SELECT * INTO OUTFILE:

```
mysql> SELECT * FROM libros INTO OUTFILE 'ELFICHERO.TXT';
```

La sentencia anterior creará un fichero con el contenido de la tabla 'libros';

2. Para importar datos

Mysql dispone de una herramienta que permite cargar datos dentro de las tablas de la BD. Esta herramienta se llama 'mysqlimport'.
Esta herramienta es similar a la instrucción **LOAD DATA INFILE.** La mayoría de las opciones corresponden directamente a opciones del comando anterior.

```
shell> mysqlimport [options] db_name textfile1 [textfile2 ...]
```

Ejemplo: supongamos que tenemos un fichero con el nombre socios.txt que contiene los registros que queremos cargar en la tabla socios:

```
socios.txt
1       Esther  Esther.Deves@uv.es      pdi
2       Juan    Juan.Gutierez@uv.es     \N
```

```
shell> mysqlimport –u usuario –p biblioteca c:\tmp\socios.txt
```

Nota: El carácter '\N' es la manera de representar un valor NULL. Los campos se separan con tabulaciones y las filas, por cambios de línea.

Esta herramienta es similar a la sentencia **LOAD DATA INFILE:**

```
mysql> LOAD DATA INFILE 'C:\\TMP\\SOCIOS.TXT' into table socios
```

3. Mantenimiento de la BD

Existe una herramienta que permite comprobar el estado de las tables que componen la BD. Esta herramienta se llama `myisamchk`.

```
shell> myisamchk [options] archivos_de_tabla
```

Las opciones más habituales de esta herramienta son:

- -a analiza la tabla
- -r repara la tabla
- -e realiza reparaciones extendidas.

En general es aconsejable utilizar myisamchk –a y myisamchk –e periódicamente. Si se encuentra con problemas serios utiliza la opción '-e' (la ejecución es más lenta).

Esta herramienta únicamente funciona con tablas MyIsam:
Comprobar la tabla socios:

```
shell> myisamchk –a c:\mysql\data\biblioteca\socios.MYI
```

Para comprobar todas las tablas dentro del directorio:

```
shell> myisamchk –a c:\mysql\data\biblioteca\*.MYI
```

La comprobación de tables se puede realizar directamente desde el cliente de mysql, ejecutando la siguiente sentencia:

```
mysql> CHECK TABLE socios;
```

A diferencia de la herramienta anterior, esta sentencia también funciona para tablas de tipo InnoDB.
En caso de que se encuentre algún error, se debería ejecutar la siguiente sentencia SQL:

```
mysql> REPAIR TABLE socios EXTENDED;
```

4. Optimización de tablas

Para optimizar y mejorar el rendimiento de la BD se recomienda optimizar las tablas mediante la sentencia SQL

```
mysql> OPTIMIZE TABLE mitabla;
```

Esta sentencia sirve para cualquier tablas tipo MyIsam, Innodb y BDB y como resultado devuelve una tabla indicando como ha ido el proceso.

Debería ejecutarse esta sentencia si se han borrado una gran cantidad de de datos de una tabla o se han modificado tablas con filas de tamaño variable (tipos varchar, blob o text). Se puede utilizar `OPTIMIZE TABLE` para reutilizar el espacio no utilizado y defragmentar el fichero de datos.

La herramienta myisamchk se puede utilizar también para optimizar las consultas mediante la opción –sort-index (recordad que solo es válido para tablas myisam).

```
shell> myisamchk –-sort-index c:\mysql\data\biblioteca\*.MYI
```

Esta ejecución lo que hace es reordenar la estructura de índice en orden de mayor a menor y así optimiza las búsquedas y hace que el acceso por clave más rápido.

5. Traza del sistema

Por defecto Mysql no trabaja con la traza del sistema (útil para la recuperación de la BD en caso de fallo). Se puede hacer que el servidor arranque con esta opción:

```
shell> mysqld  –log-update
```

```
shell> mysqld  –log-bin
```

Estas dos opciones crearán dos ficheros con el nombre del localhost en el directorio de datos (por ejemplo: c:\mysql\data\nonino.001 o c:\mysql\data\nonino-bin, para el binario). Estos ficheros se pueden visualizar con cualquier editor de texto.

De estas dos opciones se recomienda la '–log-bin' puesto que guarda todas las modificaciones realizadas por la BD en un formato más eficiente que con la opción '–log-update'.

6. Trabajar con transacciones en MySQL

MySQL en las últimas versiones soporta transacciones (siempre que se trabaje con trabajas de tipo InnoDB). Las transacciones aportan una fiabilidad superior a las bases de datos. Si dispone de una serie de consultas que deben aplicarse o no en su conjunto, con el manejo de transacciones puede tener la certeza de que nunca se quedará a medio camino en su ejecución.

Las tablas a prueba de transacciones (InnoDB) son mucho más seguras y fáciles de recuperar si se produce algún fallo en el servidor. En cambio, las transacciones pueden hacer que las consultas tarden más en ejecutarse.

Veamos la sintaxis de una transacción:

```
mysql > begin %Comienzo de transacción
mysql > sentencias SQL sobre la BD:
mysql > COMMIT; %O ROLLBACK %Final de transacción
```

Por defecto, MySQL funciona en modo autocommit. Esto significa que los cambios se guardan a disco tan pronto se realizan.

Para deshabilitar este modo:

```
mysql >SET AUTOCOMMIT=0;
```

Si quieres deshabilitar el modo autocommit para un conjunto de sentencias, se puede utilizar la sentencia start transaction:

If you want to disable autocommit mode for a single series of statements, you can use the START TRANSACTION statement. Ejemplo:

```
START TRANSACTION;
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;
UPDATE table2 SET summary=@A WHERE type=1;
COMMIT;
```

With START TRANSACTION, autocommit remains disabled until you end the transaction with COMMIT or ROLLBACK. The autocommit mode then reverts to its previous state.

Si se utiliza la sentencia ROLLBACK después de actualizar un tipo de tabla que no soporta transacciones, MySQL devolverá un warning:

8. Bloqueo de tablas

Las transacciones permiten proteger os datos y la integridad de una tabla al restringir las acciones que tienen lugar sobre la misma, pero sólo se puede aplicar a tablas a prueba de transacciones. Para tablas de otro tipo, incluido el MyISAM predeterminado, la única opción disponible para proteger las tablas durante una sucesión de consultas consiste en bloquearlas. La sintaxis para llevar a cabo un bloqueo es la siguiente:

```
LOCK TABLES
    tbl_name [AS alias] {READ [LOCAL] | [LOW_PRIORITY] WRITE}
    [, tbl_name [AS alias] {READ [LOCAL] | [LOW_PRIORITY] WRITE}] ...
UNLOCK TABLES
```

Al bloquear una tabla puede especificar un bloqueo de tipo READ, WRITE. Un bloque de lectura restringe el acceso para que únicamente se pueda leer la tabla. Con un bloqueo de escritura se impide la lectura y la escritura a cualquier usuario distinto de aquel que ha ejecutado la orden de bloqueo.

Para poder bloquear una tabla se debe tener los privilegios LOCK TABLES y SELECT..

Casos en los que conviene bloquear una tabla:

- Si se van a ejecutar muchas operaciones sobre tablas MyISAM. Si las tablas se bloquean las operaciones se ejecutan más rápido.
- Si se trabaja con un tipo de tablas que no soportan transacciones, se deben utilizar bloqueos cuando se quiere asegurar que ninguna otra operación se realizará entre un select y un update. Ejemplo:

Ejemplo:

```
mysql> LOCK TABLES trans READ, clientes WRITE;
mysql> SELECT @suma:=SUM(value) FROM trans WHERE --
customer_id=5;
mysql> UPDATE cliente set total_valor= @suma where
cliente_id=5;
mysql> UNLOCK TABLES
```

9. Optimización de consultas en mysql

Existe una setencia SQL que permite ver cual será el plan que MySQL va a utilizar para resolver la consulta. Esto puede ser muy útil pues permite, si este no es muy bueno, ayudar al optimizador para que encuentre un plan mejor.

```
mysql> explain select * from tabla1
```