[](http://www.google.com.pe/imgres?imgurl=http://cmop17.files.wordpress.com/2009/06/logo-mysql.jpg&imgrefurl=http://cmop17.wordpress.com/2009/06/21/conectar-java-con-servidor-mysql-5-1/&usg=__5eU5N_ucw-QvizIDGqZsXxZ4qgc=&h=291&w=399&sz=17&hl=es&start=1&um=1&itbs=1&tbnid=iKD7EGeAVvZDxM:&tbnh=90&tbnw=124&prev=/images?q=mysql&um=1&hl=es&sa=N&tbs=isch:1) BASE DE DATOS MYSQL

**CLAVE PRIMARIA**

Una Clave Primaria (PK) es una columna o conjunto de columnas que únicamente identifica cada fila en una tabla. Cada tabla debe tener una clave primaria y una clave primaria debe ser única.

Ejemplo

La clave primaria para la tabla EMPLEADOS consiste en la columna NUM\_EMP. Cada fila en la tabla es únicamente identificada por su valor NUM\_EMP.



Discusión

* No se permiten los duplicados en una Clave Primaria. La clave primaria debe ser única.
* Las claves primarias generalmente no pueden ser cambiadas.
* El UID de una entidad se mapeará a una Clave Primaria en su correspondiente tabla.

**CLAVES PRIMARIAS COMPUESTAS**

Una Clave Primaria que se conforma de múltiples columnas se llama Clave Primaria Compuesta.

Ejemplo

La clave primaria compuesta para la tabla CUENTA consiste en la combinación de las columnas NUM\_BANCO y NUM\_CUENTA. Cada fila es unicamente identificada por sus valores de NUM\_BANCO y NUM\_CUENTA.



**Discusión**

* Las columnas de un clave primaria compuesta debe ser única en la combinación. Las columnas individuales pueden tener duplicados, pero en la combinación, no se permiten duplicados.

**CLAVES PRIMARIAS NO NULAS**

Ninguna parte de una clave primaria puede ser NULL (nula).

Ejemplo

NUM\_EMP es la clave primaria de la tabla EMPLEADOS. Por lo tanto NUM\_EMP debe ser definido como NOT NULL.



Ejemplo

¿Cómo viola la tabla CUENTA las reglas de las Claves Primarias?



Dos de las filas contienen valores NULL en parte de la clave primaria compuesta. Tanto NUM\_BANCO como NUM\_CUENTA deben ser definidos como NOT NULLS.

**CLAVES CANDIDATAS Y CLAVES ALTERNAS**

Una tabla puede tener más de una columna o combinación de columnas que pueden servir como la clave primaria de la tabla. Cada una de ellas se llama Clave Candidata.

Ejemplo

¿Cuáles son las claves candidatas para la tabla EMPLEADOS?



NUM\_EMP e ID\_PLASUELD son claves candidatas.

Seleccione una clave candidata para ser la Clave Primaria para la tabla. Las otras candidatas se convierten en Claves Alternas.

Ejemplo



Discusión: Todas las Claves Candidatas deben ser únicas y NOT NULL.

Los UIDs secundarios se mapean a claves alternas.

**CLAVES FORÁNEAS**

Una Clave Foránea (FK) es una columna o combinación de columnas en una tabla que es clave primaria en la misma u otra tabla. Ejemplo

NUM\_DEPT es una FK en la Tabla EMPLEADOS y se refiere a los valores en la columna NUM\_DEPT en la Tabla DEPARTAMENTO.



Discusión

* Las claves foráneas son la implementación de las relaciones de un modelo de datos conceptual.
* Las claves foráneas se basan en los valores de los datos y son puramente lógicas.

Una clave foránea siempre debe parearse con el valor de una clave primaria (o de lo contrario ser NULL).

**Ejemplo**

La FK NUM\_DEPT en la tabla EMPLEADOS se refiere a los valores de la PK NUM\_DEPT en la tabla DEPARTAMENTO.



Si una Clave Foránea es parte de una Clave Primaria, esa FK no puede ser NULL.

**Ejemplo**

En la tabla CUENTA, la FK NUM\_BANCO debe ser NOT NULL porque es parte de la PK.



**INTEGRIDAD DE DATOS**

Las reglas de integridad de datos, aseguran la exactitud y consistencia de los datos.

**Restricciones a la Integridad de Datos**

* Las restricciones a la integridad de los datos definen el estado relacionalmente correcto de una base de datos relacional.
* Las restricciones de integridad aseguran que los usuarios realicen sólo operaciones que dejen a la base de datos en un estado consistente y correcto.

|  |  |
| --- | --- |
| **Tipo de Restricción** | Explicación |
| **Integridad de Entidad** | Ninguna parte de una clave primaria puede ser NULL. |
| **Integridad Referencial** | Toda clave foránea debe ser validada en la tabla que es clave primaria. |
| **Integridad de Columnas** | Una columna debe contener sólo valores consistentes con el formato de datos definidos para la columna. |
| **Integridad Definida por el usuario** | Los datos almacenados en una Base de Datos deben cumplir con las reglas del negocio. |

Todas las restricciones a la integridad de los datos deberían ser ejecutadas por el SABDR o por el software de aplicación.DiscusiónLos datos son inconsistentes si existen múltiples copias de una entrada y no todas las copias han sido actualizadas. Una base de datos inconsistente podría suministrar información incorrecta o contradictoria a sus usuarios.

* Las restricciones a la integridad de los datos definidos por el usuario pueden ser establecidas por una política de gestión o ser requeridas por leyes gubernamentales.
* Las restricciones a la integridad de los datos definidos por el usuario podrían involucrar múltiples columnas y tablas.

**Tipos de Datos**

Características de MySQL

* + Modelo relacional, multiusuario

Tipos de datos

|  |  |  |
| --- | --- | --- |
| **Tipo de datos de cadena de caracteres** | | |
| **CHAR** | | Es un sinónimo de CHAR(1), y puede contener un único carácter. |
| **CHAR(M)** | | Contiene una cadena de longitud constante.Para mantener la longitud de la cadena,se rellena ala drecha con espacios.Estos espacios se eliman al recuperar el valor.  Los valores válidos para M son de 0 a255 y de 1 a 255 para versiones de MySQL previas a 3.23 |
| **VARCHAR()** | | Contiene una cadena de longitud variable.Los valores válidos para M son de 0 a 255 |
| **Tipo de datos enteros** | | |
| **TINYINT** | | Contiene un valor entero muy pequeño.El rango con signo es entre -128 y 127.El Rango sin signo es de 0 a 255 |
| **BIT,BOOL,BOOLEAN** | | Contiene un entero corto.El Rango con signo es de -32768 a 32767 |
| **MEDIUM** | | Contiene un etero de tamaño medio,el rango está entre -8388608 y 8388607.El rango sin signo,entre 0 y 16777215 |
| **INT** | | Contiene un entero de tamaño normal.El rango con signo esta entre -2147483648 |
| **SMALLINT** | | número entero con o sin signo. Con signo el rango de valores va desde -32768 a 32767. Sin signo, el rango de valores es de 0 a 65535. |
| **INTEGER** | | Es sinónimo de INT |
| **BIGINT** | | Contiene un entero grande.El rango con signo es de -9223372036854775808 a 9223372036854775807.El rango sin signo de 0 a 18446744073709551615. |
| **Tipos de datos en coma flotante** | | |
| **FLOAT** | Contiene un numero en coma flotante de doble precision sencilla y entre 25 y 53 para numeros en coma flotante de doble precisión.Estos tipos son idénticos que los tipos FLOAT y DOUBLE descritos a continuación | |
| **xReal, Double** | número en coma flotante de precisión doble. Los valores permitidos van desde -1.7976931348623157E+308 a -2.2250738585072014E-308, 0 y desde 2.2250738585072014E-308 a 1.7976931348623157E+308 | |
| **Decimal, Dec, Numeric:** | Número en coma flotante desempaquetado. El número se almacena como una cadena | |
| **Tipos de datos Fecha** | | |
| A la hora de almacenar fechas, hay que tener en cuenta que Mysql no comprueba de una manera estricta si una fecha es válida o no. Simplemente comprueba que el mes esta comprendido entre 0 y 12 y que el día esta comprendido entre 0 y 31.  Date: tipo fecha, almacena una fecha. El rango de valores va desde el 1 de enero del 1001 al 31 de diciembre de 9999. El formato de almacenamiento es de año-mes-dia | | |
| **Date:** | | tipo fecha, almacena una fecha. El rango de valores va desde el 1 de enero del 1001 al 31 de diciembre de 9999. El formato de almacenamiento es de año-mes-dia |
| **DateTime** | | Combinación de fecha y hora. El rango de valores va desde el 1 de enero del 1001 a las 0 horas, 0 minutos y 0 segundos al 31 de diciembre del 9999 a las 23 horas, 59 minutos y 59 segundos. El formato de almacenamiento es de año-mes-dia horas:minutos:segundos |
| **TimeStamp:** | | Combinación de fecha y hora. El rango va desde el 1 de enero de 1970 al año 2037. El formato de almacenamiento depende del tamaño del campo |
| **Time:** | | almacena una hora. El rango de horas va desde -838 horas, 59 minutos y 59 segundos a 838, 59 minutos y 59 segundos. El formato de almacenamiento es de 'HH:MM:SS' |
| **Year** | | almacena un año. El rango de valores permitidos va desde el año 1901 al año 2155. dígitos |

|  |  |
| --- | --- |
| Tipos de datos para datos sin tipo o grandes bloques de datos | |
| **TinyText y TinyBlob** | Columna con una longitud máxima de 255 caracteres |
| **Blob y Text:** | un texto con un máximo de 65535 caracteres. |
| **MediumBlob y MediumText** | un texto con un máximo de 16.777.215 caracteres. |
| **LongBlob y LongText** | un texto con un máximo de caracteres 4.294.967.295. Hay que tener en cuenta que debido a los protocolos de comunicación los paquetes pueden tener un máximo de 16 Mb. |
| **Enum** | campo que puede tener un único valor de una lista que se especifica. El tipo Enum acepta hasta 65535 valores distintos |
| **Set** | un campo que puede contener ninguno, uno ó varios valores de una lista. La lista puede tener un máximo de 64 valores. |

#### Comandos

Existen dos tipos de comandos:

* Los **DDL** que permiten crear y definir nuevas bases de datos, campos e índices.
* Los **DML** que permiten generar consultas para ordenar, filtrar y extraer datos  
  de la base de datos.

#### Comandos DDL:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Comando** | **Descripción** | | **CREATE** | Utilizado para crear nuevas tablas, campos e índices | | **DROP** | Empleado para eliminar tablas e índices | | **ALTER** | Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos. | |

#### Comandos DML

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Comando** | **Descripción** | | **SELECT** | Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado | | **INSERT** | Utilizado para cargar lotes de datos en la base de datos en una única operación. | | **UPDATE** | Utilizado para modificar los valores de los campos y registros especificados | | **DELETE** | Utilizado para eliminar registros de una tabla de una base de datos | |

#### Cláusulas

Las cláusulas son condiciones de modificación utilizadas para  
definir los datos que desea seleccionar o manipular.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Comando** | **Descripción** | | **FROM** | Utilizada para especificar la tabla de la cual se van a seleccionar los registros | | **WHERE** | Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar | | **GROUP** **BY** | Utilizada para separar los registros seleccionados en grupos específicos | | **HAVING** | Utilizada para expresar la condición que debe satisfacer cada grupo | | **ORDER** **BY** | Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico | |

#### Operadores

#### Operadores Lógicos

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Operador** | **Uso** | | **AND** | Es el “y” lógico. Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas. | | **OR** | Es el “o” lógico. Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos es cierta. | | **NOT** | Negación lógica. Devuelve el valor contrario de la expresión. |  Operadores Aritméticos +, -, \*, / |

#### Operadores de Comparación

#### 

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Operador** | **Uso** | | **<** | Menor que | | **>** | Mayor que | | **<>** | Distinto de | | **<=** | Menor ó Igual que | | **>=** | Mayor ó Igual que | | **BETWEEN** | Utilizado para especificar un intervalo de valores. | | **LIKE** | Utilizado en la comparación de un modelo | | **In** | Utilizado para especificar registros de una base de datos | |

#### IS NULL Utilizado para indicar valores nulos

**IS NOT NULL Valores no nulos**

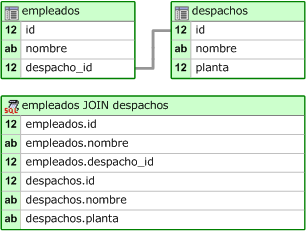
**Operadores de comparación de cadenas de caracteres**   
  
Para comparar cadenas de caracteres utilizamos el símbolo =. Ponemos un ejemplo.   
Select \* from emple where oficion='analista'   
  
Este operador no nos sirve si queremos hacer una consulta del tipo de: "sacar los datos del empleado cuyo oficio empiece por P"   
Para especificar este tipo de consultas, en SQL usamos el operador LIKE que permite utilizar los siguientes caracteres especiales en las cadenas de comparación:

**% Comodín: Representa cualquier cadena de 0 o más caracteres   
'\_' : Marcador de posición: representa un carácter cualquiera.**

**Ejemplo:**

Select \* from emple where nombre LIKE 'P%'   
  
Esta consulta nos mostraría todos los datos de los empleados cuyo nombre empezará por **P**

**OPERADORES RELACIONALES (JOIN)**

Un join es una combinación de dos o más tablas de una base de datos relacional, es una instrucción de lo más imprescindible si queremos realizar una aplicación que realice un uso correcto e intensivo de un gestor de Bases de datos cómo MySQL.  
Y es que gracias a los JOIN podemos leer datos de diferentes bases de datos en una única consulta. Lo que nos permitirá diseñar fácilmente tablas relacionadas entre ellas, veamos un ejemplo:

**Figura 1:** En este ejemplo podemos ver una relación entre las tablas de **empleados** y de **despachos**, y la representación de un **join** **entre ambas**

**MySQL** soporta [muchas sintaxis diferentes](http://dev.mysql.com/doc/refman/5.0/es/join.html) para la instrucción JOIN en comandos de SELECT, DELETE y UPDATE.  
**A continuación se explican, con sus ejemplos en MySQL, los principales tipos de JOIN.**

**INNER JOIN:**

Con esta operación se calcula el producto cruzado de todos los registros; así cada registro en la tabla A es combinado con cada registro de la tabla B; pero sólo permanecen aquellos registros en la tabla combinada que satisfacen las condiciones que se especifiquen. Este es el tipo de JOIN más utilizado por lo que es considerado el tipo de combinación predeterminado.

**A. Ejemplo de la sentencia INNER JOIN explícita:**

SELECT \* FROM empleado INNER JOIN departamento ON empleado.IDdepartamento = departamento.IDdepartamento

**B. Ejemplo de la sentencia INNER JOIN implícita:**

SELECT \* FROM empleado, departamento WHERE empleado.IDdepartamento = departamento.IDDepartamento

**Resultados:**

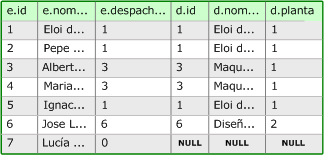
|  |  |  |  |
| --- | --- | --- | --- |
| **Empleado.Apellido** | **Empleado.IDdepartamento** | **departamento.NombreDepartamento** | **departamento.IDDepartamento** |
| Solano | 34 | Producción | 34 |
| Jordán | 33 | Ingeniería | 33 |
| Róbinson | 34 | Producción | 34 |
| Steinberg | 33 | Ingeniería | 33 |
| Andrade | 31 | Ventas | 31 |

El empleado Gaspar y el departamento de Mercadeo no son presentados en los resultados ya que ninguno de éstos tiene registros correspondientes en la otra tabla. No existe un departamento con número 36 ni existe un empleado con número de departamento 35.

**LEFT (RIGHT) JOIN**

LEFT JOIN retorna los registros de la primera tabla aunque no coincidan con ninguno de la otra.  
RIGHT JOIN realiza lo mismo pero en base a la segunda tabla. 

|  |  |
| --- | --- |
| 000 | SELECT \* FROM empleados LEFT JOIN despachos ON empleados.despacho\_id = despachos.id |



**NATURAL JOIN**  
Este JOIN retorna los registros de ambas tablas que coincidan en todos los campos con el mismo nombre ( en el ejemplo empleados.nombre y despachos.nombre ).

|  |  |
| --- | --- |
| 000 | SELECT \* FROM empleados NATURAL JOIN despachos |

join_result_03

## 

**Funciones**

**Funciones de cadena**

Las funciones de cadena trabajan con campos char y varchar por lo que los literales que escribamos se deben encerrar entre comillas simples.   
  
Estas funciones que vamos a explicar a continuación pueden manipular cadenas de letras u otros caracteres por lo que las vamos a dividir en dos grupos:   
  
**Funciones que devuelven caracteres**   
  
Este tipo de funciones devuelven un carácter o varios caracteres.

|  |  |
| --- | --- |
| Función | Propósito |
| CHR(n) | Nos devuelve el carácter cuyo valor en binario es n |
| CONCAT(cad1, cad2) | Nos devuelve cad1 concatenada con cad2 |
| UPPER(cad) | Convierte cad a mayúsculas |
| LOWER(cad) | Convierte cad a minúsculas |
| LPAD(cad1,n[,cad2]) | Con esta función añadimos caracteres a cad1 por la izquierda hasta una longitud máxima dada por n |
| INITCAP(cad) | Convierte la primera letra de cad a mayúscula |
| LTRIM(cad [,set]) | Elimina un conjunto de caracteres a la izquierda de cad, siendo set el conjunto de caracteres a eliminar |
| RPAD(cad1, n[,cad2]) | Con esta función añadimos caracteres de la misma forma que con la función LPAD pero esta vez los añadimos a la derecha |
| RTRIM(cad[,set]) | Hace lo mismo que LTRIM pero por la derecha |
| REPLACE(cad,cadena\_buscada [,cadena\_sustitucion] ) | Sustituye un conjunto de caracteres de 0 o más caracteres, devuelve cad con cada ocurrencia de cadena\_buscada sustituida por cadena\_sustitucion |
| SUBSTR(cad, m[,n]) | Devuelve la subcadena de cad que abarca desde m hasta el numero de caracteres dados por n. |
| TRANSLATE(cad1,cad2,cad3) | Convierte caracteres de una cadena en caracteres diferentes. Devuelve cad1 con los caracteres encontrados en cad2 y sustituidos por los caracteres de cad3 |

Ponemos algunos ejemplos de utilización de estas funciones:   
  
Sentencia sql que nos devuelve las letras cuyo valor asccii es el 45 y el 23   
  
select CHR(45), CHR(23) FROM TABLA;   
  
Sentencia sql que obtiene el nombre de los alumnos sacando por pantalla la siguiente frase: el nombre del alumno es (nombre que esta almacenado en la tabla)   
  
select CONCAT ('el nombre de alumno es', nombre) from alumno;   
  
Sentencia sql que me devuelve los nombres de los alumnos en mayúsculas   
  
select UPPER(nombre) from alumno;   
  
Sentencia sql que obtiene de un campo nombre, las 3 primeras letras   
  
select SUBSTR(nombre,0,3) from alumno;   
  
Y asi con el resto de funciones…

**Funciones que devuelven valores numéricos**   
  
Estas funciones nos devuelven números a modo de información.

|  |  |
| --- | --- |
| Función | Propósito |
| ASCII(cad) | Devuelve el valor ASCII de la primera letra de cad |
| INSTR(cad1, cad2[,comienzo[,m]]) | Función que busca un conjunto de caracteres dentro de una cadena. Nos devuelve la posición de cad2 en cad1 empezando a buscar en comienzo |
| LENGTH(cad) | Devuelve en número de caracteres de cad |

Sentencia sql que nos devuelve el valor ASCII de la letra ('s')   
  
select ASCII('s') from tabla;   
  
Sentencia que nos devuelve la posición de la ocurrencia 'pe' dentro de la cadena 'Los perros están bien' a partir de la posición 2   
  
select INSTR('Los perros están bien','pe',2) from tabla;   
  
Sentencia sql que nos devuelve el numero de caracteres de los nombres de los alumnos   
  
select LENGTH(nombre) from alumnos;

**Funciones de comparación de cadenas de caracteres**

MySQL convierte automáticamente números a cadenas según es necesario y viceversa.

mysql> SELECT 1+'1';

-> 2

mysql> SELECT CONCAT(2,' test');

-> '2 test'

Si quiere convertir un número a cadena explícitamente, use la función CAST() :

mysql> SELECT 38.8, CAST(38.8 AS CHAR);

-> 38.8, '38.8'

Si una función de cadena da una cadena binaria como argumento, la cadena resultante también es binaria. Un número convertido a cadena se trata como cadena binaria (esto es, es sensible a mayúsculas en comparaciones). Esto afecta sólo a comparaciones.

Normalmente, si una expresión en una comparación de cadenas es sensible a mayúsculas, la comparación se realiza con sensibilidad a mayúsculas.

* expr LIKE pat [ESCAPE 'escape-char']

Coincidencia de patrones usando comparación mediante expresiones regulares SQL. Retorna 1 (TRUE) o 0 (FALSE). Si expr o pat es NULL, el resultado es NULL.

El patrón no puede ser una cadena literal. Por ejemplo, puede especificarse como expresión de cadena o columna.

Con LIKE puede usar los siguientes dos caracteres comodines en el patrón:

|  |  |
| --- | --- |
| **Carácter** | **Descrición** |
| % | Coincidencia de cualquier número de caracteres, incluso cero caracteres |
| \_ | Coincide exactemente un carácter |

mysql> SELECT 'David!' LIKE 'David\_';

-> 1

mysql> SELECT 'David!' LIKE '%D%v%';

-> 1

Para testear instancias literales de un carácter comodín, preceda el carácter con el carácter de escape. Si no especifica el carácter ESCAPE , se asume '\' .

|  |  |
| --- | --- |
| **Cadena** | **Descrición** |
| \% | Coincide un carácter '%' |
| \\_ | Coincide un carácter '\_' |

mysql> SELECT 'David!' LIKE 'David\\_';

-> 0

mysql> SELECT 'David\_' LIKE 'David\\_';

-> 1

Para especificar un carácter de escape distinto, use la cláusula ESCAPE :

mysql> SELECT 'David\_' LIKE 'David|\_' ESCAPE '|';

-> 1

Los siguientes dos comandos ilustran que la comparación de cadenas no son sensibles a mayúsculas a no ser que uno de los operandos sea una cadena binaria:

mysql> SELECT 'abc' LIKE 'ABC';

-> 1

mysql> SELECT 'abc' LIKE BINARY 'ABC';

-> 0

En MySQL, LIKE se permite en expresiones numéricas. (Esta es una extensión del SQL estándar LIKE.)

mysql> SELECT 10 LIKE '1%';

-> 1

**Nota**: Debido a que MySQL usa sintaxis de escape C en cadenas (por ejemplo, '\n' para representar carácter de nueva línea), debe doblar cualquier '\' que use en cadenas LIKE . Por ejemplo, para buscar '\n', especifíquelo como '\\n'. Para buscar '\', especifíquelo como '\\\\'; esto es debido a que las antibarras se eliminan una vez por el parser y otra vez cuando la coincidencia con el patrón se realiza, dejando una única antibarra para comparar.

* expr NOT LIKE pat [ESCAPE 'escape-char']

Es lo mismo que NOT (expr LIKE pat [ESCAPE 'escape-char']).

* expr NOT REGEXP pat, expr NOT RLIKE pat

Es lo mismo que NOT (expr REGEXP pat).

* expr REGEXP pat, expr RLIKE pat

Realiza una comparación de patrones de una expresión de cadena de caracteres expr contra un patrón pat. El patrón puede ser una expresión regular extendida. La sintaxis para expresiones regulares se discute en [Apéndice F, Expresiones regulares en MySQL](http://dev.mysql.com/doc/refman/5.0/es/regexp.html). Retorna 1 si expr coincide con pat, de otro modo retorna 0. Si expr o pat es NULL, el resultado es NULL. RLIKE es un sinónimo de REGEXP, debido a compatibilidad con mSQL.

El patrón no necesita ser una cadena literal. Por ejemplo, puede especificarse como una expresión de cadena o columna.

**Nota:** Debido a que MySQL usa la sintaxis de escape de C en cadenas (por ejemplo, '\n' para representar una nueva línea), de doblar cualquier '\' que use en sus cadenas REGEXP .

REGEXP no es sensible a mayúsculas, excepto cuando se usa con cadenas binarias.

mysql> SELECT 'Monty!' REGEXP 'm%y%%';

-> 0

mysql> SELECT 'Monty!' REGEXP '.\*';

-> 1

mysql> SELECT 'new\*\n\*line' REGEXP 'new\\\*.\\\*line';

-> 1

mysql> SELECT 'a' REGEXP 'A', 'a' REGEXP BINARY 'A';

-> 1 0

mysql> SELECT 'a' REGEXP '^[a-d]';

-> 1

**REGEXP y RLIKE** usan el conjunto de caracteres actual (ISO-8859-1 Latin1 por defecto) al decidir el tipo de un carácter. **Atención**: Estos operadores no pueden trabajar con múltiples bytes.

* **STRCMP**(expr1,expr2)

**STRCMP**() retorna 0 si las cadenas son idénticas, -1 si el primer argumento es menor que el segundo según el orden actual, y 1 en cualquier otro caso.

mysql> SELECT STRCMP('text', 'text2');

-> -1

mysql> SELECT STRCMP('text2', 'text');

-> 1

mysql> SELECT STRCMP('text', 'text');

-> 0

**Funciones numéricas**

Este tipo de funciones trabajan únicamente con datos númericos del tipo number, y se pueden dividir en tres grupos principales.   
  
**Funciones de valores simples**   
  
Estas funciones son las más sencillas y trabajan con un único numero, variable o columna de la tabla.

|  |  |
| --- | --- |
| Función | Propósito |
| ABS(n) | Nos devuelve el valor absoluto de n |
| CEIL(n) | Nos devuelve el valor entero igual o inmediatamente superior a n |
| FLOOR(n) | Nos devuelve el valor entero igual o inmediatamente inferior a n |
| MOD(m,n) | Nos devuelve el resto de la división de m entre n |
| POWER(m, exponente) | Calcula la potencia de m elevado a exponente |
| SIGN(valor) | Nos devuelve el signo de valor |
| NVL(valor, expresión) | Función que nos sustituye valor por expresión siempre que valor sea NULL |
| ROUND(número[, m]) | Nos redondea numero a m decimales |
| SQRT(n) | Nos devuelve la raíz cuadrada de n |
| TRUNC(número[,m ]) | Trunca los números para que tengan m decimales. |

**Ejemplos:**

Select    ceil(2.3) from tabla; (esta consulta nos devolvería 3)   
  
Select mod (11,4) from tabla; ( nos devolvería 3)   
  
Select round(22.38,1) from tabla; (nos devolvería 22.4)   
  
**Funciones de grupos de valores**   
  
Este tipo de funciones se utilizan principalmente para realizar estadísticas, por lo que los valores nulos no se toman en cuenta.   
  
Dentro de este grupo se encuentran las siguientes funciones:

|  |  |
| --- | --- |
| Función | Propósito |
| AVG(n) | Nos devuelve la media de n |
| COUNT(\*|expresión) | Nos devuelve el número de veces que aparece expresión. |
| MAX(expresión) | Nos devuelve el valor máximo de expresión |
| MIN (expresión) | Nos devuelve el valor mínimo de expresión |
| VARIANCE(expresión) | Nos devuelve la varianza de expresión |
| SUM(expresión) | Nos devuelve la suma de valores de expresión. |

**Ejmplos:**

Select avg(salario) from empleado; (nos devolvería el salario medio de todos los empleados)   
  
Select count(\*)from empleado; (nos devuelve el numero de empleados que tenemos)   
  
Select min(salario) from empleado; (nos devuelve el menor salario encontrado en la tabla empleado)   
  
**Funciones de listas**   
  
Estas funciones trabajan con grupos de columnas dentro de una misma fila.   
  
Dentro de este grupo de funciones se encuentran las siguientes :

|  |  |
| --- | --- |
| Función | Propósito |
| GREATEST(valor1, valor2,…) | Obtiene el mayor valor de la lista |
| LEAST(valor1, valor2,...) | Obtiene el menor valor de la lista |

Para explicar estas funciones vamos a poner un ejemplo un poco más complicado, donde el enunciado sería el siguiente:   
Queremos mostrar por cada alumno su mayor nota de todas las que tiene.   
  
Select alumno, greatest(nota1,nota2,nota2,nota4,nota5) "nota" from alumnos;

**Funciones de fecha y hora**

Dentro de SQL tenemos un tipo de dato denominado DATE. Este campo tiene el siguiente formato predeterminado:   
  
'dd/mm/yy'   
  
Este formato lo podemos modificar con la función TO\_CHAR de la cual hablaremos más adelante en el siguiente artículo.   
  
Cabe señalar que el tipo de dato DATE también almacena la siguiente información en el siguiente orden:   
  
Siglo/Año/Mes/Día/Hora/Minutos/Segundos   
  
A continuación mostramos una tabla con las funciones para el manejo de fechas en SQL:

|  |  |
| --- | --- |
| Función | Propósito |
| SYSDATE | Devuelve la fecha del sistema |
| ADD\_MONTHS (fecha, n) | Devuelve la fecha incrementada en n meses |
| LAS\_DAY (fecha) | Devuelve la fecha del último día del mes que contiene fecha |
| MONTHS\_BETWEEN (fecha1, fecha2) | Devuelve la diferencia en meses entre la fecha1 y la fecha2 |
| NEXT\_DAY (fecha, cad) | Devuelve la fecha del primer día de la semana indicado por cad después de la fecha indicada por fecha. Cad será siempre un día de la semana escrito con letras, por ejemplo Monday. |

Ahora pasamos a poner algún ejemplo práctico para que os hagáis una idea de cómo es. De la primera función no hace falta ya que es bastante sencilla, tan solo tenéis que poner la función y os devuelve la fecha actual del sistema.   
  
**Ejemplo1:** vamos a sumar 2 meses a la fecha de alta de los usuarios   
  
Select fecha\_alta, add\_months(fecha\_alta,2) from usuarios;   
  
**Ejemplo2**: vamos a obtener el último día del mes para cada una de las fechas de alta de cada usuario   
  
Select fecha\_alta, last\_day(fecha\_alta) from usuarios;   
  
**Ejemplo3**: vamos a obtener el día que será el próximo lunes   
  
Select next\_day(sysdate, 'Monday') "siguiente dia" from tabla;

|  |  |
| --- | --- |
| Función | Propósito |
| TO\_CHAR(fecha,'formato') | Esta función nos convierte una fecha de tipo date a una fecha de tipo varchar2 |
| TO\_CHAR(numero,'formato') | Nos convierte un dato de tipo number a un tipo varchar2 |
| TO\_DATE(cad,'formato') | Convierte un tipo de dato varchar2 o char a un valor de tipo date con el formato especificado |
| TO\_NUMBER(cadena,['formato']) | Convierte una cadena a tipo de dato number, y si se le pasa, con el formato especificado |

**Ejemplo 1:**   
  
Select TO\_CHAR(fecha,'mon aaa y yyy') "fecha nueva" from emple;   
  
Esta sentencia nos devolvería una fecha con este aspecto: jun 160 1 991 Donde lo jun seria el nombre del mes, 160 el número del año, 1 el ultimo digito del año y 991 los últimos tres dígitos del año.   
  
**Ejemplo 2:**   
  
Select TO\_CHAR(TO\_DATE('01012007','ddmmyyyy'),'Month') "mes" from fechas;   
  
Esta sentencia nos devolvería algo como esto: Enero. Para llegar a este resultado lo primero que hacemos es convertir la cadena a tipo fecha y después el tipo fecha a cadena con el formato de mes.   
  
Ahora pasamos a describir 5 funciones más que nos podemos encontrar pero que se suelen utilizar poco…

|  |  |
| --- | --- |
| Función | Proposito |
| USER | Nos devuelve el nombre del usuario actual |
| VSIZE(expresión) | Nos devuelve el número de bytes que ocupa la expresión. |
| UID | Nos devuelve el identificador del usuario actual |
| DECODE(var,val1,cod1,val2,cod2…,valor-por-defecto) | Función que sustituye un valor por otro. Si var es igual a cualquier val de la lista lo sustituye por su correspondiente cod |
| DUMP(cadena,[,formato[,comienzo[,longitud]]]) | Visualiza el valor de cadena desde el comienzo hasta longitud, con el formato especificado. |

Pasamos a poner algunos ejemplos.   
**Ejemplo 1:**

Mostramos el nombre de los alumnos y si su asignatura es lengua le ponemos un código 1, si es matemáticas un código 2 y si no es ninguna de esas un 3.   
  
Select alumnos, DECODE (asignatura, 'lenguaje',1, 'matemáticas', 2, 3) "código" from alumnos;   
  
Ejemplo 2:   
  
Mostramos el usuario que está conectado.   
  
Show USER;   
  
**Ejemplo 3:**   
  
Mostramos el número de bytes que tiene la columna nombre de la tabla usuarios.   
  
Selec t VSIZE(nombre) "tamaño", nombre from usuarios;

|  |  |  |  |
| --- | --- | --- | --- |
| * [CURDATE](http://es.hscripts.com/tutoriales/mysql/fecha-hora-functiones.php#curdate) | * [CURTIME](http://es.hscripts.com/tutoriales/mysql/fecha-hora-functiones.php#curtime) | * [DATEDIFF](http://es.hscripts.com/tutoriales/mysql/fecha-hora-functiones.php#datediff) | * [DATE\_ADD](http://es.hscripts.com/tutoriales/mysql/fecha-hora-functiones.php#dateadd) |
| * [DAYNAME](http://es.hscripts.com/tutoriales/mysql/fecha-hora-functiones.php#dayname) | * [DAYOFMONTH](http://es.hscripts.com/tutoriales/mysql/fecha-hora-functiones.php#day) | * [DAYOFWEEK](http://es.hscripts.com/tutoriales/mysql/fecha-hora-functiones.php#dayweek) | * [DAYOFYEAR](http://es.hscripts.com/tutoriales/mysql/fecha-hora-functiones.php#dayyear) |
| * [HOUR](http://es.hscripts.com/tutoriales/mysql/fecha-hora-functiones.php#hour) | * [MINUTE](http://es.hscripts.com/tutoriales/mysql/fecha-hora-functiones.php#minute) | * [MONTH](http://es.hscripts.com/tutoriales/mysql/fecha-hora-functiones.php#month) | * [MONTHNAME](http://es.hscripts.com/tutoriales/mysql/fecha-hora-functiones.php#monthname) |
| * [NOW](http://es.hscripts.com/tutoriales/mysql/fecha-hora-functiones.php#now) |  |  |  |

**CURDATE ():**  
  
    Esta función devuelve la fecha actual en formato 'YYYY-MM-DD' o 'YYYYMMDD' (Año-Mes-Día)

mysql> select curdate();

--> 2007-01-03

**CURTIME ():**  
  
    Devuelve la hora actual en formato 'HH:MM:SS' o 'HHMMSS'.

mysql> select curtime();

--> 17:33:07

**DATEDIFF(expression1,expression2)** :  
  
expression1 y expression2 son expresiones fecha o fecha-y-hora. Esta función devuelve expression1 – expression2 expresados como un valor en días de una fecha a la otra. Aquí sólo las partes de fecha serán consideradas para el cálculo.

mysql> select datediff('2007-2-6 17:33:25','2007-1-1');

--> 36

**DATE\_ADD (fechahora, INTERVAL tipo expresión fechahora):**  
  
Esta función de fecha añade la expresión a la datetime suministrada.

mysql> select date\_add('2007-1-14', interval 15 day);

--> 2007-01-29

Esta función es la misma para DATE\_SUB, pero tomará lugar substrayéndola en vez de añadiéndola.   
  
**DAYNAME (fecha):**  
  
Devuelve el nombre del día para una fecha especificada.

mysql> select dayname('2007-01-04');

--> Thursday

**DAYOFMONTH (fecha) or DAY (fecha) :**  
  
Devuelve la fecha para el día del mes en el rango de 1 a 31. DAY () es un sinónimo para DAYOFMONTH ().

mysql> select dayofmonth('2007-01-04');

--> 4

**DAYOFWEEK (fecha):**  
  
Devuelve el día de la semana en formato numérico como 1 para Domingo hasta 7 para Sábado.

mysql> select dayofweek('2007-01-04');

--> 5

**DAYOFYEAR (fecha):**  
  
Devuelve el día del año para la fecha dada en formato numérico, en rago de 1 a 366.

mysql> select dayofyear('2007-07-09');

--> 190

**HOUR (hora):**  
  
Devuelve la hora del tiempo especificado en formato numérico de 0 a 23.

mysql> select hour('14:46:12');

--> 14

**MINUTE (hora):**  
  
Devuelve el minuto del tiempo especificado en formato numérico de 0 a 59.

mysql> select minute('14:46:12');

--> 46

**MONTH (fecha):**  
  
Devuelve el mes para la fecha dada en formato numérico en rango de 0 a 12.

mysql> select month('2007-07-09');

--> 7

**MONTHNAME (fecha):**  
  
Devuelve el nombre del mes para la fecha especificada.

mysql> select monthname('2007-07-09');

--> July

**NOW ():**  
  
Esta función de fecha devuelve la fecha y hora actual en formato 'YYYY-MM-DD HH:MM:SS' o YYYYMMDDHHMMSS (Año-Mes-Día Hora: Minuto: Segundo)

mysql> select now();

--> 2007-01-04 14:56:15

#### Funciones de Agregado

Las funciones de agregado se usan dentro de una cláusula **SELECT**  
en grupos de registros para devolver un único valor que se aplica a un  
grupo de registros.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Comando | Descripción | | **AVG** | Utilizada para calcular el promedio de los valores de un campo determinado | | **COUNT** | Utilizada para devolver el número de registros de la selección | | **SUM** | Utilizada para devolver la suma de todos los valores de un campo determinado | | **MAX** | Utilizada para devolver el valor más alto de un campo especificado | | **MIN** | Utilizada para devolver el valor más bajo de un campo especificado | |

## CREACION DE VISTAS

Las vistas pueden crearse a partir de varios tipos de sentencias SELECT. Pueden hacer referencia a tablas o a otras vistas. Pueden usar combinaciones, UNION, y subconsultas. El SELECT inclusive no necesita hacer referencia a otras tablas. En el siguiente ejemplo se define una vista que selecciona dos columnas de otra tabla, así como una expresión calculada a partir de ellas:

mysql> CREATE TABLE t (qty INT, price INT);

mysql> INSERT INTO t VALUES(3, 50);

mysql> CREATE VIEW v AS SELECT qty, price, qty\*price AS value FROM t;

mysql> SELECT \* FROM v;

+------+-------+-------+

| qty | price | value |

+------+-------+-------+

| 3 | 50 | 150 |

+------+-------+-------+