MySQL

1. Insertar Registros en la BD
2. Seleccionarlos
3. Consultas, Querys
4. Actualizar Registros
5. Eliminar Registros
6. Consultas Avanzadas

Instrucción INSERT

INSERT inserta nuevas filas en una tabla existente.

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    [(col_name [, col_name] ...)]
    { {VALUES | VALUE} (value_list) [, (value_list)] ...
      |
      VALUES row_constructor_list

    }
```

tbl_name es la tabla en la que se deben insertar las filas. Especifique las columnas para las que la declaración proporciona valores de la siguiente manera:

- Proporcione una lista entre paréntesis de nombres de columnas separados por comas después del nombre de la tabla. En este caso, la lista VALUES, debe proporcionar un valor para cada columna nombrada . Para el INSERT, el número de columnas en la tabla de origen debe coincidir con el número de columnas que se insertarán.
- Si no especifica una lista de nombres de columna para INSERT ... VALUES, los valores para cada columna de la tabla deben ser proporcionados por la lista de datos en el orden de las columnas de la tabla.
- Una cláusula SET indica las columnas explícitamente por nombre, junto con el valor a asignar a cada una.

Los valores de las columnas se pueden dar de varias formas:

- Si el modo SQL estricto no está habilitado, cualquier columna a la que no se le haya dado un valor explícitamente se establece en su valor predeterminado (explícito o implícito). Por ejemplo, si

especifica una lista de columnas que no nombra todas las columnas de la tabla, las columnas sin nombre se establecen en sus valores predeterminados.
Si el modo SQL estricto está habilitado, una declaración INSERT genera un error si no especifica un valor explícito para cada columna que no tiene un valor predeterminado.

- Si tanto la lista de columnas como la lista VALUES están vacías, INSERT crea una fila con cada columna establecida en su valor predeterminado:

```
INSERT INTO tbl_name () VALUES();
```

- Utilice la palabra clave DEFAULT para establecer una columna explícitamente en su valor predeterminado. Esto hace que sea más fácil escribir declaraciones INSERT que asignan valores a todas las columnas excepto a unas pocas, porque te permite evitar escribir una lista de VALUES incompleta que no incluye un valor para cada columna de la tabla. De lo contrario, debes proporcionar la lista de nombres de columna correspondientes a cada valor de la lista VALUES .

- La conversión de tipo de una expresión ***expr*** que proporciona un valor de columna puede ocurrir si el tipo de datos de la expresión no coincide con el tipo de datos de la columna. La conversión de un valor dado puede resultar en diferentes valores insertados dependiendo del tipo de columna. Por ejemplo, la inserción de la cadena '1999.0e-2' en una columna INT, FLOAT, DECIMAL(10,6), o YEAR resultaría en inserciones de la columna de valor 1999, 19.9921, 19.992100, o 1999, respectivamente. El valor almacenado en las columnas INT y YEAR es 1999 porque la conversión de cadena a número solo considera la parte inicial de la cadena como un número entero o año válido. Para las columnas FLOAT y DECIMAL, la conversión de cadena a número considera la cadena completa como un valor numérico válido.

- Una expresión ***expr*** puede hacer referencia a cualquier columna que se estableció anteriormente en una lista de valores. Por ejemplo, puede hacer esto porque el valor de col2 hace referencia a col1, que se ha asignado previamente:

```
INSERT INTO tbl_name (col1,col2) VALUES(15,col1*2);
```

Pero lo siguiente no es legal, porque el valor de col1 se refiere a col2, que se asigna después col1:

```
INSERT INTO tbl_name (col1,col2) VALUES(col2*2,15);
```

Se produce una excepción para las columnas que contienen valores AUTO_INCREMENT. Debido a que los AUTO_INCREMENT valores se generan después de otras asignaciones de valor, cualquier referencia a una columna AUTO_INCREMENT en la asignación devuelve un 0.

Insertar varios renglones

Las declaraciones que usan sintaxis VALUES pueden insertar varias filas. Para hacer esto, incluya varias listas de valores de columna separados por comas, con listas encerradas entre paréntesis y separadas por comas. Ejemplo:

```
INSERT INTO tbl_name (a,b,c)
  VALUES(1,2,3), (4,5,6), (7,8,9);
```

Cada lista de valores debe contener exactamente tantos valores como se vayan a insertar por fila. La siguiente declaración no es válida porque contiene una lista de nueve valores, en lugar de tres listas de tres valores cada una:

```
INSERT INTO tbl_name (a,b,c) VALUES(1,2,3,4,5,6,7,8,9);
```

VALUE es un sinónimo de VALUES en este contexto. Ni implica nada sobre el número de listas de valores, ni sobre el número de valores por lista. Se puede utilizar cualquiera de las dos, ya sea que haya una lista de valores única o varias listas, e independientemente del número de valores por lista. Las declaraciones que utilizan sintaxis VALUES ROW() también pueden insertar varias filas. En este caso, cada lista de valores debe estar contenida dentro de un ROW()(constructor de filas), como este:

```
INSERT INTO tbl_name (a,b,c)
  VALUES ROW(1,2,3), ROW(4,5,6), ROW(7,8,9);
```

- Establecer una columna numérica en un valor que se encuentra fuera del rango de la columna. El valor se recorta al punto final más cercano del rango.

- Asignar un valor como, por ejemplo, '10.34 a'a una columna numérica. El texto no numérico final se elimina y se inserta la parte numérica restante. Si el valor de la cadena no tiene una parte numérica inicial, la columna se establece en 0.

Sentencia SELECT

SELECTse utiliza para recuperar filas seleccionadas de una o más tablas y puede incluir declaraciones UNION y subconsultas. Una declaración SELECT puede comenzar con una cláusula WITH para definir expresiones de tabla comunes accesibles dentro de SELECT.

Las cláusulas de SELECTdeclaraciones más comúnmente utilizadas son las siguientes:

- Cada ***select_expr*** indica una columna que deseas recuperar. Debe haber al menos una ***select_expr***.
- ***table_references*** indica la tabla o tablas de las cuales vas a recuperar filas.
- La cláusula WHERE, si se proporciona, indica la condición o condiciones que las filas deben cumplir para ser seleccionadas. ***where_condition*** es una expresión que se evalúa como verdadera para cada fila a seleccionar. La declaración selecciona todas las filas si no hay ninguna cláusula WHERE.

En la expresión WHERE, podés usar cualquiera de las funciones y operadores que admite MySQL, excepto las funciones agregadas de grupo o agrupación de datos.

SELECT también se puede utilizar para recuperar filas calculadas sin referencia a ninguna tabla.

Por ejemplo:

```
mysql> SELECT 1 + 1;
    -> 2
```

Se le permite especificar DUAL como nombre de tabla ficticia en situaciones en las que no se hace referencia a tablas:

```
mysql> SELECT 1 + 1 FROM DUAL;
    -> 2
```

DUAL es puramente para la conveniencia de las personas que requieren que todas las declaraciones SELECT deben tener FROM y posiblemente otras cláusulas. MySQL no requiere FROM DUAL si no se hace referencia a tablas.

En general, las cláusulas utilizadas deben darse exactamente en el orden que se muestra en la descripción dela sintaxis. Por ejemplo, una cláusula HAVING debe ir después de cualquier cláusula GROUP BY y antes de cualquier cláusula ORDER BY. La cláusula INTO, si está presente,

puede aparecer en cualquier posición indicada por la descripción de la sintaxis, pero dentro de una declaración determinada solo puede aparecer una vez, no en varias posiciones.

La lista de términos ***select_expr*** comprende la lista de selección que indica qué columnas recuperar. Los términos especifican una columna o expresión o podés usar * o -shorthand:

- Se puede utilizar una lista de selección que consta solo de un solo no calificado * como forma abreviada para seleccionar todas las columnas de todas las tablas:

```
SELECT * FROM tabla1 INNER JOIN tabla2 ...
```

- ***tbl_name***.* se puede usar como una abreviatura calificada para seleccionar todas las columnas de la tabla nombrada:

```
SELECT t1.*, t2.* FROM t1 INNER JOIN t2 ...
```

- El uso de un elemento no calificado * con otros elementos de la lista de selección puede producir un error de análisis. Para evitar este problema, utilice una ***tbl_name***.* referencia calificada

```
SELECT AVG(score), t1.* FROM t1 ...
```

La siguiente lista proporciona información adicional sobre otras cláusulas SELECT:

- A la ***select_expr*** se le puede dar un alias usando. El alias se utiliza como nombre de la columna de la expresión y se puede utilizar en funciones o cláusulas.

Por ejemplo : AS ***alias_name*** GROUP BY ORDER BY HAVING

```
SELECT CONCAT(last_name,', ',first_name) AS full_name
  FROM mytable ORDER BY full_name;
```

La palabra clave AS es opcional cuando se asigna un alias a un identificador ***select_expr***. El ejemplo anterior podría haberse escrito así:

```
SELECT CONCAT(last_name,', ',first_name) full_name
  FROM mytable ORDER BY full_name;
```

Sin embargo, debido a que AS es opcional, puede ocurrir un problema sutíl si olvidás la coma entre dos expresiones ***select_expr*** ya que MySQL interpreta la segunda como un nombre de alias. Por ejemplo, en la siguiente declaración, columnb es interpretado por MySql como un nombre de alias:

```
SELECT columna columnb FROM mytable;
```

Por esta razón, es una buena práctica tener el hábito de usar AS explícitamente al especificar alias de columna.

No está permitido hacer referencia a un alias de columna en una cláusula WHERE, porque es posible que el valor de la columna aún no se haya determinado cuando se ejecuta la cláusula WHERE .

- La cláusula indica la tabla o tablas de las que recuperar filas. Si nombrás más de una tabla, estás realizando una combinación. Para cada tabla especificada, opcionalmente podés especificar un alias. FROM ***table_references***

```
tbl_name [[AS] alias] [index_hint]
```

El uso de sugerencias de índice proporciona al optimizador información sobre cómo elegir índices durante el procesamiento de consultas.

- Puede hacer referencia a una tabla dentro de la base de datos predeterminada como ***tbl_name*** o como ***db_name***. ***tbl_name*** para especificar una base de datos explícitamente. Se puede hacer referencia a una columna como ***col_name***, ***tbl_name***. ***col_name***, o ***db_name***. ***tbl_name***. ***col_name***.
No es necesario especificar un prefijo ***tbl_name*** o ***db_name***. ***tbl_name*** para una referencia de columna a menos que la referencia sea ambigua.

- Una referencia de tabla puede tener un alias usando AS . Estas declaraciones son equivalentes: ***tbl_name*** AS ***alias_name*** ***tbl_name alias_name***

```
SELECT t1.name, t2.salary FROM employee AS t1, info AS t2
  WHERE t1.name = t2.name;
SELECT t1.name, t2.salary FROM employee t1, info t2
  WHERE t1.name = t2.name;
```

- Se puede hacer referencia a las columnas seleccionadas para la salida en cláusulas ORDER BY y GROUP BY utilizando nombres de columna, alias de columna o posiciones de columna. Las posiciones de las columnas son números enteros y comienzan con 1:

```
SELECT college, region, seed FROM tournament
  ORDER BY region, seed;
SELECT college, region AS r, seed AS s FROM tournament
  ORDER BY r, s;
SELECT college, region, seed FROM tournament
  ORDER BY 2, 3;
```

Para ordenar en orden inverso, agregá la DESC palabra clave (descendente) al nombre de la columna en la ORDER BY cláusula por la que está ordenando. El valor predeterminado es el orden ascendente; esto se puede especificar explícitamente usando la palabra clave ASC.

Si ORDER BY ocurre dentro de una expresión de consulta entre paréntesis y también se aplica en la consulta externa, los resultados no están definidos y pueden cambiar en una versión futura de MySQL. El uso de posiciones de columna está en desuso porque la sintaxis se ha eliminado del estándar SQL.

- Antes de MySQL 8.0.13, MySQL admitía una extensión de sintaxis no estándar que permitía designadores explícitos ASC o DESC para palabra clave GROUP BY columnas. MySQL 8.0.12 y versiones posteriores son compatibles ORDER BY con funciones de agrupación, por lo que el uso de esta extensión ya no es necesario. Esto también significa que puede ordenar en una columna o columnas arbitrarias al usar GROUP BY, así:

```
SELECT a, b, COUNT(c) AS t FROM test_table GROUP BY a,b ORDER BY a,t DESC;
```

- MySQL extiende el uso de GROUP BY para permitir la selección de campos que no se mencionan en la cláusula GROUP BY.

- La HAVINGcláusula se aplica casi al final, justo antes de que los elementos se envíen al cliente, sin optimización. (LIMITse aplica después HAVING.)

El estándar SQL requiere que se HAVINGhaga referencia solo a columnas en la GROUP BY cláusula o columnas utilizadas en funciones agregadas. Sin embargo, MySQL admite una extensión para este comportamiento y permite HAVINGhacer referencia a columnas en la SELECTlista y columnas en subconsultas externas también.

Si la HAVINGcláusula se refiere a una columna que es ambigua, aparece una advertencia. En la siguiente declaración, col2es ambiguo porque se usa como alias y como nombre de columna:

```
SELECT COUNT(col1) AS col2 FROM t GROUP BY col2 HAVING col2 = 2;
```

Se da preferencia al comportamiento estándar de SQL, por lo que si HAVINGse usa un nombre de columna tanto en GROUP BYcomo como una columna con alias en la lista de columnas de salida, se da preferencia a la columna de la GROUP BYcolumna.

- No lo utilice HAVINGpara elementos que deberían estar en la WHEREcláusula. Por ejemplo, no escriba lo siguiente:

```
SELECT col_name FROM tbl_name HAVING col_name > 0;
```

Escribe esto en su lugar:

```
SELECT col_name FROM tbl_name WHERE col_name > 0;
```

- La cláusula HAVING puede hacer referencia a funciones agregadas, que la cláusula WHERE no puede procesar:

```
SELECT user, MAX(salary) FROM users
  GROUP BY user HAVING MAX(salary) > 10;
```

(Esto no funcionaba en algunas versiones anteriores de MySQL).

- MySQL permite nombres de columna duplicados. Es decir, puede haber más de uno ***select_expr*** con el mismo nombre. Esta es una extensión de SQL estándar. Debido a que MySQL también permite GROUP BY y HAVING hacer referencia a valores ***select_expr***, esto puede resultar en una ambigüedad:

```
SELECT 12 AS a, a FROM t GROUP BY a;
```

En esa declaración, ambas columnas tienen el nombre a. Para asegurarse de que se utilice la columna correcta para la agrupación, utilice nombres diferentes para cada una ***select_expr***.

- MySQL resuelve referencias de columna o alias no calificadas en ORDER BYcláusulas buscando en los valores ***select_expr***, luego en las columnas de las tablas en la cláusula FROM. Para las cláusulas GROUP BY o HAVING, busca la cláusula FROM antes de buscar en los valores ***select_expr*** . (Para GROUP BY y HAVING, esto difiere del comportamiento anterior a MySQL 5.0 que usaba las mismas reglas que para ORDER BY).

- La cláusula LIMIT se puede utilizar para restringir el número de filas devueltas por la declaración SELECT. LIMIT toma uno o dos argumentos numéricos, que deben ser constantes enteras no negativas, con estas excepciones:

- Dentro de las declaraciones preparadas, los parámetros LIMIT se pueden especificar mediante ? marcadores de posición.

- Dentro de los programas almacenados, parámetros los LIMIT se pueden especificar utilizando parámetros de rutina con valores enteros o variables locales.

Con dos argumentos, el primer argumento especifica el desplazamiento de la primera fila para devolver y el segundo especifica el número máximo de filas para devolver. El desplazamiento de la fila inicial es 0 (no 1):

```
SELECT * FROM tbl LIMIT 5,10;  # Devuelve los renglones 6 a 15
```

Para recuperar todas las filas desde un cierto desplazamiento hasta el final del conjunto de resultados, puede usar un número grande para el segundo parámetro. Esta declaración recupera todas las filas desde la fila 96 hasta la última:

```
SELECT * FROM tbl LIMIT 95,18446744073709551615;
```

Con un argumento, el valor especifica el número de filas que se devolverán desde el principio del conjunto de resultados:

```
SELECT * FROM tbl LIMIT 5;     # Devuelve los primeros 5 renglones
```

En otras palabras, es equivalente a . LIMIT ***row_count*** LIMIT 0, ***row_count***

Para declaraciones preparadas, puede utilizar marcadores de posición. Las siguientes declaraciones devuelven una fila de la tbltabla:

```
SET @a=1;
PREPARE STMT FROM 'SELECT * FROM tbl LIMIT ?';
EXECUTE STMT USING @a;
```

Las siguientes declaraciones devuelven la segunda a la sexta filas de la tabla tbl:

```
SET @skip=1; SET @numrows=5;
PREPARE STMT FROM 'SELECT * FROM tbl LIMIT ?, ?';
EXECUTE STMT USING @skip, @numrows;
```

Para compatibilidad con PostgreSQL, MySQL también admite la sintaxis. LIMIT ***row_count*** OFFSET ***offset***

Si existe un LIMIT dentro de una expresión de consulta entre paréntesis y también se aplica en la consulta externa, los resultados no están definidos y pueden cambiar en una versión futura de MySQL.

- Los modificadores ALLy DISTINCT especifican si se deben devolver filas duplicadas. ALL (el valor predeterminado) especifica que se deben devolver todas las filas coincidentes, incluidos los duplicados. DISTINCT especifica la eliminación de filas duplicadas del conjunto de resultados. Es un error especificar ambos modificadores. DISTINCT ROW es sinónimo de DISTINCT.

Declaración UPDATE

UPDATE es una declaración que modifica filas en una tabla.

Una declaración UPDATE puede comenzar con una cláusula WITH para definir expresiones de tabla comunes accesibles dentro de UPDATE.

Sintaxis de tabla única:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_reference
    SET assignment_list
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]


value:
    {expr | DEFAULT}
```

Para la sintaxis de tabla única, la declaración UPDATE actualiza columnas de filas existentes en la tabla nombrada con nuevos valores. La cláusula SET indica qué columnas modificar y los valores que se les deben dar. Cada valor se puede dar como una expresión o la palabra clave DEFAULT para establecer una columna explícitamente en su valor predeterminado. La cláusula WHERE, si se proporciona, especifica las condiciones que identifican qué filas actualizar. Sin cláusula WHERE, se actualizan todas las filas. Si ORDER BY se especifica la cláusula, las filas se actualizan en el orden especificado. La cláusula LIMIT impone un límite al número de filas que se pueden actualizar.

Para la sintaxis de varias tablas, UPDATE actualiza las filas de cada tabla nombrada y que satisfacen las condiciones. Cada fila coincidente se actualiza una vez, incluso si coincide con las condiciones varias veces.

Solo necesita el privilegio UPDATE para las columnas a las que se hace referencia en un UPDATE archivo que están realmente actualizadas. Solo necesita el privilegio SELECT para las columnas que se leen pero no se modifican.

La declaración UPDATE admite los siguientes modificadores:

- Con el IGNOREmodificador, la declaración de actualización no se cancela incluso si ocurren errores durante la actualización. Las filas para las que se producen conflictos de claves duplicadas en un valor de clave único no se actualizan. En su lugar, las filas actualizadas a valores que causarían errores de conversión de datos se actualizan a los valores válidos más cercanos.

Las declaraciones, incluidas las que tienen una ORDER BY cláusula, se marcan como inseguras para la replicación basada en declaraciones. (Esto se debe a que el orden en el que se actualizan las filas determina qué filas se ignoran).

Si accede a una columna de la tabla para actualizarla en una expresión, UPDATE usa el valor actual de la columna. Por ejemplo, la siguiente declaración se establece col1 en uno más que su valor actual:

```
UPDATE t1 SET col1 = col1 + 1;
```

La segunda asignación en la siguiente declaración establece col2 el valor actual (actualizado) col1, no el col1 valor original . El resultado es eso col1 y col2 tienen el mismo valor. Este comportamiento difiere del SQL estándar.

```
UPDATE t1 SET col1 = col1 + 1, col2 = col1;
```

Las asignaciones UPDATE de una sola tabla generalmente se evalúan de izquierda a derecha. Para las actualizaciones de varias tablas, no hay garantía de que las asignaciones se lleven a cabo en un orden en particular.

Si establece una columna en el valor que tiene actualmente, MySQL se da cuenta de esto y no la actualiza.

Si actualiza una columna que se ha declarado NOT NULL estableciendo en NULL, se produce un error si se habilita el modo SQL estricto; de lo contrario, la columna se establece en el valor predeterminado implícito para el tipo de datos de la columna y se incrementa el recuento de advertencias. El valor predeterminado implícito es 0para tipos numéricos, la cadena vacía ('') para tipos de cadena y el valor " cero " para tipos de fecha y hora.

Por ejemplo, si la tabla contiene 1 y 2 en la columna id y 1 se actualiza a 2 antes de que 2 se actualizá a 3, se produce un error. Para evitar este problema, agregue una cláusula ORDER BY para que las filas con valores id más grandes se actualicen antes que aquellas con valores más pequeños:

```
UPDATE t SET id = id + 1 ORDER BY id DESC;
```

También podés realizar operaciones UPDATE que abarquen varias tablas. Sin embargo, no puede usar ORDER BY o LIMIT con una tabla múltiple UPDATE. La cláusula ***table_references*** enumera las tablas involucradas en la combinación. Aquí hay un ejemplo:

```
UPDATE items,month SET items.price=month.price
WHERE items.id=month.id;
```

El ejemplo anterior muestra una combinación interna que usa el operador de coma, pero las declaraciones UPDATE de varias tablas pueden usar cualquier tipo de combinación permitida en declaraciones SELECT, como LEFT JOIN.

Declaración DELETE

DELETE es una declaración que elimina filas de una tabla.

Una declaración DELETE puede comenzar con una cláusula WITH para definir expresiones de tabla comunes accesibles dentro de DELETE.

Sintaxis de tabla única

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name [[AS] tbl_alias]
    [PARTITION (partition_name [, partition_name] ...)]
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]
```

La declaración DELETE elimina filas ***tbl_name*** y devuelve el número de filas eliminadas. Para verificar el número de filas eliminadas, llamá a la función ROW_COUNT().

Cláusulas principales

Las condiciones de la cláusula opcional WHERE identifican qué filas eliminar. Sin cláusula WHERE , se eliminan todas las filas.

where_condition es una expresión que se evalúa como verdadera para cada fila que se va a eliminar.

Si se especifica la cláusula ORDER BY, las filas se eliminan en el orden especificado. La cláusula LIMIT impone un límite al número de filas que se pueden eliminar. Estas cláusulas se aplican a eliminaciones de una sola tabla, pero no a eliminaciones de varias tablas.

Sintaxis de varias tablas

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    tbl_name[.*] [, tbl_name[.*]] ...
    FROM table_references
    [WHERE where_condition]
```

Privilegios

Necesita el privilegio DELETE de una tabla para eliminar filas de ella. Solo necesita el privilegio SELECT para las columnas que solo se leen, como las que se mencionan en la cláusula WHERE. Cuando no necesita saber el número de filas eliminadas, la declaración TRUNCATE TABLE es una forma más rápida de vaciar una tabla que una declaración DELETE sin cláusula WHERE. A diferencia de DELETE, TRUNCATE TABLE no se puede utilizar dentro de una transacción o si tiene un bloqueo de la tabla.

Para asegurarte de que una declaración DELETE dada no tome demasiado tiempo, la cláusula específica de MySQL para especifica el número máximo de filas que se eliminarán, la candidadde número de filas para eliminar no debe ser mayor que el límite, repetí la declaración hasta que el número de filas afectadas sea menor que el valor del total de cantidad de filas.

Columnas de incremento automático

Si eliminás la fila que contiene el valor máximo de una columna AUTO_INCREMENT, el valor no se reutiliza para una tabla MyISAMo InnoDB. Si elimina todas las filas de la tabla con (sin una cláusula where) , la secuencia comienza de nuevo para todos los motores de almacenamiento.

DELETE FROM ***tbl_name***WHERE autocommit InnoDB MyISAM InnoDB

Para las tablas MyISAM, puede especificar una columna AUTO_INCREMENT secundaria en una clave de varias columnas. En este caso, la reutilización de los valores eliminados de la parte superior de la secuencia se produce incluso para las tablas MyISAM .

Modificadores

La declaración DELETE admite los siguientes modificadores:

- Si especifica el LOW_PRIORITY modificador, el servidor retrasa la ejecución del DELETE hasta que ningún otro cliente esté leyendo de la tabla. Esto afecta solamente a los motores de almacenamiento que utilizan solamente de bloqueo de nivel de tabla (tales como MyISAM, MEMORY, y MERGE).

- Para las tablas MyISAM, si usa el modificador QUICK, el motor de almacenamiento no fusiona hojas de índice durante la eliminación, lo que puede acelerar algunos tipos de operaciones de eliminación.
- El modificador IGNORE hace que MySQL ignore errores ignorables durante el proceso de eliminación de filas. (Los errores encontrados durante la etapa de análisis se procesan de la manera habitual). Los errores que se ignoran debido al uso de IGNORE se devuelven como advertencias.

Orden de eliminación

Si la declaración DELETE incluye una cláusula ORDER BY, las filas se eliminan en el orden especificado por la cláusula. Esto es útil principalmente junto con LIMIT. Por ejemplo, la siguiente declaración busca filas que coincidan con la cláusula WHERE, las ordena timestamp_column y elimina la primera (la más antigua):

```
DELETE FROM somelog WHERE user = 'jcole'
ORDER BY timestamp_column LIMIT 1;
```

ORDER BY también ayuda a eliminar filas en el orden requerido para evitar violaciones de la integridad referencial.

Tablas InnoDB

Si estás eliminando muchas filas de una tabla grande, puede exceder el tamaño de la tabla de bloqueo para una tabla InnoDB. Para evitar este problema, o simplemente para minimizar el tiempo que la tabla permanece bloqueada, la siguiente estrategia (que no se usa DELETE en absoluto) puede ser útil:

1. Seleccione las filas que *no* se eliminarán en una tabla vacía que tenga la misma estructura que la tabla original:

```
INSERT INTO t_copy SELECT * FROM t WHERE ... ;
```

2. Utilice RENAME TABLE para mover atómicamente la tabla original fuera del camino y cambiar el nombre de la copia al nombre original:

```
RENAME TABLE t TO t_old, t_copy TO t;
```

3. Desbloqueá la tabla original:

```
DROP TABLE t_old;
```

Ninguna otra sesión puede acceder a las tablas involucradas mientras se ejecuta RENAME TABLE, por lo que la operación de cambio de nombre no está sujeta a problemas de concurrencia.

Tablas MyISAM

En las tablas MyISAM, las filas eliminadas se mantienen en una lista vinculada y las operaciones INSERT posteriores reutilizan posiciones de filas antiguas. Para recuperar el espacio no utilizado y reducir el tamaño de los archivos, use la instrucción OPTIMIZE TABLE o la utilidad myisamchk para reorganizar las tablas. OPTIMIZE TABLE es más fácil de usar, pero myisamchk es más rápido.