Manejo de Concurrencia en Mysql

Contenido

Manejo de Concurren en Mysql

Modos de bloqueo InnoDB

InnoDB implementa un bloqueo a nivel de fila estándar, donde hay dos tipos de bloqueos:

- Compartido (Shared) (*S*) le permite a una transacción leer una fila.
- Exclusivo (Exclusive) (*X*) le permite a una transacción actualizar o eliminar una fila.

Si una transacción *A* sostiene un bloqueo exclusivo (*X*) sobre una tupla *t*, entonces una solicitud de otra transacción *B* para establecer un bloqueo de cualquier tipo sobre *t* no puede ser atendida inmediatamente. En lugar de eso, la transacción *B* debe esperar a que la transacción *A* libere el bloqueo en la tupla *t*.
Si la transacción *A* sostiene un bloqueo compartido (*S*) sobre una tupla *t*, entonces

- Una solicitud de otra transacción *B* para un bloqueo *X* sobre *t* no puede ser atendido inmediatamente.
- Una solicitud de otra transacción *B* para un bloqueo *S* sobre *t* puede ser atendido inmediatamente. En consecuencia, tanto *A* como *B* sostendrán un bloqueo *S* sobre *t*.

Adicionalmente, InnoDB soporta *bloqueo de granularidad múltiple* (multiple granularity locking), el cual permite que existan simultáneamente bloqueos en registros y bloqueos en tablas enteras. Para hacer práctico el nivel de bloqueo de granularidad múltiple, se emplean tipos adicionales de bloqueo, llamados *bloqueos de intención* (intention locks). Los bloqueos de intención son bloqueos de tabla en InnoDB. La idea detrás de los mismos es que una transacción indique qué tipo de bloqueo (compartido o exclusivo) requerirá más tarde sobre una fila de esa tabla. En InnoDB se utilizan dos tipos de bloqueos de intención (asumiendo que la transacción *T* ha solicitado un bloqueo del tipo indicado en la tabla *R*):

- Intención compartida (Intention shared) (*IS*): La transacción *T* trata de establecer bloqueos *S* en tuplas individuales de la tabla *T*.
- Intención exclusiva (Intention exclusive) (*IX*): La transacción *T* trata de establecer bloqueos *X* en las tuplas.

Luego, el protocolo de bloqueo de intención es el siguiente:

- Antes de que de una determinada transacción logre un bloqueo *S* en una determinada fila, primero debe conseguir establecer un bloqueo *IS* o superior en la tabla que contiene a la fila.
- Antes de que de una determinada transacción logre un bloqueo *X* en una determinada fila, primero debe conseguir establecer un bloqueo *IX* en la tabla que contiene a la fila.

Estas reglas pueden resumirse convenientemente por medio de una *matriz de compatibilidad entre tipos de bloqueo*:

	X	**IX**	**S**	**IS**	-
X	N	N	N	N	S

IX	N	S	N	S	S
S	N	S	S	S	S
IS	N	S	S	S	S
-	S	S	S	S	S

Por lo tanto, los bloqueos de intención solamente bloquean solicitudes sobre tablas completas (Ej: `LOCK TABLES ... WRITE`). El propósito principal de *`IX`* y *`IS`* es mostrar que alguien está bloqueando una fila, o va a bloquear una fila en la tabla.

`InnoDB` y `AUTOCOMMIT`

En `InnoDB`, toda la actividad del usuario se produce dentro de una transacción. Si el modo de ejecución automática (autocommit) está activado, cada sentencia SQL conforma una transacción individual por sí misma. MySQL siempre comienza una nueva conexión con la ejecución automática habilitada.
Si el modo de ejecución automática se deshabilitó con `SET AUTOCOMMIT = 0`, entonces puede considerarse que un usuario siempre tiene una transacción abierta. Una sentencia SQL `COMMIT` o `ROLLBACK` termina la transacción vigente y comienza una nueva. Ambas sentencias liberan todos los bloqueos `InnoDB` que se establecieron durante la transacción vigente. Un `COMMIT` significa que los cambios hechos en la transacción actual se convierten en permanentes y se vuelven visibles para los otros usuarios. Por otra parte, una sentencia `ROLLBACK`, cancela todas las modificaciones producidas en la transacción actual.
Si la conexión tiene la ejecución automática habilitada, el usuario puede igualmente llevar a cabo una transacción con varias sentencias si la comienza explícitamente con `START TRANSACTION` o `BEGIN` y la termina con `COMMIT` o `ROLLBACK`.

InnoDB y TRANSACTION ISOLATION LEVEL

En los términos de los niveles de aislamiento de transacciones SQL: 1992, el nivel predeterminado en `InnoDB` es `REPEATABLE READ`. En MySQL 5.0, `InnoDB` ofrece los cuatro niveles de aislamiento de transacciones descriptos por el estándar SQL. Se puede establecer el nivel predeterminado de aislamiento por todas las conexiones mediante el uso de la opción `--transaction-isolation` en la línea de comandos o en ficheros de opciones. Por ejemplo, se puede establecer la opción en la sección `[mysqld]` de `my.cnf` de este modo:

```
[mysqld]
transaction-isolation = {READ-UNCOMMITTED | READ-COMMITTED
                         | REPEATABLE-READ | SERIALIZABLE}
```

Un usuario puede cambiar el nivel de aislamiento de una sesión individual o de todas las nuevas conexiones con la sentencia `SET TRANSACTION`. Su sintaxis es la siguiente:

```
SET [SESSION | GLOBAL] TRANSACTION ISOLATION LEVEL
                       {READ UNCOMMITTED | READ COMMITTED
                        | REPEATABLE READ | SERIALIZABLE}
```

Nótese que se usan guiones en los nombres de niveles de la opción `--transaction-isolation`, pero no en la sentencia `SET TRANSACTION`.
El comportamiento predeterminado es establecer el nivel de aislamiento a partir de la próxima transacción que se inicie. Si se emplea la palabra clave `GLOBAL`, la sentencia establece el nivel predeterminado de la transacción globalmente para todas las nuevas conexiones creadas a partir de ese punto (pero no en las existentes). Se necesita el privilegio `SUPER` para hacer esto. Utilizando la palabra clave `SESSION` se establece el nivel de transacción para todas las futuras transacciones ejecutadas en la actual conexión.
Cualquier cliente es libre de cambiar el nivel de aislamiento de la sesión (incluso en medio de una transacción), o el nivel de aislamiento para la próxima transacción.
Los niveles de aislamiento de transacciones globales y de sesión pueden consultarse con estas sentencias:

```
SELECT @@global.tx_isolation;
SELECT @@tx_isolation;
```

En el bloqueo a nivel de fila, `InnoDB` emplea bloqueo de clave siguiente (next-key). Esto significa que, además de registros de índice, `InnoDB` también puede bloquear el "vacío" que precede a un registro de índice para bloquear inserciones de otros usuarios inmediatamente antes del registro de índice. Un bloqueo de clave siguiente hace referencia a bloquear un registro de índice y la posición vacía antes de él. Bloquear una posición vacía es establecer un bloqueo que actúa solamente sobre el vacío anterior a un registro de índice.
A continuación una descripción detallada de cada nivel de aislamiento en `InnoDB`:

- `READ UNCOMMITTED`
  Las sentencias `SELECT` son ejecutadas sin realizar bloqueos, pero podría usarse una versión anterior de un registro. Por lo tanto, las lecturas no son consistentes al usar este nivel de aislamiento. Esto también se denomina "lectura sucia" (dirty read). En otro caso, este nivel de aislamiento funciona igual que `READ COMMITTED`.
- `READ COMMITTED`
  Similar en parte al mismo nivel de aislamiento de Oracle. Todas las sentencias `SELECT ... FOR UPDATE` y `SELECT ... LOCK IN SHARE MODE` bloquean solamente los registros de índice, no los espacios vacíos que los preceden, por lo tanto se permite la libre inserción de nuevos registros junto a los bloqueados. Las sentencias `UPDATE` and `DELETE` que empleen un índice único con una condición de búsqueda única bloquean solamente el registro de índice hallado, no el espacio que lo precede. En las sentencias `UPDATE` y `DELETE` que actúan sobre rangos de registros, `InnoDB` debe bloquear los espacios vacíos y bloquear las inserciones de otros usuarios en los espacios vacíos que hay dentro del rango. Esto es necesario debido a que las "filas fantasma" deben ser bloqueadas para que funcionen la replicación y recuperación en MySQL.
  Las lecturas consistentes se comportan como en Oracle: Cada lectura consistente, incluso dentro de la misma transacción, establece y lee su propia captura tomada de la base de datos. Consulte Sección 15.10.4, "Lecturas consistentes que no bloquean".
- `REPEATABLE READ`
  Este es el nivel de aislamiento predeterminado de `InnoDB`. Las sentencias `SELECT ... FOR UPDATE`, `SELECT ... LOCK IN SHARE MODE`, `UPDATE`, y `DELETE` que utilicen un índice único con una condición de búsqueda única,

bloquean solamente el registro de índice hallado, no el espacio vacío que lo precede. Con otras condiciones de búsqueda, estas operaciones emplean bloqueo de clave siguiente (next-key), bloqueando el rango de índice cubierto por la operación incluyendo los espacios vacíos, y bloqueando las nuevas inserciones por parte de otros usuarios.
En lecturas consistentes (consistent reads), hay una importante diferencia con respecto al nivel de aislamiento anterior: En este nivel, todas las lecturas consistentes dentro de la misma transacción leen de la captura de la base de datos tomada por la primer lectura. Esta práctica significa que si se emiten varias sentencias `SELECT` dentro de la misma transacción, éstas serán consistentes unas con otras. Consulte Sección 15.10.4, "Lecturas consistentes que no bloquean".

- `SERIALIZABLE`
  Este nivel es similar a `REPEATABLE READ`, pero todas las sentencias `SELECT` son convertidas implícitamente a `SELECT ... LOCK IN SHARE MODE`.

Lecturas consistentes que no bloquean

Lectura consistente significa que `InnoDB` utiliza su característica de multiversión para presentar a una consulta una captura de la base de datos en un momento determinado. La consulta ve los cambios realizados exactamente por aquellas transacciones confirmadas antes de ese momento, y no los cambios hechos con posterioridad o por transacciones no confirmadas. La excepción a esto es que la consulta ve los cambios efectuados por la transacción a donde pertenece.
Si se está ejecutando con el nivel de aislamiento predeterminado `REPEATABLE READ`, entonces todas las lecturas consistentes dentro de la misma transacción leen la captura creada por la primer lectura en esa transacción. Se puede refrescar esta captura confirmando la transacción actual y emitiendo nuevas consultas.
Lectura consistente es el modo por defecto en el cual `InnoDB` procesa las sentencias `SELECT` en los niveles de aislamiento `READ COMMITTED` y `REPEATABLE READ`. Una lectura consistente no establece ningún bloqueo en las tablas a las que accede, y, por lo tanto, otros usuarios están libres para modificar las tablas sobre las que se está haciendo la lectura consistente.

Bloquear lecturas `SELECT ... FOR UPDATE` y `SELECT ... LOCK IN SHARE MODE`

En ciertas circunstancias, no es conveniente una lectura consistente. Por ejemplo, se podría desear agregar una fila en la tabla `hijo`, y estar seguro de que dicha fila tiene una fila padre en la tabla `padre`. El siguiente ejemplo muestra cómo implementar integridad referencial en el código de la aplicación.
Suponiendo que se utiliza una lectura consistente para leer la tabla `padre` y efectivamente puede verse el registro padre para la fila hijo que se agregará, ¿puede agregarse en forma segura la fila hijo dentro de la tabla `hijo`? No, porque puede haber ocurrido que entretanto otro usuario haya borrado el registro padre de la tabla `padre`, sin que se tenga conocimiento de ello.

La solución es llevar a cabo el SELECT en un modo con bloqueo, utilizando LOCK IN SHARE MODE:

```
SELECT * FROM parent WHERE NAME = 'Jones' LOCK IN SHARE MODE;
```

Realizar una lectura en modo compartido (share mode) significa que se leen los últimos datos disponibles, y se establece un bloqueo en modo compartido en los registros que se leen. Un bloqueo en modo compartido evita que otros actualicen o eliminen la fila que se ha leido. Además, si los datos más actualizados pertenecen a una transacción todavía no confirmada de otra conexión, se espera hasta que la transacción se confirme. Luego de ver que la mencionada consulta devuelve el registro padre 'Jones', se puede agregar con seguridad el registro hijo en la tabla hijo y confirmar la transacción.

Otro ejemplo: se tiene un campo contador, entero, en una tabla llamada child_codes que se emplea para asignar un identificador único a cada registro hijo agregado a la tabla hijo. Obviamente, utilizar una lectura consistente o una lectura en modo compartido para leer el valor actual del contador no es una buena idea, puesto que dos usuarios de la base de datos pueden ver el mismo valor del contador, y agregar registros hijos con el mismo identificador, lo cual generaría un error de clave duplicada.

En este caso, LOCK IN SHARE MODE no es una buena solución porque si dos usuarios leen el contador al mismo tiempo, al menos uno terminará en un deadlock cuando intente actualizar el contador.

En este caso, hay dos buenas formas de implementar la lectura e incremento del contador: (1), actualizar el contador en un incremento de 1 y sólo después leerlo, o (2) leer primero el contador estableciendo un bloqueo FOR UPDATE, e incrementándolo luego. La última puede ser implementada como sigue:

```
SELECT counter_field FROM child_codes FOR UPDATE;
UPDATE child_codes SET counter_field = counter_field + 1;
```

Una sentencia SELECT ... FOR UPDATE lee el dato más actualizado disponible, estableciendo bloqueos exclusivos sobre cada fila leída. Es decir, el mismo bloqueo que haría UPDATE.

Nótese que el anterior es un sencillo ejemplo de cómo funciona SELECT ... FOR UPDATE. En MySQL, la tarea específica para generar un identificador único en realidad puede realizarse utilizando un sólo acceso a la tabla:

```
UPDATE child_codes SET counter_field = LAST_INSERT_ID(counter_field + 1);
SELECT LAST_INSERT_ID();
```

La sentencia SELECT simplemente recupera la información del identificador (relativa a la conexión actual). No accede ninguna tabla.

Bloqueo de la próxima clave (Next-Key Locking): evitar el problema fantasma

En el bloqueo a nivel de fila, InnoDB utiliza un algoritmo llamado *bloqueo de próxima clave*. InnoDB lleva a cabo el bloqueo a nivel de fila de tal manera que cuando busca o recorre el índice de una tabla, establece bloqueos compartidos o exclusivos en los registros de índice que encuentra. Por lo tanto, los bloqueos a nivel de fila son en realidad bloqueos sobre registros del índice.

El conjunto de bloqueos de InnoDB sobre los registros del índice también afecta al “gap” (posición vacía) que precede al registro de índice. Si un usuario tiene un bloqueo

compartido o exclusivo sobre un registro R en un índice, otro usuario no puede insertar un nuevo registro inmediatamente antes de R en el orden del índice. Este bloqueo de posiciones vacías se hace para evitar el llamado “problema fantasma”. Suponiendo que se desean leer y bloquear todos los hijos de la tabla hijos que tengan un identificador mayor a 100, con el posterior intento de actualizar algunas columnas en las filas seleccionadas:

```
SELECT * FROM child WHERE id > 100 FOR UPDATE;
```

Suponiendo que hay un índice sobre la columna id, la consulta recorre ese índice comenzando por el primer registro donde id es mayor a 100. Si el bloqueo establecido sobre el índice no bloqueara también las inserciones hechas en las posiciones vacías, durante el proceso se podría insertar una nueva fila en la tabla. Si se ejecuta la misma sentencia SELECT dentro de la misma transacción, se podría ver una nueva fila en el conjunto de resultados devuelto por la consulta. Esto es contrario al principio de aislamiento de las transacciones: una transacción deberia ejecutarse de forma que los datos que ha leido no cambien en el transcurso de la misma. Si se considera un conjunto de columnas como datos, el nuevo registro hijo “fantasma” violaría el principio de aislamiento.

Cuando InnoDB recorre un índice, también puede bloquear la posición vacía después del último registro del índice. Es precisamente lo que ocurre en el ejemplo anterior: Los bloqueos impuestos por InnoDB evitan cualquier inserción en la tabla donde id fuera mayor de 100.

Se puede emplear bloqueo de próxima clave para efectuar el control de la unicidad en una aplicación: Si se leen los datos en modo compartido y no se ve un duplicado de la fila que se va a insertar, entonces puede hacerse con la seguridad de que el bloqueo de próxima clave establecido sobre el registro que continúa a la fila insertada evita que cualquiera inserte un duplicado de ésta. Por lo tanto, el bloqueo de próxima clave permite “bloquear” la no existencia de algo en la tabla.

Un ejemplo de lectura consistente en InnoDB

Suponiendo que se está ejecutando en el nivel de aislamiento predeterminado REPEATABLE READ, cuando se realiza una lectura consistente -esto es, una sentencia SELECT ordinaria-, InnoDB le otorga a la transacción un punto en el tiempo (timepoint) del momento en que se realizó la consulta. Si otra transacción elimina una fila y confirma la acción en un momento posterior a dicho punto, no se verá la fila como borrada. Las inserciones y actualizaciones se tratan del mismo modo.

Se puede obtener un timepoint más reciente confirmando la transacción actual y emitiendo un nuevo SELECT.

Esto se llama *control de concurrencia multiversión.*

```
                Usuario A                   Usuario B

            SET AUTOCOMMIT=0;       SET AUTOCOMMIT=0;
tiempo
|           SELECT * FROM t;
|           empty set
|                                     INSERT INTO t VALUES (1,
2);
```

```
|
v           SELECT * FROM t;
            empty set
                                        COMMIT;

            SELECT * FROM t;
            empty set

            COMMIT;

            SELECT * FROM t;
            ---------------------
            |    1    |    2    |
            ---------------------
            1 row in set
```

En este ejemplo, el usuario A podrá ver la fila insertada por B solamente cuando B haya confirmado la inserción y A haya confirmado también, de modo que su timepoint avance e incluya la inserción confirmada por B.

Si se desea ver el "más reciente" estado de la base de datos, se debería emplear ya sea el nivel de aislamiento `READ COMMITTED` o bien una lectura con bloqueo:

```
SELECT * FROM t LOCK IN SHARE MODE;
```

Establecimiento de bloqueos con diferentes sentencias SQL en `InnoDB`

Una lectura con bloqueo, un `UPDATE`, o un `DELETE` generalmente establecen bloqueos sobre cada registro de índice que es examinado durante el procesamiento de la consulta SQL. No importa si en la consulta hay condiciones `WHERE` que excluirían la fila, `InnoDB` no recuerda exactamente la condición `WHERE`, solamente los rangos de índices que fueron examinados. Los bloqueos sobre los registros son normalmente bloqueos de próxima clave, que también impiden las inserciones en las posiciones vacías ("gap") inmediatamente anteriores a los registros.

Si los bloqueos a establecer son exclusivos, entonces `InnoDB` recupera también los registros de índices agrupados (clustered) y los bloquea.

Si no hay índices apropiados para la consulta y MySQL debe examinar la tabla entera para procesarla, se bloqueará cada fila en la tabla, lo que impide cualquier inserción de otros usuarios. Es importante crear índices adecuados de modo que las consultas no examinen muchas filas innecesariamente.

- `SELECT ... FROM` es una lectura consistente, que lee una captura de la base de datos y no establece bloqueos a menos que el nivel de aislamiento de la transacción sea `SERIALIZABLE`. Para el nivel `SERIALIZABLE`, se establecen bloqueos compartidos de próxima clave en los registros de índice encontrados.
- `SELECT ... FROM ... LOCK IN SHARE MODE` establece bloqueos compartidos de próxima clave en todos los registros de índice hallados por la lectura.
- `SELECT ... FROM ... FOR UPDATE` establece bloqueos exclusivos de próxima clave en todos los registros de índice hallados por la lectura.

- `INSERT INTO ... VALUES (...)` establece un bloqueo exclusivo sobre la fila insertada. Nótese que no se trata de un bloqueo de próxima clave, y no evita que otros usuarios inserten registros en la posición vacía precedente. Si ocurriese un error por duplicación de claves, se establecerá un bloqueo compartido sobre el registro de índice duplicado.
- Mientras se inicializa una columna previamente declarada `AUTO_INCREMENT`, `InnoDB` establece un bloqueo exclusivo al final del índice asociado con dicha columna. Al accederse al contador de autoincremento, `InnoDB` emplea una modo de bloqueo de tabla específico llamado `AUTO-INC`, que dura solamente hasta el final de la actual consulta SQL, en lugar de existir hasta el final de la transacción. Consulte Sección 15.10.2, "`InnoDB` y `AUTOCOMMIT`".
  En MySQL 5.0, `InnoDB` trae el valor de una columna previamente declarada `AUTO_INCREMENT` sin establecer ningún bloqueo.
- `INSERT INTO T SELECT ... FROM S WHERE ...` establece un bloqueo exclusivo (pero no de próxima clave) en cada fila insertada dentro de `T`. La búsqueda en `S` se hace como una lectura consistente, pero se establecen bloqueos compartidos de próxima clave en `S` si está activado el registro binario (binary logging) de MySQL. `InnoDB` tiene que establecer bloqueos en este último caso: en una recuperación de tipo roll-forward desde una copia de respaldo, cada semtencia SQL debe ser ejecutada en exactamente la misma manera en que se hizo originalmente.
- `CREATE TABLE ... SELECT ...` lleva a cabo el `SELECT` como una lectura consistente o con bloqueos compartidos, como en el punto anterior.
- `REPLACE` se ejecuta del mismo modo que una inserción si no hay colisiones con claves únicas. En otro caso, se coloca un bloqueo exclusivo de próxima clave en la fila que será actualizada.
- `UPDATE ... WHERE ...` establece un bloqueo exclusivo de próxima clave sobre cada registro encontrado por la búsqueda.
- `DELETE FROM ... WHERE ...` establece un bloqueo exclusivo de próxima clave sobre cada registro encontrado por la búsqueda.
- Si se define una restricción `FOREIGN KEY` sobre una tabla, cualquier inserción, actualización o eliminación que necesite la verificación de las condiciones impuestas por la restricción establecerá bloqueos compartidos a nivel de registro sobre los registros examinados durante la verificación. `InnoDB` también establece estos bloqueos en el caso de que la verificación falle.
- `LOCK TABLES` establece bloqueos de tabla, pero es la capa de MySQL de mayor nivel por debajo de la capa de `InnoDB` la que establece estos bloqueos. `InnoDB` tiene conocimiento de los bloqueos de tabla si se establecen `innodb_table_locks=1` y `AUTOCOMMIT=0`, y la capa de MySQL por debajo de `InnoDB` sabe acerca de los bloqueos a nivel de fila. En otro caso, la detección automática de deadlocks de InnoDB no puede detectar los deadlocks donde estén involucradas estas tablas. Además, puesto que la capa superior de MySQL no sabe acerca de bloqueos a nivel de fila, es posible obtener un bloqueo de tabla sobre una tabla donde otro usuario ha colocado bloqueos a nivel de fila. Sin embargo, esto no pone en peligro la integridad de la transacción, como se dice en Sección 15.10.10,

"Detección de interbloqueos (deadlocks) y cancelación de transacciones (rollbacks)". Consulte también Sección 15.16, "Restricciones de las tablas InnoDB".

¿Cuándo ejecuta o deshace implicitamente MySQL una transacción?

MySQL comienza cada conexión de cliente con el modo de ejecución automática (autocommit) habilitado por defecto. Cuando la ejecución automática está habilitada, MySQL realiza la confirmación luego de cada sentencia SQL, si dicha sentencia no devuelve un error.

Si se tiene desactivado el modo de ejecución automática y se cierra una conexión sin hacer una confirmación explícita de una transacción, MySQL cancelará dicha transacción.

Si una sentencia SQL devuelve un error, la confirmación o cancelación dependen del error. Consulte Sección 15.15, "Tratamiento de errores de InnoDB".

Las siguientes sentencias SQL (y sus sinónimos) provocan en MySQL una confirmación implícita de la transacción en curso:

- `ALTER TABLE`, `BEGIN`, `CREATE INDEX`, `DROP DATABASE`, `DROP INDEX`, `DROP TABLE`, `LOAD MASTER DATA`, `LOCK TABLES`, `RENAME TABLE`, `SET AUTOCOMMIT=1`, `START TRANSACTION`, `TRUNCATE`, `UNLOCK TABLES`.
- Antes de MySQL 5.0.8, `CREATE TABLE` provocaba la confirmación si se empleaba el registro binario (binary logging). A partir de MySQL 5.0.8, las sentencias `CREATE TABLE`, `TRUNCATE TABLE`, `DROP DATABASE`, y `CREATE DATABASE` provocan una confirmación implícita.
- La sentencia `CREATE TABLE` en `InnoDB` se procesa como una transacción individual. Esto significa que un `ROLLBACK` emitido por el usuario no cancelará las sentencias `CREATE TABLE` hechas durante una transacción.

Detección de interbloqueos (deadlocks) y cancelación de transacciones (rollbacks)

`InnoDB` detecta automáticamente un deadlock de transacciones y cancela una o más transacciones para evitarlo. `InnoDB` intenta escoger para cancelar transacciones pequeñas, el tamaño de la transacción es determinado por el número de filas insertadas, actualizadas, o eliminadas.

`InnoDB` se mantiene al tanto de los bloqueos de tablas si `innodb_table_locks=1` (1 es el valor predeterminado), y la capa MySQL por debajo sabe acerca de bloqueos a nivel de fila. En otro caso, `InnoDB` no puede detectar deadlocks cuando están involucrados un bloqueo de tabla establecido por una sentencia `LOCK TABLES` o por otro motor de almacenamiento que no sea `InnoDB`. Estas situaciones se deben resolver estableciendo el valor de la variable de sistema `innodb_lock_wait_timeout`.

Cuando `InnoDB` lleva a cabo una cancelación completa de una transacción, todos los bloqueos de la transacción son liberados. Sin embargo, si solamente se cancela como resultado de un error una sentencia SQL individual, algunos de los bloqueos impuestos por la sentencia SQL podrían mantenerse. Esto se debe a que `InnoDB` guarda los bloqueos de fila en un formato en el que no puede luego saber qué sentencia SQL originó cada bloqueo.

Cómo tratar con interbloqueos

Los deadlocks son un problema clásico de las bases de datos transaccionales, pero no son peligrosos a menos que sean tan frecuentes que no se puedan ejecutar en absoluto ciertas transacciones. Normalmente, las aplicaciones deben ser escritas de modo que esten preparadas para emitir nuevamente una transacción si ésta es cancelada debido a un deadlock.
`InnoDB` emplea bloqueos automáticos a nivel de fila. Se pueden producir deadlocks aún en el caso de transacciones que solamente insertan o eliminan una fila individual. Esto se debe a que estas operaciones no son realmente "atómicas"; sino que establecen automáticamente bloqueos enlos (posiblemente varios) registros de índice de la fila insertada o eliminada.
Con las siguientes técnicas se puede estar a cubierto de los deadlocks y reducir la probabilidad de que ocurran:

- Emplear `SHOW INNODB STATUS` para determinar la causa del último deadlock. Puede ayudar a afinar la aplicación para evitar que ocurran otros.
- Siempre hay que estar preparado para emitir nuevamente una transacción que haya fallado por un deadlock. Los deadlocks no revisten peligro, simplemente hay que intentar de nuevo.
- Confirmar las transacciones frecuentemente. Las transacciones pequeñas son menos propensas a originar conflictos.
- Si se están usando lecturas que establecen bloqueos (`SELECT ... FOR UPDATE` o `... LOCK IN SHARE MODE`), hay que intentar utilizar un nivel de aislamiento bajo, como `READ COMMITTED`.
- Acceder a las tablas y filas en un orden fijo. Entonces, las transacciones forman secuencias bien definidas y no originan deadlocks.
- Agregar a las tablas índices adecuadamente elegidos. Entonces las consultas necesitarán examinar menos registros de índice y en consecuencia establecerán menos bloqueos. Utilizar `EXPLAIN SELECT` para determinar los índices que MySQL considera más apropiados para las consultas.
- Utilizar menos el bloqueo. Si es aceptable que `SELECT` devuelva datos de una captura de la base de datos que no sea la más actualizada, no hay que agregarle las cláusulas `FOR UPDATE` o `LOCK IN SHARE MODE`. En este caso es adecuado utilizar el nivel de aislamiento `READ COMMITTED`, porque cada lectura consistente dentro de la misma transacción leerá de su propia captura más reciente.
- Si nada de esto ayuda, habrá que serializar las transacciones con bloqueos a nivel de tabla. La forma correcta de emplear `LOCK TABLES` con tablas transaccionales, como InnoDB, es establecer `AUTOCOMMIT = 0` y no invocar a `UNLOCK TABLES` hasta que se haya confirmado explícitamente la transacción. Por ejemplo, si se necesitara escribir en una tabla `t1` y leer desde una tabla `t2`, se puede hacer esto:

```
SET AUTOCOMMIT=0;
LOCK TABLES t1 WRITE, t2 READ, ...;
[aquí se hace algo con las tablas t1 y t2];
COMMIT;
     UNLOCK TABLES;
```

Los bloqueos a nivel de tabla favorecen el funcionamiento de la cola de transacciones, y evitan los deadlocks.

- Otra manera de serializar transacciones es crear una tabla "semáforo" auxiliar que contenga sólo una fila. Hay que hacer que cada transacción actualice esa fila antes de acceder otras tablas. De ese modo, todas las transacciones se producirán en serie. Nótese que el algoritmo de detección instantánea de deadlocks de `InnoDB` también funciona en este caso, porque el bloqueo de serialización es un bloqueo a nivel de fila. Con los bloqueos a nivel de tabla de MySQL, debe emplearse el método de timeout para solucionar deadlocks.
- En aquellas aplicaciones que emplean el comando de MySQL `LOCK TABLES`, MySQL no establece bloqueos de tabla si `AUTOCOMMIT=1`.

Tomado de:
http://dev.mysql.com/doc/refman/5.0/es/innodb-transaction-model.html