

Bases de datos

Unidade 6: Transaccións e control de concorrencia

1.	A03. Transaccións e control de concurrencia	3
1.1	Introdución	3
1.2	Actividade	3
1.2.1	Transaccións	3
	Uso de transaccións en MySQL	4
1.2.2	Políticas de bloqueo de táboas	8
	Comandos de bloqueo de táboas	8
	Tipos de bloqueo	8
	Adquisición-liberación dun bloqueo	9
	Bloqueos e transaccións	9
	Tipos de bloqueo en InnoDB	9
	Niveis de illamento	16
	Insercións concorrentes	21
1.3	Tarefas	21
1.3.1	Tarefa 1. Rematar unha transacción.	21
	Solución	22
1.3.2	Tarefa 2. Empregar transaccións	22
	Solución	22
1.3.3	Tarefa 3 . Empregar transaccións implícitas	22
	Solución	22
1.3.4	Tarefa 4. Executar transaccións con diferentes niveis de bloqueo	23
	Solución:	23
1.3.5	Tarefa 5. Inserir datos mediante transaccións	24
	Solución:	24
1.3.6	Tarefa 6. Validar transaccións	24
	Solución	25
1.3.7	Tarefa 7. Establecer opcións de illamento.	25
	Solución	26

1. A03. Transaccións e control de concurrencia

1.1 Introducción

Os obxectivos desta actividade son:

- Entender o concepto de transacción e a súa utilización en MySQL.
- Coñecer os tipos de bloqueo e o seu emprego en MySQL.

Nesta actividade utilizarase a plataforma WAMP (Windows-Apache-MySQL-PHP) WampServer 2.5 (última versión estable en outubro 2015), que inclúe MySQL Community Edition 5.6.17 como SXBDR (Sistema Xestor de Bases de Datos Relacional). As razóns de utilización deste software son:

- É software libre, polo que o alumnado poderá descargalo de forma gratuíta e utilizalo legalmente na súa casa.
- É unha forma sinxela de facer a instalación do software necesario para desenvolver aplicacións web.



Páxina oficial de WampServer: <http://www.wampserver.com>



Páxina oficial MySQL: <https://www.mysql.com/>

Utilizarase MySQL Workbench 6.3 como ferramenta cliente gráfica para facer as prácticas asociadas a esta actividade, xa que é a recomendada por MySQL en outubro de 2015, aínda que tamén poderían utilizarse outras como phpMyAdmin, EMS MyManager, ou MySQL Query Browser.



En <https://www.mysql.com/products/workbench/> pode obterse información detallada sobre a ferramenta MySQL Workbench e descargar o software. En <http://dev.mysql.com/doc/index-gui.html> pode descargarse o manual de MySQL Workbench.

1.2 Actividade

1.2.1 Transaccións

Unha transacción é un conxunto de instrucións SQL que se executan de maneira atómica ou indivisible como unha unidade, é dicir, ou se executan todas as instrucións ou non se executa ningunha. Se unha transacción ten éxito, todas as modificacións dos datos realizados durante a transacción se gardan na base de datos. Se unha transacción contén erros os cambios non se gardarán na base de datos.

Unha transacción debe cumprir as catro propiedades ACID:

- **Atomicidade:** asegura que se realizan todas as operacións ou ningunha, non pode quedar a medias.
- **Consistencia** ou integridade: asegura que só se empeza o que se pode acabar.

- **Illamento:** asegura que ningunha operación afecta a outras, con isto se asegura que varias transaccións sobre a mesma información sexan independentes e non xeren ningún tipo de erro.
- **Durabilidade:** asegura que unha vez realizada a operación, esta non poderá cambiar e permanecerán os cambios. Isto é, se o disco duro falla, o sistema aínda será capaz de lembrar todas as transaccións que se realizaron no sistema.

Un exemplo típico de transacción é unha transferencia económica na que debe subtraerse unha cantidade dunha conta, facer unha serie de cálculos relacionados con comisións, intereses, etc. Todo iso debe ocorrer de maneira simultánea coma se fose unha soa operación, xa que doutro xeito xeraríanse inconsistencias.

Unha das características dos sistemas xestores é se permiten ou non o uso de transaccións nas súas táboas. No caso de MySQL isto depende do tipo de táboa ou motor utilizado. MySQL soporta distintos tipos de táboas tales como ISAM, MyISAM, InnoDB e BDB (Berkeley Database). As táboas que permiten transaccións son do tipo InnoDB. Están estruturadas de xeito distinto que MyISAM, xa que se almacenan nun só arquivo en lugar de tres e, ademais de transaccións, permiten definir regras de integridade referencial.

As transaccións achegan unha fiabilidade superior ás bases de datos. Si dispomos dunha serie de operacións SQL que deben executarse en conxunto, co uso de transaccións podemos ter a certeza de que nunca quedaremos a medio camiño da súa execución. De feito, as transaccións teñen a característica de desfacer as aplicacións de bases de datos.

As táboas que soportan transaccións, son moito máis seguras e fáciles de recuperar no caso de producirse algún fallo no servidor, xa que as instrucións se executan non na súa totalidade. Por outra banda, as transaccións poden aumentar o tempo de proceso de instrucións.

No seguinte exemplo, se unha cantidade de diñeiro é transferida da conta dun cliente, requíranse polo menos dúas instrucións de actualización:

```
UPDATE tbl_contas SET balance = saldo - cantidadeTransferida WHERE idCliente='cc1';
UPDATE tbl_contas SET balance = saldo + cantidadeTransferida WHERE idCliente='cc2';
```

Estas dúas consultas deben traballar ben, pero que sucede si ocorre algún imprevisto e cáese o sistema despois de que se executa a primeira instrución e a segunda aínda non se completou? O cliente 1 terá unha cantidade de diñeiro descontada da súa conta, e crerá que realizou o seu pago, con todo, o cliente 2 pensará que non se lle depositou o diñeiro que se lle debe. Neste sinxelo exemplo, móstrase a necesidade de que as consultas, ou ben sexan executadas de maneira conxunta ou que non se execute ningunha delas. É neste tipo de situacións onde as transaccións desempeñan un papel crucial.

Unha transacción ten dous finais posibles, COMMIT (execútanse todas as instrucións e gardamos os datos) e ROLLBACK (se produce un erro e non se gardan os cambios). Por defecto, MySQL trae activado o modo “autocommit”, polo que cando se realiza unha transacción (INSERT, UPDATE o DELETE) esta se confirma automaticamente. Para desactivar esta opción se debe executar o seguinte comando (non recomendado):

```
> SET AUTOCOMMIT=0;
/* ou tamén se pode desactivar para unha serie de comandos utilizando START TRANSACTION. (Isto é o aconsellable). */
> START TRANSACTION;
> .....
> COMMIT;
```

Uso de transaccións en MySQL

Os pasos para usar transaccións en MySQL son:

- Iniciar unha transacción co uso da sentenza START TRANSACTION ou BEGIN.
- Actualizar, inserir ou eliminar rexistros na base de datos.

- No caso de querer reflectir os cambios na base de datos, completárase a transacción co uso da sentenza COMMIT. Unicamente cando se procesa un COMMIT os cambios feitos polas consultas serán permanentes.
- Se acontece algún problema, poderase facer ROLLBACK para cancelar os cambios que foron realizados polas consultas executadas ata o momento.

En táboas InnoDB, toda actividade do usuario prodúcese dentro dunha transacción. Se o modo de execución automática (autocommit) está activado, cada sentenza SQL conforma unha transacción individual por si mesma. MySQL sempre comeza unha nova conexión coa execución automática habilitada.

Se o modo de execución automática se inhabilitou con SET AUTOCOMMIT=0, entón pode considerarse que un usuario sempre ten unha transacción aberta. A transacción vixente remataríase cunha das sentenzas seguintes:

- Unha sentenza SQL COMMIT, significa que os cambios feitos na transacción actual convértense en permanentes e vólvense visibles para os outros usuarios.
- Unha sentenza ROLLBACK, cancela todas as modificacións producidas na transacción actual.

Ambas sentenzas liberan todos os bloqueos *InnoDB* que se estableceron durante a transacción vixente.

Se a conexión ten a execución automática habilitada, o usuario pode igualmente levar a cabo unha transacción con varias sentenzas se a comeza explicitamente con START TRANSACTION ou BEGIN e a remata con COMMIT ou ROLLBACK.

O primeiro que se debe facer á hora de traballar con transaccións é comprobar o estado da variable “autocommit”:

```
SHOW VARIABLES LIKE 'autocommit';
```

Se ten o valor 1 desactivámola con SET. Outra opción é realizar os exemplos con START TRANSACTION.

No seguinte exemplo, creamos unha táboa, na base de datos “*bd_ProbaTransaccions*” do tipo InnoDB e inserimos algúns datos. Para crear unha táboa *InnoDB*, procederase co código SQL estándar CREATE TABLE, pero debemos especificar que se trata dunha táboa do tipo *InnoDB* (TYPE = InnoDB). A táboa chamarase “*tbl_ProbaTransaccions*” e terá un campo numérico. Primeiro activamos a base de datos “*bd_ProbaTransaccions*” coa instrución USE para despois crear a táboa e introducir algúns valores:

```
CREATE DATABASE `bd_ProbaTransaccions`;
USE `bd_ProbaTransaccions`;
SHOW VARIABLES LIKE 'AUTOCOMMIT';
```

	Variable_name	Value
►	autocommit	ON

```
USE `bd_ProbaTransaccions`;
CREATE TABLE tbl_ProbaTransaccions (id INT NOT NULL PRIMARY KEY, s VARCHAR (30), si SMALLINT) ENGINE = InnoDB ;
INSERT INTO tbl_ProbaTransaccions ( id, s, si) VALUES (1, "primeiro", NULL);
INSERT INTO tbl_ProbaTransaccions ( id, s, si) VALUES (2, "segundo", NULL);
INSERT INTO tbl_ProbaTransaccions ( id, s, si) VALUES (3, "terceiro", NULL);
SELECT * FROM tbl_ProbaTransaccions;
```

	id	s	si
▶	1	primeiro	NULL
	2	segundo	NULL
	3	terceiro	NULL

Unha vez cargada a táboa iniciamos unha transacción:

```
BEGIN;
INSERT INTO tbl_ProbaTransaccions ( id, s, si) VALUES (4, "cuarto", NULL);
ROLLBACK;
SELECT * FROM tbl_ProbaTransaccions;
```

Ao executar un ROLLBACK, a transacción non será completada e os cambios realizados sobre a táboa non terán efecto:

	id	s	si
▶	1	primeiro	NULL
	2	segundo	NULL
	3	terceiro	NULL

Se agora facemos un SELECT para mostrar os datos de *tbl_ProbaTransaccions*”, comprobárase que non se produciu ningunha inserción.

Agora imos ver que sucede se perdemos a conexión ao servidor antes de que a transacción sexa completada.

```
BEGIN;
INSERT INTO tbl_ProbaTransaccions ( id, s, si) VALUES (4, "cuarto", NULL);
SELECT * FROM tbl_ProbaTransaccions;
```

	id	s	si
▶	1	primeiro	NULL
	2	segundo	NULL
	3	terceiro	NULL
	4	cuarto	NULL

--CAIDA DO SISTEMA SIMULADA: pechamos a conexión

Cando obtemos de novo a conexión, podemos verificar que o rexistro non se inseriu, xa que a transacción non foi completada.

```
USE `bd_ProbaTransaccions`;
SELECT * FROM bl_ProbaTransaccions;
```

	id	s	si
▶	1	primeiro	NULL
	2	segundo	NULL
	3	terceiro	NULL

No seguinte exemplo, repetirase a sentenza INSERT executada anteriormente e engadirase outra, pero facendo un COMMIT antes de perder a conexión ao servidor ao saír do monitor de MySQL.

```
BEGIN;  
INSERT INTO tbl_ProbaTransaccions ( id, s, si) VALUES (4, "cuarto", NULL);  
COMMIT;  
INSERT INTO tbl_ProbaTransaccions ( id, s, si) VALUES (5, "quinto", NULL);  
COMMIT;  
USE `bd_ProbaTransaccions`;  
SELECT * FROM tbl_ProbaTransaccions;
```

	id	s	si
▶	1	primeiro	NULL
	2	segundo	NULL
	3	terceiro	NULL
	4	cuarto	NULL
	5	quinto	NULL

Unha vez que facemos un COMMIT, a transacción é completada e todas as sentenzas SQL que foron executadas previamente afectan de maneira permanente ás táboas da base de datos.



Tarefa 1. Rematar unha transacción.

Hai instrucións SQL que dadas as súas características, implican confirmación automática. Estas son:

- Instrucións do DDL que definen ou modifican obxectos da base de datos (CREATE, ALTER, DROP, RENAME e TRUNCATE).
- Instrucións que modifican táboas da base de datos administrativa chamada *mysql*, (GRANT e REVOKE).
- Instrucións de control de transaccións e bloqueo de táboas (START TRANSACTION, BEGIN, LOCK e UNLOCK).
- Instrucións de carga de datos (LOAD DATA).
- Instrucións de administración de táboas (ANALIZE, CHECK, OPTIMIZE e REPAIR).

Existen ademais dous comandos para a xestión de transaccións:

- SAVEPOINT id: é unha instrución que permite nomear certo paso nun conxunto de instrucións dunha transacción.
- ROLLBACK TO SAVEPOINT id: permite desfacer o conxunto de operacións realizadas a partir do identificador de *savepoint*.
- RELEASE SAVEPOINT id: elimina ou libera o *savepoint* creado.

Toda operación COMMIT ou ROLLBACK sen argumentos eliminará todos os *savepoints* creados.

```
START TRANSACTION;  
INSERT INTO tbl_ProbaTransaccions ( id, s, si) VALUES (6, "sexto", NULL);  
SAVEPOINT un;  
INSERT INTO tbl_ProbaTransaccions ( id, s, si) VALUES (7, "setimo", NULL);  
INSERT INTO tbl_ProbaTransaccions ( id, s, si) VALUES (8, "oitavo", NULL);  
ROLLBACK TO un;  
SELECT * FROM tbl_ProbaTransaccions;
```

Este sinxelo exemplo fai que só se insiran realmente (se confirmen) os datos do primeiro INSERT. Comprobámolo cunha sentenza SELECT.

id	s	si
1	primeiro	NULL
2	segundo	NULL
3	terceiro	NULL
4	cuarto	NULL
5	quinto	NULL
6	sexto	NULL

1.2.2 Políticas de bloqueo de táboas

Bloquear unha táboa ou vista permite que ninguén máis poida facer uso da mesma de modo que temos a exclusividade de lectura e/ou escritura sobre ela.

MySQL permite o bloqueo de táboas por parte de clientes có obxecto de cooperar con outras sesións de clientes ou de evitar que estes poidan modificar datos que necesitamos na nosa sesión.

Os bloqueos permiten simular transaccións, así como acelerar operacións de modificación ou de inserción de datos.

Comandos de bloqueo de táboas

No bloqueo dunha táboa especificarase o seu nome (nome_táboa) ou alias, e o tipo de bloqueo (tipo_bloqueo), lectura ou escritura (READ / WRITE) na instrución LOCK para bloqueo de táboas e segue a seguinte sintaxe:

```
LOCK TABLES
  nome_táboa [[AS] alias] tipo_bloqueo
  [, nome_táboa [[AS] alias] tipo_bloqueo] ...
tipo_bloqueo:
  READ [LOCAL]
  | [LOW_PRIORITY] WRITE
```

A sintaxe da instrución para desbloquear as táboas é a seguinte:

```
UNLOCK TABLES
```

Só podemos desbloquear todas as táboas á vez.

Para obter un bloqueo de todas as táboas da base de datos empregamos:

```
LOCK TABLES WITH READ LOCK
```

Tipos de bloqueo

Existen dous tipos de bloqueo:

- READ [LOCAL]

Neste caso a sesión ou cliente que ten o bloqueo pode ler pero nin el nin ningún outro cliente poderán escribir na táboa.

É un bloqueo que poden adquirir varios clientes simultaneamente e permite que calquera, mesmo sen o bloqueo, poida ler as táboas bloqueadas.

O modificador LOCAL permite que haxa insercións concorrentes doutros clientes mentres dura o bloqueo.

▪ [LOW PRIORITY] WRITE

A sesión ou cliente que adquire este tipo de bloqueo pode ler e escribir na táboa pero ningún outro cliente poderá acceder a ela ou bloqueala.

Adquisición-liberación dun bloqueo

Cando necesitamos adquirir bloqueos debemos facelo nunha única sentenza, xa que se separamos as instrucións automaticamente se desbloquearán as anteriores quedando activo soamente o último bloqueo.

Nese momento só teremos acceso ás táboas bloqueadas.

Os bloqueos de escritura teñen por defecto maior prioridade xa que implican modificación de datos que deben facerse o máis rápido posible. Deste xeito, se unha sesión adquire un bloqueo de lectura e outra sesión pretende un de escritura, este terá prioridade sobre outras solicitudes de bloqueos de lectura subsecuentes. Desta maneira, ata que a sesión que solicitou o bloqueo de escritura non libere devandito bloqueo, non se poderán adquirir novos bloqueos de lectura.

Isto cambia se o bloqueo de escritura se adquirira coa opción `LOW_PRIORITY`, nese caso si se permite que os bloqueos de lectura se adquiran antes que o de escritura. De feito o bloqueo de escritura só se obterá cando non queden bloqueos de lectura pendentes.

A política de bloqueos de MySQL segue a seguinte secuencia:

1. Ordena internamente as táboas a bloquear.
2. Se unha táboa debe bloquearse para lectura e escritura sitúa a solicitude de bloqueo de escritura en primeiro lugar.
3. Bloquéase cada táboa ata que a sesión obtén todos os seus bloqueos.

Deste xeito se evita o coñecido *deadlock* ou bloqueo mutuo, fenómeno que fai que o acceso aos bloqueos se poida prolongar indefinidamente ao non poder ningún proceso obtelos.

Se adquirimos un bloqueo sobre unha táboa, todos os bloqueos activos nese momento libéranse automaticamente. Se comezamos unha transacción todos os bloqueos libéranse automaticamente.

Bloqueos e transaccións

O uso de bloqueos está ligado ás transaccións, especialmente para táboas de tipo transaccional como *InnoDB* e *Cluster*.

En táboas *InnoDB* os bloqueos adquirense a nivel de fila permitíndose que varios usuarios poidan bloquear varias filas simultaneamente.

Cómpre lembrar que en táboas *InnoDB* todo é unha transacción, de feito, se *autocommit* está activado (=1), cada operación SQL é en si mesma unha transacción. Neste caso podemos iniciar unha transacción con `START TRANSACTION` ou `BEGIN` e terminala con `COMMIT` para que os cambios sexan permanentes ou `ROLLBACK` para desfacer os cambios.

No caso de *autocommit* inactivo considérase que sempre hai unha transacción en curso, nese caso as sentenzas `COMMIT` e `ROLLBACK` supoñen o final de dita transacción e comezo da seguinte.

Tipos de bloqueo en InnoDB

Neste tipo de táboas diferenciamos dous tipos de bloqueo:

- Bloqueo compartido (s): permite a unha transacción a lectura de filas. Neste caso varias transaccións poden adquirir bloqueos sobre as mesmas filas pero ningunha transacción pode modificar ditas filas ata que non se liberen os bloqueos.

- Bloqueo exclusivo (x): permite a unha transacción bloquear filas para actualización ou borrado. Neste caso as transaccións que desexen adquirir un bloqueo exclusivo deberán esperar a que se libere o bloqueo sobre as filas afectadas.

Tamén se soporta o bloqueo de múltiple granularidade segundo o cal unha transacción pode indicar que vai bloquear algunhas filas dunha táboa ben para lectura (IS) ou para escritura (IX). É o que se denomina unha intención de bloqueo.

Deste xeito, é máis fácil para MySQL xestionar posibles conflitos evitando os temidos *deadlocks* xa que permite a varias transaccións compartir a reserva dunha táboa posto que o bloqueo prodúcese fila a fila, no momento da modificación. Así se dúas transaccións reservan a mesma táboa, só no momento en que unha delas está modificando unha fila, esta quedará bloqueada.

Exemplos deste tipo de bloqueo son:

- Sentenzas `SELECT ...LOCK IN SHARE MODE`: para bloqueo tipo IS. Deste xeito créase un bloqueo de lectura sobre as filas afectadas pola consulta `SELECT`.

O uso desta sentenza móstrase no seguinte exemplo. Supoñamos dous usuarios da base de datos *bd_ProbaTransaccions*.

```
CREATE USER 'usuario1'@'localhost' IDENTIFIED BY 'usuario1';
CREATE USER 'usuario2'@'localhost' IDENTIFIED BY 'usuario2';
CREATE USER 'usuario3'@'localhost' IDENTIFIED BY 'usuario3';
SELECT * FROM mysql.user;
GRANT ALL ON bd_ProbaTransaccions.* TO 'usuario1'@'localhost'
GRANT ALL ON bd_ProbaTransaccions.* TO 'usuario2'@'localhost'
GRANT ALL ON bd_ProbaTransaccions.* TO 'usuario3'@'localhost'
```

ambos usuarios desexan modificar un rexistro na táboa *tbl_ProbaTransaccions*:

```
/* Comprobación do usuario actual que corresponde a 'usuario1' */
SELECT USER();
/* Activación da base de datos */
USE `bd_ProbaTransaccions`;
/* Comezamos a transacción */
START TRANSACTION;
```

13	20:35:53	START TRANSACTION	0 row(s) affected	0.010 sec
Event 13 of type Info at 20:35:53				
Action:	START TRANSACTION			
Message:	0 row(s) affected			

```
/* Crearase un bloqueo de lectura sobre as filas a modificar* /
SELECT * FROM tbl_ProbaTransaccions WHERE id=1 LOCK IN SHARE MODE;
```

14	20:38:39	SELECT * FROM tbl_ProbaTransaccions WHERE id=1 LIMIT 0, 1000 LOCK IN SHARE MODE	1 row(s) returned	0.060 sec / 0.030 sec
Event 14 of type Info at 20:38:39				
Action:	SELECT * FROM tbl_ProbaTransaccions WHERE id=1 LIMIT 0, 1000 LOCK IN SHARE MODE			
Message:	1 row(s) returned			

	id	s	si
▶	1	primeiro	NULL

```

/* Realízase a modificación */
UPDATE tbl_ProbaTransaccions
SET s = "modificado por usuario1"
WHERE id=1;

```

15	20:44:04	UPDATE tbl_ProbaTransaccions SET s = "modificado por ...	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.000 sec
Event 15 of type Info at 20:44:04				
Action:		UPDATE tbl_ProbaTransaccions SET s = "modificado por usuario1" WHERE id=1		
Message:		1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0		

```

/* Mentres non se execute o COMMIT a táboa estará bloqueada e a modificación non SERÁ efectiva pero permite mostrala */
SELECT * FROM tbl_ProbaTransaccions WHERE id=1;

```

	id	s	si
▶	1	modificado por usuario1	NULL

```

/ Faranse efectivos o cambios e liberarase a táboa /
COMMIT;

```

17	20:47:00	COMMIT	0 row(s) affected	0.000 sec
Event 17 of type Info at 20:47:00				
Action:		COMMIT		
Message:		0 row(s) affected		

```

/* Mostra o valor actual de s que será "modificada polo usuario2" xa que tan pronto co- mo quede liberada a táboa será modificada polo outro usuario */
SELECT * FROM tbl_ProbaTransaccions WHERE id=1;
/* Comprobación do usuario actual que é 'usuario2' */
SELECT USER();
/* Activación da base de datos */
USE `bd_ProbaTransaccions`;
/* Comprobación do estado da variable autocommit que debe ser ON (1) */
SHOW VARIABLES LIKE 'autocommit';
/* Acceso ao contido da táboa xa bloqueada */
SELECT * FROM tbl_ProbaTransaccions WHERE id=1;

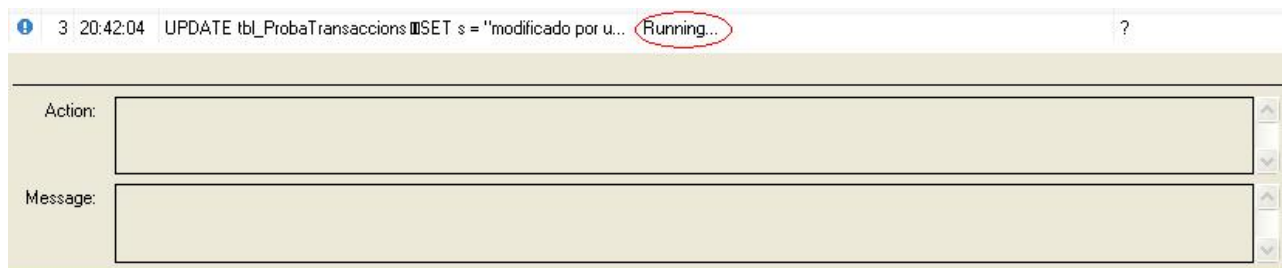
```

	id	s	si
▶	1	primeiro	NULL

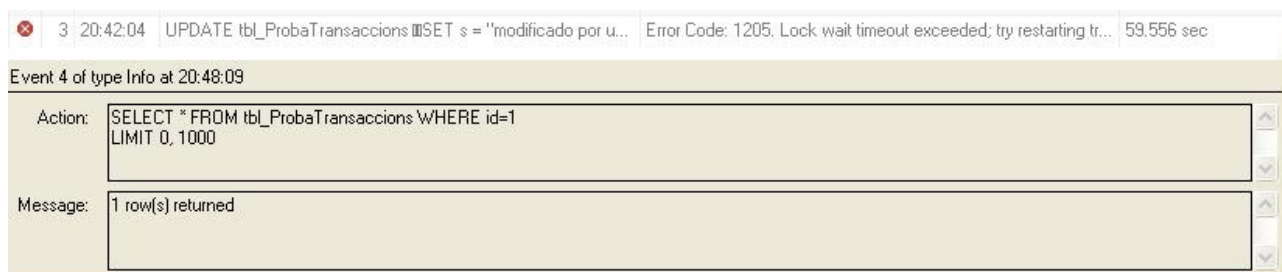
```

/* A táboa está bloqueada polo que esta instrución permanece á espera /

```



/* Se pasa moito tempo bloqueada (en mySQL é por defecto 90 segundos) a instrución anúlase polo que hai que volver a executala */



	id	s	si
▶	1	modificado por usuario1	NULL



```
UPDATE tbl_ProbaTransaccions
SET s = "modificado por usuario2"
WHERE id=1;
/* Mostrase o valor actual de "s" */
SELECT * FROM tbl_ProbaTransaccions WHERE id=1;
```

	id	s	si
▶	1	modificado por usuario2	NULL

A característica deste tipo de boqueo é que permite a varios usuarios bloquear a mesma información. No seguinte exemplo, ilústrase como un mesmo rexistro é bloqueado por dous usuarios e como un terceiro trata de modificalo.

O seguinte código corresponde a sentenzas executadas polo usuario *usuario1*:

```
/* O usuario1 bloquea o rexistro2 */
/* Comezamos a transacción */
START TRANSACTION;
/* Crearase un bloqueio de lectura sobre as filas a modificar */
```

4	11:26:32	SELECT * FROM tbl_ProbaTransaccions WHERE id=2 LIMIT 0,1000 LOCK IN SHARE MODE	1 row(s) returned	0.000 sec / 0.030 sec
Event 4 of type Info at 11:26:32				
Action:	SELECT * FROM tbl_ProbaTransaccions WHERE id=2 LIMIT 0,1000 LOCK IN SHARE MODE			
Message:	1 row(s) returned			

	id	s	si
▶	2	segundo	NULL

```
SELECT * FROM tbl_ProbaTransaccions WHERE id=2 LOCK IN SHARE MODE;
/* Realízase a modificación */
```

9	11:34:25	UPDATE tbl_ProbaTransaccions SET s = "usuario1 modifica rexistro2" WHERE id=2	Running...	?
Event 9 of type Info at 11:34:25				
Action:	UPDATE tbl_ProbaTransaccions SET s = "usuario1 modifica rexistro2" WHERE id=2			
Message:	Running...			

```
UPDATE tbl_ProbaTransaccions
SET s = "usuario1 modifica rexistro1"
WHERE id=2;
/* Ata non executar o commit a táboa estará bloqueada e a modificación non será efectiva */
SELECT * FROM tbl_ProbaTransaccions WHERE id=2;
```

	id	s	si
▶	2	usuario1 modifica rexistro1	NULL

```
/* Faranse efectivos o cambios e liberarase a táboa */
COMMIT;
SELECT * FROM tbl_ProbaTransaccions WHERE id=2;
```

O usuario 2 en paralelo executa o seguinte código:

```
/* O usuario2 bloquea o rexistro2 */
/* Comezamos a transacción */
START TRANSACTION;
/* Crearase un bloqueo de lectura sobre as filas a modificar */
SELECT * FROM tbl_ProbaTransaccions WHERE id=2 LOCK IN SHARE MODE;
```

4	11:26:32	SELECT * FROM tbl_ProbaTransaccions WHERE id=2 LIMIT 0,1000 LOCK IN SHARE MODE	1 row(s) returned	0.000 sec / 0.030 sec
Event 4 of type Info at 11:26:32				
Action:	SELECT * FROM tbl_ProbaTransaccions WHERE id=2 LIMIT 0,1000 LOCK IN SHARE MODE			
Message:	1 row(s) returned			

```
/* Executarase a sentenza de modificación pero atópase bloqueada polo Usuario1 */
UPDATE tbl_ProbaTransaccions
```

```
SET s = "usuario2 modifica rexistro2"
WHERE id=2;
```

9	11:34:25	UPDATE tbl_ProbaTransaccions SET s = "usuario1 modifi...	Running...	?
Event 9 of type Info at 11:34:25				
Action:	UPDATE tbl_ProbaTransaccions SET s = "usuario2 modifica rexistro2" WHERE id=2			
Message:	Running...			

```
/* Unha vez desbloqueada ata non executar o COMMIT a táboa mantén a informa-
ción anterior permite mostrala */
```

```
SELECT * FROM tbl_ProbaTransaccions WHERE id=2;
```

	id	s	si
▶	2	usuario2 modifica rexistro2	NULL

```
/* Faranse efectivos os cambios e liberarase a táboa */
```

```
COMMIT;
```

```
SELECT * FROM tbl_ProbaTransaccions WHERE id=2;
```

O seguinte corresponde ao código a executar polo usuario *usuario3*:

```
/* Comprobación do usuario actual que corresponde ao 'usuario3'*/
```

```
SELECT USER();
```

```
/* Activación da base de datos polo usuario3 */
```

```
USE `bd_ProbaTransaccions`;
```

```
/* Comprobación do estado da variable AUTOCOMMIT que debe ser ON (1)*/
```

```
SHOW VARIABLES LIKE 'autocommit';
```

```
/* A táboa esta bloqueada por ambos usuarios polo que esta instrución permanece á espe-
ra, se pasa moito tempo bloqueada a instrución anularase polo que habería que volver a
executala */
```

```
UPDATE tbl_ProbaTransaccions
```

```
SET s = "usuario3 modifica rexistro2"
```

```
WHERE id=2;
```

7	11:46:28	UPDATE tbl_ProbaTransaccions SET s = "usuario3 modifi...	Running...	?
Event 7 of type Info at 11:46:28				
Action:	UPDATE tbl_ProbaTransaccions SET s = "usuario3 modifica rexistro2" WHERE id=2			
Message:	Running...			

```
SELECT * FROM tbl_ProbaTransaccions WHERE id=2;
```

```
/* Unha vez desbloqueada a modificación false efectiva */
```

	id	s	si
▶	2	usuario3 modifica rexistro2	NULL

- Sentenzas SELECT ... FOR UPDATE: para bloqueo tipo IX. Deste xeito bloqueáanse para escritura as filas afectadas pola consulta SELECT e as entradas de índice asociadas. Se facemos un bloqueo de lectura é posible que máis dunha transacción acceda ao mesmo valor, pero con este tipo de bloqueo unha única transacción será a que pode acceder á información.

O código a executar polo primeiro usuario é:

```
/* Comprobación do usuario actual que é 'usuario1' */
SELECT USER();
/* Activación da base de datos */
USE `bd_ProbaTransaccions`;
/ Comezamos a transacción */
START TRANSACTION;
/* Crearase un bloqueo de escritura sobre as filas a modificar */
SELECT * FROM tbl_ProbaTransaccions WHERE id=3 FOR UPDATE;
```

4	12:36:51	SELECT * FROM tbl_ProbaTransaccions WHERE id=3 LIMIT 0,1000 FOR UPDATE	1 row(s) returned	0.020 sec / 0.030 sec
Event 4 of type Info at 12:36:51				
Action:	SELECT * FROM tbl_ProbaTransaccions WHERE id=3 LIMIT 0,1000 FOR UPDATE			
Message:	1 row(s) returned			

	id	s	si
▶	3	terceiro	NULL

```
/* Realízase a modificación */
UPDATE tbl_ProbaTransaccions
SET s = "escritura por usuario1"
WHERE id=3;
```

5	12:37:49	UPDATE tbl_ProbaTransaccions SET s = "escritura por usu... WHERE id=3	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.141 sec
Event 5 of type Info at 12:37:49				
Action:	UPDATE tbl_ProbaTransaccions SET s = "escritura por usuario1" WHERE id=3			
Message:	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0			

/* Mentres non se execute o COMMIT a táboa estará bloqueada e a modificación non será efectiva pero permitirase mostrala */

```
SELECT * FROM tbl_ProbaTransaccions WHERE id=3;
```

	id	s	si
▶	3	escritura por usuario1	NULL

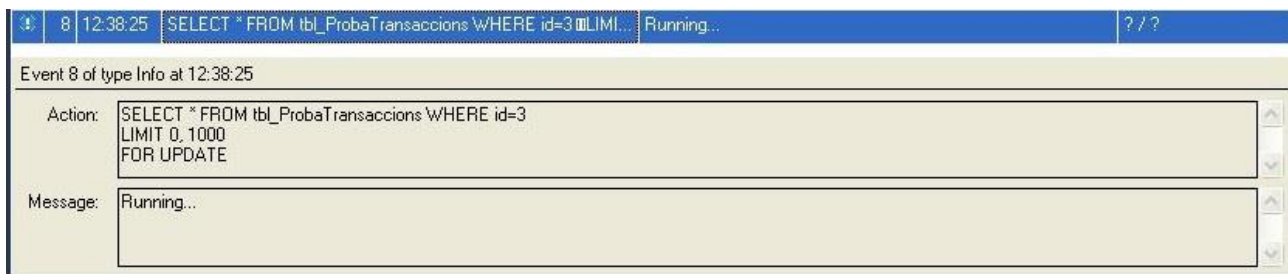
```
/* Faranse efectivos os cambios e liberarase a táboa */
COMMIT;
```

O código empregado polo usuario *usuario2* é o seguinte:

```
/* Comprobación do usuario actual que é 'usuario2' */
SELECT USER();
/* Activación da base de datos polo usuario2 */
USE `bd_ProbaTransaccions`;
/* Comprobación do estado da variable AUTOCOMMIT que debe ser ON (1) */
SHOW VARIABLES LIKE 'autocommit';
/* Comezamos a transacción */
START TRANSACTION;
```

```
/* Crearase un bloqueo de escritura sobre as filas a modificar, pero non se nos permite
xa que se atopan bloqueada para escritura con anterioridade */
```

```
SELECT * FROM tbl_ProbaTransaccions WHERE id=3 FOR UPDATE;
```



```
/* Unha vez desbloqueado o rexistro a modificación lévase a cabo aínda que non será
efectiva ata executar COMMIT */
```

```
UPDATE tbl_ProbaTransaccions
```

```
SET s = "escritura por usuario2"
```


```
WHERE id=3;
```


```
COMMIT;
```

```
/* Móstrase o valor actual de s */
```

```
SELECT * FROM tbl_ProbaTransaccions WHERE id=3;
```

	id	s	si
▶	3	escritura por usuario2	NULL

 Tarefa 2: Empregar transaccións.

 Tarefa 3. Empregar transaccións implícitas.

 Tarefa 4. Executar transaccións con diferentes niveis de bloqueo.

 Tarefa 5. Inserir datos mediante transaccións.

 Tarefa 6. Validar transaccións.

Niveis de illamento

Os niveis de illamento fan referencia ao grao de actualización dos datos que lemos dunha base de datos.

Para establecer o nivel desexado de illamento das nosas transaccións indicando se é a nivel global ou só para a nosa sesión actual, empregase o comando SET TRANSACTION.

A sintaxe é:

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

Distínguense 4 niveis:

- REPEATABLE READ

É o valor por defecto nas táboas *InnoDB*. Neste caso obtense o valor establecido ao comezo da transacción. As sentenzas `SELECT ...FOR UPDATE`, `SELECT...SHAREMODE`, `SELECT` e `DELETE` que utilicen un índice único cunha condición de busca única bloquean soamente o índice atopado e non o espazo baleiro que o precede.

Con outras opcións de busca, estas operacións empregan bloqueo de clave seguinte, bloqueando o rango de índice cuberto pola operación incluíndo os espazos baleiros, e bloqueando as novas insercións por parte de outros usuarios.

En lecturas consistentes (varios `SELECT`), existe unha importante diferenza con respecto ao nivel de illamento anterior: neste nivel, todas as lecturas consistentes dentro da mesma transacción len da captura da BD tomada pola primeira lectura. Esta práctica significa que se se emiten varias instrucións `SELECT` dentro da mesma transacción, estas serán consistentes unhas con outras.

É dicir, aínda que durante a transacción os valores cambien, se terá en conta sempre os do comezo.

No exemplo seguinte móstrase este funcionamento:

Código a executar polo usuario *usuario1*

```
/* Comprobación do usuario actual que é 'usuario1' */
SELECT USER();
/* Activación da base de datos */
USE `bd_ProbaTransaccions`;
/* BLOQUE 1: A executar por 'usuario1' */
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
START TRANSACTION;
SET @cantidad1= 100;
SET @valor1= 0;
SET @valor1= (SELECT si FROM tbl_ProbaTransaccions WHERE id = 4 LOCK IN SHARE MODE);
SELECT @valor1;
```

	@valor1
▶	0

```
SET @valor1 = @valor1 + @cantidad1;
SELECT @valor1;
```

	@valor1
▶	100

```
/* BLOQUE 3: A executar por 'usuario1'*/
UPDATE tbl_ProbaTransaccions SET si = @valor1 WHERE id = 4;
SELECT si FROM tbl_ProbaTransaccions WHERE id = 4;
```

	si
▶	100

```
/* BLOQUE 5: A executar por 'usuario1' */
SELECT si FROM tbl_ProbaTransaccions WHERE id=4;
COMMIT;
```

Código a executar polo usuario *usuario2*:

```
/* Comprobación do usuario actual que é 'usuario2' */
SELECT USER();
```

```

/* Activación da base de datos */
/* BLOQUE2: A executar por usuario2 */
USE `bd_ProbaTransaccions`;
SET @cantidade2= 500;
SET @valor2 =0;
SET @valor2 = (SELECT si FROM tbl_ProbaTransaccions WHERE id = 4);
/* Permítese o acceso de lectura á información bloqueada pero có valor anterior á modi-
ficación */

```

	@valor2
▶	0

```

SET @valor2 = @valor2 - @cantidade2;
SELECT @valor2;

```

	@valor2
▶	-500

```

/* Antes deste bloque débese executar o bloque 3 do usuario1 */
/* BLOQUE 4: A executar por usuario2 */
UPDATE tbl_ProbaTransaccions SET si = @valor2 WHERE id = 4;
/* A información atópase bloqueada polo que ata que usuario1 non execute un COMMIT nin-
gunha transacción poderá modificar a información */

```

19	17:38:27	UPDATE tbl_ProbaTransaccions SET si = @valor2 WHERE id = 4	Running...	?
Event 19 of type Info at 17:38:27				
Action:	UPDATE tbl_ProbaTransaccions SET si = @valor2 WHERE id = 4			
Message:	Running...			

```

/*BLOQUE 6: A executar por usuario2 */
SELECT si FROM tbl_ProbaTransaccions WHERE id=4;
COMMIT;

```

■ SERIALIZABLE

É como a anterior coa diferenza de que agora de maneira interna convértense os SELECT en SELECT ...LOCK IN SHARE MODE se AUTOCOMMIT está desactivado.

Seguindo co mesmo exemplo, o código a executar polo usuario *usuario1*:

```

/* Comprobación do usuario actual que é 'usuario1' */
SELECT USER();
/* Activación da base de datos */
USE `bd_ProbaTransaccions`;
/* BLOQUE 1: A executar por usuario1 */
SET AUTOCOMMIT =0;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE ;
START TRANSACTION;
SET @cantidade1= 100;
SET @valor1= 0;
SET @valor1= (SELECT si FROM tbl_ProbaTransaccions WHERE id = 4 );
SELECT @valor1;

```

	@valor1
▶	0

```
SET @valor1 = @valor1 + @cantidad1;
SELECT @valor1;
```

	@valor1
▶	100

```
/* Executarase o BLOQUE2 do usuario2 antes do bloque 3.
BLOQUE 3: A executar por usuario1 */
UPDATE tbl_ProbaTransaccions SET si = @valor1 WHERE id = 4;
SELECT si FROM tbl_ProbaTransaccions WHERE id = 4;
/* Ao terminar bloqueo a mesma transición pode acceder á información bloqueada*/
```

	si
▶	100

```
/* Executar previamente o Bloque4 do usuario2
BLOQUE 5: A executar por usuario1*/
SELECT si FROM tbl_ProbaTransaccions WHERE id=4;
COMMIT;
```

Código a executar polo usuario *usuario2*:

```
/* Comprobación do usuario actual que é 'usuario2' */
SELECT USER();
/* Activación da base de datos */
/* BLOQUE2: A executar usuario2 */
USE `bd_ProbaTransaccions`;
SET @cantidade2= 500;
SET @valor2 =0;
SET @valor2 = (SELECT si FROM tbl_ProbaTransaccions WHERE id = 4);
/* Permítese o acceso de lectura á información bloqueada pero có valor anterior á modificación*/
```

	@valor2
▶	0

```
SET @valor2 = @valor2 - @cantidade2;
SELECT @valor2;
```

	@valor2
▶	-500

```
/*Antes deste bloque vólvese a executar o bloque 3 do usuario1
BLOQUE 4: a executar por usuario2 */
UPDATE tbl_ProbaTransaccions SET si = @valor2 WHERE id = 4;
/* A información atópase bloqueada polo que ata que o usuario1 non execute un COMMIT
ningunha transacción poderá modificar a información */
```

19	17:38:27	UPDATE tbl_ProbaTransaccions SET si = @valor2 WHERE...	Running...	?
Event 19 of type Info at 17:38:27				
Action:	UPDATE tbl_ProbaTransaccions SET si = @valor2 WHERE id = 4			
Message:	Running...			

```
/* BLOQUE 6: A executar por usuario2 */
SELECT si FROM tbl_ProbaTransaccions WHERE id=4;
COMMIT;
```

■ READ COMMITTED

Todas as sentenzas SELECT ... FOR UPDATE e SELECT ... LOCK IN SHARE MODE bloquean soamente os rexistros de índice, non os espazos baleiros que os preceden, por tanto permítese a libre inserción de novos rexistros xunto aos bloqueados.

As sentenzas UPDATE and DELETE que empreguen un índice único cunha condición de procura única bloquean soamente o rexistro de índice achado, non o espazo que o precede. Nas sentenzas UPDATE e DELETE que actúan sobre rangos de rexistros, InnoDB debe bloquear os espazos baleiros e bloquear as insercións doutros usuarios nos espazos baleiros que hai dentro do rango. Isto é necesario debido a que as filas fantasma deben ser bloqueadas para que funcionen a replicación e recuperación en MySQL.

A lectura consistente mesmo dentro da mesma transacción, establece e le a súa propia captura tomada da base de datos.

No seguinte exemplo ilústrase este comportamento, código a executar polo usuario *usuario1*:

```
/* Comprobación do usuario actual que é 'usuario1' */
SELECT USER();
/* Activación da base de datos */
USE `bd_ProbaTransaccions`;
/*BLOQUE 1: A executar por usuario1 */
SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
START TRANSACTION;
SET @cantidad1= 100;
SET @valor1= 0;
SET @valor1= (SELECT si FROM tbl_ProbaTransaccions WHERE id = 4 LOCK IN SHARE MODE);
SELECT @valor1;
SET @valor1 = @valor1 + @cantidad1;
SELECT @valor1;
/* BLOQUE 3: A executar por usuario1 */
UPDATE tbl_ProbaTransaccions SET si = @valor1 WHERE id = 4; /*si pode modificar*/
SELECT si FROM tbl_ProbaTransaccions WHERE id = 4; /*Si pode ler*/
/* BLOQUE 5: A executar por usuario1 */
SELECT si FROM tbl_ProbaTransaccions WHERE id=4;
COMMIT;
```

Código a executar polo usuario *usuario2*

```
/* Comprobación do usuario actual que é 'usuario2' */
SELECT USER();
/* Activación da base de datos */
USE `bd_ProbaTransaccions`;
/* BLOQUE 2: A executar por usuario2 */
SET @cantidade2= 500;
SET @valor2 =0;
SET @valor2 = (SELECT si FROM tbl_ProbaTransaccions WHERE id = 4 LOCK IN SHARE MODE);
SET @valor2 = @valor2 - @cantidade2;
SELECT @valor2;
/* BLOQUE 4: A executar por cliente2 */
SELECT si FROM tbl_ProbaTransaccions WHERE id = 4;
UPDATE tbl_ProbaTransaccions SET si = @valor2 WHERE id = 4;
```

```
/* BLOQUE 6: A executar por cliente 1 */  
SELECT si FROM tbl_ProbaTransaccions WHERE id=4;  
COMMIT;
```

- **READ UNCOMMITTED**

As lecturas realízanse sen bloqueo de maneira que podemos estar a ler un dato que xa foi modificado por outra transacción. É o que se denomina lectura inconsistente ou sucia.



Tarefa 7. Establecer opcións de bloqueo.

Insercións concorrentes

Son insercións que permiten a lectura simultánea de datos dunha táboa. É dicir, mentres nunha sesión se está modificando unha táboa, outras sesións poden ler de devandita táboa.

O motor de MySQL MyISAM soporta insercións concorrentes para facilitar a competencia de lectura e escritura sobre este tipo de táboas. Isto dáse normalmente cando a táboa non contén ocos nos seus ficheiros de datos, é dicir, non hai filas borradas entre o resto de filas.

Este comportamento está controlado por unha variable de sistema chamada *concurrent insert* na que os valores posibles son os seguintes:

- AUTO ou 1: nese caso actívanse as insercións concorrentes.
- 0: nese caso impídense as insercións concorrentes.
- 2: permítense as insercións aínda que haxa filas borradas entre os datos da táboa.

A variable *concurrent insert*, como o resto de variables de sistema, pódese consultar e modificar cos comandos SHOW VARIABLES e SET respectivamente.

Para o caso de que se habiliten este tipo de insercións non se fai necesario o uso da cláusula DELAYED do comando INSERT.

A cláusula HIGH PRIORITY vista nas políticas de bloqueo de táboas tamén desactiva o uso de insercións concorrentes.

Para bloqueos de táboas para lectura coa cláusula LOCAL permítense o uso de insercións concorrentes que se executan mentres o bloqueo está activo.

1.3 Tarefas

As tarefas propostas son as seguintes:

- Tarefa 1. Rematar unha transacción.
- Tarefa 2. Empregar transaccións
- Tarefa 3. Empregar transaccións implícitas
- Tarefa 4. Executar transaccións con diferentes niveis de bloqueo
- Tarefa 5. Inserir datos mediante transaccións
- Tarefa 6. Validar transaccións
- Tarefa 7. Establecer opcións de bloqueo.

1.3.1 Tarefa 1. Rematar unha transacción.

Se iniciamos unha transacción e de súpeto rematamos a sesión, qué operación fai o servidor: un COMMIT, un ROLLBACK ou indefinido?

Solución

Coa variable AUTOCOMMIT co seu valor por defecto (activada) o sistema xestor actúa coma se executara un ROLLBACK (non facendo permanentes os cambios).

1.3.2 Tarefa 2. Empregar transaccións

Que diferenza existe entre START TRANSACTION e BEGIN?

Solución

Con START TRANSACTION, *autocommit* permanece non habilitado até o final da transacción con COMMIT ou ROLLBACK. O modo *autocommit* volve ao seu estado previo.

BEGIN e BEGIN WORK sopórtanse como alias para START TRANSACTION para iniciar unha transacción. A sentenza START TRANSACTION corresponde á sintaxe SQL estándar e é a forma recomendada para iniciar unha transacción *ad-hoc*. O comando BEGIN difire do uso da palabra clave BEGIN que comeza un comando composto BEGIN ... END.

1.3.3 Tarefa 3 . Empregar transaccións implícitas

Qué acontece se na metade dunha transacción sen confirmar modificamos a variable *autocommit* dándolle o valor 1?

Solución

Ao executar BEGIN ou START TRANSACTION o sistema compórtase coma se a variable *autocommit* tomara o valor 0, aínda que a efectos prácticos ignora o seu contido polo que almacena o valor 1 de xeito manual non tendo ningún efecto.

```
START TRANSACTION;
INSERT INTO tbl_ProbaTransaccions ( id, s, si) VALUES (7, "setimo", NULL);
SET AUTOCOMMIT=1;
SELECT * FROM tbl_ProbaTransaccions;
```

	id	s	si
▶	1	primeiro	NULL
	2	segundo	NULL
	3	terceiro	NULL
	4	cuarto	NULL
	5	quinto	NULL
	6	sexto	NULL
	7	setimo	NULL

```
SHOW VARIABLES LIKE 'AUTOCOMMIT';
```

	Variable_name	Value
▶	autocommit	ON

```
ROLLBACK;
SELECT * FROM tbl_ProbaTransaccions;
```

	id	s	si
▶	1	primeiro	NULL
	2	segundo	NULL
	3	terceiro	NULL
	4	cuarto	NULL
	5	quinto	NULL
	6	sexto	NULL

1.3.4 Tarefa 4. Executar transaccións con diferentes niveis de bloqueo

Continuando coa base de datos *bd_ProbaTransaccions* creada e empregada ao longo da actividade, execute o seguinte código e explique que é o que acontece:

```
/* Activación da base de datos. */
USE `bd_ProbaTransaccions`;
SET AUTOCOMMIT=1;
INSERT INTO tbl_ProbaTransaccions ( id, s, si) VALUES (7, "¿conformouse este dato?",
NULL);
SET AUTOCOMMIT=0;
CREATE TABLE tbl_ProbaTransaccions2 (id INT NOT NULL PRIMARY KEY, s2 VARCHAR (50))
ENGINE = InnoDB ;
INSERT INTO tbl_ProbaTransaccions2 ( id, s2) VALUES (1, "¿existe
tbl_ProbaTransaccions2?");
ROLLBACK;
SELECT * FROM tbl_ProbaTransaccions;
SELECT * FROM tbl_ProbaTransaccions2;
```

Solución:

Neste exemplo ao por de xeito manual a variable *autocommit* a un, o INSERT na táboa *tbl_ProbaTransaccions* confirmárase de maneira automática.

	id	s	si
▶	1	primeiro	NULL
	2	segundo	NULL
	3	terceiro	NULL
	4	cuarto	NULL
	5	quinto	NULL
	6	sexto	NULL
	7	¿conformouse este dato?	NULL

A seguinte instrución inhabilita a variable *autocommit* polo que a transacción permanecerá aberta ata o ROLLBACK, desfecendo as sentenzas CREATE e o INSERT.

Pero ao executarse os SELECT obsérvase que a táboa *tbl_ProbaTransaccions2* foi creada, isto se debe a que certas instrucións (como CREATE) se confirman automaticamente

e de forma independente ao contido da variable *autocommit*.

id	s2
----	----

1.3.5 Tarefa 5. Inserir datos mediante transaccións

Continuando coa base de datos *bd_ProbaTransaccions* creada e empregada ao longo da actividade, execute o seguinte código e explique que é o que acontece:

```
/* Activación da base de datos. */  
USE `bd_ProbaTransaccions`;  
SET AUTOCOMMIT=0;  
INSERT INTO tbl_ProbaTransaccions ( id, s, si) VALUES (8, "... comeza a proba", NULL);  
INSERT INTO tbl_ProbaTransaccions ( id, s, si) VALUES (9, "... superase o limite do tamaño de numero de caracteres da cadea, prodúcese un error...", NULL);  
COMMIT;  
SELECT * FROM tbl_ProbaTransaccions;
```

Solución:

Neste exemplo ao inhabilitar de xeito manual a variable *autocommit*, todas as instrucións pasan a ser unha única transacción ata atopar un COMMIT (ou un ROLLBACK).

Ao producirse un erro, pódese pensar que de xeito automático farase un ROLLBACK das instrucións xa executadas da transacción, pero MySQL non actúa deste xeito xa que incluírá na táboa a fila có *id*=8 a pesar de que a inserción *id*=9 errou.

	id	s	si
▶	1	primeiro	NULL
	2	segundo	NULL
	3	terceiro	NULL
	4	cuarto	NULL
	5	quinto	NULL
	6	sexto	NULL
	7	¿conformouse este dato?	NULL
	8	... comeza a proba	NULL

1.3.6 Tarefa 6. Validar transaccións

Na base de datos *bd_ProbaTransaccions* creada e empregada ao longo da actividade, execute o seguinte código e explique que é o que acontece:

```
/* Activación da base de datos. */  
USE `bd_ProbaTransaccions`;  
SET AUTOCOMMIT=0;  
INSERT INTO tbl_ProbaTransaccions ( id, s, si) VALUES (9, "... comeza outra proba", NULL);  
INSERT INTO tbl_ProbaTransaccions ( id, s, si) VALUES (10, "... superase o limite dbd_probatransaccions.comprobarErro tamaño de numero de caracteres da cadea, prodúcese un erro...", NULL);  
call comprobarErro();  
SELECT * FROM tbl_ProbaTransaccions;
```

O código do procedemento almacenado é o seguinte:


```

DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `comprobarErro`()
begin
if @@error_count > 0 OR ROW_COUNT() = 0 then ROLLBACK;
else COMMIT;
end if;
end

```

Solución

Neste exemplo conséguese, grazas ao procedemento *comprobarErro*, que cando se produza un erro non se insira unha fila e se desfaga toda a transacción.

	id	s	si
▶	1	primeiro	NULL
	2	segundo	NULL
	3	terceiro	NULL
	4	cuarto	NULL
	5	quinto	NULL
	6	sexto	NULL
	7	¿conformouse este dato?	NULL
	8	... comeza a proba	NULL

1.3.7 Tarefa 7. Establecer opcións de illamento.

Dado o seguinte código, explique que acontece coas transaccións.

Executar o seguinte código como administrador:

```

CREATE USER 'caixeiro1'@'localhost' IDENTIFIED BY 'caixeiro1';
CREATE USER 'caixeiro2'@'localhost' IDENTIFIED BY 'caixeiro2';
SELECT * FROM mysql.user;
CREATE DATABASE `bd_Banco`;
USE `bd_Banco`;
CREATE TABLE tbl_ContaCorrente (id INT NOT NULL PRIMARY KEY, nome VARCHAR (30), apelidos VARCHAR(80), saldo FLOAT, movemento VARCHAR(30)) ENGINE = InnoDB ;
INSERT INTO tbl_ContaCorrente ( id, nome, apelidos, saldo, movemento) VALUES (1, "xurxo", "perez laxe", 1000, NULL);
INSERT INTO tbl_ContaCorrente ( id, nome, apelidos, saldo, movemento) VALUES (2, "uxia", "novoa quintana", 4000, NULL);
INSERT INTO tbl_ContaCorrente ( id, nome, apelidos, saldo, movemento) VALUES (3, "xenxo", "figueiro sotelo", 8000, NULL);
SELECT * FROM tbl_ContaCorrente;
GRANT ALL ON bd_ProbaTransaccions.* TO 'caixeiro1'@'localhost';
GRANT ALL ON bd_ProbaTransaccions.* TO 'cixeiro2'@'localhost';

```

Executar na orde establecida polo número de bloque o código dos usuarios *caixeiro1* e *caixeiro2*:

```

/* Código correspondente a caixeiro1 */
USE `bd_Banco`;
SET AUTOCOMMIT =0;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
SET @novoMovementoCaixeiro1= 100;
SET @balanceActualCaixeiro1= 0;
SET @balanceActualCaixeiro1= (SELECT saldo FROM tbl_contacorrente WHERE id = 2 );
SELECT @balanceActualCaixeiro1;

```

```

SET @balanceActualCaixeiro1 = @balanceActualCaixeiro1 + @novoMovementoCaixeiro1;
SELECT @balanceActualCaixeiro1;
/*BLOQUE 3 */
UPDATE tbl_ContaCorrente SET saldo = @balanceActualCaixeiro1 WHERE id = 2;
SELECT saldo FROM tbl_ContaCorrente WHERE id = 2;
/*BLOQUE 5 */
SELECT saldo FROM tbl_ContaCorrente WHERE id = 2;
COMMIT;
SELECT saldo FROM tbl_ContaCorrente WHERE id = 2;

/* Código correspondente a caixeiro2 */
/* BLOQUE 2 */
USE `bd_Banco`;
SET AUTOCOMMIT =0;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET @novoMovementoCaixeiro2= -200;
SET @balanceActualCaixeiro2= 0;
SET @balanceActualCaixeiro2= (SELECT saldo FROM tbl_contacorrente WHERE id = 2 );
SELECT @balanceActualCaixeiro2;
SET @balanceActualCaixeiro2 = @balanceActualCaixeiro2 + @novoMovementoCaixeiro2;
SELECT @balanceActualCaixeiro2;
/* BLOQUE 4 */
UPDATE tbl_ContaCorrente SET saldo = @balanceActualCaixeiro2 WHERE id = 2;
SELECT saldo FROM tbl_ContaCorrente WHERE id = 2;
/* BLOQUE 6 */
SELECT saldo FROM tbl_ContaCorrente WHERE id = 2;
COMMIT;
SELECT saldo FROM tbl_ContaCorrente WHERE id = 2;

```

Solución

Mediante a execución do código como administrador, xerárase a estrutura da base de datos:

```

/* Creranse os usuarios */
CREATE USER 'caixeiro1'@'localhost' IDENTIFIED BY 'caixeiro1';
CREATE USER 'caixeiro2'@'localhost' IDENTIFIED BY 'caixeiro2';
SELECT * FROM mysql.user;
/* Crearase a base de datos */
CREATE DATABASE `bd_Banco`;
USE `bd_Banco`;
/* Crease unha táboa para os clientes do banco */
CREATE TABLE tbl_ContaCorrente (id INT NOT NULL PRIMARY KEY, nome VARCHAR (30), apelidos VARCHAR(80), saldo FLOAT, movemento VARCHAR(30)) ENGINE = InnoDB ;
/* Insírese información dos clientes */
INSERT INTO tbl_ContaCorrente ( id, nome, apelidos, saldo, movemento) VALUES (1, "xurxo", "perez laxe", 1000, NULL);
INSERT INTO tbl_ContaCorrente ( id, nome, apelidos, saldo, movemento) VALUES (2, "uxia", "nova quintana", 4000, NULL);
INSERT INTO tbl_ContaCorrente ( id, nome, apelidos, saldo, movemento) VALUES (3, "xenxo", "figueiro sotelo", 8000, NULL);
SELECT * FROM tbl_ContaCorrente;
/* Asignanase permisos aos caixeiros */
GRANT ALL ON bd_ProbaTransaccions.* TO 'caixeiro1'@'localhost';
GRANT ALL ON bd_ProbaTransaccions.* TO 'cixeiro2'@'localhost';

```

	id	nome	apelidos	saldo	movemento
▶	1	xurxo	perez laxe	1000	NULL
	2	uxia	nova quintana	4000	NULL
	3	xenxo	figueiro sotelo	8000	NULL

Ao executar os códigos de *caixeiro1* e *caixeiro2* segundo a orde establecida no enunciado:

```
/* Código correspondente a caixeiro1 */
/* Activación da base de datos */
USE `bd_Banco`;
/* BLOQUE 1 */
SET AUTOCOMMIT =0; /* Ata o COMMIT existe unha única transacción */
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE ; /* Declaración como serializable da transacción */
SET @novoMovementoCaixeiro1= 100; /* Realizarase un ingreso de 100 */
SET @balanceActualCaixeiro1= 0; /* Inicialización da variable */
/* Recupérase o saldo da conta para o cliente con identificador2 */
SET @balanceActualCaixeiro1= (SELECT saldo FROM tbl_contacorrente WHERE id = 2 );
SELECT @balanceActualCaixeiro1;
```

@balanceActualCaixeiro1
4000

```
/* Actualízase o valor sobre a variable */
SET @balanceActualCaixeiro1 = @balanceActualCaixeiro1 + @novoMovementoCaixeiro1;
SELECT @balanceActualCaixeiro1;
```

@balanceActualCaixeiro1
4100

Overview Output Snippets caixeiro1.sql Result				
Show output from:	Actions		Clear	
	Time	Action	Message	Duration / Fetch
1	19:41:29	USE `bd_Banco`	0 row(s) affected	0.000 sec
2	19:41:32	SET AUTOCOMMIT =0	0 row(s) affected	0.000 sec
3	19:41:35	SET TRANSACTION ISOLATION LEVEL SERIALIZABLE	0 row(s) affected	0.000 sec
4	19:41:39	SET @novoMovementoCaixeiro1= 100	0 row(s) affected	0.000 sec
5	19:41:42	SET @balanceActualCaixeiro1= 0	0 row(s) affected	0.000 sec
6	19:41:48	SET @balanceActualCaixeiro1= (SELECT saldo FROM tbl_c...	0 row(s) affected	0.000 sec
7	19:41:52	SELECT @balanceActualCaixeiro1LIMIT 0,1000	1 row(s) returned	0.000 sec / 0.090 sec
8	19:42:04	SET @balanceActualCaixeiro1 = @balanceActualCaixeiro1 +...	0 row(s) affected	0.090 sec
9	19:42:10	SELECT @balanceActualCaixeiro1LIMIT 0,1000	1 row(s) returned	0.010 sec / 0.030 sec

```
/* BLOQUE 3 */
/* Actualízase a táboa */
/* Caixeiro2 tamén bloqueou o contido deste campo polo que NON permite as actualiza-  
cións */
UPDATE tbl_ContaCorrente SET saldo = @balanceActualCaixeiro1 WHERE id = 2;
```

Overview Output Snippets				
Show output from:		Actions		Clear
	Time	Action	Message	Duration / Fetch
1	19:41:29	USE `bd_Banco`	0 row(s) affected	0.000 sec
2	19:41:32	SET AUTOCOMMIT =0	0 row(s) affected	0.000 sec
3	19:41:35	SET TRANSACTION ISOLATION LEVEL SERIALIZABLE	0 row(s) affected	0.000 sec
4	19:41:39	SET @novoMovementoCaixeiro1= 100	0 row(s) affected	0.000 sec
5	19:41:42	SET @balanceActualCaixeiro1= 0	0 row(s) affected	0.000 sec
6	19:41:48	SET @balanceActualCaixeiro1= (SELECT saldo FROM tbl_c...	0 row(s) affected	0.000 sec
7	19:41:52	SELECT @balanceActualCaixeiro1LIMIT 0,1000	1 row(s) returned	0.000 sec / 0.090 sec
8	19:42:04	SET @balanceActualCaixeiro1 = @balanceActualCaixeiro1 +...	0 row(s) affected	0.090 sec
9	19:42:10	SELECT @balanceActualCaixeiro1LIMIT 0,1000	1 row(s) returned	0.010 sec / 0.030 sec
10	19:49:22	UPDATE tbl_ContaCorrente SET saldo = @balanceActualCai...	Error Code: 1205. Lock wait timeout exceeded; try restarting tr...	51.383 sec
11	19:50:28	UPDATE tbl_ContaCorrente SET saldo = @balanceActualCai...	Error Code: 1205. Lock wait timeout exceeded; try restarting tr...	50.893 sec

Event 11 of type Error at 19:50:28

Action: UPDATE tbl_ContaCorrente SET saldo = @balanceActualCaixeiro1 WHERE id = 2

Message: Error Code: 1205. Lock wait timeout exceeded; try restarting transaction

```

SELECT saldo FROM tbl_ContaCorrente WHERE id = 2;
/* BLOQUE 5 */
SELECT saldo FROM tbl_ContaCorrente WHERE id = 2;
COMMIT;
SELECT saldo FROM tbl_ContaCorrente WHERE id = 2;
/* Código correspondente a caixeiro2 */
/* Activación da base de datos */
USE `bd_Banco`;
/* BLOQUE 2 */
SET AUTOCOMMIT =0; /* Ata o COMMIT existe unha única transacción */
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE ; /* Declaración como serializable da transacción */
SET @novoMovementoCaixeiro2= -200; /* Realizarase unha extracción de 200 */
SET @balanceActualCaixeiro2= 0; /* Inicialización da variable */
/* Recupérase o saldo da conta para o cliente con identificador2 que neste momento atópase bloqueado polo caixeiro1 */
SET @balanceActualCaixeiro2= (SELECT saldo FROM tbl_contacorrente WHERE id = 2 );
SELECT @balanceActualCaixeiro2;

```

@balanceActualCaixeiro2
4000

```

/*Ao atoparse bloqueada con anterioridade a COMMIT o valor que se recupera non é o último */
/* Actualízase o valor sobre a variable */
SET @balanceActualCaixeiro2 = @balanceActualCaixeiro2 + @novoMovementoCaixeiro2;
SELECT @balanceActualCaixeiro2;

```

@balanceActualCaixeiro2
3800

Overview Output Snippets caixeiro2.sql Result				
Show output from: Actions				Clear
	Time	Action	Message	Duration / Fetch
1	19:45:28	USE 'bd_Banco'	0 row(s) affected	0.070 sec
2	19:45:33	SET AUTOCOMMIT =0	0 row(s) affected	0.000 sec
3	19:45:37	SET TRANSACTION ISOLATION LEVEL SERIALIZABLE	0 row(s) affected	0.020 sec
4	19:45:40	SET @novoMovimentoCaixeiro2= -200	0 row(s) affected	0.040 sec
5	19:45:44	SET @balanceActualCaixeiro2= 0	0 row(s) affected	0.010 sec
6	19:45:49	SET @balanceActualCaixeiro2= (SELECT saldo FROM tbl_c...	0 row(s) affected	0.010 sec
7	19:46:00	SELECT @balanceActualCaixeiro2LIMIT 0, 1000	1 row(s) returned	0.010 sec / 0.030 sec
8	19:47:24	SET @balanceActualCaixeiro2 = @balanceActualCaixeiro2 +...	0 row(s) affected	0.000 sec
9	19:47:28	SELECT @balanceActualCaixeiro2LIMIT 0, 1000	1 row(s) returned	0.060 sec / 0.020 sec

```

/* BLOQUE 4 */
/* Atópase bloqueado para escritura polo caixeiro1 */
UPDATE tbl_ContaCorrente SET saldo = @balanceActualCaixeiro2 WHERE id = 2;

```

Overview Output Snippets				
Show output from: Actions				Clear
	Time	Action	Message	Duration / Fetch
1	19:45:28	USE 'bd_Banco'	0 row(s) affected	0.070 sec
2	19:45:33	SET AUTOCOMMIT =0	0 row(s) affected	0.000 sec
3	19:45:37	SET TRANSACTION ISOLATION LEVEL SERIALIZABLE	0 row(s) affected	0.020 sec
4	19:45:40	SET @novoMovimentoCaixeiro2= -200	0 row(s) affected	0.040 sec
5	19:45:44	SET @balanceActualCaixeiro2= 0	0 row(s) affected	0.010 sec
6	19:45:49	SET @balanceActualCaixeiro2= (SELECT saldo FROM tbl_c...	0 row(s) affected	0.010 sec
7	19:46:00	SELECT @balanceActualCaixeiro2LIMIT 0, 1000	1 row(s) returned	0.010 sec / 0.030 sec
8	19:47:24	SET @balanceActualCaixeiro2 = @balanceActualCaixeiro2 +...	0 row(s) affected	0.000 sec
9	19:47:28	SELECT @balanceActualCaixeiro2LIMIT 0, 1000	1 row(s) returned	0.060 sec / 0.020 sec
10	19:53:47	UPDATE tbl_ContaCorrente SET saldo = @balanceActualCai...	Error Code: 1205. Lock wait timeout exceeded; try restarting tr...	50.753 sec

Event 10 of type Error at 19:53:47

Action: UPDATE tbl_ContaCorrente SET saldo = @balanceActualCaixeiro2 WHERE id = 2

Message: Error Code: 1205. Lock wait timeout exceeded; try restarting transaction

```

SELECT saldo FROM tbl_ContaCorrente WHERE id = 2;
/* BLOQUE 6 */
SELECT saldo FROM tbl_ContaCorrente WHERE id = 2;
COMMIT;
SELECT saldo FROM tbl_ContaCorrente WHERE id = 2;

```