

MYSQL – RECUPERAÇÃO BASEADA EM LOG E CONTROLE DE CONCORRÊNCIA

LUCAS LAMOUNIER GONCALVES DUARTE - 2016012688

KEVIN VIEIRA PEREIRA - 2016015385

RODRIGO APARECIDO SILVA MAIA - 2016013095

Itajubá, 9 de maio de 2018



Sumário

QUESTÃO 1: RESPONDA AS SEGUINTE PERGUNTAS:	2
a) Explique as técnicas para Tratamento de Deadlock.	2
b) O que é bloqueio por granularidade múltipla?	3
c) Como a frequência dos pontos de verificação(checkpoint) afeta:	6
QUESTÃO 2: QUAIS SÃO OS TIPOS DE LOG UTILIZADOS NO SGBD? DETALHE A FUNCIONALIDADE DE CADA UM E DESCREVA TAMBÉM INFORMAÇÕES COMO TAMANHO MÁXIMO, LOCAL DE ARMAZENAMENTO. É POSSÍVEL ALTERA ESSAS CONFIGURAÇÕES? COMO?	6
QUESTÃO 3: VIMOS EM SALA QUE EXISTEM TRÊS ABORDAGENS DE RECUPERAÇÃO DE TRANSAÇÕES APÓS UMA FALHA NO SISTEMA. COMO O SGBD REALIZA ESSA RECUPERAÇÃO? É POSSÍVEL MUDAR?	11
QUESTÃO 4: SOBRE O LOG DE RECUPERAÇÃO, É POSSÍVEL ALTERAR A FREQUÊNCIA COM QUE OS CHECKPOINTS SÃO REALIZADOS? O QUE É A OPERAÇÃO DE FLUSHING?	12
QUESTÃO 5: COMO O SGBD IMPLEMENTA O CONTROLE DE CONCORRÊNCIA?	14
QUESTÃO 6: EM GERAL, OS SGBDS SÃO INSTALADOS COM UM NÍVEL DE ISOLAMENTO PADRÃO, MAS DISPONIBILIZAM UMA VARIÁVEL DE AMBIENTE ONDE É POSSÍVEL CONFIGURAR ESSE NÍVEL DE ISOLAMENTO. ASSIM SENDO, RESPONDA:	15
a) Quais são as consequências de optar por cada um dos níveis de isolamento acima citados?	15
b) Como é feita essa configuração no seu SGBD?	16
QUESTÃO DESFIO 1 (VALENDO NOTA EXTRA): EXECUTAR TRANSAÇÕES COM OS DIFERENTES NÍVEIS DE ISOLAMENTO, EM SITUAÇÕES ONDE AS DIFERENTES TRANSAÇÕES ACESSEM O MESMO DADO. VERIFICAR A OCORRÊNCIA DE ERROS.	17
QUESTÃO DESAFIO 2 (VALENDO NOTA EXTRA): ALTERAR AS CONFIGURAÇÕES DE FLUSH DO SGBD E AVALIAR O TEMPO DE EXECUÇÃO.	20
REFERÊNCIA:	20



QUESTÃO 1: RESPONDA AS SEGUINTE PERGUNTAS:

a) Explique as técnicas para Tratamento de Deadlock.

Os métodos para tratamento de Deadlock são:

Ignorar a situação: A técnica mais simples para tratar um Deadlock é simplesmente ignorá-lo, visto que se um banco é bem projetado as ocorrências de Deadlock serão mínimas e assim não há necessidade de tratamento em todas as raras ocorrências. Esse tratamento pode gerar uma sobrecarga com processamento extra em códigos de tratamento, quando as vezes é tolerável simplesmente reiniciar o banco/sistema com uma ação corretiva.

Deteção e recuperação de Deadlock: Para detectar o deadlock o sistema guarda um grafo de espera, esse grafo deve ser atualizado periodicamente sempre garantindo que não haja nenhum ciclo nele, caso encontre o sistema escolhe uma transação para ser abortada. Para a escolha da transação desfeita é necessário considerar os seguintes itens:

- Por quanto tempo a transação está em processamento e quanto tempo será ainda necessário para que a tarefa seja completada.
- Quantos itens de dados a transação esta usando.
- Quantos itens a transação ainda usará até que se complete.
- Quantas transações serão envolvidas no rollback.

Uma vez escolhida a transação a ser abortada é necessário reverter até um ponto que quebre o deadlock, porém a solução mais simples é um rollback completo, e depois reiniciar a transação.

Prevenção de deadlock: Para que o sistema preveja um deadlock o sistema analisa a transação antes dela iniciar em busca de deadlock, caso ela preveja que a transação poderá gerar um deadlock ela é abortada antes mesmo de iniciar. Para a prevenção de deadlock existem dois mecanismos:



Esperar-morrer: Defende a não-reinicialização. Quando uma transação T1 solicita um dado mantido por T2, T1 espera somente se possuir um timestamp menor que T2 (ou seja, se T1 for mais velha), caso contrário T1 é reiniciada.

Ferir-esperar: Utiliza a técnica de reinicialização. Quando uma transação T1 solicita um dado mantido por T2, T1 poderá esperar somente se possuir um timestamp maior que T2 (ou seja, se T1 for mais nova) caso contrário T1 é desfeita.

Sempre que uma transação é reinicializada o sistema garante que nenhuma transação seja reinicializada continuamente e não consiga continuar seu processamento.

b) O que é bloqueio por granularidade múltipla?

É uma técnica utilizado pelo banco para que caso uma transação necessite acesso a apenas um dado ela possa bloqueá-lo com uma solicitação, e quando necessitar de acesso a todos os dados do banco por exemplo, ela não necessite solicitar bloqueio um a um, ela envia uma única solicitação de bloqueio porém é bloqueado para ela o banco todo.

Para que isso seja possível o banco utiliza a ideia de uma árvore como a da imagem a seguir:

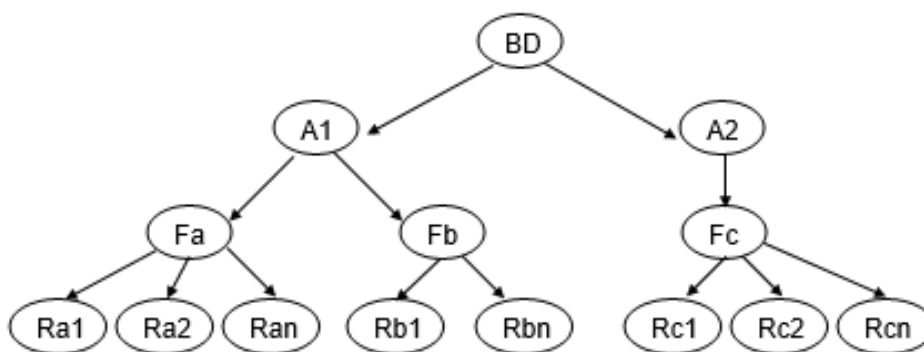


Figura 1 – Árvore de bloqueio



Na árvore temos o nó raiz, que é o banco de dados em si. Logo abaixo temos nós que representam áreas deste mesmo banco. No terceiro nível temos as tabelas. E por último os registros de cada tabela.

Os bloqueios de advertência serão denotados pelo prefixo I (indicando “intenção de”). Por exemplo, IS representa a intenção de obter um bloqueio compartilhado sobre um elemento secundário. As regras do protocolo de advertência são:

- Para impor um bloqueio S ou X sobre um elemento, devemos começar pela raiz da hierarquia.
- Se estamos no elemento que queremos bloquear, não precisamos examinar mais nada. Solicitamos um bloqueio S ou X sobre esse elemento.
- Se o elemento que desejamos bloquear estiver mais abaixo na hierarquia, devemos inserir uma advertência nesse nó; isto é, se quisermos obter um bloqueio compartilhado sobre um elemento secundário, solicitaremos um bloqueio IS nesse nó e, se quisermos um bloqueio exclusivo sobre um elemento secundário, solicitaremos um bloqueio IX sobre esse nó. Quando o bloqueio sobre o nó atual for concedido, prosseguiremos até o final apropriado.

Para decidir se um desses bloqueios pode ou não ser concedido, usamos a matriz de compatibilidade a seguir.

	IS	IX	S	X
IS	Sim	Sim	Sim	Não
IX	Sim	Sim	Não	Não
S	Sim	Não	Sim	Não
X	Não	Não	Não	Não

Análise de cada coluna:



- Coluna IS. Quando solicitamos um bloqueio IS sobre um nó N, pretendemos ler um descendente de N. O único momento em que essa intenção poderia criar um problema é se alguma outra transação já tivesse reivindicado o direito de gravar uma nova cópia do elemento do BD inteiro representado por N; desse modo, vemos “Não” na linha para X. Note que, se alguma outra transação tiver intenção de gravar apenas um elemento secundário, indicado por um bloqueio IX em N, então poderemos ter condições de conceder o bloqueio IS em N e permitir que o conflito seja resolvido em um nível mais baixo, se de fato a intenção de gravação e a intenção de leitura envolverem um elemento comum.
- Coluna IX. Se pretendemos gravar um elemento secundário do nó N, devemos evitar a leitura e gravação do elemento inteiro representado por N. Desse modo, vemos “Não” nas entradas para os modos de bloqueios S e X. Entretanto, de acordo com nossa discussão da coluna IS, outra transação que lê ou grava um elemento secundário pode ter conflitos potenciais nesse nível, e assim IX não entrará em conflito com outro IX em N ou com um IS em N.
- Coluna S. Ler o elemento correspondente ao nó N não pode entrar em conflito com outro bloqueio de leitura sobre N ou com um bloqueio de leitura sobre um elemento secundário de N, representado por IS em N. Desse modo, vemos “Sim” nas linhas S e IS. Porém, um X ou um IX significa que alguma outra transação gravará pelo menos uma parte do elemento representado por N. Portanto, não poderemos conceder o direito de ler N inteiro, o que explica as entradas “Não” na coluna S.
- Coluna X. Só tem entradas “Não”. Não podemos permitir a gravação de todo o nó N, se qualquer outra transação já tiver o direito de ler ou gravar N, ou de adquirir esse direito sobre um elemento secundário.

Resumindo, o protocolo de bloqueio de granularidade múltipla garante a serialização. Cada transação T_i pode bloquear um nó Q, usando as seguintes regras:

1. A função de compatibilidade precisa ser observada;



2. A raiz da árvore precisa ser bloqueada primeiro e pode ser bloqueada em qualquer modo
3. Um nó Q pode ser bloqueado por Ti no modo S ou IS somente se o pai de Q for bloqueado por Ti no modo IX ou IS.
4. Um nó Q pode ser bloqueado por Ti no modo X ou IX somente se o pai de Q for bloqueado por Ti no modo IX.
5. Ti pode bloquear um nó somente se ele não desbloqueou outro nó anteriormente (isto é, Ti tem duas fases).
6. Ti pode desbloquear um nó Q somente se nenhum dos filhos de Q estiver bloqueado por Ti.

c) Como a frequência dos pontos de verificação(checkpoint) afeta:

- A. Quando não há falhas o sistema de checkpoint não afeta em nada o sistema, visto que o checkpoint é utilizado apenas para a recuperação do sistema em caso de algum erro durante a execução de uma transação.
- B. Quando há uma falha o checkpoint minimiza o tempo de recuperação do sistema, visto que ao invés de o sistema verificar todo o log de recuperação para garantir que tudo realmente já foi feito ele corre até o checkpoint e sabe que ele deve recomençar a sua recuperação a partir daquele ponto do log.

QUESTÃO 2: QUAIS SÃO OS TIPOS DE LOG UTILIZADOS NO SGBD? DETALHE A FUNCIONALIDADE DE CADA UM E DESCREVA TAMBÉM INFORMAÇÕES COMO TAMANHO MÁXIMO, LOCAL DE ARMAZENAMENTO. É POSSÍVEL ALTERA ESSAS CONFIGURAÇÕES? COMO?

Considerando a máquina utilizada para a realização deste trabalho, os arquivos de log “general_query log”, “relay log”, “binary log” e “slow query log” foram encontrados no seguinte caminho:



“C:\ProgramData\MySQL\MySQL Server 5.7\Data\mysql”

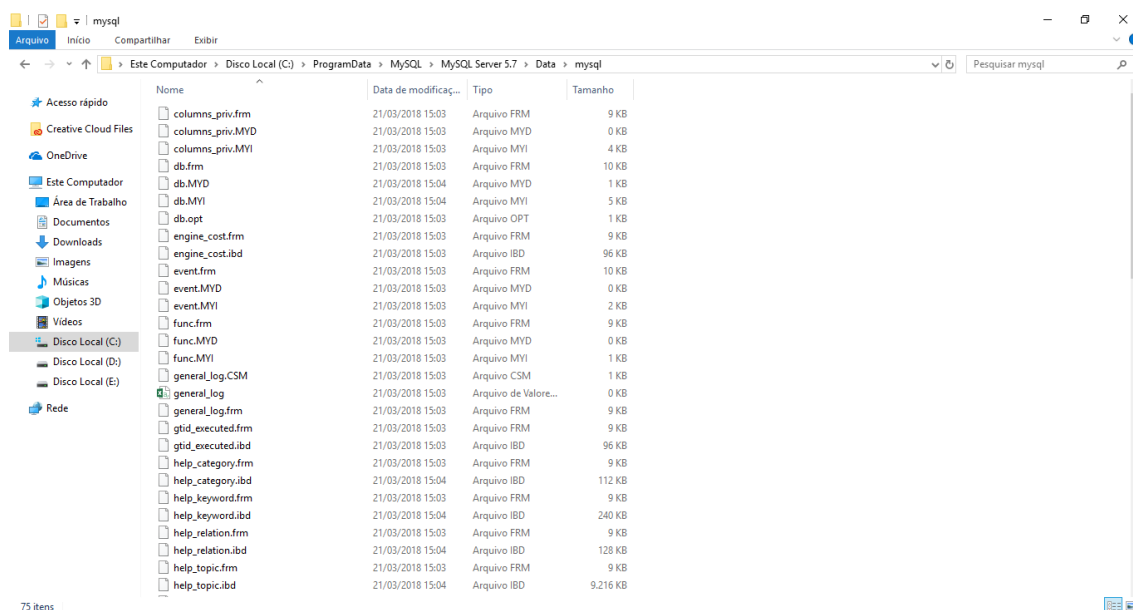


Figura 2 – Diretório dos arquivos de log

Já os arquivos “error log” e “redo log” foram encontrados no seguinte caminho (“undo log” também seria encontrado aqui caso houvesse):

C:\ProgramData\MySQL\MySQL Server 5.7\Data

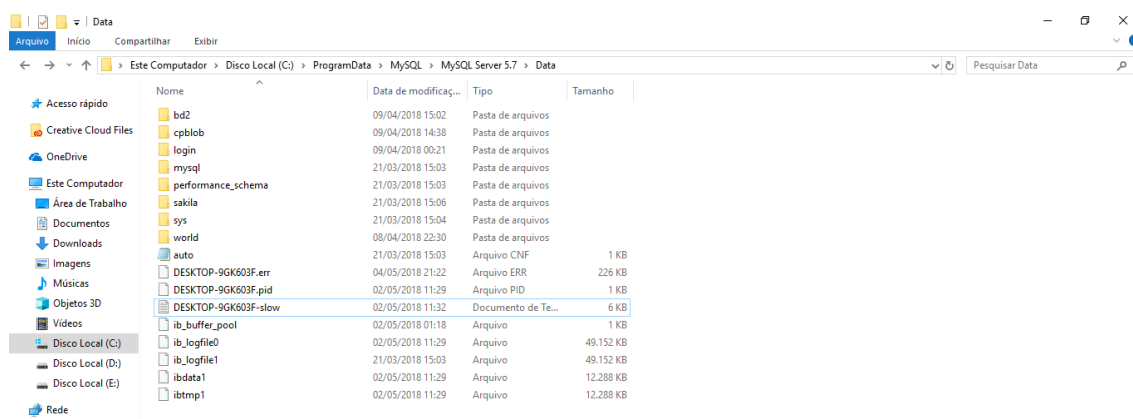


Figura 3 – Diretório dos arquivos de log

OBS: O ddl_log.log não é criado até que seja realmente necessário para gravar instruções de metadados e é removido após um início bem-sucedido do mysql.

A seguir uma tabela com os tipos de log e as informações que cada um contém:



Error log	Problemas encontrados iniciando, executando ou interrompendo o servidor mysql.
General query log	Conexões de clientes estabelecidas e declarações recebidas de clientes.
Binary log	Instruções que alteram dados. (Também usada para replicações)
Relay log	Alterações de dados recebidas de um servidor mestre de replicação.
Slow query log	Consultas que levariam mais do que o tempo definido para consultas longas. Por padrão este tempo no mysql é de 10 milissegundos.
DDL log	Operação de metadados executadas por instruções DDL.
Redo log	Informações auxiliares para a recuperação caso haja um problema durante uma transação. (Mais informações questão 3 e na explicação do log)
Undo log	Informações auxiliares para a recuperação caso haja um problema durante uma transação. (Mais informações questão 3 e na explicação do log)

Error log: O log de erros contém um registro dos tempos de inicialização e desligamento do mysql. Ele também contém mensagens de diagnóstico, como erros, avisos e anotações que ocorrem durante a inicialização do servidor e desligamento, e



enquanto o servidor está em execução. Por exemplo, se o mysql notar que uma tabela precisa ser verificada ou reparada automaticamente, ela grava uma mensagem no log de erros.

General query log: O log de consulta geral é um registro geral do que o mysql está fazendo. O servidor grava informações nesse log quando os clientes se conectam ou desconectam e registra cada instrução SQL recebida de clientes. O log de consulta geral pode ser muito útil quando você suspeita de um erro em um cliente e deseja saber exatamente o que o cliente enviou para o mysql.

O mysqld grava instruções no log de consultas na ordem em que as recebe, o que pode diferir da ordem em que são executadas. Essa ordem de log está em contraste com a do log binário, para as quais as instruções são escritas após serem executadas, mas antes que quaisquer bloqueios sejam liberados. Além disso, o log de consultas pode conter instruções que apenas selecionam dados enquanto essas instruções nunca são gravadas no log binário.

Binary log: O log binário contém "eventos" que descrevem as alterações de banco de dados, como operações de criação de tabelas ou alterações nos dados da tabela. Ele também contém eventos para instruções que potencialmente poderiam ter feito alterações (por exemplo, uma exclusão que correspondeu sem linhas), a menos que o log baseado em linha é usado. O log binário também contém informações sobre quanto tempo cada instrução levou para atualizar os dados. O log binário não é usado para instruções como SELECT ou show que não modificam dados.

Relay log: O log de retransmissão, como o log binário, consiste em um conjunto de arquivos numerados contendo eventos que descrevem alterações de banco de dados e um arquivo de índice que contém os nomes de todos os arquivos de log de retransmissão usados.

Slow query log: O log de consultas lentas consiste em instruções SQL que levou mais tempo do que o tempo mínimo predeterminado para a execução da instrução e exigido uma consulta em pelo menos um número de linhas predefinidos. O tempo para adquirir os bloqueios iniciais não é contabilizado como tempo de execução. O mysql grava uma instrução no log de consultas lenta depois de ter sido executada e depois que



todos os bloqueios foram liberados, então a ordem de log pode ser diferente da ordem de execução.

DDL log: O log de DDL, ou log de metadados, registra operações de metadados geradas por instruções de definição de dados, como `DROP TABLE` e `ALTER TABLE`. O MySQL usa esse log para recuperar de falhas que ocorrem no meio de uma operação de metadados. Ao executar a instrução `DROP TABLE T1, T2`, precisamos garantir que ambos T1 e T2 sejam descartados. Outro exemplo deste tipo de instrução SQL é `ALTER TABLE partição drop T3 P2`, onde devemos ter certeza de que a partição está completamente descartada e que sua definição é removida da lista de partições para a tabela T3.

Um registro de operações de metadados, como os descritos apenas, são gravados no arquivo `ddl_log.log`, no diretório de dados do MySQL. Este é um arquivo binário; Ele não se destina a ser legível e você não deve tentar modificar seu Sumário de forma alguma.

O `ddl_log.log` não é criado até que seja realmente necessário para gravar instruções de metadados e é removido após um início bem-sucedido do mysql. Assim, é possível para este arquivo não estar presente em um servidor MySQL que está funcionando de uma forma completamente normal.

Redo log: O redo log é uma estrutura de dados baseada em disco usada durante a recuperação de falhas para corrigir dados escritos por transações incompletas. Durante as operações normais, o redo log codifica as solicitações para alterar os dados da tabela que resultam de instruções SQL ou chamadas de API de baixo nível. As modificações que não terminaram de atualizar os arquivos de dados antes de um desligamento inesperado são reiniciadas automaticamente durante a inicialização e antes que as conexões sejam aceitas.

Por padrão, o redo log é representado fisicamente no disco como um conjunto de arquivos, denominado `ib_logfile0` e `ib_logfile1`.

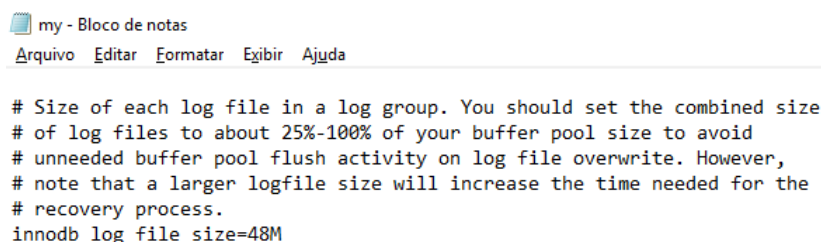
Undo log: Um undo log é uma coleção de registros de undo log associados a uma única transação. Um registro de undo log contém informações sobre como desfazer



a alteração mais recente por uma transação em um registro de índice clusterizado. Se outra transação precisar ver os dados originais (como parte de uma operação de leitura consistente), os dados não modificados são recuperados dos registros de undo log. Os undo log existem em segmentos de undo log, que estão contidos em segmentos de rollback. Por padrão, os segmentos de rollback são fisicamente parte do tablespace do sistema. No entanto, os segmentos de rollback podem residir em espaços separados.

O tamanho máximo de arquivos de log pode ser alterado no arquivo de configuração do mysql.

Para mudar o tamanho máximo dos arquivos de log:



```
my - Bloco de notas
Arquivo  Editar  Formatar  Exibir  Ajuda

# Size of each log file in a log group. You should set the combined size
# of log files to about 25%-100% of your buffer pool size to avoid
# unneeded buffer pool flush activity on log file overwrite. However,
# note that a larger logfile size will increase the time needed for the
# recovery process.
innodb_log_file_size=48M
```

Figura 4 – Configuração de tamanho de log mysql

QUESTÃO 3: VIMOS EM SALA QUE EXISTEM TRÊS ABORDAGENS DE RECUPERAÇÃO DE TRANSAÇÕES APÓS UMA FALHA NO SISTEMA. COMO O SGBD REALIZA ESSA RECUPERAÇÃO? É POSSÍVEL MUDAR?

O sistema de recuperação do MySQL é baseado no Log com modificações adiadas, com a utilização de checkpoints, em que todas as transações que estiverem antes dele já foram passadas para a memória secundária. Quando uma transação foi realizada na memória principal, mas não chegou a atualizar os dados na memória secundária por motivo de falha, então o **Redo Log**, quando o sistema foi reiniciado irá executar as transações que atingiram o estado committed. Já as transações que estão com o estado uncommitted passaram pelo **Undo Log**, para serem desfeitas.

Depois de uma falha, o sistema primeiro irá realizar o **Redo Log**, e depois iniciará a conexão, garantindo assim que as propriedades de Atomicidade e Durabilidade



sejam mantidas intactas mesmo após a falha. Depois disso irá desfazer as transações que não foram completadas, o tempo para desfazer uma transação uncommitted pode ser três ou quatro vezes o tempo para realizar a mesma transação caso ela estivesse no estado ativo.

É possível modificar a abordagem de recuperação através do comando **SET GLOBAL innodb_flush_log_at_trx_commit = 0**, caso seja executado, para cada modificação de dados será atualizado no disco de armazenamento, ou seja, a abordagem de recuperação de Log como modificações adiantadas. Se for utilizado **SET GLOBAL innodb_flush_log_at_trx_commit = 1**, para cada transação concluída será atualizado no disco de armazenamento. Também pode ser usado o comando **SET GLOBAL innodb_flush_log_at_trx_commit = 2**, será armazenado na memória principal, as transações que alcançaram o estado committed, e a cada segundo será realizado a passagem das transações para o disco de armazenamento.

```
mysql> set global innodb_flush_log_at_trx_commit = 0;  
Query OK, 0 rows affected (0.00 sec)  
  
mysql> set global innodb_flush_log_at_trx_commit = 1;  
Query OK, 0 rows affected (0.00 sec)  
  
mysql> set global innodb_flush_log_at_trx_commit = 2;  
Query OK, 0 rows affected (0.00 sec)
```

Figura 5 – que mostra a execução do comando descrito acima

QUESTÃO 4: SOBRE O LOG DE RECUPERAÇÃO, É POSSÍVEL ALTERAR A FREQUÊNCIA COM QUE OS CHECKPOINTS SÃO REALIZADOS? O QUE É A OPERAÇÃO DE FLUSHING?

É possível realizar a alteração da frequência através dos comandos:



- **SET GLOBAL innodb_flush_log_at_trx_commit = 2**, como foi apresentado anteriormente, nessa configuração será realizado um checkpoint a cada segundo as operações que foram feitas;
- **SET GLOBAL innodb_flush_log_at_timeout = X**, sendo que **X** pode variar entre 1 e 2700 segundos, ou seja, realizará um checkpoint a cada intervalo de tempo determinado. Caso haja uma falha poderá haver a perda de dados pela quantidade igual de segundos descritos pelo valor escolhido. Por padrão vem definido como 1. Entende-se que este comando funciona em conjunto com o anterior, visto que o comando anterior declara que será baseado no tempo os checkpoints e esse comando o intervalo para cada um.

```
mysql> SET GLOBAL innodb_flush_log_at_timeout = 2;  
Query OK, 0 rows affected (0.00 sec)
```

Figura 6 – que mostra a execução do comando descrito acima

```
mysql> show variables like 'innodb%flush%log%';  
+-----+-----+  
| Variable_name | Value |  
+-----+-----+  
| innodb_flush_log_at_timeout | 2 |  
| innodb_flush_log_at_trx_commit | 2 |  
+-----+-----+  
2 rows in set (0.01 sec)
```

Figura 7 – mostra que a alteração ocorreu com sucesso

Flushing é a operação de sincronizar os dados que estão contidos na memória principal para o banco de dados, ou seja, é a atualização da base de dados para que ela esteja de acordo com o que se encontra na memória principal, em que está pode estar diferente devido a transações. Pode ocorrer de maneira implícita através de um commit caso a abordagem de modificações adiadas ou após cada modificação caso seja modificação imediata, ou também pode ocorrer de forma explícita através do comando **FLUSH TABLE nomeDaTabela;**



```
mysql> FLUSH TABLE city, country, countrylanguage;  
Query OK, 0 rows affected (0.00 sec)
```

Figura 8 – apresenta a execução do comando Flush, as tabelas city, country e countrylanguage do banco de dados world

QUESTÃO 5: COMO O SGBD IMPLEMENTA O CONTROLE DE CONCORRÊNCIA?

O MySQL implementa o controle de concorrência baseado no protocolo de bloqueio em duas fases, em que se adquire todos os Locks necessários no começo de cada transação e conforme o Lock já foi usado será realizado o Unlock da tabela. Se a transação necessitar aplicar Locks para mais de uma tabela, basta apenas colocar os locks na mesma ordem que a transação utilizará as tabelas.

Os comandos que o MySQL permite para a utilização de locks estão listados abaixo:

Lock Table nomeDaTabela Read - comando do Lock usado para apenas para leitura, múltiplas transações podem acessar a mesma tabela para a operação de leitura.

Lock Table nomeDaTabela Write – comando para se utilizar o lock de escrita (também pode se realizar a leitura com esse lock), apenas uma transação poderá ter esse lock para uma dada tabela por vez(caso a tabela tenha foreign keys, a tabela referenciada também receberá o lock), sendo necessário as outras transações que desejem escrever nessa tabela esperem até o termino dessa transação.

```
mysql> use world;  
Database changed  
mysql> LOCK TABLE city READ, country READ, countrylanguage WRITE;  
Query OK, 0 rows affected (0.00 sec)
```

Figura 9 – mostra a execução dos comandos de lock de escrita e do lock de leitura, para o banco de dados world



Unlock tables - é o comando responsável por liberar as tabelas, que receberam previamente um lock.

```
mysql> UNLOCK TABLES;  
Query OK, 0 rows affected (0.00 sec)
```

Figura 10 – mostra a execução do comando unlock

As transações executadas no MySQL podem sofrer de deadlock, sendo que ele afeta principalmente a performance, mas o InnoDB consegue detectar automaticamente e lidar com eles, aplicando um rollback em uma das transações afetadas. É possível desabilitar o sistema que identifica deadlocks para que se ganhe performance, é recomendado para um sistema com alta concorrência, é possível através do comando SET GLOBAL innodb_deadlock_detect = OFF. Caso seja aplicado o comando anterior, o sistema contará com o innodb_lock_wait_timeout, em que caso se passe 50 segundos e o deadlock continuar, será executado um rollback em uma das transações que estão no deadlock.

O SGBD possui uma política para diminuir a incidência de deadlocks. Primeiro ele classifica as tabelas em uma ordem para ser aplicados os locks. Depois se a tabela for receber um lock de leitura e um de escrita, o lock de escrita será passado à frente do lock de leitura. E por fim será aplicado os locks de cada tabela, até a sessão ter todos os locks necessário.

QUESTÃO 6: EM GERAL, OS SGBDS SÃO INSTALADOS COM UM NÍVEL DE ISOLAMENTO PADRÃO, MAS DISPONIBILIZAM UMA VARIÁVEL DE AMBIENTE ONDE É POSSÍVEL CONFIGURAR ESSE NÍVEL DE ISOLAMENTO. ASSIM SENDO, RESPONDA:

- a) Quais são as consequências de optar por cada um dos níveis de isolamento acima citados?
 - **Repeatable read** – Este nível de isolamento é o padrão da engine InnoDB e garante que a mesma leitura através do SELECT se repita, ou



seja, exibe o mesmo resultado para diferentes execuções em uma mesma transação, dessa forma impede a leitura fantasma e a leitura suja e garante também uma leitura repetitiva. A leitura fantasma nesse nível também é evitada diferindo do padrão SQL.

- **Read Committed** – Este nível de isolamento permite que uma dada transação possa ler e utilizar dados já commitados por outras transações, no entanto não será possível ver dados ainda não commitados. No nível de Read Committed as leituras sujas ainda são evitadas, porém pode ocorrer leituras fantasmas e ela não garante uma leitura repetitiva.
- **Read Uncommitted** – Este nível permite que as transações possam ler e manipular dados não commitados pelas demais transações, permitido que ocorra leituras sujas, leituras fantasmas e leituras não repetitivas.
- **Serializable** – Este nível isola completamente as transações uma das outras evitando a leitura fantasma e a leitura suja e garantido também uma leitura repetitiva.

b) Como é feita essa configuração no seu SGBD?

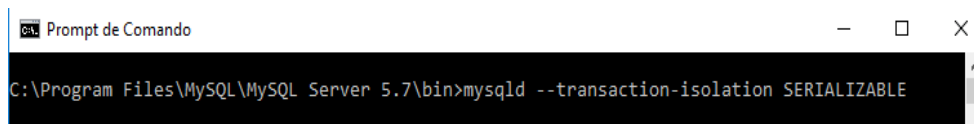
No MySQL o nível de isolamento pode ser configurado através do arquivo “my.ini” ou pelo executável “mysqld”. Para ver o nível que o banco está operando pode ser usado na interface do MySQL o comando “SHOW variables LIKE ‘%isolation%’”

```
mysql> show variables like '%isolation%';
+-----+-----+
| Variable_name | Value          |
+-----+-----+
| transaction_isolation | REPEATABLE-READ |
| tx_isolation      | REPEATABLE-READ |
+-----+-----+
2 rows in set (0.07 sec)
```

Figura 11 – apresenta o nível de isolamento que o banco está operando



Para alterar via “mysqld” o nível de isolamento apenas é necessário executar o comando “mysqld --transaction-isolation SERIALIZABLE”



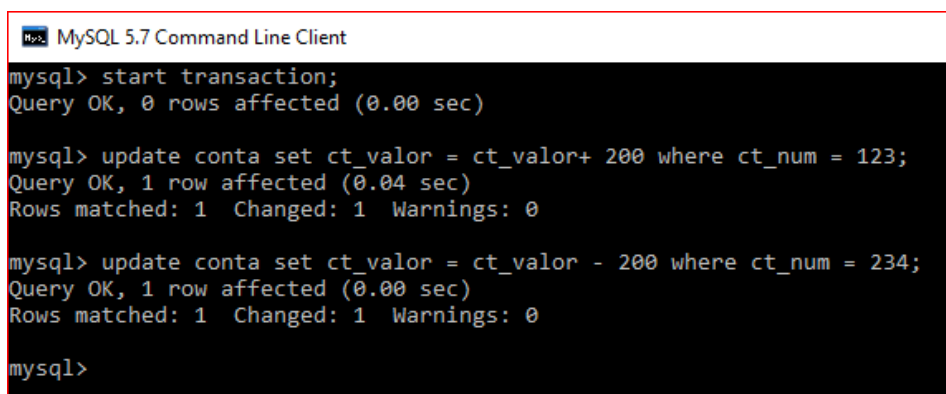
```
Prompt de Comando
C:\Program Files\MySQL\MySQL Server 5.7\bin>mysqld --transaction-isolation SERIALIZABLE
```

Figura 12 – mostra a alteração do nível de isolamento

Pode ser feita alteração adicionando ao arquivo “my.ini” a linha `transaction_isolation = {READ-UNCOMMITTED | READ-COMMITTED | REPEATABLE-READ | SERIALIZABLE}`, caso a linha já exista no arquivo altere apenas o nível de isolamento.

QUESTÃO DESFIO 1 (VALENDO NOTA EXTRA): EXECUTAR TRANSAÇÕES COM OS DIFERENTES NÍVEIS DE ISOLAMENTO, EM SITUAÇÕES ONDE AS DIFERENTES TRANSAÇÕES ACESSEM O MESMO DADO. VERIFICAR A OCORRÊNCIA DE ERROS.

- **SERIALIZABLE** – Ao se executada a primeira transação a segunda não conseguiu ser efetuada com sucesso.



```
MySQL 5.7 Command Line Client
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> update conta set ct_valor = ct_valor+ 200 where ct_num = 123;
Query OK, 1 row affected (0.04 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> update conta set ct_valor = ct_valor - 200 where ct_num = 234;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql>
```

Figura 13 – Transação não commit operando em Serializable.



```
ca. Prompt de Comando - mysql -u userOne -h 25.68.17.37 -p

mysql> start transaction;
Query OK, 0 rows affected (0.09 sec)

mysql> update conta set ct_valor = ct_valor - 100 where ct_num = 432;
Query OK, 1 row affected (0.58 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> update conta set ct_valor = ct_valor + 100 where ct_num = 234;
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
-----
3 rows in set (0.95 sec)
```

Figura 14 – Transação em paralelo operando em Serializable

- **REPEATABLE READ** – Ao executar a primeira a segunda transação não conseguiu ser efetuada.

```
MySQL 5.7 Command Line Client

+-----+-----+-----+
| ct_num | ct_dono      | ct_valor |
+-----+-----+-----+
| 123    | Roberto Chagas | 550      |
| 234    | Daniel Alves  | 300      |
| 432    | Ariel Costantino | 800      |
+-----+-----+-----+
3 rows in set (0.06 sec)

mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> update conta set ct_valor = ct_valor - 500 where ct_num=432;
Query OK, 1 row affected (0.05 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> update conta set ct_valor = ct_valor + 500 where ct_num=234;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

Figura 15 – Transação não commit operando em repeatable read

```
ca. Prompt de Comando - mysql -u userOne -h 25.68.17.37 -p

mysql> start transaction;
Query OK, 0 rows affected (0.09 sec)

mysql> update conta set ct_valor = ct_valor - 100 where ct_num = 432;
Query OK, 1 row affected (0.58 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> update conta set ct_valor = ct_valor + 100 where ct_num = 234;
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
mysql>
```

Figura 16 – Transação em paralelo operando em repeatable read

- **READ UNCOMMITTED** – A o executar a transação foi possível, através de um SELECT executado concorrente a ela, visualizar os dados que ainda não foram



compromissados.

```
MySQL 5.7 Command Line Client

mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> update conta set ct_valor = ct_valor - 100 where ct_num=123;
Query OK, 1 row affected (0.04 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> update conta set ct_valor = ct_valor + 100 where ct_num=432;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

Figura 17 – Transação não commit operando em read uncommitted.

```
MySQL 5.7 Command Line Client

mysql> select * from conta;
+-----+-----+-----+
| ct_num | ct_dono      | ct_valor |
+-----+-----+-----+
| 123    | Roberto Chagas | 450      |
| 234    | Daniel Alves   | 600      |
| 432    | Ariel Costantino | 600      |
+-----+-----+-----+
3 rows in set (0.11 sec)

mysql> _
```

Figura 18 – SELECT em paralelo com a transação operando em read uncommitted.

- **READ COMMITTED** – Ao executar um SELECT em paralelo com a transação não foi possível visualizar os dados ainda não comprometidos.

```
MySQL 5.7 Command Line Client

mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> update conta set ct_valor = ct_valor - 150 where ct_num = 123;
Query OK, 1 row affected (0.34 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> update conta set ct_valor = ct_valor + 150 where ct_num = 234;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

Figura 19 – Transação não commit operando em read committed



```
ca Prompt de Comando - mysql -u userOne -h 25.68.17.37 -p
mysql> select * from conta;
+-----+-----+-----+
| ct_num | ct_dono      | ct_valor |
+-----+-----+-----+
| 123    | Roberto Chagas | 450      |
| 234    | Daniel Alves   | 600      |
| 432    | Ariel Costantino | 600      |
+-----+-----+-----+
3 rows in set (0.11 sec)

mysql> _
```

Figura 20 – SELECT em paralelo com a transação operando em read committed.

QUESTÃO DESAFIO 2 (VALENDO NOTA EXTRA): ALTERAR AS CONFIGURAÇÕES DE FLUSH DO SGBD E AVALIAR O TEMPO DE EXECUÇÃO.

REFERÊNCIA:

ORACLE: **MySQL 5.7 Reference Manual**, 2018. Disponível em: <
<https://dev.mysql.com/doc/refman/5.7/en/innodb-backup-recovery.html> >. Acesso em
05/05/2018.