



GESTÃO DE TRANSAÇÕES

Mestrado em Engenharia Informática

Tecnologias de Bases de Dados (2017/18)

Docente:

Professor Doutor João Muranho

Discentes:

André Monteiro m8465

José Manteigueiro m8776

José Monteiro m8781

Índice

Índice.....	2
Índice de Figuras	4
Índice de Tabelas	4
1. Introdução.....	5
1.1. Objetivos	5
1.2. Descrição do Trabalho Prático	5
2. Transações	7
2.1. Gestão	7
2.2. Controlo	7
2.3. Níveis de Isolamento.....	9
2.3.1. Read Uncommitted	10
2.3.2. Read Committed	10
2.3.3. Repeatable Reads.....	10
2.3.4. Serializable	11
2.4. Recuperação	11
3. Tipos de Locks	13
3.1. Fechos binários	13
3.2. Fechos multi-modo	13
3.2.1. Shared ou Partilhado	14
3.2.2. Exclusive ou Exclusivo	14
3.3. Two-Phase Locking – 2PL	14
4. Testes e Análises de Resultados.....	16
4.1. Efeitos dos Níveis de Isolamento	16
4.1.1. Transações	16

4.1.2. Tabelas de Resultados.....	18
4.2. Testes da Aplicação Desenvolvida	21
4.2.1. Funcionalidade Edit.....	22
4.2.2. Funcionalidade Work	23
4.2.3. Funcionalidade Browser.....	24
5. Conclusão.....	26
6. Bibliografia	27

Índice de Figuras

Figura 1 - Tabelas da Base de Dados.....	6
Figura 2 - Two-Phase Locking.....	15
Figura 3 - Two-Phase Locking Estrito	15
Figura 4 - Barra de Ferramentas da Aplicação	21
Figura 5 - Definições de Conexão da Aplicação	22
Figura 6 - Funcionalidade Edit da Aplicação	23
Figura 7 - Funcionalidade Work da Aplicação	24
Figura 8 - Funcionalidade Browser da Aplicação	25

Índice de Tabelas

Tabela 1 - Efeitos de concorrência que podem ocorrer nos diferentes níveis de isolamento.	11
--	----

1. Introdução

1.1. Objetivos

Este trabalho acerca da gestão de transações, realizado no âmbito da unidade curricular de Tecnologias de Bases de Dados, visa obter uma maior compreensão sobre as transações e o processamento destas num sistema de gestão de bases de dados.

Com a investigação e desenvolvimento deste trabalho, pretende-se que, no final, seja possível perceber o que é o isolamento de dados, os diferentes níveis de isolamento existentes e os efeitos da concorrência na base de dados para cada nível de isolamento.

1.2. Descrição do Trabalho Prático

Este trabalho prático conta com duas vertentes: a base de dados e o programa desenvolvido.

Na base de dados existem três tabelas diferentes: Factura, FactLinha e LogOperations. As tabelas Factura e FactLinha estão associadas através do identificador de Factura. O nome da tabela Factura é autoexplicativo, a tabela FactLinha contém as linhas (p. ex. os bens comprados num supermercado) que constam em tal Factura. A tabela LogOperations incluirá os registos sobre a criação, modificação e eliminação de dados da tabela Factura. A mostra todas as colunas das tabelas referidas anteriormente.

O programa deve visualizar e modificar a base de dados em três instâncias:

- *Edit*: edição individual de um registo da tabela Factura. Através desta funcionalidade deve ser possível estudar o efeito da edição de um registo *uncommitted* adicionado por outra transação, nos casos em que é *committed* ou *rolledback* posteriormente, quer seja editado por um ou vários clientes diferentes;
- *Work*: execução de um número (definido pelo utilizador) de transações aleatórias de *insert*, *update* e *delete* à tabela Factura. Queremos, com este tópico, colocar o sistema de gestão de base de dados a gerir várias transações, provenientes de vários clientes, num

curto espaço de tempo, e estudar os efeitos da concorrência nos casos em que a contenção é baixa ou alta.

- *Browser*: ver os dados nas tabelas Factura e LogOperations a partir da aplicação. Neste ponto deve ser exequível verificar o efeito do *Work* na tabela Factura, através da tabela LogOperations, que mostra os tempos de início e fim das transações.

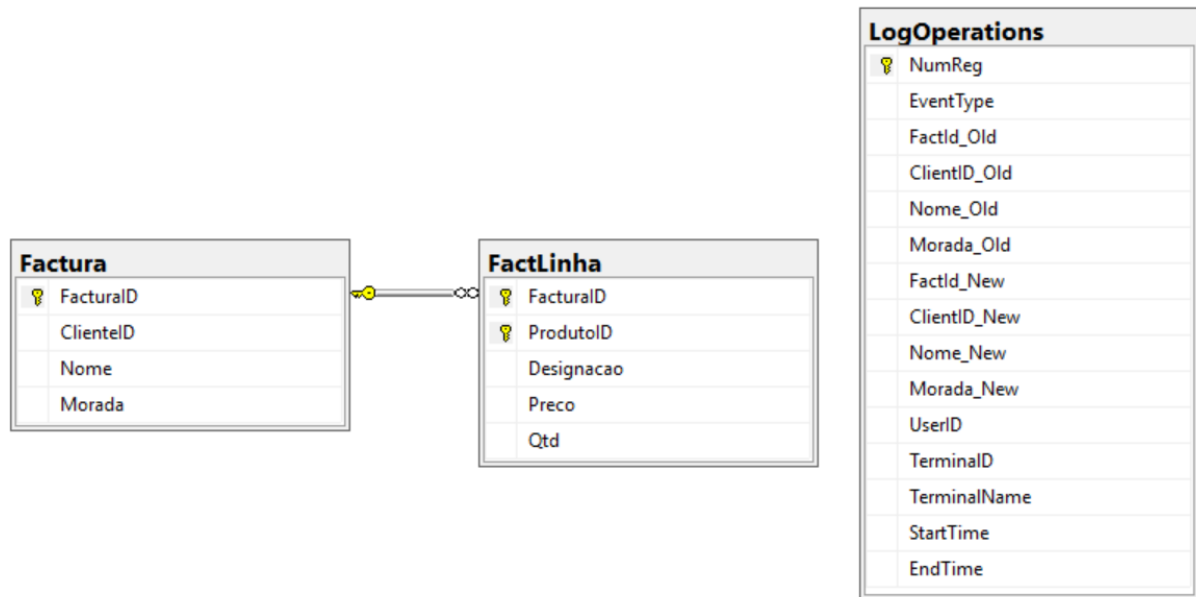


FIGURA 1 - TABELAS DA BASE DE DADOS

2. Transações

2.1. Gestão

A gestão de transações é um fator essencial num sistema de gestão de bases de dados (SGBD). Este deve assegurar as propriedades ACID de modo a preservar a integridade dos dados na base de dados. Estas propriedades são:

- **Atomicidade** - Uma transação é a unidade atômica de processamento que é realizada completamente ou, simplesmente, não é realizada.
- **Consistência** - A execução isolada de uma transação preserva consistência da base de dados.
- **Isolamento** - As modificações feitas por uma transação são invisíveis para outras transações enquanto esta não atinge o estado *committed*.
- **Durabilidade** - Se uma transação altera a base de dados e atinge o estado *committed*, as alterações feitas nunca se perdem mesmo que ocorra uma falha posterior.

O SGBD trata da gestão automaticamente, no entanto há limites. Por exemplo, caso um utilizador inicie uma transação e não faça um *commit* ou *rollback*, poderá estar a limitar transações que outros utilizadores possam fazer.

2.2. Controlo

O controlo de concorrência das transações é feito pelo SGBD. A execução concorrente de transações efetuadas por vários utilizadores necessita de ser organizada de tal modo que cada transação não pode interferir com as restantes. O objetivo é que cada transação pareça estar a executar isoladamente.

O SGBD trata automaticamente dos pedidos de bloqueio e desbloqueio (trincos) e escalona as ações das diferentes transações, de forma a assegurar que a execução final é equivalente à execução das transações uma a uma, por ordem.

Caso o controle de concorrência não esteja a ser bem efetuado, podem surgir problemas, tais como:

- *Dirty Reads* - Uma leitura suja ocorre quando uma transação lê dados que ainda não foram *committed*. Por exemplo, uma transação edita os dados de uma linha e, uma segunda transação, faz a leitura dessa mesma linha, antes da primeira transação efetuar o *commit* e esta faz um *rollback* das modificações. A segunda transação obtém assim dados que se consideram nunca ter existido.
- *Non-repeatable Reads* - Uma leitura que não é possível repetir ocorre quando uma transação lê um conjunto de dados e, quando repete a transação, o conjunto de dados foi alterado, por outra transação (que realizou um *commit* após a primeira leitura).
- *Phantom Reads* - Uma leitura fantasma é um tipo especial de *Non-repeatable read*, em que uma transação repete um *SELECT* com cláusula *WHERE*, entre dois valores, e, entre essas duas operações, uma segunda transação cria uma operação de *INSERT*, num valor que pertence à cláusula de *WHERE* da primeira transação. Neste cenário, poderão ser retornados valores diferentes, em cada uma das cláusulas de *SELECT*.

2.3. Níveis de Isolamento

As transações possuem um nível de isolamento que define quão isolada essa transação deve ser de modificações de dados feitas por outras transações. Os níveis de isolamento são descritos pelo tipo de efeitos secundários de concorrência, como leituras fantasma ou leituras sujas (*phantom reads* e *dirty reads*), que são permitidos, conforme demonstrado na tabela 1.

Os níveis de isolamento permitem controlar:

- Qual o tipo de bloqueios aplicados e se estes são utilizados nas leituras de dados;
- Quanto tempo duram os bloqueios de leitura;
- Se uma operação de leitura, que interage com dados modificados por outra transação, pode:
 - Ser bloqueada até que o bloqueio acabe;
 - Obter a versão *committed* dos dados existentes nesse momento ou no início dessa transação;
 - Obter a versão *uncommitted* dos dados modificados;

No entanto, a escolha do nível de isolamento não afeta os bloqueios necessários para proteger as modificações de dados. Uma transação tem sempre um bloqueio exclusivo em qualquer dado que modifique, que se mantém até que a transação termine, independentemente do nível de isolamento dessa transação. Para as operações de leitura, os níveis de isolamento das transações definem, principalmente, quão protegida será essa leitura em relação às modificações feitas por outras transações.

Um nível de isolamento mais baixo permite a existência de vários acessos a esses dados em simultâneo, mas ao mesmo tempo aumenta também o número de efeitos de concorrência, como leituras sujas ou modificações perdidas, que os utilizadores podem encontrar. Pelo contrário, um nível de isolamento mais alto diminui o tipo de efeitos de concorrência que os utilizadores podem encontrar, mas requer mais recursos e aumenta a chance de uma transação bloquear outra.

A escolha de um nível de isolamento apropriado deve ter em conta o balanço entre a necessidade de manter a integridade dos dados e a sobrecarga de cada nível de isolamento. O nível de isolamento mais alto, *serializable*, garante que a transação irá retornar sempre os mesmos dados cada vez que repete uma operação de leitura, mas cria um nível de bloqueio que muito provavelmente irá influenciar outros

utilizadores do sistema. Por outro lado, o nível de isolamento mais baixo, *read uncommitted*, pode retornar dados que já foram modificados, mas não *committed* por outras transações. Todos os efeitos de concorrência podem ocorrer no nível mais baixo, mas não há qualquer bloqueio para os outros utilizadores.

2.3.1. Read Uncommitted

O *Read Uncommitted* é o nível de isolamento mais baixo. Neste nível, todos os efeitos de concorrência podem ocorrer, pois não existe qualquer bloqueio para outras transações. De modo a não afetar outras transações, as transações feitas neste modo são, por norma, apenas de leitura.

2.3.2. Read Committed

O *Read Committed* é o nível de isolamento utilizado por omissão no Microsoft SQL Server. Neste nível de isolamento já não são permitidas leituras sujas (*dirty reads*), ou seja, não é possível ler dados que ainda não estão *committed* na base de dados.

2.3.3. Repeatable Reads

Neste nível de isolamento é feito um bloqueio ao nível da leitura e escrita (obtida na leitura) até ao fim da transação. No entanto não são efetuados *range-locks*, portanto leituras fantasma podem ocorrer. É possível também que haja *write skew*, ou seja, que haja alterações feitas por dois utilizadores na mesma coluna. No entanto, já não é possível que duas leituras obtenham resultados diferentes, se não houver alterações.

2.3.4. Serializable

É o mais alto nível de isolamento, é feito um bloqueio ao nível da concorrência e deixam de ser possíveis leituras e escritas (por parte dos outros utilizadores dos dados seleccionados) devido aos bloqueios, que só são terminados no fim da transação. São também feitos *range-locks* quando é feito um SELECT com uma cláusula WHERE, de modo a evitar leituras fantasma.

Nível de Isolamento	<i>Dirty Reads</i>	<i>Non repeatable Read</i>	<i>Phantom Read</i>
<i>Read uncommitted</i>			
<i>Read committed</i>			
<i>Repeatable read</i>			
<i>Serializable</i>			

TABELA 1 - EFEITOS DE CONCORRÊNCIA QUE PODEM OCORRER NOS DIFERENTES NÍVEIS DE ISOLAMENTO.

2.4. Recuperação

A capacidade de recuperar transações é uma necessidade dos sistemas de gestão de bases de dados, de modo a garantirem as propriedades ACID no caso de falhas de energia ou de software/hardware. Esta recuperação pode ser feita através de um ficheiro log, que guarda as alterações feitas na base de dados no disco e assim é salvaguardado de alguns tipos de falhas. Outras formas de recuperar os dados podem passar por existir uma duplicação de dados (*mirroring*), em que são mantidas duas cópias simultâneas da base de dados, ou uma salvaguarda periódica dos dados (*backup*), em que é realizado um *dump* periodicamente para um armazenamento externo.

Existem dois tipos de falhas: Falhas catastróficas e falhas não catastróficas.

No caso de uma falha catastrófica é feita uma cópia anterior da DB a partir de um backup arquivado. Se, quando iniciada, a base de dados estiver inconsistente ou não tiver sido corretamente desligada, o SGBD procura transações *uncommitted* e faz um *rollback* nas alterações feitas por estas transações. Por outro lado, todas as transações que tiverem sido *committed*, mas que não se encontrem materializadas na base de dados vão ser refeitas. Estas operações são realizadas de forma a assegurar a atomicidade e a durabilidade das transações.

No caso de uma falha não catastrófica, são invertidas as operações que causaram inconsistência, desfazendo as alterações por estas provocadas e refeitas alterações legítimas que se perderam. Estas alterações são feitas com base no *system log*, não sendo necessário usar uma cópia completa da base de dados.

3. Tipos de Locks

Record locking ou, em português, bloqueio de registos é um mecanismo que previne o acesso simultâneo a dados de modo a salvaguardar a consistência destes. No sistema de gestão de base de dados utilizado, Microsoft SQL Server, existem dois tipos de fechos, que serão abordados de seguida.

3.1. Fechos binários

Este tipo de trinco pode ter dois estados: *locked* e *unlocked*. Consequentemente, existem dois tipos de operações para bloquear e desbloquear itens na base de dados. Se um recurso está bloqueado por uma transação, nenhuma outra transação tem acesso a esse recurso. Se estiver desbloqueado, uma transação pode aceder a qualquer momento.

Ao efetuar uma transação, esta deve trancar os registos a que acede, e liberta-os apenas quando acaba, permitindo assim o acesso a esses registos por parte de outras transações que estejam à espera.

Este tipo de fecho é demasiado restritivo para termos práticos pois apenas consente a uma transação a posse do trinco, forçando todas as outras transações a aguardar que o trinco seja libertado.

3.2. Fechos multi-modo

O SGBD deve permitir a várias transações o acesso ao mesmo recurso se apenas pretendem ler os dados. No entanto, deve prevenir a inconsistência de dados, e, como tal, bloquear a entrada de várias transações em simultâneo a dados que vão ser alterados. Para isso existem os trincos multi-modo, que permitem uma gestão dinâmica das permissões de acesso à informação.

3.2.1. Shared ou Partilhado

Este tipo de *lock* é utilizado em operações de leitura que não alteram os dados, como a operação `SELECT`. Nenhuma outra transação pode alterar os dados enquanto existir um bloqueio deste tipo nos registos a editar. Este tipo de *locks* é libertado após a operação de leitura, exceto para os níveis de isolamento *Repeatable Read* ou superior.

3.2.2. Exclusive ou Exclusivo

Locks do tipo *Exclusive* previnem o acesso concorrente a registos: nenhuma outra transação pode alterar os dados, e as operações de leitura só podem ser realizadas através do uso de `NOLOCK` ou do nível de isolamento *Read Uncommitted*. São criados quando uma operação DML (*Data Manipulation Language*) é feita, como as operações `INSERT`, `DELETE` e `UPDATE`.

3.3. Two-Phase Locking – 2PL

O mecanismo de fecho também tem falhas (como, por exemplo, a perda da atualização de dados quando os *locks* são libertados cedo demais). Para impedir a ocorrência desses erros, podemos utilizar um protocolo que se denomina de trancamento em duas fases.

O protocolo 2PL divide a fase de execução de uma transação em três partes distintas:

1. A transação, quando a sua execução inicia, vai procurar permissão para os fechos que necessita.
2. A transação obtém todas as transações que precisa.
3. Quando liberta o primeiro fecho, a terceira parte começa; A partir deste momento, a transação não pode adquirir mais locks, apenas libertá-los quando acaba de os utilizar.

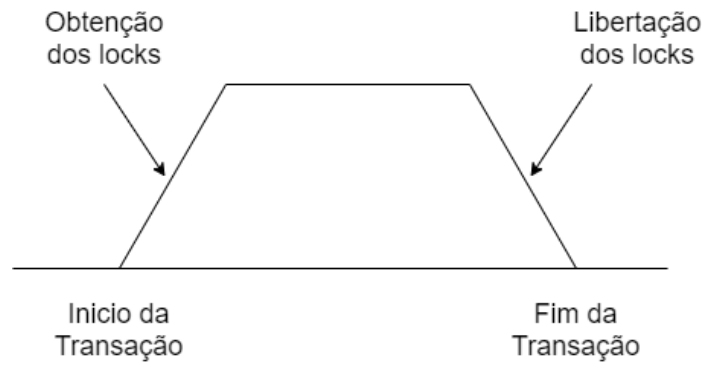


FIGURA 2 - TWO-PHASE LOCKING

Uma variante deste protocolo é o 2PL estrito, em que a transação mantém todos os locks até acabar (quando é chamado o commit ou rollback), em vez de libertá-los gradualmente após a utilização de cada um.

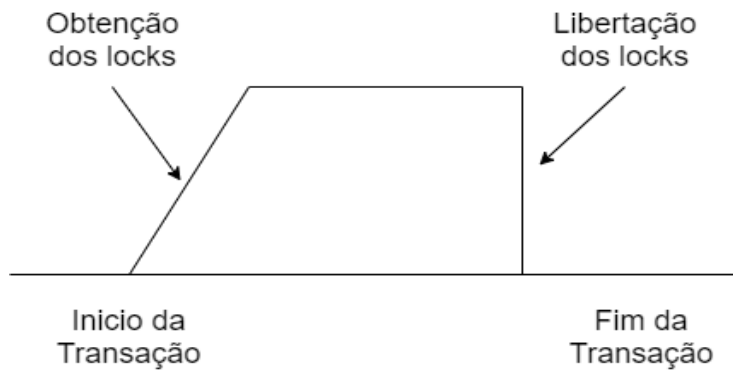


FIGURA 3 - TWO-PHASE LOCKING ESTRITO

4. Testes e Análises de Resultados

4.1. Efeitos dos Níveis de Isolamento

Para testar os efeitos dos diferentes níveis de isolamento nas transações criámos um caso de estudo em que dois clientes, denominados por Cliente 1 e Cliente 2 neste teste, executam várias transações (com nomes Transação 1 e Transação 2, respetivamente) simultaneamente, para todas as combinações de níveis de isolamento. As transações estão divididas por passos, de (A) a (G), o que facilitará a compreensão das tabelas de resultados por nós geradas.

4.1.1. Transações

Transação 1	Transação 2
<hr/> <pre>begin;</pre> <p>(A)</p> <pre>select * from Factura;</pre> <pre>rollback;</pre> <hr/>	<pre>begin;</pre> <pre>update Factura set</pre> <pre>Nome='XX' where FacturaID=1;</pre> <pre>rollback;</pre> <hr/>
<pre>begin;</pre> <pre>select * from Factura;</pre> <p>(C)</p> <pre>select * from Factura;</pre> <p>(D)</p> <pre>select * from Factura;</pre> <hr/>	<pre>begin;</pre> <p>(B)</p> <pre>update Factura set</pre> <pre>Nome='XX2' where FacturaID=1;</pre> <pre>commit;</pre> <pre>rollback;</pre> <hr/>
<pre>begin;</pre> <pre>select * from Factura;</pre>	<pre>begin;</pre>

(F)

```
update Factura set  
Nome='ZZ1' where FacturaID=1;
```

```
rollback;
```

(E)

```
select * from Factura;
```

(G)

```
update Factura set  
Nome='ZZ2'  
where FacturaID=1;
```

```
rollback;
```

4.1.2. Tabelas de Resultados

Legenda dos Níveis de Isolamento:

- Nível 1 - Read Uncommitted
- Nível 2 - Read Committed
- Nível 3 - Repeatable Read
- Nível 4 - Serializable

Tabelas (linhas correspondem ao Cliente 1, colunas correspondem ao Cliente 2)

Passo (a)

	<u>Read Uncommitted</u>	<u>Read Committed</u>	<u>Repeatable Read</u>	<u>Serializable</u>
<u>Read Uncommitted</u>	O primeiro nome é atualizado	O primeiro nome é atualizado	O primeiro nome é atualizado	O primeiro nome é atualizado
<u>Read Committed</u>	O primeiro nome mantém-se	O primeiro nome mantém-se	O primeiro nome mantém-se	O primeiro nome mantém-se
<u>Repeatable Read</u>	O primeiro nome mantém-se	O primeiro nome mantém-se	O primeiro nome mantém-se	O primeiro nome mantém-se
<u>Serializable</u>	O primeiro nome mantém-se	O primeiro nome mantém-se	O primeiro nome mantém-se	O primeiro nome é alterado para "XX", no entanto o primeiro Cliente executa um rollback, portanto o segundo Cliente atualiza o nome.

- Para o nível 1 do Cliente1, o nome é sempre atualizado;
- Para o nível 2 e 3 do Cliente1 (independentemente do nível do Cliente 2), e para o nível 4 do Cliente 1 enquanto nível 1, 2 e 3 do Cliente 2, o nome mantém-se;

- Para o nível 4 em ambos os Clientes 1 e 2, o nome é alterado, e o Cliente 2 espera que o select seja executado, após o rollback do Cliente 2

Passos (b), (c), (d)

	<u>Read Uncommitted</u>	<u>Read Committed</u>	<u>Repeatable Read</u>	<u>Serializable</u>
<u>Read Uncommitted</u>	b é executado, c e d mostram valores atualizados, independentemente do commit	b é executado, c e d mostram valores atualizados, independentemente do commit	b é executado, c e d mostram valores atualizados, independentemente do commit	b é executado, c e d mostram valores atualizados, independentemente do commit
<u>Read Committed</u>	c mostra o valor antigo, enquanto d , após o commit, mostra o valor atualizado	c mostra o valor antigo, enquanto d , após o commit, mostra o valor atualizado	c mostra o valor antigo, enquanto d , após o commit, mostra o valor atualizado	c mostra o valor antigo, enquanto d , após o commit, mostra o valor atualizado
<u>Repeatable Read</u>	c mantém os valores antigos	c mantém os valores antigos	c mantém os valores antigos	c mantém os valores antigos
<u>Serializable</u>	O cliente 2 fica à espera em b até o rollback ser executado pelo cliente 1	O cliente 2 fica à espera em b até o rollback ser executado pelo cliente 1	O cliente 2 fica à espera em b até o rollback ser executado pelo cliente 1	O cliente 2 fica à espera em b até o rollback ser executado pelo cliente 1

- Para o Cliente 1 - nível 1, **b** é executado, **c** e **d** mostram valores atualizados, independentemente do commit;
- Para o Cliente 1 - nível 2, **b** é executado, **c** mantém o valor antigo, e **d** mostra o valor atualizado;
- Para o Cliente 1 - nível 3, **b** é executado, **c** e **d** mantém o valor antigo;
- Para o Cliente 1 - nível 4, a execução é congelada em **b**, até o rollback ser executado pelo cliente 1. O update é feito, no entanto, o cliente 1 já tinha dado rollback, pelo que não é visível (poderá ser visível numa segunda iteração no cliente 1)

Passos (e), (f), (g)

	<u>Read Uncommitted</u>	<u>Read Committed</u>	<u>Repeatable Read</u>	<u>Serializable</u>
<u>Read Uncommitted</u>	A query de SELECT deu-nos o valor antigo, f e g são executados sem qualquer espera.	A query de SELECT deu-nos o valor antigo, f e g são executados sem qualquer espera.	A query de SELECT deu-nos o valor antigo, f terminou e g ficou em <u>lock</u> , até o cliente1 executar o rollback.	A query de SELECT deu-nos um valor desatualizado, f ficou em <u>lock</u> e g deu-nos um <u>deadlock</u> .
<u>Read Committed</u>	A query de SELECT deu-nos o valor antigo, f terminou e g ficou em <u>lock</u> , até o cliente1 executar o rollback.	A query de SELECT deu-nos o valor antigo, f terminou e g ficou em <u>lock</u> , até o cliente1 executar o rollback.	A query de SELECT deu-nos o valor antigo, f terminou e g ficou em <u>lock</u> , até o cliente1 executar o rollback.	A query de SELECT deu-nos um valor desatualizado, f ficou em <u>lock</u> e g deu-nos um <u>deadlock</u> .
<u>Repeatable Read</u>	A query de SELECT deu-nos o valor antigo, f terminou e g ficou em <u>lock</u> , até o cliente1 executar o rollback.	A query de SELECT deu-nos o valor antigo, f terminou e g ficou em <u>lock</u> , até o cliente1 executar o rollback.	A query de SELECT deu-nos o valor antigo, f terminou e g ficou em <u>lock</u> , até o cliente1 executar o rollback.	A query de SELECT deu-nos um valor desatualizado, f ficou em <u>lock</u> e g deu-nos um <u>deadlock</u> .
<u>Serializable</u>	A query de SELECT deu-nos o valor antigo, f terminou e g ficou em <u>lock</u> , até o cliente1 executar o rollback.	A query de SELECT deu-nos o valor antigo, f terminou e g ficou em <u>lock</u> , até o cliente1 executar o rollback.	A query de SELECT deu-nos o valor antigo, f terminou e g ficou em <u>lock</u> , até o cliente1 executar o rollback.	A query de SELECT deu-nos um valor desatualizado, f ficou em <u>lock</u> e g deu-nos um <u>deadlock</u> .

- Para o Cliente 1 - nível 1, e níveis 1 e 2 do Cliente 2, e deu-nos um valor desatualizado, enquanto que **f** e **g** foram executados sem demora;
- Para o Cliente 1 - nível 2, e Cliente 2 - nível 3, e deu-nos um valor desatualizado, **f** foi executado e **g** teve a sua execução em lock;
- Para o Cliente 1 - nível 2, 3 e 4, e níveis 1, 2 e 3 do Cliente 2, e deu-nos um valor desatualizado, **f** foi executado e **g** teve a sua execução em lock;
- Para qualquer nível do Cliente 1 e nível 4 do Cliente 2, e deu-nos um valor desatualizado, **f** teve a sua execução em lock e **g** deu-nos um deadlock.

4.2. Testes da Aplicação Desenvolvida

A aplicação concebida conta com três funcionalidades obrigatórias para a execução dos testes: *Edit*, *Work* e *Browser*. A explicação dos testes através de cada uma das funcionalidades será abordada nos próximos subcapítulos.

Para esclarecimento de como utilizar as funcionalidades, explica-se que a aplicação dispõe de uma barra de ferramentas com três ícones como a que podemos ver na figura 4. O ícone com o símbolo “+” permite utilizar a funcionalidade *Edit*, o segundo ícone a funcionalidade *Work*. O terceiro ícone permite refrescar a tabela que esteja visível.

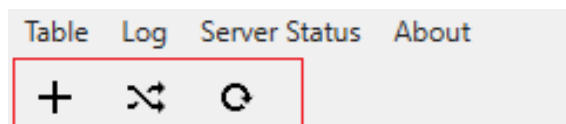


FIGURA 4 - BARRA DE FERRAMENTAS DA APLICAÇÃO

Na figura 8 (no subcapítulo 4.2.3) é possível ver o ecrã principal do programa. Para alterar o nível de isolamento das transações, basta alterar o valor na caixa *dropdown* que se encontra na barra lateral direita.

O programa dispõe ainda de ferramentas para verificar e executar a conexão à base de dados. Na barra de ferramentas (figura 4), é possível ver um item que diz “Server Status”. Ao clicar, aparecerá um

log com a informação sobre a conexão à base de dados (eventuais problemas, informação de conexão bem-sucedida, etc.). Na barra lateral direita, existe um círculo com uma cor que indica o estado da conexão (vermelho para não conectado, amarelo quando tenta estabelecer, e verde para conectado) e um botão que diz “Connection Settings”, que ao clicar faz aparecer a janela de parâmetros de conexão (figura 5). Aí, é possível editar o servidor, nome da base de dados, nome de utilizador e palavra-passe.

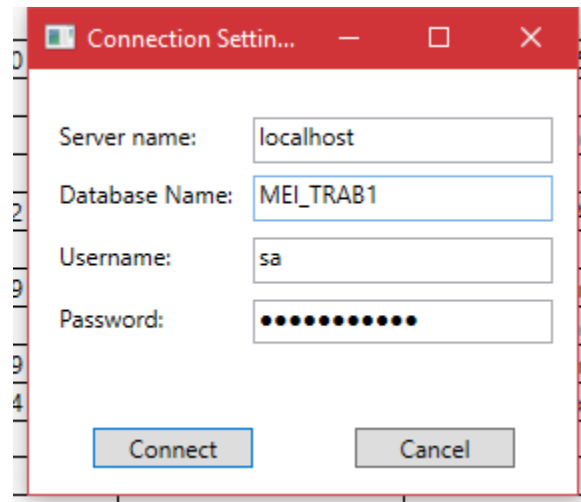


FIGURA 5 - DEFINIÇÕES DE CONEXÃO DA APLICAÇÃO

4.2.1. Funcionalidade *Edit*

O programa desenvolvido permite a edição individual de registos da tabela Factura através de uma janela específica.

Para o caso da edição de uma fatura *uncommitted*, foi utilizado o Microsoft SQL Management Studio para executar a primeira transação (sem dar *commit*):

```
begin tran
insert into Factura
values (1, 1, 1, 1)
```

De seguida, foi utilizada a função *Edit* da aplicação para editar esse registo. Para finalizar, no Management Studio, variámos entre o uso do *commit* e do *rollback*, e foram encontrados dois comportamentos:

- Para o caso do *commit*, o registo foi inserido na tabela com os dados introduzidos na janela de *Edit*, ou seja, os dados originais perderam-se.
- No caso do *rollback*, o registo não permaneceu na tabela, mesmo após a edição do registo do lado da aplicação.

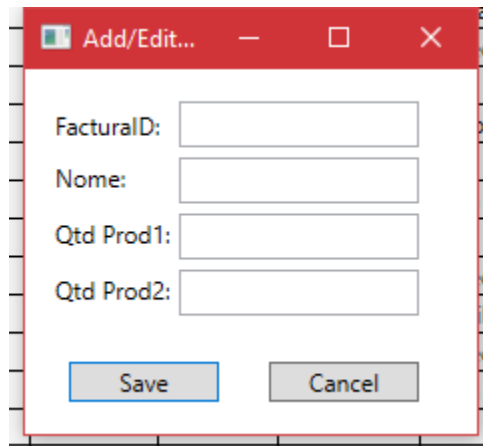


FIGURA 6 - FUNCIONALIDADE EDIT DA APLICAÇÃO

4.2.2. Funcionalidade *Work*

A funcionalidade *Work* permite colocar o sistema de gestão de bases de dados sobre carga, isto é, cria várias transações de inserção, modificação e eliminação de dados e executa-as. A janela que inicia esta funcionalidade permite escolher um número positivo de transações a serem criadas.

Testámos esta funcionalidade com vários clientes, para um pequeno número de transações e também para um pequeno número.

Sobre os níveis de isolamento, verificámos que as transações são instantâneas quando são executadas com o nível *Read Uncommitted*. Para qualquer outro nível, a primeira transação executada obtém o trinco sobre o registo e não permite o acesso aos dados, e a transação seguinte espera até o trinco ser libertado para poder ler e editar.

Em termos de número de transações criadas, apurámos que a contenção é maior quando o número de transações é menor: os clientes vão tentar aceder, nas operações de UPDATE e DELETE, aos mesmos registos mais frequentemente. Tal pode ser visto na tabela LogOperations, que mostra os tempos

de início e fim das transações. A diferença entre esses dois tempos aumenta consoante a contenção dos acessos.

Averiguámos também que ao aumentar o número de clientes o acesso aos mesmos registos aumenta também, e, por consequência, o intervalo entre o tempo de início e fim da transação também aumenta.

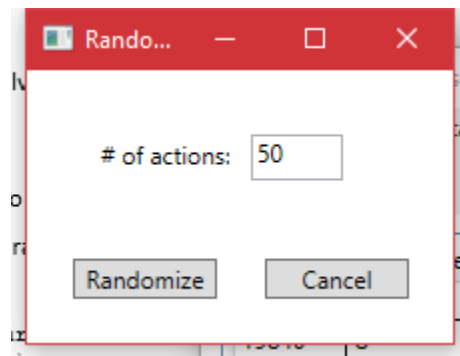


FIGURA 7 - FUNCIONALIDADE *Work* DA APLICAÇÃO

4.2.3. Funcionalidade *Browser*

A solução criada permite a visualização das tabelas *Factura* e *LogOperations* em tempo real, e contém um temporizador para refrescamento destas tabelas, que vai desde 200 milissegundos a 3000 milissegundos (ou 3 segundos).

Para testar esta função, colocámos o temporizador no valor mais baixo num cliente e, com vários outros clientes, submetemos carga (cerca de 50 operações por cliente) na base de dados através da opção *Work*.

Verificámos que, por vezes, o programa ficava em espera e não conseguia atualizar as tabelas devido aos *locks* das outras operações. Isto deve-se ao facto de o nível de isolamento por defeito da base de dados ser *Read Committed*, o que impossibilita a leitura dos dados pois outras transações, provenientes do *Work*, estão na posse de fechos sobre os dados (neste caso, fechos exclusivos).

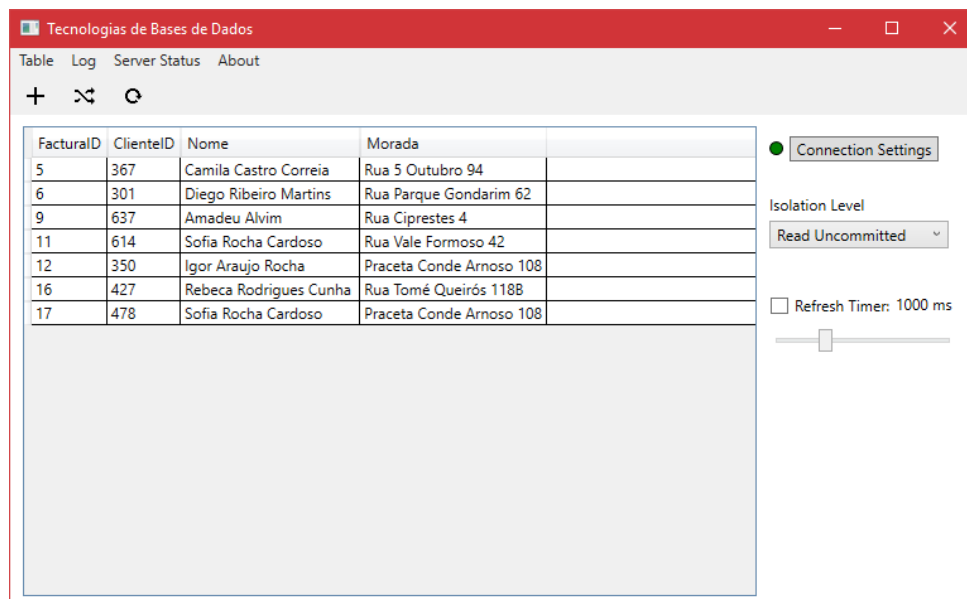


FIGURA 8 - FUNCIONALIDADE BROWSER DA APLICAÇÃO

5. Conclusão

Uma transação simboliza uma unidade de trabalho, realizado num sistema de gestão de base de dados (ou similar). Deve ter como propósitos a recuperação correta da base de dados, especialmente em casos de falhas, e providenciar isolamento a acessos concorrentes à base de dados. Uma transação deverá ter sempre as propriedades de atomicidade, consistência, isolamento e durabilidade, bem como seguir uma proposição de "tudo ou nada", ou seja, uma transação deverá sempre terminar totalmente ou não ter qualquer efeito.

Em sistemas de bases de dados, o isolamento determina como a integridade de uma transação é visível a outros utilizadores e sistemas. Um nível de isolamento menor, aumenta a capacidade de um número elevado de utilizadores aceder à mesma informação, ao mesmo tempo, aumentando, em contrapartida, o número de efeitos de concorrência (tais como *dirty reads* ou obtenção de informação desatualizada) encontrados. Por outro lado, um nível de isolamento superior reduz o número de tipos de efeitos de concorrência que um utilizador pode encontrar, requerendo, no entanto, uma maior utilização de recursos de sistema e aumentando a chance de uma transação bloquear outra, providenciando um desempenho catastrófico numa base de dados com muitos acessos concorrentes.

Foi desenvolvida uma aplicação, com interface gráfica, com o intuito de ser fácil de utilizar por qualquer pessoa. Através desta, fomos capazes de compreender facilmente o funcionamento das transações num sistema de gestão de base de dados (Microsoft SQL Server), bem como compreender as interações entre os diversos níveis de isolamento. A aplicação desenvolvida permite simular um elevado número de transações (com *queries* de INSERT, UPDATE e DELETE), automaticamente e aleatoriamente. Foi também introduzida a possibilidade de inserir e editar valores na base de dados, manualmente, visualizar a tabela principal da base de dados (para constataremos efeitos de concorrência), escolher o número de transações que são feitas, e o nível de isolamento.

6. Bibliografia

- I. Ramakrishnan, R., & Gehrke, J. (2011). Database management systems. Boston: McGraw-Hill.
- II. ACID Properties in DBMS. (2017, July 11). Retrieved October 29, 2017, from <http://www.geeksforgeeks.org/acid-properties-in-dbms/>
- III. Isolation Levels in the Database Engine. (n.d.). Retrieved October 29, 2017, from <https://technet.microsoft.com/en-us/library/ms189122%28v=SQL.105%29.aspx>
- IV. Point, T. (2017, August 15). DBMS Concurrency Control. Retrieved October 29, 2017, from https://www.tutorialspoint.com/dbms/dbms_concurrency_control.htm
- V. Lock Modes. (n.d.). Retrieved October 29, 2017, from <https://technet.microsoft.com/en-us/library/ms175519%28v=sql.105%29.aspx>