

Universidade da Beira Interior

Departamento de Informática



**Departamento de
Informática**

Nº Trabalho Prático 1: *Controlo de Transações*

Elaborado por:

M13939 João Martins

M13943 Rui Simões

E11307 Fábio Dias

Orientador:

Professor Doutor João Manuel da Silva Fernandes Muranho

10 de novembro de 2023

UBI Universidade da Beira Interior

ID Identifier

SGBD Sistema de Gestão de Base de Dados

ACID *Atomicity, Consistency, Isolation, Durability*

SQL *Structured Query Language*

IDE *Integrated Development Environment*

UI *User Interface*

Conteúdo

Conteúdo	3
Lista de Figuras	4
1 Introdução	7
1.1 Âmbito, Enquadramento e Motivação	7
1.2 Objetivos Gerais	7
1.3 Organização do Documento	7
2 Gestão e Controlo de Transações	9
2.1 Introdução	9
2.2 Transações	9
2.3 Níveis de Isolamento	11
2.4 Transações Incompletas, <i>Crash</i> de Sistemas e Recuperação . . .	12
2.5 Trincos (Locks)	13
3 Implementação e Testes	15
3.1 Introdução	15
3.2 Implementação	15
3.3 Testes	21
4 Conclusão	25
4.1 Conclusões Principais	25
4.2 Sugestões de Melhoria	25
Bibliografia	27

Lista de Figuras

2.1	Relação entre os níveis de isolamento e os fenómenos associados.	12
2.2	representação de <i>deadlocks</i>	14
3.1	Janela de login	16
3.2	Imagem ilustrativa da Aplicação Edit	17
3.3	Imagem ilustrativa da Aplicação Edit	17
3.4	Imagem ilustrativa da aplicação Log Tempo antes de uma alteração.	21
3.5	Imagem ilustrativa da Aplicacao Edit em Teste.	22
3.6	Imagem ilustrativa da Aplicacao Edit em Teste.	22
3.7	Imagem ilustrativa da Aplicacao Edit em Teste.	23

Lista de Excertos de Código

2.1	Instrução SQL para a definição base de uma transação com possíveis níveis de isolamento e especificações de leitura e escrita. .	9
3.1	Instrução SQL para guardar ou descartar as alterações feitas na transação.	16
3.2	Instrução SQL que guarda na tabela em questão dados referentes ao acesso da encomenda	18
3.3	Identificação do nível de isolamento escolhido e início da transação com esse nível.	18
3.4	Pesquisar encomenda (com produtos associados) e imprimir valores na tabela.	19
3.5	Queries para alterar a morada da encomenda e quantidade de determinado produto na mesma.	19
3.6	Query <i>Structured Query Language</i> (SQL) para a leitura de todos os dados das primeiras linhas da tabela <i>LogOperations</i> cujo evento é do tipo I/U/D.	21

Capítulo 1

Introdução

1.1 Âmbito, Enquadramento e Motivação

Este documento, o relatório escrito, do segundo trabalho prático da unidade curricular de Sistemas de Gestão de Bases de Dados (SGBD), tem como objetivo estudar como se processa o controlo de transações, nível de isolamento, tipos de *locks* e *deadlocks*.

1.2 Objetivos Gerais

Este trabalho requer conhecimentos multidisciplinares para a sua realização, como bases teóricas sobre Sistema de Gestão de Base de Dados (SGBD), linguagens de programação de alto nível, neste caso C# para criação e configuração de aplicações gráficas.

O principal objetivo deste trabalho é estudar o controlo de transações, nomeadamente as questões em torno dos níveis de isolamento e *locks*.

Para que este estudo possa ser feito, todas as funcionalidades das quatro aplicações *Edit*, *Browser*, *Log Tempo* e *Log* terão de ser postas em prática com a finalidade de testar o controlo de transações. Estas aplicações serão aprofundadas em capítulos posteriores.

1.3 Organização do Documento

Este documento foi estruturado da seguinte forma:

- Capítulo 1: Descreve, de modo geral, quais os tópicos abordados, os objetivos do trabalho em questão e as métodos utilizados para alcançar os mesmos.

- Capítulo 2: Os conceitos de trasação (e os diferentes tipos das mesmas), nível de isolamento e *locks* serão abordados.
- Capítulo 3: Descreve, em detalhe, a implementação trabalhos e explica o funcionamento das quatro aplicações. Explica quais os testes feitos e o seu propósito.
- Capítulo 4: Irão se discutir as conclusões do trabalho, bem como pontos de melhoria.

Capítulo 2

Gestão e Controlo de Transações

2.1 Introdução

Uma transação é um conjunto de operações perfeitamente delimitado em que garantidamente são executadas todas as instruções ou então nenhuma produzirá efeitos sobre a base de dados [1].

Para preservar a integridade dos dados, o SGBD tem de assegurar as propriedades *Atomicity, Consistency, Isolation, Durability (ACID)*.

Na secção 2.2, é aprofundado o que é a gestão e o controlo de transações realizada pelo SGBD. As secções 2.3 e 2.4 abordam temas como isolamento e os trincos, respetivamente, no controlo de transações.

2.2 Transações

O modelo de transação é baseado em três instruções SQL: o início da transação é demarcado por *START TRANSACTION* e o fim por *ROLLBACK* ou por *COMMIT*.

Uma transação é uma unidade lógica básica de execução num sistema de informação. É uma sequência de operações que são executadas como um todo, e que leva uma base de dados dum estado consistente para outro. Por outras palavras, é uma unidade indivisível de processamento. [2]

Uma transação tem um formato base que está ilustrado no excerto de código 2.1.

```
SET TRANSACTION
[READ ONLY | READ WRITE ] |
[ ISOLATION LEVEL READ UNCOMMITTED | READ COMMITTED |
  REPEATABLE READ | SERIALIZABLE ]
```

Excerto de Código 2.1: Instrução SQL para a definição base de uma transação com possíveis níveis de isolamento e especificações de leitura e escrita.

Cada vez que se começa uma transação e caso se esteja a utilizar um log, é criado no log uma entrada nova com um número único que a identifica. A menos de alguns detalhes específicos de implementação, os registos principais que compõem o log são os seguintes[3]:

1. [start_transaction, transactionid] - o início da execução de uma transação é identificado por esta instrução.
2. [read_item, transactionid, X] - a transação identificada pelo transactionid lê o valor do item X da base de dados. É opcional em alguns protocolos.
3. [write_item, transactionid, X, old_value, new_value] - a transação identificada pelo transactionid altera o valor do item X da base de dados de old_value para new_value. Repare-se que, no caso de alteração, o log guarda o valor antigo e o novo.
4. [commit, transactionid] - a transação identificada pelo transactionid completou todos os acessos à base de dados com sucesso e o seu efeito pode ser gravado permanentemente (**committed**).
5. [abort, transactionid] - a transação identificada pelo transactionid abortou (**abort**).

Para garantir a consistência e a integridade dos dados, é fundamental criar um plano de execução eficiente que controle o acesso simultâneo às informações (calendarização).

Um ***schedule*** é então responsável por garantir o isolamento das transações e a manutenção da consistência do banco de dados, respeitando a atomicidade e a durabilidade. Isso envolve a criação de um plano de execução que determina a ordem em que as transações são executadas e como elas bloqueiam e libertam recursos partilhados, como tabelas ou registos.

A escolha e implementação de algoritmos de calendarização adequados são essenciais para garantir que as propriedades ACID sejam mantidas em ambientes com múltiplas transações concorrentes em um SGBD.

Execução Concorrente de Transações

A maioria dos SGBD são sistemas multi-utilizador logo a execução concorrente de transações submetidas por vários utilizadores tem de ser organizada

de tal modo que cada transação não interfere com as restantes, pois só assim se pode garantir que não há resultados incorretos.

A execução concorrente de transações tem de ser tal que cada transação pareça estar a executar isoladamente.

O objetivo principal da execução concorrente de transações é **maximizar** o uso de recursos do sistema, melhorar o desempenho e aumentar a capacidade de atender a múltiplos utilizadores simultaneamente. Foca-se em otimizar a utilização da CPU, memória e outros recursos, a fim de fornecer respostas rápidas e eficientes às solicitações dos utilizadores.

É importante fazer distinção entre os dois tipos de *schedule*:

1. *Schedule* Serializável: um plano de execução de transações em que o resultado é equivalente ao de uma execução sequencial das transações, como se cada transação fosse executada uma após a outra, sem concorrência.
2. *Schedule* Serializável: um plano de execução em que as transações são executadas concorrentemente, mas o resultado não é equivalente ao de uma execução sequencial das mesmas transações.

2.3 Níveis de Isolamento

Quando se trata de isolamento em SGBD, referimo-nos à capacidade do sistema de garantir que as operações concorrentes de diferentes transações não interfiram umas com as outras. Para qualquer transação existem quatro níveis de isolamento em três fenómenos. Os fenómenos são:

- *Dirty Read* - A transação lê dados escritos por uma transação simultânea não submetida (uncommitted).
- *Nonrepeatable Read* - A transação lê dados que já foram lidos anteriormente e analisa quais aqueles que já foram alterados por outra transação.
- *Phantom Read* - A transação executa uma consulta uma segunda vez e retorna um conjunto de registos que satisfazem a condição de procura e descobre que o conjunto de registos retornados é diferente devido a uma transação realizada recentemente.

Os níveis de isolamento são os seguintes:

1. **Read Uncommitted**: as transações não precisam esperar que outras terminem antes de ler dados. Isso significa que uma transação pode ler dados que estão no meio de uma atualização por outra transação, o que pode levar a resultados inconsistentes e a problemas de integridade.
2. **Read Committed**: as transações só podem ler dados que foram confirmados por outras transações. Isso evita que as mesmas acessem a dados não confirmados, o que melhora a consistência, mas ainda permite a leitura de dados sujeitos a alterações.
3. **Repeatable Read**: uma vez que uma transação lê um conjunto de dados, esses dados não serão modificados por outras transações até que a transação atual seja concluída. Isso evita leituras inconsistentes, mas ainda permite a inserção de novos dados.
4. **Serializable**: é o nível de isolamento mais elevado, garante que as transações se comportem como se fossem executadas sequencialmente, evitando conflito e garantindo a máxima consistência, embora possa levar a *deadlocks*.

Isolation Level	Dirty Read	Non Repeatable Read	Phantom
Read uncommitted	Yes	Yes	Yes
Read committed	No	Yes	Yes
Repeatable read	No	No	Yes
Snapshot	No	No	No
Serializable	No	No	No

Figura 2.1: Relação entre os níveis de isolamento e os fenómenos associados.

2.4 Transações Incompletas, *Crash* de Sistemas e Recuperação

Enquanto um utilizador realiza uma transação existe a possibilidade de esta ser cancelada. O sistema pode ter um erro e ser incapaz de completar a transação, pode ocorrer perda de ligação devido a um problema de conexão ou de um dos sistemas *crashar*, ou o utilizador pode simplesmente querer cancelar a transação.

De qualquer maneira, o SGBD deve garantir que quais queres transações incompletas sejam invertidas para manter a consistência da base de dados.

Para isto, é utilizado o **log** mantido pelo SGBD para inverter cada ação realizada pelo utilizador antes do cancelamento da transação.

2.5 Trincos (Locks)

O acesso concorrente a dados pode causar inconsistências na base de dados, por exemplo, se dois utilizadores quiserem incrementar um valor de um registo de uma tabela, pode acontecer que ambos leiam o mesmo valor da base de dados e apenas é registado o incremento do último utilizador a escrever o novo valor. Para evitar isto utilizam-se **locks**. Os *locks* são variáveis que estão presentes em todos os registos, com estes podemos impedir outros utilizadores de alterar ou até aceder aos valores de um registo. Existem dois tipos de *locks*, **binários** e **multi-modo**.

- **Locks binários:** como o nome indica possuem dois estados, trancado e destrancado. Estes são os *locks* mais simples em que um processo verifica se um registo está disponível antes de executar a sua transação e tranca esse registo enquanto faz o seu trabalho, após concluir, desbloqueia o registo para outros processos poderem acedê-lo.

- lock_item(X)
 - unlock_item(X)

- **Locks multi-modo:** são semelhantes aos locks binários, a sua diferença consiste na existência de 2 estados trancados, um deles proíbe a alteração do registo, o **read locked** ou **shared locked** e o outro proíbe também a leitura do registo, o **write locked** ou **exclusive locked**

- read_lock(X)
 - write_lock(X)
 - unlock(Y)

Deadlocks

Ocorrem quando duas ou mais transações requerem acesso a recursos trancados por outra transação.

Estes são um problema muito grave, visto poderem causar com que o sistema não consiga realizar qualquer operação na base de dados, e por isso deve haver sistemas que os previnam. Estes sistemas obrigam uma das transações

a abdicar dos seus recursos temporariamente ou até cancelá-la, devem também existir critérios de seleção para escolher qual das transações será forçada a esperar.

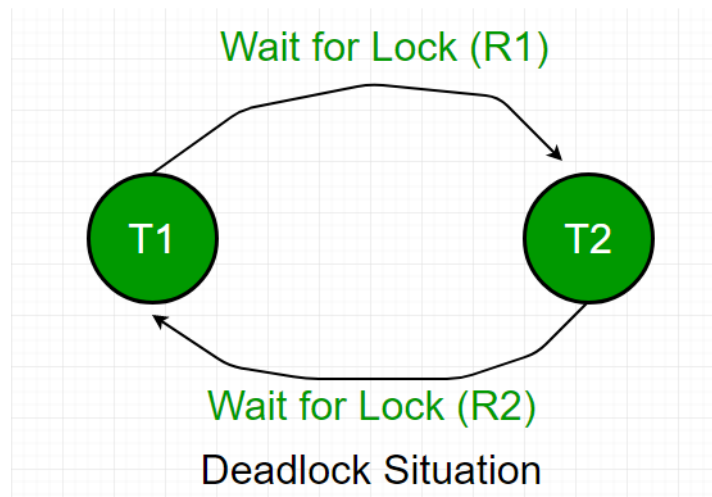


Figura 2.2: representação de *deadlocks*

Capítulo 3

Implementação e Testes

3.1 Introdução

As várias aplicações foram desenvolvidas na linguagem c#, usando o *Integrated Development Environment* (IDE) Visual Studio e as ferramentas deste programa para facilitar o processo de desenvolvimento das várias aplicações, e rapidamente criar o *User Interface* (UI) do programa. Os detalhes da implementação de cada aplicação serão desenvolvidos neste capítulo, bem como os testes realizados para verificar o funcionamento do programa.

3.2 Implementação

Todas as aplicações possuem uma janela de login para aceder a base de dados. Nesta, é pedido ao utilizador os dados necessários para aceder á base de dados, nomeadamente o *host name* que define o dispositivo que vamos aceder por nome ou por ip, o nome da base de dados, e as credenciais do utilizador, o *username* e a palavra passe.

Aplicação Edit

O principal **objetivo** desta aplicação é permitir ao utilizador editar/alterar os dados de uma encomenda específica no sistema. A aplicação segue um conjunto de passos para realizar essa tarefa, incluindo a gravação de registos numa tabela *LogOperations* para rastrear as operações realizadas.

Com esta aplicação o utilizador poderá realizar as seguintes funcionalidades:

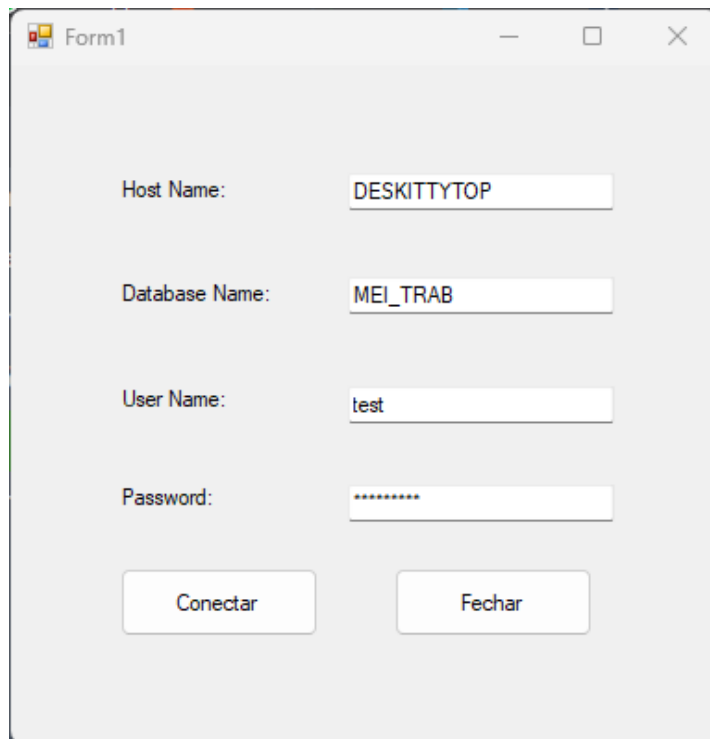
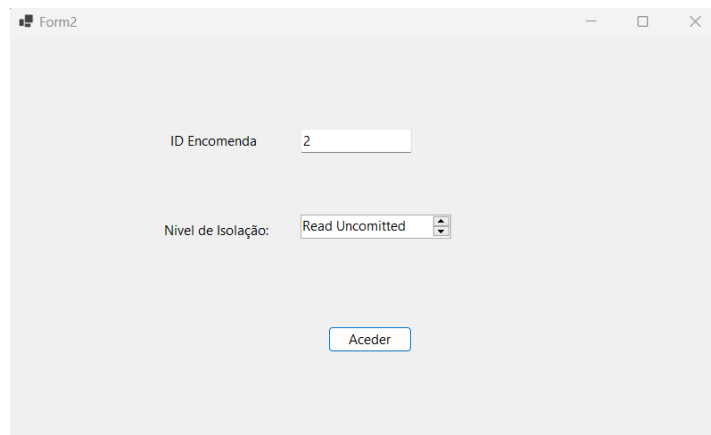


Figura 3.1: Janela de login

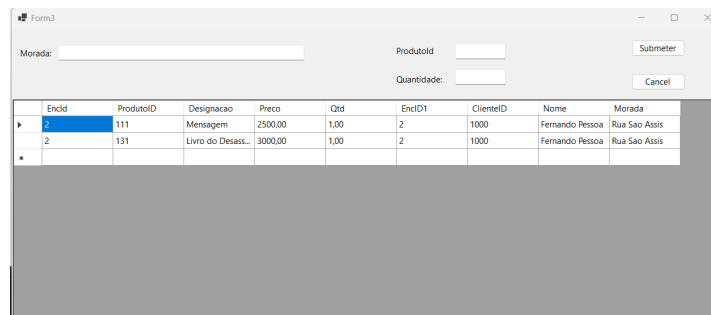
- Consultar Encomenda por Identifier (ID) - No início, é mostrado ao utilizador uma caixa de texto que lhe permitirá inserir o ID da encomenda que este deseja aceder e escolher o nível de isolamento associado à transação que este irá despoletar, como ilustrado na figura 3.4
- Alterar Dados da Encomenda - O utilizador insere os novos dados no campo "Morada" e/ou campo "Quantidade" e no campo "ProdutoID" de forma a indicar que produto da encomenda este deseja alterar a quantidade (campos ilustrados na figura 3.3).
- Submeter ou Cancelar (Terminar a Transação) - O utilizador tem a possibilidade de guardar (Commit) ou descartar (Rollback) as alterações efetuadas como ilustrado na figura 3.3. As *queries* 3.1 são usadas para guardar ou descartar os dados alterados, nesta ordem e assim acabar a transação.

```
transaction.Commit();  
transaction.Rollback();
```

The screenshot shows a window titled 'Form2'. It contains two input fields: 'ID Encomenda' with the value '2' and 'Nivel de Isolação' with the value 'Read Uncommitted'. Below these fields is a button labeled 'Aceder'.

Figura 3.2: Imagem ilustrativa da Aplicação Edit



The screenshot shows a window titled 'Form3'. It contains a table with the following data:

EnclID	ProdutoID	Designacao	Preco	Qtd	EnclID1	ClientID	Nome	Morada
2	111	Mensagem	2500.00	1,00	2	1000	Fernando Pessoa	Rua Sao Assis
2	131	Livro do Desass...	3000.00	1,00	2	1000	Fernando Pessoa	Rua Sao Assis

Below the table is a large grey rectangular area. Above the table, there are input fields for 'Morada:', 'ProdutoID', and 'Quantidade:', and buttons for 'Submeter' and 'Cancel'.

Figura 3.3: Imagem ilustrativa da Aplicação Edit

Excerto de Código 3.1: Instrução SQL para guardar ou descartar as alterações feitas na transação.

A aplicação segue o seguinte *flow*:

1. A aplicação começa por solicitar ao utilizador para introduzir o ID da encomenda.
2. Escreve um registo na tabela *LogOperations*, cujo *query* está ilustrado no trecho de código 3.2 .
3. Inicia uma transação (no nível de isolamento apropriado), cujo trecho de código responsável está ilustrado no excerto de código 3.3 .
4. Lê os dados da encomenda e mostra-os ao utilizador, cujo trecho de código está ilustrado no excerto de código 3.4.

5. Aceita as alterações do utilizador (morada e/ou quantidades) e atualiza a base de dados, termina a transação, escreve um registo na tabela *LogOperations*.

Este passo conclui três funções (acionadas após o *click* no botão "Submiter") com cada uma a ter três parâmetros em comum `sqlConnection`, `connection`, `SqlTransaction transaction`, `SqlCommand command` para não se perder a conexão e a transação que foi iniciada antes.

- `private void AtualizarMorada(string morada)`
- `private void AtualizarMoradaEQuantidade(string morada, int produtoId, int quantidade)`
- `private void AtualizarQuantidade(int produtoId, int quantidade)`
- Como exemplo, está ilustrado no trecho de código ??o *query* em formato *string* utilizado para alterar a morada da encomenda e quantidade de determinado produto na mesma.

```
string User_Reference = "G1-" + date_now.ToString("yyMMddHmss");

string logReguisterQuery = "INSERT INTO LogOperations (EventType,
    Objecto, Valor, Referencia) Values('O', '\" + IdEnc + '\" + \"\", \"
    + '\" + date_now + '\" + \"\", \"\", \"\" + User_Reference + '\" + \"
    )\"";
```

Excerto de Código 3.2: Instrução SQL que guarda na tabela em questão dados referentes ao acesso da encomenda

```
switch (nivel_isolamento)
{
    case "Read Uncommitted":
        isolationLevel = IsolationLevel.ReadUncommitted;
        break;

    case "Read Committed":
        isolationLevel = IsolationLevel.ReadCommitted;
        break;

    case "Repeatable Read":
        isolationLevel = IsolationLevel.RepeatableRead;
        break;

    case "Serializable":
        isolationLevel = IsolationLevel.Serializable;
        break;
    default:
        isolationLevel = IsolationLevel.ReadUncommitted;
```

```
        break;
    }
    Debug.WriteLine("Attempting to open the connection.");

    sqlConnection.Open(); // Attempt to open the connection

    if (sqlConnection.State == ConnectionState.Open)
    {
        // Connection was successfully opened
        Debug.WriteLine("Connection successfully opened.");

        using (transaction = sqlConnection.BeginTransaction(isolationLevel)
        )
    }
```

Excerto de Código 3.3: Identificação do nível de isolamento escolhido e início da transação com esse nível.

```
string query = "SELECT * FROM EncLinha FULL JOIN Encomenda ON EncLinha.
    EncId = Encomenda.EncID WHERE EncLinha.EncId = " + Form2.IdEnc + ";";
;
Debug.WriteLine("SQL Query: " + query);

if (sqlConnection.ConnectionString != FormLogin.connectionString)
{
    sqlConnection.ConnectionString = FormLogin.connectionString;
}

SqlDataAdapter sqlDataAdapter = new SqlDataAdapter(query, sqlConnection)
;
var dataTable = new DataTable();
sqlDataAdapter.Fill(dataTable);
dataGridView1.DataSource = dataTable;
```

Excerto de Código 3.4: Pesquisar encomenda (com produtos associados) e imprimir valores na tabela.

```
// Executa a consulta SQL para atualizar a morada na tabela Encomenda
string queryMorada = "UPDATE Encomenda SET Morada = @Morada WHERE EncId
    = @EncId";
SqlCommand commandMorada = new SqlCommand(queryMorada, connection,
    transaction);
commandMorada.Parameters.AddWithValue("@Morada", morada);
commandMorada.Parameters.AddWithValue("@EncId", Form2.IdEnc);

// Alterar a Quantidade
// Executa a consulta SQL para atualizar a quantidade na tabela EncLinha
string query = "UPDATE EncLinha SET Qtd = " + quantidade + " WHERE EncId
    = " + Form2.IdEnc + " AND ProdutoId = " + produtoId + ";";
```

```
command.CommandText = query;  
command.ExecuteNonQuery();
```

Excerto de Código 3.5: Queries para alterar a morada da encomenda e quantidade de determinado produto na mesma.

Aplicação Browser

A aplicação *Browser* procura visualizar as encomendas e as suas linhas. A janela desta aplicação têm duas grelhas, uma para as encomendas e a outra com os produtos (linhas) da encomenda selecionada. Possui os dados ordenados por ordem decrescente do número de encomenda.

Esta foi implementada de forma relativamente simples. Após o login são apresentadas 2 tabelas ao utilizador. A primeira mostra a tabela com todas as encomendas, e a segunda mostra os vários produtos que se encontram nessa encomenda. Para isto é realizada uma query á base de dados para obter o conteúdo da tabela com as encomendas. Ao preencher a tabela, se o utilizador selecionar uma dada encomenda, uma segunda query será realizada, e a segunda tabela será preenchida com so dados dessa encomenda. A atualização das tabelas pode ser realizada de 2 formas. O auto-update opcional, que é realizado após um intervalo de tempo selecionado pelo utlizador, ou um botão que atualiza os dados uma única vez.

Aplicação “Log Tempo”

A aplicação “Log Tempo” desenvolvida, tem como objetivo de mostrar as operações do tipo "outro" (EventType = 'O'), e os campos "UserId", "EncId" e "Tempo", onde "Tempo" corresponde ao instante em que a encomenda foi editada pelo utilizador. A *data grid*, onde a informação é exposta ao utilizador, possui um timer que executa um *refresh* da grelha, mostrando os valores previamente referidos, mais atualizados disponíveis na base de dados.

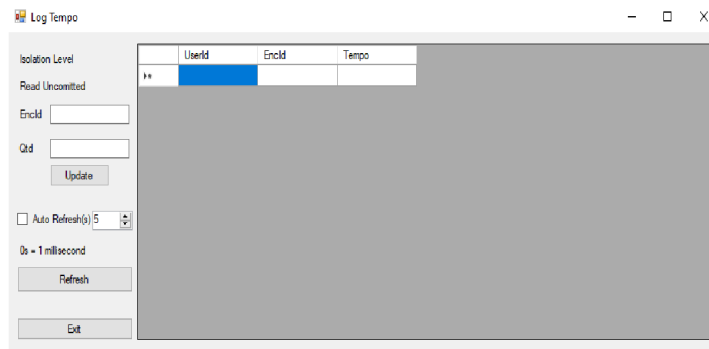


Figura 3.4: Imagem ilustrativa da aplicação Log Tempo antes de uma alteração.

Aplicação Log

O objetivo da aplicação "Log" é exibir os eventos mais recentes da tabela *LogOperations*, especificamente aqueles com os tipos de evento "I" (Inserção), "U" (Atualização) e "D" (Eliminação). A aplicação possui uma janela principal com uma grelha (tabela) que mostra esses eventos.

Para imprimir os resultados na tabela foi utilizado o *query* ilustrado no excerto de código seguinte 3.6

```
string query = "SELECT TOP " + numLinhas + " * FROM LogOperations WHERE
                EventType = 'I' OR EventType = 'U' OR EventType = 'D' ORDER BY
                DCriacao DESC;";
```

Excerto de Código 3.6: Query SQL para a leitura de todos os dados das primeiras linhas da tabela *LogOperations* cujo evento é do tipo I/U/D.

A variável *numLinhas* é o valor numérico que representa o número de linhas mais recentes que o utilizador deseja consultar da tabela *LogOperations*.

Além disso, a aplicação inclui um *timer* para atualização automática dos eventos na grelha (que ocorre a cada cinco segundos).

3.3 Testes

Para testar a Aplicação *Edit* e a Aplicação *Browser* foi executado na base de dados o *script "DoWork.sql"*, disponibilizado pelo professor na página da unidade curricular. Este *script* permite realizar uma simulação de vários utilizadores a operarem concorrentemente, o que permite verificar se a nossa aplicação está apta para concorrência.

Aplicação Edit

Teste: Alterar Dados da Encomenda - Nivel de Isolamento: *Uncommitted*

1. Colocar *script* a correr com a percentagem de atualizar a 100%, para não correremos o risco de (na aplicação) tentar entrar em encomendas eliminadas pelo *script*. Inserir o ID da encomenda a modificar, neste caso é 2. Ilustrado na figura 3.5.

Form2

ID Encomenda: 2

Nivel de Isolação: Read Uncommitted

Aceder

Figura 3.5: Imagem ilustrativa da Aplicacao Edit em Teste.

2. Mudar os dados da encomenda. (Ilustrado na figura 3.6)

Form3

Morada: Rua Santo António IV

ProdutoID: 111

Quantidade: 3

Submiter

Cancel

Encid	ProdutoID	Designacao	Preço	Qtd	EncID1	ClienteID	Nome	Morada
2	111	Mensagem	2500,00	12,00	2	1000	Fernando Pessoa	Rua Bento 16
2	131	Livro do Desas...	3000,00	1,00	2	1000	Fernando Pessoa	Rua Bento 16

Figura 3.6: Imagem ilustrativa da Aplicacao Edit em Teste.

3. *Submeter novos dados*: Aparece uma *DialogBox* a confirmar a mudança de dados (quando dá sucesso ou erro) e atualiza-se o *datagrid*, ilustrado na figura 3.7.

EncId	ProdutoId	Designacao	Preço	Qtd	EncID1	ClientID	Nome	Morada
2	111	Mensagem	2500,00	3,00	2	1000	Fernando Pessoa	Rua Santo Antõ...
2	131	Livro de Desass...	3000,00	1,00	2	1000	Fernando Pessoa	Rua Santo Antõ...

Figura 3.7: Imagem ilustrativa da Aplicacao Edit em Teste.

Como foi possível editar os dados da encomenda enquanto corria o *script* que simulava outro utilizador na base de dados, concluiu-se que a nossa aplicação está apta para concorrência.

Aplicação Browser

A aplicação correu como esperado, sem qualquer problema, no entanto, é de notar que definir o tempo de auto atualização para valores muito inferiores, como um milissegundo, causa uma grande perda de *performance* em computadores com menos poder de processamento. Além disso, ao testar ligações entre dispositivos diferentes descobriu-se que se houvesse perda de ligação entre os dispositivos a aplicação *crashava*.

Capítulo 4

Conclusão

4.1 Conclusões Principais

Com a realização deste trabalho prático, de Controlo de Transações no âmbito da unidade curricular de Sistemas de Gestão de Bases de Dados (SGBD), foi possível concluir a importância da gestão e controlo das transações a ocorrer sobre um sistema de gestão de bases de dados.

As aplicações demonstraram-nos a importância da gestão e controlo de transações, bem como a visualização de fenómenos como *locks* e *deadlocks*, sendo estes um fator crítico durante a execução de transações concorrentes. Concluímos que os níveis de isolamento têm impacto real sobre as transações em aplicações concretas e que o controlo de transações pode ser utilizados para evitar, ou causar fenómenos como *locks* e *deadlocks*.

4.2 Sugestões de Melhoria

As aplicações desenvolvidas encontram-se finalizadas, mas seria necessário uma maior exploração do impacto do controlo de transações, sendo este um trabalho realizado no início do semestre e como tal, não tendo lecionado a matéria teórica necessária para completar todo o trabalho, no final do tempo dedicado para a concretização do mesmo, conclui-se que ainda é necessário explorar com mais profundidade o impacto real do controlo de transações em aplicações concretas.

Bibliografia

- [1] J. Muranho H. Proença P. Prata R. Cardoso. Notas de apoio às aulas, 2022. [Online] https://moodle.ubi.pt/pluginfile.php/678733/mod_resource/content/2/BD-Notas%20de%20Apoio%20%C3%A0s%20Aulas%20v5.7.pdf. Último acesso a 6 de Novembro de 2023.
- [2] Johannes Gehrke and Raghu Ramakrishnan. *Database Management Systems*. 1996.
- [3] Pedro R. M. Inácio. Aula 9, 2013. [Online] https://moodle.ubi.pt/pluginfile.php/678732/mod_resource/content/1/Sebenta-BD-II.pdf. Último acesso a 7 de Novembro de 2023.