

Trabalho Experimental 1

Engenharia Informática
Técnicas Avançadas de Bases de Dados
António Marques
Paulo Martins

Autores

João Lomar – A159991

Marcelo Pinto – A160102

Tomás Azevedo – A160979

Vila Real, 2017

Resumo

A definição de políticas de segurança e acesso aos dados é de extrema importância para a segurança do primeiro nível de acesso à base de dados. O estudo deste tema é importante na medida em que é de grande utilidade e maior segurança para uma aplicação web ter este tipo de implementações ao nível da base de dados. Neste trabalho pretendemos definir essas políticas de segurança e além disso, apresentamos soluções para problemas de concorrência em sistemas de gestão de bases de dados centralizadas.

Índice

1. Introdução	5
2. Enquadramento Teórico.....	6
2.1.SQL SERVER	6
2.2. Exemplo de como criar um LOGIN	6
2.3. Exemplo de como criar um USER associado a um LOGIN.....	6
2.3.ROLES.....	6
2.4.Criar role.....	7
2.4.1.Adicionar users ao Role criado.....	7
2.5.Comandos usados nos Roles.....	7
2.6. Nível de Isolamento das transações.....	7
3. Objetivos da Etapa do Trabalho Prático	8
4. Resultados Atingidos	9
4.1. Políticas de segurança e acesso aos dados.....	9
4.2.1. Criar Autor	12
4.2.2. Criar Categoria.....	12
4.2.3. Criar Loja	13
4.2.4. Editar Autor	14
4.2.5. Editar Categoria	15
4.2.6. Editar Loja	16
4.2.7. Criar Relação Livro-Categoria.....	16
4.2.8. Criar Relação Livro-Autor	17
4.2.9. Criar Relação Livro-Loja	18
4.2.10. Apagar Relação Livro-Categoria	19
4.2.11. Apagar Relação Livro-Autor	20
4.2.12. Apagar Relação Livro-Loja	20

4.2.13.	Editar Disponível	21
4.2.14.	Criar Livro	22
4.2.15.	Editar Livro	23
4.2.16.	Ocultar/Não Ocultar Livro	24
4.2.17.	Criar Leu	25
4.2.18.	Criar Possui	26
4.2.19.	Criar Pedido	27
4.2.20.	Apagar Leu.....	28
4.2.21.	Apagar Possui	29
4.2.22.	Cancelar Pedido	29
4.2.23.	Editar Leu.....	30
4.2.24.	Editar Possui	31
4.2.25.	Editar Pedido.....	32
4.2.26.	Criar Empréstimo	33
4.2.27.	Devolver Livro	34
4.2.28.	Registrar Utilizador	36
4.2.29.	Editar Utilizador.....	37
4.2.30.	Bloquear Utilizador.....	38
4.2.31.	Desbloquear Utilizador	39
4.2.32.	Ativar Utilizador	40
4.2.33.	Editar Admin.....	41
5.	Análise e Discussão dos Resultados	43
6.	Bibliografia	44
7.	Anexos	45
7.1.	Diagrama da Base de Dados	45
7.2.	Código SQL.....	46

1. Introdução

No âmbito da Unidade Curricular de Técnicas Avançadas de Bases de Dados foi proposto um trabalho prático com o objetivo de definir políticas de segurança e acesso aos dados e resolver problemas de concorrência em sistemas de gestão de bases de dados centralizadas numa aplicação Web que permita suportar o funcionamento de uma biblioteca pessoal. O estudo destes temas é importante na medida em que é de grande utilidade e maior segurança para uma aplicação web ter este tipo de implementações ao nível da base de dados.

Procurou-se dividir o relatório em duas fases: a primeira, em que se fornecemos um enquadramento teórico sobre as políticas de acesso aos dados, o conceito de transações em Bases de Dados e os seus diferentes níveis de isolamento; a segunda, em que se contextualiza os conhecimentos e se explora a sua aplicabilidade ao exemplo prático.

Para a análise deste processo utilizou-se uma metodologia de investigação-ação, uma vez que se baseou essencialmente na análise da Base de Dados fornecida e da descrição do protocolo, e a partir dos nossos pontos de vista das tarefas pretendidas para a aplicação web foram definidas as políticas de segurança e acesso aos dados e resolvidos os problemas de concorrência em sistemas de gestão de bases de dados centralizadas.

A ferramenta de trabalho utilizado foi o Microsoft SQL Server 2014.

2. Enquadramento Teórico

2.1.SQL SERVER

Como vamos usar o MS SQL Server neste trabalho é importante ter, no mínimo, uma noção básica das suas funcionalidades e saber usá-las. Para isso, neste pequeno enquadramento teórico vamos tentar explicar o fundamental de uma forma sucinta e de forma a que se entenda bem o conteúdo a ser explorado.

O MS SQL Server apresenta dois domínios de segurança que são o servidor e a base de dados. Todos os processos são executados na base de dados mas para podermos aceder à mesma temos de aceder anteriormente ao servidor. Para tal ser possível, temos de usar os logins sendo que existem dois tipos deles, os autenticação do MS SQL Server e os por autenticação do Windows. Conseguida a entrada no servidor estamos a um passo de entrar na base de dados. Para entrar na base de dados existem os users que estão associados ao respetivo login que foi feito no passo anterior. Através dos users acedemos à base de dados e estamos prontos para começar a executar queries.

2.2. Exemplo de como criar um LOGIN

```
CREATE LOGIN ClienteLog WITH PASSWORD='123456'
```

2.3. Exemplo de como criar um USER associado a um LOGIN

```
CREATE USER ClienteU FOR LOGIN ClienteLog  
CREATE USER ClienteA FOR LOGIN ClienteLog
```

2.3.ROLES

De maneira a simplificar os utilizadores, criam-se roles. O role não é nada mais nada menos do que a criação de grupos nos diversos utilizadores que fazem as mesmas tarefas. Por exemplo, nos websites existem os utilizadores comuns, que visitam o site e usam para seu próprio benefício, e os administradores, que fazem a gestão do site. Estes últimos, que gerem o site tem mais funcionalidades do que os utilizadores comuns. Então, para não andarmos a tirar permissões aos utilizadores comuns um a um, associamos todos os utilizadores a um role “RoleUserComum” e seleccionamos as permissões que eles podem exercer.

2.4.Criar role

```
CREATE ROLE RoleUserComum
```

2.4.1.Adicionar users ao Role criado

```
ALTER ROLE RoleUserComum ADD MEMBER ClienteU
```

```
ALTER ROLE RoleUserComum ADD MEMBER ClienteA
```

2.5.Comandos usados nos Roles

GRANT -> adiciona uma ou mais permissões.

Exemplo: Permissão do role RoleUserComum poder ver todas as tabelas.

```
GRANT SELECT TO RoleUserComum
```

DENY -> retira uma ou mais permissões

Exemplo: Remove Permissão do role RoleUserComum de editar o saldo na tabela Contas.

```
DENY UPDATE UPDATE ON Contas(saldo) TO RoleUserComum
```

REVOKE -> remove uma regra existente atribuída por GRANT ou DENY

Exemplo: Remove a permissão dada anteriormente ao RoleUserComum de poder ver todas as tabelas.

```
REVOKE SELECT FROM RoleUserComum
```

2.6. Nível de Isolamento das transações...

...tem a ver com a forma de como é gerido o mecanismo de controlo de concorrência.

Dirty read -> É quando uma conexão pode ler informação que ainda não foi “comitada”.

Non repeatable read -> É quando na execução de uma transação se pode ler a informação mais do que uma vez diferente.

Phantom Read -> É quando na execução de uma transação podem ser inseridos/eliminados registros.

Nível de Isolamento	Dirty Read	Non repeatable Read	Phantom Read
Read Uncommitted	Sim	Sim	Sim
Read Committed(Por defeito)	Não	Sim	Sim
Repeatable Read	Não	Não	Sim
Serializable	Não	Não	Não

Read uncommitted: é o nível de isolamento mais baixo e que permite a leitura de dados não submetidos (committed), este tipo de leitura é designado de “Dirty Read”.

Read committed: é um nível de isolamento que garante a leitura dos dados foi submetida (committed) no momento em que é lida. Mas não garante a integridade da informação, pois os dados podem ser alterados depois de serem lidos mesmo não tendo acabado a transação.

Repeatable read: é um nível de isolamento que garante o nível de “Read Committed” e também que qualquer dado lido não pode ser alterado durante aquela transação, ou seja, durante uma transação deste nível não podem ser feitos UPDATES à tabela lida. Este nível de isolamento garante integridade ao contrário do read committed

Serializable: é o nível de isolamento mais alto, sendo que garante o nível de “Repeatable Read” e que nenhuma outra transação possa ocorrer às tabelas durante o decorrer desta. Em caso de dúvida devemos usar este nível de isolamento.

3. Objetivos da Etapa do Trabalho Prático

- ✓ Definir políticas de segurança e acesso aos dados;
- ✓ Resolver problemas de concorrência em sistemas de gestão de bases de dados centralizadas.

4. Resultados Atingidos

Após análise e discussão dos requisitos pretendidos para a aplicação web proposta, tendo como base o protocolo e a base de dados fornecida, chegamos aos resultados apresentados nesta etapa.

4.1. Políticas de segurança e acesso aos dados

Para a definição das políticas de segurança e acesso aos dados, idealizamos três atores: o Administrador, o Utilizador e o Visitante. De seguida apresentamos uma descrição das tarefas CRUD que os atores poderão fazer nas tabelas:

1. Administrador:

- a. Poderá criar, atualizar e visualizar as tabelas Autor, Categoria, Loja e Livro mas não poderá apaga-las, uma vez que a lógica do site poderia ficar comprometida (Exemplo: Apagando um Autor, os livros do mesmo teriam de ser apagados ou ficar sem o autor associado);
- b. Poderá criar, atualizar, visualizar e apagar nas relações Escreve, Pertence, Disponível, uma vez que a gestão destas relações deve ser feita com total liberdade pelo Administrador;
- c. Poderá apenas visualizar as tabelas Leu, Possui, Pede e Empresta uma vez que estas relações são totalmente geridas pelos utilizadores e o Admin não poderá alterar esta lógica;
- d. Por fim, o Administrador poderá visualizar os utilizadores existentes assim como atualizar esta tabela, uma vez que o estado do utilizador poderá ser mudado pelo Administrador no ato de bloquear/desbloquear.

2. Utilizador:

- a. Poderá apenas visualizar as tabelas Autor, Escreve, Categoria, Pertence, Loja, Disponível e Livro, uma vez que a gestão destas tabelas é feita pelo Administrador;
- b. Poderá criar, atualizar, visualizar e apagar nas tabelas Leu, Possui, Pede e Empresta, uma vez que a gestão destas tabelas é feita pelos utilizadores;
- c. A tabela Utilizador pode ser atualizada pelo mesmo de forma a mudar alguns dos seus dados assim como poderá visualizar os seus dados;

- d. Nas tabelas Bloqueia e Administrador, o Utilizador não terá qualquer tipo de acesso.

3. Visitante:

- a. Poderá visualizar as tabelas Autor, Escreve, Categoria, Pertence e Livro de forma a poder ver a página inicial do site normalmente assim como algumas informações de cada livro;
- b. Não terá acesso às tabelas Loja e Disponível uma vez que terá de se registar para poder ter acesso aos links das lojas;
- c. Não terá acesso às tabelas Leu, Possui, Pede e Empresta. Terá de se registar para poder ter uma biblioteca pessoal e emprestar/receber livros.
- d. Poderá criar um registo na tabela Utilizador no ato de registo no site. Não terá acesso à tabela Bloqueia e Administrador.

De seguida, apresentamos a tabela 1 que contém o resumo das tarefas CRUD que cada ator poderá assumir nas tabelas:

Atores Tabelas	Administrador	Utilizador	Visitante
Autor	INSERT UPDATE SELECT	SELECT	SELECT
Escreve	INSERT UPDATE SELECT DELETE	SELECT	SELECT
Categoria	INSERT UPDATE SELECT	SELECT	SELECT
Pertence	INSERT UPDATE SELECT DELETE	SELECT	SELECT
Loja	INSERT UPDATE SELECT	SELECT	N/D
Disponível	INSERT UPDATE	SELECT	N/D

	SELECT DELETE		
Livro	INSERT UPDATE SELECT	SELECT	SELECT
Leu	SELECT	INSERT UPDATE SELECT DELETE	N/D
Possui	SELECT	INSERT UPDATE SELECT DELETE	N/D
Pede	SELECT	INSERT UPDATE SELECT DELETE	N/D
Empresta	SELECT	INSERT UPDATE SELECT DELETE	N/D
Utilizador	UPDATE SELECT	UPDATE SELECT	INSERT
Bloqueia	INSERT UPDATE SELECT	N/D	N/D
Administrador	UPDATE SELECT	N/D	N/D

Tabela 1: Permissões de Acesso às tabelas por parte dos atores.

4.2. Transações

Pretendemos resolver problemas de concorrência em sistemas de gestão de bases de dados centralizadas com recurso a transações. Serão também utilizados alguns procedimentos que não sendo transações, complementam a base de dados. De seguida apresentamos a explicação de cada transação, o seu código e a explicação para a escolha do nível de isolamento.

4.2.1. Criar Autor

Nível de isolamento escolhido: Serializable; Uma vez que esta transação lida com a criação de uma nova linha numa tabela, originando-se um novo ID, todo o tipo de leituras deve ser bloqueado até que esteja confirmada a inserção na tabela. Caso não acontecesse, poderia ser lido este ID, criado outro autor e caso houvesse um erro, a aplicação poderia entrar em conflito uma vez que foi já teria sido criado outro autor. Assim, não será permitida *Dirty Read*, *Non repeatable read* nem *Phanton read*. Como visto no enquadramento teórico, o nível de isolamento que bloqueia estas leituras é o Serializable e, portanto, é o escolhido para esta transação.

Transação: Nesta transação é apenas criado um autor, recebendo os parâmetros necessários. Em caso de erro é retornado -1 e em caso de sucesso é retornado 1.

Código da transação:

```
CREATE PROCEDURE CriarAutor
@Nome VARCHAR(60),
@Pseudonimo VARCHAR(60),
@Biografia VARCHAR(1000)
AS
SET TRANSACTION ISOLATION LEVEL Serializable
BEGIN TRANSACTION

INSERT INTO Autor(Nome, Pseudonimo, Biografia) VALUES (@Nome,
@Pseudonimo, @Biografia)--Criar registo para um novo autor

IF (@@ERROR <> 0)
begin
    rollback
    RETURN -1
end

COMMIT
RETURN 1

GO
```

4.2.2. Criar Categoria

Nível de isolamento escolhido: Serializable; Uma vez que esta transação lida com a criação de uma nova linha numa tabela, originando-se um novo ID, todo o tipo de leituras deve ser bloqueado até que esteja confirmada a inserção na tabela. Caso não acontecesse, poderia ser lido este ID, criado outro autor e caso houvesse um erro, a aplicação poderia entrar em conflito uma vez que foi já teria sido criado outro autor. Assim, não será permitida *Dirty Read*, *Non repeatable read* nem *Phanton read*. Como visto no enquadramento teórico, o nível de isolamento que bloqueia estas leituras é o Serializable e, portanto, é o escolhido para esta transação.

Transação: Nesta transação é apenas criada uma categoria, recebendo os parâmetros necessários. Em caso de erro é retornado -1 e em caso de sucesso é retornado 1.

Código da transação:

```
CREATE PROCEDURE CriarCategoria
@Nome VARCHAR(60)
AS
SET TRANSACTION ISOLATION LEVEL Serializable
BEGIN TRANSACTION

    INSERT INTO Categoria(Nome) VALUES (@Nome) -- Criar registo para uma
nova Categoria

    IF (@@ERROR <> 0)
    begin
        rollback
        RETURN -1
    end

    COMMIT
    RETURN 1
GO
```

4.2.3. Criar Loja

Nível de isolamento escolhido: Serializable; Uma vez que esta transação lida com a criação de uma nova linha numa tabela, originando-se um novo ID, todo o tipo de leituras deve ser bloqueado até que esteja confirmada a inserção na tabela. Caso não acontecesse, poderia ser lido este ID, criado outro autor e caso houvesse um erro, a aplicação poderia entrar em conflito uma vez que foi já teria sido criado outro autor. Assim, não será permitida *Dirty Read*, *Non repeatable read* nem *Phanton read*. Como visto no enquadramento teórico, o nível de isolamento que bloqueia estas leituras é o Serializable e, portanto, é o escolhido para esta transação.

Transação: Nesta transação é apenas criada uma loja, recebendo os parâmetros necessários. Em caso de erro é retornado -1 e em caso de sucesso é retornado 1.

Código da transação:

```
CREATE PROCEDURE CriarLoja
@Nome VARCHAR(60)
AS
SET TRANSACTION ISOLATION LEVEL Serializable
BEGIN TRANSACTION

    INSERT INTO Loja(Nome) VALUES (@Nome) -- Criar registo para uma nova
Loja
```

```

        IF (@@ERROR <> 0)
        begin
            rollback
            RETURN -1
        end

        COMMIT
        RETURN 1
    GO

```

4.2.4. Editar Autor

Nível de isolamento escolhido: Read committed; A transação lida com a edição de uma linha na tabela, e portanto, a *Dirty Read* não deve ser permitida uma vez que não deve ser possível ler as informações que ainda não foram confirmadas como editadas. No entanto, a *Non-Repeatable Read* deve ser possível porque uma pode-se ler a informação mais do que uma vez diferente e a *Phanton Read* também deve ser permitida uma vez que a edição de uma linha não interfere com a possibilidade de serem inseridas ou apagadas outras linhas. Assim, o nível de isolamento escolhido é o *Read Committed*, uma vez que permite *Non-Repeatable Read* e *Phanton Read* e não permite *Dirty Read*.

Transação: A transação lida com a edição de um autor, apresentando o respetivo update. São passados como parâmetros todos os dados do autor, sendo assim todos os dados novamente submetidos. Em caso de erro é retornado -1 e em caso de sucesso é retornado 1.

Código da transação:

```

CREATE PROCEDURE EditarAutor
@ID_Autor INT,
@Nome VARCHAR(60),
@Pseudonimo VARCHAR(60),
@Biografia VARCHAR(1000)
AS

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION

    UPDATE Autor
    SET Nome = @Nome, Pseudonimo = @Pseudonimo, Biografia = @Biografia
    WHERE ID_Autor = @ID_Autor

    IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
    begin
        rollback
        RETURN -1
    end

```

```

COMMIT
RETURN 1
GO

```

4.2.5. Editar Categoria

Nível de isolamento escolhido: Read committed; A transação lida com a edição de uma linha na tabela, e portanto, a *Dirty Read* não deve ser permitida uma vez que não deve ser possível ler as informações que ainda não foram confirmadas como editadas. No entanto, a *Non-Repeatable Read* deve ser possível porque uma pode-se ler a informação mais do que uma vez diferente e a *Phanton Read* também deve ser permitida uma vez que a edição de uma linha não interfere com a possibilidade de serem inseridas ou apagadas outras linhas. Assim, o nível de isolamento escolhido é o *Read Committed*, uma vez que permite *Non-Repeatable Read* e *Phanton Read* e não permite *Dirty Read*.

Transação: A transação lida com a edição de uma categoria, apresentando o respetivo update, atualizando o nome. Em caso de erro é retornado -1 e em caso de sucesso é retornado 1.

Código da transação:

```

CREATE PROCEDURE EditarCategoria
@ID_Categoria INT,
@Nome VARCHAR(60)
AS

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION

    UPDATE Categoria
    SET Nome = @Nome
    WHERE ID_Categoria = @ID_Categoria

    IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
    begin
        rollback
        RETURN -1
    end

    COMMIT
    RETURN 1
GO

```

4.2.6. Editar Loja

Nível de isolamento escolhido: Read committed; A transação lida com a edição de uma linha na tabela, e portanto, a *Dirty Read* não deve ser permitida uma vez que não deve ser possível ler as informações que ainda não foram confirmadas como editadas. No entanto, a *Non-Repeatable Read* deve ser possível porque uma pode-se ler a informação mais do que uma vez diferente e a *Phanton Read* também deve ser permitida uma vez que a edição de uma linha não interfere com a possibilidade de serem inseridas ou apagadas outras linhas. Assim, o nível de isolamento escolhido é o *Read Committed*, uma vez que permite *Non-Repeatable Read* e *Phanton Read* e não permite *Dirty Read*.

Transação: A transação lida com a edição de uma loja, apresentando o respetido update, recebendo o novo nome. Em caso de erro é retornado -1 e em caso de sucesso é retornado 1.

Código da transação:

```
CREATE PROCEDURE EditarLoja
@ID_Loja INT,
@Nome VARCHAR(60)
AS

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION

    UPDATE Loja
    SET Nome = @Nome
    WHERE ID_Loja = @ID_Loja

    IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
    begin
        rollback
        RETURN -1
    end

    COMMIT
    RETURN 1
GO
```

4.2.7. Criar Relação Livro-Categoria

Nível de isolamento escolhido: Serializable; Uma vez que esta transação lida com a criação de uma nova linha numa tabela, originando-se um novo ID, todo o tipo de leituras deve ser bloqueado até que esteja confirmada a inserção na tabela. Caso não acontecesse, poderia ser lido este ID, criado outro autor e caso houvesse um erro, a aplicação poderia entrar em conflito uma

vez que foi já teria sido criado outro autor. Assim, não será permitida *Dirty Read*, *Non repeatable read* nem *Phantom read*. Como visto no enquadramento teórico, o nível de isolamento que bloqueia estas leituras é o Serializable e, portanto, é o escolhido para esta transação.

Transação: Nesta transação é criada a relação Pertence, para associar um livro a uma categoria (pode ser associado a várias), recebendo os parâmetros necessários. Em caso de erro é retornado -1 e em caso de sucesso é retornado 1.

Código da transação:

```
CREATE PROCEDURE CriarLivroCategoria
@ID_Categoria INTEGER,
@ID_Livro INTEGER
AS
SET TRANSACTION ISOLATION LEVEL Serializable

BEGIN TRANSACTION

INSERT INTO Pertence(ID_Categoria, ID_Livro) VALUES (@ID_Categoria,
@ID_Livro)

IF (@@ERROR <> 0)
begin
    rollback
    RETURN -1
end

COMMIT
RETURN 1

GO
```

4.2.8. Criar Relação Livro-Autor

Nível de isolamento escolhido: Serializable; Uma vez que esta transação lida com a criação de uma nova linha numa tabela, originando-se um novo ID, todo o tipo de leituras deve ser bloqueado até que esteja confirmada a inserção na tabela. Caso não acontecesse, poderia ser lido este ID, criado outro autor e caso houvesse um erro, a aplicação poderia entrar em conflito uma vez que foi já teria sido criado outro autor. Assim, não será permitida *Dirty Read*, *Non repeatable read* nem *Phantom read*. Como visto no enquadramento teórico, o nível de isolamento que bloqueia estas leituras é o Serializable e, portanto, é o escolhido para esta transação.

Transação: Nesta transação é criada a relação Escreve, associando um livro a um autor (pode ser associado a vários autores), recebendo os parâmetros necessários. Em caso de erro é retornado -1 e em caso de sucesso é retornado 1.

Código da transação:

```
CREATE PROCEDURE CriarLivroAutor
@ID_Autor INTEGER,
@ID_Livro INTEGER
AS
SET TRANSACTION ISOLATION LEVEL Serializable

BEGIN TRANSACTION

INSERT INTO Escreve(ID_Autor, ID_Livro) VALUES (@ID_Autor, @ID_Livro)

IF (@@ERROR <> 0)
begin
    rollback
    RETURN -1
end

COMMIT
RETURN 1

GO
```

4.2.9. Criar Relação Livro-Loja

Nível de isolamento escolhido: Serializable; Uma vez que esta transação lida com a criação de uma nova linha numa tabela, originando-se um novo ID, todo o tipo de leituras deve ser bloqueado até que esteja confirmada a inserção na tabela. Caso não acontecesse, poderia ser lido este ID, criado outro autor e caso houvesse um erro, a aplicação poderia entrar em conflito uma vez que foi já teria sido criado outro autor. Assim, não será permitida *Dirty Read*, *Non repeatable read* nem *Phantom read*. Como visto no enquadramento teórico, o nível de isolamento que bloqueia estas leituras é o Serializable e, portanto, é o escolhido para esta transação.

Transação: Nesta transação é criada a relação Disponível, associando um livro a uma loja (pode ser associado a várias lojas), recebendo os parâmetros necessários. Em caso de erro é retornado -1 e em caso de sucesso é retornado 1.

Código da transação:

```
CREATE PROCEDURE CriarLivroLoja
@ID_Loja INTEGER,
@ID_Livro INTEGER,
@Link VARCHAR(300)
AS
SET TRANSACTION ISOLATION LEVEL Serializable

BEGIN TRANSACTION

INSERT INTO Disponivel(ID_Loja, ID_Livro, Link) VALUES (@ID_Loja,
@ID_Livro, @Link)
```

```

        IF (@@ERROR <> 0)
        begin
            rollback
            RETURN -1
        end

        COMMIT
        RETURN 1
GO

```

4.2.10. Apagar Relação Livro-Categoria

Nível de isolamento escolhido: Serializable; É apagado um registo na tabela, e portanto, a operação Phantom Read não pode ser permitida porque não devem ser inseridos novos registos enquanto não se confirmar que o registo desta transação foi apagado. Caso contrário, a aplicação poderia gerar um novo ID que teria o mesmo que o registo da transação que deveria ser apagado. Em caso de erro, os 2 registos teriam o mesmo ID. Assim sendo, o único nível de isolamento que não permite Phantom Read é o Serializable e por isso é o escolhido.

Transação: Nesta transação é apagada a relação Livro-Categoria (Pertence), recebendo as chaves primárias. Em caso de erro é retornado -1 e em caso de sucesso é retornado 1.

Código da transação:

```

CREATE PROCEDURE ApagarLivroCategoria
@ID_Categoria INTEGER,
@ID_Livro INTEGER
AS

SET TRANSACTION ISOLATION LEVEL Serializable

BEGIN TRANSACTION

Delete from Pertence
where ID_Categoria = @ID_Categoria and ID_Livro = @ID_Livro

IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
begin
    rollback
    RETURN -1
end

COMMIT
RETURN 1
GO

```

4.2.11. Apagar Relação Livro-Autor

Nível de isolamento escolhido: Serializable; É apagado um registo na tabela, e portanto, a operação Phantom Read não pode ser permitida porque não devem ser inseridos novos registos enquanto não se confirmar que o registo desta transação foi apagado. Caso contrário, a aplicação poderia gerar um novo ID que teria o mesmo que o registo da transação que deveria ser apagado. Em caso de erro, os 2 registos teriam o mesmo ID. Assim sendo, o único nível de isolamento que não permite Phantom Read é o Serializable e por isso é o escolhido.

Transação: Nesta transação é apagada a relação Livro-Autor (Escreve), recebendo as chaves primárias. Em caso de erro é retornado -1 e em caso de sucesso é retornado 1.

Código da transação:

```
CREATE PROCEDURE ApagarLivroAutor
@ID_Autor INTEGER,
@ID_Livro INTEGER
AS

SET TRANSACTION ISOLATION LEVEL Serializable

BEGIN TRANSACTION

Delete from Escreve
where ID_Autor = @ID_Autor and ID_Livro = @ID_Livro

IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
begin
    rollback
    RETURN -1
end

COMMIT
RETURN 1

GO
```

4.2.12. Apagar Relação Livro-Loja

Nível de isolamento escolhido: Serializable; É apagado um registo na tabela, e portanto, a operação Phantom Read não pode ser permitida porque não devem ser inseridos novos registos enquanto não se confirmar que o registo desta transação foi apagado. Caso contrário, a aplicação poderia gerar um novo ID que teria o mesmo que o registo da transação que deveria ser apagado. Em caso de erro, os 2 registos teriam o mesmo ID. Assim sendo, o único nível de isolamento que não permite Phantom Read é o Serializable e por isso é o escolhido.

Transação: Nesta transação é apagada a relação Livro-Loja (Disponível), recebendo as chaves primárias. Em caso de erro é retornado -1 e em caso de sucesso é retornado 1.

Código da transação:

```
CREATE PROCEDURE ApagarLivroLoja
@ID_Loja INTEGER,
@ID_Livro INTEGER
AS

SET TRANSACTION ISOLATION LEVEL Serializable

BEGIN TRANSACTION

Delete from Disponivel
where ID_Loja = @ID_Loja and ID_Livro = @ID_Livro

IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
begin
    rollback
    RETURN -1
end

COMMIT
RETURN 1

GO
```

4.2.13. Editar Disponível

Nível de isolamento escolhido: Read committed; A transação lida com a edição de uma linha na tabela, e portanto, a *Dirty Read* não deve ser permitida uma vez que não deve ser possível ler as informações que ainda não foram confirmadas como editadas. No entanto, a *Non-Repeatable Read* deve ser possível porque uma pode-se ler a informação mais do que uma vez diferente e a *Phanton Read* também deve ser permitida uma vez que a edição de uma linha não interfere com a possibilidade de serem inseridas ou apagadas outras linhas. Assim, o nível de isolamento escolhido é o *Read Committed*, uma vez que permite *Non-Repeatable Read* e *Phanton Read* e não permite *Dirty Read*.

Transação: Nesta transação é editado o link da relação Disponível, recebendo as chaves primárias e link. Em caso de erro é retornado -1 e em caso de sucesso é retornado 1.

Código da transação:

```
CREATE PROCEDURE EditarDisponivel
@ID_Loja INTEGER,
@ID_Livro INTEGER,
@Link VARCHAR(300)
AS
```

```

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION

Update Disponivel
Set Link = @Link
Where ID_Loja = @ID_Loja and ID_Livro = @ID_Livro

IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
begin
    rollback
    RETURN -1
end

COMMIT
RETURN 1

GO

```

4.2.14. Criar Livro

Nível de isolamento escolhido: Serializable; Uma vez que esta transação lida com a criação de uma nova linha numa tabela, originando-se um novo ID, todo o tipo de leituras deve ser bloqueado até que esteja confirmada a inserção na tabela. Caso não acontecesse, poderia ser lido este ID, criado outro autor e caso houvesse um erro, a aplicação poderia entrar em conflito uma vez que foi já teria sido criado outro autor. Assim, não será permitida *Dirty Read*, *Non repeatable read* nem *Phanton read*. Como visto no enquadramento teórico, o nível de isolamento que bloqueia estas leituras é o Serializable e, portanto, é o escolhido para esta transação.

Transação: Nesta transação é criado um livro, recebendo todos os parâmetros necessários. São também recebidos o ID de uma loja, uma categoria e um autor para inserção da associação. Posteriormente, podem ser inseridos mais autores, categorias e lojas. Em caso de erro num dos procedimentos de inserção da categoria, loja ou autor, o livro deverá ser inserido na mesma. Em caso de não inserção do livro retorna -1; Em caso de inserção do livro com autor, loja e categoria retorna 1, e em caso de inserção do livro sem categoria, autor ou loja retorna 2.

Código da transação:

```

CREATE PROCEDURE CriarLivro
@ISBN VARCHAR(30),
@Titulo VARCHAR(50),
@Editora VARCHAR(50),
@Sinopse VARCHAR(MAX),
@EdicaoData DATE,
@ID_Categoria INTEGER,
@ID_Autor INTEGER,

```

```

@ID_Loja INTEGER,
@Link VARCHAR(300)
AS

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

BEGIN TRANSACTION

    INSERT INTO Livro(ISBN, Titulo, Editora, Sinopse, EdicaoData, Estado)
        VALUES (@ISBN, @Titulo, @Editora, @Sinopse, @EdicaoData, 1)

    IF (@@ERROR <> 0)
    begin
        rollback
        return -1
    end
    -- -1 -nao inseriu livro
    -- 1 - inseriu livro COM autor, loja e categoria
    -- 2 - inseriu livro SEM autor, loja ou categoria
    declare @valreturn int = 1
    EXEC CriarLivroCategoria @ID_Categoria, @@IDENTITY
    IF (@@ERROR <> 0)
    begin
        set @valreturn=2
    end

    EXEC CriarLivroAutor @ID_Autor, @@IDENTITY
    IF (@@ERROR <> 0)
    begin
        set @valreturn=2
    end

    EXEC CriarLivroLoja @ID_Loja, @@IDENTITY, @Link
    IF (@@ERROR <> 0)
    begin
        set @valreturn=2
    end

    RETURN @valreturn
    COMMIT
GO

```

4.2.15. Editar Livro

Nível de isolamento escolhido: Read committed; A transação lida com a edição de uma linha na tabela, e portanto, a *Dirty Read* não deve ser permitida uma vez que não deve ser possível ler as informações que ainda não foram confirmadas como editadas. No entanto, a *Non-Repeatable Read* deve ser possível porque uma pode-se ler a informação mais do que uma vez diferente e a *Phanton Read* também deve ser permitida uma vez que a edição de uma linha não interfere com a possibilidade de serem inseridas ou apagadas outras linhas. Assim, o nível de isolamento escolhido

é o *Read Committed*, uma vez que permite *Non-Repeatable Read* e *Phanton Read* e não permite *Dirty Read*.

Transação: Nesta transação acontece a edição de um livro, sendo passados todos os parâmetros. Em caso de erro é retornado -1 e em caso de sucesso 1.

Código da transação:

```
CREATE PROCEDURE EditarLivro
@ID_Livro INTEGER,
@ISBN VARCHAR(30),
@Titulo VARCHAR(50),
@Editora VARCHAR(50),
@Sinopse VARCHAR(MAX),
@EdicaoData DATE
AS

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION

    Update Livro
    Set ISBN = @ISBN , Titulo = @Titulo, Editora = @Editora, Sinopse =
@Sinopse, EdicaoData = @EdicaoData
    Where ID_Livro = @ID_Livro

    IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
    begin
        rollback
        RETURN -1
    end

    COMMIT
    RETURN 1

GO
```

4.2.16. Ocultar/Não Ocultar Livro

Nível de isolamento escolhido: Read committed; Acontece a modificação do estado do livro e assim sendo a transação lida com a edição de uma linha na tabela, e portanto, a *Dirty Read* não deve ser permitida uma vez que não deve ser possível ler as informações que ainda não foram confirmadas como editadas. No entanto, a *Non-Repeatable Read* deve ser possível porque uma pode-se ler a informação mais do que uma vez diferente e a *Phanton Read* também deve ser permitida uma vez que a edição de uma linha não interfere com a possibilidade de serem inseridas ou apagadas outras linhas. Assim, o nível de isolamento escolhido é o *Read Committed*, uma vez que permite *Non-Repeatable Read* e *Phanton Read* e não permite *Dirty Read*.

Transação: Nesta transação acontece a edição do estado do livro, sendo passado o mesmo. Em caso de erro é retornado -1 e em caso de sucesso 1.

Código da transação:

```
CREATE PROCEDURE OcultarLivro
@ID_Livro INTEGER,
@Estado INTEGER
AS

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION

Update Livro
Set Estado = @Estado
Where ID_Livro = @ID_Livro

IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
begin
    rollback
    RETURN -1
end

COMMIT
RETURN 1

go
```

4.2.17. Criar Leu

Nível de isolamento escolhido: Serializable; Uma vez que esta transação lida com a criação de uma nova linha numa tabela, originando-se um novo ID, todo o tipo de leituras deve ser bloqueado até que esteja confirmada a inserção na tabela. Caso não acontecesse, poderia ser lido este ID, criado outro autor e caso houvesse um erro, a aplicação poderia entrar em conflito uma vez que foi já teria sido criado outro autor. Assim, não será permitida *Dirty Read*, *Non repeatable read* nem *Phanton read*. Como visto no enquadramento teórico, o nível de isolamento que bloqueia estas leituras é o Serializable e, portanto, é o escolhido para esta transação.

Transação: Nesta transação é criada a relação Leu. Caso o utilizador tenha comentado o livro, a data de inserção do comentário também é inserida, caso contrário a data do comentário ficará nula. Quando o comentário for submetido, a data será submetida com GetDate(). Caso haja um erro é retornado -1, caso contrário é retornado 1.

Código da transação:

```
CREATE PROCEDURE CriarLeu
@ID_Livro INT,
@ID_Utilizador INT,
@Comentario VARCHAR(500)
AS
```

```

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

BEGIN TRANSACTION

    If(@Comentario <> null)
    begin
        INSERT INTO Leu(ID_Livro, ID_Utilizador, Data_Comentario,
Comentario, Estado)
        VALUES (@ID_Utilizador, @ID_Livro, GETDATE(), @Comentario, 1)
    end
    Else
    begin
        INSERT INTO Leu(ID_Livro, ID_Utilizador, Data_Comentario,
Comentario, Estado)
        VALUES (@ID_Utilizador, @ID_Livro, Null, Null, 1)
    end

    IF (@@ERROR <> 0)
    begin
        rollback
        RETURN -1
    end

    COMMIT
    RETURN 1
GO

```

4.2.18. Criar Possui

Nível de isolamento escolhido: Serializable; Uma vez que esta transação lida com a criação de uma nova linha numa tabela, originando-se um novo ID, todo o tipo de leituras deve ser bloqueado até que esteja confirmada a inserção na tabela. Caso não acontecesse, poderia ser lido este ID, criado outro autor e caso houvesse um erro, a aplicação poderia entrar em conflito uma vez que foi já teria sido criado outro autor. Assim, não será permitida *Dirty Read*, *Non repeatable read* nem *Phanton read*. Como visto no enquadramento teórico, o nível de isolamento que bloqueia estas leituras é o Serializable e, portanto, é o escolhido para esta transação.

Transação: Nesta transação é criada a relação Possui. Caso haja um erro é retornado -1, caso contrário é retornado 1.

Código da transação:

```

CREATE PROCEDURE CriarPossui
@ID_Utilizador INT,
@ID_Livro INT,
@Visibilidade BIT
AS

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

```

```

BEGIN TRANSACTION

INSERT INTO Possui(ID_Livro, ID_Utilizador, Visibilidade, Estado)
VALUES (@ID_Utilizador, @ID_Livro, @Visibilidade, 1)

IF (@@ERROR <> 0)
begin
    rollback
    RETURN -1
end

COMMIT
RETURN 1

GO

```

4.2.19. Criar Pedido

Nível de isolamento escolhido: Serializable; Uma vez que esta transação lida com a criação de uma nova linha numa tabela, originando-se um novo ID, todo o tipo de leituras deve ser bloqueado até que esteja confirmada a inserção na tabela. Caso não acontecesse, poderia ser lido este ID, criado outro autor e caso houvesse um erro, a aplicação poderia entrar em conflito uma vez que foi já teria sido criado outro autor. Assim, não será permitida *Dirty Read*, *Non repeatable read* nem *Phanton read*. Como visto no enquadramento teórico, o nível de isolamento que bloqueia estas leituras é o Serializable e, portanto, é o escolhido para esta transação.

Transação: Nesta transação é criada a relação *Pede*. Caso haja um erro é retornado -1, caso contrário é retornado 1

Código da transação:

```

CREATE PROCEDURE CriarPedido
@ID_Livro INTEGER,
@ID_Utilizador INTEGER
AS

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

BEGIN TRANSACTION

INSERT INTO Pede(ID_Livro, ID_Utilizador, Data_Criacao, Estado_Pedido)
VALUES (@ID_Utilizador, @ID_Livro, GETDATE(), 0)

IF (@@ERROR <> 0)
begin
    rollback
    RETURN -1
end

COMMIT
RETURN 1

GO

```

4.2.20. Apagar Leu

Nível de isolamento escolhido: Serializable; É apagado um registro na tabela, e portanto, a operação Phantom Read não pode ser permitida porque não devem ser inseridos novos registros enquanto não se confirmar que o registro desta transação foi apagado. Caso contrário, a aplicação poderia gerar um novo ID que teria o mesmo que o registro da transação que deveria ser apagado. Em caso de erro, os 2 registros teriam o mesmo ID. Assim sendo, o único nível de isolamento que não permite Phantom Read é o Serializable e por isso é o escolhido.

Transação: Nesta transação é apagada uma linha da relação Leu, recebendo como parâmetros as chaves primárias. Caso haja um erro é retornado -1, caso contrário é retornado 1

Código da transação:

```
CREATE PROCEDURE ApagarLeu
@ID_Livro INT,
@ID_Utilizador INT
AS

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

BEGIN TRANSACTION

Delete from Leu
where ID_Utilizador = @ID_Utilizador and ID_Livro = @ID_Livro

IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
begin
    rollback
    RETURN -1
end

COMMIT
RETURN 1

GO
```

4.2.21. Apagar Possui

Nível de isolamento escolhido: Serializable; É apagado um registro na tabela, e portanto, a operação Phantom Read não pode ser permitida porque não devem ser inseridos novos registros enquanto não se confirmar que o registro desta transação foi apagado. Caso contrário, a aplicação poderia gerar um novo ID que teria o mesmo que o registro da transação que deveria ser apagado. Em caso de erro, os 2 registros teriam o mesmo ID. Assim sendo, o único nível de isolamento que não permite Phantom Read é o Serializable e por isso é o escolhido.

Transação: Nesta transação é apagada uma linha da relação Possui, recebendo como parâmetros as chaves primárias. Caso haja um erro é retornado -1, caso contrário é retornado 1

Código da transação:

```
CREATE PROCEDURE ApagarPossui
@ID_Livro INT,
@ID_Utilizador INT
AS

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
BEGIN TRANSACTION

Delete from Possui
where ID_Utilizador = @ID_Utilizador and ID_Livro = @ID_Livro

IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
begin
    rollback
    RETURN -1
end

COMMIT
RETURN 1

GO
```

4.2.22. Cancelar Pedido

Nível de isolamento escolhido: Read committed; Acontece a modificação do estado do pedido e assim sendo a transação lida com a edição de uma linha na tabela, e portanto, a *Dirty Read* não deve ser permitida uma vez que não deve ser possível ler as informações que ainda não foram confirmadas como editadas. No entanto, a *Non-Repeatable Read* deve ser possível porque uma pode-se ler a informação mais do que uma vez diferente e a *Phantom Read* também deve ser permitida uma vez que a edição de uma linha não interfere com a possibilidade de serem inseridas

ou apagadas outras linhas. Assim, o nível de isolamento escolhido é o *Read Committed*, uma vez que permite *Non-Repeatable Read* e *Phanton Read* e não permite *Dirty Read*.

Transação: Nesta transação é cancelado um pedido, ou seja editada uma linha da relação Pede, passando o estado para 2. Caso haja um erro é retornado -1, caso contrário é retornado 1

Código da transação:

```
CREATE PROCEDURE CancelarPedido 0-pendente 1-aceite 2-cancelado
@ID_Livro INTEGER,
@ID_Utilizador INTEGER,
@Data_Criacao DATE,
@Estado_Pedido INTEGER
AS

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION

Update Pede
Set Estado_Pedido = 2
Where ID_Livro = @ID_Livro and ID_Utilizador = @ID_Utilizador and
Data_Criacao = @Data_Criacao

IF (@@ERROR <> 0 or @@ROWCOUNT = 0)
begin
    rollback
    RETURN -1
end

COMMIT
RETURN 1

GO
```

4.2.23. Editar Leu

Nível de isolamento escolhido: Read committed; A transação lida com a edição de uma linha na tabela, e portanto, a *Dirty Read* não deve ser permitida uma vez que não deve ser possível ler as informações que ainda não foram confirmadas como editadas. No entanto, a *Non-Repeatable Read* deve ser possível porque uma pode-se ler a informação mais do que uma vez diferente e a *Phanton Read* também deve ser permitida uma vez que a edição de uma linha não interfere com a possibilidade de serem inseridas ou apagadas outras linhas. Assim, o nível de isolamento escolhido é o *Read Committed*, uma vez que permite *Non-Repeatable Read* e *Phanton Read* e não permite *Dirty Read*.

Transação: Nesta transação editada uma linha da relação pede. Pode ser submetido um comentário do livro por parte do utilizador, havendo uma edição da relação Leu. Caso haja um erro é retornado -1, caso contrário é retornado 1

Código da transação:

```
CREATE PROCEDURE EditarLeu
@ID_Utilizador INTEGER,
@ID_Livro INTEGER,
@Comentario VARCHAR(500),
@Estado BIT
AS

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION

    UPDATE Leu
    SET      Comentario = @Comentario, Data_Comentario = GETDATE(), Estado
= @Estado
    WHERE ID_Utilizador = @ID_Utilizador and ID_Livro = @ID_Livro

    IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
    begin
        rollback
        RETURN -1
    end

    COMMIT
    RETURN 1
GO
```

4.2.24. Editar Possui

Nível de isolamento escolhido: Read committed; A transação lida com a edição de uma linha na tabela, e portanto, a *Dirty Read* não deve ser permitida uma vez que não deve ser possível ler as informações que ainda não foram confirmadas como editadas. No entanto, a *Non-Repeatable Read* deve ser possível porque uma pode-se ler a informação mais do que uma vez diferente e a *Phanton Read* também deve ser permitida uma vez que a edição de uma linha não interfere com a possibilidade de serem inseridas ou apagadas outras linhas. Assim, o nível de isolamento escolhido é o *Read Committed*, uma vez que permite *Non-Repeatable Read* e *Phanton Read* e não permite *Dirty Read*.

Transação: Nesta transação é editada uma linha da relação Possui, recebendo todos os atributos. Caso haja um erro é retornado -1, caso contrário é retornado 1

Código da transação:

```
CREATE PROCEDURE EditarPossui
@ID_Utilizador INT,
@ID_Livro INT,
@Visibilidade BIT,
@Estado BIT
AS

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION

    UPDATE Possui
    SET     Visibilidade = @Visibilidade, Estado = @Estado
    WHERE ID_Utilizador = @ID_Utilizador and ID_Livro = @ID_Livro

    IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
    begin
        rollback
        RETURN -1
    end

    COMMIT
    RETURN 1

GO
```

4.2.25. Editar Pedido

Nível de isolamento escolhido: Read committed; A transação lida com a edição de uma linha na tabela, e portanto, a *Dirty Read* não deve ser permitida uma vez que não deve ser possível ler as informações que ainda não foram confirmadas como editadas. No entanto, a *Non-Repeatable Read* deve ser possível porque uma pode-se ler a informação mais do que uma vez diferente e a *Phanton Read* também deve ser permitida uma vez que a edição de uma linha não interfere com a possibilidade de serem inseridas ou apagadas outras linhas. Assim, o nível de isolamento escolhido é o *Read Committed*, uma vez que permite *Non-Repeatable Read* e *Phanton Read* e não permite *Dirty Read*.

Transação: Nesta transação é editada uma linha da relação *Pede*, recebendo todos os atributos. Caso haja um erro é retornado -1, caso contrário é retornado 1

Código da transação:

```
CREATE PROCEDURE EditarPedido 0-pendente 1-aceite 2-cancelado
@ID_Livro INTEGER,
@ID_Utilizador INTEGER,
@Data_Criacao DATE,
@Estado_Pedido INTEGER
AS
```



```

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION

    Update Pedre
    Set Estado_Pedido = @Estado_Pedido
    Where ID_Livro = @ID_Livro and ID_Utilizador = @ID_Utilizador and
Data_Criacao = @Data_Criacao

    IF (@@ERROR <> 0 or @@ROWCOUNT = 0)
    begin
        rollback
        RETURN -1
    end

    COMMIT
    RETURN 1
GO

```

4.2.26. Criar Empréstimo

Nível de isolamento escolhido: Serializable; Uma vez que esta transação lida com a criação de uma nova linha numa tabela, originando-se um novo ID, todo o tipo de leituras deve ser bloqueado até que esteja confirmada a inserção na tabela. Caso não acontecesse, poderia ser lido este ID, criado outro autor e caso houvesse um erro, a aplicação poderia entrar em conflito uma vez que foi já teria sido criado outro autor. Assim, não será permitida *Dirty Read*, *Non repeatable read* nem *Phantom read*. Como visto no enquadramento teórico, o nível de isolamento que bloqueia estas leituras é o *Serializable* e, portanto, é o escolhido para esta transação.

Transação: Nesta transação é criado um empréstimo. É necessário alterar o estado da relação possui referente ao livro emprestado e ao utilizador que empresta assim como é necessário alterar o estado da relação Pedre para 1, relação referente ao pedido que o utilizador que recebe o livro fez. Caso haja qualquer erro numa destas alterações, o empréstimo não ocorre. É retornado um número diferente para erros diferentes e retornado 1 em caso de sucesso.

Código da transação:

```

CREATE PROCEDURE CriarEmprestimo
@ID_Livro INTEGER,
@ID_Utilizador_Pediu INTEGER,
@ID_Utilizador_Emprestou INTEGER,
@Data_Criacao DATE
AS

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

```

```

BEGIN TRANSACTION

        INSERT INTO Empresta(ID_Livro, ID_Utilizador_pediu,
ID_Utilizador_recebeu, Data_Emprestimo)
        VALUES (@ID_Livro, @ID_Utilizador_Pediu,
@ID_Utilizador_Emprestou, GETDATE())

        IF (@@ERROR <> 0)
        begin
                rollback
                RETURN -1
        end

        EXECUTE EditarPossui @ID_Utilizador_Emprestou, @ID_Livro, 1, 0 --
visibilidade 1 e Estado 0
        IF (@@ERROR <> 0)
        begin
                rollback
                RETURN -2
        end

        EXEC EditarPedido @ID_Livro, @ID_Utilizador_Pediu, @Data_Criacao, 1 -
-estado 1, pedido aceite
        IF (@@ERROR <> 0)
        begin
                rollback
                RETURN -3
        end

        COMMIT
        RETURN 1
GO

```

4.2.27. Devolver Livro

Nível de isolamento escolhido: Read committed; Acontece a modificação de estados e assim sendo a transação lida com a edição de linhas em tabelas, e portanto, a *Dirty Read* não deve ser permitida uma vez que não deve ser possível ler as informações que ainda não foram confirmadas como editadas. No entanto, a *Non-Repeatable Read* deve ser possível porque uma pode-se ler a informação mais do que uma vez diferente e a *Phanton Read* também deve ser permitida uma vez que a edição de uma linha não interfere com a possibilidade de serem inseridas ou apagadas outras linhas. Assim, o nível de isolamento escolhido é o *Read Committed*, uma vez que permite *Non-Repeatable Read* e *Phanton Read* e não permite *Dirty Read*.

Transação: Nesta transação é devolvido ao utilizador que emprestou o livro emprestado ao utilizador que pediu. É necessário atualizar a tabela “Empresta” para que a data de devolução seja preenchida com a data atual. De seguida vamos atualizar a tabela Possui através do utilizador que emprestou o livro e mudar os atributos visibilidade e estado para “1” para que fique visível e possa ser emprestado novamente a outros utilizadores. Por fim só resta atualizar a tabela “Leu” do utilizador que devolve o livro para que esse livro fiquem marcado como lido pelo utilizador que pediu o livro.

Código da transação:

```
CREATE PROCEDURE DevolverLivro
@ID_Livro INTEGER,
@ID_Utilizador_Pediu INTEGER,
@ID_Utilizador_Emprestou INTEGER,
@Data_Empresta DATE,
@LeuLivro BIT,
@Comentario VARCHAR(500)
AS

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION

    UPDATE Empresta
    SET Data_Devolucao = GETDATE()
    WHERE ID_Utilizador_pediu = @ID_Utilizador_Pediu and ID_Livro =
@ID_Livro and ID_Utilizador_recebeu = @ID_Utilizador_Emprestou and
Data_Emprestimo = @Data_Empresta

    IF (@@ERROR <> 0)
    begin
        rollback
        RETURN -1
    end

    EXECUTE EditarPossui @ID_Utilizador_Emprestou, @ID_Livro, 1, 1 --
visibilidade 1 e Estado 1
    IF (@@ERROR <> 0)
    begin
        rollback
        RETURN -2
    end

    EXEC EditarLeu @ID_Utilizador_Pediu, @ID_Livro, @Comentario,
@LeuLivro --estado 1, pedido aceite
    IF (@@ERROR <> 0)
    begin
        rollback
        RETURN -3
    end

    COMMIT
    RETURN 1

GO
```

4.2.28. Registrar Utilizador

Nível de isolamento escolhido: Serializable; Uma vez que esta transação lida com a criação de uma nova linha numa tabela, originando-se um novo ID, todo o tipo de leituras deve ser bloqueado até que esteja confirmada a inserção na tabela. Caso não acontecesse, poderia ser lido este ID, criado outro autor e caso houvesse um erro, a aplicação poderia entrar em conflito uma vez que foi já teria sido criado outro autor. Assim, não será permitida *Dirty Read*, *Non repeatable read* nem *Phanton read*. Como visto no enquadramento teórico, o nível de isolamento que bloqueia estas leituras é o *Serializable* e, portanto, é o escolhido para esta transação.

Transação: Nesta transição é feito o registo de um novo utilizador. Para isso temos de inserir uma nova linha na tabela utilizador com os dados que vão ser inseridos pelo utilizador a registar-se. O estado fica a 0 até o utilizador confirmar a sua conta e só nesse momento passa para 1.

Código da transação:

```
CREATE PROCEDURE RegistrarUtilizador
@Username VARCHAR(20),
@Pass VARCHAR(20),
@Nome VARCHAR(60),
@DataNasc DATE,
@EMail VARCHAR(255),
@Morada VARCHAR(500)
AS

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

BEGIN TRAN
    INSERT INTO Utilizador(Username, Pass, Nome, Estado, Data_Nascimento,
Email, MoradaLocalidade)
    VALUES(@Username, @Pass, @Nome, 0, @DataNasc, @EMail, @Morada) --
Estado a 0, Pendente

    IF (@@ERROR <> 0)
    begin
        rollback
        RETURN -1
    end

    COMMIT
    RETURN 1
GO
```

4.2.29. Editar Utilizador

Nível de isolamento escolhido: Read committed; A transação lida com a edição de uma linha na tabela, e portanto, a *Dirty Read* não deve ser permitida uma vez que não deve ser possível ler as informações que ainda não foram confirmadas como editadas. No entanto, a *Non-Repeatable Read* deve ser possível porque uma pode-se ler a informação mais do que uma vez diferente e a *Phanton Read* também deve ser permitida uma vez que a edição de uma linha não interfere com a possibilidade de serem inseridas ou apagadas outras linhas. Assim, o nível de isolamento escolhido é o *Read Committed*, uma vez que permite *Non-Repeatable Read* e *Phanton Read* e não permite *Dirty Read*.

Transação: Nesta transação o utilizador pode editar os seus dados. Para isso vamos à tabela utilizador e alteramos a linha que tem o ID_Utilizador igual ao do id do utilizador atual. Esta transição é similar à anterior, mas em vez de inserir vamos atualizar os dados.

Código da transação:

```
CREATE PROCEDURE EditarUtilizador
@ID_Utilizador INTEGER,
@Username VARCHAR(50),
@Pass VARCHAR(16),
@Nome VARCHAR(50),
@DataNasc VARCHAR(255),
@EMail VARCHAR(255),
@Morada VARCHAR(255)
AS

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRAN

    UPDATE Utilizador
    SET Username = @Username, Pass = @Pass, Nome = @Nome, Data_Nascimento
    = @DataNasc, Email = @EMail, MoradaLocalidade = @Morada
    WHERE ID_Utilizador = @ID_Utilizador

    IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
    begin
        rollback
        RETURN -1
    end

    COMMIT
    RETURN 1

GO
```

4.2.30. Bloquear Utilizador

Nível de isolamento escolhido: Read committed; Acontece a modificação do estado do Utilizador e assim sendo a transação lida com a edição de uma linha na tabela, e portanto, a *Dirty Read* não deve ser permitida uma vez que não deve ser possível ler as informações que ainda não foram confirmadas como editadas. No entanto, a *Non-Repeatable Read* deve ser possível porque uma pode-se ler a informação mais do que uma vez diferente e a *Phanton Read* também deve ser permitida uma vez que a edição de uma linha não interfere com a possibilidade de serem inseridas ou apagadas outras linhas. Assim, o nível de isolamento escolhido é o *Read Committed*, uma vez que permite *Non-Repeatable Read* e *Phanton Read* e não permite *Dirty Read*.

Transação: Nesta transição vamos bloquear um utilizador. Vão entrar como parâmetros o id do administrador que bloqueia, o id do utilizador a ser bloqueado e o motivo do bloqueio. Se houver um motivo, o mesmo, vai ser inserido na tabela Bloqueia assim como os ids de utilizador e administrador e a data de bloqueio, que é a atual, se não houver um motivo serão inseridos todos os dados na mesma com a exceção do atributo “motivo”. Depois disto só falta atualizar o estado do utilizador que entrou como parâmetro para 2, que é o estado bloqueado.

Código da transação:

```
CREATE PROCEDURE BloquearUtilizador
@ID_Admin INTEGER,
@ID_Utilizador INTEGER,
@Motivo VARCHAR(200)
AS

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

BEGIN TRANSACTION

    If(@Motivo <> null)
    begin
        INSERT INTO Bloqueia(ID_Administrador, ID_Utilizador,
Data_Bloqueio, Motivo)
        VALUES (@ID_Admin, @ID_Utilizador, GETDATE(), @Motivo)
    end
    Else
    begin
        INSERT INTO Bloqueia(ID_Administrador, ID_Utilizador,
Data_Bloqueio)
        VALUES (@ID_Admin, @ID_Utilizador, GETDATE())
    end
    IF (@@ERROR <> 0)
    begin
        rollback
        RETURN -1
    end
end
```

```

UPDATE Utilizador
Set Estado = 2
Where ID_Utilizador = @ID_Utilizador
IF (@@ERROR <> 0 or @@ROWCOUNT = 0)
begin
    rollback
    RETURN -2
end

COMMIT
RETURN 1

GO

```

4.2.31. Desbloquear Utilizador

Nível de isolamento escolhido: Read committed; Acontece a modificação de um estado e assim sendo a transação lida com a edição de uma linha na tabela, e portanto, a *Dirty Read* não deve ser permitida uma vez que não deve ser possível ler as informações que ainda não foram confirmadas como editadas. No entanto, a *Non-Repeatable Read* deve ser possível porque uma pode-se ler a informação mais do que uma vez diferente e a *Phanton Read* também deve ser permitida uma vez que a edição de uma linha não interfere com a possibilidade de serem inseridas ou apagadas outras linhas. Assim, o nível de isolamento escolhido é o *Read Committed*, uma vez que permite *Non-Repeatable Read* e *Phanton Read* e não permite *Dirty Read*.

Transação: Nesta transação vamos desbloquear um utilizador que foi anteriormente bloqueado. Para isso precisamos dos parâmetros de entrada do id do administrador que vai desbloquear e do id do utilizador a ser desbloqueado. De seguida temos de atualizar a tabela bloqueia e preencher o campo data de desbloqueio com a data atual. Por fim só temos de atualizar a tabela utilizador e mudar o seu estado para 1 do utilizador que tem o id do utilizador de entrada.

Código da transação:

```

CREATE PROCEDURE DesbloquearUtilizador
@ID_Admin INTEGER,
@ID_Utilizador INTEGER
AS

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION

UPDATE Bloqueia
SET Data_Desbloqueio = GETDATE()

```

```

WHERE ID_Administrador = @ID_Admin and ID_Utilizador = @ID_Utilizador

IF (@@ERROR <> 0)
begin
    rollback
    RETURN -1
end

UPDATE Utilizador
Set Estado = 1
Where ID_Utilizador = @ID_Utilizador
IF (@@ERROR <> 0 or @@ROWCOUNT = 0)
begin
    rollback
    RETURN -2
end

COMMIT
RETURN 1

GO

```

4.2.32. Ativar Utilizador

Nível de isolamento escolhido: Read committed; Acontece a modificação do estado do Utilizador e assim sendo a transação lida com a edição de uma linha na tabela, e portanto, a *Dirty Read* não deve ser permitida uma vez que não deve ser possível ler as informações que ainda não foram confirmadas como editadas. No entanto, a *Non-Repeatable Read* deve ser possível porque uma pode-se ler a informação mais do que uma vez diferente e a *Phanton Read* também deve ser permitida uma vez que a edição de uma linha não interfere com a possibilidade de serem inseridas ou apagadas outras linhas. Assim, o nível de isolamento escolhido é o *Read Committed*, uma vez que permite *Non-Repeatable Read* e *Phanton Read* e não permite *Dirty Read*.

Transação: Nesta transação vamos proceder à ativação do utilizador. Quando um utilizador se regista o estado dele fica, pendentemente, a 0. Depois de ele ir ao email confirmar a sua conta temos de ir à tabela Utilizador e através do seu id atualizamos o seu estado para 1, que é o estado ativo.

Código da transação:

```

CREATE PROCEDURE AtivarUtilizador
@ID_Utilizador INT
AS

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION

```



```

UPDATE Utilizador
Set Estado = 1
Where ID_Utilizador = @ID_Utilizador

IF (@@ERROR <> 0 or @@ROWCOUNT = 0)
begin
    rollback
    RETURN -2
end

COMMIT
RETURN 1

GO

```

4.2.33. Editar Admin

Nível de isolamento escolhido: Read Committed; A transação lida com a edição de uma linha na tabela, e portanto, a *Dirty Read* não deve ser permitida uma vez que não deve ser possível ler as informações que ainda não foram confirmadas como editadas. No entanto, a *Non-Repeatable Read* deve ser possível porque uma pode-se ler a informação mais do que uma vez diferente e a *Phanton Read* também deve ser permitida uma vez que a edição de uma linha não interfere com a possibilidade de serem inseridas ou apagadas outras linhas. Assim, o nível de isolamento escolhido é o *Read Committed*, uma vez que permite *Non-Repeatable Read* e *Phanton Read* e não permite *Dirty Read*.

Transação: Nesta transação é feita uma atualização aos dados do administrador. Aqui o administrador pode editar os seus próprios dados se assim desejar. Entram como parâmetros o id do administrador, a palavra passe e o email. Para editar o admin precisamos de aceder à tabela “Administrador” e modificar o administrador que tenha o mesmo id que o ID_Administrador que entrou como parâmetro. Depois disso é só alterar os dados e a atualização do administrador está concluída.

Código da transação:

```

CREATE PROCEDURE EditarAdmin
@ID_Admin INTEGER,
@Pass VARCHAR(100),
@EMail VARCHAR(255)
AS

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRAN

```

```
UPDATE Administrador
SET Pass = @Pass, Email = @EMail
WHERE ID_Administrador = @ID_Admin

IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
begin
    rollback
    RETURN -1
end

COMMIT
RETURN 1

GO
```

5. Análise e Discussão dos Resultados

Após a concretização deste trabalho pensamos ter conseguido alcançar os objetivos propostos. Consideramos que as políticas de segurança e acesso aos dados definidas ilustram de forma correta o sistema, tendo em conta a nossa visão dos requisitos.

Por sua vez os problemas de concorrência em sistemas de gestão de bases de dados centralizadas foram resolvidos, tendo em conta a matéria teórica lecionada nas aulas de TABD.

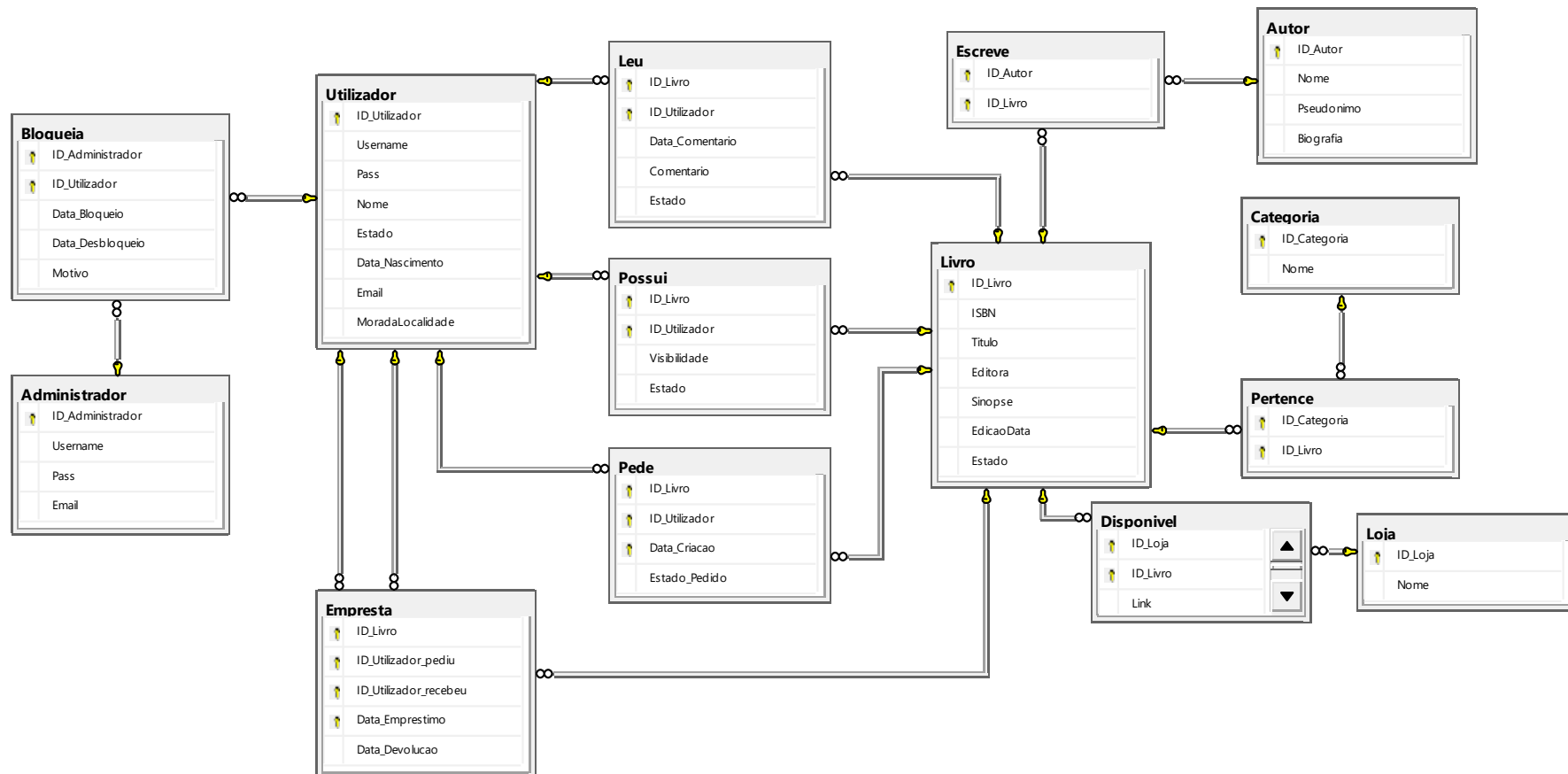
Assim, pensamos ter atingido a 100% os nossos objetivos.

6. Bibliografia

- [1] Microsoft. “Níveis de isolamento no Mecanismo de Banco de Dados”. Disponível em: [https://technet.microsoft.com/pt-BR/library/ms189122\(v=sql.105\).aspx](https://technet.microsoft.com/pt-BR/library/ms189122(v=sql.105).aspx). Acesso em 28 de março de 2017.
- [2] Microsoft. “Server and Database Roles in SQL Server“. Disponível em [https://msdn.microsoft.com/en-us/library/bb669065\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/bb669065(v=vs.110).aspx). Acesso em 29 de março de 2017.
- [3] Martins P. “Apontamentos Teóricos”.

7. Anexos

7.1. Diagrama da Base de Dados



7.2.Código SQL

```
use BookList
go

-----
--      Criar Logins      --
-----

CREATE LOGIN UtilizadorLog      WITH PASSWORD = 'Utilizador'
CREATE LOGIN AdministradorLog  WITH PASSWORD = 'Administrador'
CREATE LOGIN VisitanteLog      WITH PASSWORD = 'Visitante'

-----
--      Criar Users      --
-----

CREATE USER Utilizador_TABD      FOR LOGIN UtilizadorLog
CREATE USER Administrador_TABD  FOR LOGIN AdministradorLog
CREATE USER Visitante_TABD      FOR LOGIN VisitanteLog

-----
--      Criar Roles      --
-----

CREATE ROLE roleAdministrador
EXEC sp_addrolemember 'roleAdministrador', 'Administrador_TABD'

CREATE ROLE roleUtilizador
EXEC sp_addrolemember 'roleUtilizador', 'Utilizador_TABD'

CREATE ROLE roleVisitante
EXEC sp_addrolemember 'roleVisitante', 'Visitante_TABD'

-----
--      Criar Permissões  --
-----

-----> Visitante:
GRANT SELECT ON Livro          TO roleVisitante
GRANT SELECT ON Escreve        TO roleVisitante
GRANT SELECT ON Autor          TO roleVisitante
GRANT SELECT ON Pertence       TO roleVisitante
GRANT SELECT ON Categoria      TO roleVisitante
GRANT INSERT ON Utilizador     TO roleVisitante

-----> Utilizador:
GRANT SELECT                                ON Autor          TO roleUtilizador
GRANT SELECT                                ON Escreve         TO roleUtilizador
GRANT SELECT                                ON Livro          TO roleUtilizador
GRANT SELECT                                ON Pertence       TO roleUtilizador
GRANT SELECT                                ON Categoria      TO roleUtilizador
GRANT SELECT                                ON Disponivel     TO roleUtilizador
GRANT SELECT                                ON Loja           TO roleUtilizador
GRANT SELECT, update                       ON Utilizador     TO roleUtilizador
GRANT SELECT, INSERT, update, delete ON Possui          TO roleUtilizador
GRANT SELECT, INSERT, update, delete ON Pedir            TO roleUtilizador
GRANT SELECT, INSERT, update, delete ON Emprestar        TO roleUtilizador
GRANT SELECT, INSERT, update, delete ON Leu              TO roleUtilizador
```

```

-----> Administrador:
GRANT SELECT ON Administrador TO roleAdministrador
GRANT SELECT ON Possui TO roleAdministrador
GRANT SELECT ON Pedir TO roleAdministrador
GRANT SELECT ON Empresta TO roleAdministrador
GRANT SELECT ON Leu TO roleAdministrador
GRANT SELECT, update ON Utilizador TO roleAdministrador
GRANT SELECT, INSERT, update ON Bloqueia TO roleAdministrador
GRANT SELECT, INSERT, update ON Livro TO roleAdministrador
GRANT SELECT, INSERT, update, delete ON Escreve TO roleAdministrador
GRANT SELECT, INSERT, update ON Autor TO roleAdministrador
GRANT SELECT, INSERT, update, delete ON Pertence TO roleAdministrador
GRANT SELECT, INSERT, update ON Categoria TO roleAdministrador
GRANT SELECT, INSERT, update, delete ON Disponivel TO roleAdministrador
GRANT SELECT, INSERT, update ON Loja TO roleAdministrador

-----
--Permissões para Procedures--
-----
-----> Visitante:
GRANT EXECUTE ON RegistrarUtilizador TO roleVisitante

-----> Utilizador:
GRANT EXECUTE ON CriarLeu TO roleUtilizador
GRANT EXECUTE ON CriarPossui TO roleUtilizador
GRANT EXECUTE ON CriarPedido TO roleUtilizador
GRANT EXECUTE ON ApagarLeu TO roleUtilizador
GRANT EXECUTE ON ApagarPossui TO roleUtilizador
GRANT EXECUTE ON EditarLeu TO roleUtilizador
GRANT EXECUTE ON EditarPossui TO roleUtilizador
GRANT EXECUTE ON EditarPedido TO roleUtilizador
GRANT EXECUTE ON CriarEmprestimo TO roleUtilizador
GRANT EXECUTE ON DevolverLivro TO roleUtilizador
GRANT EXECUTE ON EditarUtilizador TO roleUtilizador

-----> Administrador:
GRANT EXECUTE ON CriarAutor TO roleAdministrador
GRANT EXECUTE ON CriarCategoria TO roleAdministrador
GRANT EXECUTE ON CriarLoja TO roleAdministrador
GRANT EXECUTE ON EditarAutor TO roleAdministrador
GRANT EXECUTE ON EditarCategoria TO roleAdministrador
GRANT EXECUTE ON EditarLoja TO roleAdministrador
GRANT EXECUTE ON CriarLivroCategoria TO roleAdministrador
GRANT EXECUTE ON CriarLivroAutor TO roleAdministrador
GRANT EXECUTE ON CriarLivroLoja TO roleAdministrador
GRANT EXECUTE ON ApagarLivroCategoria TO roleAdministrador
GRANT EXECUTE ON ApagarLivroAutor TO roleAdministrador
GRANT EXECUTE ON ApagarLivroLoja TO roleAdministrador
GRANT EXECUTE ON EditarDisponivel TO roleAdministrador
GRANT EXECUTE ON CriarLivro TO roleAdministrador
GRANT EXECUTE ON EditarLivro TO roleAdministrador
GRANT EXECUTE ON OcultarLivro TO roleAdministrador
GRANT EXECUTE ON BloquearUtilizador TO roleAdministrador
GRANT EXECUTE ON DesbloquearUtilizador TO roleAdministrador
GRANT EXECUTE ON AtivarUtilizador TO roleAdministrador
GRANT EXECUTE ON EditarAdmin TO roleAdministrador

-----

```

```

-- Standard Procedures --
-----

USE BookList
GO

-----> Criar Autor <-----
CREATE PROCEDURE CriarAutor
@Nome VARCHAR(60),
@Pseudonimo VARCHAR(60),
@Biografia VARCHAR(1000)
AS
SET TRANSACTION ISOLATION LEVEL Serializable
BEGIN TRANSACTION

    INSERT INTO Autor(Nome, Pseudonimo, Biografia) VALUES (@Nome, @Pseudonimo,
@Biografia)

    IF (@@ERROR <> 0)
    begin
        rollback
        RETURN -1
    end

    COMMIT
    RETURN 1
GO

-----> Criar Categoria <-----
CREATE PROCEDURE CriarCategoria
@Nome VARCHAR(60)
AS
SET TRANSACTION ISOLATION LEVEL Serializable
BEGIN TRANSACTION

    INSERT INTO Categoria(Nome) VALUES (@Nome)

    IF (@@ERROR <> 0)
    begin
        rollback
        RETURN -1
    end

    COMMIT
    RETURN 1
GO

-----> Criar Loja <-----
CREATE PROCEDURE CriarLoja
@Nome VARCHAR(60)
AS
SET TRANSACTION ISOLATION LEVEL Serializable
BEGIN TRANSACTION

    INSERT INTO Loja(Nome) VALUES (@Nome)

    IF (@@ERROR <> 0)
    begin
        rollback
        RETURN -1
    end

```



```

        end

        COMMIT
        RETURN 1
GO

----> Editar Autor <----
CREATE PROCEDURE EditarAutor
@ID_Autor INT,
@Nome VARCHAR(60),
@Pseudonimo VARCHAR(60),
@Biografia VARCHAR(1000)
AS

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION

        UPDATE Autor
        SET Nome = @Nome, Pseudonimo = @Pseudonimo, Biografia = @Biografia
        WHERE ID_Autor = @ID_Autor

        IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
        begin
                rollback
                RETURN -1
        end

        COMMIT
        RETURN 1
GO

----> Editar Categoria <----
CREATE PROCEDURE EditarCategoria
@ID_Categoria INT,
@Nome VARCHAR(60)
AS

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION

        UPDATE Categoria
        SET Nome = @Nome
        WHERE ID_Categoria = @ID_Categoria

        IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
        begin
                rollback
                RETURN -1
        end

        COMMIT
        RETURN 1
GO

----> Editar Loja <----
CREATE PROCEDURE EditarLoja
@ID_Loja INT,
@Nome VARCHAR(60)
AS

```

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
```

```
BEGIN TRANSACTION
```

```
UPDATE Loja
SET Nome = @Nome
WHERE ID_Loja = @ID_Loja

IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
begin
    rollback
    RETURN -1
end

COMMIT
RETURN 1
```

```
GO
```

```
-----> Criar Relação Livro-Categoria <-----
```

```
CREATE PROCEDURE CriarLivroCategoria
```

```
@ID_Categoria INTEGER,
```

```
@ID_Livro INTEGER
```

```
AS
```

```
SET TRANSACTION ISOLATION LEVEL Serializable
```

```
BEGIN TRANSACTION
```

```
INSERT INTO Pertence(ID_Categoria, ID_Livro) VALUES (@ID_Categoria, @ID_Livro)
```

```
IF (@@ERROR <> 0)
begin
    rollback
    RETURN -1
end
```

```
COMMIT
RETURN 1
```

```
GO
```

```
-----> Criar Relação Livro-Autor <-----
```

```
CREATE PROCEDURE CriarLivroAutor
```

```
@ID_Autor INTEGER,
```

```
@ID_Livro INTEGER
```

```
AS
```

```
SET TRANSACTION ISOLATION LEVEL Serializable
```

```
BEGIN TRANSACTION
```

```
INSERT INTO Escreve(ID_Autor, ID_Livro) VALUES (@ID_Autor, @ID_Livro)
```

```
IF (@@ERROR <> 0)
begin
    rollback
    RETURN -1
end
```

```
COMMIT
RETURN 1
```

```
GO
```

```

----> Criar Relação Livro-Loja <----
CREATE PROCEDURE CriarLivroLoja
@ID_Loja INTEGER,
@ID_Livro INTEGER,
@Link VARCHAR(300)
AS
SET TRANSACTION ISOLATION LEVEL Serializable

BEGIN TRANSACTION

INSERT INTO Disponivel(ID_Loja, ID_Livro, Link) VALUES (@ID_Loja, @ID_Livro, @Link)

IF (@@ERROR <> 0)
begin
    rollback
    RETURN -1
end

COMMIT
RETURN 1
GO

----> Apagar relação Livro-Categoria <----
CREATE PROCEDURE ApagarLivroCategoria
@ID_Categoria INTEGER,
@ID_Livro INTEGER
AS

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

BEGIN TRANSACTION

Delete from Pertence
where ID_Categoria = @ID_Categoria and ID_Livro = @ID_Livro

IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
begin
    rollback
    RETURN -1
end

COMMIT
RETURN 1
GO

----> Apagar relação Livro-Autor <----
CREATE PROCEDURE ApagarLivroAutor
@ID_Autor INTEGER,
@ID_Livro INTEGER
AS

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

BEGIN TRANSACTION

Delete from Escreve
where ID_Autor = @ID_Autor and ID_Livro = @ID_Livro

IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
begin
    rollback

```

```

        RETURN -1
    end

    COMMIT
    RETURN 1
GO

-----> Apagar relação Livro-Loja <-----
CREATE PROCEDURE ApagarLivroLoja
@ID_Loja INTEGER,
@ID_Livro INTEGER
AS

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

BEGIN TRANSACTION

    Delete from Disponivel
    where ID_Loja = @ID_Loja and ID_Livro = @ID_Livro

    IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
    begin
        rollback
        RETURN -1
    end

    COMMIT
    RETURN 1
GO

-----> Editar Disponivel <-----
CREATE PROCEDURE EditarDisponivel
@ID_Loja INTEGER,
@ID_Livro INTEGER,
@Link VARCHAR(300)
AS

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION

    Update Disponivel
    Set Link = @Link
    Where ID_Loja = @ID_Loja and ID_Livro = @ID_Livro

    IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
    begin
        rollback
        RETURN -1
    end

    COMMIT
    RETURN 1
GO

-----> Criar Livro <-----
CREATE PROCEDURE CriarLivro
@ISBN VARCHAR(30),
@Titulo VARCHAR(50),
@Editora VARCHAR(50),
@Sinopse VARCHAR(MAX),

```

```

@EdicaoData DATE,
@ID_Categoria INTEGER,
@ID_Autor INTEGER,
@ID_Loja INTEGER,
@Link VARCHAR(300)
AS

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

BEGIN TRANSACTION

    INSERT INTO Livro(ISBN, Titulo, Editora, Sinopse, EdicaoData, Estado)
        VALUES (@ISBN, @Titulo, @Editora, @Sinopse, @EdicaoData, 1)

    IF (@@ERROR <> 0)
    begin
        rollback
        return -1
    end
-- -1 -nao inseriu livro
-- 1 - inseriu livro COM autor, loja e categoria
-- 2 - inseriu livro SEM autor, loja ou categoria
declare @valreturn int = 1
EXEC CriarLivroCategoria @ID_Categoria, @@IDENTITY
IF (@@ERROR <> 0)
begin
    set @valreturn=2
end

EXEC CriarLivroAutor @ID_Autor, @@IDENTITY
IF (@@ERROR <> 0)
begin
    set @valreturn=2
end

EXEC CriarLivroLoja @ID_Loja, @@IDENTITY, @Link
IF (@@ERROR <> 0)
begin
    set @valreturn=2
end

RETURN @valreturn
COMMIT

GO

----> Editar Livro <----
CREATE PROCEDURE EditarLivro
@ID_Livro INTEGER,
@ISBN VARCHAR(30),
@Titulo VARCHAR(50),
@Editora VARCHAR(50),
@Sinopse VARCHAR(MAX),
@EdicaoData DATE
AS

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION

    Update Livro

```

```

        Set ISBN = @ISBN , Titulo = @Titulo, Editora = @Editora, Sinopse = @Sinopse,
EdicaoData = @EdicaoData
        Where ID_Livro = @ID_Livro

        IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
        begin
            rollback
            RETURN -1
        end

        COMMIT
        RETURN 1
GO

-----> Ocultar/Desocultar Livro <-----    0-Oculto, 1-Visivel
CREATE PROCEDURE OcultarLivro
@ID_Livro INTEGER,
@Estado INTEGER
AS

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION

        Update Livro
        Set Estado = @Estado
        Where ID_Livro = @ID_Livro

        IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
        begin
            rollback
            RETURN -1
        end

        COMMIT
        RETURN 1
go

-----> Criar Leu <-----
CREATE PROCEDURE CriarLeu
@ID_Livro INT,
@ID_Utilizador INT,
@Comentario VARCHAR(500)
AS

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

BEGIN TRANSACTION

        If(@Comentario <> null)
        begin
            INSERT INTO Leu(ID_Livro, ID_Utilizador, Data_Comentario, Comentario,
Estado)
            VALUES (@ID_Utilizador, @ID_Livro, GETDATE(), @Comentario, 1)
        end
        Else
        begin
            INSERT INTO Leu(ID_Livro, ID_Utilizador, Data_Comentario, Comentario,
Estado)
            VALUES (@ID_Utilizador, @ID_Livro, Null, Null, 1)
        end
end

```

```

        IF (@@ERROR <> 0)
        begin
            rollback
            RETURN -1
        end

        COMMIT
        RETURN 1
GO

-----> Criar Possui <-----
CREATE PROCEDURE CriarPossui
@ID_Utilizador INT,
@ID_Livro INT,
@Visibilidade BIT
AS

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

BEGIN TRANSACTION

        INSERT INTO Possui(ID_Livro, ID_Utilizador, Visibilidade, Estado)
        VALUES (@ID_Utilizador, @ID_Livro, @Visibilidade, 1)

        IF (@@ERROR <> 0)
        begin
            rollback
            RETURN -1
        end

        COMMIT
        RETURN 1
GO

-----> Criar Pedido <-----
CREATE PROCEDURE CriarPedido
@ID_Livro INTEGER,
@ID_Utilizador INTEGER
AS

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

BEGIN TRANSACTION

        INSERT INTO Pede(ID_Livro, ID_Utilizador, Data_Criacao, Estado_Pedido)
        VALUES (@ID_Utilizador, @ID_Livro, GETDATE(), 0)

        IF (@@ERROR <> 0)
        begin
            rollback
            RETURN -1
        end

        COMMIT
        RETURN 1
GO

-----> Apagar Leu <-----
CREATE PROCEDURE ApagarLeu
@ID_Livro INT,

```

```

@ID_Utilizador INT
AS

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

BEGIN TRANSACTION

Delete from Leu
where ID_Utilizador = @ID_Utilizador and ID_Livro = @ID_Livro

IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
begin
    rollback
    RETURN -1
end

COMMIT
RETURN 1
GO

-----> Apagar Possui <-----
CREATE PROCEDURE ApagarPossui
@ID_Livro INT,
@ID_Utilizador INT
AS

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

BEGIN TRANSACTION

Delete from Possui
where ID_Utilizador = @ID_Utilizador and ID_Livro = @ID_Livro

IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
begin
    rollback
    RETURN -1
end

COMMIT
RETURN 1
GO

-----> Editar Leu <-----
CREATE PROCEDURE EditarLeu
@ID_Utilizador INTEGER,
@ID_Livro INTEGER,
@Comentario VARCHAR(500),
@Estado BIT
AS

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION

UPDATE Leu
SET Comentario = @Comentario, Data_Comentario = GETDATE(), Estado = @Estado
WHERE ID_Utilizador = @ID_Utilizador and ID_Livro = @ID_Livro

```



```

        IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
        begin
            rollback
            RETURN -1
        end

        COMMIT
        RETURN 1
GO

-----> Editar Possui <-----
CREATE PROCEDURE EditarPossui
@ID_Utilizador INT,
@ID_Livro INT,
@Visibilidade BIT,
@Estado BIT
AS

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION

    UPDATE Possui
    SET     Visibilidade = @Visibilidade, Estado = @Estado
    WHERE ID_Utilizador = @ID_Utilizador and ID_Livro = @ID_Livro

    IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
    begin
        rollback
        RETURN -1
    end

    COMMIT
    RETURN 1
GO

-----> Editar Pedido <----- 0-pendente 1-aceite 2-cancelado
CREATE PROCEDURE EditarPedido
@ID_Livro INTEGER,
@ID_Utilizador INTEGER,
@Data_Criacao DATE,
@Estado_Pedido INTEGER
AS

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION

    Update Pedo
    Set Estado_Pedido = @Estado_Pedido
    Where ID_Livro = @ID_Livro and ID_Utilizador = @ID_Utilizador and Data_Criacao =
@Data_Criacao

    IF (@@ERROR <> 0 or @@ROWCOUNT = 0)
    begin
        rollback
        RETURN -1
    end

    COMMIT
    RETURN 1

```

GO

-----> Criar Emprestimo (Aceitar Pedido)<-----

CREATE PROCEDURE CriarEmprestimo

@ID_Livro INTEGER,

@ID_Utilizador_Pediu INTEGER,

@ID_Utilizador_Emprestou INTEGER,

@Data_Criacao DATE

AS

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

BEGIN TRANSACTION

INSERT INTO Empresta(ID_Livro, ID_Utilizador_pediu, ID_Utilizador_recebeu,
Data_Emprestimo)

VALUES (@ID_Livro, @ID_Utilizador_Pediu, @ID_Utilizador_Emprestou,
GETDATE())

IF (@@ERROR <> 0)

begin

rollback

RETURN -1

end

EXECUTE EditarPossui @ID_Utilizador_Emprestou, @ID_Livro, 1, 0 --visibilidade 1 e
Estado 0

IF (@@ERROR <> 0)

begin

rollback

RETURN -2

end

EXEC EditarPedido @ID_Livro, @ID_Utilizador_Pediu, @Data_Criacao, 1 --estado 1,
pedido aceite

IF (@@ERROR <> 0)

begin

rollback

RETURN -3

end

COMMIT

RETURN 1

GO

-----> Devolver Livro <-----

CREATE PROCEDURE DevolverLivro

@ID_Livro INTEGER,

@ID_Utilizador_Pediu INTEGER,

@ID_Utilizador_Emprestou INTEGER,

@Data_Empresta DATE,

@LeuLivro BIT,

@Comentario VARCHAR(500)

AS

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION

```

        UPDATE Empresta
        SET Data_Devolucao = GETDATE()
        WHERE ID_Utilizador_pedi = @ID_Utilizador_Pedi and ID_Livro = @ID_Livro and
ID_Utilizador_recebeu = @ID_Utilizador_Emprestou and Data_Emprestimo = @Data_Empresta

        IF (@@ERROR <> 0)
        begin
            rollback
            RETURN -1
        end

EXECUTE EditarPossui @ID_Utilizador_Emprestou, @ID_Livro, 1, 1 --visibilidade 1 e
Estado 1
        IF (@@ERROR <> 0)
        begin
            rollback
            RETURN -2
        end

EXEC EditarLeu @ID_Utilizador_Pedi, @ID_Livro, @Comentario, @LeuLivro --estado 1,
pedido aceite
        IF (@@ERROR <> 0)
        begin
            rollback
            RETURN -3
        end

        COMMIT
        RETURN 1
GO

----> Registrar Utilizador <----
CREATE PROCEDURE RegistrarUtilizador
@Username VARCHAR(20),
@Pass VARCHAR(20),
@Nome VARCHAR(60),
@DataNasc DATE,
@EMail VARCHAR(255),
@Morada VARCHAR(500)
AS

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

BEGIN TRAN
    INSERT INTO Utilizador(Username, Pass, Nome, Estado, Data_Nascimento, Email,
MoradaLocalidade)
    VALUES(@Username, @Pass, @Nome, 0, @DataNasc, @EMail, @Morada) --Estado a 0,
Pendente

    IF (@@ERROR <> 0)
    begin
        rollback
        RETURN -1
    end

    COMMIT
    RETURN 1
GO

----> Editar Utilizador <----

```

```

CREATE PROCEDURE EditarUtilizador
@ID_Utilizador INTEGER,
@Username VARCHAR(50),
@Pass VARCHAR(16),
@Nome VARCHAR(50),
@DataNasc VARCHAR(255),
@EMail VARCHAR(255),
@Morada VARCHAR(255)
AS

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRAN

    UPDATE Utilizador
    SET Username = @Username, Pass = @Pass, Nome = @Nome, Data_Nascimento = @DataNasc,
    Email = @EMail, MoradaLocalidade = @Morada
    WHERE ID_Utilizador = @ID_Utilizador

    IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
    begin
        rollback
        RETURN -1
    end

    COMMIT
    RETURN 1
GO

-----> Bloquear Utilizador <-----          0-Pendente, 1-Ativo, 2-Bloqueado
CREATE PROCEDURE BloquearUtilizador
@ID_Admin INTEGER,
@ID_Utilizador INTEGER,
@Motivo VARCHAR(200)
AS

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

BEGIN TRANSACTION

    If(@Motivo <> null)
    begin
        INSERT INTO Bloqueia(ID_Administrador, ID_Utilizador, Data_Bloqueio, Motivo)
        VALUES (@ID_Admin, @ID_Utilizador, GETDATE(), @Motivo)
    end
    Else
    begin
        INSERT INTO Bloqueia(ID_Administrador, ID_Utilizador, Data_Bloqueio)
        VALUES (@ID_Admin, @ID_Utilizador, GETDATE())
    end
    IF (@@ERROR <> 0)
    begin
        rollback
        RETURN -1
    end

    UPDATE Utilizador
    Set Estado = 2
    Where ID_Utilizador = @ID_Utilizador
    IF (@@ERROR <> 0 or @@ROWCOUNT = 0)
    begin

```

```

        rollback
        RETURN -2
    end

    COMMIT
    RETURN 1
GO

----> Desbloquear Utilizador <----
CREATE PROCEDURE DesbloquearUtilizador
@ID_Admin INTEGER,
@ID_Utilizador INTEGER
AS

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION

    UPDATE Bloqueia
    SET Data_Desbloqueio = GETDATE()
    WHERE ID_Administrador = @ID_Admin and ID_Utilizador = @ID_Utilizador

    IF (@@ERROR <> 0)
    begin
        rollback
        RETURN -1
    end

    UPDATE Utilizador
    Set Estado = 1
    Where ID_Utilizador = @ID_Utilizador
    IF (@@ERROR <> 0 or @@ROWCOUNT = 0)
    begin
        rollback
        RETURN -2
    end

    COMMIT
    RETURN 1
GO

----> Ativar Utilizador <----
CREATE PROCEDURE AtivarUtilizador
@ID_Utilizador INT
AS

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRANSACTION

    UPDATE Utilizador
    Set Estado = 1
    Where ID_Utilizador = @ID_Utilizador

    IF (@@ERROR <> 0 or @@ROWCOUNT = 0)
    begin
        rollback
        RETURN -2
    end

    COMMIT

```

```

        RETURN 1
GO

----> Editar Admin <----
CREATE PROCEDURE EditarAdmin
@ID_Admin INTEGER,
@Pass VARCHAR(100),
@EMail VARCHAR(255)
AS

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

BEGIN TRAN

    UPDATE Administrador
    SET Pass = @Pass, Email = @EMail
    WHERE ID_Administrador = @ID_Admin

    IF (@@ERROR <> 0 OR @@ROWCOUNT = 0)
    begin
        rollback
        RETURN -1
    end

    COMMIT
    RETURN 1
GO

```