Trabalho de Pesquisa

- Transações e Concorrência em SQL Server

João Ramos (20200255)

Martim Bento (20200488)

Pedro Cunha (20200908)

16/03/2022

Faculdade de Design, Tecnologia e Comunicação
Universidade Europeia

Equipa de projecto

Apresentação de competências

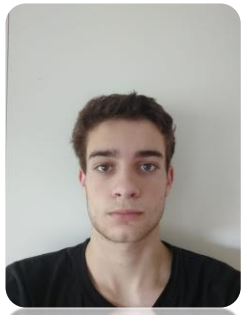
- João Ramos
- E-mail: 20200255@iade.pt

Hard Skills

- Curso técnico profissional de gestão de equipamentos informáticos
- Python,css,JavaScipt,C#,Arduino,Java,HTML ,SQL

Soft Skills

- Liderança, organização, flexibilidade , Aprendizagem Rápida



- Martim Bento
- E-mail: 20200488@iade.pt

Hard Skills

- Ensino Secundário - Ciências e Tecnologias.
- Java, Python, HTML, CSS, JavaScript, SQL

Soft Skills

- Espirito de colaboração, flexibilidade e empenho.

- Pedro Cunha
- E-mail: 20200908@iade.pt

Hard Skills

- Ensino Secundário - Ciências e Tecnologias
- Python, Java, HTML, CSS, Javascript, R, SQL
- Bases de Dados (16), Fundamentos de Programação (15), Desenvolvimento de Interfaces Web (16), Criatividade e Pensamento Crítico (17)

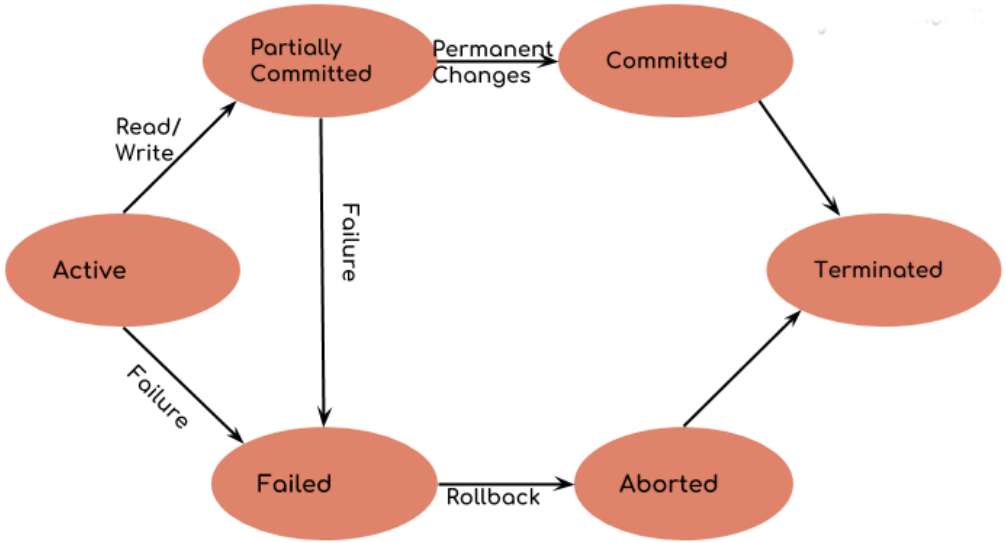
Soft Skills

- Organização, empenho, flexibilidade, colaboração

O que é uma Transação?

- Unidade lógica de trabalho processada pelo SGBD
- Uma transação consiste numa sequência de operações executadas, recorrendo a uma ou mais instruções SQL e que alteram os dados armazenados na base de dados
- Uma maneira de assegurarmos a integridade e consistência dos dados

- Os seguintes comandos são utilizados para o controlo de transações:
  - **BEGIN TRANSACTION:** Sintaxe utilizada para inicializar o bloco de instruções de uma transação
  - **COMMIT**: Quando executado as modificações efetuadas pela transação tornam-se permanentes.
  - **ROLL BACK**: Quando executado as modificações efetuadas pela transação são descartadas.
  - **SAVEPOINT**: Utilizada para identificar um ponto numa transação para o qual podemos reverter mais tarde.



Tipos de Transações

Auto Commit Transaction

- Cada instrução é uma transação por si só. Quando uma instrução produz um erro, a transação é revertida automaticamente.

Explicit Transaction

- O programador define onde a transação é inicializada e finalizada ou revertida, recorrendo a instruções de: **BEGIN TRANSACTION**, **COMMIT** e **ROLLBACK**.

Implicit Transaction

Nested Transactions e Savepoints

- As **Nested Transactions**, consistem em transações inseridas dentro de outras transações, ou seja, a primeira transação ser executada e concluída com sucesso, de seguida a segunda transação, que está contida dentro da primeira deve ser também executada e concluída com sucesso.
- **Savepoints,** instrução utilizada dentro da transação que permite desfazer todos os comandos executados após o seu estabelecimento, revertendo a transação para o seu estado quando o Savepoint foi estabelecido.

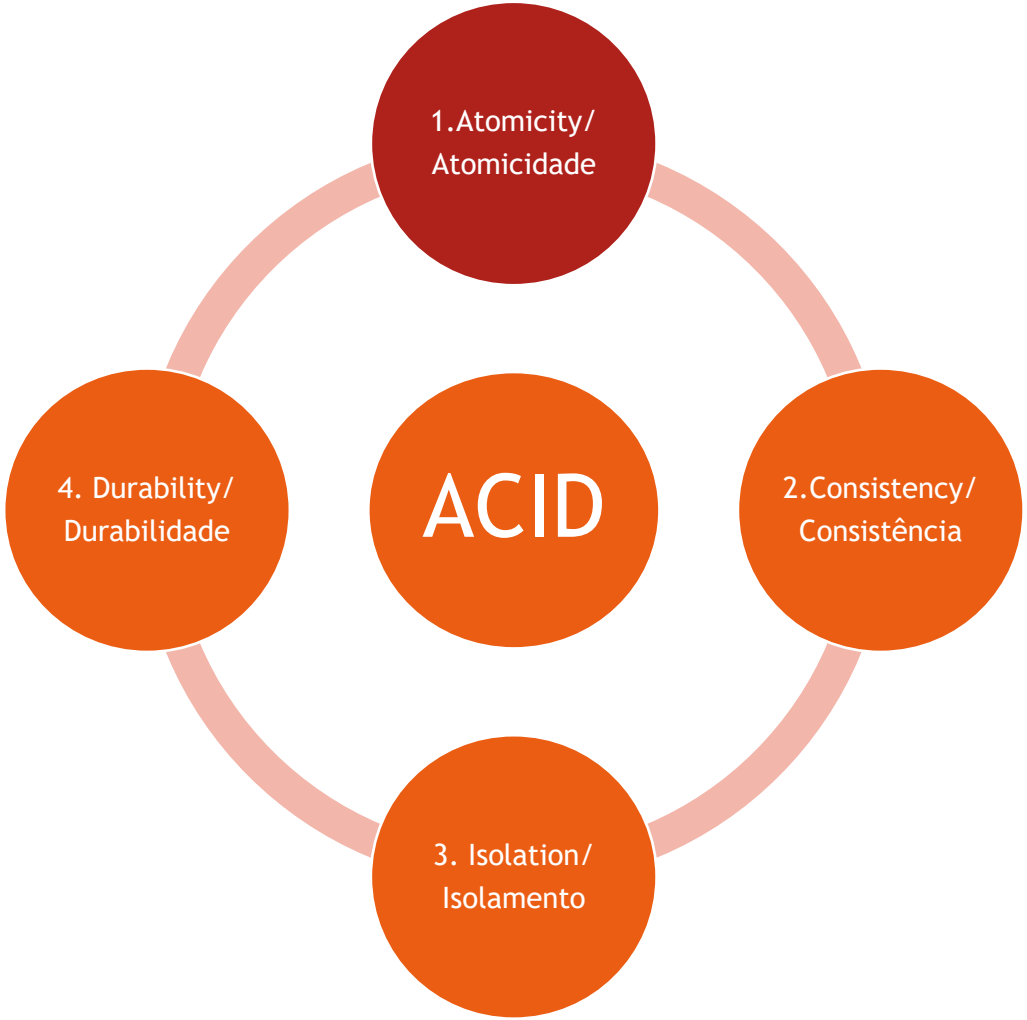
```
/*************************************************************
*  Exemplo de Nested Transaction com SavePoints
*************************************************************/
BEGIN TRANSACTION T1
    SAVE TRANSACTION SavePoint1
    insert into [dbo].[Customer] values(250,253,'Jessica','F')
    insert into [dbo].[Customer] values(190,323,'Carolina','F')

     BEGIN TRANSACTION T2
          SAVE TRANSACTION SavePoint2
               insert into [dbo].[Customer] values(450,753,'carla','F')
                insert into [dbo].[Customer] values(590,623,'lucia','F')

     COMMIT TRANSACTION T2

COMMIT TRANSACTION T1
Select * from Customer order by 1 desc

ROLLBACK TRANSACTION SavePoint2
```

O que é uma Transação?

Propriedades ACID

Um sistema de gestão de bases de dados é considerado um sistema de gestão de bases de dados relacional (SGBD-R) se seguir as propriedades transacionais, ACID, ou seja:

1. **Atomicity/Atomicidade:**
- Todas as operações dentro da transação têm de ser concluídas com sucesso, caso contrário, a transação é abortada no ponto de falha e todas as operações anteriores são revertidas para seu estado anterior.



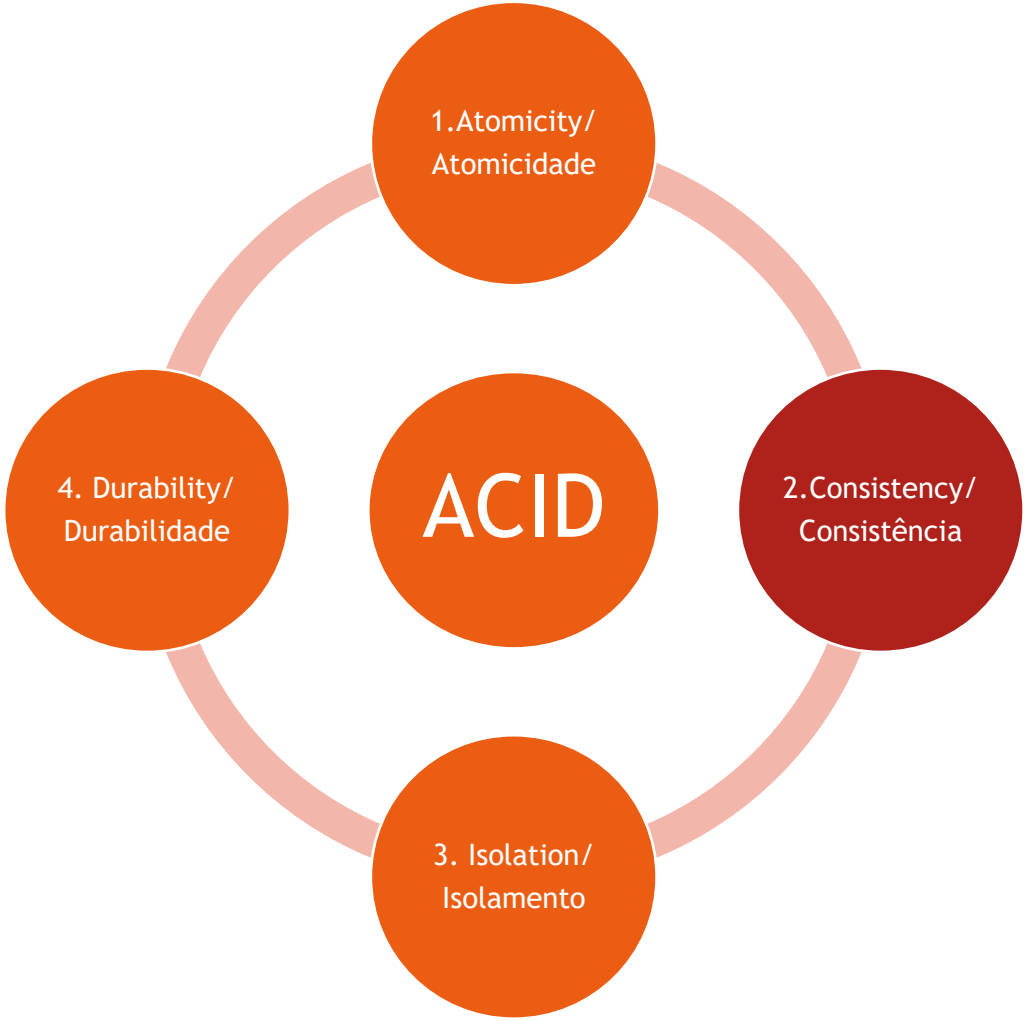
O que é uma Transação?

Propriedades ACID

Um sistema de gestão de bases de dados é considerado um sistema de gestão de bases de dados relacional (SGBD-R) se seguir as propriedades transacionais, ACID, ou seja:

2. Consistency/Consistência:

- A base de dados não deve ficar num estado inconsistente, ou seja, os dados sobre os quais a transação é aplicada devem estar logicamente corretos, de acordo com as restrições de integridade do sistema.



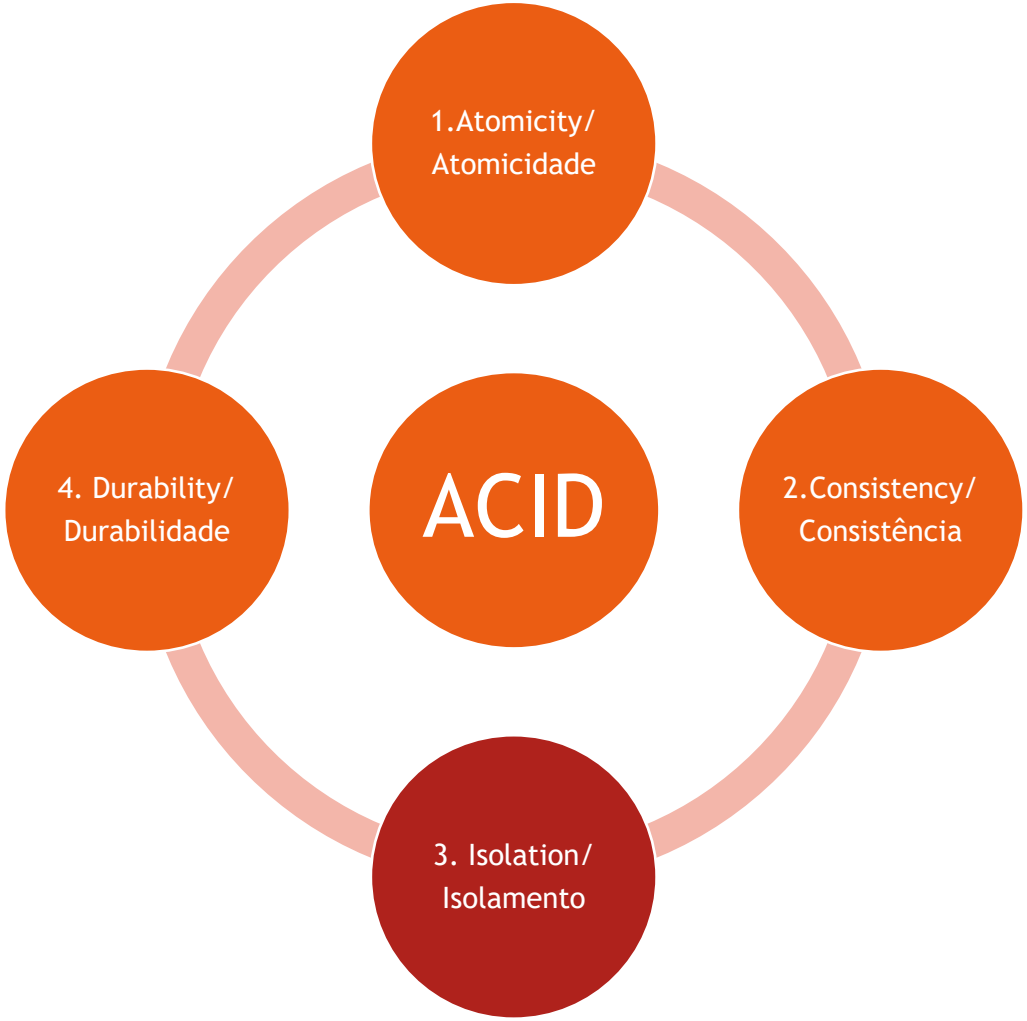
O que é uma Transação?

Propriedades ACID

Um sistema de gestão de bases de dados é considerado um sistema de gestão de bases de dados relacional (SGBD-R) se seguir as propriedades transacionais, ACID, ou seja:

3. Isolation/Isolamento:

- Se duas transações forem aplicadas numa base de dados, ambas as transações deverão ser isoladas uma da outra. Ou seja, se um processo de transação estiver a meio, a outra transação deverá aguardar até que a primeira transação seja concluída.
- Pode levar a problemas de **Concorrência.**



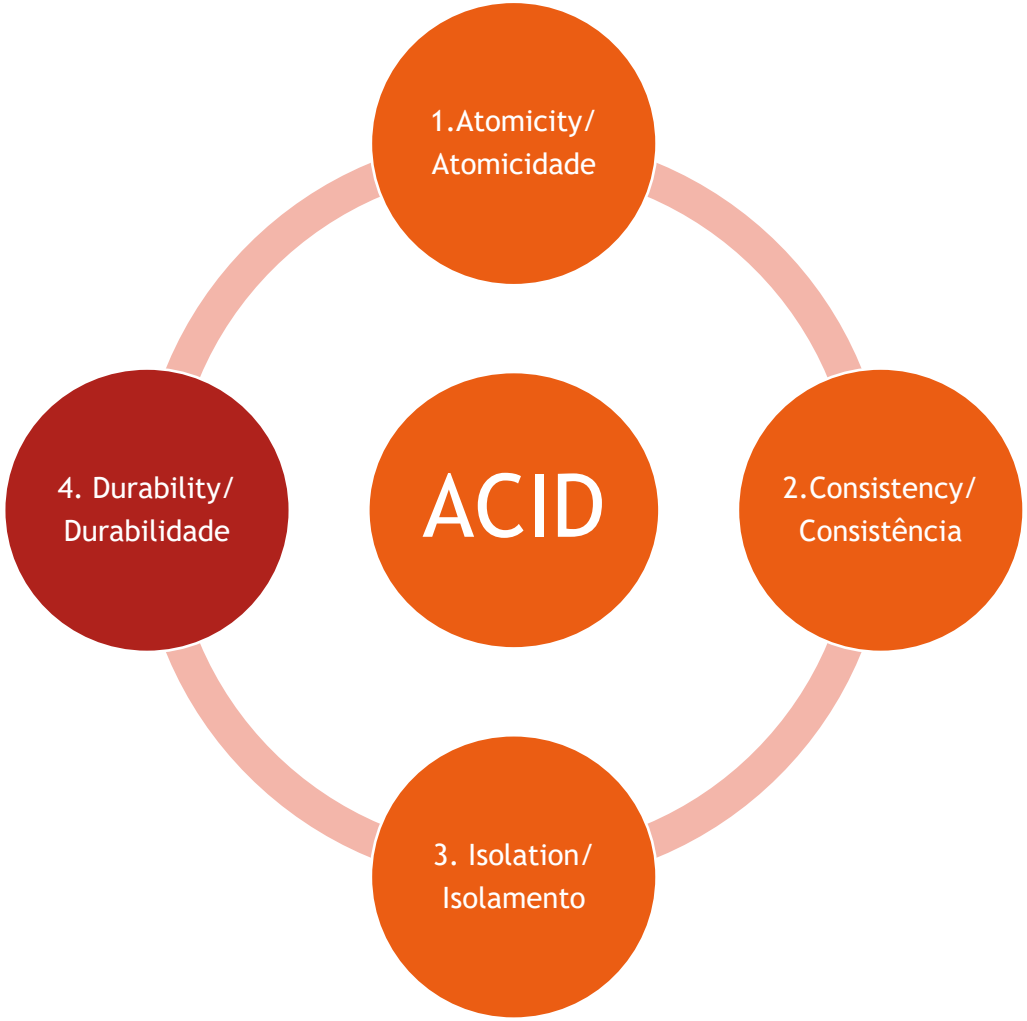
O que é uma Transação?

Propriedades ACID

Um sistema de gestão de bases de dados é considerado um sistema de gestão de bases de dados relacional (SGBD-R) se seguir as propriedades transacionais, ACID, ou seja:

4. **Durability/Durabilidade:**
- Caso o sistema falhe, a transação deve ser persistente, o que significa que, se o sistema falhar durante um processo de transação, a transação deve ser descartada, não afetando os dados.

- O SQL Server trata das vertentes de Atomicity, Consistency, e Durability do sistema, o utilizador tem apenas que que se preocupar com a propriedade de Isolation da transação.



Concorrência

Cenário: Imaginemos 2 utilizadores, o Esteves e o Bento.

- Se o Esteves executar um processo de transação no dado D1 e antes deste processo estar concluído, o Bento executa outro processo de transação no mesmo dado D1.
- A **propriedade de isolamento** isolará os processos de transação do Esteves e do Bento, e então o processo de transação do Bento só será inicializado depois do processo de transação do Esteves estar concluído.

- **Concorrência**: quando 2 ou mais utilizadores estão a tentar aceder aos mesmos dados.
  - **Problema**: O acesso simultâneo aos dados por utilizadores diferentes pode levar a resultados inconsistentes.
  - O problema da concorrência surge principalmente quando ambos os utilizadores tentam gravar os mesmos dados, ou quando um está a gravar e o outro está a ler. Os problemas mais comuns de concorrência são:
    - Dirty Reads
    - Lost Updates
    - Non-Repeatable Reads
    - Phantom Reads

Isolation Levels

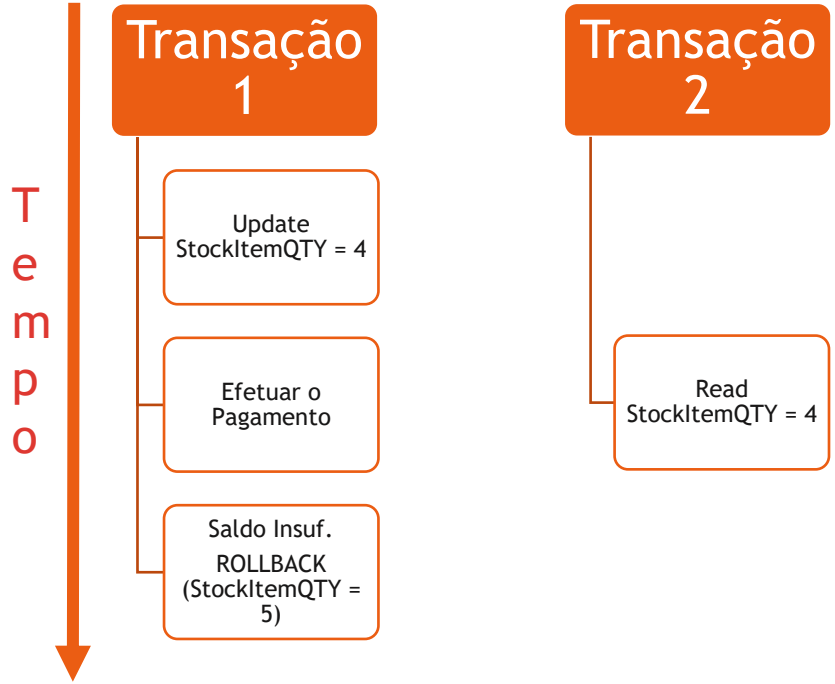
- Os Isolation Levels do SQL Server são utilizados para definir o grau em que uma transação deve ser isolada de modificações de dados feitas por outras transações simultâneas. Os diferentes níveis de isolamento estão representados na seguinte tabela, bem como os problemas de concorrência que cada um resolve:

Sintaxe:
SET TRANSACTION ISOLATION LEVEL { READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SNAPSHOT | SERIALIZABLE }

Isolation Level	Dirty Reads	Lost Update	Non Repeatable Reads	Phantom Reads
Read Uncommitted	Sim	Sim	Sim	Sim
Read Comitted	Não	Sim	Sim	Sim
Repeatable Read	Não	Não	Não	Sim
Snapshot	Não	Não	Não	Não
Serializable	Não	Não	Não	Não

Dirty Reads

- Quando uma transação lê os dados alterados por outra transação anterior a esta, mas que não foram COMMITTED ou ROLLED BACK, o que leva a um estado inconsistente para o leitor
- **Cenário**: Um cliente pretende adquirir uma Bola. O Vendedor 1 dá baixa do produto no stock ao ler o código de barras - o stock passa de 5 para 4.
  - Logo de seguida, o Vendedor 2 verifica o stock de bolas e verifica que existem 4 em stock.
  - No entanto, quando o cliente realiza o pagamento da bola, a transação é recusada pois não possui saldo suficiente na conta bancária - o stock de bolas é atualizado para 5.



Transação 1

```
/*****************************************************************
*  Exemplo de Dirty Read
*****************************************************************/
BEGIN TRANSACTION
    UPDATE Stock SET StockItemQTY=0 WHERE StockItemNO=1;
    --A passar a fatura
    WAITFOR DELAY '00:00:15'
    -- sem dinheiro na conta disponivel
ROLLBACK TRANSACTION

SELECT * FROM Stock WHERE StockItemNO=1;
```

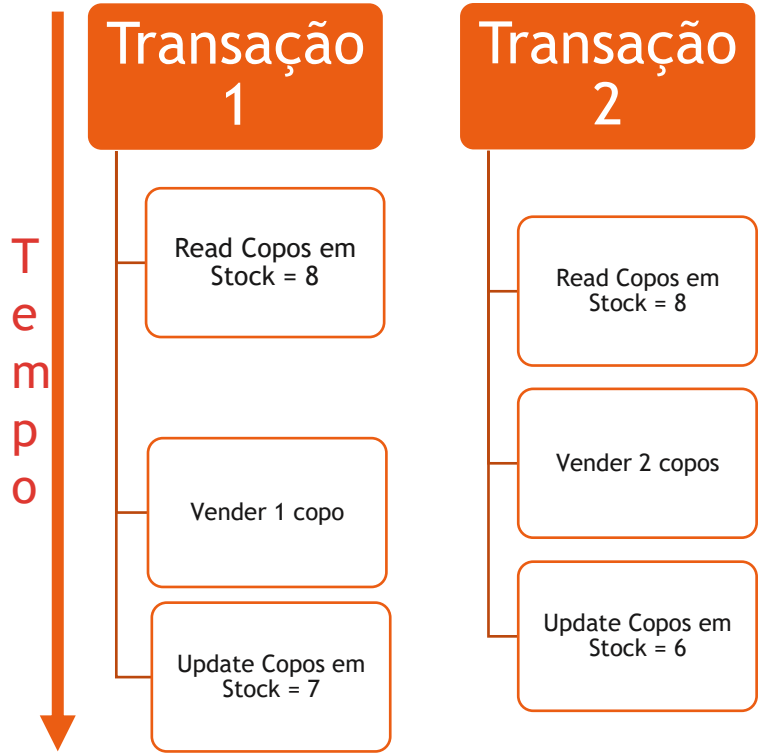
Transação 2

```
/*****************************************************************
*  Exemplo de Dirty Read
*****************************************************************/

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED -- Ler dados que ainda n foram comited
SELECT * FROM Stock WHERE StockItemNO=1;

/*****************************************************************
* Solução de Dirty Read
*****************************************************************/

SET TRANSACTION ISOLATION LEVEL READ COMMITTED -- Ler dados que ja foram comited
SELECT * FROM Stock WHERE StockItemNO=1;
--ou
SELECT * FROM Stock (NOLOCK) WHERE StockItemNO=1
```

Lost Updates

- Quando duas transações distintas tentam manipular os mesmos dados simultaneamente, isto pode levar à perda de dados ou então, a segunda transação pode substituir dados que já tinham sido alterados na primeira transação.

- **Cenário**: Dois Vendedores distintos visualizam a quantidade de Copos em stock com poucos segundos de diferença, entretanto o vendedor 2 efetua uma venda de 2 copos atualizando o stock para 6, depois disto o vendedor 1 efetua a venda de 1 copo, o que fará com que o stock diminua para 7 em vez de 6.



Transação 1

```
/*************************************************************
*  Exemplo de Lost Update
*************************************************************/
BEGIN TRANSACTION
  DECLARE @QuantidadeDisponivel int
  SELECT @QuantidadeDisponivel = StockItemQTY FROM Stock WHERE StockID=2
  -- a fazer a transação
  WAITFOR DELAY '00:00:10'
  SET @QuantidadeDisponivel = @QuantidadeDisponivel - 1
  UPDATE Stock SET StockItemQTY = @QuantidadeDisponivel  WHERE StockID=2
  Print @QuantidadeDisponivel
COMMIT TRANSACTION

SELECT *FROM Stock where StockID=2
```

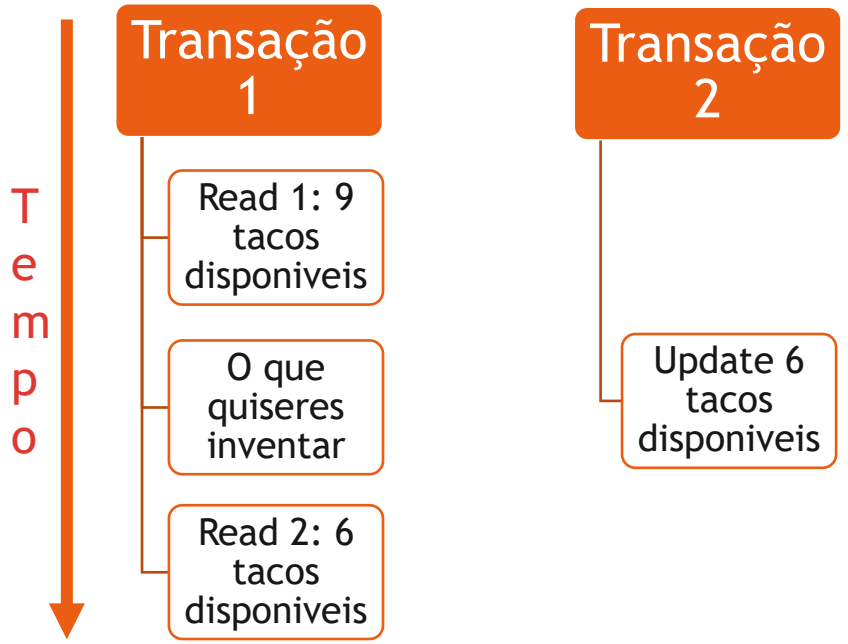
Transação 2

```
/*************************************************************
*  Exemplo de Lost Update
*************************************************************/
 BEGIN TRANSACTION
  DECLARE @QuantidadeDisponivel int
  SELECT @QuantidadeDisponivel = StockItemQTY FROM Stock WHERE StockID=2
  SET @QuantidadeDisponivel = @QuantidadeDisponivel - 2
  UPDATE Stock SET StockItemQTY = @QuantidadeDisponivel WHERE StockID=2
  Print @QuantidadeDisponivel
COMMIT TRANSACTION
```

Non-Repeatable Reads

- Este problema ocorre quando uma transação lê dados duas vezes e no meio dessas leituras é efetuada uma segunda transação que altera os dados de uma leitura para a outra.

- **Cenário**:O cliente 1 está a fazer compras online e pretende comprar 8 tacos ,verifica que existem 9 tacos em stock, entretanto vai almoçar, enquanto isto, um cliente 2 adquire 3 tacos. Quando o cliente 1 retoma as suas compras verifica que só existem 6 tacos em vez de 9 como tinha verificado anteriormente.



Transação 1

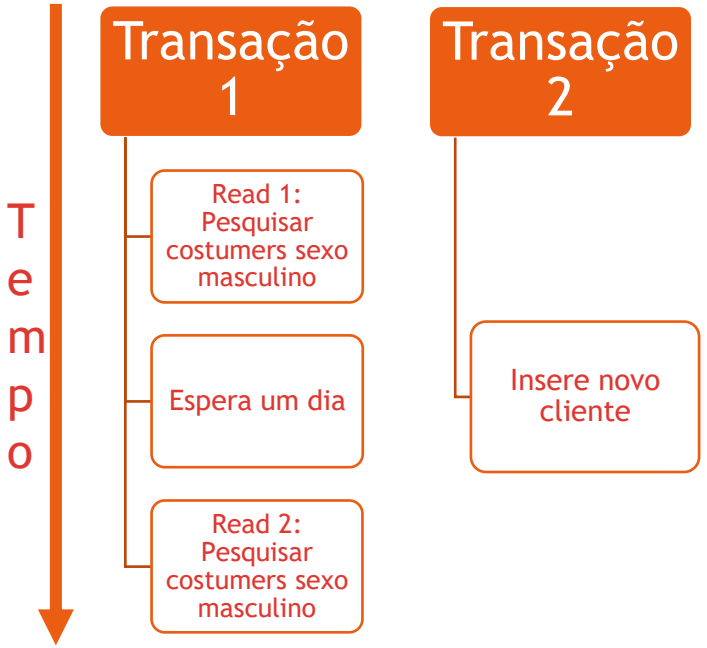
```
/*************************************************************
*  Exemplo de Non Repeatable Read
*************************************************************/
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRANSACTION
SELECT StockItemQTY FROM Stock WHERE StockID = 3
-- a fazer algo
WAITFOR DELAY '00:00:10'
SELECT StockItemQTY FROM Stock WHERE StockID = 3
COMMIT TRANSACTION
```

Transação 2

```
/*************************************************************
*  Exemplo de Non Repeatable Read
*************************************************************/

SET TRANSACTION ISOLATION LEVEL READ COMMITTED
UPDATE Stock SET StockItemQTY = 10 WHERE StockID = 3;
```

Phantom reads

- Quando uma transação executa uma query duas vezes e retorna um numero diferente de linhas para cada execução. Isto acontece quando uma segunda transação insere uma nova linha numa tabela, e essa linha satisfaz uma clausula WHERE da query executada pela primeira transação
- **Cenário** Uma loja está a fazer um estudo para uma campanha de marketing e é necessário estudar alguns dados dos clientes do sexo masculino. É feita uma leitura na base de dados e são apresentados 4 clientes. No dia seguinte é inserido um novo cliente na base de dados, e portanto se for feita uma nova leitura na base de dados serão apresentados 5 clientes em vez de 4.



Transação 1

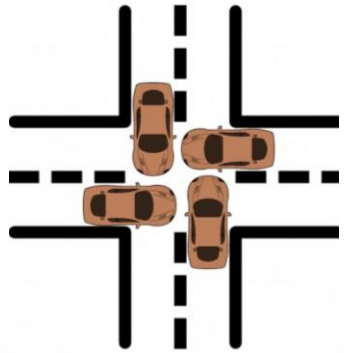
```
/*************************************************************
*  Exemplo de Phantom Read
*************************************************************/

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION
SELECT * FROM Customer where CustomerGender='M'; -- DEVOLVE 4 LINHAS
-- a fazer algo
WAITFOR DELAY '00:00:10'
SELECT * FROM Customer where CustomerGender='M';-- DEVOLVE 5 LINHAS
COMMIT TRANSACTION
```

Transação 2

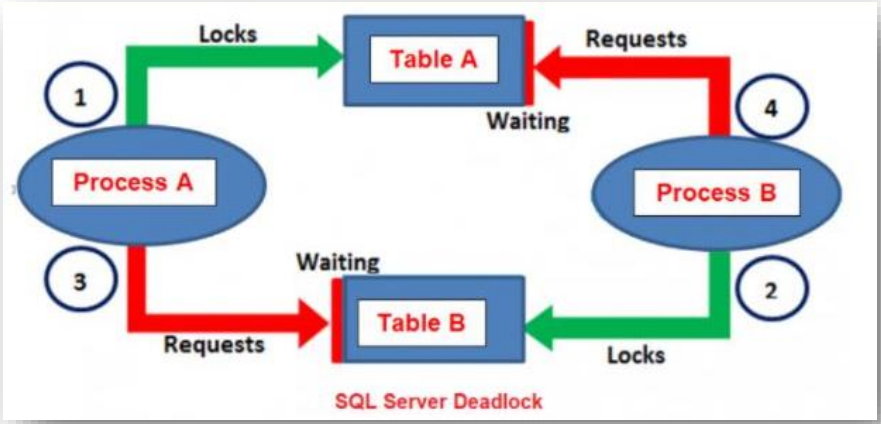
```
/*************************************************************
*  Exemplo de Phantom Read
*************************************************************/

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRANSACTION
insert into Customer values
(18,178,'jao almeida','M');
COMMIT TRANSACTION
```

Deadlock

O que é?



- Um deadlock ocorre quando a base de dados deteta que 2 ou mais processos que estão a aguardar uns pelos outros para continuarem as suas atividades.
- Se a base de dados não detetar o deadlock e eliminar um dos processos em espera, todos os processos ficarão em espera eternamente.
- Quando a base de dados deteta um deadlock, cancela um dos processos, para que os demais possam prosseguir.
- O processo que é cancelado é chamado de deadlock victim.
- O único problema, é que o deadlock victim fica em estado de erro, e portanto o utilizador precisará de executar novamente as instruções, isto porque o SQL não faz nada para além o cancelar




```
BEGIN TRANSACTION
    INSERT INTO CUSTOMER values(10000,1000,'joana','M')
    --executar transação 2
    DELETE CUSTOMER
    ROLLBACK
```

```
BEGIN TRANSACTION
    INSERT INTO CUSTOMER values(10010,1100,'joanaS','M')

    DELETE CUSTOMER
ROLLBACK
```

Messages

```
Msg 1205, Level 13, State 51, Line 84
Transaction (Process ID 66) was deadlocked on lock resources with another process and has been chosen as the deadlock victim. Rerun the transaction.

Completion time: 2022-03-13T16:42:46.4656470+00:00
```

Capturar o deadlock

Diferentes formas de capturar a ocorrência de um deadlock

Para monitorar um deadlock temos várias formas:

- Flags
- Extended events

```
/************************************************************
*  Consultar log de DeadLock
************************************************************/
if OBJECT_ID('tempdb.dbo.#error_log') is not null drop table #error_log
create table #error_log (logdate datetime,processinfo varchar(1000),text varchar(max))
insert into #error_log exec xp_ReadErrorLog
select * from #error_log where logdate in (select logdate from #error_log where text like 'DeadLock encountered%')
```

110 %

Results | Messages

	logdate	processinfo	text
1	2022-03-10 22:01:35.400	spid7s	Deadlock encountered .... Printing deadlock information
2	2022-03-10 22:09:08.660	spid7s	Deadlock encountered .... Printing deadlock information
3	2022-03-10 22:09:08.660	spid7s	Wait-for graph
4	2022-03-10 22:09:08.660	spid7s	
5	2022-03-10 22:09:08.660	spid7s	Node:1
6	2022-03-10 22:09:08.660	spid7s	KEY: 1:72057594043498496 (8194443284a0) CleanCnt:...
7	2022-03-10 22:09:08.660	spid7s	Grant List 0:
8	2022-03-10 22:09:08.660	spid7s	Owner:0x00000193F2A6E040 Mode: X Flg:0x40 R...
9	2022-03-10 22:09:08.660	spid7s	SPID: 67 ECID: 0 Statement Type: DELETE Line #: 1
10	2022-03-10 22:09:08.660	spid7s	Input Buf: Language Event: DELETE CUSTOMER
11	2022-03-10 22:09:08.660	spid7s	Requested by:
12	2022-03-10 22:09:08.660	spid7s	ResType:LockOwner Stype:'OR'Xdes:0x00000193EDB...
13	2022-03-10 22:09:08.660	spid7s	
14	2022-03-10 22:09:08.660	spid7s	Node:2
15	2022-03-10 22:09:08.660	spid7s	KEY: 1:72057594043498496 (8f2db235afd3) CleanCnt:2 ...
16	2022-03-10 22:09:08.660	spid7s	Grant List 2:
17	2022-03-10 22:09:08.660	spid7s	Owner:0x00000193F2A69F80 Mode: X Flg:0x40 R...
18	2022-03-10 22:09:08.660	spid7s	SPID: 66 ECID: 0 Statement Type: DELETE Line #: 1
19	2022-03-10 22:09:08.660	spid7s	Input Buf: Language Event: DELETE CUSTOMER
20	2022-03-10 22:09:08.660	spid7s	Requested by:
21	2022-03-10 22:09:08.660	spid7s	ResType:LockOwner Stype:'OR'Xdes:0x00000193E358...
22	2022-03-10 22:09:08.660	spid7s	
23	2022-03-10 22:09:08.660	spid7s	Victim Resource Owner:
24	2022-03-10 22:09:08.660	spid7s	ResType:LockOwner Stype:'OR'Xdes:0x00000193EDBC...
25	2022-03-10 22:09:08.660	spid45s	deadlock-list
26	2022-03-10 22:09:08.660	spid45s	deadlock victim=process193f807d088
27	2022-03-10 22:09:08.660	spid45s	process-list
28	2022-03-10 22:09:08.660	spid45s	process id=process193f807d088 taskpriority=0 logused...

Referências Bibliográficas

- “Microsoft SQL Documentation - SQL Server,” *SQL Server | Microsoft Docs*. [Online]. Available: https://docs.microsoft.com/en-us/sql/?view=sql-server-ver15. [Accessed: 13-Mar-2022].
- “Transactions (transact-SQL) - SQL server,” *SQL Server | Microsoft Docs*. [Online]. Available: https://docs.microsoft.com/en-us/sql/t-sql/language-elements/transactions-transact-sql?view=sql-server-ver15. [Accessed: 13-Mar-2022].
- *Transaction in DBMS*. [Online]. Available: https://www.w3schools.in/dbms/transaction/. [Accessed: 13-Mar-2022].
- “Monitoring SQL database deadlocks,” *Business Central | Microsoft Docs*. [Online]. Available: https://docs.microsoft.com/en-us/dynamics365/business-central/dev-itpro/administration/monitor-database-deadlocks. [Accessed: 13-Mar-2022].

Yes we Can!

Faculdade de Design,
Tecnologia e Comunicação
Universidade Europeia