



Faculdade
IMPACTA
TECNOLOGIA



Linguagem SQL / Banco de Dados

Aula 08 - Transações (ACID) e tratamento de erros

Gustavo Bianchi Maia
gustavo.maia@faculddeimpacta.com



Agenda

- Definição: ACID
- Transações
- Tratamento de erros
- Demonstrações





Alta Concorrência - Overview

- Sistemas de banco de dados devem permitir que um grande número de conexões simultâneas faça uso de seus dados para leitura ou escrita de informações.
- Um dos objetivos dos sistemas de gerenciamento de banco de dados é fornecer a cada usuário a ilusão, sempre que possível, que é o único usuário no sistema.
- Sistemas de banco de dados lutam com a necessidade de equilibrar a consistência e simultaneidade. Muitas vezes há um trade-off ("perde-e-ganha") direto entre estes dois objetivos. O desafio é reduzir o impacto da concorrência, mantendo a consistência suficiente.





- Uma transação é uma sequência de operações executadas como uma única unidade lógica de trabalho.

```
INSERT INTO Veiculo VALUES (1, 'Strada')
```

```
UPDATE Veiculo SET Descricao = 'Strada Adventure'  
WHERE id = 1
```

```
INSERT INTO Veiculo VALUES (2, 'Montana')
```

```
INSERT INTO Veiculo VALUES (3, 'Saveiro')
```

```
DELETE Veiculo WHERE id = 1
```



Propriedades das Transações - ACID

- Transações devem apresentar quatro propriedades que são conhecidas como ACID, responsáveis por assegurar que várias modificações de dados são processadas como uma unidade. Por exemplo, uma transação bancária pode creditar uma conta e debitar outra. Ambas as etapas devem ser concluídas em conjunto ou não concluídas (desfazendo qualquer modificação realizada). SGBDs devem suportar o processamento de transações para gerenciar múltiplas transações.

Atomicidade

Consistência

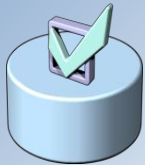
Isolamento

Durabilidade

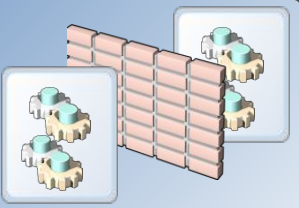




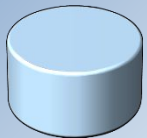
Uma transação é uma unidade de trabalho **Atômica**



Uma transação deixa os dados num estado **Consistente**



Uma transação é **Isolada** de outras transações correntes



Uma transação é **Durável**



- Transações usam LOCK (travas) para impedir que outros usuários alterem ou leiam os dados em uma transação que não foi concluída.
- Locks são a reserva de recursos para serem utilizados em uma transação.
- O BLOCK (bloqueio) é necessário no processamento de transações online (OLTP) para sistemas multi-usuário.
- Blocks são causados quando mais uma transação deseja utilizar os recursos já reservados (LOCK) por outro processo.





- Existem 3 tipos de transações
 - Explícitas - declaradas intencionalmente pelo usuário (manual)
 - Implícitas - abertas automaticamente pelo cliente / sessão.
 - Automáticas - abertas e gerenciadas pelo sistema.





Transação Explícita

É uma transação que é definida explicitamente através de um início e um final. Isso é feito com as declarações abaixo

- **BEGIN TRANSACTION**
- **COMMIT TRANSACTION**
- **ROLLBACK TRANSACTION**

```
BEGIN TRANSACTION FundsTransfer;  
    EXEC Banking.DebitAccount '100', 'account1';  
    EXEC Banking.CreditAccount '100', 'account2';  
COMMIT TRANSACTION;
```



Transação Explícita – Testes

```
SELECT * FROM cliente
```

```
/*
```

```
id nome          cpf          telefone professor
```

```
-----
```

```
1 Almir dos Santos 78654552421 (11)91234-5678 0
```

```
*/
```

```
--Teste ROLLBACK
```

```
BEGIN TRANSACTION
```

```
INSERT INTO cliente(nome,cpf,telefone,professor)
```

```
SELECT 'Klaus Petherson','12345678901','11976531251',1
```

```
ROLLBACK TRANSACTION
```

```
SELECT * FROM cliente
```

```
/*
```

```
id nome          cpf          telefone professor
```

```
-----
```

```
1 Almir dos Santos 78654552421 (11)91234-5678 0
```

```
0 que mais ele retornou ?
```

```
*/
```





Transação Explícita – Testes

```
SELECT * FROM cliente
```

```
/*
```

```
id nome          cpf          telefone    professor
```

```
-----
```

```
1 Almir dos Santos 78654552421 (11)91234-5678 0
```

```
*/
```

```
--Teste COMMIT
```

```
BEGIN TRANSACTION
```

```
INSERT INTO cliente(nome,cpf,telefone,professor)
```

```
SELECT 'Klaus Petherson','12345678901','11976531251',1
```

```
COMMIT TRANSACTION
```

```
SELECT * FROM cliente
```

```
/*
```

```
id nome          cpf          telefone    professor
```

```
-----
```

```
1 Almir dos Santos 78654552421 (11)91234-5678 0
```

```
0 que mais ele retornou ?
```

```
*/
```





Transação Explícita – ISOLAMENTO

- Note que a transação segue as propriedades ACID.
- Por padrão, há o ISOLAMENTO dos dados enquanto a transação estiver ABERTA (sem receber Commit ou Rollback), ou seja, os dados que estão sendo alterados, só ficam disponíveis para a transação que estiver atuando na alteração (transação ativa).
- Demais SESSÕES que precisarem verificar os dados que estão sendo modificados, não conseguirão acesso até que a transação ativa seja FECHADA (quando receber Commit ou Rollback).





Transação Explícita – ISOLAMENTO

- Esse é um mecanismo inerente e necessários em bancos transacionais, não há nenhum problema com este comportamento, pelo contrário, este mecanismo é o que garantirá a consistência das informações.
- Há opções de mudar este comportamento natural do banco de dados (nolock, set isolation, ...) que são opções avançadas e geram efeitos diferentes em SELECT, UPDATE, DELETE e INSERT, por isso devem ser evitados, a menos que sejam bem estudados para esta alteração.





ISOLAMENTO - Testes

*/*Testes de isolamento*/*

--Sessão 1

SELECT * FROM cliente

*/**
id nome cpf telefone professor

1 Almir dos Santos 78654552421 (11)91234-5678 0

**/*

BEGIN TRANSACTION

UPDATE cliente

SET nome = 'Joacir dos Santos'

WHERE nome = 'Almir dos Santos'

SELECT * FROM cliente

*/**
id nome cpf telefone professor

1 ????? dos Santos 78654552421 (11)91234-5678 0

Almir ou Joacir dos Santos ?

**/*





ISOLAMENTO - Testes

*/*Testes de isolamento*/*

--Sessão 2

SELECT * FROM cliente WITH(nolock)

*/**
id nome cpf telefone professor

1 ????? dos Santos 78654552421 (11)91234-5678 0

Almir ou Joacir dos Santos ?

**/*

SELECT * FROM cliente

*/**

O que aconteceu ?

**/*

*/**

O que é o WITH(NOLOCK) ?

**/*





Transação – Auto Commit

- Auto Commit (Modo de transação Default) - Motor de banco de dados opera em **autocommit** até que uma transação explícita seja iniciada.
- Toda declaração TSQL é confirmada (**Commit**) ou revertida (**Rollback**) quando for concluída. **Confirmada** se for bem sucedida e **Revertida** em caso de erro.
- Erros de COMPILAÇÃO não permitem a execução do BATCH.
- Erros em TEMPO DE EXECUÇÃO (**RUNTIME errors**) podem permitir que parte do lote seja Confirmado.
- XACT_ABORT configurado para ON converte declarações terminadas com erros de **Run Time** em erros de compilação, isso faz com que um simples erro em tempo de execução não seja confirmado dentro de um lote, cancelando assim toda a operação.

```
SET XACT_ABORT ON;
```




Transação Implícita

- Definindo o mode de transação implícita

```
SET IMPLICIT_TRANSACTIONS ON;
```

- Uma transação implícita começa quando uma das seguintes declarações é executada e a declaração não faz parte de uma transação existente

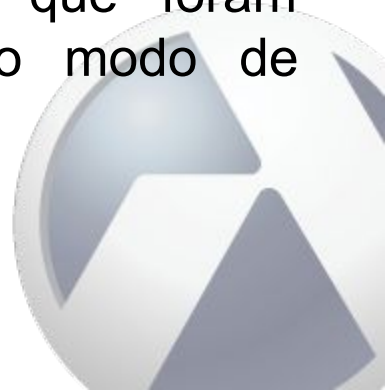
ALTER TABLE	INSERT
CREATE	OPEN
DELETE	REVOKE
DROP	SELECT
FETCH	TRUNCATE TABLE
GRANT	UPDATE

- Transação deve ser explicitamente concluída com COMMIT or ROLLBACK TRANSACTION



Transação Implícita

- Por padrão, o modo de transação implícita está desligado (default), o banco de dados funciona em modo de confirmação automática (auto commit).
- Transações aninhadas (onde uma transação é iniciada dentro de outra transação) não são permitidas no modo de transação implícita. Se a conexão já está em uma transação, essas declarações não iniciam uma nova transação.
- Na maioria dos casos, é melhor trabalhar no modo de confirmação automática e definir operações explicitamente usando a instrução `BEGIN TRANSACTION`. No entanto, para aplicações que foram originalmente desenvolvidas em diferentes sistemas, o modo de transação implícita pode ser útil.





Transação Implícita - Testes

```
SET implicit_transactions ON;
```

```
INSERT INTO cliente(nome,cpf,telefone,professor)  
SELECT 'Edward Strays','14367866578','11878764351',      1
```

```
/*  
    Ele entrou em uma transação, como fazer para verificar ?  
    Ele já salvou o cliente Edward em definitivo ?  
    O que acontece se eu esquecer o COMMIT ?  
*/
```





Há uma série de considerações gerais que precisam ser mantidas em mente quando se trabalha com transações.

- **Mantenha Transações mais curtas possível**

- Transações mais longas aumentam a probabilidade de que os usuários não serão capazes de acessar os dados bloqueados.
- Não abra a transação enquanto navega através de dados, se possível. Transações não devem começar até que todas as análises de dados preliminares sejam concluídas.
- INSERT, UPDATE e DELETE devem ser as primeiras declarações em uma transação e elas devem ser escritas para **afetar o menor número de linhas**.
- **Acesse a menor quantidade possível de dados durante uma transação.** Isto diminui o número de linhas bloqueadas e reduz a contenção de dados.
- Certifique-se de que a indexação está apropriada, pois isso reduz o número de páginas que precisam ser acessadas e bloqueadas.





***Menos indicado:**

```
begin tran
    while
        begin
        end
commit
```

***Mais indicado:**

```
while
    begin
        begin tran
        commit
    end
```

(*) às vezes, apesar de menos indicado, seu processo precisa que TODO o while seja transacionado, nestas situações, o primeiro exemplo é o único que nos atende.





Tratamento de erros

- Erro é a identificação de que algo não aconteceu como previsto.
- O objetivo de uma mensagem de erro é informar o usuário que está executando a instrução do que precisa ser corrigido, orientando-lhe na busca pela solução.

EX:

```
SELECT * FROM cliente
```

```
/*
```

```
  id  nome          cpf          telefone    professor
```

```
-----
```

```
  1  Almir dos Santos  78654552421  (11)91234-5678  0
```

```
*/
```

```
INSERT INTO cliente(nome,cpf,telefone,professor)
```

```
SELECT 'Almir dos Santos','78654552421','11912345678',1
```

Mensagem 2627, Nível 14, Estado 1, Linha 151

Violação da restrição UNIQUE KEY 'UQ_ClienteCPF'. Não é possível inserir a chave duplicada no objeto 'dbo.cliente'. O valor de chave duplicada é (78654552421).

A instrução foi finalizada.

- Todo erro possui um número e uma descrição
- Erros são classificados em níveis (ou severidades) e estados





Níveis ou Severidades:

0-9	Mensagens informativas que retornam informações de status ou reportam erros que não sejam severos. O Mecanismo de Banco de Dados não gera erros de sistema com severidades de 0 a 9.
10	O Mecanismo converte a severidade 10 em 0 antes de retornar as informações de erro ao aplicativo.
11-16	Indica erros que podem ser corrigidos pelo usuário.
11	Indica que um determinado objeto ou entidade não existe.
12	Severidade especial para consultas que não usam bloqueio por causa de dicas de consulta especiais.
13	Indica erros de deadlock de transação.
14	Indica erros relacionados à segurança, como uma permissão negada.
15	Indica erros de sintaxe no comando Transact-SQL.
16	Indica erros gerais que podem ser corrigidos pelo usuário.



Níveis ou Severidades:

-
- | | |
|-------|--|
| 17-19 | Indica erros de software que não podem ser corrigidos pelo usuário.
O usuário deve informar o problema ao seu administrador de sistema. |
|-------|--|
-
- | | |
|----|---|
| 17 | Indica que a instrução fez o SQL Server ficar sem recursos (como memória, bloqueios ou espaço em disco para o banco de dados) ou exceder algum limite definido pelo administrador de sistema. |
|----|---|
-
- | | |
|----|---|
| 18 | Indica um problema no software Mecanismo de Banco de Dados , mas a instrução conclui a execução e a conexão com a instância do Mecanismo de Banco de Dados é mantida. |
|----|---|
-
- | | |
|----|--|
| 19 | Indica que um limite do Mecanismo de Banco de Dados não configurável foi excedido e que o processo em lotes atual foi encerrado. Mensagens de erro com nível de severidade 19 ou maior pararam a execução do lote atual. |
|----|--|
-





Tratamento de erros

Níveis ou Severidades:

20-24 Indique problemas de sistema que são erros fatais, ou seja, a tarefa do Mecanismo de Banco de Dados que está executando uma instrução ou um lote que não está mais em execução. Mensagens de erro nesse intervalo podem afetar todos os processos que acessam dados no mesmo banco de dados e indicar que um banco de dados ou objeto está danificado.

Mensagens de erro com nível de severidade de 19 a 24 são gravadas no log de erros.

20 Indica que uma instrução encontrou um problema. Como o problema afetou apenas a tarefa atual, é improvável que o banco de dados tenha sido danificado.

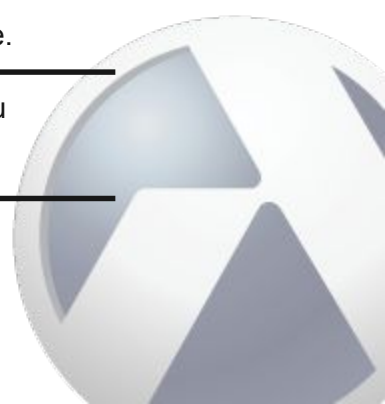
21 Indica que foi encontrado um problema que afeta todas as tarefas no banco de dados atual, mas é improvável que o banco de dados tenha sido danificado.

22 Indica que a tabela ou o índice especificado na mensagem foi danificado por um problema de software ou hardware.

Por exemplo, se a mensagem informar que a instância do Mecanismo de Banco de Dados encontrou uma linha com comprimento 0 em um índice não clusterizado, exclua o índice e crie-o novamente.

23 Indica que a integridade do banco de dados inteiro está em risco por um problema de software ou hardware.

24 Indica uma falha de mídia. O administrador de sistema pode ter que restaurar o banco de dados. Também pode ser necessário contatar o seu fornecedor de hardware.





Gerando mensagens de erro formatadas:

```
RAISERROR('Por favor Digite um valor entre 1 e 10, valor fornecido: %d',16,1,99)
```

```
RAISERROR(  
'Número de veículos estacionados (%d) é superior à capacidade do estabelecimento (%d)'  
,16,1,13,10)
```

```
DECLARE @datahoraTextual VARCHAR(255) = CONVERT(VARCHAR, Getdate(), 113)
```

```
RAISERROR('Só são aceitos estacionamento até às 12:00, agora são: %s',16,1  
,@datahoraTextual)
```

Fonte:

<https://docs.microsoft.com/pt-br/sql/t-sql/language-elements/raiserror-transact-sql>

⊗ Cuidado

Níveis de severidade de 20 a 25 são considerados fatais. Se um nível de severidade fatal for encontrado, a conexão de cliente é encerrada depois de receber a mensagem, e o erro é registrado nos logs de erro e de aplicativo.



Tratamento de erros - THROW

Gerando mensagens de erro:

*--Comparação entre throw e raiserror
--O parâmetro error_number não precisa ser definido em sys.messages.
--O parâmetro message não aceita a formatação de estilo printf.
--Não há nenhum parâmetro severity. Quando THROW é usado para iniciar a exceção, a severidade é sempre definida como 16*

```
THROW 51000, 'Id do cliente não encontrado.', 1;
```

```
THROW 51000, 'Id do cliente não encontrado.', 1;
```





Tratamento de erros - @@ROWCOUNT

Podemos utilizar o **@@rowcount** para determinar se uma operação devolveu o número esperado de linhas:

BEGIN TRAN

```
DECLARE @linhasAfetadas INT = 0
```

```
SELECT @linhasAfetadas = Count(*)  
FROM estacionamento  
WHERE idveiculo = 1
```

```
UPDATE estacionamento SET valorcobrado += 0.10
```

```
IF @@ROWCOUNT = @linhasAfetadas  
    COMMIT  
ELSE  
    ROLLBACK
```

```
IF @@TRANCOUNT > 0  
    RAISERROR('Ooops, você não deveria ainda estar dentro de uma transação' 16,1)
```





Tratamento de erros - @@ERROR

Podemos utilizar o @@error para capturar o erro da **última** instrução executada.

```
BEGIN TRAN
```

```
    DECLARE @erroCapturado INT = 0
```

```
    SELECT * FROM estacionamento WHERE valorcobrado / valorcobrado > 0
```

```
    SET @erroCapturado = @@error
```

```
    IF @erroCapturado = 0
```

```
        COMMIT
```

```
    ELSE IF @erroCapturado = 8134
```

```
        BEGIN
```

```
            PRINT 'Sim, você realmente dividiu por zero'
```

```
            ROLLBACK
```

```
        END
```

```
IF @@TRANCOUNT > 0
```

```
    RAISERROR('Ooops, você não deveria ainda estar dentro de uma transação', 16, 1)
```





Tratamento de erros - @@ERROR

Por que o código anterior estava correto e este parece não fazer o esperado ???

BEGIN TRAN

```
SELECT * FROM estacionamento WHERE valorcobrado / valorcobrado > 0
```

```
IF @@error = 0
```

```
    COMMIT
```

```
ELSE IF @@error = 8134
```

```
    BEGIN
```

```
        PRINT 'Sim, você realmente dividiu por zero'
```

```
        ROLLBACK
```

```
    END
```

```
IF @@TRANCOUNT > 0
```

```
    RAISERROR('Ops, você não deveria ainda estar dentro de uma transação',16,1)
```





Tratamento de erros - Try Catch

TRY é um comando que encapsula um comando (ou bloco de comandos) e que, caso um erro ocorra, ele redireciona o fluxo da execução para a instrução de captura (**CATCH**).

```
BEGIN try
    -- Generate a divide-by-zero error.
    SELECT 1 / 0;
END try
BEGIN catch
    SELECT Error_number()    AS ErrorNumber,
           Error_severity()  AS ErrorSeverity,
           Error_state()     AS ErrorState,
           Error_procedure() AS ErrorProcedure,
           Error_line()      AS ErrorLine,
           Error_message()   AS ErrorMessage;
END catch;

go
```





Tratamento de erros

A instrução Throw pode ser reenviada mesmo depois de capturada dentro de um bloco CATCH.

```
CREATE TABLE dbo.testrethrow (    id INT PRIMARY KEY    );

BEGIN try
    INSERT dbo.testrethrow (id) VALUES(1);
    -- Force error 2627, Violation of PRIMARY KEY constraint to be raised.
    INSERT dbo.testrethrow (id) VALUES(1);
END try
BEGIN catch
    PRINT 'In catch block.';

    THROW;
END catch;
```





Faculdade
IMPACTA
TECNOLOGIA



Obrigado !

Gustavo Maia
Gustavo.Maia@FaculdadeImpacta.com.br

