# Instituto Federal

**Campus Goiânia**

## Bacharelado em Sistemas de Informação

# Banco de Dados II

**Prof. Dory Gonzaga Rodrigues**

MySQL®

ORACLE®

PostgreSQL
the world's most advanced open source database

## Agenda

### Motivação 1

**UPDATE** contas **SET** saldo = saldo – 1000.00 **WHERE** codigo = 1;

**UPDATE** contas **SET** saldo = saldo + 1000.00 **WHERE** codigo = 2;

Neste exemplo de transferências entre contas bancárias:

1) montante de 1000.00 deve ser debitado da conta 1 e creditado na conta 2.

2) Se a ligação com a Base de Dados for perdida ou o Banco de Dados travar, após o primeiro UPDATE, o dinheiro retirado da primeira conta desaparece, isto é, não chega a ser creditado na conta 2.

3) É conveniente que as duas atualizações sejam executadas ou então nenhuma.

Agenda

## Motivação 2

| codigo | nome | saldo | data |
|---|---|---|---|
| 1 | Dory | 5000 | 1976-03-31 |
| 2 | Caroline | 100.3 | 1980-05-20 |
| 3 | Ana | 50.1 | 2013-12-01 |
| 4 | Isabelle | 800 | 2010-06-01 |

```
UPDATE contas
    SET saldo = saldo – 1000.00
 WHERE codigo = 1;
```

```
UPDATE contas
    SET saldo = saldo + 1000.00
 WHERE codigo = 2;
```

```
UPDATE contas
    SET saldo = saldo – 2000.00
 WHERE codigo = 1;
```

```
UPDATE contas
    SET saldo = saldo + 2000.00
 WHERE codigo = 3;
```

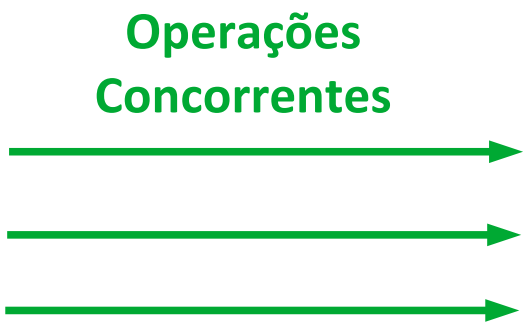
Neste exemplo de transferências simultâneas sobre a mesma conta bancária:

1) A conta 1 poderia ser debitada em apenas R$ 2000,00 em vez de R$ 3000,00!
2) As transferências não ocorrem **isoladas** uma da outra.

Intuitivamente, seria desejável que o efeito lógico do conjunto de comandos fosse equivalente a uma **sequência de transferências** !

## Agenda

### Motivação 2




| codigo | nome | saldo | data |
|---|---|---|---|
| 1 | Dory | 5000 | 1976-03-31 |
| 2 | Caroline | 100.3 | 1980-05-20 |
| 3 | Ana | 50.1 | 2013-12-01 |
| 4 | Isabelle | 800 | 2010-06-01 |

Neste exemplo de transferências simultâneas sobre a mesma conta bancária, temos uma situação comum e normal de acesso concorrente a uma base de dados

Um SGBD deve estar preparado para lidar com **concorrência** sobre uma base de dados, isto é, várias sessões simultâneas e independentes **Sessão 1** | … | **Sessão n** a executar operações na BD.

## Transactions

**O que são Transações ?**

O modelo relacional descreve a unidade lógica de processamento de dados como uma transação.

Cada transação pode ser definida como um conjunto de declarações (comandos / operações) feita em sequência, que podem atualizar, deletar, inserir e retornar dados.

Uma transação ocorre com sucesso apenas se todas as operações que compõe a transação forem bem sucedidas.

Se uma operação dentro da transação falhar, o efeito das demais operações parcialmente concluídas pode ser desfeito.

## Transactions

### Propriedades Fundamentais

As propriedades **ACID** são fundamentais para o processamento de transações em banco de dados relacionais:

**A**tomicidade, **C**onsistência, **I**solamento e **D**urabilidade

Cada uma destas propriedades visa garantir que uma transação no banco de dados ocorra sem erro. E caso venha a ocorrer um erro, nenhum dado será perdido ou perderá sua validade/integridade.

Vejamos o que significa cada uma dessas propriedades.

## Transactions

**Propriedades Fundamentais para a garantia dos Dados - ACID**

**Atomicidade**:

A atomicidade é uma propriedade que garante que cada transação seja tratada como uma entidade única, a qual deve ser executada por completo ou cancelada por completo caso ocorra uma falha.

Desta forma, todas as operações da transação devem ser executadas com sucesso para que a transação tenha sucesso. Caso haja falha em qualquer operação da transação, o banco de dados será retornado ao estado anterior ao início da transação.

Chamamos a esse retorno de estado de ***Rollback*** (“transação desfeita”).

Caso a transação tenha sucesso, o BD é alterado de forma permanente, em um processo denominado ***Commit*** (“efetivação”)

## Transactions

**Propriedades Fundamentais para a garantia dos Dados - ACID**

**Consistência**:

Já a propriedade da consistência visa assegurar que uma transação somente leve o banco de dados de um estado válido a outro, respeitando as restrições de integridade e mantendo a estabilidade do banco.

Os dados que são gravados devem sempre ser válidos, de acordo com regras definidas, e isso inclui qualquer operação, como triggers, constraints, function (stored procedure), ou outras que determinem a validade dos dados inseridos.

Desta forma, é evitada a corrupção do banco de dados que pode ser causada por uma transação ilegal.

Exemplo: tentar registrar a venda de um produto que não existe na tabela “produtos”, fará com que a transação não seja concluída.

## Transactions

**Propriedades Fundamentais para a garantia dos Dados - ACID**

**Isolamento**:

Os bancos de dados normalmente são acessados de forma concorrente, ou seja, de forma que várias tabelas sejam lidas ou alteradas por vários usuários simultaneamente.

A propriedade do isolamento controla a execução concorrente/simultâneas, mantendo a integridade do banco de dados, como se as transações estivessem sendo executadas em sequência.

Exemplo: Duas transações concorrentes tentando efetuar o registro da compra de um único produto em estoque. O primeiro a finalizar a compra fará com que a transação do outro seja interrompida e o comando *rollback* seja executado.

## Transactions

**Propriedades Fundamentais para a garantia dos Dados - ACID**

**Durabilidade**:

A durabilidade garante que uma transação, uma vez executada (efetivada), permanecerá neste estado mesmo que haja um problema grave no sistema, como travamento de sistema ou falta de energia elétrica no servidor.

Para isso, as transações finalizadas são gravadas em dispositivos de memória permanente (não-volátil), como discos rígidos, de modo que os dados estejam sempre disponíveis, mesmo que a instância do BD seja reiniciada.

## Transactions

### SQL – Transação

Para definir uma transação explicitamente, usamos:

- a declaração **BEGIN** para iniciar a transação.
- a declaração **END ou COMMIT** para concluir a transação.

**Sintaxe:**

| | |
|---|---|
| **BEGIN** | - -Inicia a transação |
| - - Comandos | |
| **SAVEPOINT** | - - ponto de salvamento |
| - - Comandos | |
| **COMMIT** | - - indica o término da transação/persistir dados |
| **ROLBACK TO** | - - volta a transação para o ponto de salvamento |
| **ROLBACK** | - - indica o término da transação/efeitos desfeitos |
| **END** | - - mesma função do COMMIT |

## Transactions

### SQL – Transação

**Exemplo:**

```
BEGIN;
    DROP TABLE IF EXISTS contas;
    CREATE TABLE contas(
        codigo INT PRIMARY KEY,
        nome VARCHAR(40),
        saldo NUMERIC,
        data DATE
    );
    INSERT INTO contas VALUES (1, 'Dory', 5000, '31-03-1976');
    INSERT INTO contas VALUES (2, 'Caroline', 100.30, '20-05-1980');
    INSERT INTO contas VALUES (3, 'Ana', 50.10, '1-12-2013');
    INSERT INTO contas VALUES (4, 'Isabelle', 800, '8-06-2010');
COMMIT;
```

## Transactions

### SQL – Transação

**Exemplo continuação:**

Vamos realizar a transação onde iremos transferir R$ 1.000,00 da conta do Dory (código=1) para a conta da Caroline (código=2).

```
BEGIN;
    UPDATE contas SET saldo = saldo – 1000.00 WHERE codigo = 1;
    SAVEPOINT debito;
    UPDATE contas SET saldo = saldo + 1000.00 WHERE codigo = 3;
    -- Conforme consta no enunciado o correto seria creditar na conta 2 !
    -- O comando ROLLBACK TO volta a situação do banco para o status salvo no
       SAVEPOINT “debito” e continua a execução das instruções contidas após
      o comando  ROLLBACK TO.
    ROLLBACK TO debito;
    UPDATE contas SET saldo = saldo + 1000.00 WHERE codigo = 2;
COMMIT;
```

# Transactions

## Controle de Concorrência nas transações do PostgreSQL

- qualquer comando SQL é tratado como sendo executado dentro de uma transação.

- para cada transação, o PostgreSQL faz um snapshot (instantâneo) dos dados, como estes eram no exato momento em que a transação foi iniciada, desprezando as mudanças ocorridas depois disso.

- trabalha com SAVEPOINTS (pontos de salvamento), guardando as informações e as operações existentes antes do SAVEPOINT e que foram executadas com sucesso.

- Com o comando ROLLBACK TO a execução da transação irá voltar para o status salvo até o momento do SAVEPOINT indicado e, depois, continuará a execução das instruções seguintes normalmente.

- mantém a consistência dos dados utilizando o modelo multiversão MVCC (Multiversion Concurrency Control), que permite que leitura não bloqueie escrita e nem escrita bloqueie leitura.

## Transactions

### Níveis de Isolamento de transação

O padrão SQL define quatro níveis de isolamento de transação.

O mais rigoroso é o ***Serializable***, onde qualquer execução simultânea de um conjunto de transações "Serializáveis" tem a garantia de produzir o mesmo efeito que executá-las uma por vez em alguma ordem.

Os outros três níveis são definidos em termos de fenômenos, resultantes da interação entre transações concorrentes, que não devem ocorrer em cada nível.

No uso do padrão ***Serializable***, nenhum dos fenômenos proibidos ocorre, já que na teoria os comandos poderiam ser executados em sequência.

Os fenômenos que podem ocorrer em transações simultâneas/concorrentes:

Leitura suja
Leitura não repetível
Leitura fantasma
Anomalia de serialização

## Transactions

**Níveis de Isolamento de transação**

**Os fenômenos que podem ocorrer em transações simultâneas/concorrentes:**

***Leitura suja***

Uma transação lê dados gravados por uma transação simultânea não confirmada.

***Leitura não repetível***

Uma transação relê os dados lidos anteriormente e descobre que os dados foram modificados por outra transação (que foi confirmada desde a leitura inicial).

## Transactions

**Níveis de Isolamento de transação**

**Os fenômenos que podem ocorrer em transações simultâneas/concorrentes:**

***Leitura fantasma***

Uma transação executa novamente uma consulta retornando um conjunto de linhas que satisfazem uma condição de pesquisa e descobre que o conjunto de linhas que satisfazem a condição mudou devido a outra transação confirmada recentemente.

***Anomalia de serialização***

O resultado da confirmação bem-sucedida de um grupo de transações é inconsistente com todas as ordens possíveis de execução dessas transações, uma de cada vez.

# Transactions

## Níveis de Isolamento x Fenômenos

| **Nível de isolamento** | **Dirty Read** (Leitura Suja) | **Nonrepeatable Read** (Leitura Não repetível) | **Phantom Read** (Leitura Fantasma) | **Serialization Anomaly** (Anomalia de Serialização) |
|---|---|---|---|---|
| Read uncommitted | Permitido, mas não ocorre no PostgreSQL | Pode ocorrer | Pode ocorrer | Pode ocorrer |
| Read committed | Não ocorre | Pode ocorrer | Pode ocorrer | Pode ocorrer |
| Repeatable read | Não ocorre | Não ocorre | Permitido, mas não ocorre no PostgreSQL | Pode ocorrer |
| Serializable | Não ocorre | Não ocorre | Não ocorre | Não ocorre |

Obs: os níveis de isolamento definem quais fenômenos não devem acontecer, não quais fenômenos devem acontecer.

No PostgreSQL, você pode solicitar qualquer um dos quatro níveis de isolamento de transação, mas internamente apenas três níveis de isolamento distintos são implementados. O modo Read Uncommitted se comporta como Read Committed.

# Transactions

## Níveis de Isolamento de transação

**READ COMMITTED** (Leitura Confirmada)

Este é o padrão no PostgreSQL

Uma instrução só lê as linhas confirmadas antes do instante em que a leitura iniciar.

| Leitura Suja | Leitura Não repetível | Leitura Fantasma | Anomalia de Serialização |
|---|---|---|---|
| Nunca ocorre | Pode ocorrer | Pode ocorrer | Pode ocorrer |

**Conexão 1**

```
Prompt de Comando - psql  -d transacao -U postgres

C:\Users\doryg>psql -d transacao -U postgres
Password for user postgres:
psql (12.1)
WARNING: Console code page (850) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

transacao=# begin;
BEGIN
transacao=# insert into contas values (5,'Pedro', 0.00, '2000-01-01');
INSERT 0 1
transacao=#
```

Esta transação ainda não foi confirmada !

**Conexão 2**

```
Prompt de Comando - psql  -d transacao -U postgres

C:\Users\doryg>psql -d transacao -U postgres
Password for user postgres:
psql (12.1)
WARNING: Console code page (850) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

transacao=# select * from contas order by codigo;
 codigo |   nome   |  saldo  |    data
--------+----------+---------+------------
      1 | Dorival  | 4000.00 | 1976-03-31
      2 | Carol    | 2000.30 | 1980-05-20
      3 | Ana      | 5000.10 | 2013-12-01
      4 | Isabelle |     800 | 2010-06-08
(4 rows)

transacao=#
```

Esta transação não leu a alteração não confirmada da conexão 1 !

## Transactions

### Níveis de Isolamento de transação

**READ COMMITTED** (Leitura Confirmada)

No entanto:

- SELECT vê os efeitos das atualizações anteriores executadas em sua própria transação, mesmo que ainda não tenham sido confirmadas.

- Dois comandos SELECT sucessivos podem ver dados diferentes, mesmo que estejam dentro de uma única transação, se outras transações confirmarem alterações após o início do primeiro SELECT e antes do início da execução do segundo SELECT.

Mais detalhes sobre este tipo de isolamento:
https://www.postgresql.org/docs/current/transaction-iso.html

# Transactions

## Níveis de Isolamento de transação

**REPEATABLE READ** (Leitura Repetível)

Todas as instruções da transação vê apenas os dados confirmados antes do início da transação (“repetível”). Ou seja, dados não confirmados ou alterações confirmadas durante a execução da transação por transações simultâneas, não serão vistos.

| Leitura Suja | Leitura Não repetível | Leitura Fantasma | Anomalia de Serialização |
|---|---|---|---|
| Nunca ocorre | Nunca ocorre | Permitido (ñ PostgreSQL) | Pode ocorrer |

Conexã

Prompt de Comando - psql -d transacao -U postgres

```
transacao=# begin transaction isolation level repeatable read;
BEGIN
transacao=# select * from contas where codigo = 1;
 codigo |  nome   |  saldo  |    data
--------+---------+---------+------------
      1 | Dorival | 4000.00 | 1976-03-31
(1 row)


transacao=# select * from contas where codigo = 1;
 codigo |  nome   |  saldo  |    data
--------+---------+---------+------------
      1 | Dorival | 4000.00 | 1976-03-31
(1 row)


transacao=#
```

Esta transação ainda não foi confirmada !

Prompt de Comando - psql -d transacao -U postgres

```
transacao=# begin;
BEGIN
transacao=# select * from contas where codigo = 1;
 codigo |  nome   |  saldo  |    data
--------+---------+---------+------------
      1 | Dorival | 4000.00 | 1976-03-31
(1 row)


transacao=# update contas set saldo = 10000.00 where codigo = 1;
UPDATE 1
transacao=# select * from contas where codigo = 1;
 codigo |  nome   |  saldo   |    data
--------+---------+----------+------------
      1 | Dorival | 10000.00 | 1976-03-31
(1 row)


transacao=# commit;
COMMIT
transacao=#
```

Esta transação foi confirmada após o início da transação da conexão 1!

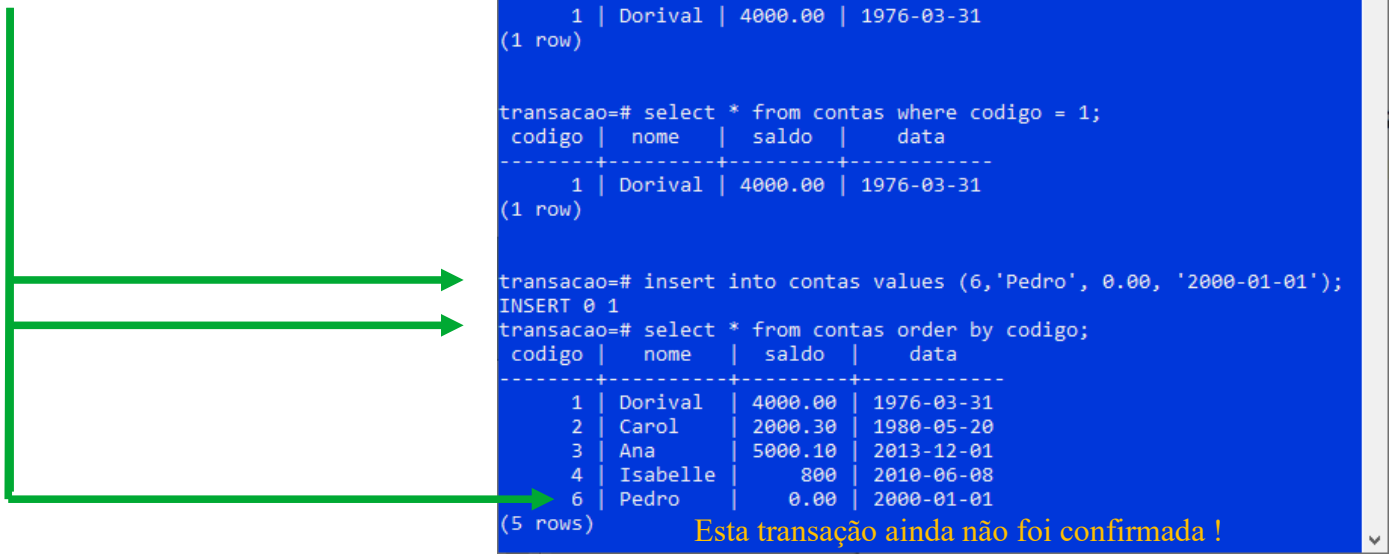
## Transactions

### Níveis de Isolamento de transação

**REPEATABLE READ** (Leitura Repetível)

No entanto:

- a consulta vê os efeitos das atualizações anteriores executadas em sua própria transação, mesmo que ainda não tenham sido confirmadas.

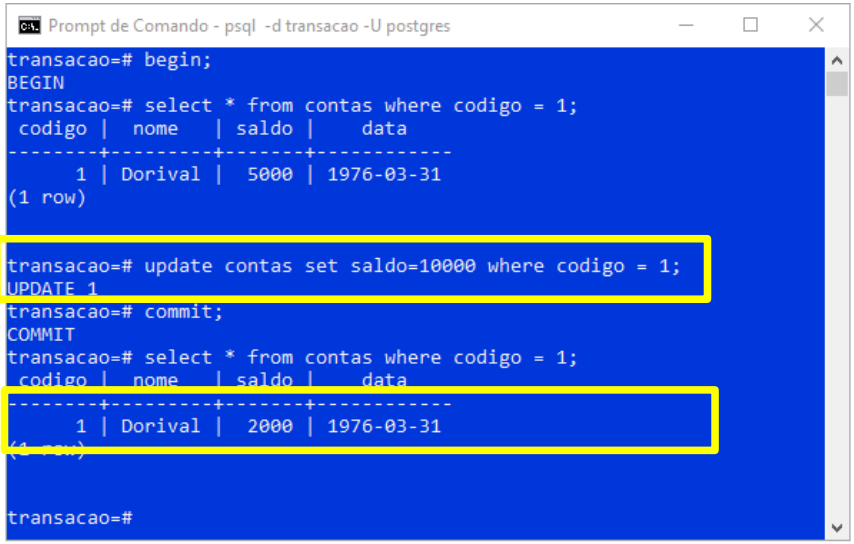


# Transactions

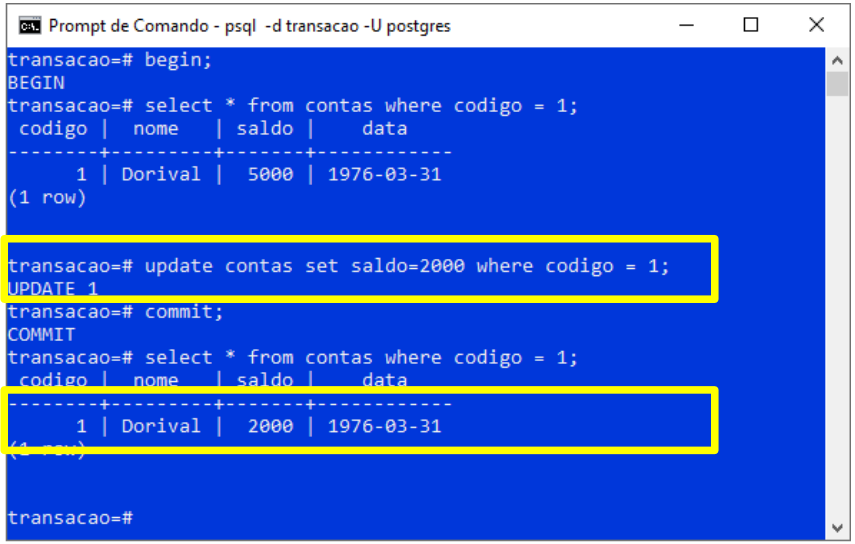
## Níveis de Isolamento de transação

Os níveis de isolamento anteriores permitem que ocorra o problema de atualizações perdidas. Ou seja, as atualizações realizadas em uma transação podem ser "perdidas" ou substituídas por outra transação que seja executada simultaneamente.

**Conexão 1**




**Conexão 2**




Aqui, o UPDATE da conexão 2 é bloqueado, porque o PostgreSQL coloca um bloqueio para evitar outra atualização até que a primeira transação seja concluída. No entanto, realizando o COMMIT na conexão 1, o UPDATE da conexão 2 é executado. Havendo a confirmação das transações, a alteração da primeira transação é perdida, porque a segunda "sobrescreveu" a linha. Se esse tipo de comportamento não for aceitável, você pode definir o nível de isolamento para SERIALIZÁVEL.

Transactions

## Níveis de Isolamento de transação

**SERIALIZABLE** (Serializável)

Este é o tipo de isolamento de transação mais restrito. Ele emula a execução da transação serial para todas as transações confirmadas, como se as transações tivessem sido executadas uma após a outra, em série, em vez de simultaneamente.

| Leitura Suja | Leitura Não repetível | Leitura Fantasma | Anomalia de Serialização |
|---|---|---|---|
| Nunca ocorre | Nunca ocorre | Nunca ocorre | Nunca ocorre |

Embora o nível de isolamento de transação Serializável do PostgreSQL só permita que transações simultâneas sejam confirmadas se puder provar que há uma ordem de execução serial que produziria o mesmo efeito, ele nem sempre evita o surgimento de erros que não ocorreriam na execução serial verdadeira.

Se um padrão de leituras e gravações entre transações serializáveis concorrentes criaria uma situação que não poderia ter ocorrido para qualquer execução serial (uma de cada vez) dessas transações, uma delas será revertida com um ***serialization_failure error***

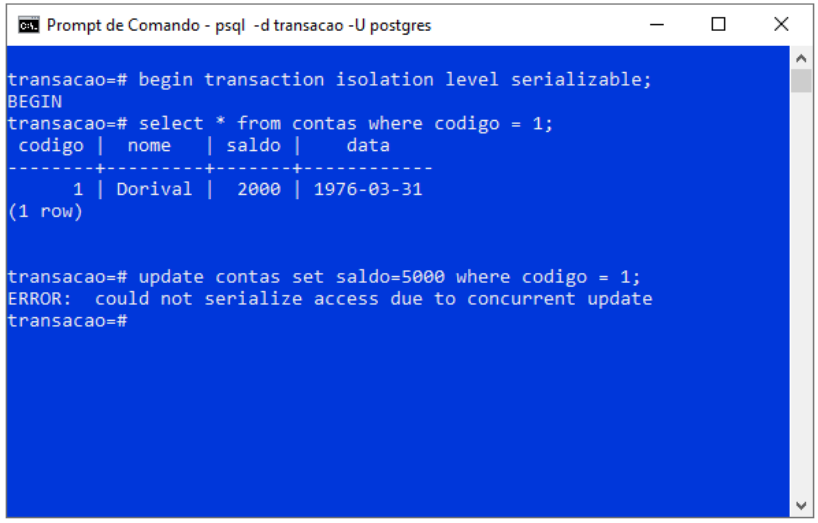
Transactions

## Níveis de Isolamento de transação

**SERIALIZABLE** (Serializável)

No nível SERIALIZABLE, o mesmo exemplo feito anteriormente (atualização perdida), agora o COMMIT da segunda transação falha.

As ações da segunda transação foram baseadas em fatos que foram invalidados no momento em que ela estava prestes a ser confirmada.

## Transactions

### Níveis de Isolamento de transação

Para definir o nível de uma transação, usamos:

**Sintaxe:**

**SET TRANSACTION ISOLATION LEVEL** *nível modo*

**SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION LEVEL** *nível modo*

Onde:

**Nível {** SERIALIZABLE | REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED **}**

**Modo {** READ WRITE | READ ONLY **}**

## Transactions

### Modo de acesso da transação

**READ WRITE | READ ONLY**

O modo de acesso à transação determina se a transação é de leitura / gravação ou somente leitura.

Quando uma transação é somente leitura, os seguintes comandos SQL não são permitidos:

- INSERT, UPDATE, DELETE, e COPY FROM se a tabela a ser escrita não é uma tabela temporária;

- CREATE, ALTER e DROP comandos;

- COMMENT, GRANT, REVOKE, TRUNCATE; e EXPLAIN ANALYZE e EXECUTE se o comando que eles executariam está entre os listados.