Transações

BDII

Guilherme Dal Bianco

Introdução

Atualizar o saldo de uma conta

```
UPDATE Contas SET Saldo = Saldo - 50 WHERE NoConta = 123
```

Transação com mais operações: transfere 50 reais da conta 123 para conta 456

```
begin;
UPDATE Contas SET Saldo = Saldo - 50 WHERE NoConta = 123;
UPDATE Contas SET Saldo = Saldo + 50 WHERE NoConta = 456;
end;
```

Introdução

Atualizar o saldo de uma conta

```
UPDATE Contas SET Saldo = Saldo - 50 WHERE NoConta = 1
```

Transação com mais operações: transfere 50 reais da conta 001 para conta 002

```
UPDATE Contas SET Saldo = Saldo - 50 WHERE NoConta = 001;
UPDATE Contas SET Saldo = Saldo + 50 WHERE NoConta =002;
```

Introdução

Simplificando as operações:

```
Read(A)
A=A-50
Write(A)
Read(B)
B=B+50
Write(B)
```

Onde *A* e *B* representam os saldos das duas contas contas correntes conhecidas

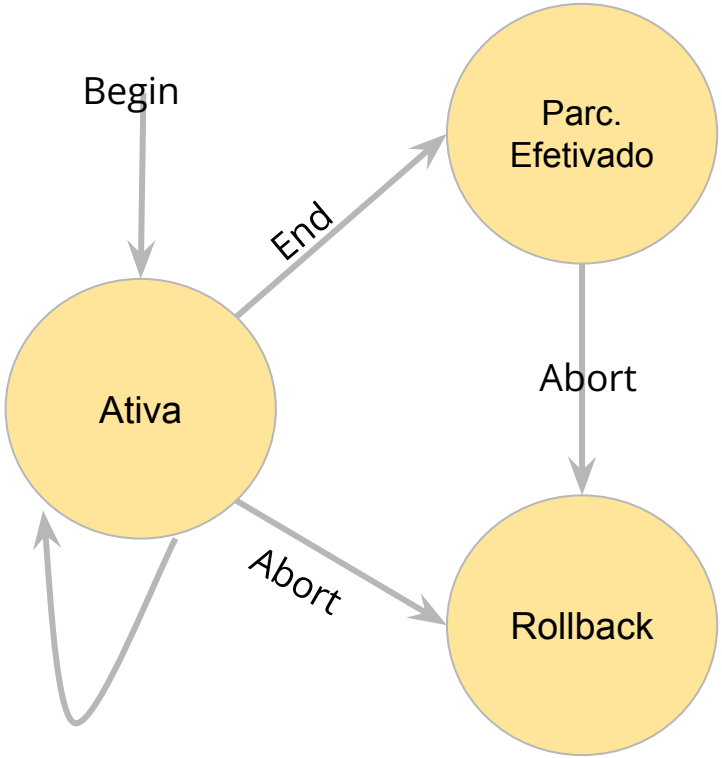
Introdução

Transação

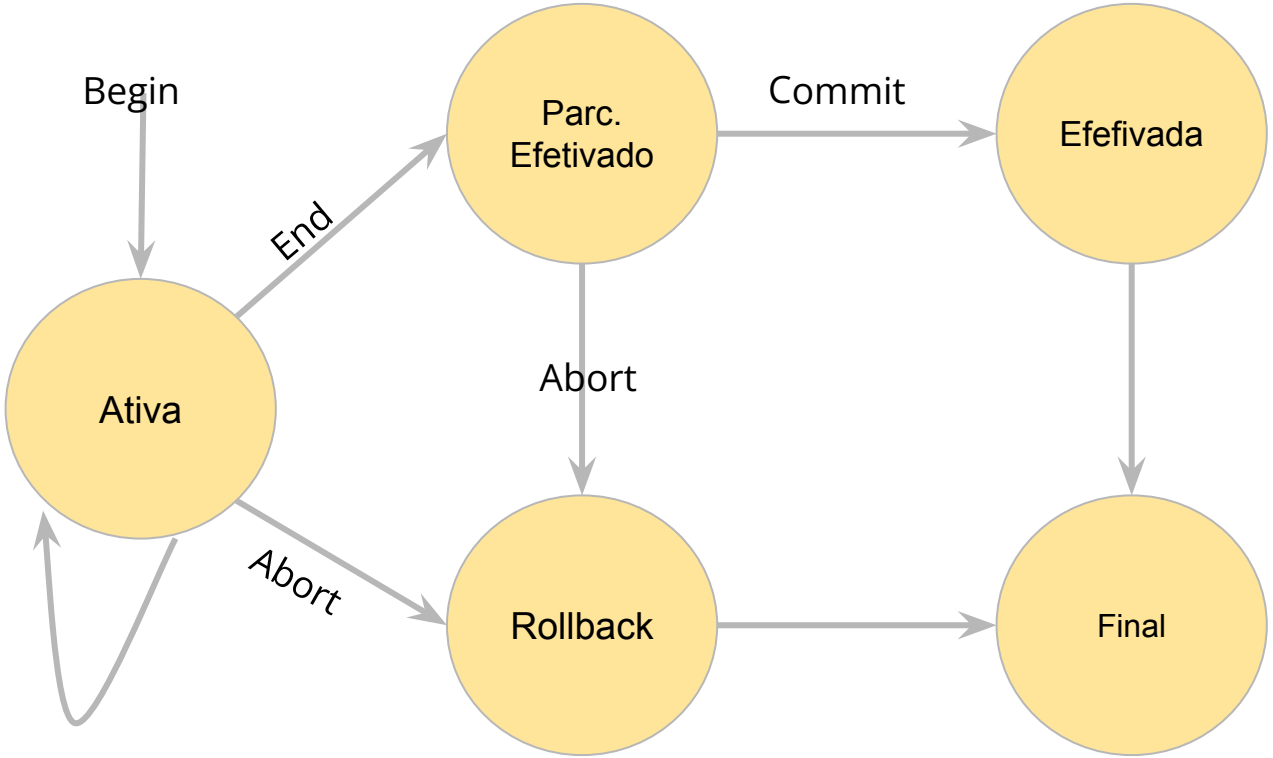
Fim normal = Commit

Fim anormal= abort (rollback)

Transaçōes- Estados



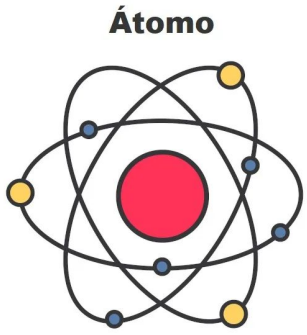
Transaçōes- Estados



Transações-ACID

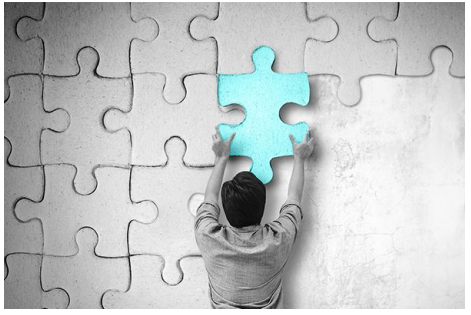
- Atomicidade
- Consistência
- Isolamento
- Durabilidade

Transações-ACID



Atomicidade: todas as operações de uma transação devem ser efetivadas; ou, na ocorrência de uma falha, nada deve ser efetivado

Transações-ACID



Atomicidade: todas as operações de uma transação devem ser efetivadas; ou, na ocorrência de uma falha, nada deve ser efetivado

Consistência: transações preservam a consistência da base

Transações-ACID

Atomicidade: todas as operações de uma transação devem ser efetivadas; ou, na ocorrência de uma falha, nada deve ser efetivado

Consistência: transações preservam a consistência da base

Isolamento: a maneira como várias transações em paralelo interagem (o que pode ser lido e o que pode ser escrito por cada uma) deve ser bem definida

Transações-ACID

Atomicidade: todas as operações de uma transação devem ser efetivadas; ou, na ocorrência de uma falha, nada deve ser efetivado

Consistência: transações preservam a consistência da base

Isolamento: a maneira como várias transações em paralelo interagem (o que pode ser lido e o que pode ser escrito por cada uma) deve ser bem definida

Durabilidade: uma vez consolidada (committed) a transação, suas as alterações permanecem no banco até que outras transações aconteçam

Exemplo Transaçōes

```
create table trans (id int, nome varchar(50), saldo float);

begin;
    insert into trans values (1,'joao', 1000);
    insert into trans values (2,'maria, 2000);
commit;
```

Exemplo Transações A

```
--Transação A
begin;
   update trans set saldo = 1200 where id = 1;
commit;

--Transação B
begin;
   select * from trans where id = 1;
commit;
```

Exemplo Transações B

create table dados (id serial, nome varchar(50));

#seleciona o número da transação atual

1. select txid_current();

#acha o local dos dados

2. select pg_relation_filepath('dados');

#mostra o dataset

3. sudo hexdump -C /var/lib/postgresql/14/main/<End tabela>

#insere dados

4. insert into dados(nome) values (‘joao’)

Exemplo de transação em Python

```
import psycopg2

conn = None
try:
    conn = psycopg2.connect(database="teste", user = "guilherme", password = "     ", host = "127.0.0.1", port =
"5432")
    cur = conn.cursor()
    cur.execute(“insert into employee values (1,’maria’)”)
    cur.execute(“insert into employee values (2,’jose’)”)
    conn.commit()
    cur.close()
except psycopg2.DatabaseError as error:
    print(error)
finally:
    if conn is not None:
        conn.close()
```

Exemplo de transação em Java

```
...
Connection connection = DriverManager.getConnection("127.0.0.1", "guilherme", "     ");
try {
    connection.setAutoCommit(false);
    PreparedStatement firstStatement = connection .prepareStatement("insert into employee values
(1,'maria')");
    firstStatement.executeUpdate();
    PreparedStatement secondStatement = connection .prepareStatement("insert into employee values
(2,'jose')");
    secondStatement.executeUpdate();
    connection.commit();
} catch (Exception e) {
    connection.rollback();
}
...
```

slido

Uma atualização, dentro de uma transação, tem seus dados persistidos em disco quando:

ⓘ Start presenting to display the poll results on this slide.

slido

Uma transação, altera do estado "parcialmente efetivada" para "efetivada" quando

ⓘ Start presenting to display the poll results on this slide.

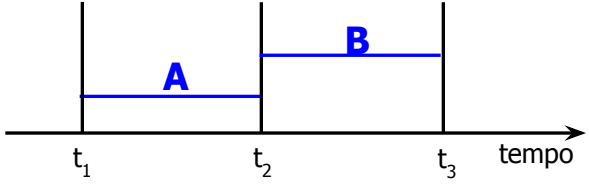
slido

Quando o software falha, a informação deve estar persistida. Qual propriedade ACID que garante isso?

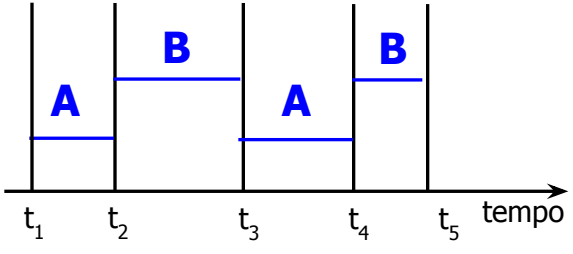
ⓘ Start presenting to display the poll results on this slide.

Concorrência

Execução Serial (sequencial):



Execução Intercalada:



Controle de Concorrência

Execução Serial (sequencial): diversas transações executadas em sequência

- deixa a base de dados em estado correto e consistente

Execução Intercalada: comandos de diversas transações são intercalados

- pode levar a inconsistências

Controle de Concorrência

Execução Intercalada

- Toda execução serial é consistente

- Mas uma execução intercalada só é consistente se for igual ao resultado de uma execução em sequência (em ordem conhecida)
  - esta execução é dita **serializável**

Problemas de Execução Intercalada

Ocorrência de anomalias

1. leitura inválida
2. leitura não repetível
3. leitura fantasma

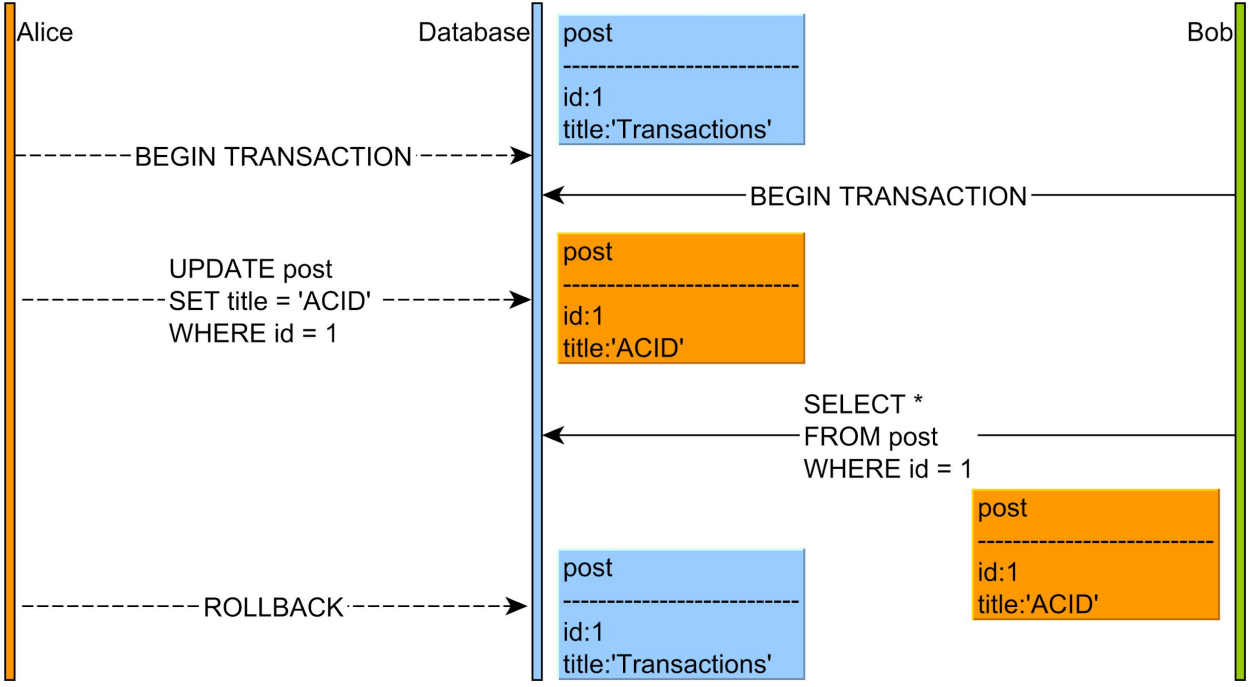
Problemas de Execução Intercalada

1) Leitura inválida (*Dirty Read*):

transação T2 lê um dado modificado por uma transação T1 que ainda não terminou;

Problemas de Execução Intercalada

Leitura inválida (*Dirty Read*):



Fonte: https://vladmihalcea.com/dirty-read/

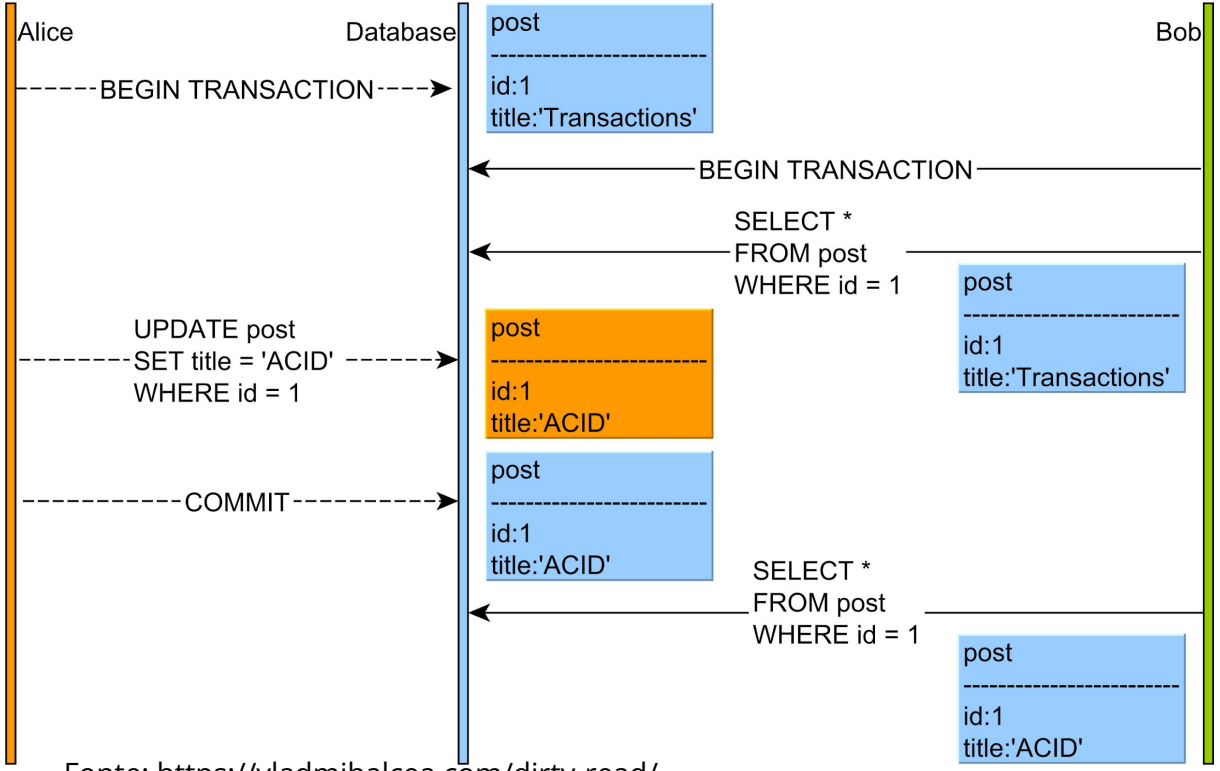
Problemas de Execução Intercalada

2) Leitura não repetível (*Nonrepeatable Read*):

- Ocorre quando uma transação lê duas vezes uma variável com valores diferentes;

Problemas de Execução Intercalada

Leitura não repetível (*Nonrepeatable Read*):



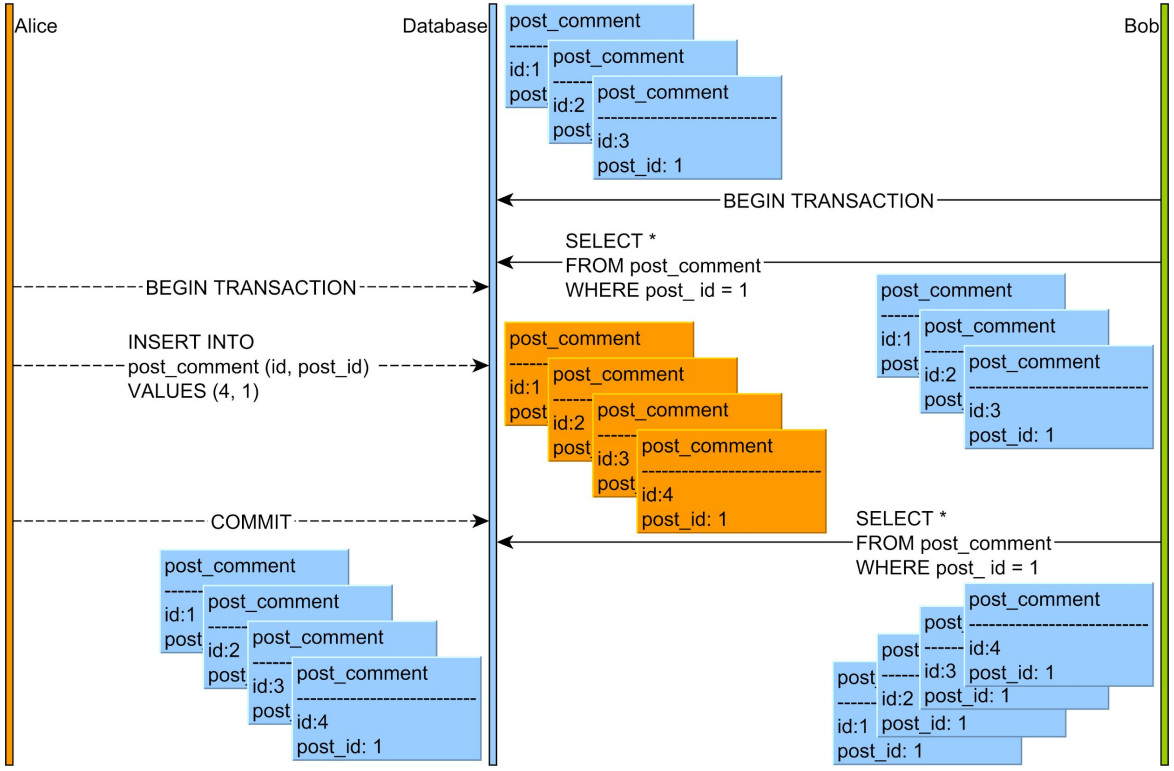
Fonte: https://vladmihalcea.com/dirty-read/

Problemas de Execução Intercalada

Leitura fantasma (*Phantom Read*):

- Ocorre quando uma consulta transação quando executada mais de uma vez resulta em diferente linhas;

Problemas de Execução Intercalada



Fonte: https://vladmihalcea.com/dirty-read/

Problemas de Execução Intercalada

Repeatable read vs Phantom read

→ *Repeatable read:* lê valores diferentes de **um** mesmo dado que ainda está lá, mas foi alterado

→ *Phanton read*: lê **conjuntos** de dados diferentes, sendo que um dos conjuntos possui dados que não existem no(s) outro(s) conjunto(s) – fantasmas.

Transações Postgres

SET TRANSACTION ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | **READ COMMITTED** | READ UNCOMMITTED }

READ WRITE | READ ONLY

Níveis de isolamento

	1) Leitura inválida	2) Leitura não repetível	3) Leitura fantasma
Read uncommitted	Sim	Sim	Sim
Read committed	Não	Sim	Sim
Repeatable read	Não	Não	Sim
Serializable	Não	Não	Não

slido

Considere os seguintes comportamentos em transações de banco de dados:Dirty Read | Nonrepeatable Read | Phantom Read.O(s) comportamento(s) possível(eis) no nível de isolamento READ COMMITTED do padrão SQL-92 é(são):

ⓘ Start presenting to display the poll results on this slide.

slido

Assinale a alternativa que contém o nível de isolamento de transação que impede a ocorrência de leituras sujas (dirty reads), leituras fuzzy (nonrepeatable reads), e leituras fantasma (phantom reads).

ⓘ Start presenting to display the poll results on this slide.

Controle de concorrência no Postgres

- Delete não **apaga** de fato o dado
- Update não atualiza o dado de fato

Multiversion Concurrency Control- MVCC

- Baseado no conceito que **conflitos são infrequentes**
- Deixa executar as transações concorrentemente
- Conflito -> Abortar
- Cada transação enxerga sobre uma “cópia” dos dados e não “lê” alterações de outras transações não comitadas

Multiversion Concurrency Control- MVCC

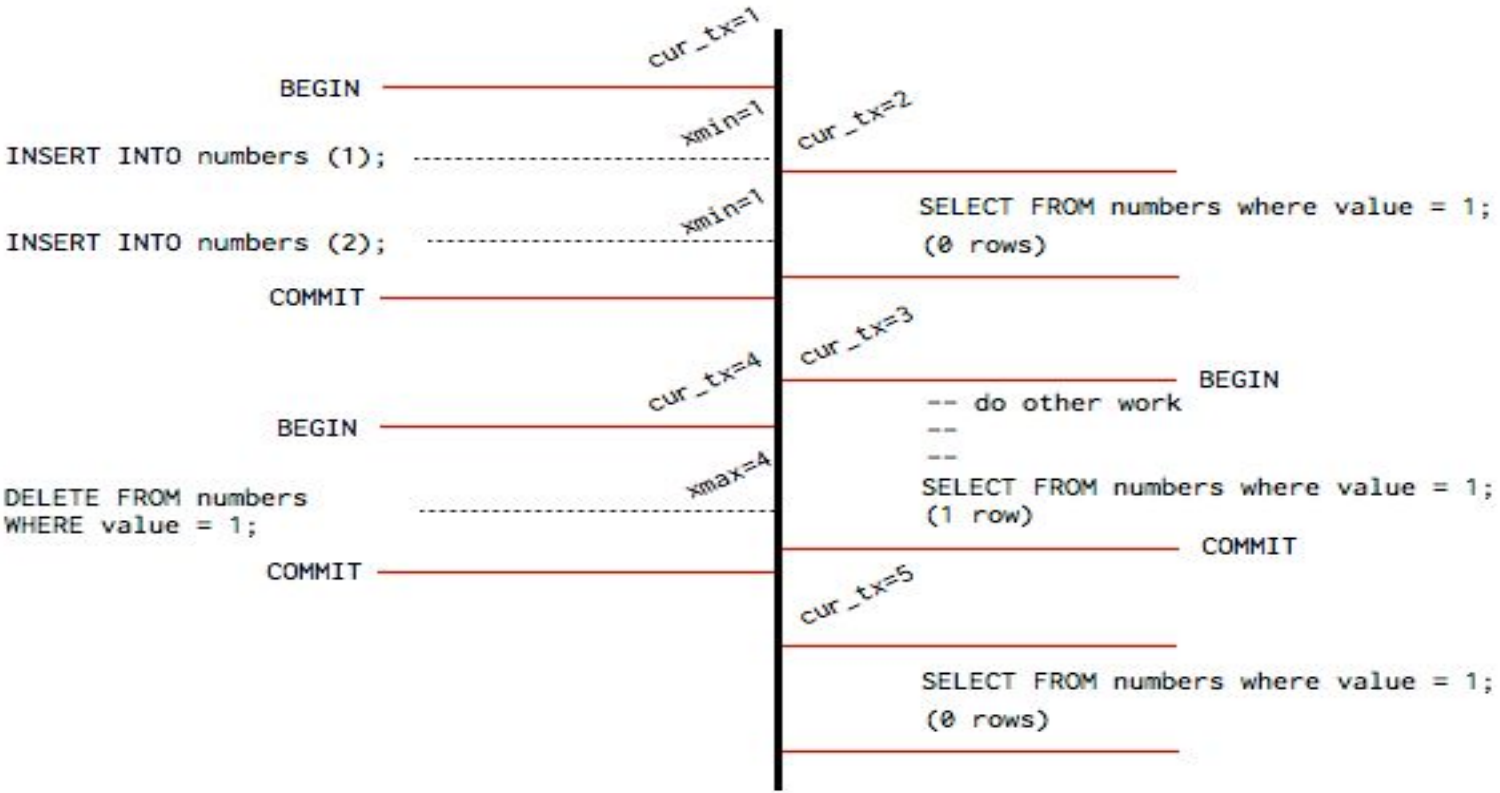
Variáveis:

- **Xmin**: armazena a transação que fez a última alteração

- **Xmax**: reporta se o registro está em processo de remoção

- **Ctid**: mostra o deslocamento e a página que a tupla se encontra;

Multiversion Concurrency Control- MVCC



Exemplo 1

A) Crie a tabela teste (id integer, value char(1000))

B) Adicione 2 linhas com os seguintes valores:

```
Insert into teste values (1,'a'),(2,'b'),(3,'c'),(4,'d');
```

C) Verifique o tamanho da tabela: `\dt+ teste`

D) Verifique a posição de cada registro com o comando

```
SELECT ctid,* from teste;
```

E) Atualize todas as linhas para o id receber +1

```
Update teste set id=id+1;
```

F) Verifique o tamanho da tabela novamente e a posição de cada registro. **Descreva o que aconteceu.**

Atividade 1

A. Abra um terminal no Postgres:

sudo -u postgres psql postgres

B. Crie a seguinte tabela:

cliente(numero serial primary key, cpf int , nome varchar(50))

C. Crie as seguintes operações e descreva o que acontece em cada caso

a. Uma transação com “commit” (faça 3 inserts)
b. Uma transação com “rollback” (faça 3 inserts)
c. Uma transação tentando acessar dados de outra transação ainda não “comitados”. (abra dois terminais com o postgres). O que acontece?

Atividade 2

A) Abra uma transação A:

   Rode SELECT txid_current()

   Rode SELECT xmin, xmax, ctid, * FROM teste

B) Apague uma tupla em outra transação (B) (outro terminal)

   SELECT txid_current()

C) Rode SELECT xmin, xmax, ctid, * FROM teste
D) Rode novamente o SELECT xmin, xmax, ctid, * FROM teste. **Explique o que aconteceu?**
E) Faça o mesmo com o comando update, atividade A-D.. **Explique novamente o que aconteceu?**

Atividade 3

A)Crie a tabela teste (id integer primary key, value char(2000))

B)Adicione 4 tuplas

C) Verifique o tamanho da tabela

D) Insira uma tupla com id =10 em uma transação A (sem comitar)

E) Insira uma tupla com id =10 em uma transação B (sem comitar)

Explique o que aconteceu com as transações já que os dados são operados em snapshots diferentes?

Atividade 4

Qual é a saída da linha 9?

Linha	Terminal 1	Terminal 2
1	create table txn (id int);	
2	begin;	
3	insert into txn values (1);	
4	insert into txn values (2);	
5		insert into txn values (3);
6		insert into txn values (4);
7	insert into txn values (4);	
8	rollback;	
9	select **xmin, xmax, ctid, *** from txn;	

Extra

Construa uma aplicação em uma linguagem de programação capaz de executar 1000 inserções usando uma **transação** na tabela. A aplicação deve ser capaz também de listar a tabela após as inserções. Trate a exceção no caso de uma inserção de uma chave já existente (**rollback**).

Os dados podem ser gerados usando a ferramenta:
https://www.mockaroo.com/

Atividade Extra

Usar o dataset reddit_vm para responder às seguintes perguntas:

1- Qual o tempo de execução na inserção de 10000 tuplas com o autocommit True e False? Explique o que aconteceu. OBS: rodar 5 vezes e fazer a média e desvio padrão dos tempos de execução

2- Abra dois terminais e execute, ao mesmo tempo, o código da questão anterior com o autocommit False. Além disso, setar o nível de isolamento SERIALIZABLE. Reportem o tempo (5 execuções com o desvio padrão). Explique o que acontece na prática neste caso?

Atividade prática

- Vamos voltar ao SQL-DOJO;
- Execute o código data.py com o N=10000 e com as transações auto-commit=True. Qual o tempo de execução total?

  (time python data.py)

- Execute o código data.py com o N=10000 e com as transações auto-commit=true. Qual o tempo de execução total?

Vacuum Postgres

Vacuum -> executa a limpeza das tuplas mortas sem reconstruir a página. Não reduz o espaço ocupado pela tabela.

Vacuum Full -> executa a limpeza reconstruindo a tabela em uma nova página (bloqueio). Reduz o espaço ocupado.

Código Python

```
import psycopg2

import time

from psycopg2 import extensions, connect

print ("Opened database successfully")

conn=None

try:

    conn = psycopg2.connect(database="teste", user = "guilherme", password = "1761791", host = "127.0.0.1", port = "5432")

    conn.autocommit = False

    cur = conn.cursor()

    starttime = time.time()

    for i in range(10000):

        cur.execute("insert into employee values (1,'joao',100000)")

        if(i%5000==0):
```