Curso Superior em Sistemas para Internet

INSTITUTO FEDERAL
Paraíba
Campus João Pessoa

Banco de Dados II

Transação, concorrência e recuperação de falhas

PostgreSQL

Profa. Damires Souza

damires@ifpb.edu.br

O que é uma transação??

Um exemplo

- **Transferência *poupança -> conta corrente***, valor de R$ **800.00**

  1. Deduzir valor da poupança
  2. Acrescentar valor na conta corrente
  3. Registrar no diário de transações

Em SQL

TRANSAÇÃO

```
UPDATE contaPoupança
        SET saldo = saldo – 800
        WHERE numconta = 2345;
UPDATE contaCorrente
        SET saldo = saldo + 800
        WHERE numconta = 1456;
INSERT INTO registroTransação
        VALUES(234,current_date,2345,1456,800);
COMMIT;
```

Transação

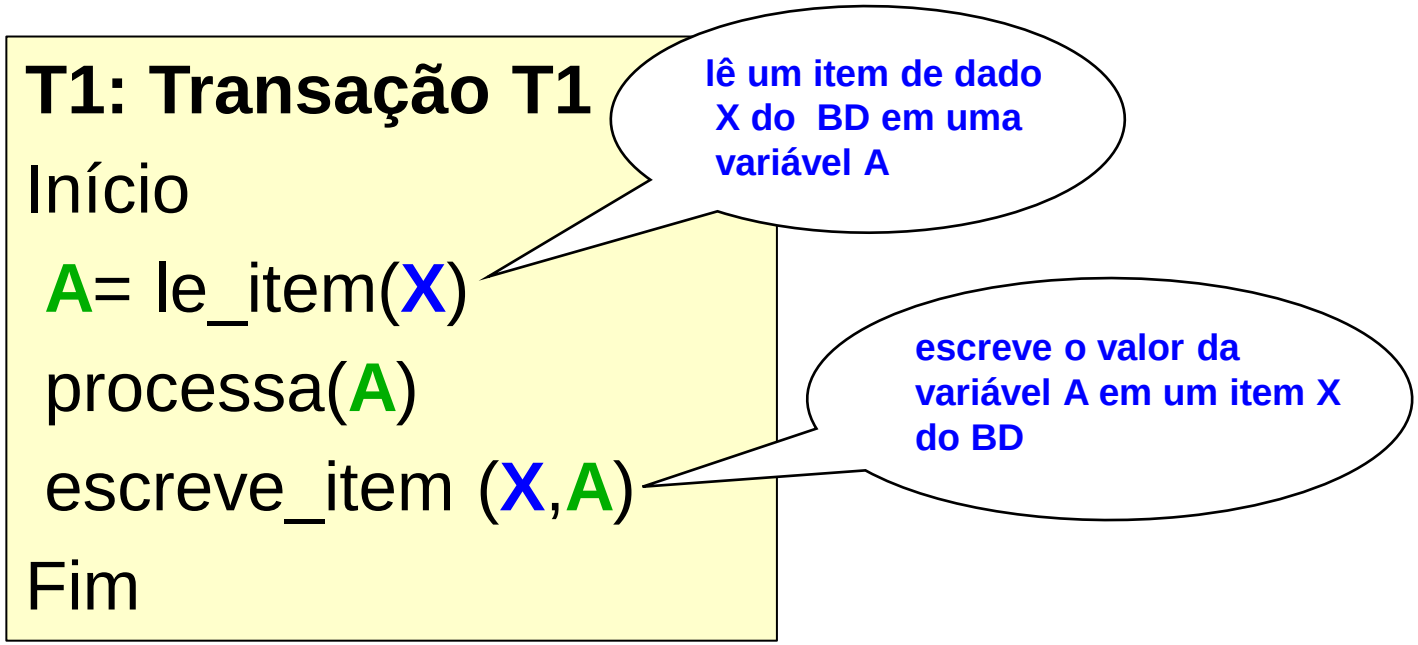
Uma **transação** é um "**programa em execução**" que forma uma **unidade lógica de processamento** no banco de dados

- Contém **uma ou mais operações** de acesso
  - **inclusão, exclusão, alteração ou recuperação**
- Seus limites podem ser determinados em **SQL**
  - Podem estar embutidas em **nível de aplicação,** no **SGBD** ou especificadas interativamente **via SQL**

=>Necessário prover mecanismos de gerenciamento de transações

Estrutura da Transação

- O BD pode ser visto como uma **coleção de itens de dados**
  - Item de dados no **Modelo Relacional** = **campo**, **registro**, **tabela**



O que poderia acontecer se duas transações ocorressem ao mesmo tempo sobre os mesmos itens??

T1	T2
Read(X)	
X=X-N	
	Read(X)
	X=X+M
write(X)	
read (Y)	
	write(X)
Y=Y+N	
write (Y)	

Estado inicial: X = 2 N = 1 M = 4

[T1] X = 2
[T1] X = 2 – 1 = 1
[T2] X = 1
[T2] X = 1 + 4 = 5
[T1] X = 5
...

=>Necessário prover mecanismos de gerenciamento de concorrência

E se ocorrer uma falha no SGBD??

T1	T2
Read(X)	
X=X-N	Read(X)
	X=X+M
write(X)	write(X)
read (Y)	
Y=Y+N	
write (Y)	

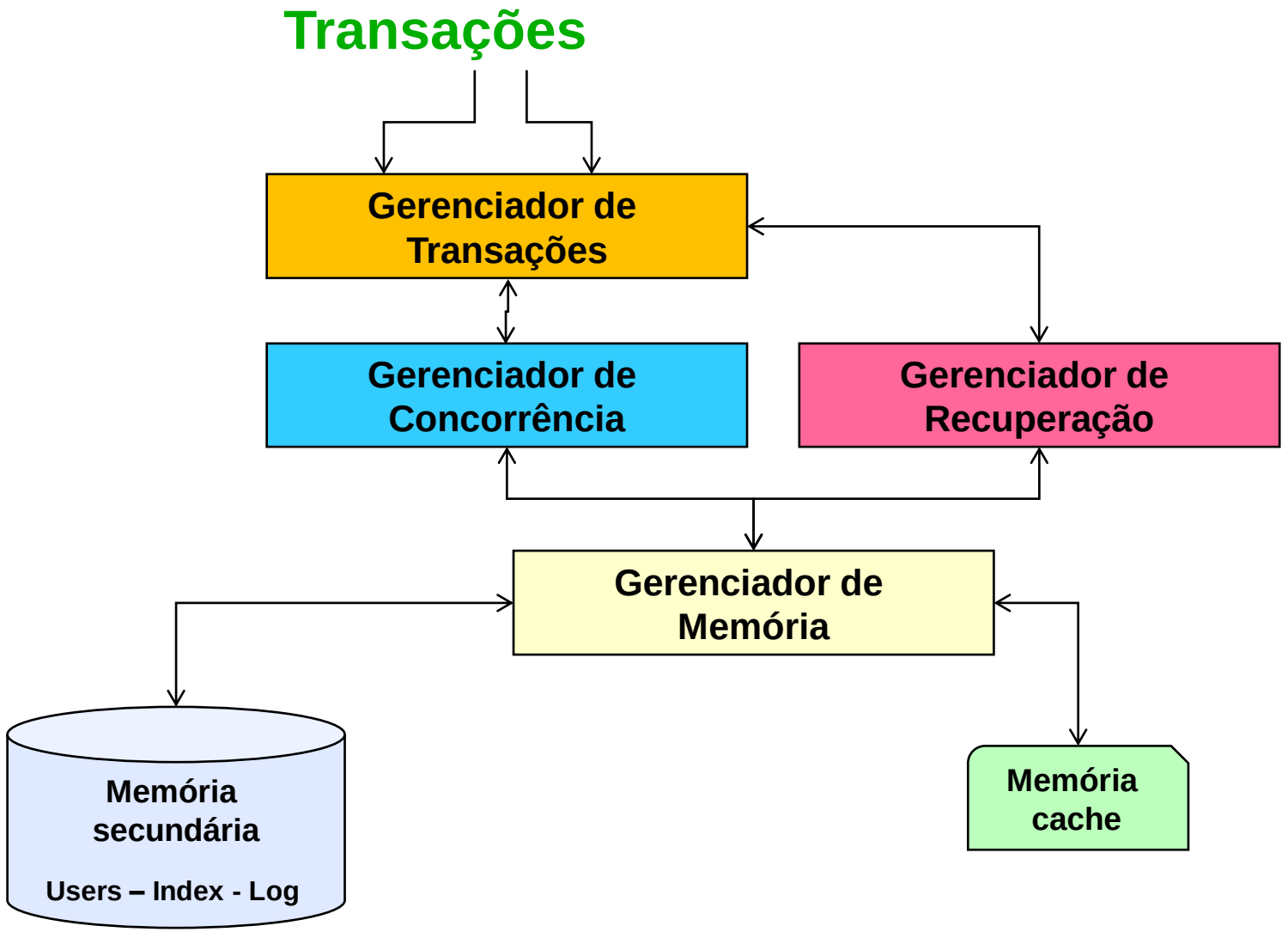
Estado inicial: X = 2 N = 1 M = 4

[T1] X = 2
[T1] X = 2 – 1 = 1
[T2] X = 1
[T2] X = 1 + 4 = 5

????

=>Necessário prover mecanismos de recuperação de falhas

Arquitetura Geral de um SGBD Relacional



Propriedades das transações no Modelo Relacional

Propriedades ACID

Atomicidade

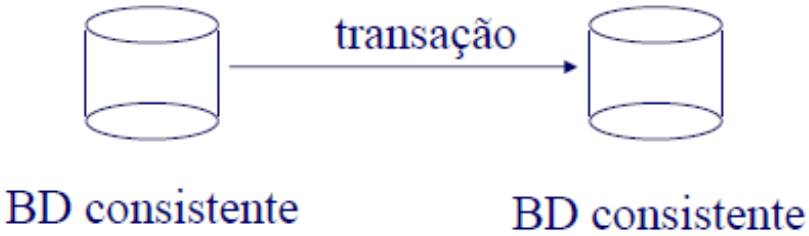
Consistência

Isolamento

Durabilidade

Propriedades ACID

- **Atomicidade**: ou **todas** as operações da transação são refletidas corretamente no banco de dados ou nenhuma será
  - Princípio do “Tudo ou Nada”
- **Consistência**: a execução de uma transação deve preservar a consistência do BD
  - Uma transação sempre conduz o BD de um estado consistente para outro estado também consistente
  - Restrições de integridade definidas devem ser preservadas



Propriedades ACID

- **Isolamento**: cada transação **não toma conhecimento** da execução de outras transações concorrentes no sistema

  "como se ela executasse de forma isolada"

  - Não deve sofrer interferências de outras transações executando concorrentemente

- **Durabilidade**: depois da transação completar com sucesso, as mudanças que ela produz no BD persistem até mesmo se houverem falhas no sistema.

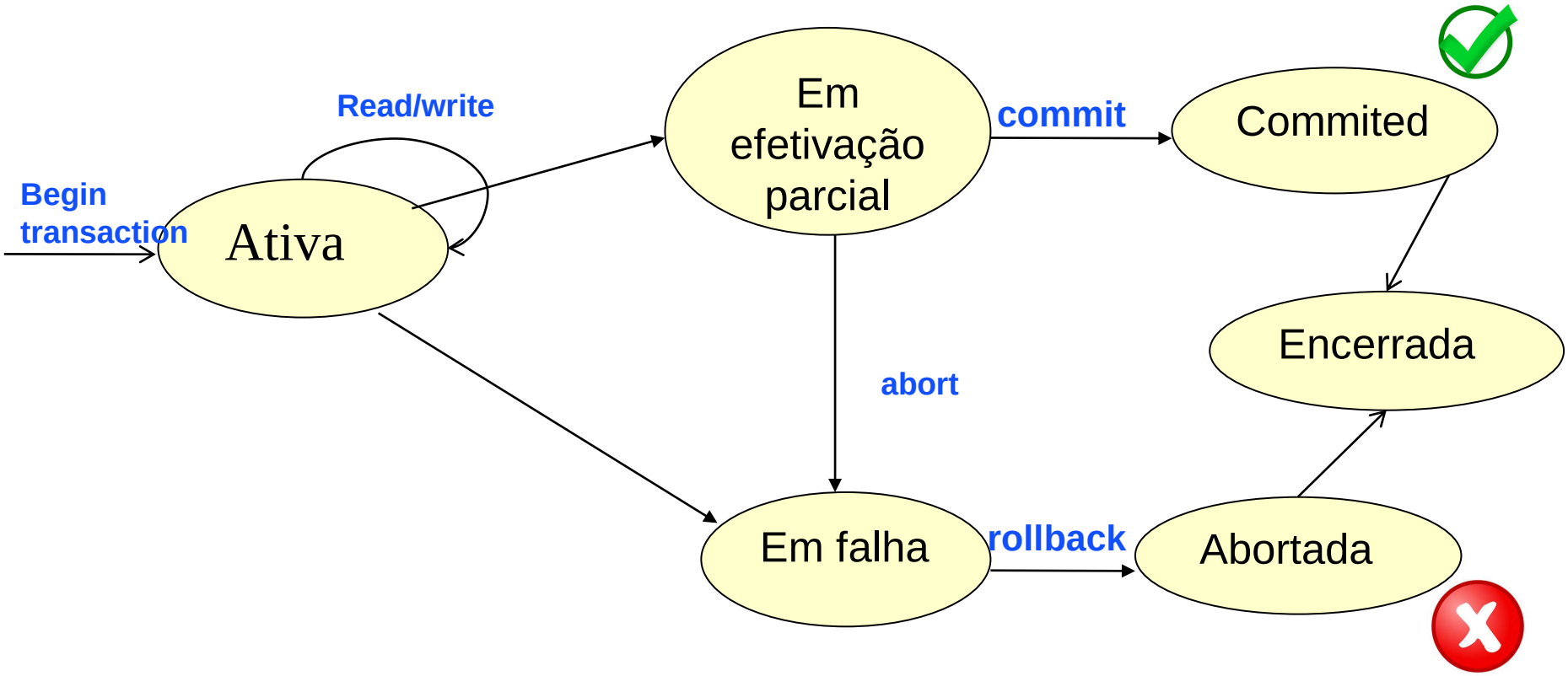
  - Nenhuma falha posterior ocorrida no BD deve perder essas modificações

Transação

- Ponto essencial:
  - Várias operações são englobadas em uma **"única" de tudo ou nada**
- Uma transação pode ser

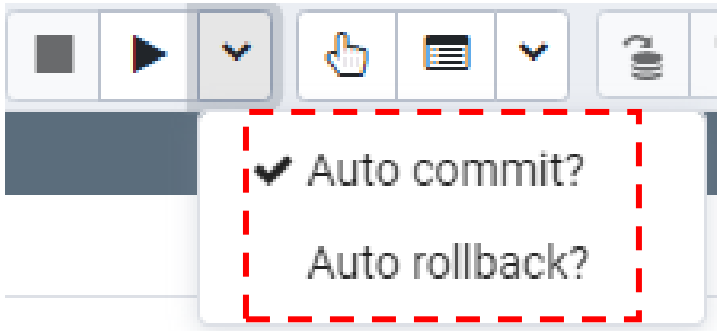
bem sucedida ou **mal sucedida**

Estados de uma Transação



- Uma transação inicia com uma instrução BEGIN [TRANSACTION] (ou por um comando SQL) e termina com um COMMIT ou ROLLBACK
- Se não atualizar o BD, é chamada de transação read-only

No Postgres/pgAdmin...



BD: PEDIDOS

-- desabilite o autocommit;

Vamos a exemplos...

BD: PEDIDOS

Commit - Exemplo

```
select * from vendedor;
insert into vendedor values(default,'Melissa Gonçalves', '28/07/1988',
   3300.80,'B');
insert into vendedor values(default,'Debora Maciel', '20/04/1990',
   3300.80,'B');
insert into vendedor values(default,'Alicia Silva', '28/06/1998',
   2300.80,'C');
select * from vendedor;
commit;
```

Commit, *commit transaction* ou *commit work* : finaliza uma transação com sucesso

- torna **persistentes** as alterações realizadas pela transação

BD: PEDIDOS

Rollback - Exemplo

```
select * from vendedor;
insert into vendedor values(default,'Moacir Ribeiro', '27/07/1988',
   3300.80,'B');
insert into vendedor values(default,'Daniel Moura', '20/03/1990',
   3300.80,'B');
insert into vendedor values(default,'Alvaro Soares', '28/04/1998',
   2300.80,'C');
select * from vendedor;
rollback;
select * from vendedor;
```

Rollback ou *rollback work* ou *rollback transaction*: finaliza uma transação sem sucesso

- Desfaz os efeitos das operações da transação

Ou seja...

```
select * from vendedor;
insert into vendedor
values(default,'Moacir Ribeiro',
'27/07/1988', 3300.80,'B');
insert into vendedor
values(default,'Daniel Moura',
'20/03/1990', 3300.80,'B');
insert into vendedor
values(default,'Alvaro Soares',
'28/04/1998', 2300.80,'C');
select * from vendedor;
ROLLBACK;
select * from vendedor;
```

- **Um SGBD relacional deve garantir:**
  - **ou a transação é executada por completo ou nenhuma parte dela é**
  - Se alguma falha ocorrer impedindo a transação de chegar até o fim, então nenhum dos passos intermediários deve afetar o BD

Um pouco mais

- Um SGBD relacional deve garantir:
  - Os estados dos passos intermediários da transação **não devem ser visíveis** para outras transações concorrentes
  - **Cada usuário** deve receber visões **consistentes** dos dados, incluindo modificações feitas por ele e por transações de outros

Como???

- Uma transação **T** alcança seu ponto de efetivação (**commit point**) quando todas as suas operações que acessam o BD estão sendo executadas **com sucesso,** e o efeito será gravado de modo permanente
- Para isso, o sistema mantém um **log**
  - Para controlar todas as operações da transação que afetem valores do BD
  - O log é normalmente mantido em disco

Sistema de LOG

- Tipos de entradas no log:

  [start_transaction, T]

  [write_item,T,X,valor antigo, valor novo]

  [read_item,T,X]

  [commit,T]

  [abort,T]

- Antes da transação alcançar seu **ponto de commit**, qualquer parte do log que ainda não tenha sido gravada no disco deve ser
  - Para auxiliar na recuperação de falhas

Vamos a exemplos...

Usuários no Postgres

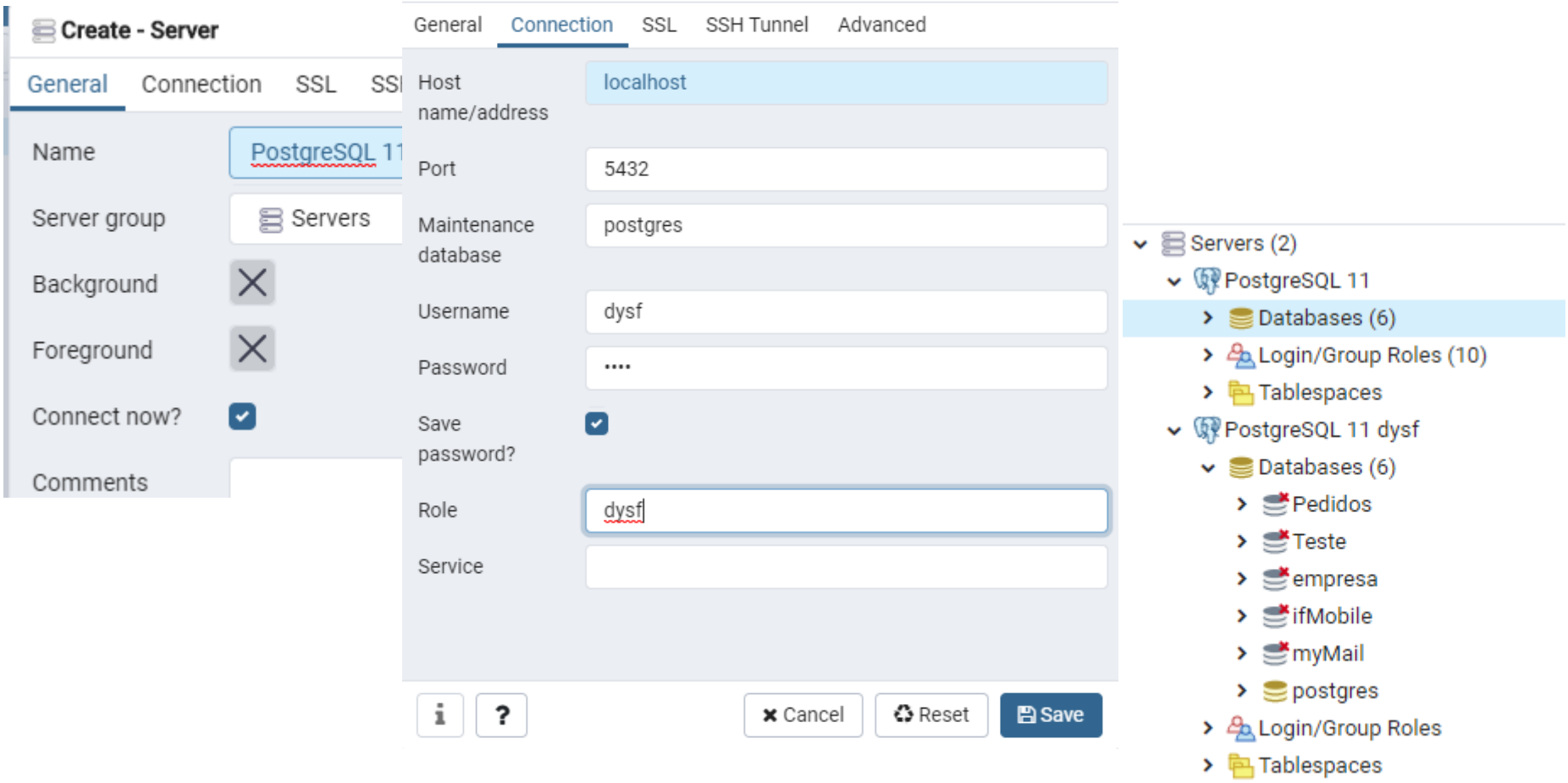
Lembrando...

- Superusuário **postgres:**
  - Conexão inicial, normalmente criado na instalação

**** Vamos criar um role (usuário) com suas iniciais**
(caso não tenha)

```
CREATE ROLE dysf LOGIN
  PASSWORD ‘bd2'
  SUPERUSER CREATEDB CREATEROLE ;
```

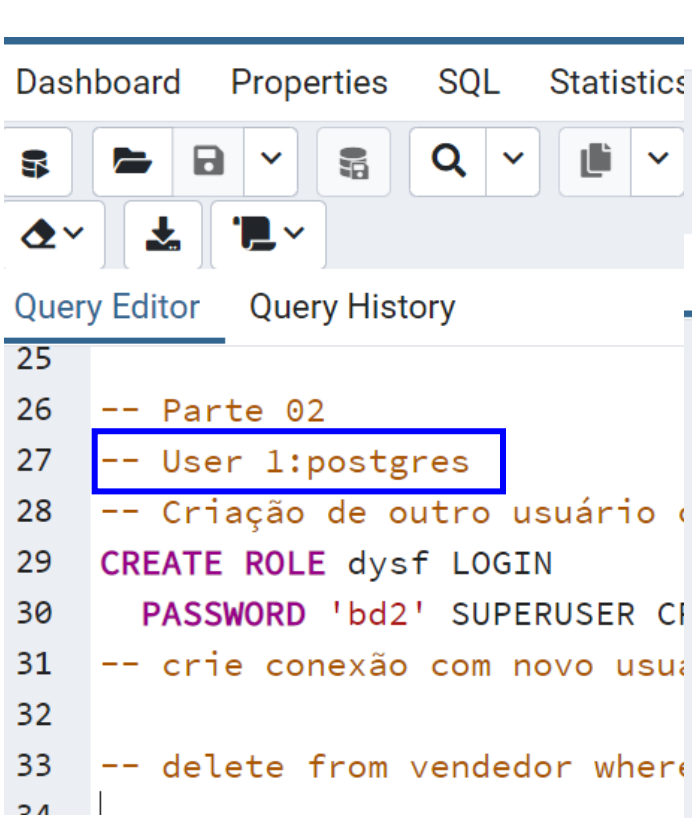
Adicione uma nova conexão para “dysf”

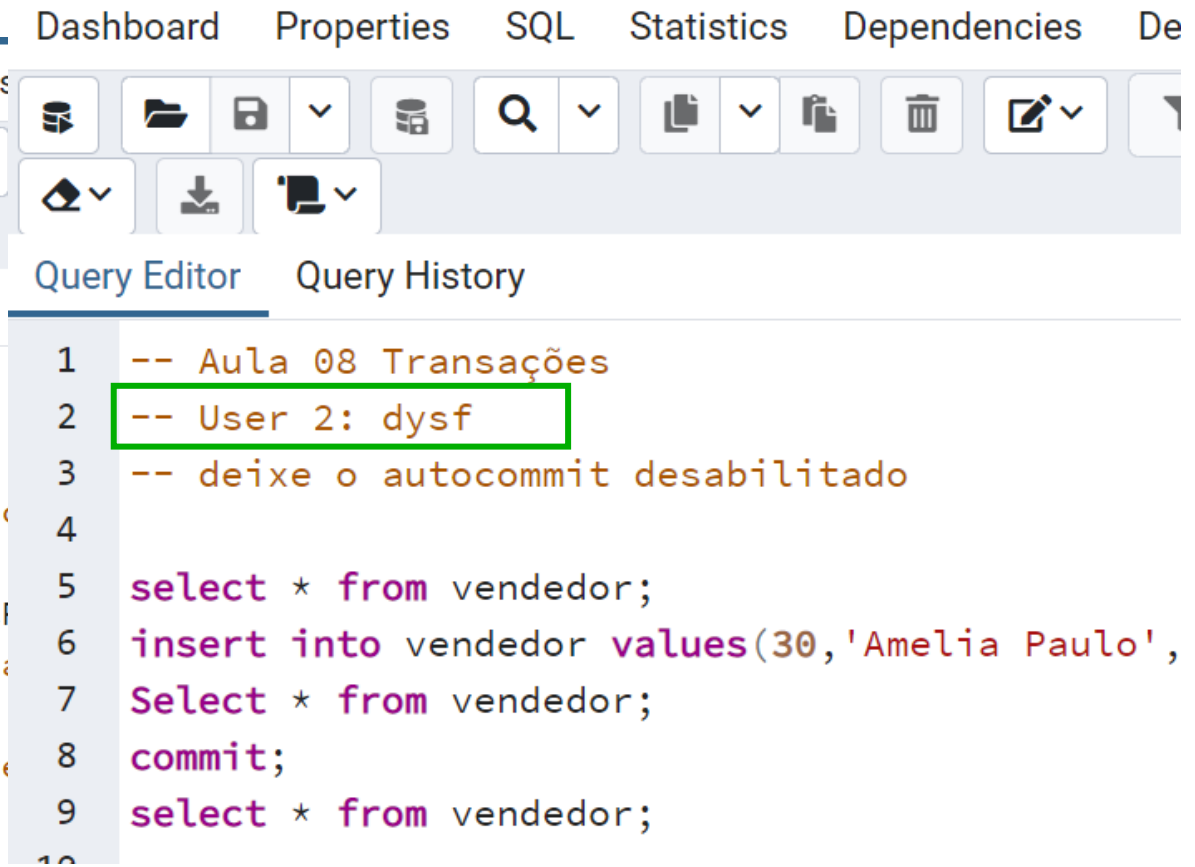


Deixe duas sessões

Deixe duas query tools:

- **01 com postgres**
- **01 com seu *usuário***





Transação: testando a concorrência 1

Dois usuários: postgres e dysf

BD: Pedidos

[postgres] Grant all on vendedor to dysf;

[dysf] Select * from vendedor;

[postgres] Select * from vendedor;

[postgres] insert into vendedor values(12,'Clovis Paulo', '12/01/1994', 5300.80,'A');

[dysf] insert into vendedor values(20,'Amelia Paulo', '12/03/1996', 5300.80,'A');

[postgres] Select * from vendedor;

[dysf] Select * from vendedor;

Lembrar de desabilitar o autocommit!!

[postgres] commit;

[dysf] commit;

[postgres] select * from vendedor;

[dysf] select * from vendedor;

O que aconteceu???

Exemplo: testando a concorrência 2

Dois usuários: postgres e dysf

O que acontece se os dois inserirem **o mesmo valor de chave primária** na **mesma tabela**??

[postgres] insert into vendedor values(40,’Augusto Paulo', '12/03/1996', 5300.80,'A');

X

[dysf] insert into vendedor values(40,’Augusto Paulo', '12/03/1996', 5300.80,'A');

Lembrar de desabilitar o autocommit!!

Verifique: o que aconteceu???

BD: Pedidos

Estratégias

- Baseadas em protocolos (conjuntos de regras)
  - Buscam assegurar a **serialização** de planos de execução de transações
    - Protocolos baseados em **bloqueio**
    - Protocolos baseados em ***timestamp***
    - **Multiversão** dos dados - MVCC

Protocolo baseado em Bloqueio

- **Regras** seguidas por todas as transações para **solicitação e liberação de bloqueios**
  - Princípio: impedir que múltiplas transações acessem os itens ao mesmo tempo

Se a transação **A** precisa de acesso a um recurso/item **N**

Então **A** vai bloquea-lo;

Após o uso, o bloqueio será liberado.

Se a transação **B** quer acessar o recurso/item **N** enquanto **A** está usando

Então **B** deve esperar.

Bloqueio Compartilhado/Exclusivo

- Três estados de bloqueio:
  - **read_locked** – bloqueio compartilhado (**S**hared)
    - permite que outras operações leiam o item
  - **write_locked** – bloqueio exclusivo (e**X**clusive)
    - uma única transação controla exclusivamente o bloqueio no item
  - **unlocked** - item desbloqueado
    - **lock_X(X)**, **lock_S(X)** e **unlock(X)**
      - onde **X** é um item de dado

Ideia geral

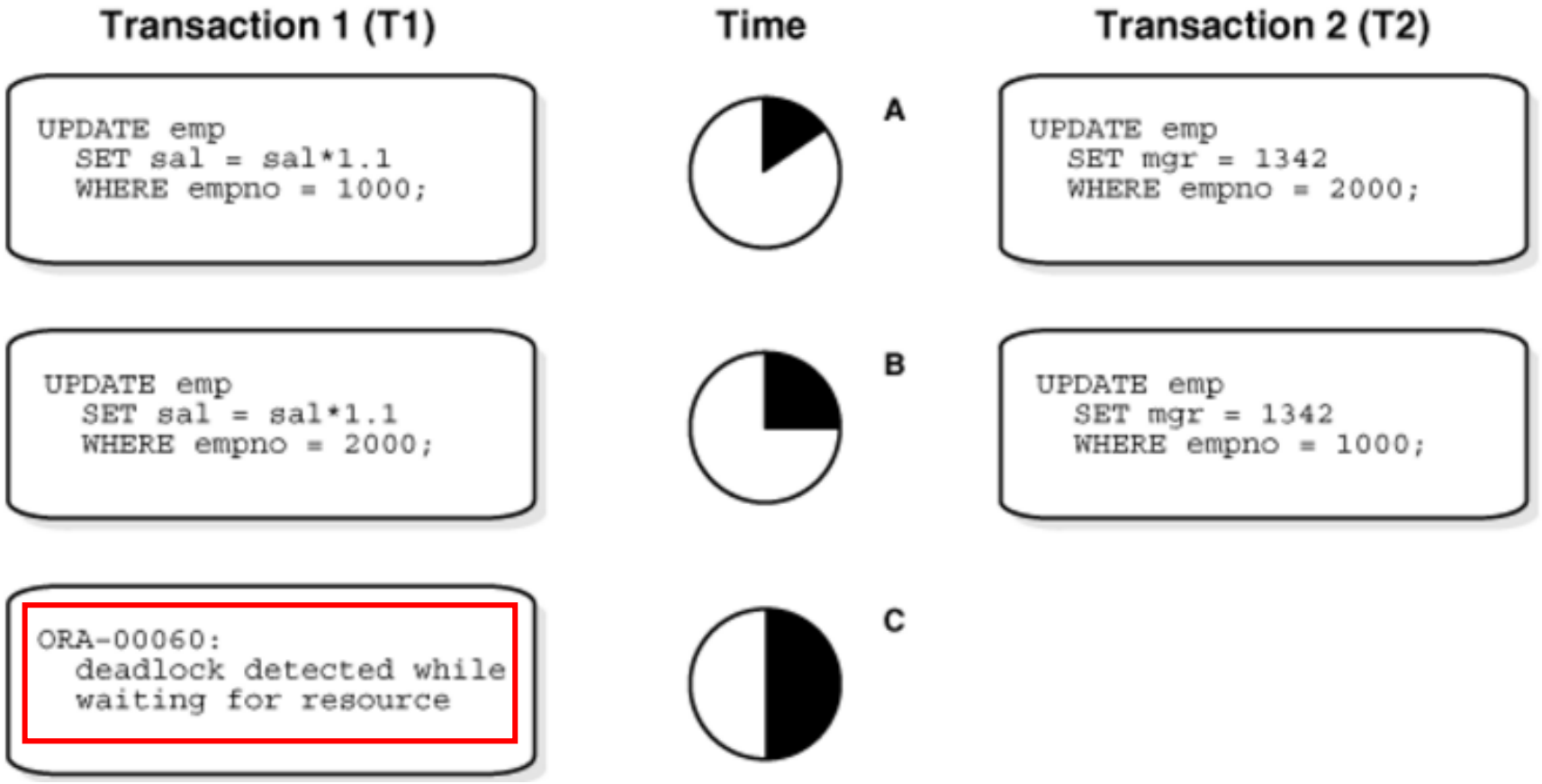
T_1	T_2	Gerenciador de Controle de concorrência
lock-X (B)		
		grant-X(B,T_1)
read(B)		
$B := B - 50$		
write (B)		
unlock(B)		
	lock-S(A)	
		grant-S(A,T_2)
	read(A)	
	unlock(A)	
	lock-S(B)	
		grant-S(B,T_2)
	read(B)	
	unlock(B)	
	display($A + B$)	
lock-X(A)		
		grant-X(A,T_2)
read(A)		
$A := A + 50$		
write (A)		
unlock(A)		

Ok...

E se uma transação **A** estiver esperando um **recurso/item N** que a transação **B** está bloqueando;

e se **B** estiver esperando um **recurso/item M** que **A** está bloqueando???

Exemplo

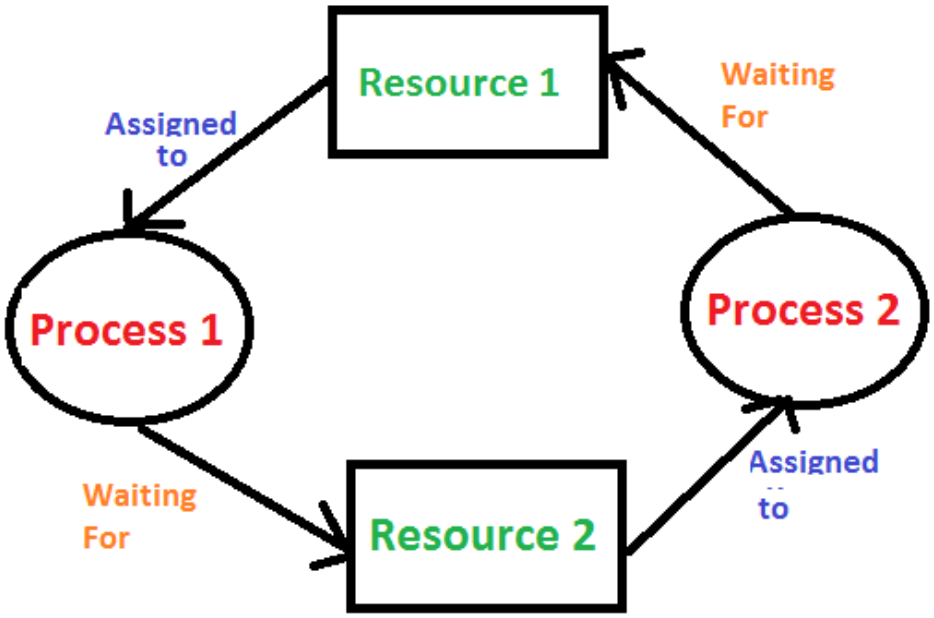


Deadlock (impasse): quando dois ou mais processos/transações ficam impedidos de continuar suas execuções, esperando uns pelos outros.


You release the lock first
Once I have finished
my task, you can continue.
Why should I?
You release the lock first
and wait until
I complete my task.

Deadlock em SGBD

- Como resolver?
  - Detectar o Deadlock
  - Prevenir o Deadlock



Detectar o Deadlock

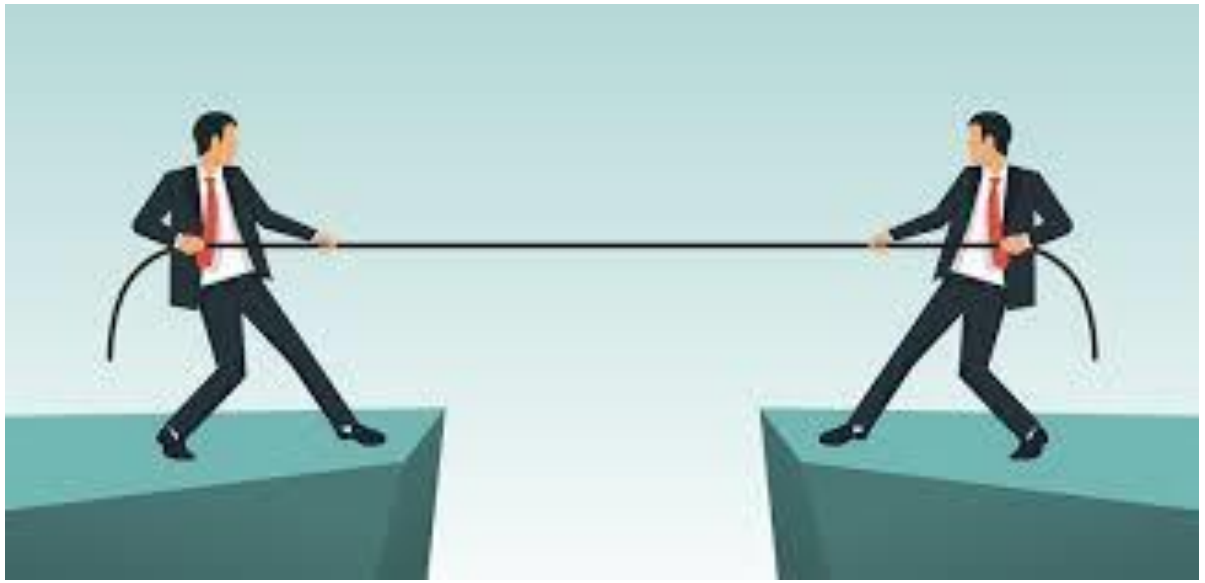
- Periodicamente verificam-se **dependências entre transações**, formando um grafo
  - **Caso o grafo forme um ciclo, há um deadlock!**
    - Uma transação é escolhida para ser **desfeita e refeita** (undo e redo): a mais antiga ou a mais recente ou a que bloqueou mais recursos; a escolha varia de fabricante para fabricante
- Observações:
  - É uma estratégia simples
  - Resolver um deadlock é **caro** pois transações com muitas operações podem precisar ser desfeitas ou refeitas
  - Pode matar uma mesma transação muitas vezes

Prevenir deadlock

- Para cada tentativa de bloqueio, o SGBD constrói o grafo de dependência ***a priori***
  - Caso o bloqueio venha a causar **deadlock**, ele é "**atrasado**" por alguns instantes, e nova tentativa é feita
- Observações:
  - De modo geral a prevenção demanda **mais esforço**
    - **Construção de grafos de dependência a cada tentativa de bloqueio**
  - A prevenção pode deixar uma transação "de castigo" por muito tempo

Outra Situação

Além de bloqueios, como eu resolvo **problemas de concorrência** no **mundo real**??



Alternativa...

⇒Mecanismo de concorrência baseado no registro do tempo

- **Cada transação tem um *timestamp* (registro de tempo) emitido quando entra no sistema.**

Se uma transação antiga Ti tem timestamp TS(Ti), uma nova transação Tj recebe o timestamp TS(Tj) de modo que TS(Ti) < TS(Tj)

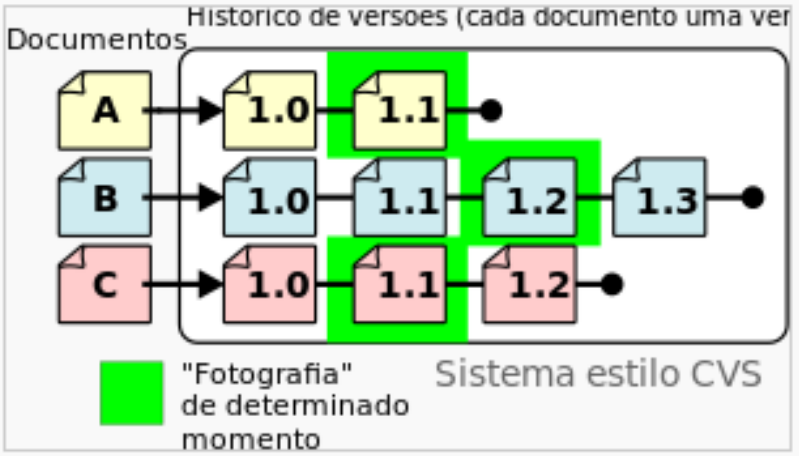
****** clock do sistema ou contador lógico

** O protocolo de ordenação por timestamp garante que quaisquer operações read e write em conflito sejam executadas por ordem de timestamp

Humm... Alguma outra forma?

MVCC = *multiversion concurrency control*

SGBDs mantêm os valores antigos de um item de dado quando o item é atualizado



- Cada item (registro) no sistema pode ter várias **versões visíveis** a **diferentes transações**.
  - E ao longo do tempo

⇒O SGBD passa a trabalhar com diversas versões

Voltando para a Linguagem SQL ...

E para o PostgreSQL

Início e fim de uma transação no postgres

BEGIN - -iniciar

- - comandos
- COMMIT - -comitar/confirmar
- ROLLBACK - -parar/cancelar
- END - -mesma função do COMMIT

➢ **Fim:**

- É emitida a cláusula **commit, end** ou **rollback**
- Um usuário se desconecta do postgres (commit automaticamente executado)
- Um processo de usuário termina anormalmente (há o rollback)

Por default, todo comando individual é considerado uma transação

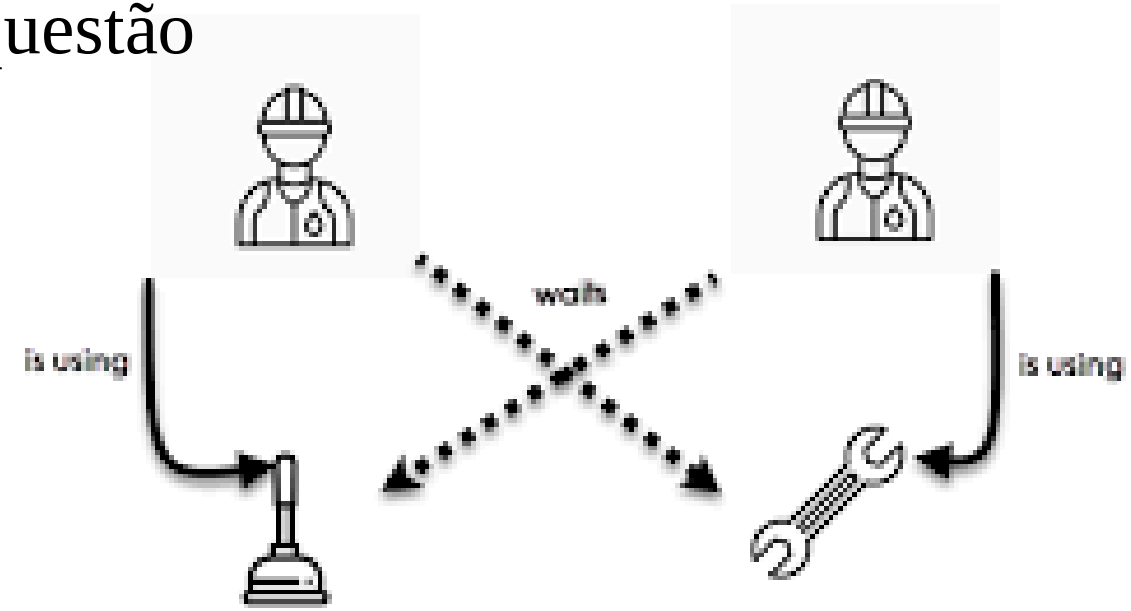
T1 = DELETE FROM Teste;

No Postgres

- Protocolos em conjunto
  - MVCC + Bloqueios + timestamp
  - Locks podem ser explícitos, caso necessários
    - LOCK TABLE produto IN SHARE MODE;
      - Permite que outras transações apenas consultem a tabela
    - LOCK TABLE produto IN EXCLUSIVE MODE;
      - Apenas uma transação pode obter um bloqueio exclusivo por tabela
      - Evita que outras transações façam locks nela

Deadlock no postgres

- Automaticamente detecta um deadlock
  - Resolve a situação fazendo um **rollback** de uma das sentenças que provocou o deadlock
    - Assim, libera um conjunto dos recursos em questão



Savepoint

SAVEPOINT *savepoint_name*

- Marcador intermediário
  - Dentro do contexto de uma transação.
  - Divide uma transação longa em partes menores.
  - Pode-se fazer um rollback antes de um desses pontos, sem necessariamente voltar ao início da transação.
  - Ex: SAVEPOINT sp1;

- Geralmente usado com o comando ROLLBACK TO

Exemplo

BD: Pedidos
User: postgres

Lembrar de desabilitar o autocommit e o autorollback!!

Não deixem de testar esse código passo a passo

```
Begin;
UPDATE vendedor
 SET salariofixo = 6000 WHERE codvend = 1;
SAVEPOINT a_v1;
UPDATE vendedor
 SET salariofixo = 5000 WHERE codvend = 2;
SAVEPOINT a_v2;
Select * from vendedor;
SELECT sum(salariofixo) FROM vendedor;
ROLLBACK TO SAVEPOINT a_v1;
SELECT SUM(salariofixo) FROM vendedor;
Select * from vendedor;
UPDATE vendedor
 SET salariofixo = 5000 WHERE codvend = 2;
Select * from vendedor;
COMMIT;
```