



Transações - PostgreSQL

Profa. Maria Camila Nardini Barioni

camila.barioni@ufu.br

Bloco B - sala 1B137

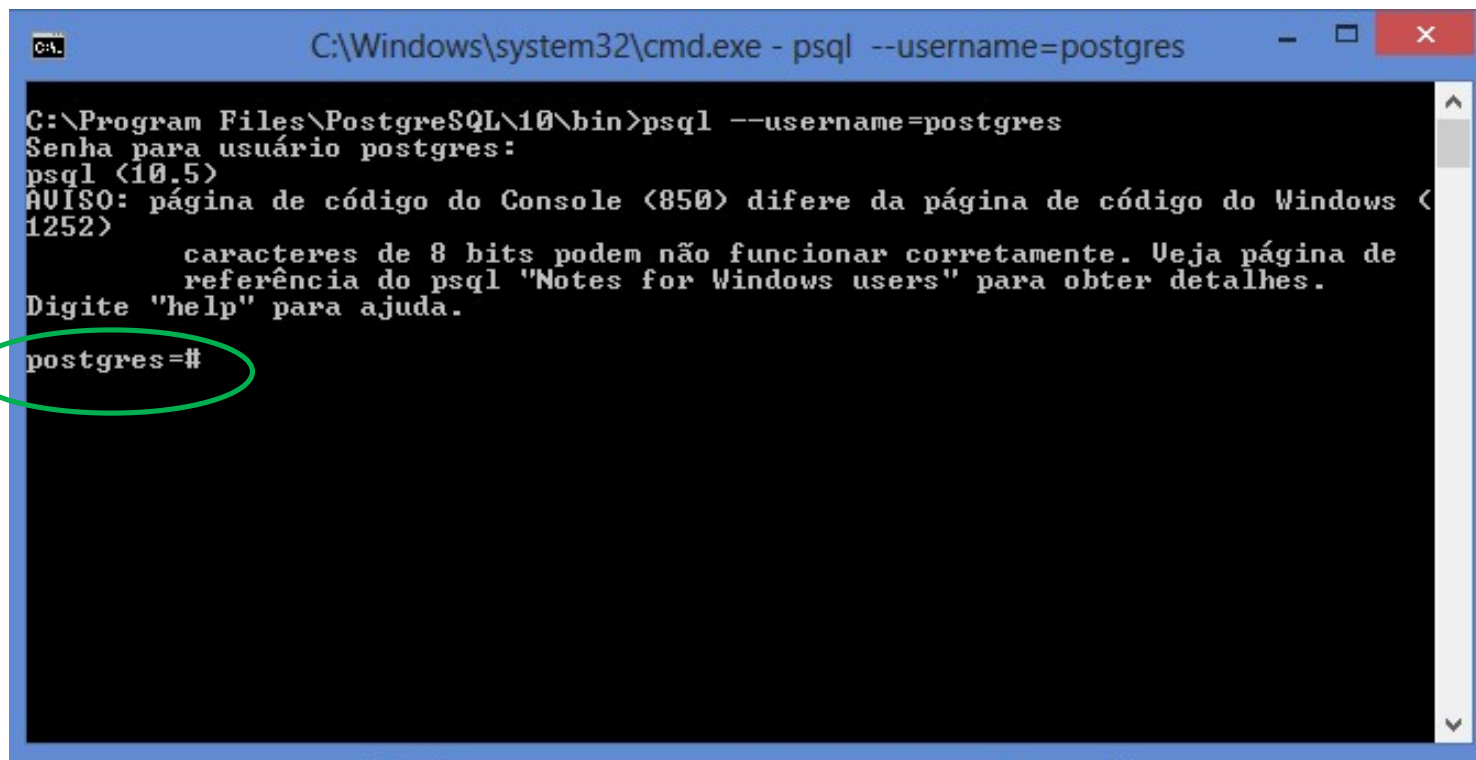
2º semestre de 2023

Orientações

- ◆ Essa atividade deverá ser executada em linha de comando
- ◆ Executar os comandos conforme indicado nos slides
- ◆ Enviar a resolução na tarefa no Teams incluindo os nomes dos integrantes do grupo
- ◆ Prazo para o envio: até o final da aula (05/04)

Abrir Prompt de comando (para quem instalou o servidor PostgreSQL localmente)

- ◆ Verifique onde está instalado o PostgreSQL na sua máquina



```
C:\Windows\system32\cmd.exe - psql --username=postgres

C:\Program Files\PostgreSQL\10\bin>psql --username=postgres
Senha para usuário postgres:
psql (10.5)
AVISO: página de código do Console (850) difere da página de código do Windows (1252)
      caracteres de 8 bits podem não funcionar corretamente. Veja página de
      referência do psql "Notes for Windows users" para obter detalhes.
Digite "help" para ajuda.
postgres=#
```

Esse é o banco de dados selecionado

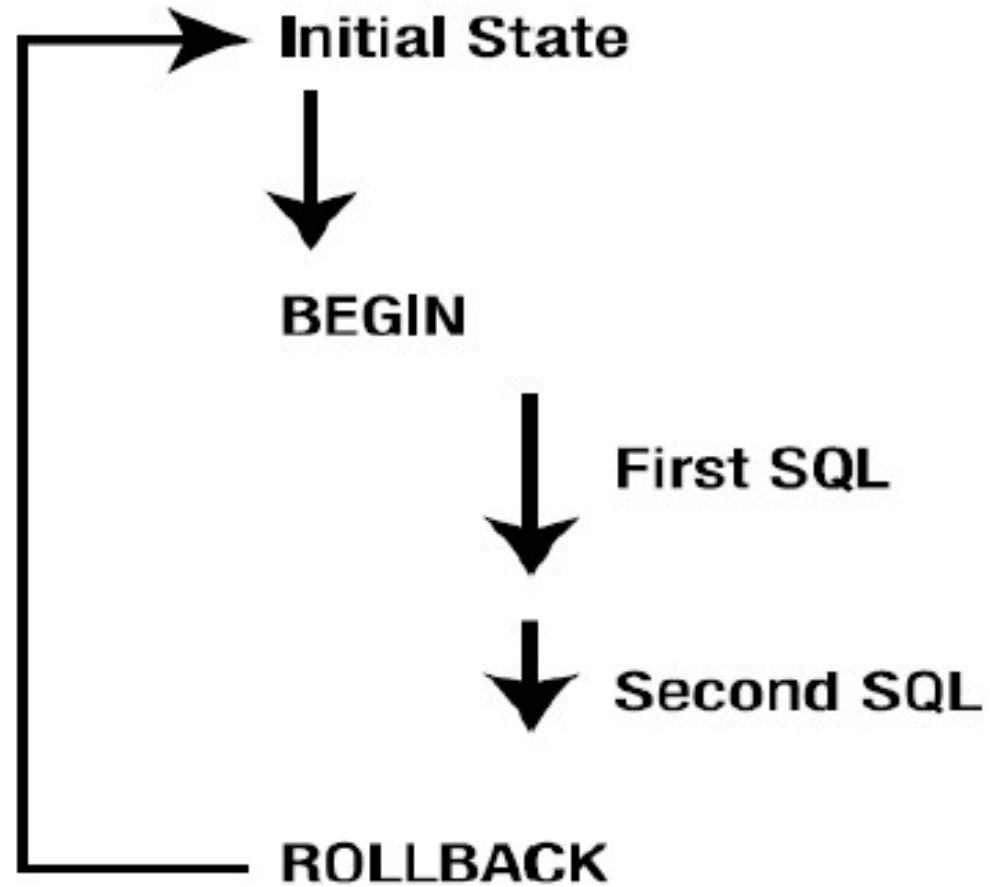
Relembrando: Propriedades desejáveis

- ◆ Para garantir a integridade dos dados
 - **Atomicidade:** todas as operações da transação são refletidas corretamente no BD, ou nenhuma delas
 - **Consistência:** a execução de uma transação isolada preserva a consistência do BD
 - ◆ Estado inicial consistente -> Estado final consistente
 - **Isolamento:** Cada transação não está ciente das outras transações executando simultaneamente no sistema
 - **Durabilidade:** As alterações realizadas por uma transação que completou com sucesso são persistidas mesmo que ocorram falhas no sistema

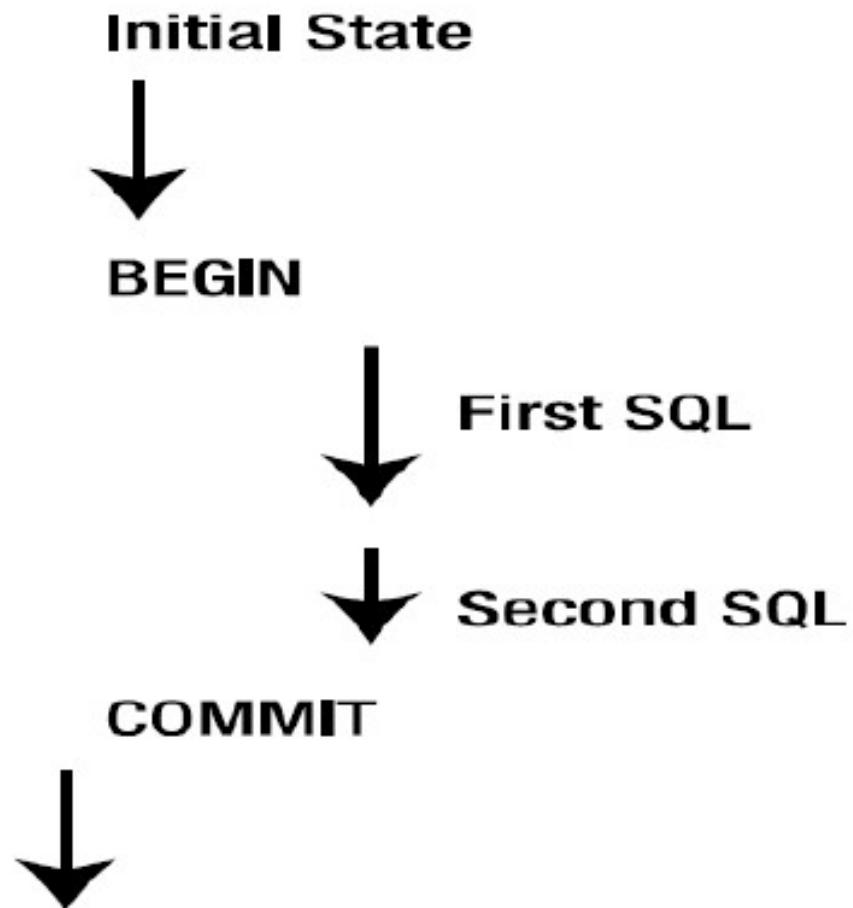
Relembrando: Conceitos de Transação e Sistema

- ◆ Uma transação é uma unidade atômica de trabalho que é completada ***integralmente*** ou ***não é realizada***
- ◆ O gerenciador de recuperação mantém o controle das seguintes operações
 - **BEGIN_TRANSACTION**: marca o início da execução da transação
 - **READ** ou **WRITE**: especificam operações de leitura ou gravação em itens de banco de dados, que são executados como parte da transação
 - **END_TRANSACTION**: Especifica que as operações de READ e WRITE terminaram, e marca o fim da execução da transação. Nesse ponto, verifica se as mudanças devem ser efetivadas ou se a transação deve ser abortada
 - **COMMIT_TRANSACTION**: Indica o término com sucesso da transação de forma que as atualizações podem ser seguramente efetivadas
 - **ROLLBACK**: Indica que a transação não terminou com sucesso e possíveis atualizações devem ser desfeitas

ROLLBACK



COMMIT



EXEMPLO

◆ Execute

```
CREATE TABLE TESTE1(  
  atributo11 integer,  
  atributo12 varchar(60));
```

```
CREATE TABLE TESTE2(  
  atributo21 integer,  
  atributo22 varchar(60));
```


EXEMPLO

◆ Execute

```
INSERT INTO TESTE1 (atributo11, atributo12) VALUES (1, 'David');
```

```
BEGIN;
```

```
UPDATE TESTE1 SET atributo12 = 'Dave' WHERE atributo11 = 1;
```

```
SELECT atributo12 FROM TESTE1 WHERE atributo11 = 1;
```

Responda: Qual a saída?

EXEMPLO

◆ Execute

ROLLBACK;

SELECT atributo12 FROM TESTE1 WHERE atributo11 = 1;

Responda: Qual a nova saída?

Exemplo 2

◆ Execute:

```
DELETE FROM TESTE1;
```

```
DELETE FROM TESTE2;
```

```
INSERT INTO TESTE1 (atributo11, atributo12) VALUES (1, 'David');
```

```
BEGIN;
```

```
INSERT INTO TESTE2 (atributo21, atributo22) VALUES (2, 'Arthur');
```

```
UPDATE TESTE1 SET atributo12 = 'Robert' WHERE atributo11 = 1;
```

```
SELECT * FROM TESTE1;
```

Exemplo 2

◆ Execute:

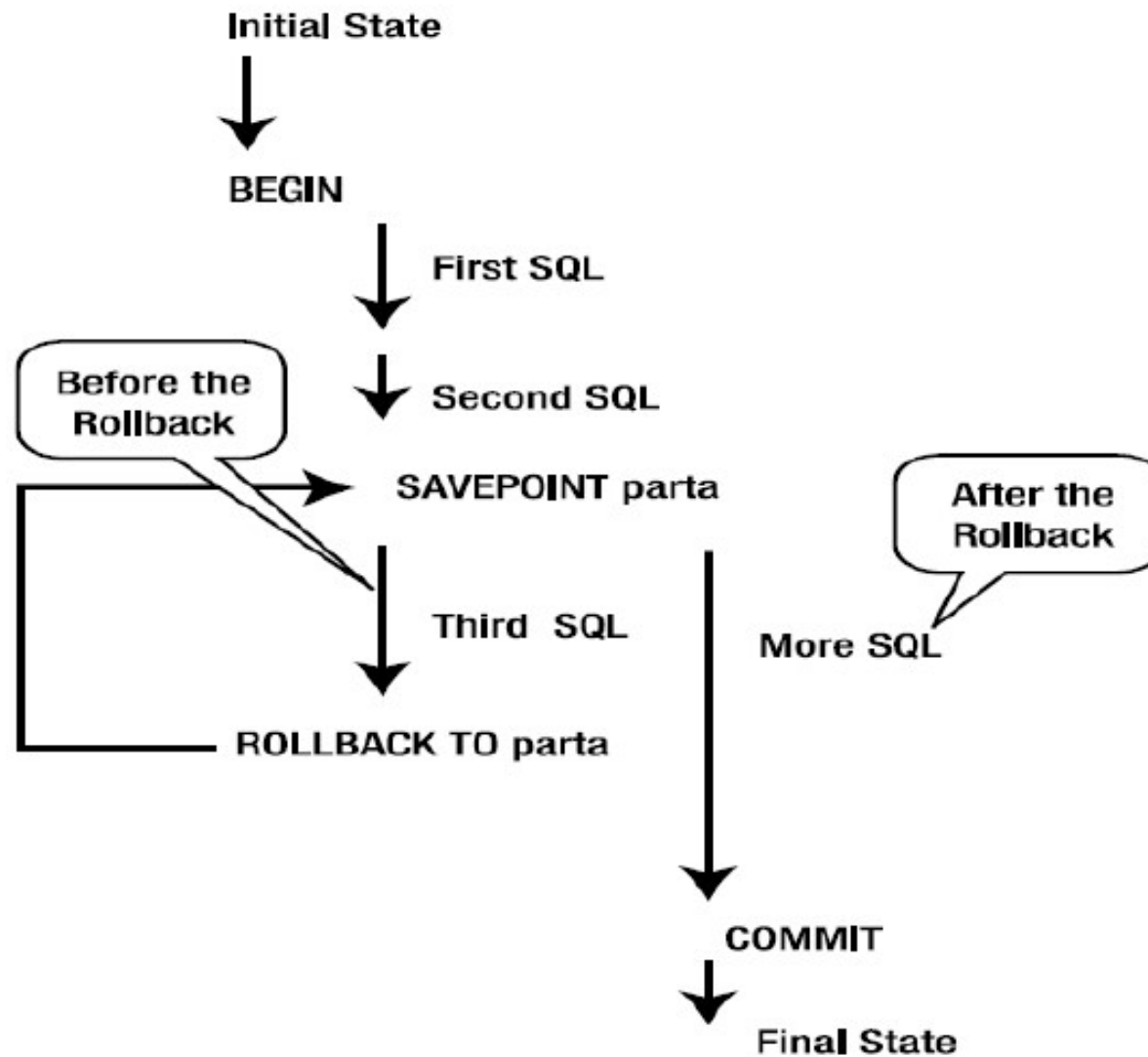
ROLLBACK;

SELECT * FROM TESTE1;

SELECT * FROM TESTE2;

Responda: Quais as saídas?

SAVEPOINTS



Exemplo

◆ Execute:

```
DELETE FROM TESTE1;
```

```
DELETE FROM TESTE2;
```

```
INSERT INTO TESTE1 (atributo11, atributo12) VALUES (1, 'David');
```

```
BEGIN;
```

```
INSERT INTO TESTE2 (atributo21, atributo22) VALUES (2, 'Arthur');
```

```
SAVEPOINT FIRST;
```

```
UPDATE TESTE1 SET atributo12 = 'Robert' WHERE atributo11 = 1;
```

```
SELECT * FROM TESTE1;
```

Responda: Qual a saída?

Exemplo

◆ Execute:

ROLLBACK TO FIRST;

SELECT * FROM TESTE1;

SELECT * FROM TESTE2;

Responda: Qual a saída?

ROLLBACK;

SELECT * FROM TESTE1;

SELECT * FROM TESTE2;

Responda: Qual a saída?

Exemplo

◆ Após o último ROLLBACK a transação está completa

- Execute

```
INSERT INTO TESTE2 (atributo21, atributo22) VALUES (9, 'Chris');
```

```
COMMIT;
```


Exemplo

◆ Após o COMMIT não é possível fazer um ROLLBACK

- Execute e observe as saídas

```
SELECT * FROM TESTE2;
```

```
BEGIN;
```

```
UPDATE TESTE2 SET atributo22 = 'Gill' WHERE atributo21 = 9;
```

```
COMMIT;
```

```
ROLLBACK;
```

```
SELECT * FROM TESTE2;
```

Limitações das transações

◆ Aninhamento de transação

- Não é possível em PostgreSQL (e vários outros SGBDs)

◆ Tamanho

- Procure manter as transações pequenas para não sobrecarregar com travamentos ("lock")

◆ Tempo

- Os dados envolvidos em uma transação ficam travados ("lock") para outros usuários e somente são disponibilizados após um COMMIT ou ROLLBACK. Por essa razão não é recomendável transações longas

Bloqueio (locking)

- ◆ A maioria dos BD implementam transações, particularmente para o isolamento de diferentes transações de usuários
- ◆ Um “sharedlock” (bloqueio compartilhado) permite aos outros usuários lerem, mas não atualizarem os dados
- ◆ Um “exclusive lock” (bloqueio exclusivo) impede inclusive que outras transações de leiam os dados
- ◆ Exemplo: um servidor irá bloquear as linhas que estão sendo alteradas por uma transação até que a transação esteja completa –isso é feito automaticamente

Isolamento versus performance multiusuário

◆ “Trade-off” entre

- concorrência, performance e minimização do número de locks de um lado,
- consistência e comportamento ideal do outro
- aumentar o isolamento diminui a performance multiusuário

Bloqueio

- ◆ Duas circunstâncias para a realização de um locking(bloqueio)
 - Evitar deadlocks
 - Bloqueio específico de uma aplicação

Deadlocks

- ◆ O que acontece se duas aplicações diferentes tentam alterar o mesmo dado ao mesmo tempo

Deadlocks – Exemplo

- ◆ Abra dois prompts e execute os comandos abaixo, na ordem indicada, e veja o que acontece

```
-- no primeiro console
SET SEARCH_PATH TO LOCADORA;
BEGIN;
UPDATE CLIENTE SET
nome= 'João Gilberto'
WHERE numcliente = '1';
```

```
UPDATE CLIENTE
SET nome= 'Marina Lima'
WHERE numcliente = '2';
```

```
-- no segundo console
SET SEARCH_PATH TO LOCADORA;
BEGIN;
```

```
UPDATE CLIENTE SET
nome= 'Laura Martins'
WHERE numcliente = '2';
```

```
UPDATE CLIENTE
SET nome= 'João Gama'
WHERE numcliente = '1';
```

Deadlocks – Exemplo

◆ **Resposta: qual a saída?**

◆ **Observação:**

- A seção que mostra o erro será cancelada (ROLLBACK) e na outra seção poderá ser executado um COMMIT
- O Deadlock é detectado automaticamente pelo PostgreSQL

Use o comando `\q` para encerrar cada uma das conexões

Deadlocks – Exemplo

```
-- no primeiro console  
SET SEARCH_PATH TO LOCADORA;  
BEGIN;  
UPDATE CLIENTE SET  
nome= 'João Gilberto'  
WHERE numcliente = '1';  
TRAVA LINHA NUMCLIENTE = 1
```

```
UPDATE CLIENTE  
SET nome= 'Marina Lima'  
WHERE numcliente = '2';  
AGUARDA O DESTRAVAMENTO DA  
LINHA NUMCLIENTE = 2
```

```
-- no segundo console  
SET SEARCH_PATH TO LOCADORA;  
BEGIN;
```

```
UPDATE CLIENTE SET  
nome= 'Laura Martins'  
WHERE numcliente = '2';  
TRAVA LINHA NUMCLIENTE = 2
```

```
UPDATE CLIENTE  
SET nome= 'João Gama'  
WHERE numcliente = '1';  
AGUARDA O DESTRAVAMENTO DA  
LINHA NUMCLIENTE = 1
```

Evitando deadlocks

- ◆ Sempre mantenha a transação pequena, a menor possível. Isso evita o bloqueio de tabelas e linhas e diminui a chance de um deadlock
- ◆ Outra opção é sempre usar a mesma ordem no processamento de linhas e colunas
- ◆ Se a ordem de alteração no exemplo anterior fosse a mesma o deadlock não ocorreria

Bibliografia

- ◆ Elmasri, Ramez; Navathe, Shamkant B. **Sistemas de banco de dados**. 4 ed. São Paulo: Addison Wesley, 2005, 724 p. Bibliografia: p. [690]-714.
- ◆ Material Didático produzido pelo professor Bruno Travençolo