



# AS33C

## Banco de Dados 2

Professor: Eduardo Cotrin Teixeira



[cotrin@utfpr.edu.br](mailto:cotrin@utfpr.edu.br)



# *SQL – Transações*

---

- Sistemas de BD geralmente são acessados por muitos usuários ou processos ao mesmo tempo.
- Esses acessos podem ser de consulta ou modificação de dados.
- A capacidade de trabalhar com múltiplas operações ao mesmo tempo pode garantir o bom desempenho em sistemas muito acessados, ou em ambientes com muitos processos concorrentes.
- O SGBD precisa evitar os problemas causados pela interação entre os processos.



# *SQL – Transações*

---

- Exemplo (clássico) de uma interação entre processos:
- Em uma instituição financeira, um usuário X transfere R\$ 100,00 para um usuário Y. Esta operação, na realidade, é composta de duas operações (cuja ordem não importa):
  - Débito de R\$ 100,00 para X.
  - Crédito de R\$ 100,00 para Y.

**E se uma das operações falhar ????**



# *SQL – Transações*

---

- O SGBD deve garantir transações ACID para lidar com as operações com segurança.
- Transações ACID são:
  - Atômicas: ou a transação completa é feita, ou nada é feito.
  - Consistentes: as restrições do BD devem ser garantidas.
  - Isoladas: para o usuário, deve parecer que há somente o seu processo em execução.
  - Duráveis: os efeitos do processo devem "sobreviver" a uma falha.



# *SQL – Transações*

---

- O controle das operações é feito pelo SGBD com o conceito de **Transações**.
- **Transação** = processo envolvendo consultas e/ou modificações no BD.
- Geralmente são conjuntos de instruções que devem ser tratadas com uma única operação.
- Em SQL, é formada por comandos simples **ou pelo controle explícito do programador**.



# SQL – Transações

- No PostgreSQL, os comandos de controle das transações são:
- **BEGIN:** marca o **início** de uma transação
- **COMMIT:** comando SQL que causa o **encerramento** da transação. Depois do COMMIT, as modificações feitas no BD na transação (a partir do BEGIN) se tornam permanentes.
- **ROLLBACK:** comando SQL que também causa o encerramento da transação, mas **abortando-a**. Após um ROLLBACK nenhuma modificação é feita no BD de fato.

*\* Obs.: Para comandos simples e processos isolados, o SGBD dispara automaticamente os comandos de controle necessários.*



# SQL – Transações

---

- O uso de transações é especialmente importante em **sistemas concorrentes**.
- Alguns **problemas de isolamento** já são bastante conhecidos, e tratados pela maioria dos SGBDs:
- **Dirty Read** (ou "leitura suja"): acontece quando uma transação lê algum dado alterado por outra transação que ainda não foi confirmada.
- **Nonrepeatable Read** ("leitura não repetível"): uma mesma transação lê novamente dados lidos anteriormente, e descobre que algum dado foi alterado ou apagado por outra transação (que efetivou a operação após a primeira leitura).
- **Phantom Read** ("leitura fantasma"): parecido com o das leituras não repetíveis, porém ocorre quando a transação executa uma segunda vez uma consulta e descobre que foi inserido um novo dado.



# SQL – Transações

---

- Resumindo:
    - ***Dirty Read*** (ou "leitura suja"): ler um dado não confirmado.
    - ***Nonrepeatable Read*** ("leitura não repetível"): leitura muda na mesma transação (update ou delete sem commit → com commit).
    - ***Phantom Read*** ("leitura fantasma"): leitura muda na mesma transação (insert sem commit → com commit).
- \* Cada usuário deve enxergar o BD isoladamente em uma mesma transação.**



# SQL – Transações

- PostgreSQL **não** permite *Dirty Read* por padrão:

## Instância 1 (psql)

```
#begin;  
BEGIN  
#update Peca set PeNome='Espelho'  
where PeNome='Painel';  
UPDATE 1  
#select PeNome from Peca;  
penome
```

-----  
Cinto  
Volante  
Lanterna  
Limpador  
Espelho  
(5 registros)

## Instância 2 (psql)

```
#begin;  
BEGIN  
#select PeNome from Peca;  
penome
```

-----  
Cinto  
Volante  
Lanterna  
Limpador  
Painel  
(5 registros)

→  
A alteração sem commit  
da Instância 1 não é vista ,  
até aqui tudo bem!

# SQL – Transações

- Mas permite *Nonrepeatable Read* !

## continuando Instância 1

```
#commit;  
COMMIT
```

**Cada usuário deve enxergar  
o BD isoladamente em uma  
mesma transação.**

## continuando Instância 2

```
select PeNome from Peca;  
penome
```

-----  
Cinto  
Volante  
Lanterna  
Limpador  
**Espelho**  
(5 registros)  
#commit;  
COMMIT

Mesmo SELECT retornou  
resultado diferente na  
mesma transação !

# SQL – Transações

- E permite também *Phantom Read* :

## Instância 1

```
#begin;  
BEGIN  
#insert into Peca values  
('PE6','Banco', 50,'Preto');  
INSERT 0 1  
#select PeNome from Peca;  
penome
```

-----  
Cinto  
Volante  
Lanterna  
Limpador  
Espelho  
Banco  
(6 registros)

## Instância 2

```
#begin;  
BEGIN  
#select PeNome from Peca;  
penome
```

-----  
Cinto  
Volante  
Lanterna  
Limpador  
Painel  
(5 registros)

→ A inserção sem commit da Instância 1 não é vista.

# SQL – Transações

## ■ *Phantom Read* :

### continuando Instância 1

```
commit;  
COMMIT
```

### continuando Instância 2

```
select PeNome from Peca;  
penome
```

-----  
Cinto  
Volante  
Lanterna  
Limpador  
Espelho  
**Banco**  
(6 registros)  
commit;  
COMMIT

Mesmo SELECT retornou  
mais um registro na  
mesma transação !



# SQL – Transações

---

- SQL prevê ***níveis de isolamento*** para tratar estes problemas.
- No PostgreSQL, o comando **SET TRANSACTION** define o nível de isolamento.
- Um comando **SET TRANSACTION ISOLATION LEVEL *REPEATABLE READ***; logo após o BEGIN que inicia a transação impede os problemas de isolamento descritos.

# SQL – Transações

- Por exemplo, impedindo o *Nonrepeatable Read* :

## Instância 1

```
#begin;  
BEGIN  
#update Peca set  
PeNome='Painel' where PeNome='Espelho';  
UPDATE 1  
#select PeNome from Peca;  
penome
```

-----

Cinto  
Volante  
Lanterna  
Limpador  
Banco  
Painel  
(6 registros)

## Instância 2

```
#begin;  
BEGIN  
#SET TRANSACTION  
ISOLATION LEVEL REPEATABLE READ;  
SET  
#select PeNome from Peca;  
penome
```

-----

Cinto  
Volante  
Lanterna  
Limpador  
Espelho  
Banco  
(6 registros)

→ Não houve Dirty  
Read (padrão).

# SQL – Transações

- Impedindo o *Nonrepeatable Read* :

## continuando Instância 1

**commit;**

**COMMIT**

**select PeNome from Peca;  
penome**

-----  
**Cinto  
Volante  
Lanterna  
Limpador  
Banco  
Painel  
(6 registros)**

## continuando Instância 2

**select PeNome from Peca;  
penome**

-----  
**Cinto  
Volante  
Lanterna  
Limpador  
Espelho  
Banco  
(6 registros)  
commit;  
COMMIT**

Mesmo SELECT agora  
retorna o mesmo  
resultado !