CEFET/RJ

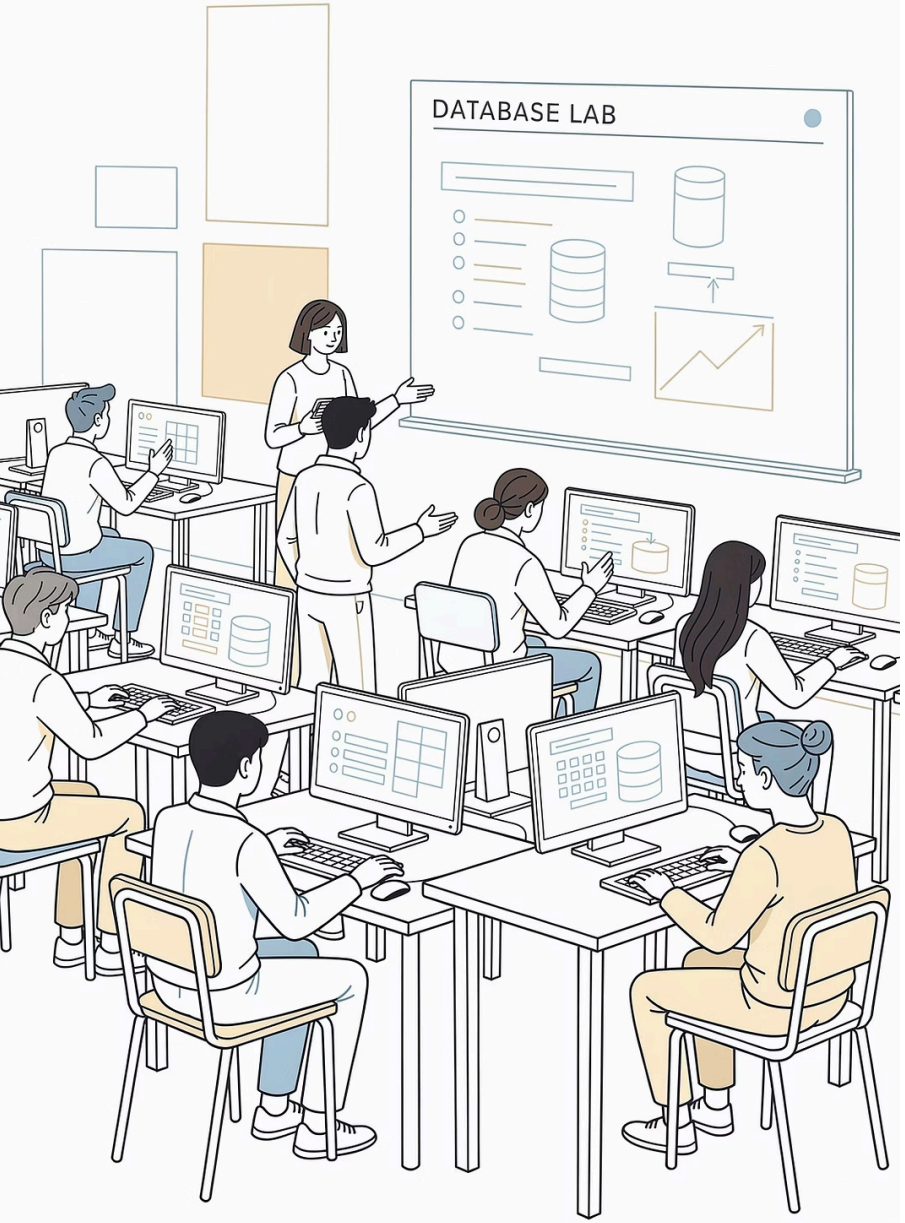
MATERIAL COMPLEMENTAR

Prática de Controle de Concorrência

Nesta aula prática, vamos explorar como o PostgreSQL gerencia transações simultâneas e garante a integridade dos dados em ambientes com múltiplos usuários acessando o banco de dados ao mesmo tempo.

Eduardo Ogasawara

eduardo.ogasawara@cefet-rj.br
https://eic.cefet-rj.br/~eogasawara



Objetivos da Prática

Compreender Concorrência

Entender como o PostgreSQL gerencia transações simultâneas e mantém a consistência dos dados

Observar Comportamentos

Analisar MVCC, locks explícitos, níveis de isolamento e transações concorrentes em ação

Relacionar Teoria e Prática

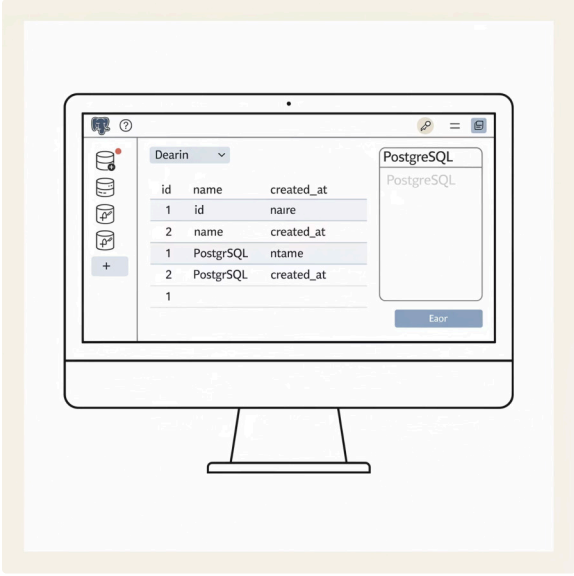
Conectar os conceitos teóricos vistos em aula com exemplos práticos e observáveis

Preparação do Ambiente

Configuração Necessária

Para realizar os experimentos, você precisará configurar um ambiente com múltiplas sessões simultâneas conectadas ao PostgreSQL. Cada sessão representará uma transação independente, permitindo observar o comportamento do controle de concorrência.

- SGBD: PostgreSQL
- Ferramentas: pgAdmin, psql ou DBeaver
- Abrir 2 ou 3 sessões simultâneas
- Todas conectadas ao mesmo banco de dados



Preparação da Base de Dados

01

Criar Tabelas

Execute o script Postgresql-CreateTables.sql para criar o esquema de exemplo com todas as tabelas necessárias

02

Popular Dados

Execute o script Postgresql-Inserts.sql para inserir os dados iniciais nas tabelas criadas

03

Confirmar Dados

Verifique que a tabela EMPREGADO contém os dados iniciais corretamente inseridos

```
-- Verificar dados iniciais
SELECT * FROM EMPREGADO;
```

CONCEITO FUNDAMENTAL

PostgreSQL e MVCC

O PostgreSQL utiliza MVCC (Multi-Version Concurrency Control), um mecanismo sofisticado que permite alta concorrência sem comprometer a consistência dos dados. Este sistema cria versões múltiplas dos dados para diferentes transações.

Leituras Não Bloqueiam Escritas

Transações de leitura podem acessar dados enquanto outras transações estão modificando-os

Escritas Não Bloqueiam Leituras

Transações de escrita não impedem outras transações de ler os dados

Dirty Reads Não Permitidos

Transações nunca veem dados não confirmados de outras transações

Bloqueios Específicos

Locks aparecem apenas em casos específicos de conflito de escrita

Níveis de Isolamento no PostgreSQL

O PostgreSQL oferece diferentes níveis de isolamento para controlar como as transações interagem entre si. Cada nível oferece um equilíbrio diferente entre desempenho e consistência.

READ COMMITTED

Nível padrão do PostgreSQL. Lê apenas dados já confirmados.

REPEATABLE READ

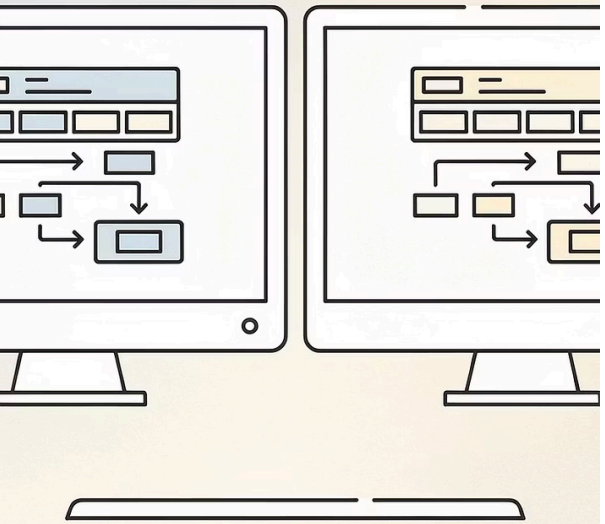
Garante leituras consistentes durante toda a transação.

SERIALIZABLE

Maior nível de isolamento, garante equivalência serial.

Observação importante: READ UNCOMMITTED é tratado como READ COMMITTED no PostgreSQL.

```
SHOW default_transaction_isolation;
```



Teste Básico de Transação

Observando o MVCC em Ação

Sessão 1

```
BEGIN;

INSERT INTO EMPREGADO
VALUES ('John','E','Borg',0,
        '1929-11-10',
        'Wall Street','M',
        55000,NULL,1);
```

Transação iniciada mas não confirmada

Sessão 2

```
SELECT * FROM EMPREGADO;
```

Perguntas:

- O novo registro aparece?
- Por quê?

Conceito Observado: Isolamento de Transações

Sessão 2 Não Vê o Registro

A inserção não aparece na consulta da Sessão 2

Transação Não Confirmada

A Sessão 1 ainda não executou COMMIT

Visão Consistente

PostgreSQL fornece snapshot consistente dos dados

MVCC Evita Dirty Reads

Proteção automática contra leituras sujas

Importante: Este comportamento garante que cada transação veja uma versão consistente do banco de dados, mesmo com múltiplas transações simultâneas modificando os dados.

Monitorando Sessões Ativas

O PostgreSQL oferece views do sistema que permitem observar o estado das transações em tempo real. A view `pg_stat_activity` é essencial para diagnóstico e monitoramento.

```
SELECT pid, state, wait_event_type,
      wait_event, query
FROM pg_stat_activity
WHERE datname = current_database();
```

Sessões Ativas

Identificar quais conexões estão atualmente ativas no banco de dados

Sessões em Espera

Detectar transações que estão aguardando recursos ou locks

Consultas em Execução

Visualizar os comandos SQL executados em cada sessão

Observando Locks no PostgreSQL

Consulta de Locks

```
SELECT pid, locktype,
      mode, granted,
      relation::regclass
FROM pg_locks;
```

Esta consulta revela informações detalhadas sobre todos os locks ativos no sistema.

Informações Obtidas

- **Tipo de Lock**

  Identifica se é lock de tabela, linha, transação, etc.

- **Modo do Lock**

  Indica o nível de restrição (compartilhado, exclusivo, etc.)

- **Status de Concessão**

  Mostra se o lock foi concedido ou está aguardando

Leitura Consistente em READ COMMITTED

O nível de isolamento READ COMMITTED é o padrão no PostgreSQL. Vamos observar como ele se comporta com transações concorrentes.

Sessão 2: Inicia Transação

1

```
BEGIN;
SELECT COUNT(*) FROM EMPREGADO;
```

2

Sessão 1: Insere e Confirma

```
BEGIN;
INSERT INTO EMPREGADO
VALUES ('John','E','Borg',1,
       '1930-01-01',
       'Wall Street','M',
       60000,NULL,1);
COMMIT;
```

Sessão 2: Nova Consulta

3

```
SELECT COUNT(*) FROM EMPREGADO;
```

O que acontece com o resultado?

Conceito Observado: READ COMMITTED

Comportamento do Snapshot

No nível READ COMMITTED, cada comando SQL usa um snapshot atualizado do banco de dados. Isso significa que a transação pode ver diferentes versões dos dados ao longo de sua execução.

- Cada comando usa um snapshot atualizado
- Não vê mudanças não confirmadas
- Pode ver mudanças confirmadas entre comandos
- READ COMMITTED é o padrão no PostgreSQL

Este comportamento permite maior concorrência, mas pode resultar em leituras não repetíveis dentro da mesma transação.

Snapshot Estável em REPEATABLE READ

O nível de isolamento REPEATABLE READ oferece maior consistência ao fixar o snapshot no início da transação.

Sessão 1: Define Isolamento

```
SET TRANSACTION ISOLATION
LEVEL
REPEATABLE READ;
BEGIN;
SELECT COUNT(*) FROM
EMPREGADO;
```

Sessão 2: Insere Novo Registro

```
INSERT INTO EMPREGADO
VALUES ('Jane','A','Doe',2,
'1931-02-02','Broadway',
'F',58000,NULL,1);
COMMIT;
```

Sessão 1: Consulta Novamente

```
SELECT COUNT(*) FROM
EMPREGADO;
```

O resultado será o mesmo da primeira consulta!

Conceito Observado: REPEATABLE READ

Snapshot Fixado

O snapshot é estabelecido no início da transação e permanece inalterado até o final, garantindo que todas as leituras vejam a mesma versão dos dados.

Isolamento de Mudanças

Mudanças confirmadas por outras transações não aparecem durante a execução da transação atual, mesmo após COMMIT de outras sessões.

Leitura Consistente

Garante que todas as consultas dentro da transação vejam exatamente os mesmos dados, evitando leituras não repetíveis e anomalias fantasma.

Bloqueio Explícito com FOR UPDATE

Sessão 1: Bloqueia Linha

```
BEGIN;

SELECT * FROM EMPREGADO
WHERE SSN = 0
FOR UPDATE;
```

A cláusula `FOR UPDATE` cria um lock explícito na linha selecionada, impedindo que outras transações a modifiquem.

Sessão 2: Tenta Atualizar

```
UPDATE EMPREGADO
SET SALARIO = 56000
WHERE SSN = 0;
```

Resultado: A Sessão 2 bloqueia e fica aguardando a liberação do lock pela Sessão 1.

EXEMPLO PRÁTICO

SERIALIZABLE e Conflitos

Quando duas sessões operam no nível SERIALIZABLE, o PostgreSQL monitora dependências de leitura/escrita para garantir a serializabilidade.

Sessão 1

```
SET TRANSACTION ISOLATION LEVEL
SERIALIZABLE;
BEGIN;
SELECT SUM(SALARIO)
FROM EMPREGADO
WHERE DNO = 1;
```

Sessão 2

```
SET TRANSACTION ISOLATION LEVEL
SERIALIZABLE;
BEGIN;
INSERT INTO EMPREGADO
VALUES ('Mark','T','Smith',3,
'1932-03-03','5th Avenue',
'M',59000,NULL,1);
COMMIT;
```

Resultado em SERIALIZABLE

Quando a Sessão 1 tenta fazer commit após a Sessão 2, pode ocorrer um erro de serialização:

> **could not serialize access due to read/write dependencies**

1

Detecção

PostgreSQL detecta conflito de dependências

2

Abort

Transação é abortada para manter corretude

3

Reexecução

A transação deve ser reexecutada

Conceito Observado: SERIALIZABLE

O nível SERIALIZABLE oferece o mais alto grau de isolamento, mas com custos de desempenho associados.

Equivalência Serial

PostgreSQL garante que o resultado é equivalente a uma execução serial das transações.

Predicate Locks

Utiliza predicate locks para detectar conflitos entre transações concorrentes.

Abort Controlado

Pode abortar transações estrategicamente para manter a corretude dos dados.

Trade-off

Maior custo de processamento em troca de maior isolamento e consistência.

BOAS PRÁTICAS

Encerramento e Limpeza

É fundamental encerrar transações adequadamente e realizar manutenção periódica para evitar o acúmulo de versões antigas de dados.

Comandos Essenciais

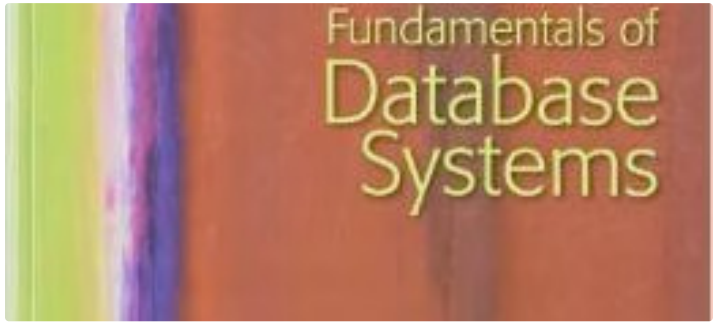
```
COMMIT;
ROLLBACK;
DELETE FROM EMPREGADO
WHERE SSN IN (0,1,2,3);
VACUUM (ANALYZE);
```

O comando VACUUM é importante para recuperar espaço e atualizar estatísticas do banco de dados.

Por que é importante?

- Evita crescimento de versões antigas
- Melhora o desempenho
- Libera espaço em disco
- Atualiza estatísticas do otimizador

Referências



Elmasri & Navathe

Fundamentals of Database Systems

Pearson, 2016

Referência abrangente sobre fundamentos de sistemas de bancos de dados, cobrindo aspectos teóricos e práticos.

Korth, Sudarshan & Silberschatz

Database System Concepts

McGraw-Hill, 2019

Texto fundamental que serviu como base para a maioria dos exemplos apresentados nesta apresentação.

Özsu & Valduriez

Principles of Distributed Database Systems

Springer Nature, 2019

Obra especializada em sistemas de bancos de dados distribuídos, essencial para compreensão avançada.

Nota: Os conceitos e exemplos apresentados baseiam-se principalmente na literatura clássica de sistemas de bancos de dados, em especial *Database System Concepts* e *Fundamentals of Database Systems*.