Banco de Dados – IMD0401

Aula 25 – Controle de Transação em Banco de Dados

João Carlos Xavier Júnior

jcxavier@imd.ufrn.br

INSTITUTO METRÓPOLE DIGITAL

UFRN UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

Controle de Transação

- **Transação**:
  - Uma transação é uma **unidade lógica** de execução que **acessa** e, possivelmente, **atualiza** vários itens de dados.
  - Uma sequência de ações que são consideradas uma unidade atômica (indivisível) de trabalho.
  - Acões elementares do SGBD:
    - Leituras (**read**s) e escritas (**write**s);
    - Ações especiais: **commit** (compromissamento ou efetivação de transação), **abort** (aborto de transação).

Controle de Transação

- **Transação**:
  - O SGBD 'vê' cada transação como uma sequência de **leituras** e **escritas** delimitada por:
    - comandos begin
    - e commit (ou abort).

Controle de Transação

- **Exemplo:**
  - Transferência bancária.

```
BEGIN [WORK]
    update conta1
    set saldo = saldo - 100

    update conta2
    set saldo = saldo + 100
COMMIT [WORK]
```

Estado da Transação

- Uma transação pode assumir os seguintes estados:
  - Ativa: estado inicial
  - Em efetivação: execução da última operação.
  - Em falha: erro na execução normal.
  - Efetivada: conclusão com sucesso.
  - Retornada: recuperação do estado anterior.

- Uma transação é dita concluída se estiver no estado **efetivada** (committed) ou **retornada** (rolled back).

Propriedades de uma Transação

- Uma transação deve preservar as seguintes propriedades:
  - **A**tomicidade: ou todas as operações são refletidas corretamente no banco de dados ou nenhuma será.
  - **C**onsistência: a execução de uma transação preserva a consistência dos dados.
  - **I**solamento: uma transação não toma conhecimento da existência de outras (concorrência de transações).
  - **D**urabilidade: após a conclusão de uma transação, os dados modificados persistem até que sejam alterados novamente.

Propriedades de uma Transação

- Considere a transação T que transfere 50 reais da conta A para a conta B:

```
1. leia(A);
2. A = A - 50;
3. escreva(A);
4. leia(B);
5. B = B + 50;
6. escreva(B);
```

- Suponha que 100 e 200 são, respectivamente, os valores de A e B antes da transação. Ao final da execução de T, os valores serão 50 e 250.

Propriedades de uma Transação

- A **consistência** significa que os valores de A com B deve permanecer inalterada após a execução da transação.
  - A propriedade de consistência é de responsabilidade do DBA (**Regras de Integridade**).

- A **atomicidade** possibilita que os valores antigos de A e B sejam mantidos durante uma transação, e no caso dela não ser completada, esses **valores são restabelecidos**.
  - Se houver uma falha na gravação em B, os valores seriam 50 e 200, havendo uma inconsistência.

Propriedades de uma Transação

- A **durabilidade** garante que as atualizações em A e B persistirão no banco de dados.
  - As propriedades de atomicidade e durabilidade são garantidas pelo **gerenciador de recuperação do SGBD**.

- O **isolamento** permite que outras transações sejam executadas simultaneamente.
  - A execução de uma transação **não deve sofrer interferência** de outras transações concorrentes.
  - A propriedade de isolamento é assegurada pelo **gerenciador de controle de concorrência** do SGBD.

Execução de Transações em SGBD

- Em sistemas de banco de dados, as transações podem ser executadas de **modo concorrente**.
  - A **concorrência** se faz necessária para se ter um **melhor uso dos recursos** do sistema.
- Uma transação concorre com outra quando ambas disputam o mesmo recurso (mesma bloco de registros).
- O sistema de banco de dados deve controlar a interação entre as **transações concorrentes**.
  - Identificação das **ordens de execução** das transações a partir de escalas de execução.

Execução de Transações em SGBD

- As transações podem ser executadas em qualquer ordem. Cada ordem é chamada de escala de execução (ou schedule).
- Uma escala de execução pode ser **sequencial** ou **concorrente**.
  - Na escala sequencial as operações de cada transação são executadas em sequência, ou seja, **uma por vez**.
  - Na escala concorrente as operações de cada transação são executadas **simultaneamente**.

Execução de Transações em SGBD

- **Escala sequencial**
  - Exemplo:

T1	T2
`leia(A);`	
`A = A - 50;`	
`escreva(A);`	
`leia(B);`	
`B = B + 50;`	
`escreva(B);`	
	`leia(A);`
	`A = A - 50;`
	`escreva(A);`
	`leia(B);`
	`B = B + 50;`
	`escreva(B);`

Execução de Transações em SGBD

- Escala concorrente **pode gerar inconsistência**.
  - Exemplo:

T1	T2
`leia(A);`	
`A = A - 50;`	
	`leia(A);`
	`A = A - 50;`
	`escreva(A);`
	`leia(B);`
`escreva(A);`	
`leia(B);`	
`B = B + 50;`	
`escreva(B);`	
	`B = B + 50;`
	`escreva(B);`

Transações em SQL

- Em geral, todo comando SQL é considerado uma transação.
  - Exemplo: **DELETE FROM Alunos** (exclui todas ou não exclui nenhuma tupla de Alunos).

- Para o padrão SQL-92 .
  - BEGIN ou START TRANSACTION
  - COMMIT
  - ROLLBACK

Transações em PostgreSQL

- A transação é definida envolvendo os comandos SQL da transação pelos comandos **BEGIN** e **COMMIT**.
- Exemplo:

```
BEGIN work;

UPDATE produto
 SET areaplantada = areaplantada + (areaplantada * 0.1)
WHERE descricao ilike 'Feijão verde';

UPDATE trabalhador
 SET especialidade = 'hortelão'
WHERE idtrabalhador = (SELECT idtrabalhador FROM produto
WHERE descricao ilike 'Feijão verde');

COMMIT work;
```

Transações em PostgreSQL

- É possível controlar os comandos na transação através dos pontos de salvamento (*savepoints*).
- Exemplo:

```
BEGIN;

UPDATE produto
 SET areaplantada = areaplantada + (areaplantada * 0.3)
WHERE descricao ilike 'Batata inglesa';

SAVEPOINT ponto_salvamento01;

UPDATE trabalhador
 SET especialidade = 'agrónomo'
WHERE idtrabalhador = (SELECT idtrabalhador FROM produto
                       WHERE descricao ilike 'Batata inglesa');

ROLLBACK TO ponto_salvamento01;
COMMIT;
```

Dúvidas...

Níveis de Isolamento

- **Níveis de isolamento**:
  - Serializável (**SERIALIZABLE**): cada transação executa com completo isolamento – padrão.
  - Leitura repetitiva (**REPEATABLE READ**): cada transação lê apenas tuplas efetivadas e nenhuma outra pode atualizar uma tupla que está em uso.
  - Leitura com efetivação (**READ COMMITTED**): cada transação lê apenas tuplas efetivadas e qualquer outra pode atualizar uma tupla que está em uso.
  - Leitura sem efetivação (**READ UNCOMMITTED**): cada transação lê apenas tuplas não efetivadas.

Problemas de Isolamento

- Em cenários **standalone** é possível desfazer todo um bloco de procedimentos apenas com um simples **Rollback**.
- Em sistemas concorrentes isso é mais complexo e difícil de garantir.
- Caso o isolamento não seja garantido alguns problemas podem ocorrer:
  - Dirty read;
  - Unrepeatable Reads;
  - Phantom Read.

Problemas de Isolamento

- **Dirty read**:
  - Acontece quando uma transação lê algum dado alterado por outra que acaba não confirmando suas alterações.
  - Problema preocupante.
- **Unrepeatable Reads**:
  - Acontece quando uma transação lê algum dado que será alterado (update ou delete) por outra transação.
  - Não chega a ser um problema preocupante.
- **Phantom Read**:
  - Parecido com o anterior. O que muda é que uma das transações em vez de alterar o conjunto de dados, insere um dado novo.

Problemas de Isolamento

- Exemplos:

Read Committed

What happens when one (unfinished) transaction inserts rows in a table and the other (also unfinished) transaction tries to read all rows in the table? If the second transaction is able to see the rows inserted by the first, then that read is called a **dirty read** – because the first transaction can rollback and the second transaction would have read "phantom" rows that never existed.

The *read committed* isolation level guarantees that dirty reads will never happen. Here is an example:

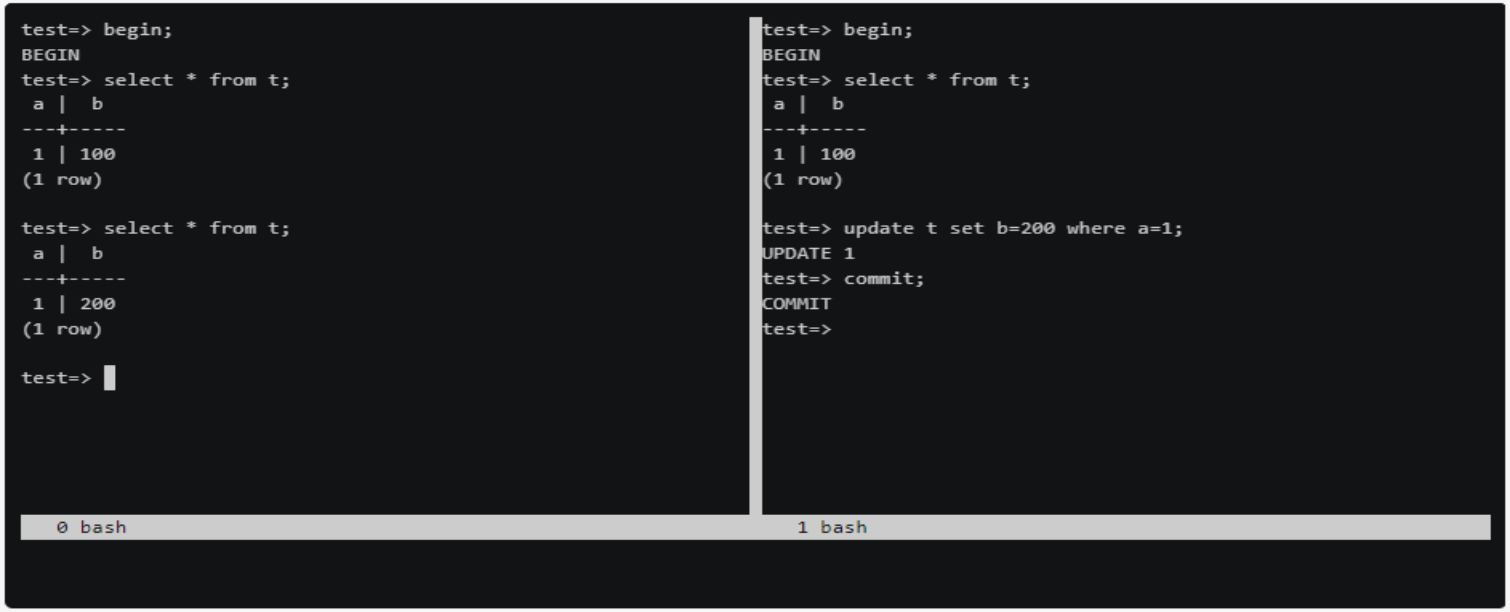
https://pgdash.io/blog/postgres-transactions.html

Problemas de Isolamento

- Exemplos:

Repeatable Read

Yet another problem is that of non-repeatable reads. These happen when a transaction reads a row, and then reads it again a bit later but gets a different result – because the row was updated in between by another transaction. The read has become **non-repeatable**, as shown in this example:



https://pgdash.io/blog/postgres-transactions.html

Isolamento Apropriado

- **Em resumo:**

Nível de isolamento	Dirty Read	Nonrepeatable Read	Phantom Read
Read uncommitted	Possível	Possível	Possível
Read committed	Impossível	Possível	Possível
Repeatable read	Impossível	Impossível	Possível
Serializable	Impossível	Impossível	Impossível

Transações em PostgreSQL

- **Modificando o nível de isolamento da transação corrente:**

```
SET TRANSACTION [ISOLATION
LEVEL { SERIALIZABLE |
REPEATABLE READ | READ
COMMITTED | READ UNCOMMITTED
} ]   [ {READ WRITE | READ
ONLY} ]
```

Transações em PostgreSQL

- Exemplo:

```
BEGIN;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
INSERT INTO ALUNO VALUES (100,'ALUNO100');
INSERT INTO ALUNO VALUES (101,'ALUNO101');
INSERT INTO ALUNO VALUES (102,'ALUNO102');
INSERT INTO ALUNO VALUES (103,'ALUNO103');
INSERT INTO ALUNO VALUES (104,'ALUNO104');
INSERT INTO ALUNO VALUES (105,'ALUNO105');
INSERT INTO ALUNO VALUES (106,'ALUNO106');
INSERT INTO ALUNO VALUES (107,'ALUNO107');
INSERT INTO ALUNO VALUES (108,'ALUNO108');
INSERT INTO ALUNO VALUES (109,'ALUNO109');
INSERT INTO ALUNO VALUES (110,'ALUNO110');
COMMIT;
```

Dúvidas...

Obrigado