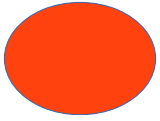
# Transações

## Conceitos básicos

Prof. André Britto



# Transações

- Coleção de operadores que forma uma **única unidade lógica** de trabalho.
- Conjunto de operações de manipulação de dados que executam uma única tarefa.
- Essa operações incluem inserções, remoções, atualizações ou recuperações de consultas.
- Todas as operações devem ser feitas ou

  nenhuma deve ser feita.

# Introdução

- Uma transação inclui uma ou mais operações de acesso ao banco de dados
  - Inserção, remoção, atualização, consulta.
- O conjunto de operações que forma uma transação pode ser especificado diretamente através de comandos SQL.

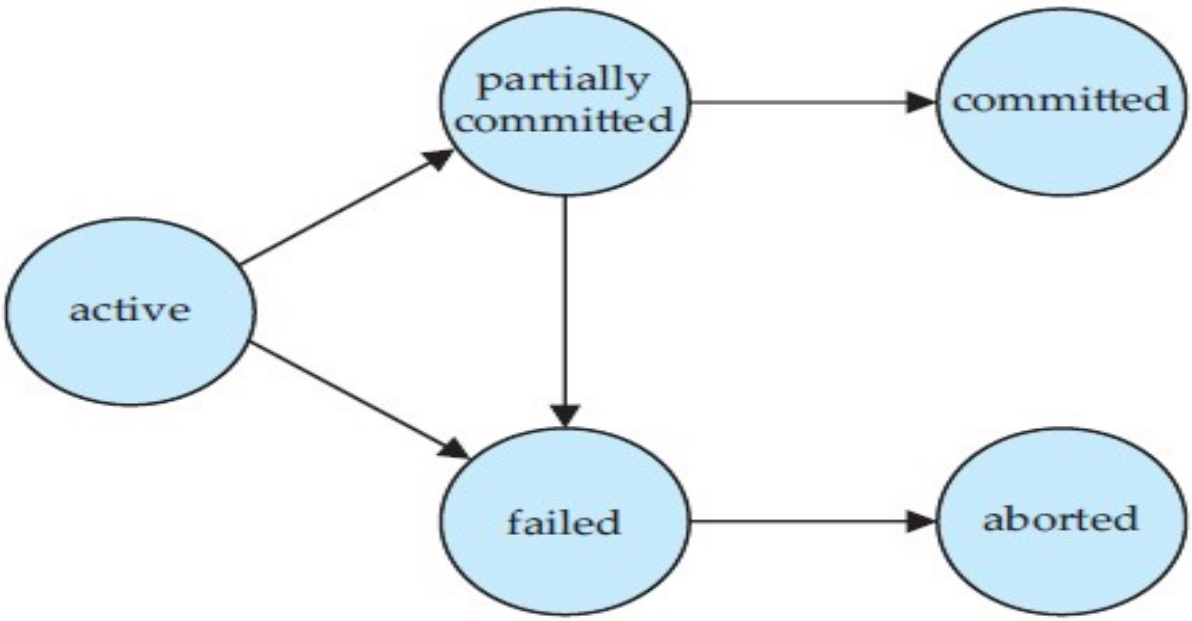
# Transações

- Apesar de ser um conceito simples, transações sofrem diversos problemas num ambiente concorrente e suscetível a falhas.
- Várias transações vão acessar os itens de dados ao mesmo tempo.
  - Preciso de um controle de concorrência.
- O SGBD pode falhar a qualquer momento.
  - O que fazer se a falha ocorrer no meio de uma transação?
  - Preciso de um controle de recuperação.

# Estado da transação

- Na ausência de falhas, todas as transações são completadas com sucesso.
- Porém, uma transação nem sempre pode completar sua execução.
- Nesse caso, ela é considerada abortada.
- Uma transação que completa sua execução é considerada confirmada (*commited*).

# Estado da transação



# Estado da transação

- Uma transação pode assumir cinco diferentes estados:
  - **Ativa**: Quando a transação é iniciada é definida como ativa e permanece nesse estado durante sua execução
  - **Falha**: No momento da falha a transação é entra no estado de falha. O gerenciador de recuperação deve decidir se a transação deve ser reiniciada ou morta. Mesmo que a transação seja reiniciada ela é interrompida e em seguida iniciada novamente.
  - **Abortada**: Após ser interrompida a transação passa para o estado de abortada.

# Estado da transação

- Uma transação pode assumir cinco diferentes estados:
  - **Confirmada**: Se a transação ocorrer sem problemas ela passa para o estado de confirmada.
  - **Parcialmente confirmada**: Porém, antes de passar para o estado de confirmada, a transação é definida como parcialmente confirmada. Ela só passa para o estado final após a gravação de todos os dados em disco. Esse estado é importante para o gerenciado de recuperação.

# Log do sistema

- Para recuperar-se de falhas o sistema mantém um log que registra as operações de todas as transações.
  - Um arquivo sequencial apenas para inserção.
- Entradas do log:
  - Start, write_item, read_item, commit, abort.
- Protocolos de recuperação se baseiam nas entradas do log.

# Confirmação de uma transação

- Um transação alcança seu ponto de confirmação:
  - Todas suas operações tiverem sido executados com sucesso.
  - Todas as operações da transação foram registradas no log.
    - Quando a entrada *commit* é gravada no log.

# Quando é necessária a recuperação?

- Quando uma transação é submetida ao SGDB, ele deve garantir que todas as operações devem ser completadas ou a transação não deve ter efeito nenhum.
- No primeiro caso dizemos que a operação é committed (confirmada) ou abortada.
- Na presença de falhas, o SGDB deve desfazer todas as operações de uma transação não confirmada.

# Tipos de falhas

- System crash
  - Problemas de hardware, software ou de rede que acontecem durante a execução da transação.

# Tipos de falhas

- Erro na transação ou no sistema
  - Alguma operação na  na transação causa a falha.
    - Divisão por zero, etc.
  - Podem ocorrer devido erros de lógica de  programação.
  - A transação ter sido abortada pelo usuário.

# Tipos de falhas

- Exceções identificadas pela transação
  - Algumas exceções podem ser identificadas pela transação e causa a interrupção da execução.
    - Ex: falta de saldo na conta corrente.
  - É uma exceção que causa a interrupção da transação mas não é considerada uma falha.

# Tipos de falhas

- Execução do controle de concorrência
  - Métodos de controle de concorrência podem decidir em abortar uma transação.
  - Utilizam essa decisão para garantir a concorrência.
    - Deadlock e starvation.

# Introdução

- Sistemas de processamento de transação
  - Grandes bancos de dados.
  - Vários usuários simultâneos que executam transações de banco de dados.

# Introdução

- Sistemas multiusuários são obtidos através do conceitos de multiprogramação.
- A execução simultânea é obtida através de um acesso intercalado.
- Um SGBD multiusuário é baseado no conceito de **concorrência intercalada**.
  - Se duas ou mais transações quiserem acessar um mesmo item de dados, cada uma terá a sua vez para acessá-lo.

# Transações

- Coleção de operadores que forma uma **única unidade lógica** de trabalho.
- Conjunto de operações de manipulação de dados que executam uma única tarefa.
- Essa operações incluem inserções, remoções, atualizações ou recuperações de consultas.
- Todas as operações devem ser feitas ou

  nenhuma deve ser feita.

# Por que é necessário um controle de concorrência?

- Alguns problemas podem ocorrer:
  - O sistema falhar no meio da transação antes que todas as operações tenham sido efetuadas.
  - Outra transação pode ocorrer de forma simultânea e acessar os mesmos itens de dados.

# Por que é necessário um controle de concorrência?

- Não teremos problema se somente uma operação for executada por vez.
- Porém, uma transação é uma coleção de operadores que forma uma única unidade lógica de trabalho

# Introdução

- O que consiste uma transferência bancária?

# Introdução

- O que consiste uma transferência bancária?
  - Uma certa quantia de dinheiro é movida de uma conta para outra.
  - Diversas operações (Transferir N da conta X para Y):
    - Ler o saldo de X (SELECT);
    - Retirar o valor N de X;
    - Atualizar o valor de X (UPDATE);
    - Ler o saldo de Y (SELECT);
    - Soma o valor N em Y.
    - Atualizar o valor de Y (UPDATE);

# Introdução

- O que consiste uma transferência bancária?
  - A operação de escrita em X não pode ser desassociada da operação de escrita em Y
  - Caso contrário,o dinheiro sairá de uma conta e não entrará na outra

# Propriedades das transações

- Sistemas de gerenciamento de banco de dados garantem suporte a transações.
- Para garantir esse suporte é necessário um conjunto de características (ACID).
  - **A**tomicidade.
  - **C**onsistência.
  - **I**solamento.
  - **D**urabilidade

# Propriedades das transações

- Consistência
  - Devem preservar o estado consistente do banco de dados.
  - Se o banco de dados estiver consistente antes da transação, deve permanecer consistente após a execução.

# Propriedades das transações

- Atomicidade
  - Todas as operações da transação devem ocorrer de forma correta ou nenhuma operação deve ocorrer.
  - Exemplo:
    - Se ocorrer uma falha após a operação write_item(A), o banco de dados se encontra num estado consistente.
  - Para garantir atomicidade, o banco de dados deve reverter as operações executadas até então, pois nem todas as operações foram efetuadas de forma correta.

# Propriedades das transações

- Isolamento
  - Num SGBD, várias transações podem ocorrer de forma simultânea (acesso concorrente).
  - Essas transações pode efetuar operações em mesmos itens de dados.
  - Solução simples é executar as transações de forma serial.
    - Uma após a outra.
  - Porém,
    - Subutiliza o processamento
    - Se torna um gargalo de acesso ao banco.

# Propriedades das transações

- Isolamento
  - O isolamento garante que dado um par de transações ti e tj executando em paralelo:
    - ti não está ciente de tj e vice-versa.
  - Embora executando em paralelo se tem o efeito que as transações estão executando de forma sequencial.

# Propriedades das transações

- Durabilidade
  - Quando a transação for efetuada com sucesso é preciso garantir a persistência dos dados mesmo na presença de falhas.
  - Uma vez consolidada a transação, suas alterações permanecem no banco até que outras transações aconteçam.

# Transações

- Vamos estudar problemas e soluções de transações que são executadas de forma concorrente.
- Por simplicidade, vamos considerar que nosso banco de dados contenha uma coleção de itens de dados.
- Vamos identificar um item de dados por um rótulo: X, Y, etc.
- Um item de dados pode representar um conjunto registro, tuplas, uma tabela, etc.

# Transações

- Vamos definir duas operações efetuadas em transações:
  - read_item(x) - item de dados X é lido.
  - write_item(x) - item de dados X é escrito de volta no banco de dados.
- A leitura corresponde trazer o item de dados para a memória principal.
- A escrita corresponde salvar o item de dados da memória em disco.

# Transações

- Exemplo: Transferências entre contas

```
read_item(X);
X = X – N;
write_item(X);
read_item(Y);
Y = Y + N;
write_item(Y);
```

# Transações

- Exemplo: Depósito em um conta

```
read_item(X);
X = X + N;
write_item(X);
```

# Transações

- Exemplo de duas transações sequenciais

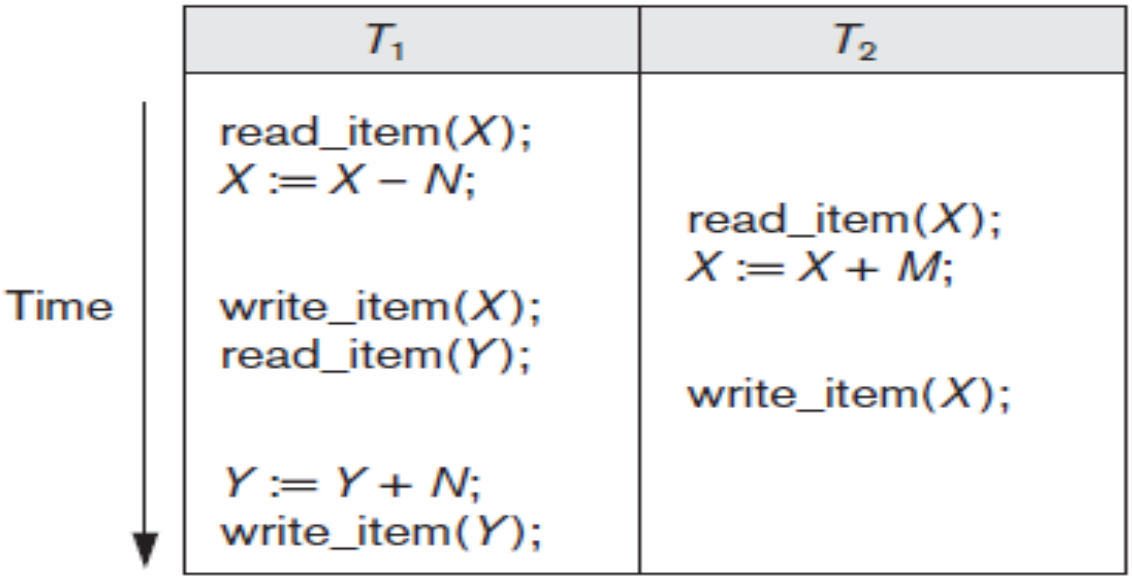
  X = 50, Y = 30, N = 10, M = 20
- Transferir N = 10 de X para Y
  - Valor final: X = 40, Y = 40
- Em seguida, depositar M = 20 em X
  - Valor final: x = 60, Y = 40

# Problemas

- Atualização perdida
  - Duas transações que acessam o mesmo item de dados tem suas operações intercaladas em uma maneira que o valor de alguns itens fiquem incorretos.

# Problemas

- Atualização perdida




Seja X = 50, Y = 30, N = 10 e M= 20

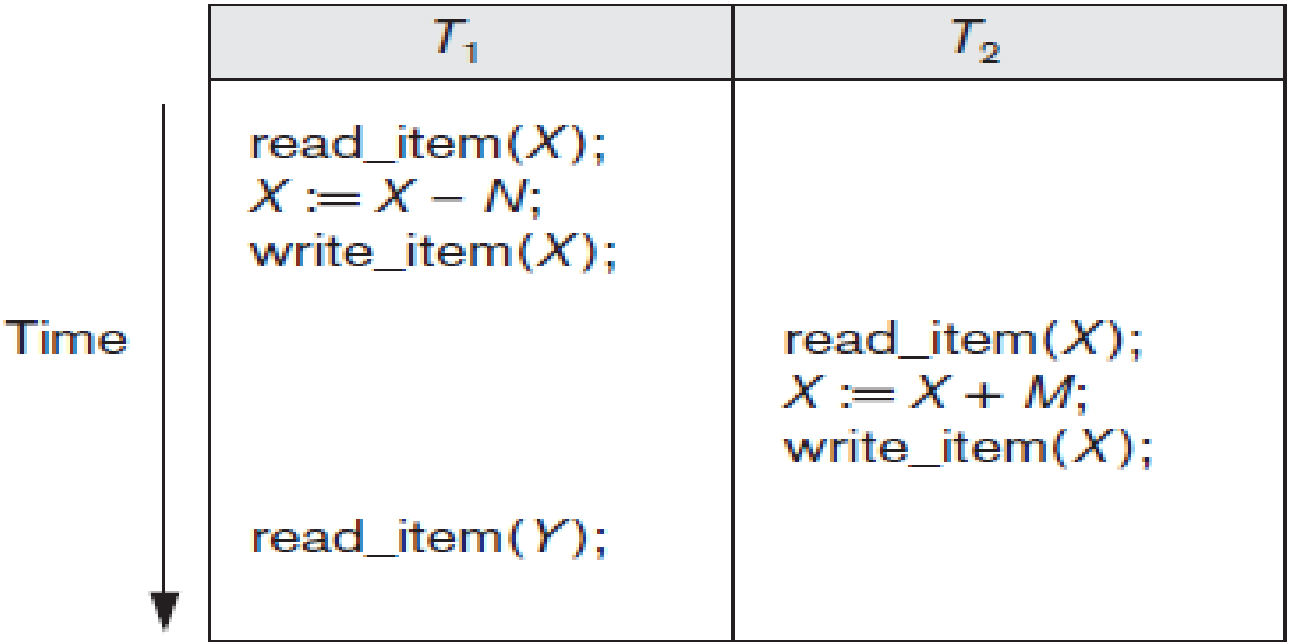
T1
read_item(X) - 50
X = 50 – 10 = 40
T2
read_item(X) - 50
X = 50 + 20 = 70
T1
write_item(X) - 40
read_item(Y) - 30
T2
write_item(X) - 70
T1
Y = 30 + 10 = 40
write_item(Y) – 40
Valor final: X = 70, Y = 40

# Problemas

- Atualização temporária (Dirty read)
  - Ocorre quando duas transações acessam um mesmo item de dados e por alguma motivo uma transação falha.
  - Assim, a transação que falhou é abortada e suas operações são canceladas.
  - Porém, a outra transação utiliza os valores modificados pela transação que falhou.

# Problemas

- Atualização temporária




T1 atualiza o valor de X e escreve o valor no banco.

Em seguida, T2 lê o valor de X atualizado por T1.

Porém, considere que item Y não está acessível, logo há uma falha em T1, que deve ser cancelada.

T2 terminou a execução com um modificado por uma transação abortada.

# Problemas

- Leitura não repetível
  - Ocorre quando uma transação t1 lê um item de dados duas vezes, porém no intervalo da leitura o item é alterado por uma outra transação t2.
  - Assim, t1 recebe valores diferentes do item de dados.

# Problemas

- Leitura fantasma (Phantom read)
  - Ocorre quando uma transação executa uma consulta e recebe um conjunto de dados.
    - Esse conjunto satisfaz as condições da consulta.
  - Num segundo momento, na mesma transação, a consulta é executada e um conjunto de dados diferentes é retornado.

# Transações no PostgresSQL

- O postgresql específica os comandos SQL Begin e Commit para definir uma transação.

```
BEGIN
   UPDATE departamento SET orcamenteo = orcamento +1000
WHERE cod_depto = 'DCOMP'
```

# Transações no PostgresSQL

- Se por algum motivo não quisermos dar o COMMIT na transação, podemos utilizar o comando ROLLBACK.

# Aplicações com Transações

- O postgresql não permite escrevermos transações dentro de stored procedures.
- Por padrão, cada comando SQL efetua um commit.
  - Auto-commit.

# Aplicações com Transações

- Porém, podemos utilizar a biblioteca JDBC para efetuar transações.
- Podemos desabilitar a opção de auto-commit e efetuar as operações de commit e rollback indivudalmente.

# Transações - JDBC

- Por default, todos os comandos do JDBC são auto-commit
  - Um comando é uma transação.
- Porém, podemos desabilitar essa opção e executar uma sequência de comandos como uma transação.
  - Definidos pelo comando commit();
- Pode também, executar um rollback explicitamente.
  - Comando rollback().

# Transações - JDBC

```
public void exemploTransacao() throws SQLException{
    conexao.setAutoCommit(false);
    System.out.println("Início da transação");
    Statement comando = conexao.createStatement();
    String sql1 = "INSERT INTO professor VALUES('1300',
'Prof J', 'DMAT', 4500)";
    System.out.println("Inserindo: " + sql1);
    comando.executeUpdate(sql1);
```

# Transações - JDBC

```
    String sql2 = "SELECT nome FROM professor WHERE departamento = 'DMAT' ";
    System.out.println("Consulta: " + sql2);
    ResultSet resultado = comando.executeQuery(sql2);
    while(resultado.next()){
        String nome = resultado.getString("nome");
        System.out.println(nome);
    }
    String sql3 = "DELETE FROM professor WHERE mat_professor = '1300'";
    System.out.println("Remoção: " + sql3);
    comando.executeUpdate(sql3);
    conexao.commit();
}
```

# Postgresql – Níveis de isolamento

- Estudamos que para garantir o isolamento um SGBD deve gerar somente schedules serializáveis.
- Porém, quanto maior for o nível de isolamento menor é a concorrência.
  - Dependendo do protocolo ainda há a possibilidade de problemas como deadlock e starvation.

# Postgresql – Níveis de isolamento

- O postgresql permite a possibilidade de definirmos níveis de isolamento.
  - Ganhamos em concorrência, porém não garantimos mais as propriedades de uma transação.
  - Definidos pelo padrão SQL.

# Postgresql – Níveis de isolamento

- Sem isolamento, alguns problemas podem ocorrer:
  - Phantom read
  - Atualização temporária (dirty read)
  - Leitura não-repetível
- Os níveis de isolamento são baseados nesses problemas.

# Postgresql – Níveis de isolamento

| Isolation Level | Dirty Read | Nonrepeatable Read | Phantom Read |
|---|---|---|---|
| Read uncommitted | Possible | Possible | Possible |
| Read committed | Not possible | Possible | Possible |
| Repeatable read | Not possible | Not possible | Possible |
| Serializable | Not possible | Not possible | Not possible |

# Postgresql – Níveis de isolamento

- O nível padrão do postgressql é o Read Committed.
- Além disso,é possível obter o bloqueio explícito de itens de dados.
  - Porém, pode causar deadlock.

# Leitura obrigatória

ELMASRI, R; NAVATHE, S.B. **Sistemas de Banco de Dados**, Addison Wesley, 6ª Edição.

- Capítulo 21

Silberschatz, A; Korth H.F.; Sudarshan S. **Sistemas de Banco de Dados**, Editora Campus, 6ª Edição.

- Capítulo 14