

# TRANSAÇÕES NÍVEIS DE ISOLAMENTO

---

# Níveis de Isolamento

- Permitir níveis de isolamento fracos significa que pode-se gerar schedules que não são serializáveis
  - Ou seja, onde pode-se acabar lendo informações inconsistentes
- Algumas aplicações aceitam viver com níveis fracos de isolamento
  - Ex. Uma transação read-only (apenas leitura) que precisa de um total aproximado do saldo de todas as contas
- Tradeoff de precisão por desempenho

# Níveis de Isolamento no SQL-92

- Existem quatro níveis (do mais fraco ao mais forte)
  - Read uncommitted
  - Read committed
  - Repeatable read (padrão)
  - Serializable
- O isolamento garantido por um nível também é garantido pelo nível acima

# Níveis de Isolamento no SQL-92

- **Read uncommitted**

- Até mesmo registros não comitados podem ser lidos.

- **Read committed**

- Apenas registros comitados podem ser lidos
  - Leituras repetidas de um mesmo registro podem retornar valores diferentes (mas ambos previamente comitados).

- **Repeatable read**

- apenas registros comitados podem ser lidos
  - Leituras repetidas de um mesmo registro devem retornar o mesmo valor
  - No entanto, leituras repetidas podem trazer conjuntos diferentes de registros

- **Serializable**

- Nível mais forte de isolamento (padrão)

# Read Uncommitted

- A transação que quer ler o item de dados não espera que locks exclusivos sejam liberados
- Assim, registros não comitados por outras transações podem ser lidos.
- Isso pode levar a **leituras sujas**
  - **dirty reads**

# Read Uncommitted

- T1 leu um valor de idade gerado por T2
- Mas T2 abortou
- Ou seja, o valor lido está errado
- Isso é um Dirty Read

T1	T2
<b>SELECT</b> idade <b>FROM</b> func <b>WHERE</b> id = 1; <i>Commit</i>	<b>UPDATE</b> func <b>SET</b> idade = 21 <b>WHERE</b> id = 1;  <i>Abort</i>

A idade inicial do funcionário de id=1 é 20

# Read Committed

- A transação que quer ler o item de dados **deve** esperar que locks exclusivos sejam liberados
  - Dessa forma, apenas registros comitados podem ser lidos
- No entanto, a transação que adquiriu um lock compartilhado para leitura libera o lock assim que concluir a leitura
  - Dessa forma, leituras repetidas de um mesmo registro podem retornar valores diferentes
  - mas ambos previamente comitados
- Esse efeito é chamado de **leitura não repetida**
  - non-repeatable read

# Read Committed

- T1 leu dois valores válidos de idade
  - O primeiro antes da execução de T2
  - O segundo após a execução de T2
- As duas leituras geraram valores diferentes
- Isso é um **non-repeatable read**

T1	T2
<b>SELECT</b> idade <b>FROM</b> func <b>WHERE</b> id = 1;	
<b>SELECT</b> idade <b>FROM</b> func <b>WHERE</b> id = 1; <b>Commit</b>	<b>UPDATE</b> func <b>SET</b> idade = 21 <b>WHERE</b> id = 1; <b>Commit</b>

A idade inicial do funcionário de id=1 é 20

# Repeatable Read

- A transação que adquiriu um lock compartilhado para leitura segura o lock até o final da execução
  - Dessa forma, leituras repetidas de um mesmo registro devem retornar o mesmo valor
- No entanto, esse nível não prevê locks de intervalo (range locks)
  - Dessa forma, pode acontecer de leituras repetidas trazerem conjuntos diferentes de registros, se outra transação gerou registros novos.
- Esse efeito é chamado de **leitura fantasma**
  - **Phantom read**

# Repeatable Read

- T1 leu dois conjuntos de registros de func com o mesmo filtro de intervalo
  - O primeiro não possui ‘Bob’
  - O segundo possui ‘Bob’
- A aparição de ‘Bob’ é chamada de **phantom read**

T1	T2
<pre><b>SELECT</b> nome <b>FROM</b> func <b>WHERE</b> idade <b>BETWEEN</b> 10 <b>AND</b> 30;</pre>  <pre><b>SELECT</b> nome <b>FROM</b> func <b>WHERE</b> idade <b>BETWEEN</b> 10 <b>AND</b> 30; <b>Commit</b></pre>	<pre><b>INSERT INTO FUNC</b> (id, nome,idade) <b>VALUES</b> (3, ‘Bob’, 27) <b>Commit</b></pre>

# Serializable

- O nível mais forte de isolamento
- Simplesmente permite com que apenas uma transação por vez execute
- Isso força a geração de schedules seriais

# Definição de Transação no SQL

- Linguagens de manipulação de dados devem incluir um construtor para delimitar as instruções contidas em uma transação.
- Em SQL
  - uma transação é iniciada de forma implícita.
  - O término é indicado por :
  - **Commit:** encerra transação atual e inicia uma nova.
  - **Rollback:** leva ao término mal sucedido da transação.
- Em quase todos os SGBDs, por padrão, toda instrução isolada é comitada
  - Esse commit implícito pode ser desativado usando alguma diretiva
    - Ex. em JDBC, `connection.setAutoCommit(false);`