Transações em bases de dados

Bases de Dados (CC2005)
Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto

Eduardo R. B. Marques — DCC/FCUP

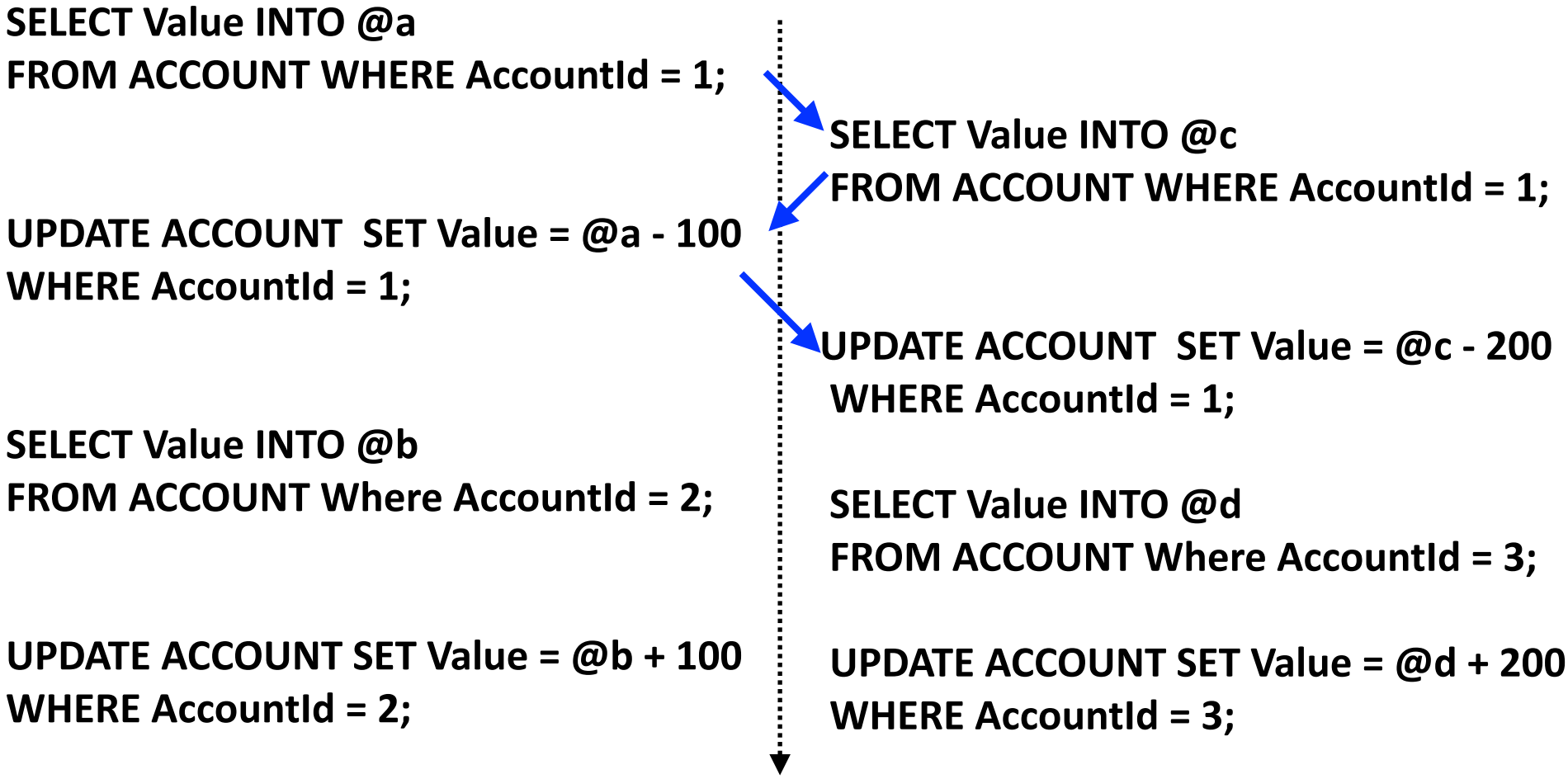
Introdução

Motivação

```
SELECT Value INTO @v
FROM ACCOUNT WHERE AccountId = 1;

UPDATE ACCOUNT  SET Value = @v - 100
WHERE AccountId = 1;

SELECT Value INTO @v2
FROM ACCOUNT Where AccountId = 2;

UPDATE ACCOUNT SET Value = @v2 + 100
WHERE AccountId = 2;
```

- Exemplo simplista de transferências entre contas bancárias: montante de **100** debitado da conta **1** e creditado na conta **2**. Se ligação à BD for perdida ou a BD "crashar" após o primeiro **UPDATE**, o dinheiro "desaparece", i..e, não chega a ser creditado na conta **2**.
- É conveniente que as 2 actualizações ou **tenham ambas efeito ou então nenhuma.**

Motivação (cont.)



- Transferências simultâneas sobre a mesma conta podem dar valores errados. No exemplo, conta **1** poderia ser debitada em apenas **200** euros em vez de **300**!
- As transferências não executam "**isoladas**" uma da outra. Intuitivamente, seria desejável que o efeito lógico conjunto fosse equivalente a uma **sequência** de transferências.

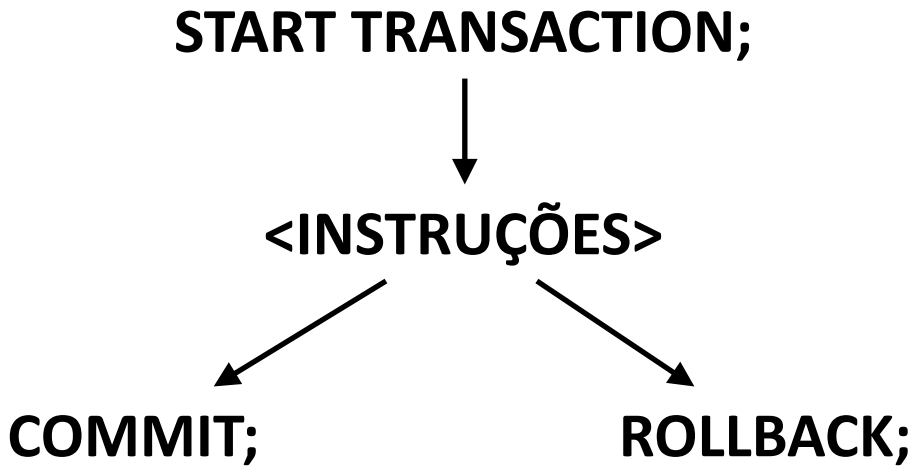
Motivação (cont.)

- O exemplo anterior é uma instância do caso geral (e normal) de acessos concorrentes a uma BD.
- Um SGBD deve estar preparado para lidar com **concorrência** sobre uma BD, isto é, várias sessões simultâneas e independentes **Sessão 1 | ... | Sessão n** a executar operações sobre a BD.

Noção de transação / propriedades ACID

- **Transação** : sequência de operações de uma sessão.
- Propriedades desejáveis (**ACID**) para uma transação
  - **Atomicidade**: uma transação tem um efeito atómico; as suas operações **ou** completam todas num único momento logicamente "instantâneo" **ou** não têm qualquer efeito.
  - **Consistência**: uma transação deixa o estado da BD num estado consistente, respeitando as restrições de integridade.
  - **Isolamento**: uma transação não é afectada por transações concorrentes.
  - **Durabilidade:** o estado da BD resultante da transação é persistente após esta ter terminado.

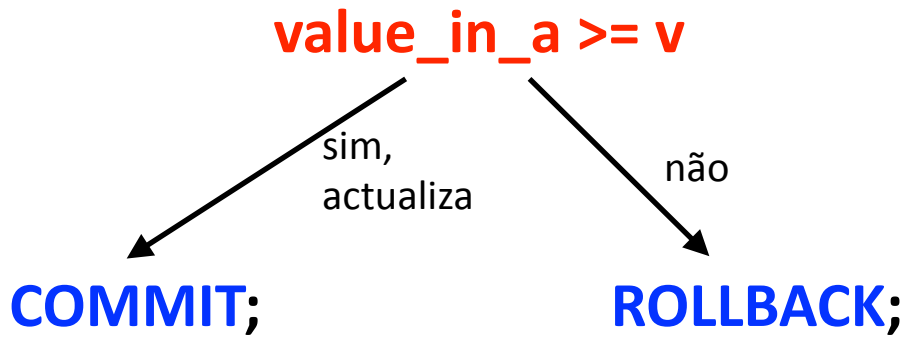
Definição de transações em SQL



- **START TRANSACTION:** inicia uma transação.
- **COMMIT:** indica término da transação, especificando que os efeitos da mesma se tornem **persistentes**.
- **ROLLBACK :** indica término da transação, especificando que os efeitos da mesma sejam **desfeitos**.

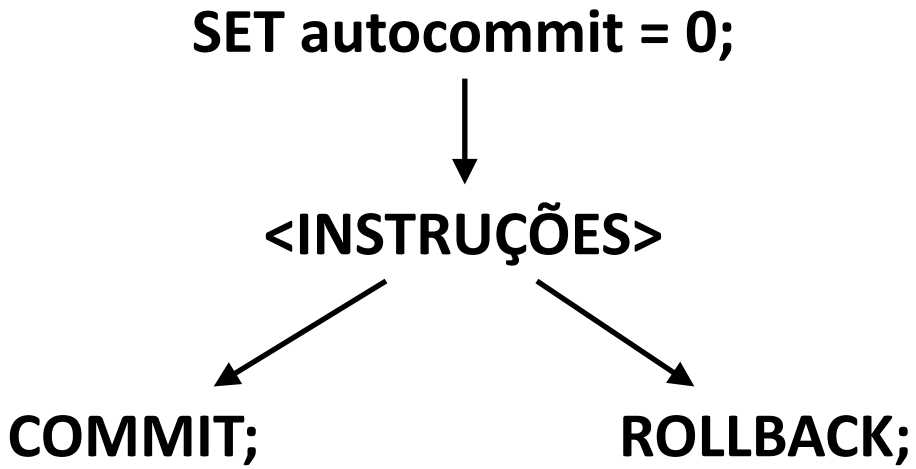
Exemplo

```
CREATE PROCEDURE transfer(IN a INT, IN b INT, IN v INT)
BEGIN
   DECLARE value_in_a INT;
   START TRANSACTION;
   SELECT Value INTO value_in_a
   FROM ACCOUNT WHERE AccountId = a;
   IF value_in_a >= v THEN
    UPDATE ACCOUNT  SET Value = Value - v
    WHERE AccountId = a;
    UPDATE ACCOUNT SET Value = Value + v
    WHERE AccountId = b;
    COMMIT;
   ELSE
     ROLLBACK;
    END IF;
END$
```



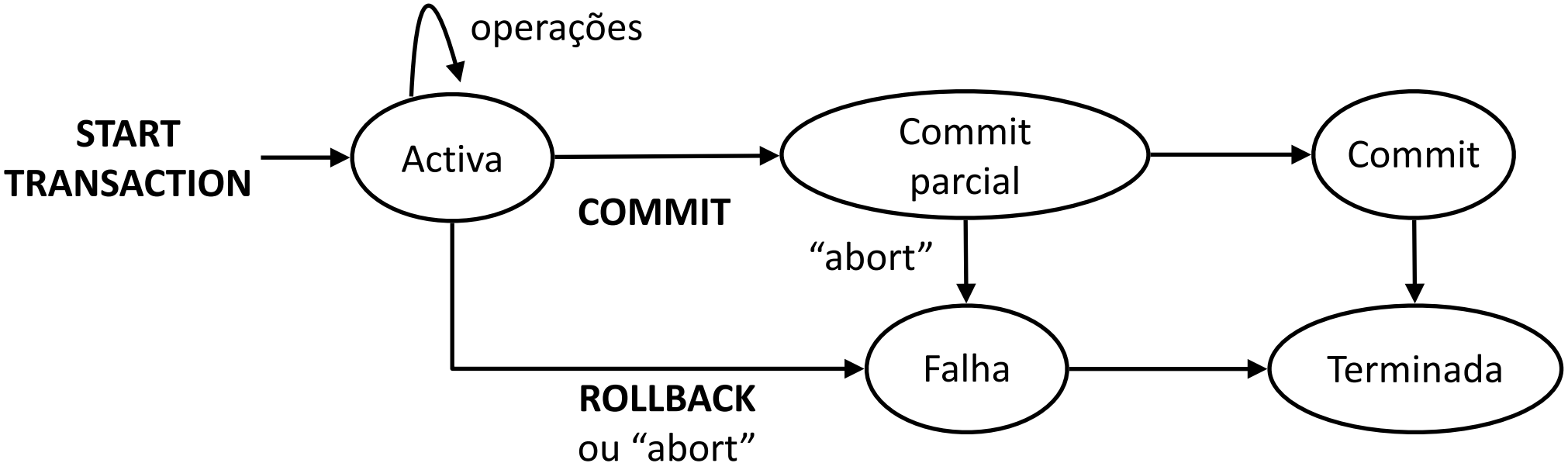
- Garantimos que **transfer(a,b,v)** só sucede se o saldo de **a** for igual ou superior a **v**.
- E também que em caso de erro interno à BD durante a execução, por exemplo uma perda de ligação após o primeiro UPDATE, que a transação é completamente desfeita.

Definição transações em SQL (cont.)



- Tipicamente uma sessão sobre um SGBD SQL tem inicialmente a parameterização de **"auto-commit" habilitada**. Em MySQL podemos desabilitar o "auto-commit" atribuindo à variável de sistema **autocommit** o valor **0.**
  - **autocommit = 1**: a cada instrução SQL corresponderá implicitamente uma transação que é automaticamente "commited".
  - **autocommit = 0**: sequência de instruções formam implicitamente uma transação que deverá ser terminada com **COMMIT** ou **ROLLBACK**.
  - Independentemente do valor de **autocommit**, podemos sempre iniciar uma transação explicitamente com **START TRANSACTION** (o valor de "auto-commit" é reposto após o término da transação).

Estados de uma transação



- **Activa**: até uma instrução explícita de **COMMIT** ou **ROLLBACK,** ou **"abort"** pelo SGDB (ex. por via de um erro interno do SGBD ou incorreção/erro numa operação da transação).
- **Commit parcial / Commit:** após **COMMIT**, no estado de **"commit parcial"**, o SGBD tem de validar se a transação pode ter efeito persistente na BD. Se assim fôr, a transação passa ao estado de **"commit"** propriamente dito, em que o SGBD se compromete a tornar permanentes as alterações. Caso contrário, a transação é abortada.
- **Falha**: após **ROLLBACK** ou "abort" da transação, os efeitos da transação têm de ser desfeitos.

Mecanismos para garantias de propriedades ACID

Propriedades	Mecanismos
Consistência	Consistência resulta das **restrições de integridade** definidas no esquema da BD (ex. chaves primárias e externas, tipos de dados, triggers de validação, etc), e da **boa programação de aplicações** para "regras de negócio" não reflectidas no esquema**.**
Isolamento	Arbitragem de concorrência é norteada pela propriedade lógica de **serialização**. SGDBs tipicamente empregam algoritmos baseados em **"two-phase locking"** e suportam diferentes **níveis de isolamento,** correspondendo transações serializáveis ao nível mais alto/desejável de isolamento.
Atomicidade **Durabilidade**	Mecanismos de recuperação de falhas, tipicamente baseados na manutenção e processamento do **log de transações**.

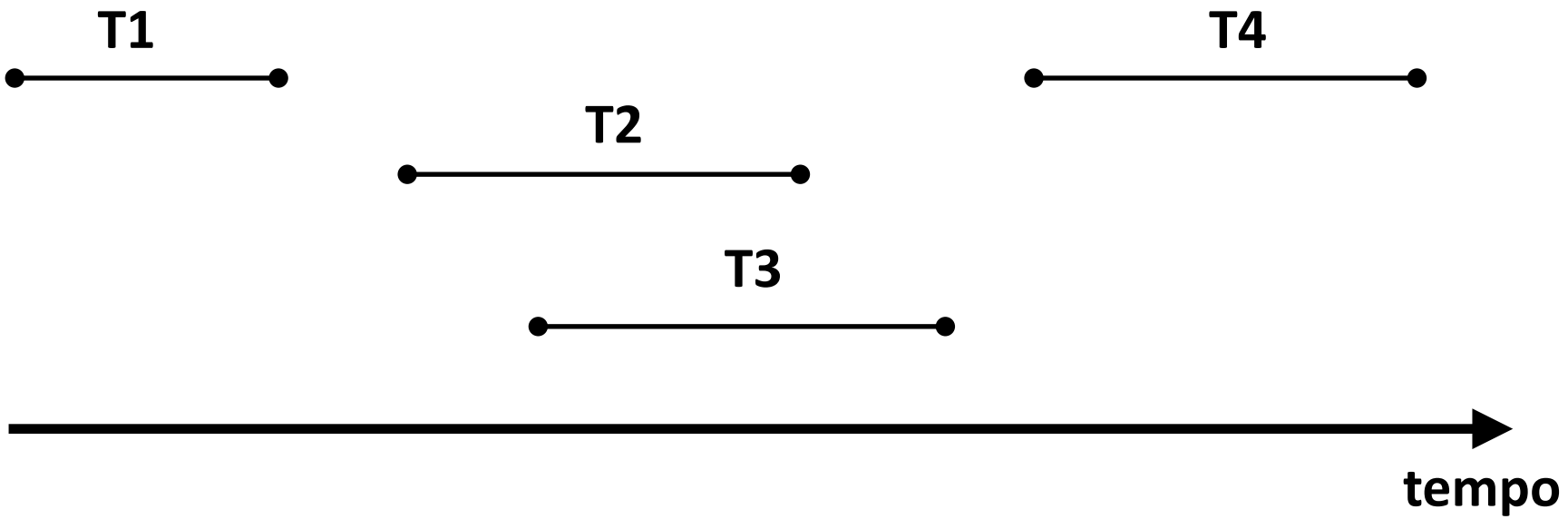
Isolamento / concorrência entre transações

Operações ao nível do SGBD

- Para raciocinar sobre transações, para além de **START TRANSACTION, COMMIT** e **ROLLBACK,** um SGDB tem em conta as operações de leituras e escrita induzidas por instruções SQL.
  - **read(R)**: lê um registo **R**
  - **write(R,V)**: escreve **V** para o registo **R**
  - **Obs.:** remoção pode ser modelada como uma "escrita especial"

SQL	Operações no SGBD
UPDATE ACCOUNT **SET Value = Value + 100** **WHERE AccountId = 1234;**	**X = read(ACCOUNT[1234]);** **X.Value = X.Value + 100;** **write(ACCOUNT[1234], X);**

Isolamento e serialização



- O isolamento de transações é formulada em termos de uma propriedade lógica chamada **serialização.**
- Uma execução concorrente de transações diz-se **serializável** se o seu efeito for equivalente uma execução sequencial (também dita serial) das mesmas transações.
- No exemplo: **T1** precede **T2** e **T3**, que por sua vez precedem **T4**. A execução deverá ser equivalente a uma das seguintes execuções sequenciais: **1) T1, T2, T3, T4** ou **2) T1, T3, T2, T4.**

Serialização (cont.)

T1: UPDATE ACCOUNT
SET Value = 100
WHERE AccountId = 1234;

T2: UPDATE ACCOUNT
SET Value = Value + 200
WHERE AccountId = 1234;

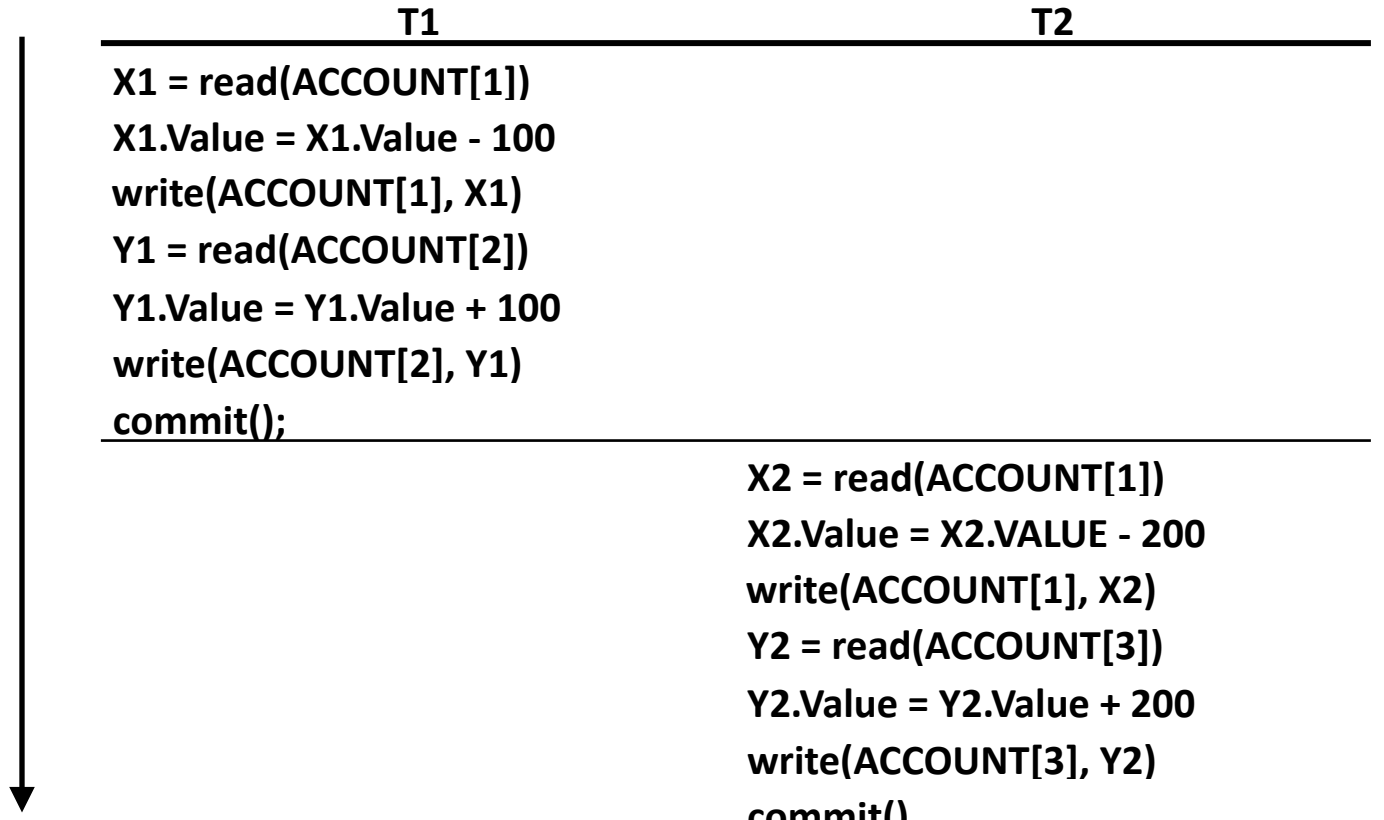
- Suponha que **T1** e **T2** executam concorrentemente a partir de um estado em que **ACCOUNT.Value = 0** para o registo **1234.**
- Uma execução serializável deverá ser equivalente a **T1** seguida de **T2**, ou **T2** seguida de **T1,** não significando isso o mesmo estado para a BD: supondo que o valor na conta é **0** antes de **T1** e **T2**, ficamos com **300** para **T1** seguida por **T2** e **100** para **T2** seguida por **T1**.

Serialização (cont.)

- Um **escalonamento sequencial** de transações concorrentes corresponde a executar as operações de cada transação em sequência do princípio ao fim para cada transação.
  - ... uma transação activa a cada instante — Concorrência zero => grande impacto no desempenho!
  - Mas define a base para raciocinarmos sobre serialização ...
- Em um **escalonamento não-sequencial,** a ordem de operações por cada transação é preservada, mas operações de diferentes transações podem ser intercaladas.
- Um escalonamento **serializável** é um escalonamento com efeitos sobre a BD equivalentes a algum escalonamento sequencial.

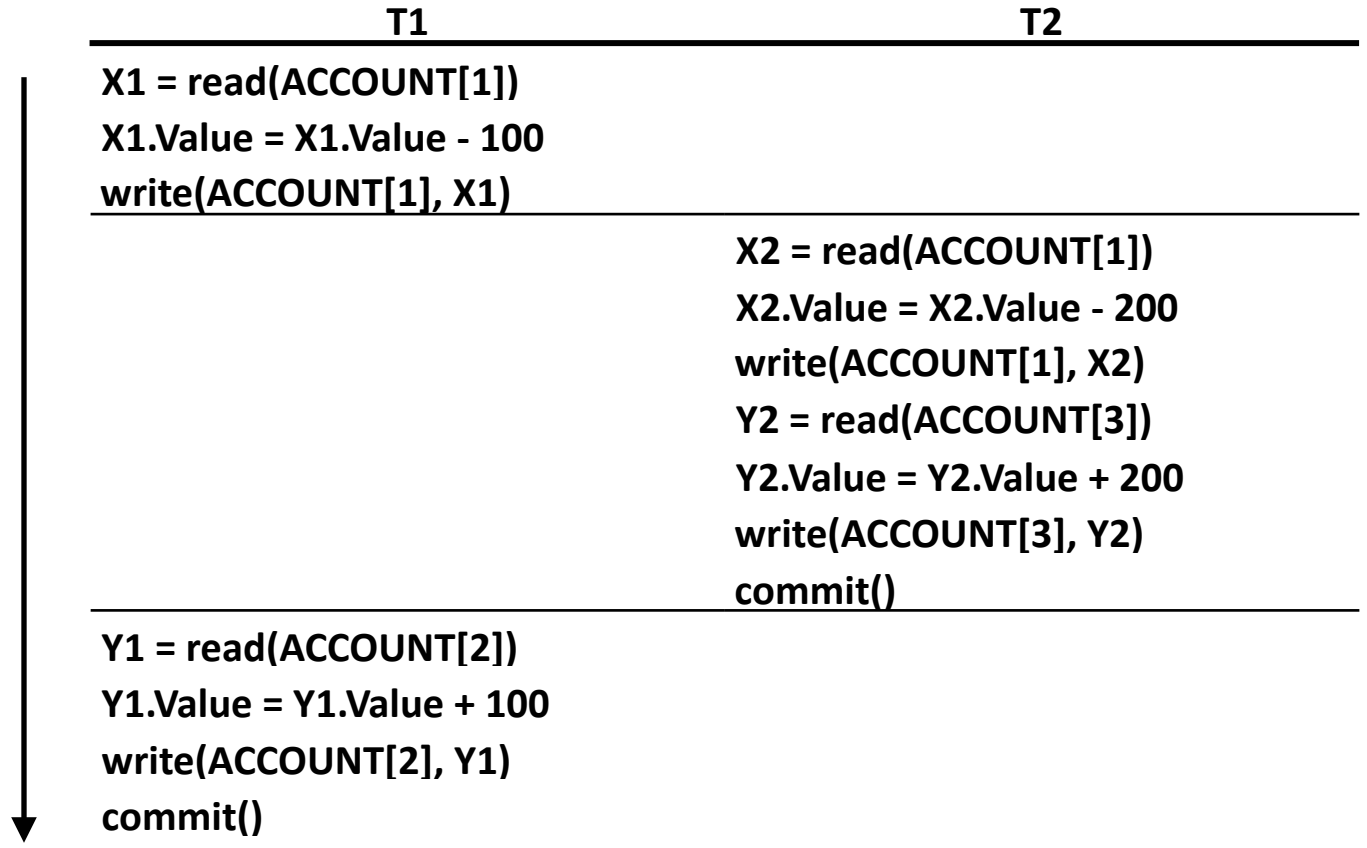
Escalonamento sequencial — exemplo

```
T1:  UPDATE ACCOUNT  SET Value = Value - 100 WHERE AccountId = 1;
     UPDATE ACCOUNT  SET Value = Value + 100 WHERE AccountId = 2;

T2:  UPDATE ACCOUNT  SET Value = Value - 200 WHERE AccountId = 1;
     UPDATE ACCOUNT  SET Value = Value + 200 WHERE AccountId = 3;
```



Escalonamento não-sequencial e serializável

(T1 e T2 como no slide anterior)



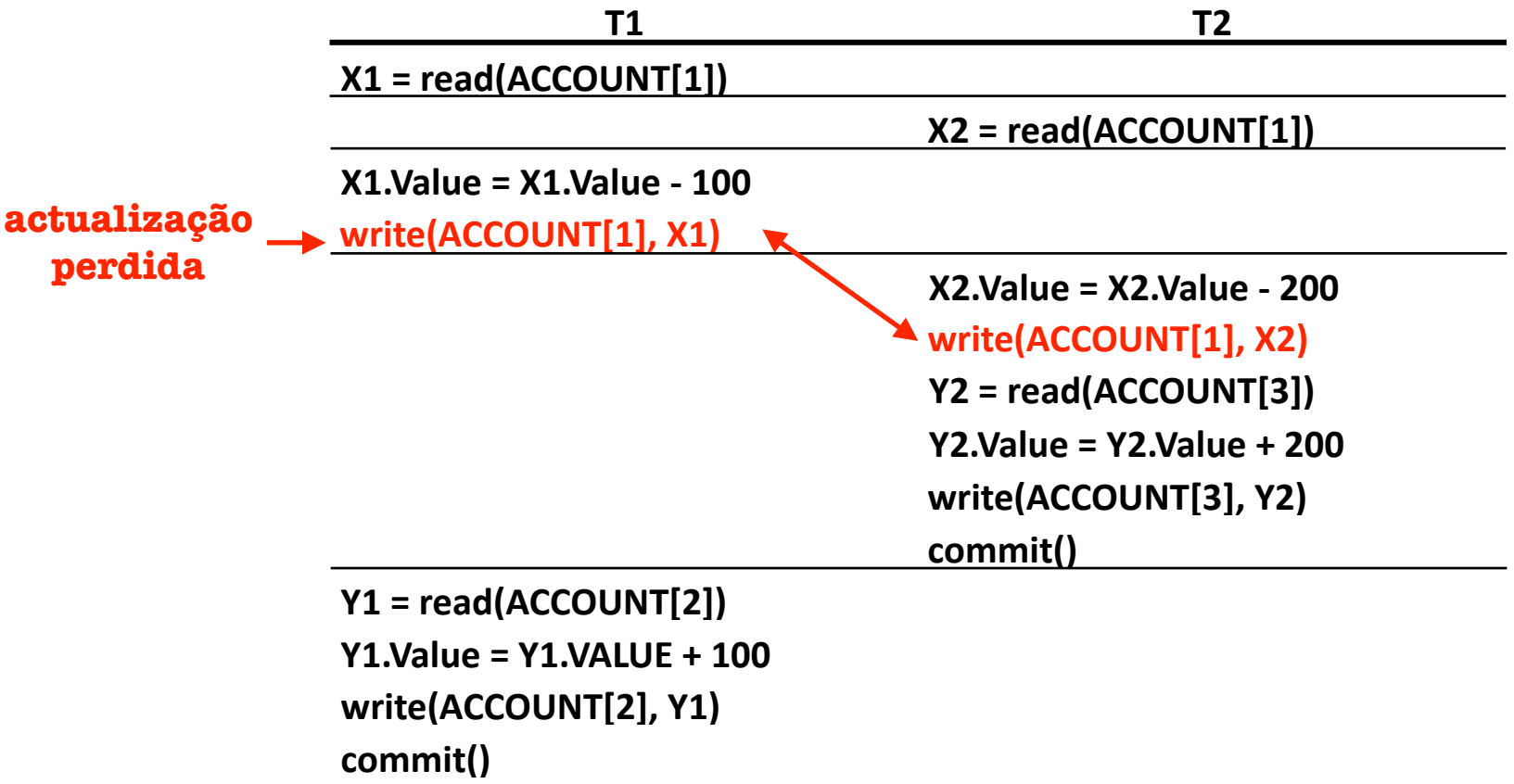
Execução não-sequencial é equivalente à execução sequencial ilustrada no slide anterior — o estado final na BD é o mesmo — portanto **serializável**.

Escalonamento não-serializáveis / anomalias comuns

- Escalonamentos não-serializáveis de transações levem a algumas classes de anomalias comuns tratadas (ou não) por um SGDB.
- **Actualizações perdidas ("lost updates")**
  - Escritas concorrentes por transações levam à "perda" de escritas.
- **Leituras sujas ("dirty reads"):**
  - Leitura de dados resultantes de escritas ainda não "commited" por outra transação.
- **Leituras irrepetíveis ("unrepeatable reads")**
  - Leituras de dados com valores diferentes pela mesma transação resultantes de escritas intervenientes por outra transação
- **Tuplos fantasma ("phantom tuples")**
  - Retorno de conjuntos de tuplos diferentes pela mesma transação resultante de de escritas intervenientes por outra transação

Actualizações perdidas — exemplo

(T1 e T2 como anteriormente)

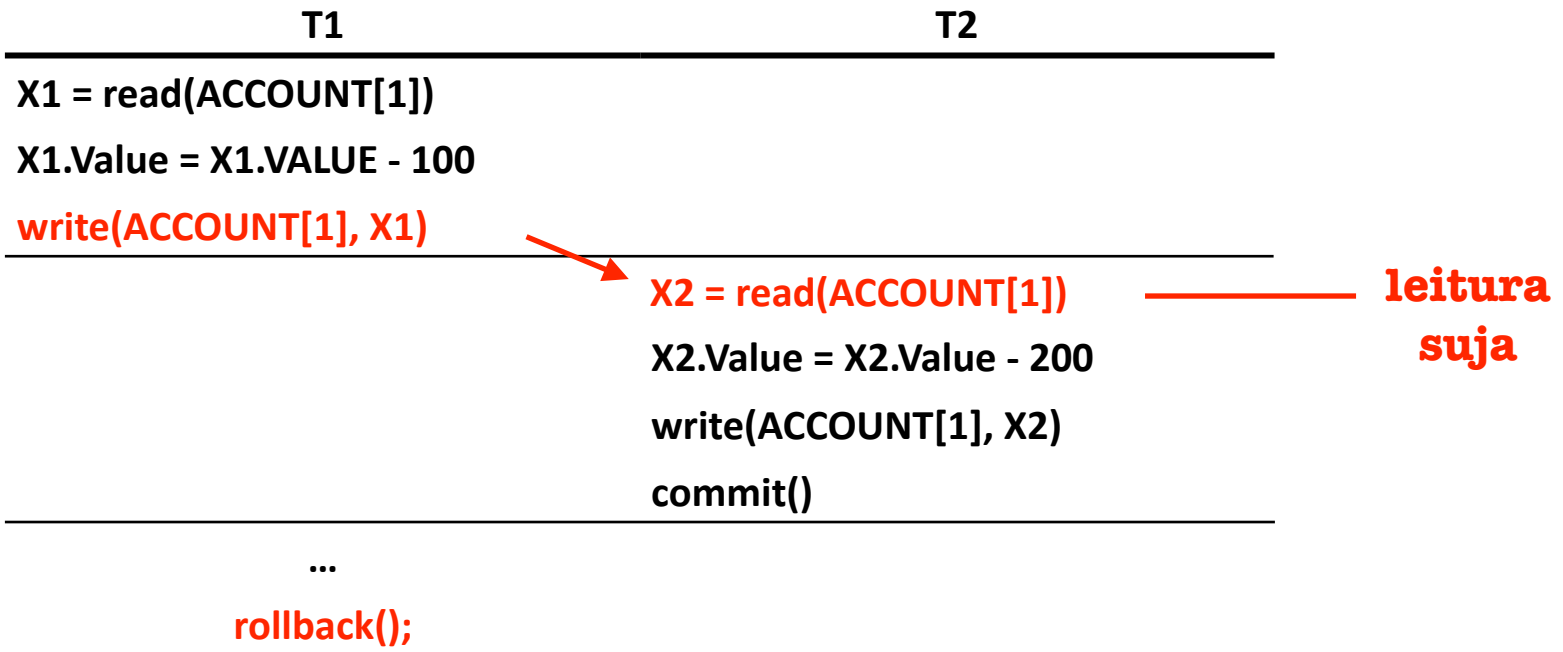


Conflito entre operações de escrita leva a que apenas uma escrita tenha na prática efeito. No exemplo: o valor da conta **1** é apenas debitado no valor de **200** (escrita de **T2**) em vez de **100 + 200 = 300**.

Leituras sujas

T1: UPDATE ACCOUNT SET Value = Value - 100 WHERE AccountId = 1;
… ROLLBACK; // falha

T2: UPDATE ACCOUNT SET Value = Value - 200 WHERE AccountId = 1;



Conflito entre uma operação de escrita "uncommited" e uma leitura subsequente leva a inconsistência dos dados. No exemplo: **T2** é "commited", o valor deveria ser decrementado apenas em **200** embora porque **T1** falha, a leitura suja por **T2** pode levar a um decremento de **300**.

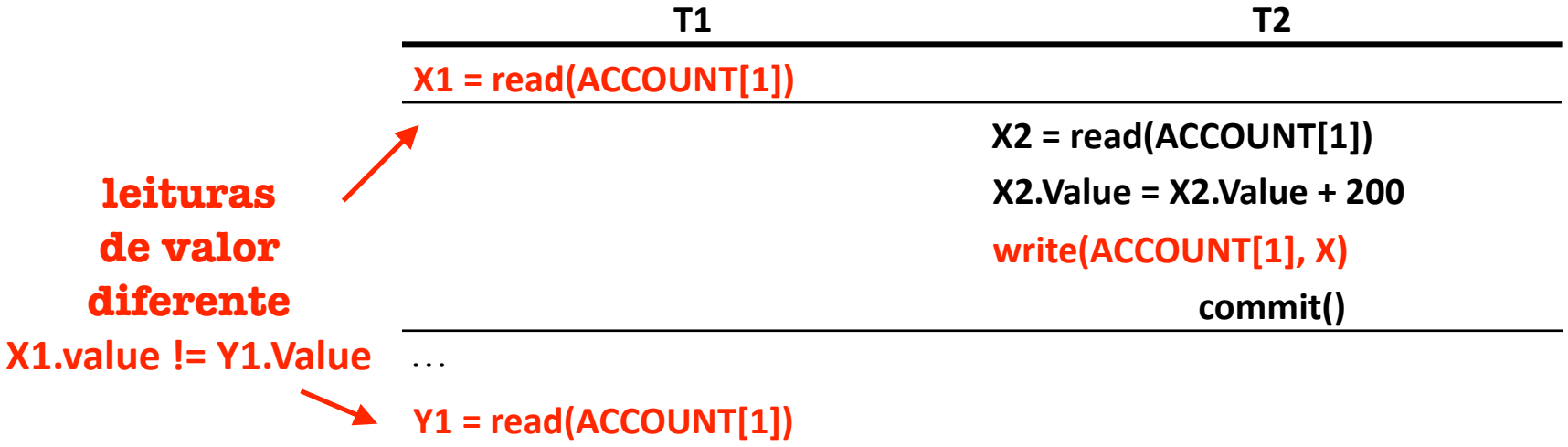
Leituras irrepetíveis

T1: SELECT Value FROM ACCOUNT WHERE AccountId = 1;

…

SELECT Value FROM ACCOUNT WHERE AccountId = 1;

T2: UPDATE ACCOUNT SET Value = Value + 200 WHERE AccountId = 1;



Leituras do mesmo registo (mesmo que não "sujas"), devolvem **valores diferentes por causa de uma escrita interveniente por outra transação**.

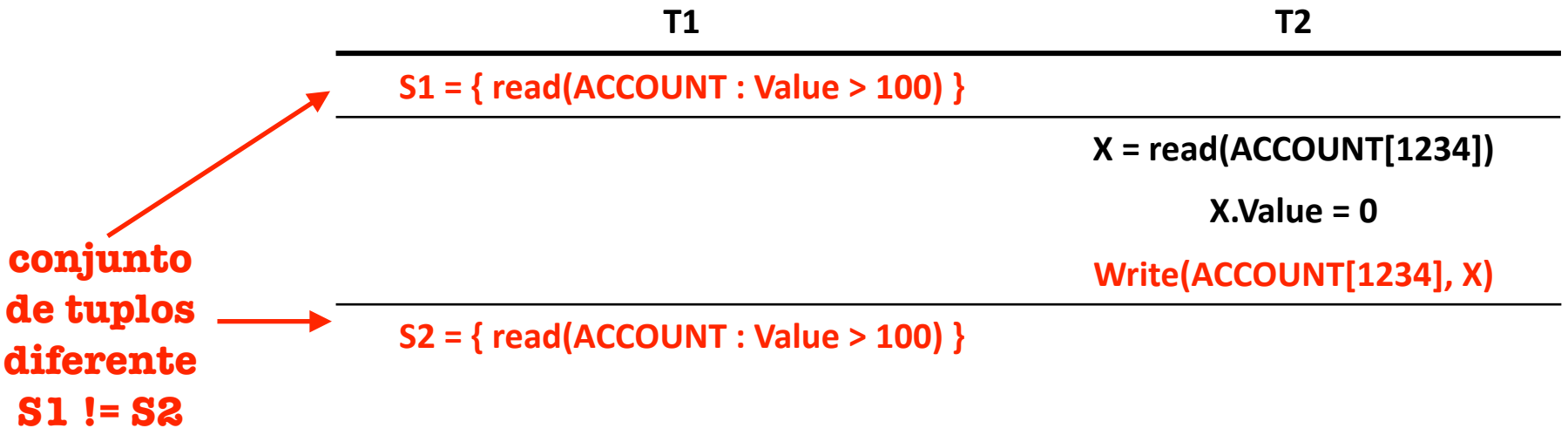
Tuplos fantasma

T1: SELECT AccountId FROM ACCOUNT WHERE Value > 100;

…

SELECT AccountId FROM ACCOUNT WHERE Value > 100;

T2: UPDATE ACCOUNT SET Value = 0 WHERE AccountId = 1234;

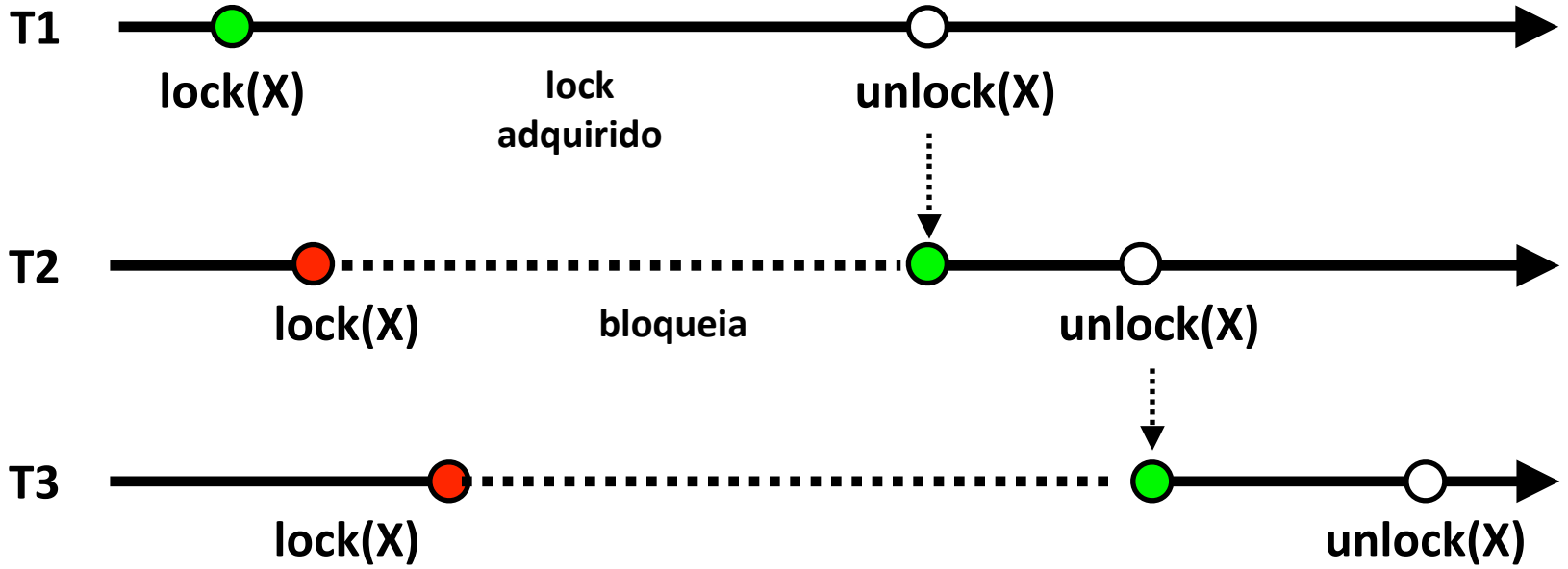


Leituras do "mesmo" conjunto de tuplos devolve resultados diferentes por causa de uma escrita interveniente por outra transação.

Obtenção de escalonamentos serializáveis

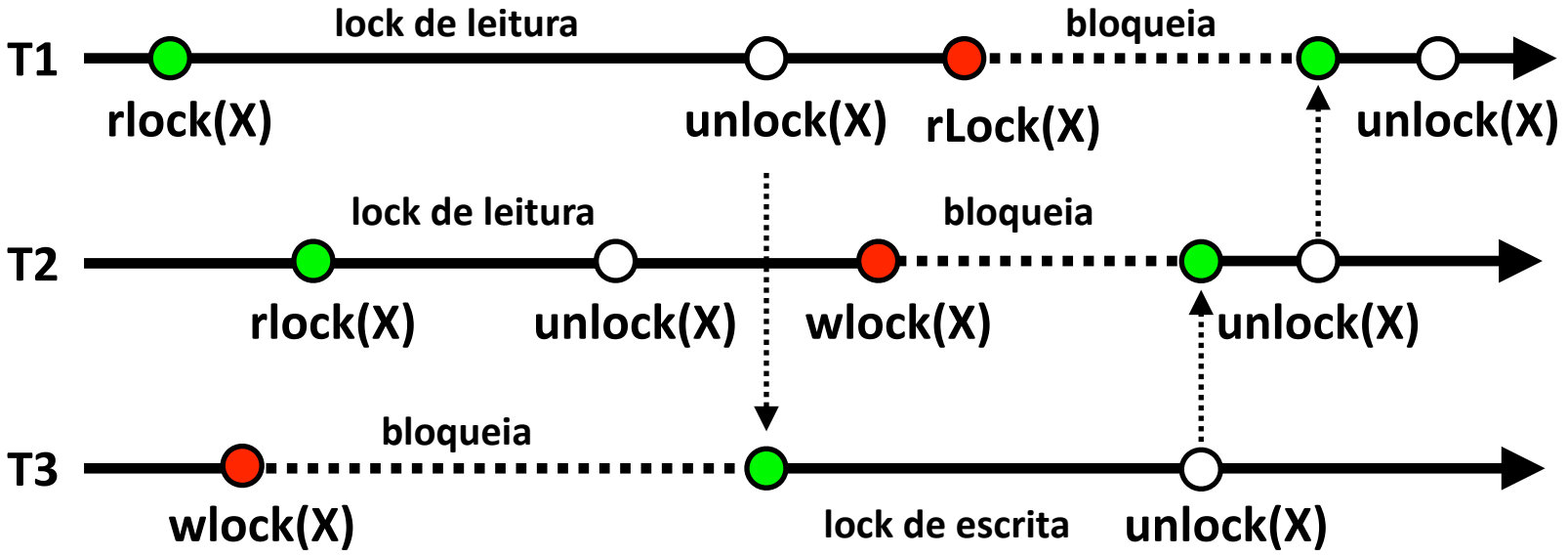
- Para escalonamentos serializáveis, SGBDs tipicamente empregam "locks" para exclusão mútua no acesso a registos e protocolos de **"two-phase locking"**.
- **Locks**
  - Um lock sobre um registo pode ser adquirido por uma transação, inibindo outras de fazer o mesmo, e depois libertado.
  - Iremos ver como funcionam dois tipos de lock: **binários** e **"read/write"**.
- **"Two-phase locking"**
  - **Fase 1 - expansão** - Transação adquire locks progressivamente.
  - **Fase 2: - contração** - Transação liberta locks progressivamente.

Locks binários (também chamados exclusivos)



- **lock(X)**: Adquire lock sobre **X**. Se lock estiver adquirido por outra transação, a transação **bloqueia** até que essa transação liberte o lock.
- **unlock(X)**: Liberta lock sobre **X**. Se houverem transações bloqueadas na aquisição do lock para **X**, uma delas pode adquirir o lock.

Locks "read/write"



- **rlock(X)**: Adquire **lock de leitura, também chamado lock de partilha** sobre **X**. Operação bloqueia enquanto outra transação tiver um lock de escrita. **São permitidos múltiplos locks de leitura sobre um item X, mas não em simultâneo com um lock de escrita.**
- **wlock(X)**: Adquire **lock de escrita, também chamado lock exclusivo**. Operação bloqueia enquanto outra transação tiver um lock de escrita ou de leitura. **É permitido apenas um lock de escrita por item X**.
- **unlock(X)**: Liberta lock de leitura ou de escrita sobre **X**.

Escalonamento serializável usando locks

T1: UPDATE ACCOUNT SET Value = Value - 100 WHERE AccountId = 1;

T2: UPDATE ACCOUNT SET Value = Value - 200 WHERE AccountId = 1;

T3: SELECT Value FROM ACCOUNT WHERE AccountId = 1;

T1	T2	T3
wlock(ACCOUNT[1])		
		rlock(ACCOUNT[1])
	wlock(ACCOUNT[1])	
X1 = read(ACCOUNT[1]) X1.Value = X1.Value - 100 write(ACCOUNT[1], X1) commit() unlock(ACCOUNT[1])	bloqueada	bloqueada
		X3 = read(ACCOUNT[1]) commit() unlock(ACCOUNT[1])
	X2 = read(ACCOUNT[1]) X2.Value = X2.Value - 200 write(ACCOUNT[1], X2) commit() unlock(ACCOUNT[1])	

Escalonamento não-serializável usando locks

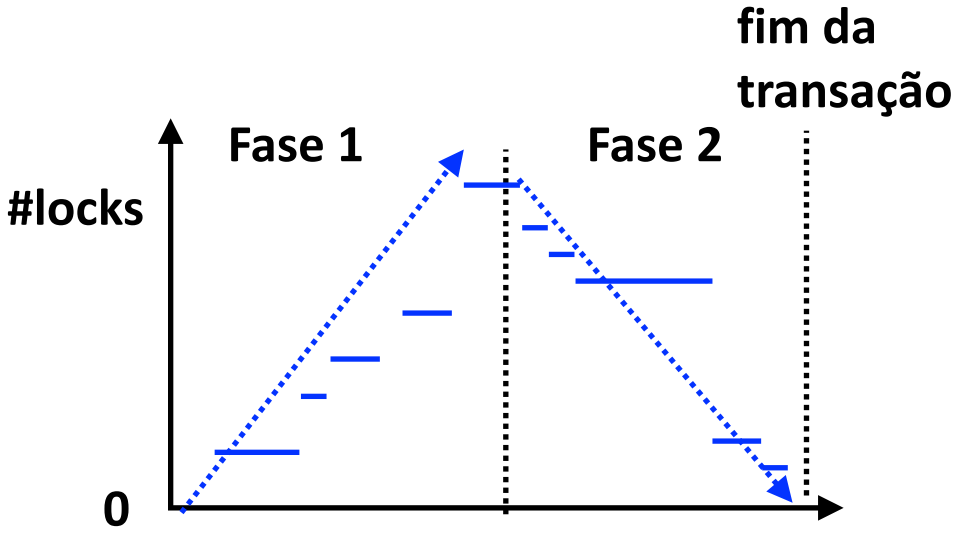
(T1 e T2 como no slide anterior)

T3: SELECT Value FROM ACCOUNT WHERE AccountId = 1; …
SELECT Value FROM ACCOUNT WHERE AccountId = 1;

T1	T2	T3
wlock(ACCOUNT[1]		
		rlock(ACCOUNT[1])
	wlock(ACCOUNT[1])	
X1 = read(ACCOUNT[1]) X1.Value = X1.Value - 100 write(ACCOUNT[1], X1) commit() unlock(ACCOUNT[1])	bloqueada	bloqueada
		X3 = read(ACCOUNT[1]) unlock(ACCOUNT[1])
	X2 = read(ACCOUNT[1])	
		rlock(ACCOUNT[1]
	X2.Value = X2.Value - 200 write(ACCOUNT[1], X2) commit() unlock(ACCOUNT[1])	bloqueada
		Y3 = read(ACCOUNT[1]) unlock(ACCOUNT[1])

leituras de valor diferente
X3.value != Y3.Value

"Two-phase locking"



- Para garantir serialização empregam-se protocolos baseados em **"two-phase" locking" (2PL)**
  - **Fase 1 — expansão**: nesta fase uma transação só adquire locks.
  - **Fase 2 — contração:** nesta fase uma transação só liberta locks.
- Um protocolo 2PL comum em SGBDs é **o 2PL estrito — "Strict 2PL (S2PL)** — em que todos os **locks de escrita só são libertados no final da transação** (tanto no caso de "commit" como de "rollback"). Na variação "**2PL estrito forte" (Strong Strict 2PL — SS2PL)** tanto os locks de escrita como de leitura são libertados apenas no final da transação.

Escalonamento 2PL — exemplo

(T1 e T2 como no exemplo anterior)

T3: SELECT Value FROM ACCOUNT WHERE AccountId = 1; …
SELECT Value FROM ACCOUNT WHERE AccountId = 1;

T1	T2	T3
wlock(ACCOUNT[1]		
		rlock(ACCOUNT[1])
	wlock(ACCOUNT[1])	
X1 = read(ACCOUNT[1])		bloqueada
X1.Value = X1.Value - 100		
write(ACCOUNT[1], X1)		
commit()		
unlock(ACCOUNT[1])		
		X3 = read(ACCOUNT[1])
		Y3 = read(ACCOUNT[1])
		commit()
		unlock(ACCOUNT[1])
	X2 = read(ACCOUNT[1])	
	X2.Value = X2.Value - 200	
	write(ACCOUNT[1], X2)	
	commit()	
	unlock(ACCOUNT[1])	

Escalonamento 2PL — outro exemplo

T1: UPDATE ACCOUNT SET Value = Value - 100 WHERE AccountId = 2;
UPDATE ACCOUNT SET Value = Value + 100 WHERE AccountId = 1;

T2: UPDATE ACCOUNT SET Value = Value - 200 WHERE AccountId = 3;
UPDATE ACCOUNT SET Value = Value + 200 WHERE AccountId = 1;

T1	T2
wlock(ACCOUNT[2])	
	wlock(ACCOUNT[3] … actualização de ACCOUNT[3] …
… actualização de ACCOUNT[2] … wlock(ACCOUNT[1])	
	wlock(ACCOUNT[1])
… actualização de ACCOUNT[1] … commit() unlock(ACCOUNT[1]) unlock(ACCOUNT[2])	bloqueada
	… actualização de ACCOUNT[1] … commit() unlock(ACCOUNT[1]) unlock(ACCOUNT[3])

Possibilidade de deadlock

T1: UPDATE ACCOUNT SET Value = Value - 100 WHERE AccountId = 2;
UPDATE ACCOUNT SET Value = Value + 100 WHERE AccountId = 1;

T2: UPDATE ACCOUNT SET Value = Value - 200 WHERE AccountId = 1;
UPDATE ACCOUNT SET Value = Value + 200 WHERE AccountId = 2;

T1	T2
	wlock(ACCOUNT[1])
	… actualização de ACCOUNT[1] …
wlock(ACCOUNT[2])	
… actualização de ACCOUNT[2] …	
wlock(ACCOUNT[1])	
bloqueada	wlock(ACCOUNT[2])
	bloqueada

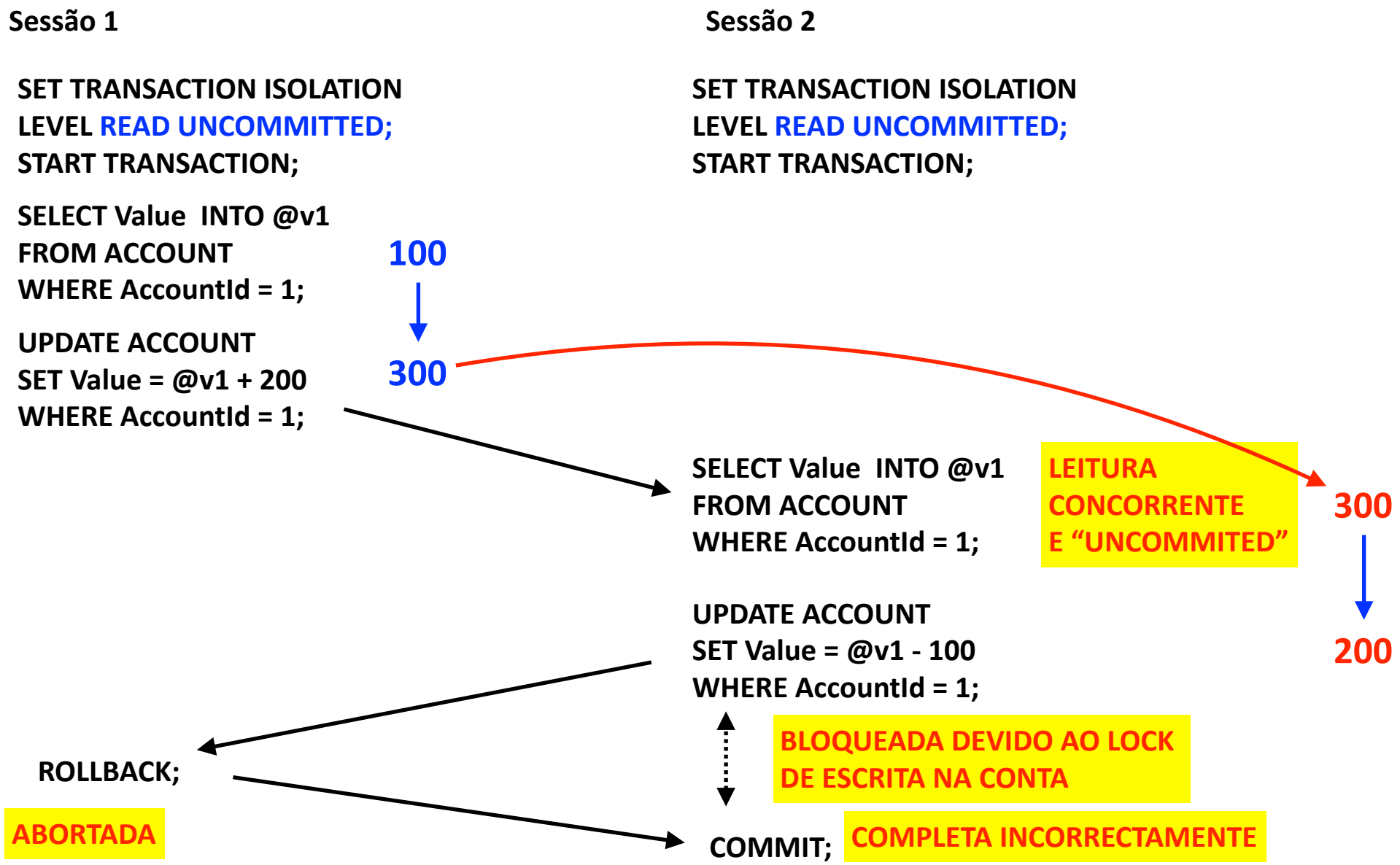
Nenhuma das transações consegue progredir. Temos uma situação de deadlock. SGDB deve tentar prevenir deadlocks ou no mínimo detectá-los. Vamos ver depois como no caso MySQL a questão é resolvida.

SQL — níveis de isolamento

Nível de isolamento	Leituras sujas	Leituras irrepetíveis	Leituras fantasma
READ UNCOMMITED	Permitidas	Permitidas	Permitidas
READ COMMITED	—	Permitidas	Permitidas
REPEATABLE READ	—	—	Permitidas
SERIALIZABLE	—	—	—

- O standard SQL'92 introduziu níveis standard de isolamento para transações. Estes variam nos três tipo de anomalias de leitura discutidas anteriormente desde o nível de isolamento mais fraco (**READ UNCOMMITED**) ao mais forte (**SERIALIZABLE**).
- Em MySQL podemos configurar o nível de isolamento usando o comando **SET TRANSACTION ISOLATION LEVEL.**

Exemplo — leituras sujas e locks de escrita



Exemplo — READ COMMITED

Sessão 1

```
SET TRANSACTION ISOLATION
LEVEL READ COMMITTED;
START TRANSACTION;

SELECT Value  INTO @v1
FROM ACCOUNT
WHERE AccountId = 1;

UPDATE ACCOUNT
SET Value = @v1 + 200
WHERE AccountId = 1;

ROLLBACK;
```

ABORTADA

Sessão 2

```
SET TRANSACTION ISOLATION
LEVEL READ COMMITTED;
START TRANSACTION;

SELECT Value  INTO @v1
FROM ACCOUNT
WHERE AccountId = 1;

UPDATE ACCOUNT
SET Value = @v1 - 100
WHERE AccountId = 1;

COMMIT;
```

100 → 300

LEITURA CONCORRENTE

100 → 0

BLOQUEADA DEVIDO AO LOCK DE ESCRITA NA CONTA

COMPLETA CORRECTAMENTE

Exemplo — SERIALIZABLE

Sessão 1

```
SET TRANSACTION ISOLATION
LEVEL SERIALIZABLE;
START TRANSACTION;

SELECT Value  INTO @v1
FROM ACCOUNT
WHERE AccountId = 1;

UPDATE ACCOUNT
SET Value = @v1 + 200
WHERE AccountId = 1;
```

100 ↓ 300

```
ROLLBACK;
```

ABORTADA

Sessão 2

```
SET TRANSACTION ISOLATION
LEVEL SERIALIZABLE;
START TRANSACTION;

SELECT Value  INTO @v1
FROM ACCOUNT
WHERE AccountId = 1;
```

LEITURA CONCORRENTE

100 ↓ 0

BLOQUEADA DEVIDO AO LOCK DE ESCRITA E LEITURA NA CONTA

```
UPDATE ACCOUNT
SET Value = @v1 - 100
WHERE AccountId = 1;

COMMIT;
```

COMPLETA CORRECTAMENTE

Exemplo de deadlock

Sessão 1

```
SET TRANSACTION ISOLATION
LEVEL SERIALIZABLE;
START TRANSACTION;

UPDATE ACCOUNT
SET Value = 100
WHERE AccountId = 1;
```

Sessão 2

```
SET TRANSACTION ISOLATION
LEVEL SERIALIZABLE;
START TRANSACTION;

UPDATE ACCOUNT
SET Value = 200
WHERE AccountId = 2;
```

```
SELECT Value  INTO @v2
FROM ACCOUNT
WHERE AccountId = 2;
```

BLOQUEADA

```
SELECT Value  INTO @v1
FROM ACCOUNT
WHERE AccountId = 1;
```

CONSEGUE CONTINUAR

```
ERROR 1213 (40001): Deadlock found when
trying to get lock; try restarting
transaction
```