Transacções (motivação)

- Acesso concorrente: Vários utilizadores (ou programas) fazem selects, updates, deletes e inserts, em simultâneo.

- O SGBD tem de orquestrar os acessos concorrentes de modo a que o estado da BD seja sempre coerente.

- Exemplos críticos: Bancos, reservas de aviões, vendas no amazon.com, etc.

Exemplo: Multibanco

- Exemplo: marido e mulher tiram dinheiro de uma conta conjunta a partir de caixas multibanco distintas, mais ou menos ao mesmo tempo.

```
CREATE TABLE conta (
  numero INTEGER PRIMARY KEY,
  nome   VARCHAR(50),
  saldo  FLOAT
);
```

- Saldo inicial = 400 euros.

- Marido tira 100 euros, mulher tira 70.

- Saldo final = ???

Multibanco (cont.)

```
-- utilizador introduz nº da conta
meuSaldo := SELECT saldo FROM conta
            WHERE numero = meuNumero;
-- mostra o saldo actual
-- utilizador introduz quantidade a retirar
meuSaldo := meuSaldo - quantidade;
IF( meuSaldo > 0 ) THEN
   UPDATE conta SET saldo = meuSaldo
   WHERE numero = meuNumero;
END IF;
```

- Acesso concorrente à mesma conta pode dar problemas.

- Mas acesso concorrente a contas distintas não faz mal.

Outro exemplo

- Transferir 100 euros da conta nº 333 para a conta nº 888.

```
UPDATE conta SET saldo = saldo - 100
WHERE numero = 333;

UPDATE conta SET saldo = saldo + 100
WHERE numero = 888;
```

- SGBD tem um "crash" no meio. O que é que acontece?

Transacções

- Uma transacção é uma sequência de operações SQL que é tratada como um todo.

- Por defeito, uma instrução de SQL é uma transacção.

- As transacções devem obedecer às propriedades "ACID" :
  - *Atomicity*
  - *Consistency*
  - *Isolation*
  - *Durability*

Propriedades "ACID"

- *Atomic*: Transacção deve ser tratada como uma unidade. Ou tudo é executado, ou nada é executado.

- *Consistent*: As restrições na BD devem ser mantidas.

- *Isolated*: Dá a ilusão ao utilizador de que a transacção é executada isoladamente, sem interferências de outras transacções.

- *Durable*: Os efeitos das transacções não se podem perder devido a "crashes" do sistema.

COMMIT e ROLLBACK

- Em SQL, a instrução BEGIN WORK dá início a uma transacção.

- COMMIT WORK termina uma transacção (as modificações são feitas fisicamente na BD).

- ROLLBACK WORK também termina uma transacção (mas o efeito da transacção é anulado, uma espécie de "undo").

- Por defeito, uma instrução SQL é uma transacção.

Encadeamento de instruções

número	nome	saldo
333	Zé	300
888	Maria	250

Programa 1

```
UPDATE conta                          // (updt1)
SET saldo = saldo - 100
WHERE numero = 333;

UPDATE conta                          // (updt2)
SET saldo = saldo + 100
WHERE numero = 888;
```

Programa 2

```
UPDATE conta                          // (updt3)
SET saldo = saldo + 150
WHERE numero = 888;

SELECT SUM(saldo) FROM conta;         // (sum)
```

Encadeamento de instruções (cont.)

- A única restrição que existe é que (updt1) tem de vir antes de (updt2), e que (updt3) tem de vir antes de (sum).

- Qualquer outro tipo de encadeamento de instruções é possível.

Exemplo

- Vamos supor que as instruções são executadas pela ordem (updt1) (updt3) (sum) (updt2).

Instrução:	(updt1)	(updt3)	(sum)	(updt2)
Resultado:			600	
			???	

- sum=600 é inconsistente, deveria ser 700.

A solução é usar transacções

- Agrupamos (updt1) e (updt2) numa transacção.

- O programa 2 só pode ver o estado da BD antes da transferência ter sido iniciado, ou depois da transferência ter sido completada.

Programa 1

```
BEGIN WORK;

UPDATE conta                          // (updt1)
SET saldo = saldo - 100
WHERE numero = 333;

UPDATE conta                          // (updt2)
SET saldo = saldo + 100
WHERE numero = 888;

COMMIT WORK;
```

Níveis de isolamento

- Os utilizadores (programas) só vêem resultados de transacções que fizeram COMMIT.

- E se estivermos no meio de uma transacção, e outra transacção fizer COMMIT?

- SQL define 4 níveis de isolamento de transacções

  1. SERIALIZABLE
  2. REPEATABLE READ
  3. READ COMMITTED
  4. READ UNCOMMITTED

Exemplo

Programa 1

```
BEGIN WORK;

UPDATE conta                        // (updt1)
SET saldo = saldo - 100
WHERE numero = 333;

UPDATE conta                        // (updt2)
SET saldo = saldo + 100
WHERE numero = 888;

COMMIT WORK;
```

Programa 2

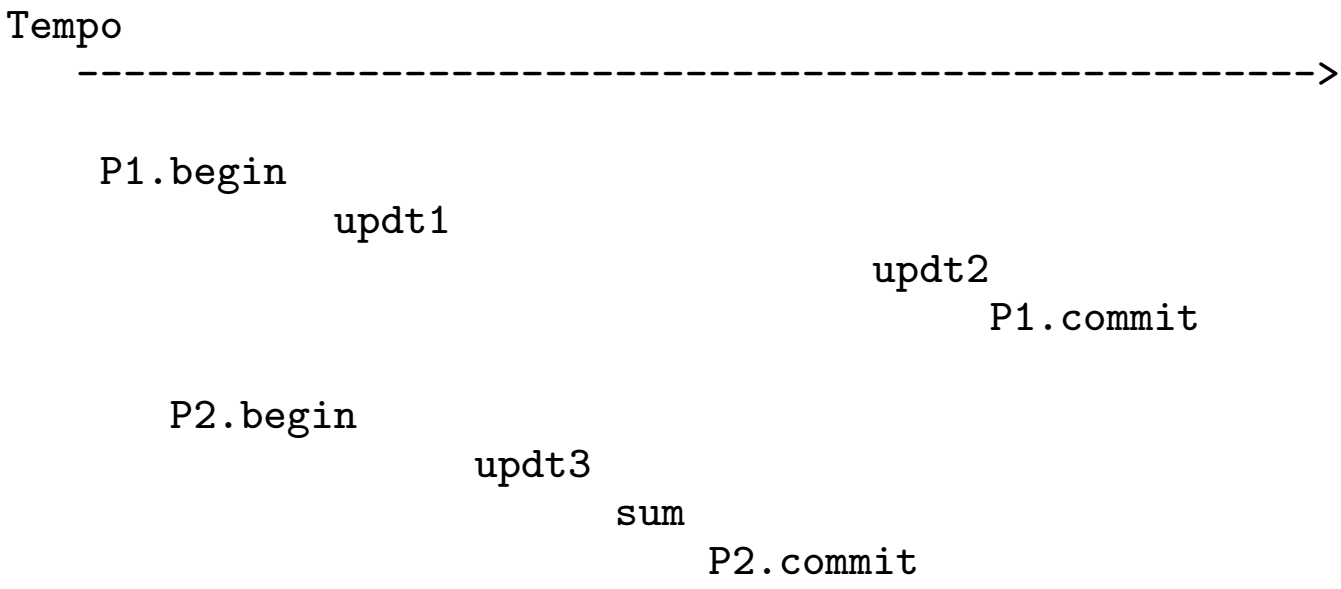
```
BEGIN WORK;

UPDATE conta                        // (updt3)
SET saldo = saldo + 150
WHERE numero = 888;

SELECT SUM(saldo) FROM conta;       // (sum)

COMMIT WORK;
```

Exemplo (cont.)



- Entre o final de (updt1) e o início de (updt2), o estado da BD alterou-se.

- (updt2) deve ver o novo estado da BD?

- SQL define 4 níveis de isolamento para tratar este problema. Compete ao utilizador (programador) especificar o nível de isolamento pretendido.

Níveis de isolamento (cont.)

- SQL define 4 níveis de isolamento de transacções

  1. SERIALIZABLE
  2. REPEATABLE READ
  3. READ COMMITTED
  4. READ UNCOMMITTED

Em SQL:

```
SET TRANSACTION ISOLATION LEVEL <X>
   onde <X> é um dos 4 níveis
   especificados acima
```

SERIALIZABLE

- Prog1 = (begin)(updt1)(updt2)(commit)
  Prog2 = (begin)(updt3)(sum)(commit)

- Se Prog1 correr com nível de isolamento SERIALIZABLE, então apenas verá o estado da BD antes ou depois de Prog2 ter sido executado, nunca no meio.

READ COMMITTED

- Se Prog1 correr com nível de isolamento READ COMMITTED, então o encadeamento (updt1) (updt3) (max) (p2.commit) (updt2) é possível.

REPEATABLE READ

- É semelhante ao READ COMMITTED, com a diferença de que se um tuplo é lido uma vez, então terá de ser forçosamente devolvido se a leitura for repetida.

READ UNCOMMITTED

- Se uma transacção correr com READ UNCOMMITTED, poderá ler dados que tenham sido escritos temporariamente por outras transacções (mesmo que essas transacções venham a fazer ROLLBACK!)

- Exemplo, vamos supor que Prog2 faz ROLLBACK em vez de COMMIT. Nesse caso, o estado da BD vai ficar incoerente.

Em PostgreSQL

- PostgreSQL só define SERIALIZABLE e READ COMMITTED.

- SERIALIZABLE dá um maior nível de isolamento.

- READ COMMITTED dá menos isolamento mas é mais eficiente de implementar pelo SGBD.

- A escolha depende da aplicação.