# 資料後蓋前問題

- **帳戶A（1000）分別使用兩個交易同時轉帳給帳戶B(500)及帳戶C(600)**

| | AccountName | balance |
|---|---|---|
| 資料庫 | A | 1000 |

0s

時間

tx1

```
select * from accounts where
accountName="A";
```

| AccountName | balance |
|---|---|
| A | 1000 |

tx2

```
select * from accounts where
accountName="A";
```

| AccountName | balance |
|---|---|
| A | 1000 |

2s

時間

tx1

轉給 B 500元

```
update accounts set balance=500 where accountName='A';
update accounts set balance=1500 where accountName='B';
commit;
```

| AccountName | balance |
|---|---|
| A | 500 |
| B | 1500 |

tx2

計算中

| AccountName | balance |
|---|---|
| A | 1000 |
| C | 1000 |

3s

時間

tx1

tx2

轉帳給 C 600

```
update accounts set balance=400 where accountName='A';
update accounts set balance=1600 where accountName='C';
commit;
```

| AccountName | balance |
|---|---|
| A | 400 |
| C | 1600 |

# Locking

- **交易隔離的處理方式有兩種，一種是 Pessimistic Locking（悲觀的），另一種是 Optimistic Locking (樂觀的)**
- **Pessimistic Locking(悲觀鎖定)**
  - **資料在操作的過程中交易寫入很頻繁而造成資料不一致。必須採用較高的isolation level進行保護。**
- **Optimistic Locking(樂觀鎖定)**
  - **資料大多為『讀取』的狀態，極少的情況下會發生不完整性，而且能同時讀取增加效能比防止同時更新來得重要。會採用 *TRANSACTION_READ_COMMITTED* 取得較佳的效能並利用『*Optimistic Locking Attribute*』來防止資料更新時造成的問題。**

# *Optimistic Locking Attribute*

- 利用一個數字型式的欄位每次更新即加1來保護資料。
  - 例如：**Hibernate Versioning**

# Versioning

- **增加一個version的欄位型態為int。**
- **每更新一次該筆資料這個version欄位必須加1。**
- **更新時必須將version欄位放入where條件式中。**

| | AccountName | balance | version |
|---|---|---|---|
| 資料庫 | A | 1000 | 0 |

0s

時間

tx1

```
select * from accounts where
accountName="A";
```

| AccountName | balance | version |
|---|---|---|
| A | 1000 | 0 |

tx2

```
select * from accounts where
accountName="A";
```

| AccountName | balance | version |
|---|---|---|
| A | 1000 | 0 |

2s

時間

tx1

轉給 B 500元

```
update accounts set balance=500,version=version+1 where
accountName='A' and version=0;
update accounts set balance=1500,version=version+1 where
accountName='B' and version=0;
commit;
```

| AccountName | balance | version |
|---|---|---|
| A | 500 | 1 |
| B | 1500 | 1 |

tx2

計算中

| AccountName | balance | version |
|---|---|---|
| A | 1000 | 0 |
| C | 1000 | 0 |

| AccountName | balance | version |
|---|---|---|
| A | 500 | 1 |
| B | 1500 | 1 |

3s

時間

tx2

```
轉帳給 C 600
update accounts set balance=400,version=version+1 where
accountName='A' and version=0;
update accounts set balance=1600,version=version+1 where
accountName='C' and version=0;
commit;   //
```

| AccountName | balance | version |
|---|---|---|
| A | 400 | ? |
| C | 1600 | ? |

# Versioning

- 如果更新不到資料，則 java.sql.Statement中

  int executeUpdate(String sql) 回傳值為0，即可知道沒有更新到資料，可執行rollback並丟出例外。

```
Statement stmt = conn.createStatement();

int rowUpated = stme.executeUpdate("update emp set .....   version=0");

if ( rowUpated == 0 ){
  //如果沒有任何筆數被更新，則rollback transaction
   conn.rollback();
   throw new RuntimeException("data inconsistency");
}
```