Transaction & ACID

Jimmy Fu

什麼是Transaction？

Transaction
交易/事務

資料庫執行過程中的一個「邏輯單位」

一個transaction = 一組一連串對資料庫進行存取、讀取的行為

兩種結局

成功 or **失敗**

全部SQL執行成功　　-> commit

只要有任一個SQL失敗 -> rollback

為什麼要有Transaction？

- 解決**需要一起發生**的事件**但**事件或事件的參與者**不同時或不一致**的問題
- 簡單說transaction存在的兩個目的：
  1. 失敗中的救難英雄/發生災難的時光倒轉機器

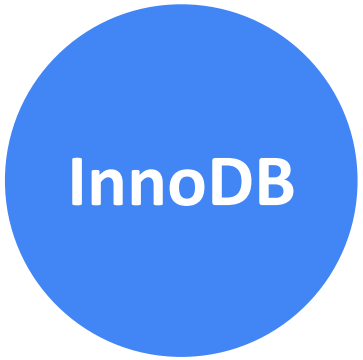
     allow correct recovery from failures and keep a database consistent

     （會讓一切回到初始狀態）
  2. 獨立作業的守護者

     provide isolation between programs accessing a database **concurrently**
- 簡單的舉例：轉帳

以 MySQL 為例

不支援交易功能



有支援交易功能
可進行mysql執行
SHOW TABLE STATUS;
檢查engine這個欄位

Transaction需要的指令

- START TRANSACTION
- read/write query
- COMMIT
- ROLLBACK

```
CREATE PROCEDURE importCustomerWithTransaciton ()
    BEGIN
        DECLARE exit handler for sqlexception BEGIN ROLLBACK; END;
        DECLARE exit handler for sqlwarning BEGIN ROLLBACK; END;
        START TRANSACTION;
            insert into customer set name= 'Gin2', money = 10000;
            insert into customer set name= 'Jimmy2', money = 10;
            insert into customer set name= 'imerror2', money = 'abc';
        COMMIT;
    END;
//
```

Code範例：大量匯入客戶資料

儲存點

- 交易過程中，可標示多個不同的儲存點，
  有需要時可 ROLLBACK 到某個儲存點。
- 建立儲存點：SAVEPOINT 名稱
  刪除儲存點：RELEASE SAVEPOINT 名稱
- ROLLBACK 到某個儲存點：ROLLBACK TO SAVEPOINT 名稱
- 如果建立新儲存點時，已有同名稱的舊儲存點，舊儲存點將被刪除，
  並建立新的儲存點。

儲存點範例

```
CREATE PROCEDURE rollbackToSavePoint ()
    BEGIN
        DECLARE exit handler for sqlexception BEGIN ROLLBACK to firstsavepoint; END;
        DECLARE exit handler for sqlwarning BEGIN ROLLBACK; END;
        START TRANSACTION;
            insert into customer set name= 'Gin', money = 10000;
            SAVEPOINT firstsavepoint;
            insert into customer set name= 'Jimmy', money = 10;

            insert into customer set name= 'imerror2', money = 'abc';
        COMMIT;
    END;
//
```

會造成自動終止交易並 COMMIT 的指令

- 執行這些指令時，如同先執行了 commit ➔ 會先有 commit 的效果。
- 會造成自動終止交易的指令：
  - SET AUTOCOMMIT=1、 BEGIN、START TRANSACTION
  - DDL 指令：ALERT TABLE、CREATE INDEX、CREATE TABLE、DROP TABLE、DROP DATABASE、RENAME TABLE、TRUNCATE、LOCK TABLES、UNLOCK TABLES…等

DDL

- DDL (Data Definition Language)

  用來定義資料庫、資料表、檢視表、索引、預存程序、觸發程序、函數等資料庫物件。
- 可以用來建立、更新、刪除 table,schema,domain,index,view

  常用的有CREATE、DROP、ALTER
- **DDL 指令為不能 ROLLBACK 的指令**

AUTOCOMMIT 自動提交設定

- AUTOCOMMIT 的設定值，預設一般都是 1

  查詢目前 AUTOCOMMIT 的設定值：SELECT@@AUTOCOMMIT
- 將 AUTOCOMMIT 改為 0 時 (SET AUTOCOMMIT=0),

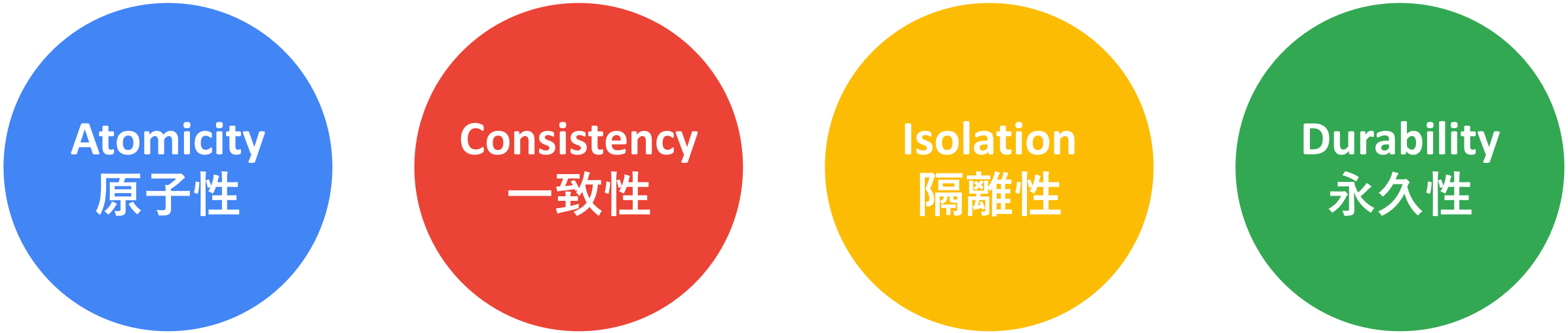
  就算沒使用 START TRANSACTION 或 BEGIN ,

  整個連線執行的 SQL 指令，都會等到下達 COMMIT 提交後,

  才會真正儲存變更。

  ➔ 當 AUTOCOMMIT=0 時，就跟開始了一個 Transaction 是一樣意思。

交易四大特性：ACID


Atomicity
原子性
Consistency
一致性
Isolation
隔離性
Durability
永久性

交易四大特性：ACID

Atomicity
原子性

把整個交易視為一個原子，
是一個不可分割的邏輯單位。

所以要嘛整個交易成功，不然就是整個交易失敗，
沒有成功一半這種事。
若有其中一個操作沒有完成，那就會回到初始狀態。

交易四大特性：ACID

Consistency
一致性

交易前後保持一致性。

Consistency ensures that a transaction can only **bring the database from one valid state to another, maintaining database invariants**

資料庫**從一個有效狀態經過交易之後變成另一個有效狀態**。
→ 要符合有效狀態，輸入的所有data就都要符合規則。
→ 交易進行後，資料庫的完整性沒有被破壞。
→ 不會因為進行了任何一個transaction，導致invariant有任何改變 (無論這個transaction成功與否)

交易四大特性：ACID

Isolation
隔離性

transaction過程中
不能被其他其他transaction影響

因為transaction只有成功或失敗，　無法預知最後的資料庫到底長怎樣

所以要確保每個transaction在進行的時候，是互不干擾的,

A transaction 跟 B transaction 同時進行的時候用的資料庫應該要長一樣,

不會發生 B transaction 抓到的資料是 A transaction 進行到一半的資料庫。

交易四大特性：ACID

Durability
永久性

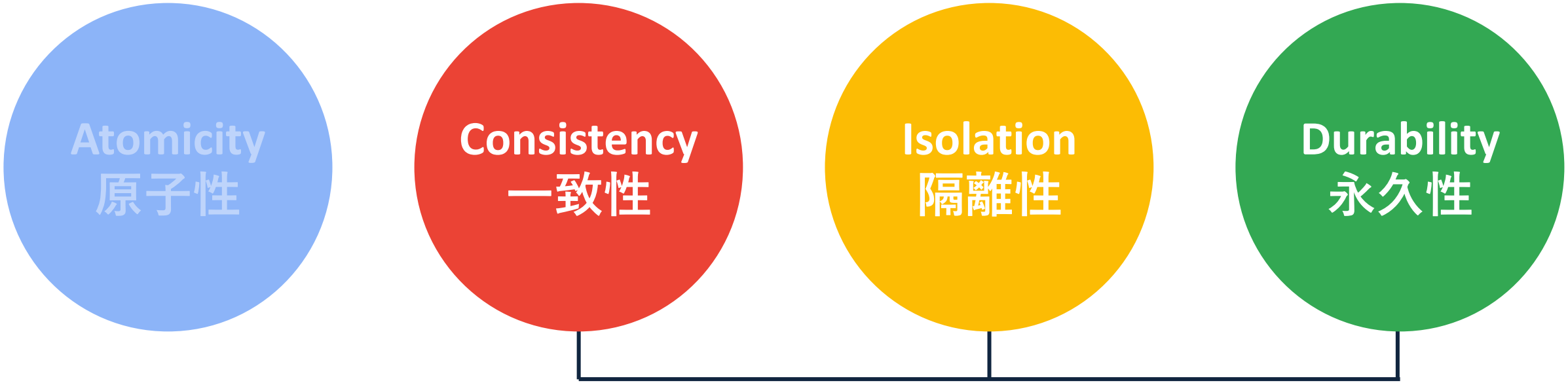
一旦資料修改完成，變永續存在，
就算系統發生故障也不應該毀損。

轉帳成功就成功了，

不會因為你的網銀App當掉，

剛剛轉好的帳又發生問題。

交易四大特性：ACID



也有人說 CID 不只是交易的特性
也是在處理資料庫query應該要注意或是遵守的特性

併發的 Transaction → 讀取錯誤

Dirty Read　髒讀

Non-repeatable read
無法重複讀取到的結果

Phantom Read　幻讀

兩個併發的 Transaction

Dirty Read　髒讀

Transaction A	Transaction B
BEGIN; SELECT money FROM customer WHERE id = 1; /* money = 100 */	
	BEGIN; UPDATE customer SET money = 1000 WHERE id = 1; /* money = 1000 */
SELECT money FROM customer WHERE id = 1; /* money = 1000 */	
髒讀(dirty read) /* money = 1000 */ /* money 實際為 100，但 Transaction A 以為是 1000 */	ROLLBACK; /* money = 100 */

兩個併發的 Transaction

Non-repeatable read
無法重複讀取到的結果

Transaction A	Transaction B
BEGIN; SELECT money FROM customer WHERE id = 1; /* money = 500 */	
	BEGIN; UPDATE customer SET money = 100 WHERE id = 1; /* money = 1000 */
	COMMIT; /* money = 1000 */
SELECT money FROM customer WHERE id = 1; /* money = 1000 */	
無法重覆讀取到相同結果(non-repeatable read) 第一次讀到 money = 500 第二次讀到 money = 1000	

兩個併發的 Transaction

Phantom Read　幻讀

Transaction A	Transaction B
BEGIN; SELECT * FROM product ; /* id=1 , price= 300 */	
	BEGIN; INSERT INTO product VALUES (2, 500); COMMMIT;
SELECT * FROM product ; /* id=1 , price= 300 */ /* id=2 , price= 500 */	
幻讀(phantom read) 兩次取得的筆數不相同	

一致性的根本解決之道

交易的四種隔離層級(Isolation Level)

交易的四種隔離層級(Isolation Level)

這是最低的層級。

SELECT 可以讀取其他交易中尚未 commit 的資料。

如果讀取的資料，最後被 rollback,

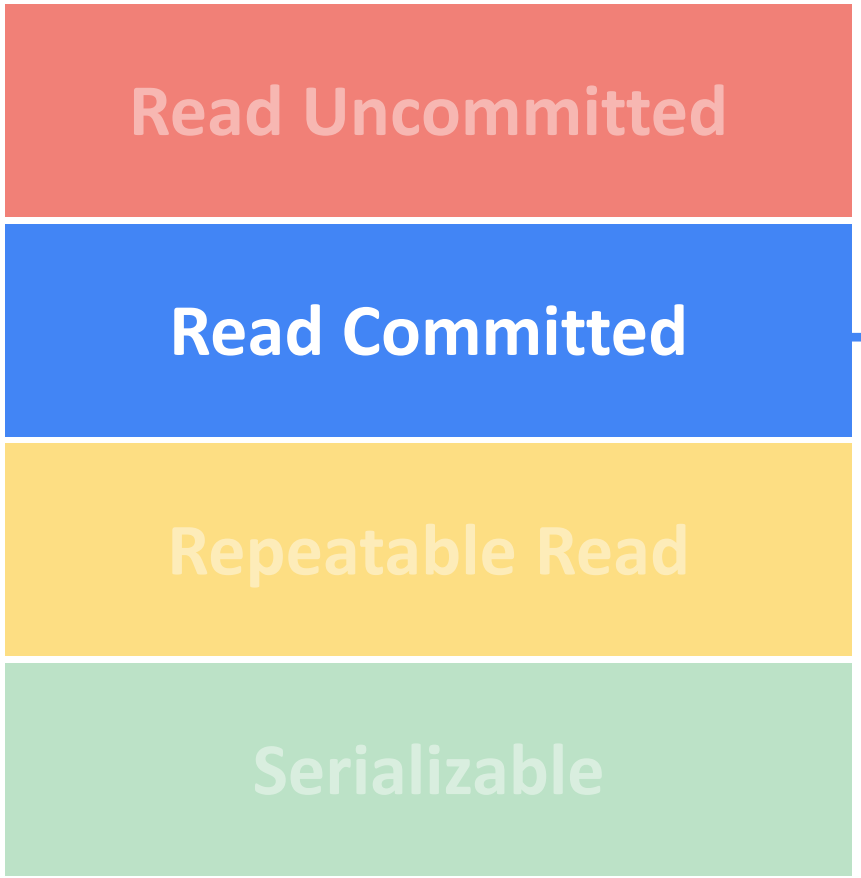
便會造成讀取到被取消的資料 (dirty read)。

⚠ SELECT 不會被阻擋，如果是 UPDATE 仍會被阻擋⚠

▲ 可能產生：

- 髒讀 (dirty read)
- 無法重覆讀取到相同結果 (non-repeatable read)
- 幻讀 (phantom read)

交易的四種隔離層級(Isolation Level)



此層級會考慮其他交易的執行結果。

→ SELECT 可以讀取其他交易 commit 後的結果。

尚未 commit 的結果不能讀取

→ 不會有前一個層級 dirty read 的問題。

⚠ 若兩個 SELECT 之間，有其他交易 commit 過資料了，

會造成第一次跟第二次取得的資料不一樣，

也就是重覆讀取可能結果不一樣 (non-repeatable read)

▲ 可能產生：

- 無法重覆讀取到相同結果 (non-repeatable read)
- 幻讀 (phantom read)

交易的四種隔離層級(Isolation Level)

此為 innodb 預設的隔離層級，不會考慮其他交易的修改。

同一交易內，除非自己修改，

否則重覆 SELECT 的結果一定相同

→ 不會有前一個層級 non-repeatable read 的問題。

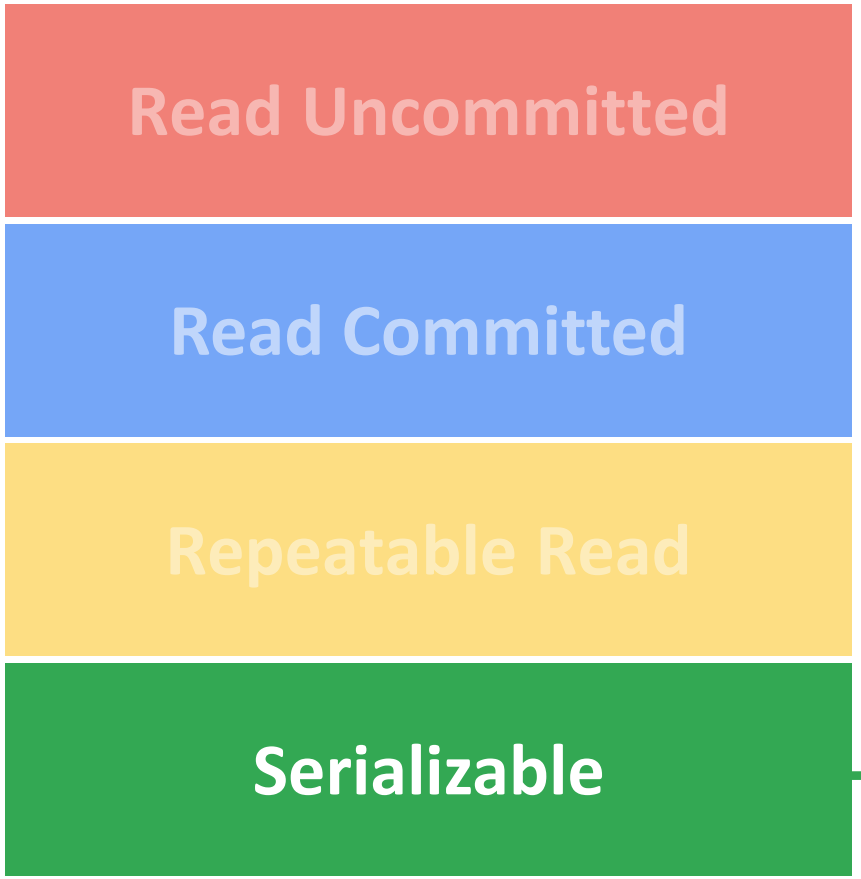
⚠ 注意：此說明僅針對 innodb ⚠

一般來說，此層級會有幻讀(phantom read)的問題，

但 innodb 使用了 Next-Key Locking 的方式，

避免了 phantom read。

交易的四種隔離層級(Isolation Level)



跟 REPEATABLE READ 類似,

但是將所有的 SELECT 指令

都隱含轉換為 SELECT … LOCK IN SHARE MODE

死結 Deadlock

- 併發的 Transaction 會產生讀取錯誤的狀況中

  有一個比較特別的名詞：Race condition

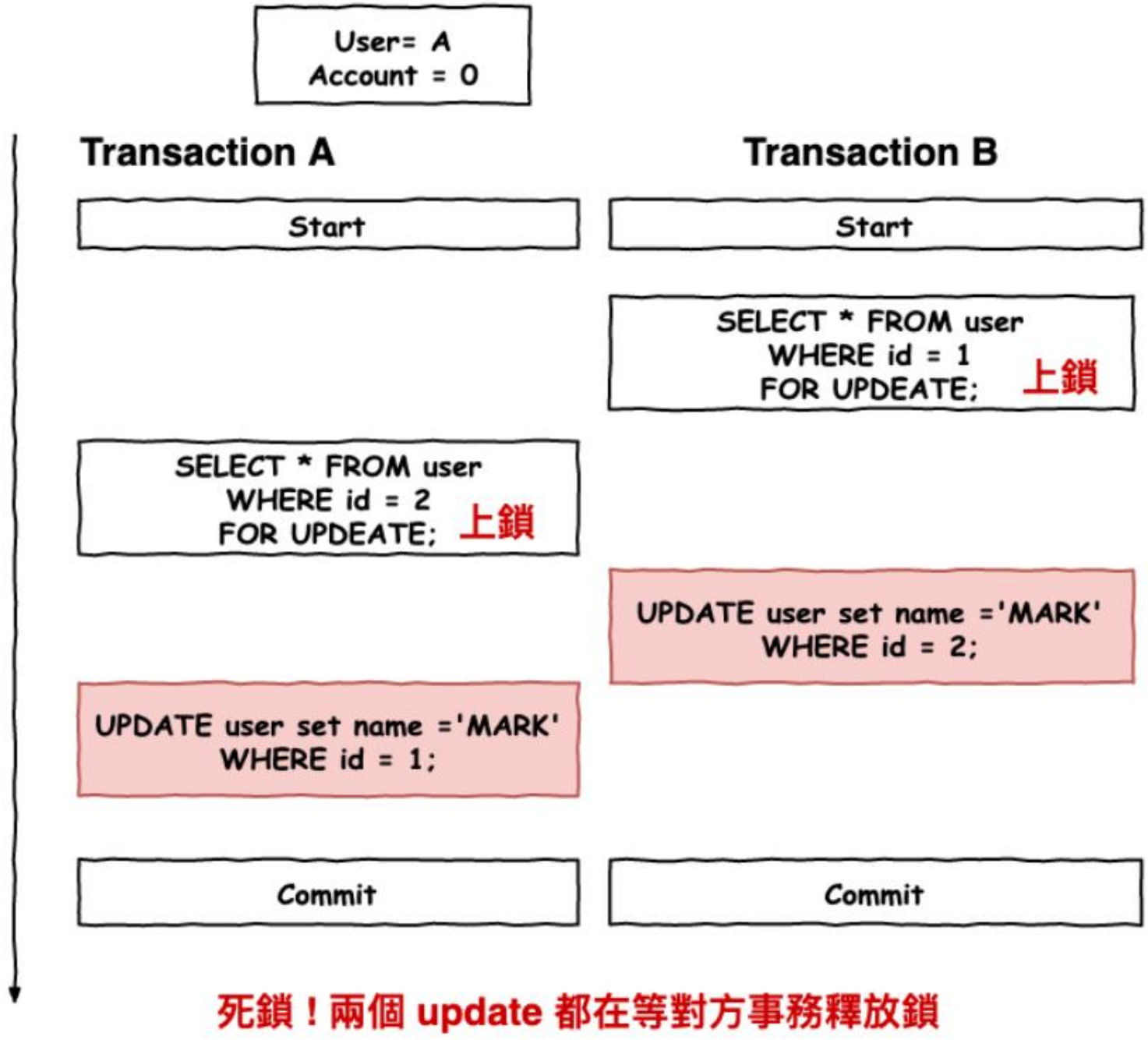
- Race condition 在任何多執行緒的程序中都有可能發生,

  → 用一些有 ‘鎖’ 效果的工具，讓其他執行緒不能讀取現在正在使用的資源。

- 不同交易之間，無窮盡互相等待的情況稱為死結。一般死結的行程會自動 ROLLBACK。

  MySQL 可以設定 innodb_lock_wait_timeout = n 的秒數，此為最長等待時間。

  避免發生無法預測的死結，而一直等待。


User= A
Account = 0
Transaction A
Transaction B
Start
Start
SELECT * FROM user
WHERE id = 1
FOR UPDEATE;
上鎖
SELECT * FROM user
WHERE id = 2
FOR UPDEATE;
上鎖
UPDATE user set name ='MARK'
WHERE id = 2;
UPDATE user set name ='MARK'
WHERE id = 1;
Commit
Commit
死鎖！兩個 update 都在等對方事務釋放鎖

Q & A