鎖與交易

朱克剛

測試環境問題

- 相對比較沒問題的測試環境目前有兩個： mysql 文字管理介面、 Azure Data Studio（推薦）
- phpMyAdmin 屬於網頁，交易指令最後若未執行 rollback 或 commit，會自動 rollback，必須加上 do sleep(n) 來延長交易結束時間，操作不太方便
- phpMyAdmin 會自動在 SQL 查詢指令最後方加上 Limit 0, 25，導致手動上鎖指令為錯誤指令，因手動上鎖指令必須放在最後
- DBeaver 對一些指令會誤認為錯誤指令，例如 do sleep(n)，此時要改為 select sleep(n)，建議可改成以下寫法

```
select sleep(3) into @null;
```

交易

- 將多個 SQL 指令打包成一個，交易結束後，這些 SQL 指令只有全部成功或全部失敗兩種狀況
- 交易失敗後，資料庫資料要回復到交易前的狀態

```
start transaction;
   -- SQL 指令寫這

   rollback; -- 交易失敗
   commit; -- 交易成功
```

二擇一，執行後交易結束

範例

- 交易失敗。吃顆後悔藥，所有異動恢復到交易前狀態。

```
start transaction;
   delete from UserInfo where uid = 'A04';
   insert into House (address) values ('花蓮市民族路33號');
   rollback;
```

- 交易成功。資料寫入資料庫，已無後悔餘地。

```
start transaction;
   delete from UserInfo where uid = 'A04';
   insert into House (address) values ('花蓮市民族路33號');
   commit;
```

鎖的種類

鎖種類	說明
S	共享鎖
X	獨佔鎖、排他鎖
IS	意圖共享鎖
IX	意圖獨佔鎖

- 同一筆資料可以有多個 S 鎖但只能有一個 X 鎖
- S 鎖與 X 鎖互斥，也就是不可同時存在
- 當資料有 X 鎖時，MySQL 預設可以讀取，為求效率此時不上 S 鎖

觀察工具 - 1

- 只有在資料要上鎖卻上不去的時候才會查到資料

```
select * from information_schema.innodb_locks;
select * from information_schema.innodb_lock_waits;
```

結果　訊息

	lock_id	lock_trx_id	lock_mode	lock_type	lock_table	lock_index	lock_space	lock_pag
1	27503:104:3:19	27503	X	RECORD	`addressbook`.`userinfo`	PRIMARY	104	3
2	27481:104:3:19	27481	X	RECORD	`addressbook`.`userinfo`	PRIMARY	104	3

	requesting_trx_id	requested_lock_id	blocking_trx_id	blocking_lock_id
1	27503	27503:104:3:19	27481	27481:104:3:19

觀察工具 – 2

- 必須在終端機中使用 mysql 連進資料庫，然後執行

```
show engine innodb status\G
```

- 觀察

```
------------
TRANSACTIONS
------------
Trx id counter 27504
Purge done for trx's n:o < 27470 undo n:o < 0 state: running but idle
History list length 1
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 421821907214536, not started
0 lock struct(s), heap size 1128, 0 row lock(s)
---TRANSACTION 27481, ACTIVE 397 sec
2 lock struct(s), heap size 1128, 1 row lock(s), undo log entries 1
MySQL thread id 9, OS thread handle 13092745216, query id 20 localhost root
```

測試一

- 開兩個終端機使用 mysql 指令連進資料庫
- 終端機一執行下列指令

```
start transaction;
  delete from UserInfo where uid = 'A02';
  do sleep(10);
  rollback;
```

- 終端機二執行下列指令

```
update UserInfo set cname = null where uid = 'A02';
select * from UserInfo where uid = 'A02';
```

- 結果：終端機二指令被阻擋，因為終端機一的交易未結束
  - *此時下指令可以觀察到資料出現 X 鎖*

測試二

- 開兩個終端機使用 mysql 指令連進資料庫
- 終端機一執行下列指令

```
update UserInfo set cname = '李大媽' where uid = 'A02';
start transaction;
   update UserInfo set cname = 'David' where uid = 'A02';
   do sleep(10);
   commit;
```

- 終端機二執行下列指令

```
select * from UserInfo where uid = 'A02';
```

- 結果：MySQL 為求查詢效率，終端機二的查詢不會上 S 鎖，並且查詢結果為已 commit 資料，也就是李大媽

超賣問題 – 通用解法

- 當多人要同時修改同一筆資料，並且該資料會透過判斷式來決定是否修改時，必須使用交易，例如商品有數量限制時。
- 總共只有10樣東西可賣，但最後發現賣超過數量了
- 原因在於「取出數量」、「檢查數量」、「決定賣出」為三個獨立程序，在多工環境下產生的必然現象
- 解決方式：將這三個程序合併成一個程序，並且產生臨界區間

注意效能問題

- 臨界區間一次只能服務一個人，資料庫效能再強都沒用
- 臨界區間從誕生到消失的持續時間越短越好

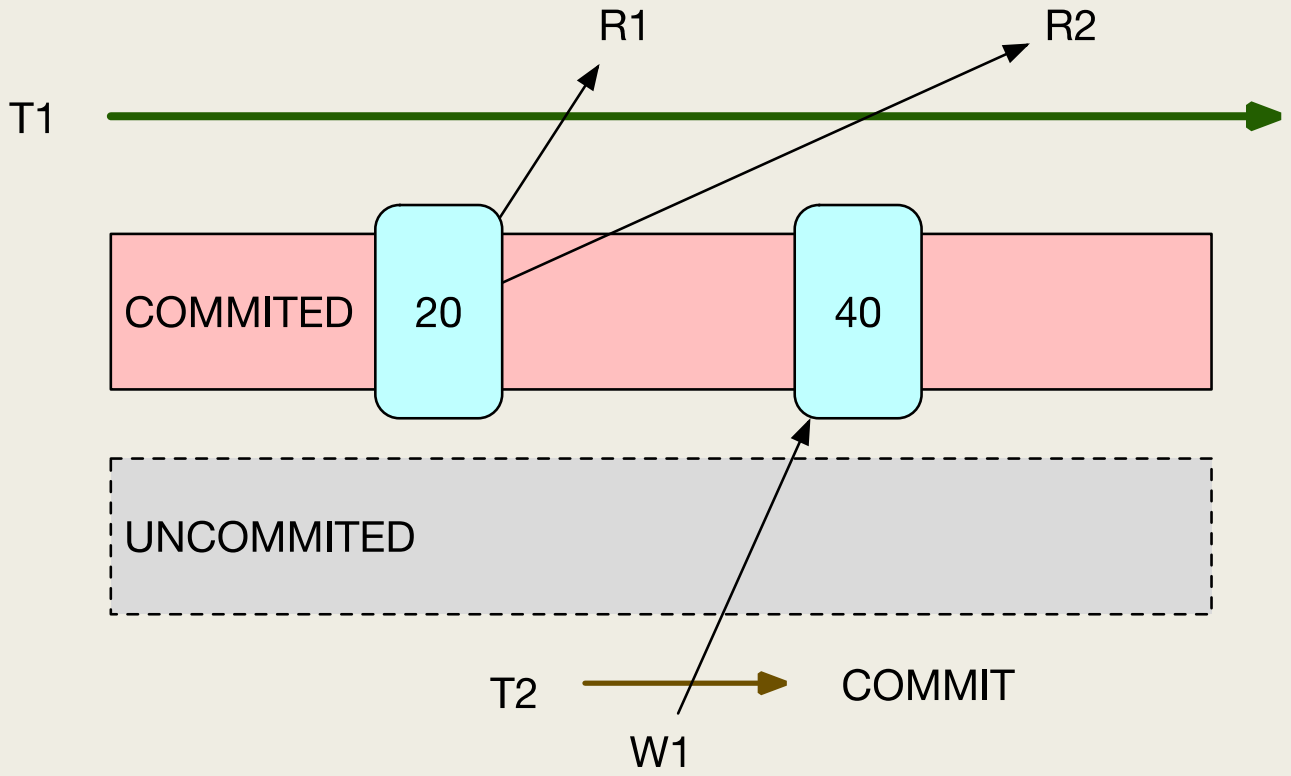
交易需滿足 ACID

- 交易必須滿足資料一致性要求（ACID）
- Atomicity：原子性
  - *在交易中的各個異動指令算一個，不可再分割，要就全部成功不然全部失敗*
- Consistency：一致性
  - 資料在交易前後必須滿足設定的條件，如果不滿足交易就必須失敗
  - 例如商品數量總和必須等於庫存 + 賣出數量，而且庫存不可為負
- Isolation：隔離性
  - *各個交易在自己的 domain 中執行，不會互相干擾*
- Durability：持久性
  - *交易一旦 commit，資料寫入資料庫，無法再後悔*

隔離等級

- REPEATABLE READ：預設等級
  - *確保在交易中的多次讀取都是同樣的結果*
- READ COMMITTED
  - *只會讀到已經 commit 的資料*
  - *uncommit 的資料會忽略，不會造成阻塞*
- READ UNCOMMITTED
  - *可讀到另一交易已修改但尚未 commit 的資料，造成髒讀取*
- SERIALIZABLE
  - *確保一群交易依序執行，不會多個交易同時執行導致讀寫交錯*
  - *因為需要更多的鎖來控制執行順序，因此會影響效率*

REPEATABLE READ



REPEATABLE READ - 測試

- T1 執行下列指令

```
update UserInfo set cname = null where uid = 'A06';
set transaction isolation level REPEATABLE READ;
start transaction;
   select *, 'R1' from UserInfo where uid = 'A06';
   do sleep(10);
   select *, 'R2' from UserInfo where uid = 'A06';
   commit;
```

NULL

NULL

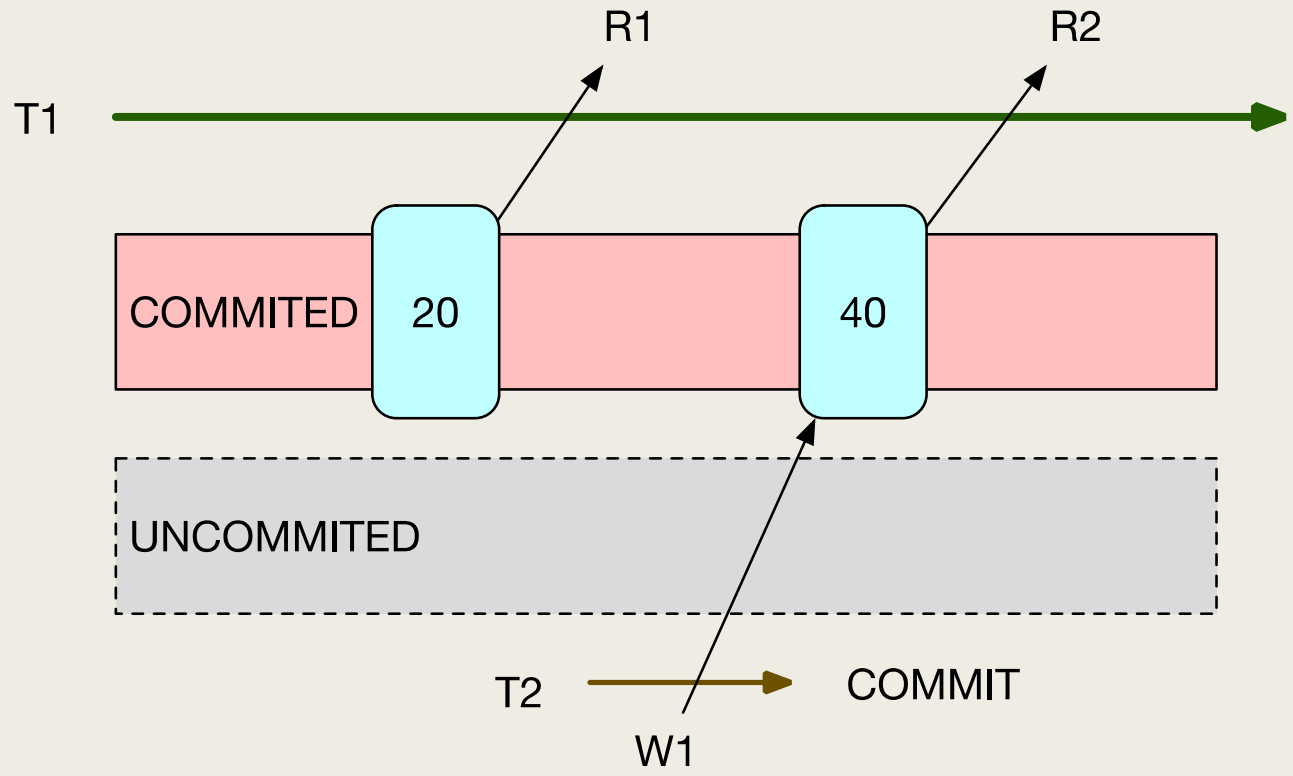
- 在 T1 交易狀態為 sleep 時，執行 T2 指令

在這個時間點執行

```
update UserInfo set cname = '黃小芳' where uid = 'A06'
```

- 結果：T1 的兩次查詢結果**都一樣**

READ COMMITTED



READ COMMITTED - 測試

- T1 執行下列指令

```
update UserInfo set cname = null where uid = 'A06';
set transaction isolation level READ COMMITTED;
start transaction;
   select *, 'R1' from UserInfo where uid = 'A06';
   do sleep(10);
   select *, 'R2' from UserInfo where uid = 'A06';
   commit;
```

NULL

黃小芳

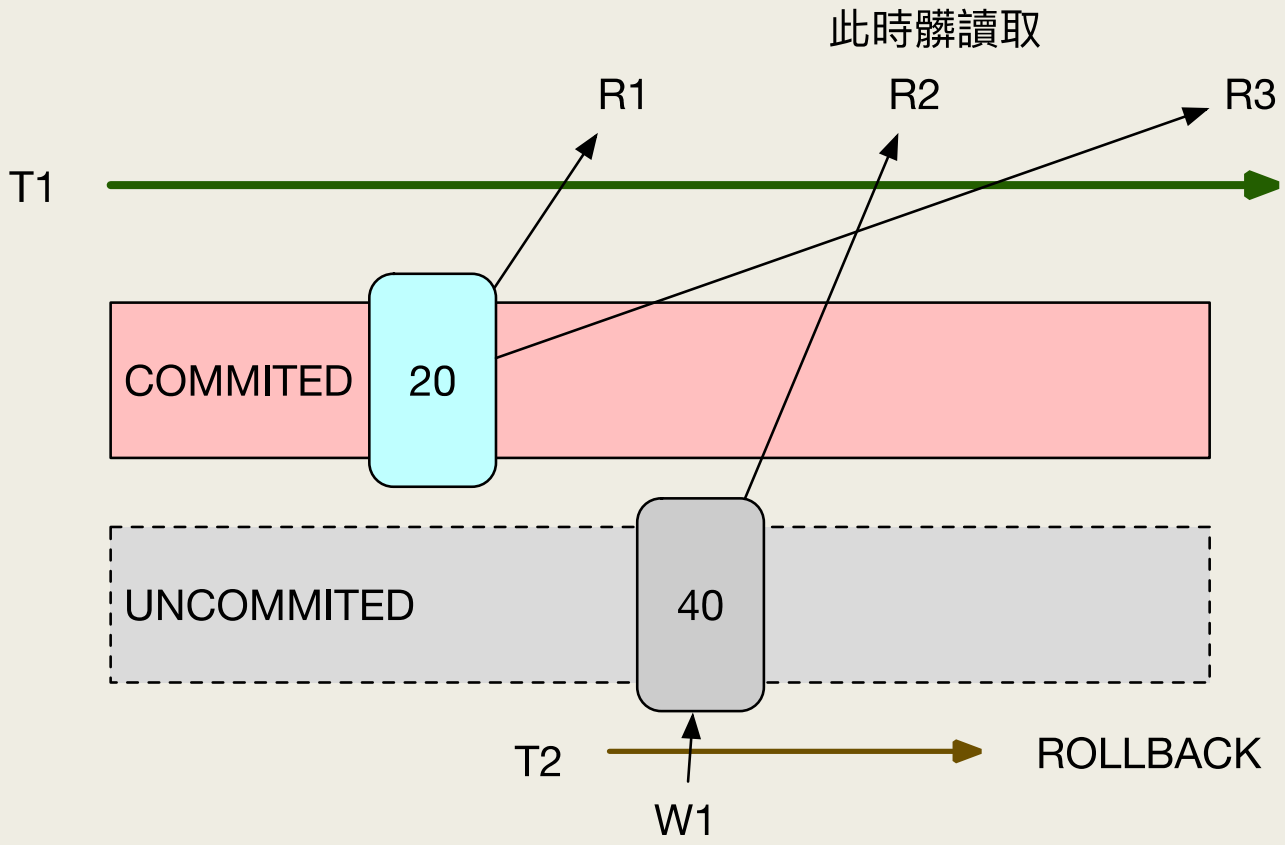
- 在 T1 交易狀態為 sleep 時，執行 T2 指令

```
update UserInfo set cname = '黃小芳' where uid = 'A06'
```

在這個時間點執行

- 結果：T1 的 R2 查詢結果是**黃小芳**

READ UNCOMMITTED



READ UNCOMMITTED - 測試

- T1 執行下列指令

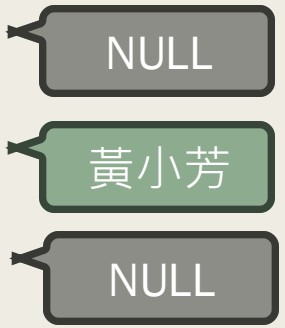
```
update UserInfo set cname = null where uid = 'A06';
set transaction isolation level READ UNCOMMITTED;
start transaction;
   select *, 'R1' from UserInfo where uid = 'A06';
   do sleep(10);
   select *, 'R2' from UserInfo where uid = 'A06';
   do sleep(10);
   select *, 'R3' from UserInfo where uid = 'A06';
   commit;
```

- 在 T1 交易狀態為 sleep 時，執行 T2 指令

```
start transaction;
   update UserInfo set cname = '黃小芳' where uid like 'A06';
   do sleep(15);
   rollback;
```

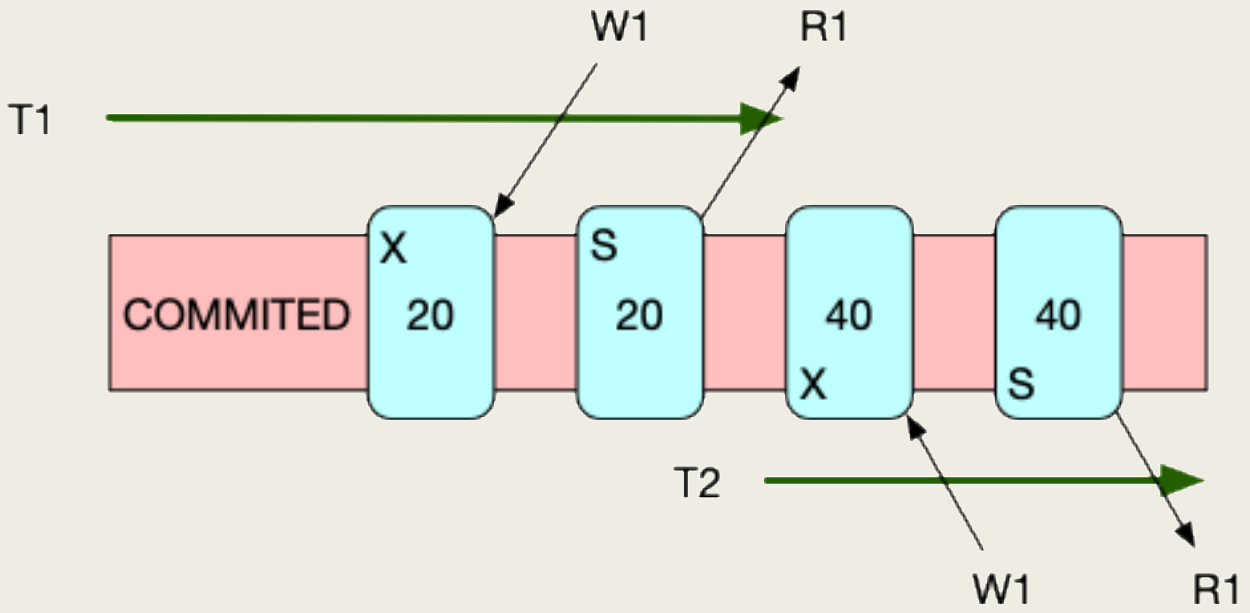
在這個時間點執行

- 結果：T1 的 R2 查詢結果是**黃小芳 <= 髒讀取**



SERIALIZABLE

- 透過各種鎖，確保各交易間讀寫不交錯，若是讀取資料也會上S鎖



SERIALIZABLE – 測試

- T1 執行下列指令

```
set transaction isolation level SERIALIZABLE;
start transaction;
   select * from UserInfo where uid = 'A06';
   do sleep(20);
   select 'done';
   commit;
```

這裡可以觀察到 A06 出現 S 鎖，持續到交易結束

- 在 T1 的交易狀態為 sleep 時，T2 執行下列指令

```
start transaction;
   update UserInfo set cname = 'David' where uid = 'A06';
   select 'done';
   commit;
```

- 結果：T2 的 update 指令會被阻擋，直到 T1 的交易結束後才執行，確保兩個交易不會讀寫交錯

強制手動上鎖

若要在 phpMyAdmin 中執行手動上鎖指令，必須在指令前自己加上 limit 指令，否則 phpMyAdmin 會在 SQL 最後加上 limit 0, 25 導致語法錯誤，例如

```
select uid, cname
from UserInfo where uid = 'A01'
limit 0, 25
for update
```

- 上 S 鎖

```
start transaction;
   select * from UserInfo lock in share mode;
```

- 上 X 鎖

```
start transaction;
   select * from UserInfo for update;
```

超賣的 MySQL 解法

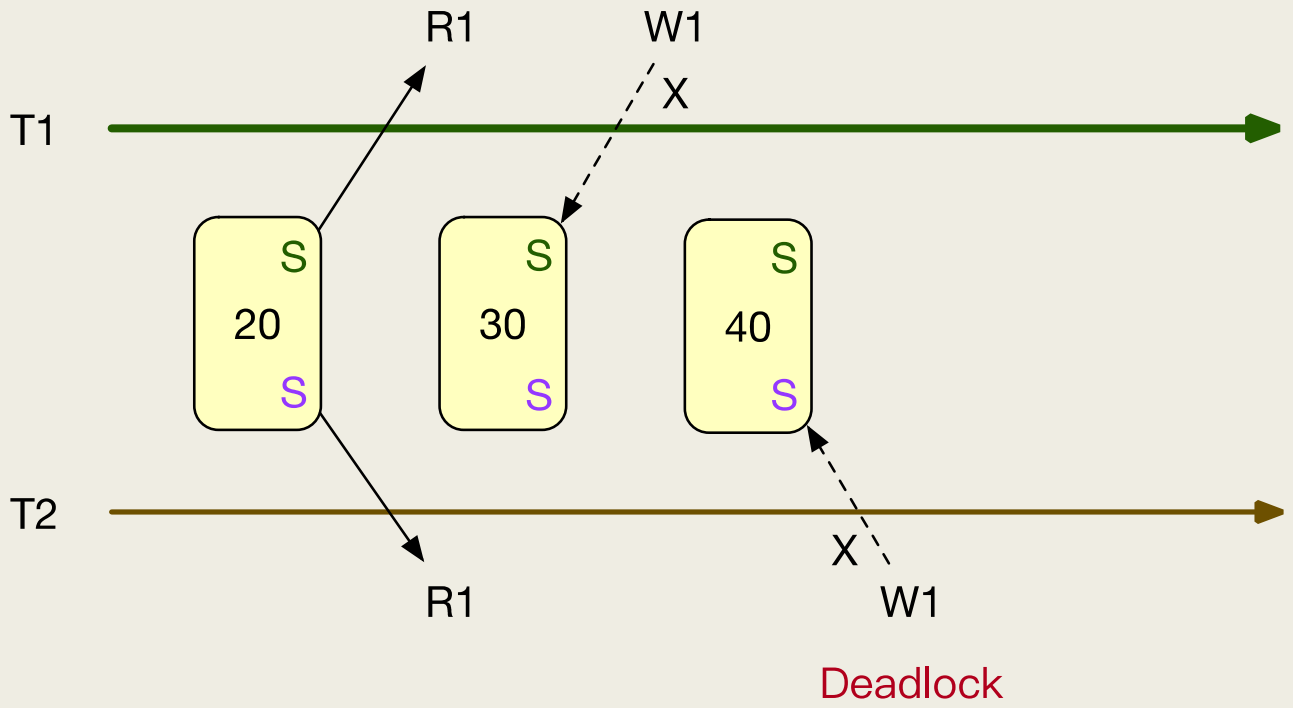
- 如果還是希望順序是讀取、判斷、修改時，要在讀取階段就上 X 鎖

```
start transaction;
  select * from UserInfo where uid = 'A04' for update;
```

- 注意 MySQL 的超賣不要使用 serializable 的隔離等級來解。因為超賣問題讀寫都是同一個欄位，這時當第一個交易未完成時，其餘交易會出現死結，導致第二個以後的交易因死結全部 rollback

死結

- 在 T1 的 W1 時，需要等 T2 commit 才能解鎖 T2 上的 S 鎖
- 在 T2 的 W1 時，需要等 T1 commit 才能解鎖 T1 上的 S 鎖
- 此時 T1、T2 進入互相等待狀態，形成死結



處理堵塞

- 找到哪一個程序造成堵塞

```
show processlist;
```

- 下指令刪掉他，刪掉的指令會產生交易失敗

```
kill Id
```

鎖定時間

- 查詢上鎖的逾時時間（預設50秒）

```
show variables like 'innodb_lock_wait_timeout';
```

- 設定鎖定逾時時間，單位秒

```
set innodb_lock_wait_timeout=5;
```

- 目的：避免出現死結狀態