

鎖與交易

朱克剛



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

```
begin transaction;  
-- SQL 指令寫這
```

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

鎖的種類

鎖種類	說明
S	共享鎖
X	獨佔鎖、排他鎖
U	修改鎖。修改資料前上U鎖，實際寫入時改X鎖
IS	意圖共享鎖
IX	意圖獨佔鎖
IU	意圖修改鎖
...	https://learn.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/sp-lock-transact-sql?view=sql-server-ver16

觀察工具

```
CREATE VIEW lockinfo AS
```

```
SELECT
```

```
    object_name(b.object_id) as TableName,  
    resource_type,  
    request_mode,  
    request_session_id
```

```
FROM sys.dm_tran_locks as a join sys.partitions as b  
    on a.resource_associated_entity_id = b.hobt_id
```

測試

- 開兩個頁籤
- 第一個頁籤執行下列指令

```
begin transaction
```

```
update UserInfo set cname = '珠小妹' where uid = 'A04';
```

- 第二個頁籤執行下列指令

```
select * from UserInfo
```

- 結果：第二個頁籤的指令被阻擋，因為第一個頁籤的交易未結束
 - 此時下指令可以觀察到資料出現X 鎖

處理堵塞

- 找到哪一個 request session id 造成堵塞
- 下指令刪掉他，刪掉的指令會產生交易失敗

kill SPID

鎖定時間

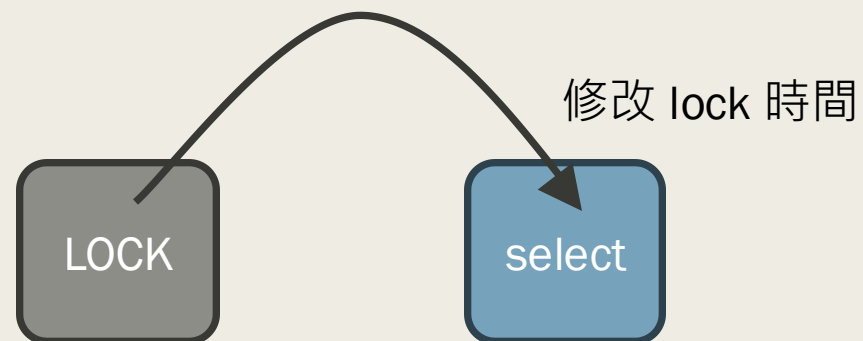
- 查詢鎖定時間

```
print @@lock_timeout
```

- 設定鎖定逾時時間，單位毫秒

```
set lock_timeout 5000
```

- 目的：避免出現死結狀態



範例

- 第一個頁籤執行下列指令

```
begin transaction  
  update UserInfo set cname = '珠小妹' where uid = 'A04';
```

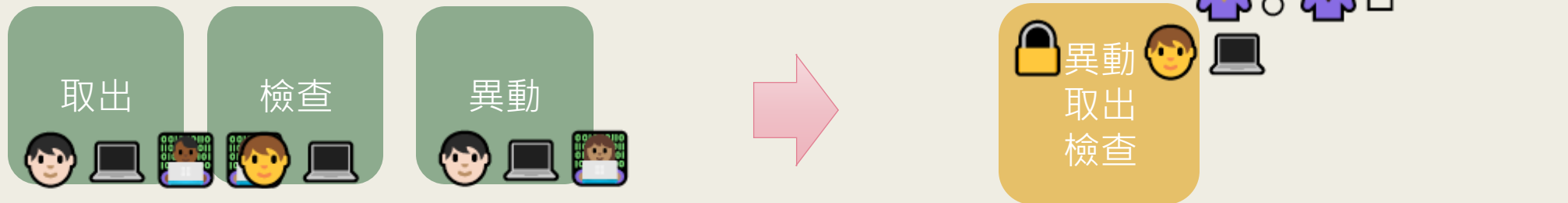
- 第二個頁籤執行下列指令

```
set lock_timeout 5000  
begin try  
  select * from UserInfo  
end try  
begin catch  
  
end catch
```

讀到出現錯誤為止
例如：A04以後都不會出現

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
- 不要過度膨脹臨界區間範圍，例如為了幾筆資料卻把整個資料表鎖住
- 查詢範圍必須避開臨界區間，否則原本很快的查詢也會變的很慢

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*，資料就「落盤」不會不見

隔離等級

髒讀取：讀到另外一個交易已更動但尚未 commit 的資料

- **READ UNCOMMITTED**：不管資料有沒有上鎖都可以讀取，但可能髒讀取
`set transaction isolation level READ UNCOMMITTED`
- **READ COMMITTED** (SQL Server 預設)
 - 確保讀到的資料都是已確認的
- **REPEATABLE READ**
 - 交易過程中會對讀取資料上 S 鎖，直到交易結束才解鎖，確保交易過程中多次查詢的資料結果都是一致的
- **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
- **SNAPSHOT**：必須開啟後才能使用
`ALTER DATABASE AddressBook SET ALLOW_SNAPSHOT_ISOLATION ON`
`ALTER DATABASE AddressBook SET READ_COMMITTED_SNAPSHOT ON`
- 文件：<https://learn.microsoft.com/zh-tw/sql/t-sql/statements/set-transaction-isolation-level-transact-sql?view=sql-server-ver16>

手動設定鎖 – SQL Server 專有

■ NOLOCK

- 讀取資料時不設定S鎖，若此時資料已經有X鎖，照樣可以讀取，目的是增加讀取效率，但可能髒讀取
- 等同隔離等級中的READ UNCOMMITTED

■ READPAST

- 跟NOLOCK類似但讀取到的資料不包含已上鎖資料，因此不會髒讀取，但資料會少

■ UPDLOCK / XLOCK

- 讀取資料時可對資料上修改鎖或獨佔鎖，直到交易結束

■ ROWLOCK

- 某些情況為避免觸發更上層鎖造成效率銳減，因此可設定僅對行上鎖

髒讀取

- 在第一個頁籤執行

```
begin transaction
```

```
update UserInfo set cname = '珠小妹' where uid = 'A04';
```

- 在第二個頁籤執行

```
select * from UserInfo with (nolock)
```

或

```
set transaction isolation level READ UNCOMMITTED
```

```
select * from UserInfo
```

- 結果：讀到已經修改但還沒 commit 的資料

4	A04	珠小妹
---	-----	-----

死結 1/3

- 互相等對方解鎖時就形成死結
- 首先建立兩個測試用資料表

```
drop table if exists TAA  
drop table if exists TBB
```

```
select 1 as id, 0 as n into TAA  
select 1 as id, 0 as n into TBB
```

死結 2/3

- 在第一個頁籤執行，先 TAA 再 TBB

begin TRAN

update TAA set n = n + 1 where id = 1

waitfor delay '00:00:05'

update TBB set n = n + 1 where id = 1

commit

- 在第二個頁籤執行，先 TBB 再 TAA

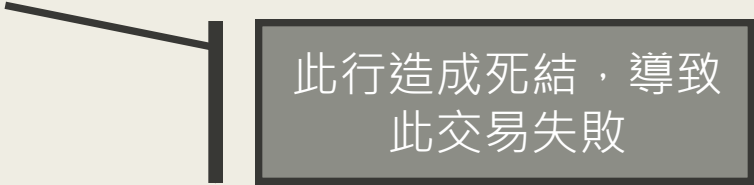
begin TRAN

update TBB set n = n + 1 where id = 1

waitfor delay '00:00:05'

update TAA set n = n + 1 where id = 1

commit



此行造成死結，導致
此交易失敗

死結 3/3

- 手動上 U 鎖
- 在兩個頁籤的交易開始時都對資料行加上 U 鎖

```
begin TRAN
  select * from TAA with (updlock) where id = 1
  select * from TBB with (updlock) where id = 1
```

- 兩個頁籤都執行完畢後，應該 n 的值是 2

	id	n
1	1	2

超賣問題 – SQL Server 專屬作法

```
create proc p_proc as
BEGIN
  declare @time varchar(20)
  declare @value int
  set nocount on
  set @time = concat('00:00:00.', floor(rand() * 1000))

  begin tran
    select @value = value from Test with (updlock) where id = 1
    if @value >= 10
    begin
      rollback
      return 0
    end

    waitfor delay @time
    update Test set value = value + 1 where id = 1
    commit
    return 1
  end tran
END
```

	id	value
1	1	0

放在資料表後面

延遲隨機秒數