第12章 交易管理與並行控制

12-1 交易的基础

- 12-1-1 交易状态
- 12-1-2 交易的四大特性

12-1 交易的基礎-說明

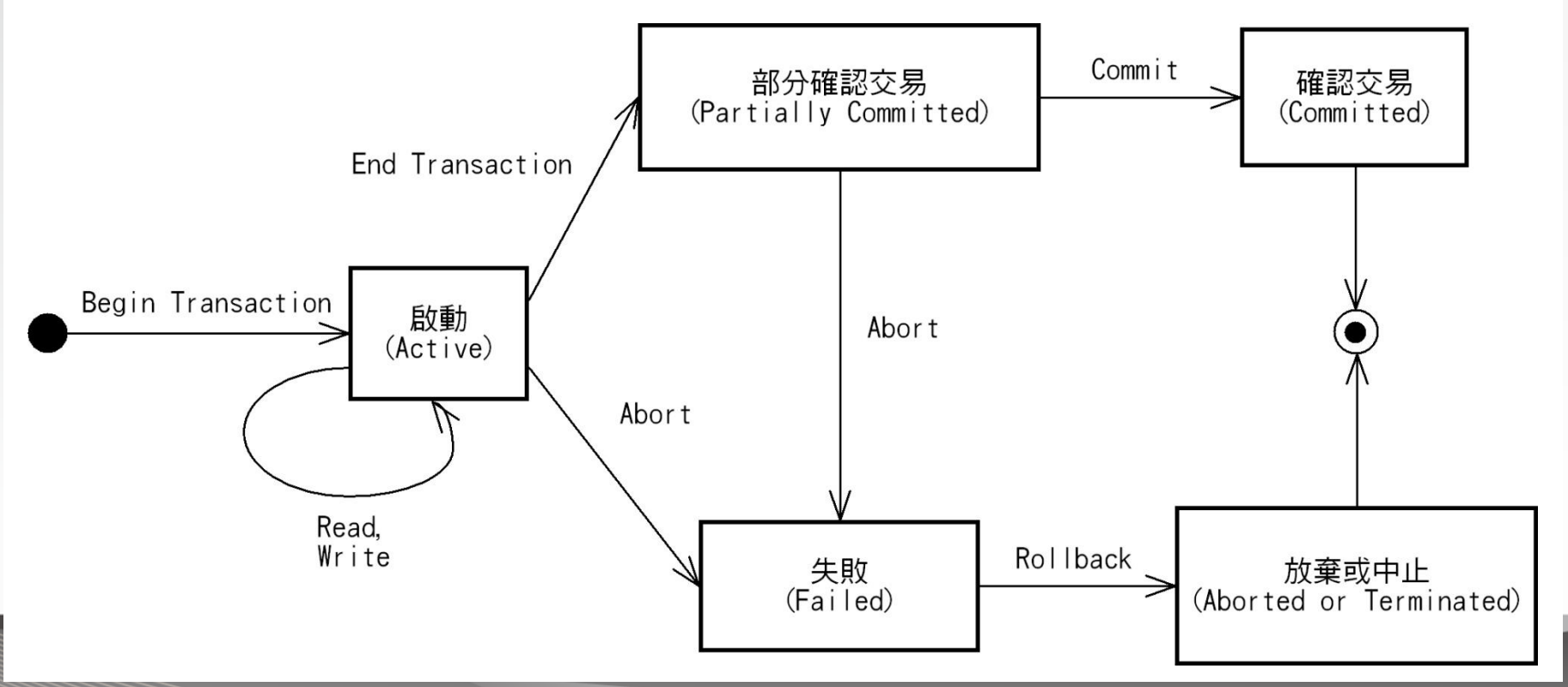
- 一般來說，資料庫系統最主要的操作是存取資料庫中儲存的資料，如果有多個存取操作需要執行，而且這些操作是無法分割的單位，則整個操作過程，對於資料庫系統來說是一個「交易」（Transaction），或譯成「異動」，在本書將統一使用「交易」來說明。
- 交易是將多個資料庫單元操作視為同一個不可分割的邏輯單元，這些資料庫單元的操作，只有兩種結果：一種是全部執行完成；另一種就是通通不執行，而且不可能有執行一半的情形發生。

12-1 交易的基礎-單元操作

- 交易是將資料庫單元操作的集合視為一個不可分割的邏輯單位（Logical Unit），然後使用並行控制（Concurrency Control）和回復處理（Recovery）機制來保障交易執行成功。
- 資料庫的單元操作只有兩種，如下所示：
  - **讀取（Read）**：從資料庫讀取資料。
  - **寫入（Write）**：將資料寫入資料庫。

12-1-1 交易狀態-圖例

- 資料庫管理系統執行整個交易的過程可以分成數種交易狀態（Transaction State），如下圖所示：



12-1-1 交易狀態-種類1

- 啟動狀態（Active State）：當交易開始執行時，就是進入啟動狀態的初始狀態，依序執行交易的讀取或寫入等資料庫單元操作。
- 部分確認交易狀態（Partially Committed State）：當交易的最後一個資料庫單元操作執行完後，也就是交易結束，就進入部分確認交易狀態。
- 確認交易狀態（Committed State）：在成功完成交易進入部分確認交易狀態後，還需要確認系統沒有錯誤，可以真正將資料寫入資料庫。在確認沒有錯誤後，就可以進入確認交易狀態，表示交易造成的資料庫更改，將真正寫入資料庫，而且不會再取消更改。

12-1-1 交易狀態-種類2

- 失敗狀態（Failed State）：當發現交易不能繼續執行下去時，交易就進入失敗狀態，準備執行回復交易。
- 放棄或中止狀態（Aborted or Terminated State）：交易需要回復到交易前的狀態，在取消所有寫入資料庫單元操作影響的資料後，就進入此狀態。簡單的說，資料庫管理系統如同根本沒有執行過此交易。

12-1-1 交易狀態-交易停止執行的原因1

- 在資料庫管理系統執行交易的過程中，共有三種情況會停止交易的執行，如下所示：

交易成功

- 交易成功就是正常結束交易的執行，它是指交易的資料庫單元操作全部執行完成。以交易狀態來說，如果交易從啟動狀態開始，可以到達確認交易狀態，就表示交易成功。

12-1-1 交易狀態-交易停止執行的原因2

交易失敗

- 交易失敗是送出放棄指令（Abort或Rollback）來結束交易的執行。以交易狀態來說，就是到達放棄或中止狀態。交易失敗分為兩種，如下：
  - **放棄交易**：交易本身因為條件錯誤、輸入錯誤資料或使用者操作而送出放棄指令（Abort或Rollback）來放棄交易的執行，正確的說，此時的交易是進入放棄狀態。
  - **中止交易**：因為系統負載問題或死結（Deadlock）情況，由資料庫管理系統送出放棄指令，讓交易進入中止狀態。

12-1-1 交易狀態-交易停止執行的原因3

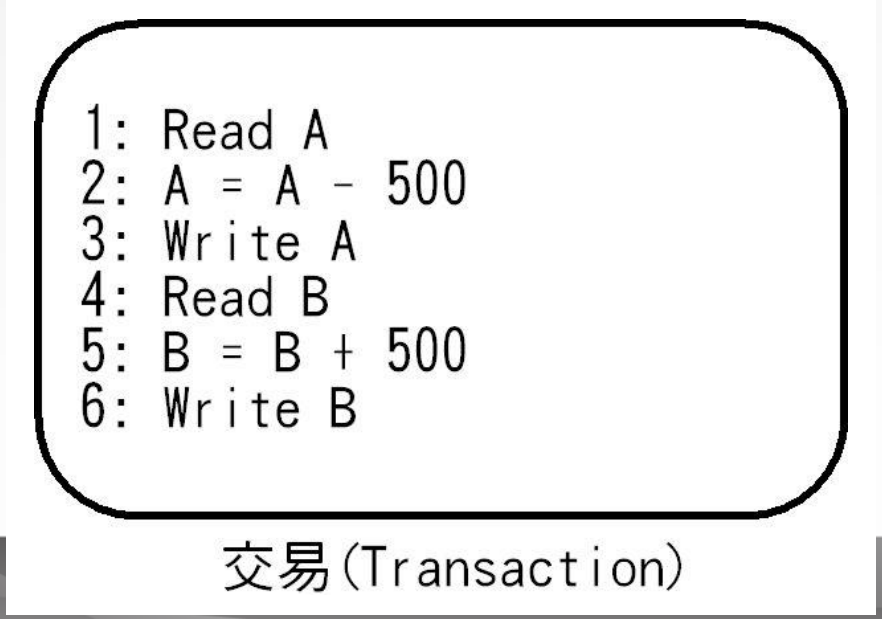
交易未完成

- 交易有可能因為系統錯誤、硬體錯誤或當機而停止交易的執行，因為沒有送出放棄指令，此時交易是尚未完成的中斷狀態，即只執行到一半就被迫中斷執行。
- 因為資料庫管理系統並不允許此情況發生，所以在重新啟動後，其回復處理（Recovery）機制會從中斷點開始，重新執行交易至交易成功或失敗來結束交易的執行。

12-1-2 交易的四大特性1

單元性（Atomicity）

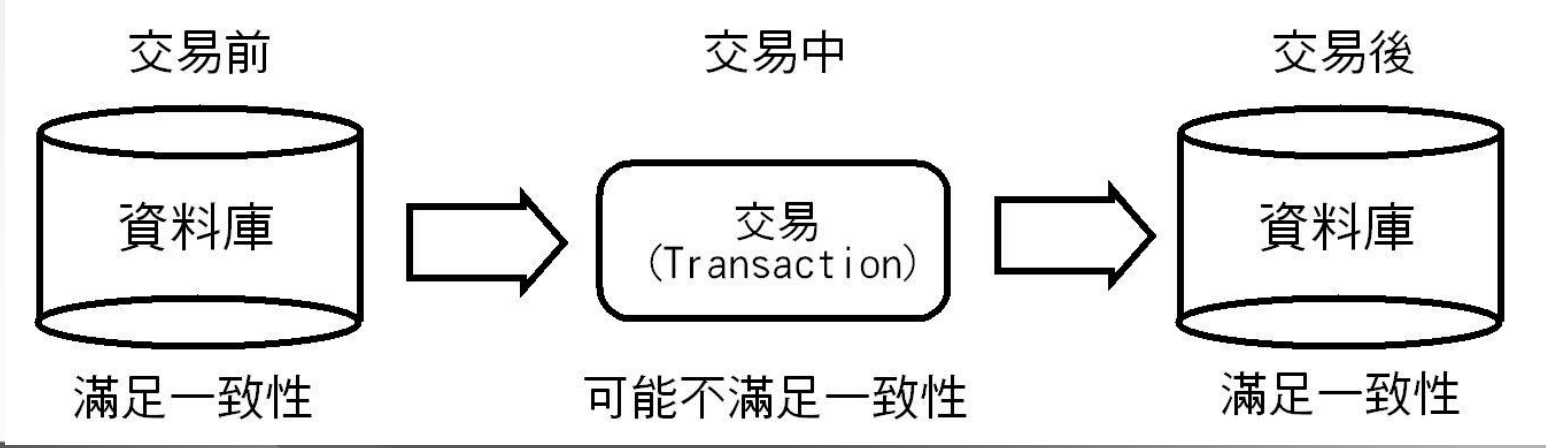
- 單元性是將交易過程的所有資料庫單元操作視為同一項工作，不是全部執行完，就是通通不執行，這是一種不能分割的邏輯單位，如下圖所示：



12-1-2 交易的四大特性2

一致性（Consistency）

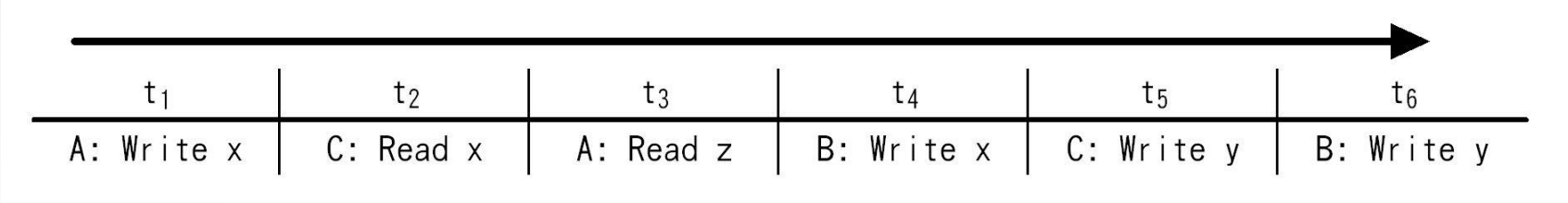
- 一致性是指交易可能會更改或更新資料庫的資料，不過在交易之前和之後，資料庫的資料仍然需要滿足完整性限制條件，維持資料的一致性，如下圖所示：



12-1-2 交易的四大特性3

隔離性（Isolation）

- 隔離性是指在執行多個交易時，雖然各交易是並行執行，不過各交易之間應該滿足獨立性，也就是說，一個交易不會影響到其他交易的執行結果，或被其他交易所干擾，如下圖所示：



12-1-2 交易的四大特性4

永久性（Durability）

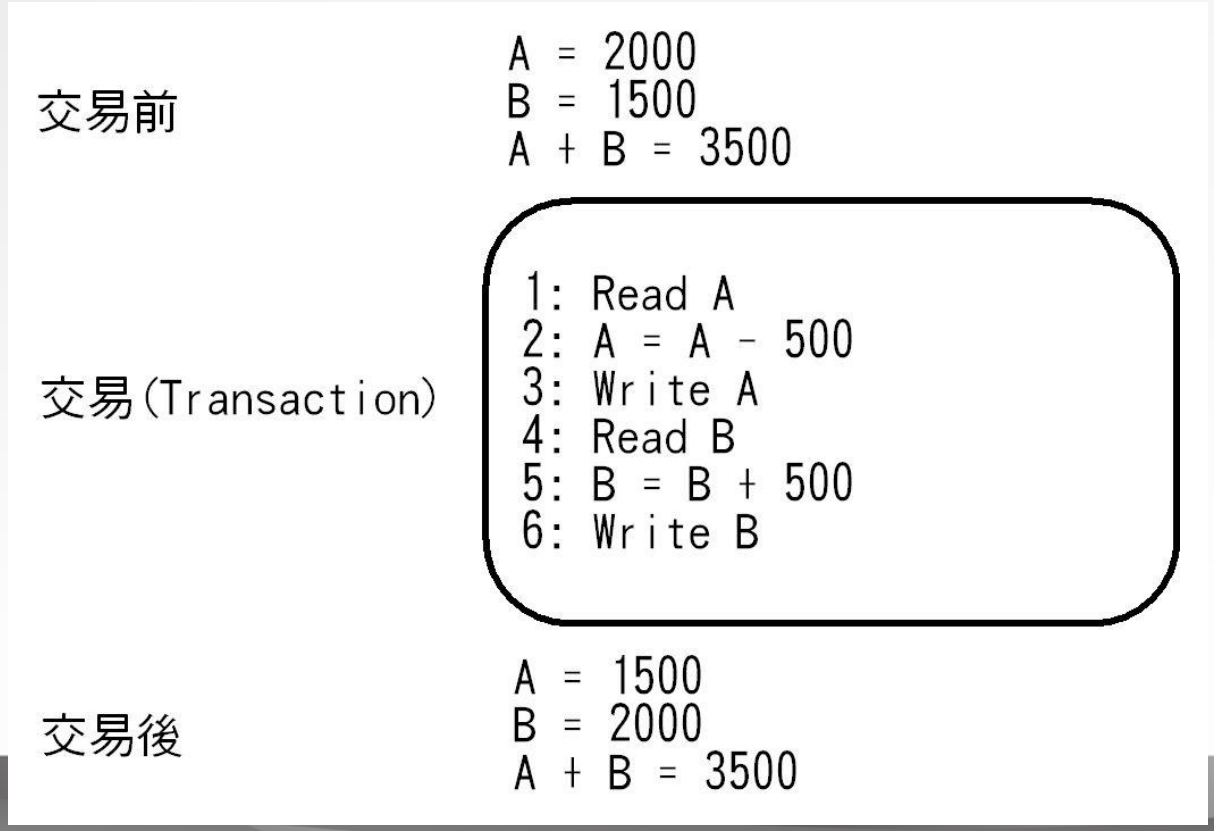
- 永久性（Durability）是指當交易完成執行確認交易（Commit）後，其執行操作所更動的資料已經永久改變，資料庫管理系統不只需要將資料從資料庫緩衝區實際寫入儲存裝置，而且不會因任何錯誤，而導致資料的流失。

12-2 資料庫管理系統的交易管理

- 12-2-1 交易管理的基礎
- 12-2-2 SQL語言的交易支援

12-2-1 交易管理的基礎-說明

- 資料庫管理系統的交易管理在執行交易時，需要滿足四大特性。



12-2-1 交易管理的基礎-需求1

一致性需求（Consistency Requirement）

- 資料庫管理系統需要維持資料庫資料的一致性，同樣的，交易管理也擁有一致性的需求，所以，執行交易前帳戶A和B的總和是3500元，在執行完交易後，帳戶A和B的總和還是3500元，在交易前後的帳戶總額並沒有改變。

12-2-1 交易管理的基礎-需求2

單元性需求（Atomicity Requirement）

- 如果在交易第4步驟Read B的讀取操作發生錯誤，交易管理需要避免第3步驟Write A的資料庫寫入操作，並不會真正寫入資料庫，因為資料庫單元操作沒有全部執行，就都不能執行。

12-2-1 交易管理的基礎-需求3

永久性需求（Durability Requirement）

- 當交易到達部分確認交易狀態（Partially Committed State），即當使用者送出COMMIT指令確認交易後，在使用者的認知上，帳戶A已經將500元轉帳到帳戶B。

12-2-1 交易管理的基礎-需求4

隔離性需求（Isolation Requirement）

- 如果在交易第3~6步驟時，交易管理允許其他交易存取帳戶A的餘額，從帳戶A提款200元，其資料庫單元操作，如下所示：

  Read A

  A = A -200

  Write A

- 上述操作是在第6步驟前執行完，因為存取了資料庫中不一致的資料，所以，最後A+B的帳戶總和是1300+2000 = 3300元，比實際的結果少。

12-2-2 SQL語言的交易支援-ANSI-SQL(Commit)

- ANSI-SQL的交易並沒有提供明確指令指出交易的開始，在交易執行的控制方面提供：COMMIT和ROLLBACK兩個指令，如下所示：
  - 確認交易（Commit）：如果交易執行過程沒有錯誤，下達COMMIT指令，將交易更改的資料實際寫入資料庫，以便執行下一個交易，如下所示：

```
DELETE FROM Students WHERE sid = 'S001'
//
COMMIT
```

12-2-2 SQL語言的交易支援-ANSI-SQL(Rollback)

- 回復交易（Rollback）：如果交易執行過程有錯誤，就是下達ROLLBACK指令放棄交易，並將資料庫回復到交易前狀態，如下所示：

```
UPDATE Students SET birthday='1968-09-12', GPA=3.0
WHERE sid = 'S001'
//
ROLLBACK
```

12-2-2 SQL語言的交易支援-Transact-SQL

- 在Transact-SQL的交易是使用BEGIN TRAN指令開始，如果交易成功，就使用確認交易COMMIT TRAN指令結束，如下所示：

```
BEGIN TRAN
……
COMMIT TRAN
```

- 如果交易失敗，回復交易是使用ROLLBACK TRAN指令結束，如下所示：

```
BEGIN TRAN
……
ROLLBACK TRAN
```

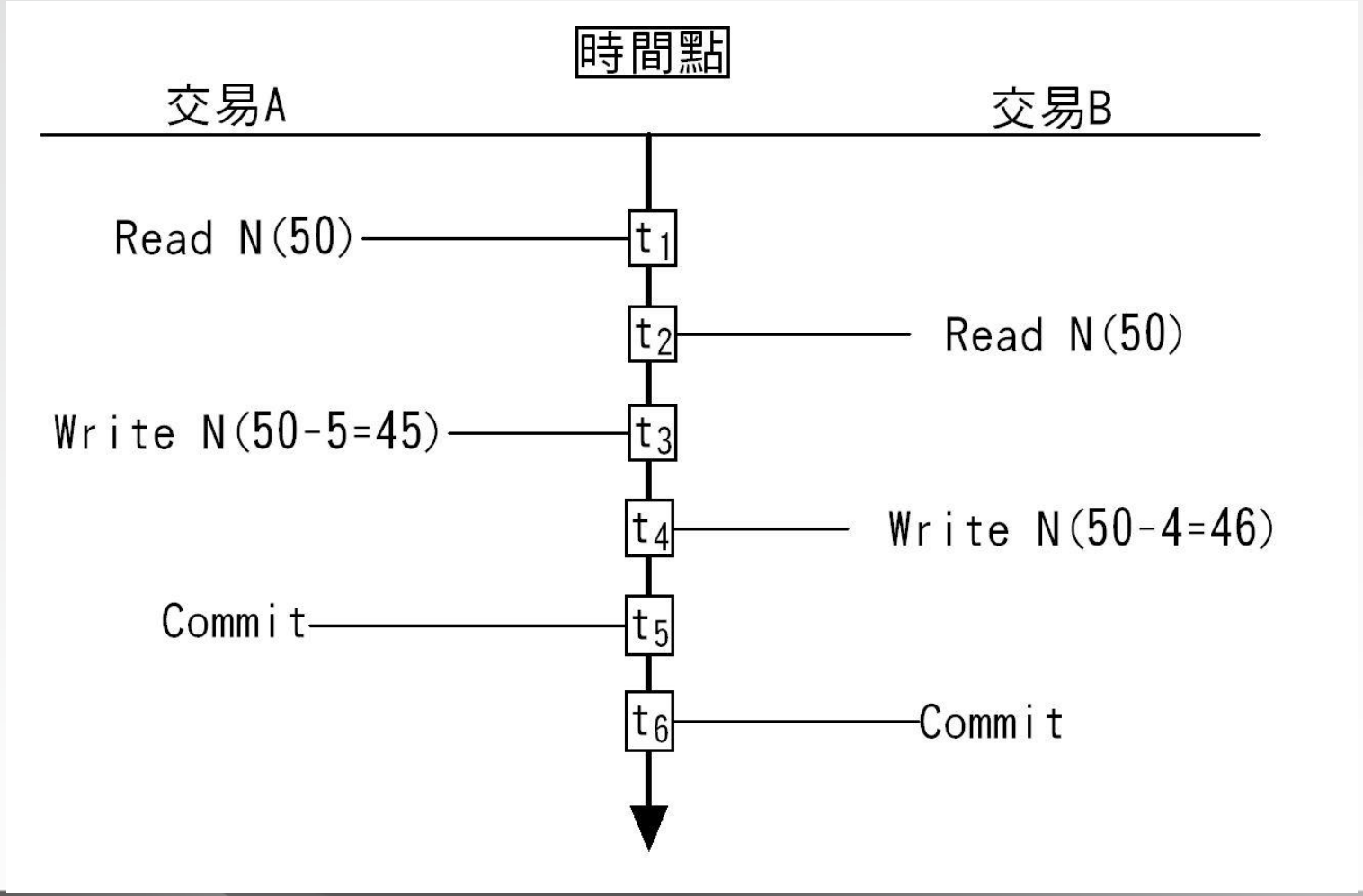
12-3 並行控制的三種問題

- 12-3-1 遺失更新問題
- 12-3-2 未確認交易相依問題
- 12-3-3 不一致分析問題

12-3-1 遺失更新問題-說明

- 遺失更新（Lost Update）問題是指交易已經更新的資料被另一個交易覆寫，換句話說，整個交易等於白忙一場。
- 例如：交易A和B同時存取飛機訂位資料庫航班編號CI101的機位數，目前機位數尚餘50個，交易A希望預訂5個機位，交易B預訂4個機位。
- 最後飛機訂位資料庫的機位數是46個，交易A等於沒有執行，因為交易A更新的機位數已經被交易B覆寫。

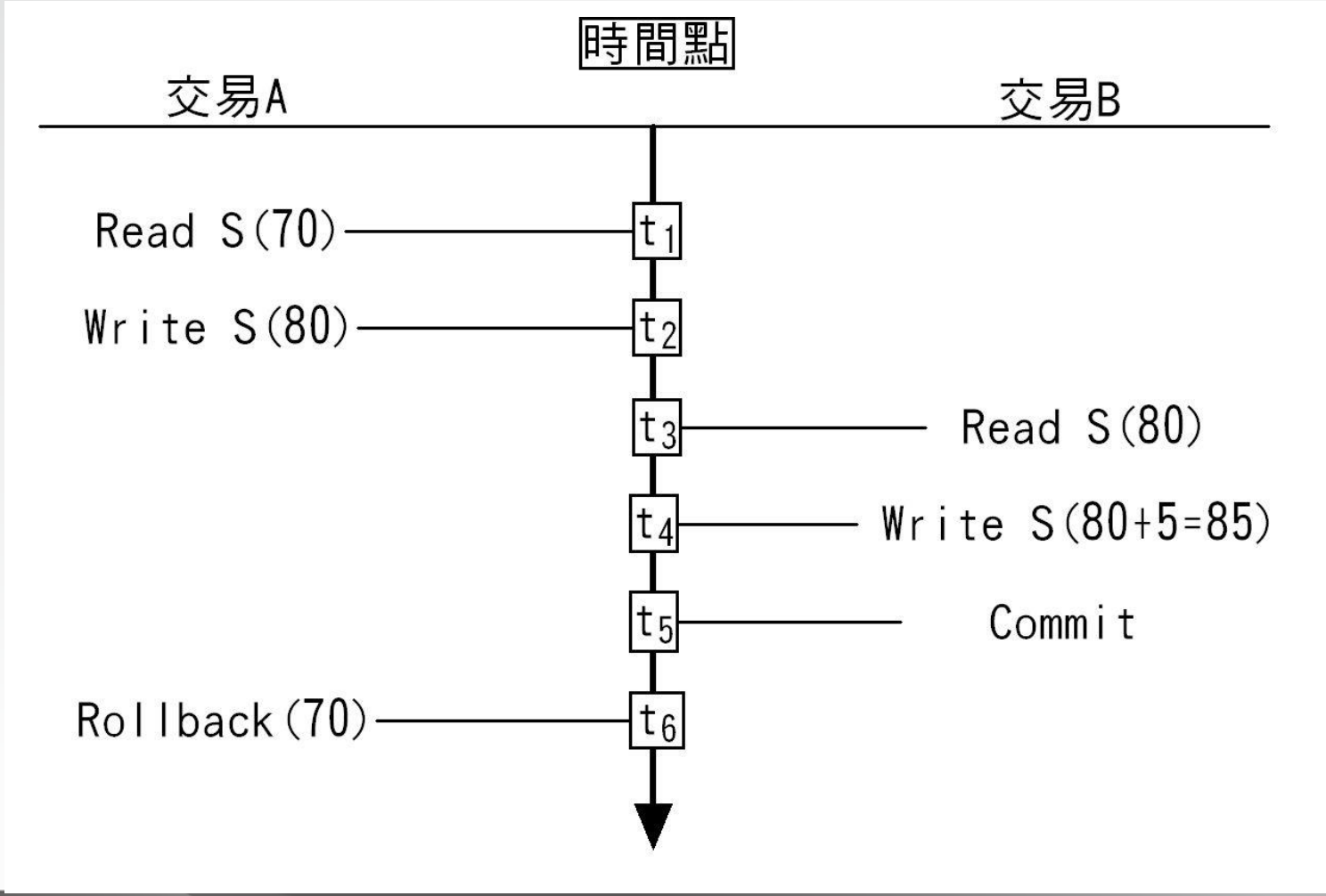
12-3-1 遺失更新問題-圖例



12-3-2 未確認交易相依問題-說明

- 未確認交易相依（Uncommitted Dependency）問題是指存取已經被另一個交易更新，但尚未確認交易的中間結果資料。
- 例如：交易A和B存取同一筆學生的記錄資料，交易A因為成績登記錯誤，需要從70分改為80分，交易B因為題目出錯，整班每位學生的成績都加5分。
- 因為交易B讀取的是尚未確認交易的中間結果資料，雖然交易B認為已完成交易，但事實，最後學生的成績不但沒有加5分，且還原到最原始的70分。

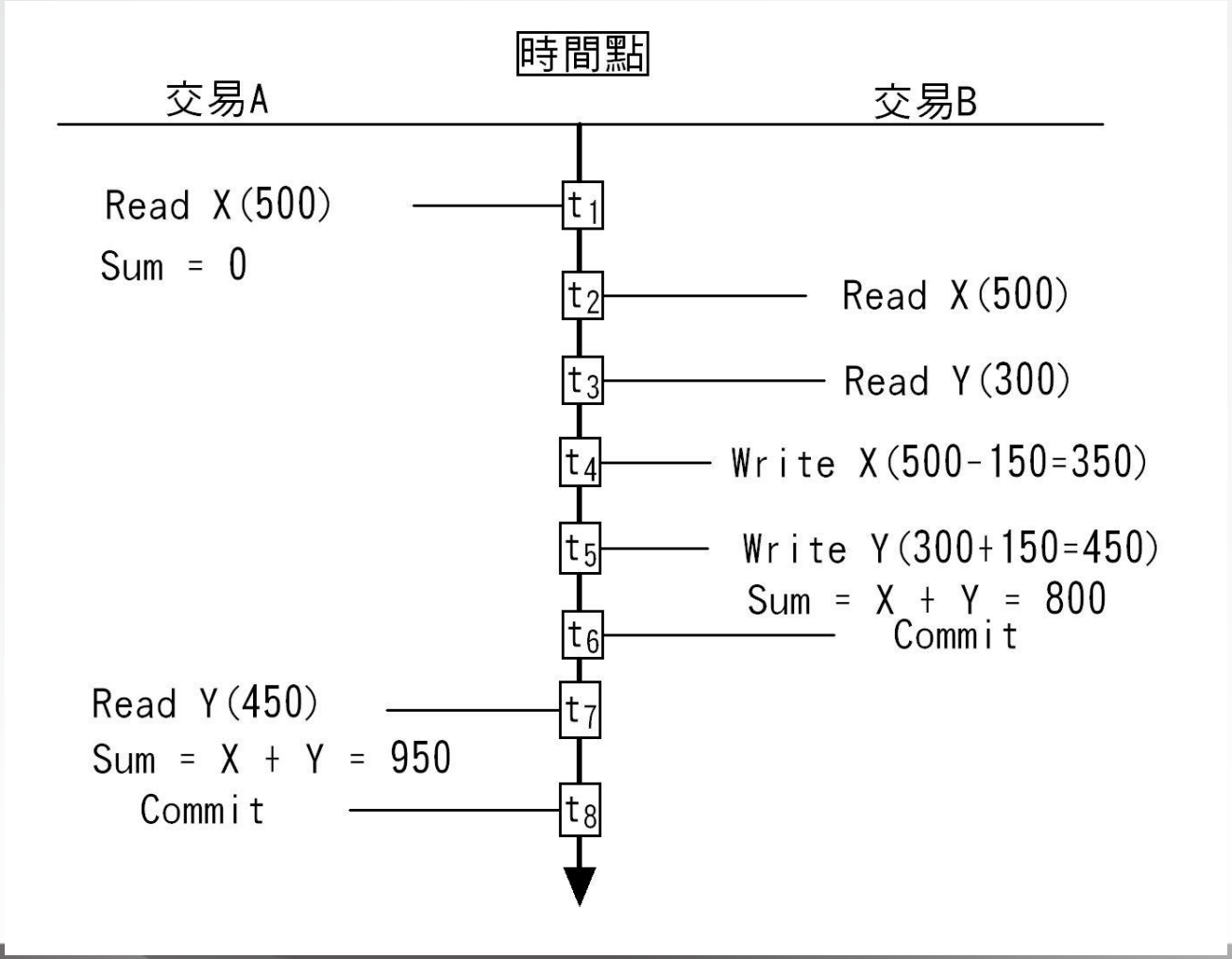
12-3-2 未確認交易相依問題-圖例



12-3-3 不一致分析問題-說明

- 不一致分析（Inconsistent Analysis）問題也稱為「不一致取回」（Inconsistent Retrievals）問題，這是因為並行執行多個交易，造成其中一個交易讀取到資料庫中不一致的資料。
- 例如：交易A和B存取同一位客戶在銀行的X和Y兩個帳戶，交易前兩個帳戶餘額分別為500和300元，交易A可以計算兩個帳戶的存款總額，交易B從帳戶X轉帳150元至帳戶Y。
- 因為交易A讀取的資料有部份是來自交易B更新前的帳戶餘額（帳戶X=500），這些是資料庫中不一致資料，所以造成計算結果的存款總額成為950元，而不是800元。

12-3-3 不一致分析問題-圖例



12-4 並行控制與排程

- 12-4-1 並行控制的排程
- 12-4-2 排程是否是等價的
- 12-4-3 並行控制演算法
- 12-4-4 SQL語言的隔離性等級

12-4-1 並行控制的排程-排程

- 「排程」（Schedules）是在執行一系列並行交易時，各交易資料庫單元操作和運算的完整執行順序，主要是針對寫入（Write）和讀取（Read）資料庫單元操作的執行順序。

12-4-1 並行控制的排程-循序性排程

循序性排程（Serial Schedules）

- 循序性排程是一種在各交易之間沒有並行性的排程，各交易的資料庫單元操作的執行順序並不會「交錯」（Interleaving）執行，如下圖所示：

Schedule A

時間點	交易A	交易B	交易C
t_1	Write x		
t_2	Read z		
t_3			Read x
t_4			Write y
t_5		Write x	
t_6		Write y	

12-4-1 並行控制的排程-可循序性排程

可循序性排程（Serializable Schedules）

- 排程的多交易的資料庫單元操作能夠交錯（Interleaving）執行的排程，如下圖所示：

Schedule B

時間點	交易A	交易B	交易C
t_1	Write x		
t_2			Read x
t_3	Read z		
t_4		Write x	
t_5			Write y
t_6		Write y	

12-4-1 並行控制的排程-可循序性

- 如果交錯執行的排程與循序性排程的執行結果相同，我們就稱此兩個排程為等價關係（Equivalent），並稱此交錯執行的排程滿足「可循序性」（Serializable）。
- Schedule A和Schedule B兩個排程是等價的，因為Schedule A是循序性排程，Schedule B是滿足可循序性（Serializable）的可循序性排程（Serializable Schedules）。

12-4-2 排程是否是等價的-衝突情況

- 衝突是發生在兩個交易的資料庫單元操作存取相同資料，三種資料庫單元操作的衝突情況，如下所示：
  - **都讀取資料**：讀取資料並不會產生衝突。
  - **一個讀取和一個寫入資料**：可能產生衝突。
  - **都寫入資料**：可能產生衝突。

12-4-2 排程是否是等價的-衝突定義

- 從上述的衝突分析，可以歸納出一個結論，兩個交易的資料庫單元操作產生衝突（Conflicting）的定義，如下所示：

定義12.1：兩個交易的資料庫單元操作產生「衝突」（Conflicting），如果滿足：

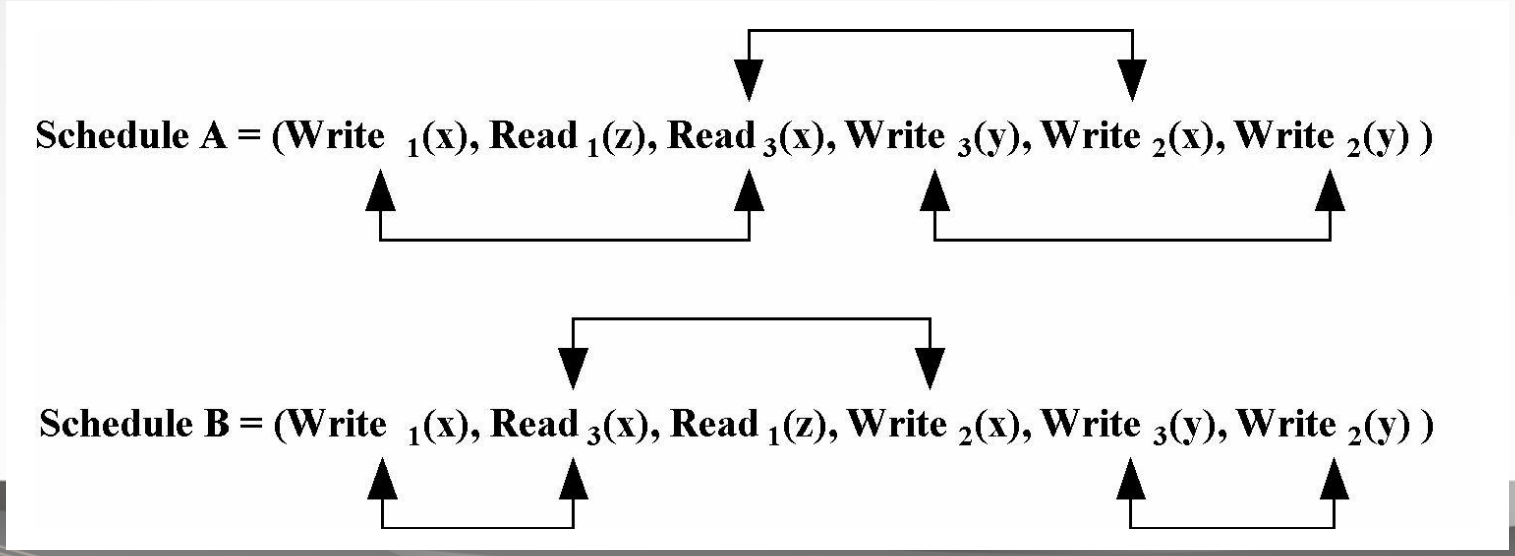
(1) 兩個交易的資料庫單元操作存取同一個資料。

(2) 兩個資料庫單元操作中，至少有一個是寫入操作。

12-4-2 排程是否是等價的-排程等價

定義12.2：如果兩個排程所有衝突情況的順序是相同的，就表示這兩個排程是「等價的」（Equivalent）。

- 在Schedule A和Schedule B兩個排程可以找出3個衝突情況，依序如下所示：



12-4-3 並行控制演算法-說明

- 在資料庫管理系統使用「並行控制演算法」（Concurrency Control Algorithms）執行交易的並行控制，不過使用演算法建立的排程是否正確，就是檢查排程是否滿足可循序性（Serializable），因為滿足可循序性的排程可以保持資料的一致性。

12-4-3 並行控制演算法-圖例

- 例如：一個不滿足可循序性的Schedule C排程，如下圖所示：

Schedule C

時間點	交易A	交易B	交易C
t_1	Write x		
t_2		Read x	
t_3			Write z
t_4	Read z		
t_5		Read y	
t_6			Write y
t_7		Write x	

12-4-3 並行控制演算法-先行圖

- 「先行圖」（Precedence Graph）檢查排程是否滿足可循序性，也稱為「循序圖」（Serialization Graph），其定義如下所示：

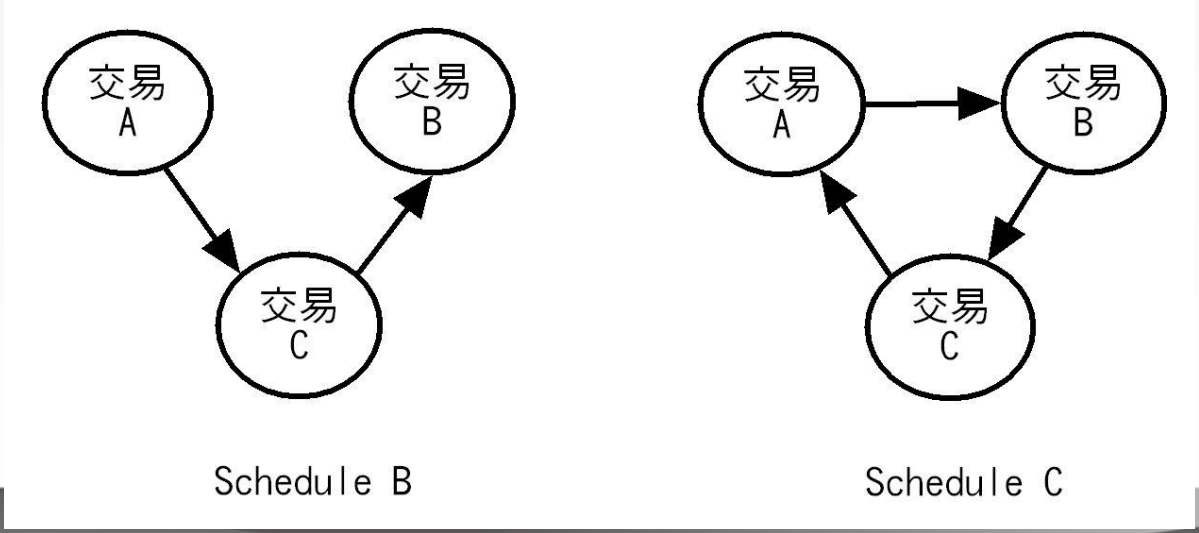
定義12.3：「先行圖」（Precedence Graph）是一種有方向性的圖形，連接排程S中各交易的圖形，寫成G(S)，如下所示：

(1) 在排程S中的每一個交易T，是以圖形的一個節點來表示。

(2) 在圖形G存在一條邊線Ti→Tj，如果排程S中擁有兩個衝突的資料庫單元操作oi和oj，且oi是在oj之前，即交易Ti的資料庫單元操作是在交易Tj前發生。

12-4-3 並行控制演算法-先行圖範例

- 依據先行圖的定義，可以建立Schedule B和Schedule C排程的先行圖，先行圖G(S)沒有迴圈（Loop），就表示排程S滿足可循序性（Serializable），以此例Schedule B沒有迴圈滿足可循序性，Schedule C擁有迴圈，不滿足可循序性。



12-4-4 SQL語言的隔離性等級-語法

- ANSI-SQL和Transact-SQL提供SET TRANSACT指令敘述指定四種交易的隔離性等級（Isolation Level），其語法如下所示：

  SET TRANSACT ISOLATION LEVEL isolation_level

12-4-4 SQL語言的隔離性等級-種類1

- READ COMMITTED：交易一定要在執行COMMIT確認交易後，才允許其他交易讀取。
- READ UNCOMMITTED：交易就算尚未執行COMMIT確認交易，也允許其他交易讀取。
- REPEATABLE READ：交易在尚未COMMIT確認交易前，不論讀取幾次的結果都相同。例如：交易A讀取資料x = 100後，交易B讀取變更相同資料x = 200後確認交易，此時如果交易A再次讀取x，x的值仍然是100，而不是交易B更改後的200。

12-4-4 SQL語言的隔離性等級-種類2

- SERIALIZABLE：即第12-4-1節的可循序性排程。
- SNAPSHOT：提供SERIALIZABLE隔離性等級的另一種實作方式，它是改用第12-5-4節的樂觀控制法來處理並行控制，而不是第12-5-1節的鎖定法，在某些情況下，可以提供更佳的執行效能。

12-5 並行控制的處理方法

- 12-5-1 鎖定法
- 12-5-2 死結
- 12-5-3 饑餓問題
- 12-5-4 避免死結的並行控制處理方法

12-5-1 鎖定法-說明

- 鎖定法（Locking）是並行控制最常見的處理方法，當交易A執行資料讀取（Read）或寫入（Write）的資料庫單元操作時，會先將資料鎖定（Lock），若同時交易B存取相同資料，因為資料已經被鎖定，所以交易B需要等待，直到交易A解除資料鎖定（Unlock）。
- 鎖定法使用的鎖定方法主要有兩種：
  - 二元鎖定（Binary Locks）
  - 共享與互斥鎖定（Shared/Exclusive Locks）

12-5-1 鎖定法-二元鎖定

二元鎖定（Binary Locks）

- 在鎖定資源時只有兩種狀態：鎖定（Locked）和非鎖定（Unlocked），這是一種互斥的鎖定方法，如同位元的0和1，也就是說，資源在同一個時間只能有一個交易進行存取。

12-5-1 鎖定法-共享與互斥鎖定

共享與互斥鎖定（Shared/Exclusive Locks）

- 共享與互斥鎖定（Shared/Exclusive Locks）針對這兩種單元操作將鎖定也分為兩種，如下所示：
  - **互斥鎖定（Exclusive Lock）**：也稱為「寫入鎖定」（Write Lock），稱為互斥是因為當交易A提出資料的互斥鎖定後，如果交易B提出相同資料的任何鎖定請求，都將拒絕鎖定請求。
  - **共享鎖定（Shared Lock）**：也稱為「讀取鎖定」（Read Lock），這是指當交易A提出資料的共享鎖定後，如果有交易B提出相同資料的共享鎖定請求，會同意其請求，如果是互斥鎖定請求則拒絕。

12-5-1 鎖定法-鎖定是否相容

- 鎖定方式間是否相容的情況，整理如下表所示：

	共享鎖定 (Shared Lock)	互斥鎖定 (Exclusive Lock)
共享鎖定 (Shared Lock)	相容	不相容
互斥鎖定 (Exclusive Lock)	不相容	不相容

12-5-1 鎖定法-鎖定衝突

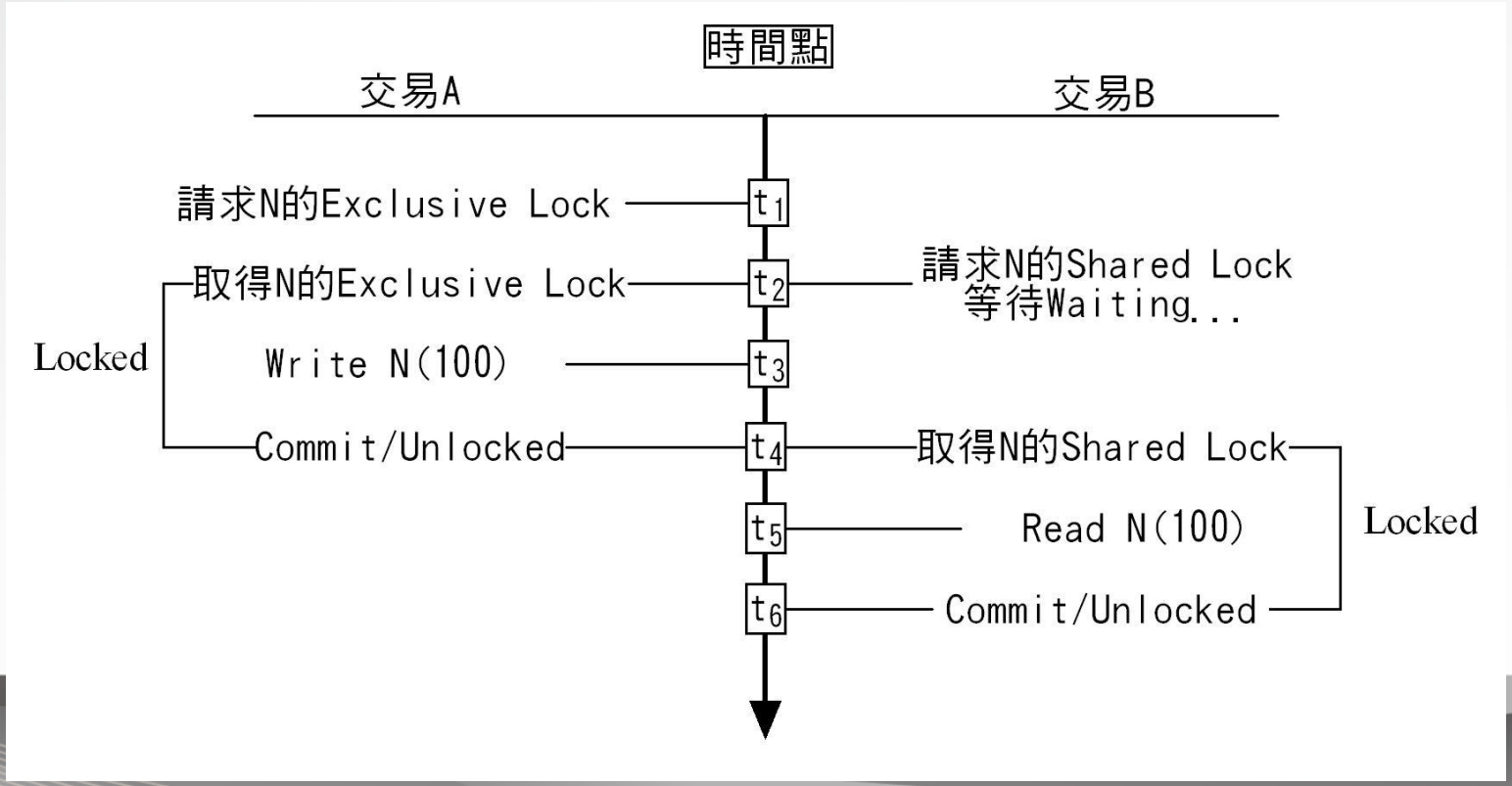
- 鎖定方式間不相容表示鎖定衝突存在，換句話說，兩個交易產生鎖定的「衝突」（Conflicting）條件，如下所示：
  - 兩個交易同時鎖定相同資料。
  - 其中有一個鎖定是互斥鎖定（Exclusive Lock）。
- 兩個交易都是寫入操作，稱為「寫寫衝突」（WW-conflict），一個讀取，一個寫入稱為「讀寫衝突」（RW-conflict）。

12-5-1 鎖定法-鎖定顆粒度

- 資料鎖定的範圍稱為「鎖定顆粒度」（Lock Granularity），各家資料庫管理系統的作法並不相同，有的是以一筆一筆記錄為單位的「列範圍」（Rows），有的是以多筆記錄為單位的「頁範圍」（Pages）來鎖定資料。
- SQL Server可以鎖定整個資料表或資料庫。

12-5-1 鎖定法-圖例

- 例如：N的值原來是80，交易A寫入N成為100，交易B讀取N，如果沒有使用鎖定方法，交易B可能讀取資料庫中不一致資料N=80。

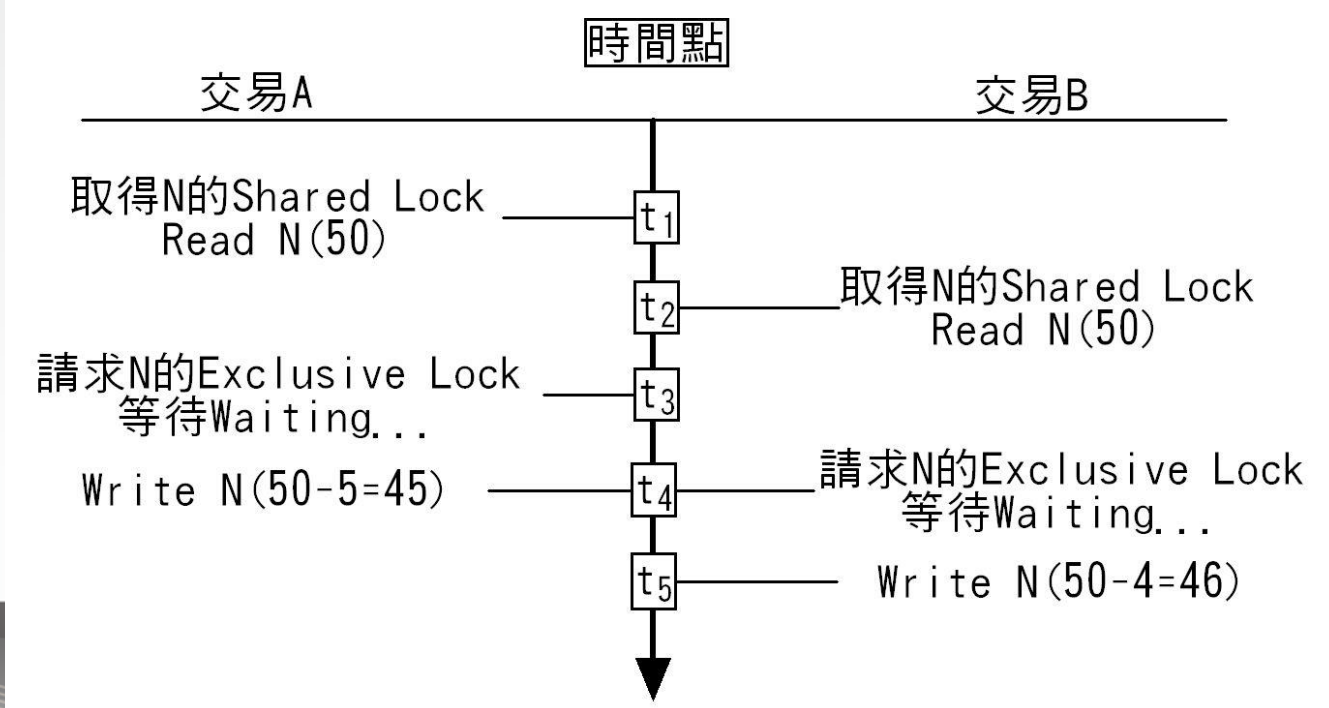


12-5-1 鎖定法-圖例說明

- 交易A在t_1請求互斥鎖定（Exclusive Lock）準備寫入資料，在t_2取得N的互斥鎖定，同時，交易B請求同一筆記錄的共享鎖定（Shared Lock），因為互斥鎖定和共享鎖定不相容，產生讀寫衝突，所以交易B需要等待N解除鎖定Unlock。
- 交易A在t_3和t_4寫入N=100後確認交易，也就解除資料鎖定，所以交易B取得共享鎖定，接著在t_5讀取N=100，最後在t_6確認交易和解除鎖定。

12-5-2 死結-基礎

- 死結是因為多個交易相互鎖定對方需要的資料，以至交易被卡死，導致多個交易都無法繼續執行的情況。例如：並行控制的更新遺失問題就一定會產生死結，如下圖所示：



12-5-2 死結-死結條件

- 彼此互斥（Mutual Exclusion）：交易對資源的寫入鎖定（Write Lock）皆視為互斥鎖定，同一時間同一資源只允許一個交易對其做寫入鎖定。
- 鎖定且等待（Lock and Wait）：每個交易皆對某些資源做寫入鎖定，並等待其他交易解除該資源的寫入鎖定或讀取鎖定（Read Lock）。
- 不得強佔（No Preemption）：當資源被某個交易做了寫入鎖定，別的交易不得強行做寫入鎖定或讀取鎖定。
- 循環等待（Cyclic Waits）：所有交易都在等待其他交易解除寫入鎖定，這些交易的等待情況會造成一個迴圈。

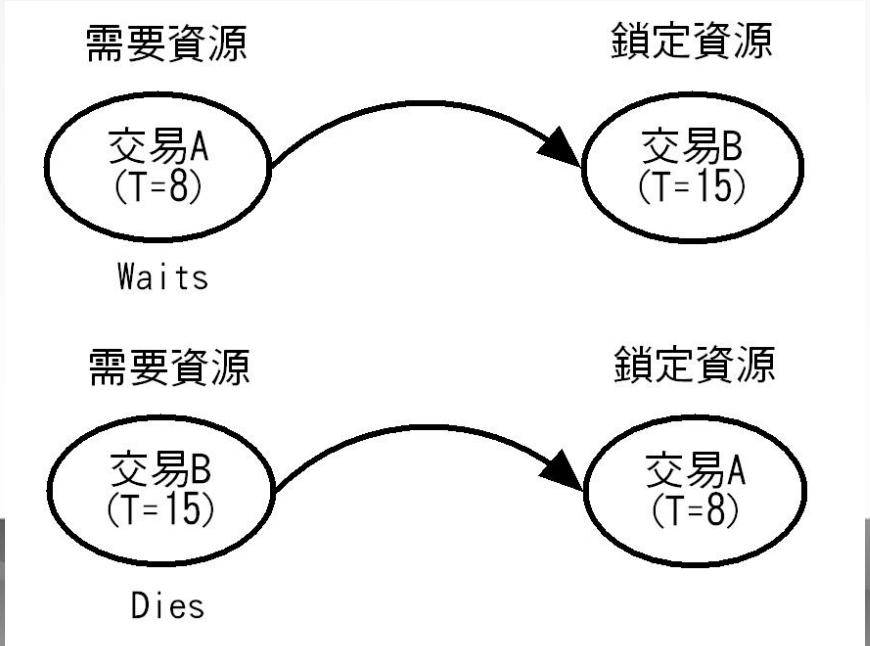
12-5-2 死結-死結處理(死結預防1)

死結預防（Deadlock Prevention）

- 死結預防是在執行交易前就檢查執行中交易的所有狀態，以確認是否會導致死結。
- 它是使用時間戳記（Timestamping）作為比較基礎來預防死結的發生，也就是說，並行控制的多個交易依其執行先後順序，需要給予每一個交易一個遞增的時間戳記。

12-5-2 死結-死結處理(死結預防2)

- Wait-die演算法：較早開始的交易A如果需要使用較晚開始交易B已鎖定的資源時，交易A就等待（Waits）；較晚開始的交易B如果需要使用較早開始交易A已鎖定的資源，就中止（Dies）交易B，並且在回復交易B後重新開始，如下圖所示：

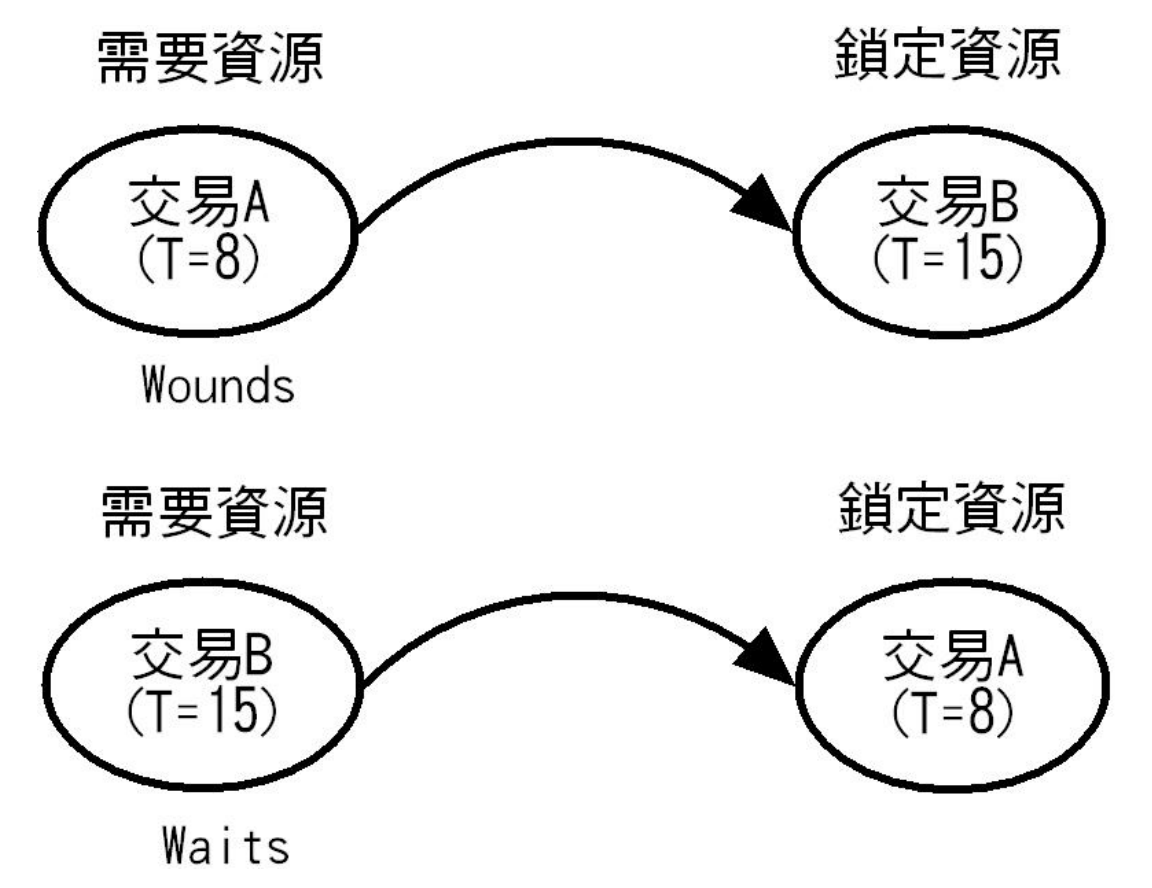


12-5-2 死結-死結處理(死結預防3)

- Wound-wait演算法：較早開始的交易A如果需要使用較晚開始交易B已鎖定的資源時，交易A會搶先執行，它的作法是送出交易B已經受傷（Wounded）的訊息，如此並行控制就會中止交易B，並且在回復交易B後重新開始；較晚開始的交易B如果需要使用較早開始交易A已鎖定的資源時，交易B就等待（Waits）。

12-5-2 死結-死結處理(死結預防4)

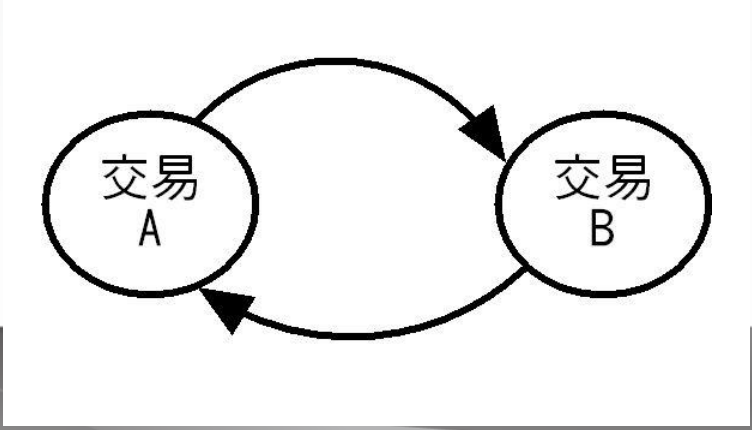
- Wound-wait演算法，如下圖所示：



12-5-2 死結-死結處理(死結偵測)

死結偵測（Deadlock Detection）：

- 資料庫管理系統在一定的間隔時間檢查處於等待狀態的交易，以確定是否產生死結，如果有，強迫交易回復交易（Rollback）後重新開始。死結偵測的方式可以使用「等待圖」（Wait-for Graph）進行檢查，這種圖形是以各交易為節點，$T_i \rightarrow T_j$邊線表示交易T_i在等待T_j鎖定的資料，如果圖形有迴圈就表示產生死結，例如：前述更新遺失問題的等待圖即擁有迴圈，如下圖所示：



12-5-2 死結-死結處理(死結避免)

死結避免（Deadlock Avoidance）：

- 根本避免並行控制產生死結的情況，資料庫管理系統需要使用一些避免死結的並行控制處理方法。

12-5-3 饑餓問題-說明

- 當並行控制使用鎖定法時，除了第12-5-2節的死結問題外，另一種問題是饑餓（Starvation）問題。這是指其他交易都可正常執行的情況下，交易持續一段不確定時間，都沒有辦法繼續執行。
- 簡單的說，饑餓是指交易很想執行資源的讀取和寫入操作，但是一直都無法如願執行，因為交易一直吃不到，所以持續處於饑餓狀態。

12-5-3 饑餓問題-解決方案

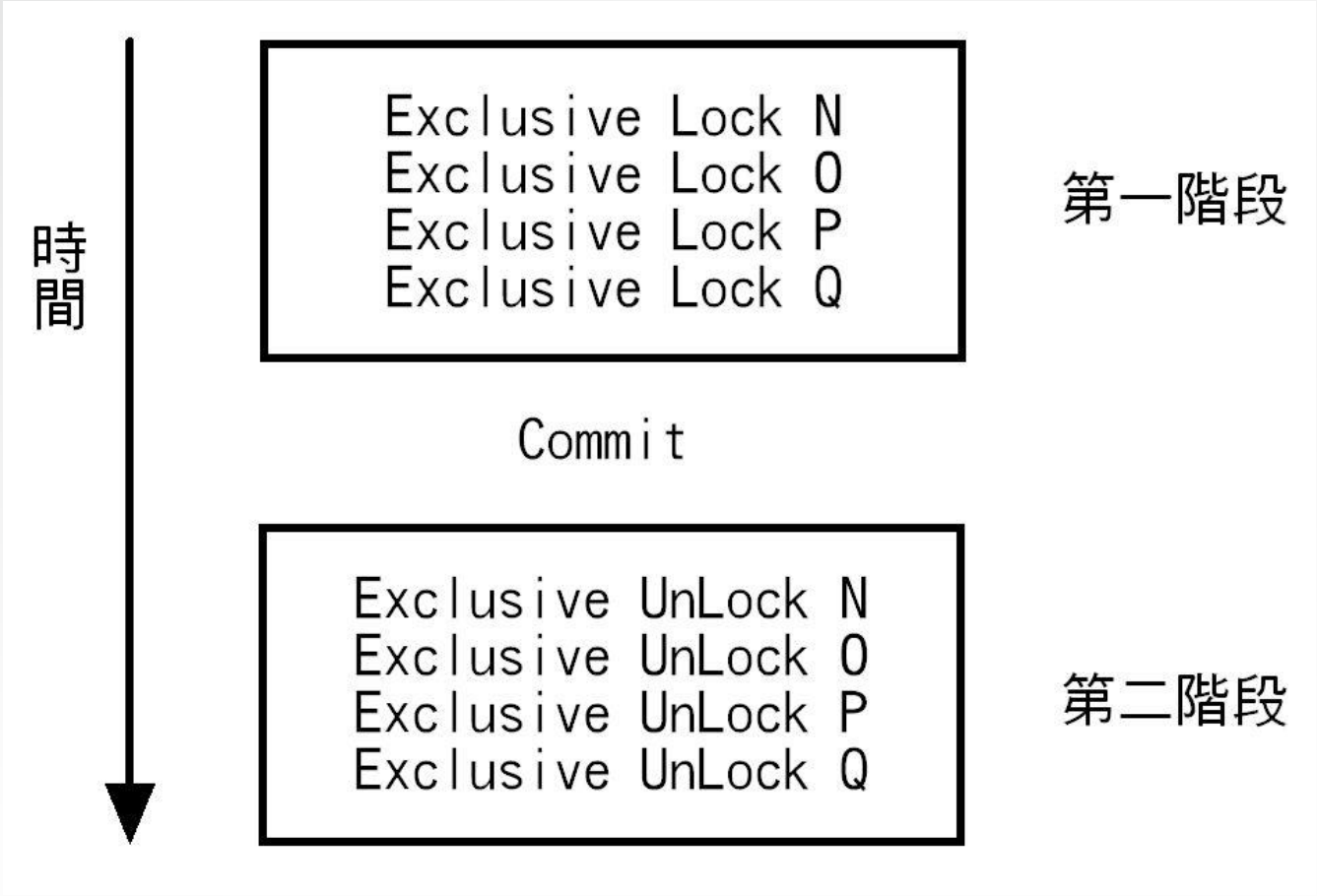
- 饑餓問題解決方案有兩種，如下所示：
  - **先請求鎖定擁有較高的優先權**：依據交易請求資源鎖定的順序來讓交易鎖定資源，先到先服務，愈先請求鎖定的交易擁有愈高的優先權。如同排隊上車，只是時間的先後，但是一定可以上車，所以並不會發生饑餓問題。
  - **等待愈久擁有愈高的優先權**：交易等待的時間愈久，就擁有愈高的優先權，因為等到最後，交易一定可以得到最高的優先權，然後進行鎖定，所以不會發生饑餓問題。

12-5-4 避免死結的並行控制處理方法-二階段鎖定法

二階段鎖定法（Two-phase Locking）

- 交易使用二階段來鎖定與解除資料鎖定，如下：
  - **第一階段為擴展階段（Expanding or Growing Phase）：**在交易A執行資料庫單元操作前，交易A需要請求所有需要存取資料的互斥鎖定，如果有資料已經被交易B鎖定，就等待直到交易B解除資料的鎖定。
  - **第二階段為縮減階段（Shrinking Phase）：**當交易A執行完操作後，就解除鎖定交易A所有鎖定的資料。

12-5-4 避免死結的並行控制處理方法-二階段鎖定法



12-5-4 避免死結的並行控制處理方法-時間戳記法

時間戳記法（Time-Stamp Order）

- 時間戳記法是指並行控制的多個交易，依其執行先後順序，給予每一個交易一個遞增的時間戳記（Time Stamping）。
- 當交易存取資料時，就將交易和資料的時間戳記進行比較，如下所示：
  - 如果交易的時間戳記大於或等於資料的寫入或讀取時間戳記，就執行資料存取，並且更新資料的時間戳記成為交易的時間戳記。
  - 如果交易的時間戳記小於資料的寫入或讀取時間戳記，交易將回復交易（Rollback）且重新啟動交易，以便讓交易可以得到更大的時間戳記來完成交易。

12-5-4 避免死結的並行控制處理方法-樂觀控制法

樂觀控制法（Optimistic Control）

- 樂觀控制法（Optimistic Control）是指各交易的資料庫單元操作都百分之百可以順利執行，直到確認交易（Commit）時，再由資料庫管理系統檢查是否會造成資料庫的不一致資料。如果有，就回復交易（Rollback）。
- 因為樂觀控制法一定會執行每一個交易，所以並不會產生死結情況。


End