



Fjord, Norway, by Frank S.C. Tseng
(<http://www2.nkfust.edu.tw/~imfrank>)

第十一章 異動管理



本章內容

- 11.1 前言
- 11.2 異動的四個特性
- 11.3 異動的回復處理 (Failure Recovery)
- 11.4 異動的並行控制 (Concurrency Control)
- 11.5 資料備份的建議



前言

- 「人力資源 + 有效資料」 為企業最重要資產。
- 異動管理的目的是維持資料間的一致性 (Consistency) 與正確性 (Correctness)。
- 異動管理分成兩大部份：
 - 回復處理 (Failure Recovery) 與
 - 並行控制 (Concurrency Control) 兩大機制



異動管理的目標

- 使系統的執行更有效率 (Efficiency)。
- 使系統的運轉更可靠 (Reliability)。
- 使每一個異動都能同時執行 (Concurrency)，不必排隊等候，造成效能不佳的情況。
- 維持最少的執行成本。
- 提高系統的妥善率 (Availability)。



異動的四個特性 ACID

- Jim N. Gray (1981) 所提出來
 - 單元性 (Atomicity)
 - 一致性 (Consistency)
 - 隔離性 (Isolation)
 - 持續性 (Durability)



異動管理 vs. ACID

- 要維持「單元性」與「持續性」是靠「回復處理」(Failure Recovery) 的機制來達成，
- 要確保「一致性」與「隔離性」則透過「並行控制」(Concurrency Control) 的機制來完成，
- 這兩個機制合起來便稱為資料庫的「異動管理」



單元性 (Atomicity)

- 異動整個必須要看成一個不可分割的個體，
- 也就是說：整個異動必須全部做完，否則萬一有任何錯誤時，必須要回復到未執行異動前的原點，也就是全部不做。
- 如果異動正常執行則透過「委任」(Commit) 命令將異動結果反應到資料庫中。
- 如果異動的執行異常，則透過「撤回」(Abort) 命令回復到未執行前的原點



異動被撤回的原因

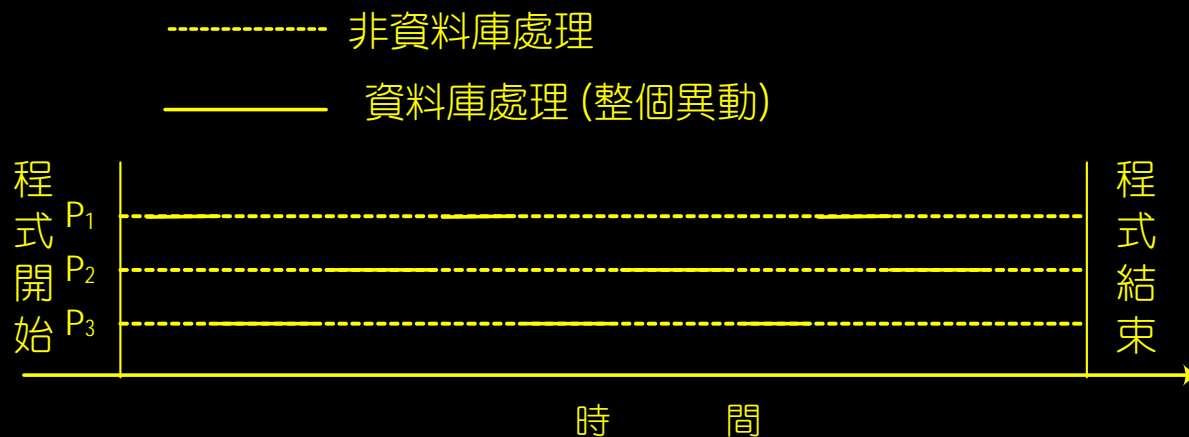
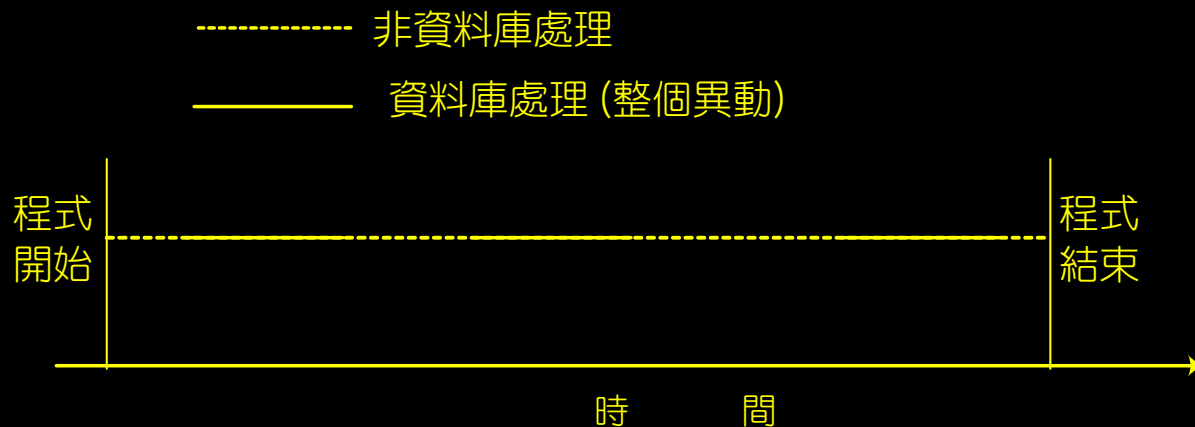
- 由異動本身所發出的撤回命令
 - 如：提款金額大於目前的存款金額
 - 如：提款時錯按密碼
 - 使用者要求而導致撤回，如：提款時按 “取消”
- 由系統本身所發出的撤回命令
 - 系統本身負載太大
 - 系統本身發生了「死結」
- 系統中途發生當機或無法抗拒的意外



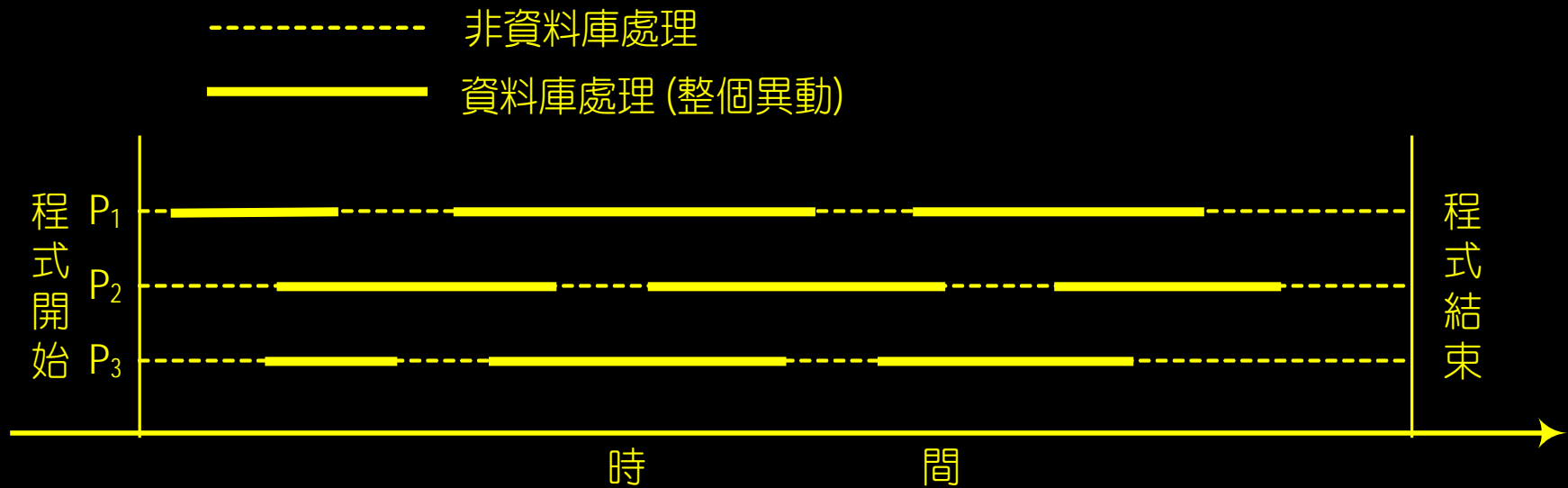
異動執行後，有底下三種情況

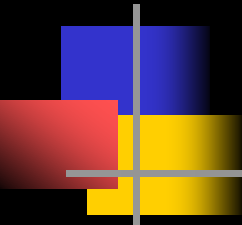
- 正常結束：begin transaction ... commit
- 異動發出撤回命令：begin transaction ... abort
(或 rollback)
- 系統於執行過程中發生當機或錯誤：
 - begin transaction ... × (當機)
 - (1) 沒有造成永久資料的損毀。
 - (2) 造成永久資料的損毀

一致性 (Consistency)



一致性 (續)





一致性 (續)

- 異動 T_1, T_2, \dots, T_n 同時交錯執行，則它們執行完畢後的異動結果必須要與這些異動按某個順序 (可以是： T_1, T_2, \dots, T_n 、 $T_2, T_3, \dots, T_n, T_1$ 、或 T_n, T_{n-1}, \dots, T_1 ，共 $N!$ 組排列情況) 執行的結果相同才行，
- 條件是：必須將資料庫從某個一致狀態帶到另一個一致狀態才行，
- 此一特性稱為異動的「一致性」(Consistency)。

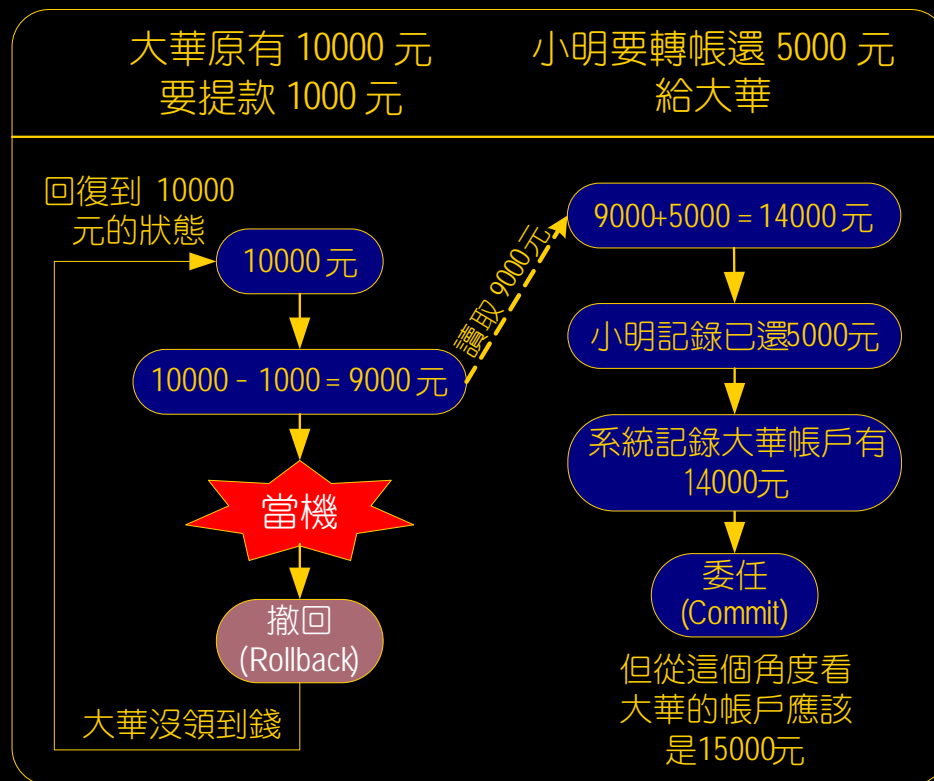


一致性 (Consistency)

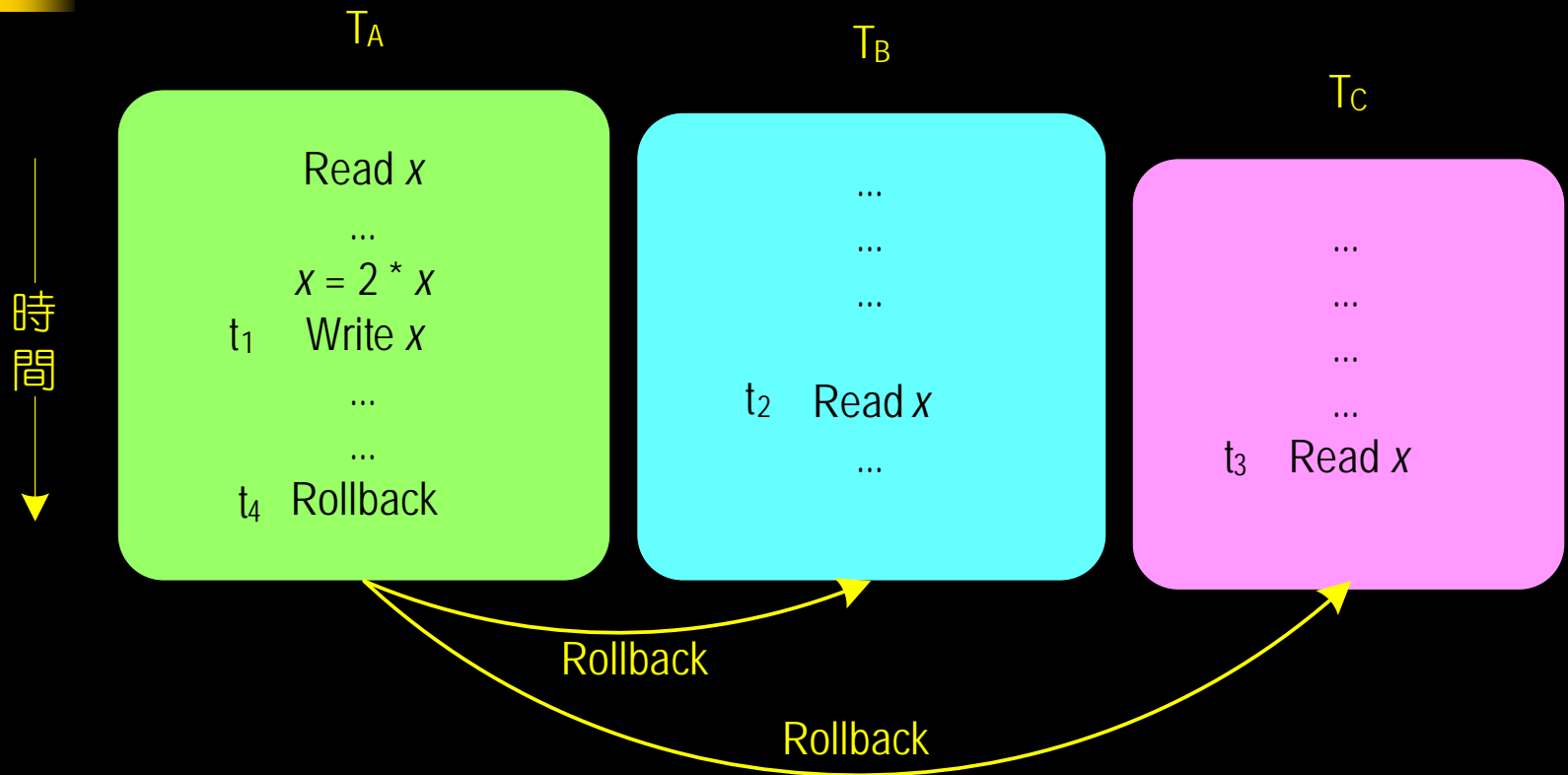
- 在提款與轉帳的例子中，結果要與下面兩件工作按某種順序排隊執行的一樣（誰先都對）
 - 大華先提款
 - 小明轉帳
- 有時數個異動以不同順序執行，會產生不同的結果，但以系統的眼光來看都算對
- 如果上述情況以人的眼光來看是錯的，則必須妥善控制

隔離性 (Isolation)

- 異動在尚未委任前的中間執行結果不得讓其它同時在執行的異動存取



若無隔離性會造成連鎖撤回

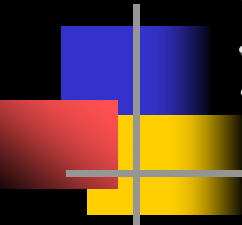


連鎖撤回效應



持續性 (Durability)

- 一但異動完成委任後，即使系統出了問題，也要恢復原貌
- 「異動記錄」(Transaction Log) 是維持此一特性的主角
 - 異動的過程要隨時做異動記錄
 - 當機時再根據異動記錄來回復
- 資料庫管理師必須瞭解其過程



持續性（續）

- 要配合下列步驟來達成持續性：
 - 定期備份 (Backup)。
 - 磁碟複本 (Disk Mirroring)。
 - 將運算資料 (Operational Data) 與異動記錄 (Transaction Log) 分開擺放在不同的磁碟上以免造成同時損毀。
 - 妥善擬定企業的備份策略。



回復處理 (Failure Recovery)

- 系統發生錯誤的情況
 - 系統發生錯誤，但不損害任何資料。
 - 系統發生錯誤，且損害記憶體中的資料。
 - 系統發生錯誤，且損害磁碟中的資料。
- 要降低此種錯誤所帶來的傷害，作法為
 - 將資料與異動記錄分開擺放
 - 使用具錯誤容忍度的磁碟陣列 或 RAID
 - 定期備份運作資料與異動記錄的例行工作
 - 使用不斷電系統 (UPS) 防止突發性的斷電



RAID 技術分成七個等級

- RAID (Redundant Array of Independent Disks)
- RAID 0 :
 - 沒有額外儲存檢查位元
 - 不具備容錯的能力，價格較便宜。
 - 做法是將所有磁碟組成一個虛擬的磁碟陣列，可同時對多個磁碟做讀與寫的動作
 - 又稱為 Data Striping 。



RAID 1

- 做法是將每個磁碟抄一份，複本儲存於另一個的完全相同的磁碟中，
- 存取時則同時對兩個完全相同的磁碟做讀與寫的動作，並保持其資料的一致性，
- 又稱為「磁碟複本」(Disk Mirroring)。
- 其中一個損壞時仍然可以由另一個磁碟存取。
- 兩個完全相同的磁碟也可以同時被讀取以增進存取效率。



RAID 2

- 做法是將每一筆資料都以位元為單位，分別存入每一部磁碟中，並選定一部特定的磁碟來儲存檢查位元 (Parity Bit)
- 又稱為「位元穿插術」 (Bit-Interleaving)
- 當某個磁碟損壞時，其內部資料可以由其它的磁碟與檢查位元的磁碟來回復。
- 檢查位元通常會佔全部資料的 10%~20% ，所以並不太適合實際上的應用。



RAID 3

- 做法與 RAID 2 類似，只是以位元組 (Bytes) 做為資料的分佈單位
- 將檢查資料以位元組為單位，只存放在一個磁碟上，做為容錯處理時參考使用
- 又稱為「位元組穿插術」 (Byte-Interleaving)。
- 優點是便宜且符合實際上的應用，而且在大量存取資料的效率會比小量存取資料時要來得好。



RAID 4

- 是針對上述 RAID 3 在小量讀寫時效率不佳的缺點來加以改進，
- 做法是將讀寫的資料量加大，又稱為「穿插傳輸區段」(Interleaves Transfer Blocks)。
- 由於每次在做寫入動作時，還要額外儲存檢查資料以支援容錯處理，所以當資料量很大時，存放檢查資料的磁碟可能會造成效能上的瓶頸，所以在實際的應用也有其不足之處。



RAID 5

- RAID 4 的缺點來加以改進，
- 做法是將檢查資料平均分佈在各磁碟中以支援容錯處理，又稱 Disk-Striping。
- 由於各個磁碟都有存放容錯資料，所以任何一個磁碟損壞了，都可以由其它磁碟回復。
- 此等級可以做平行存取，而且也不致於造成瓶頸，在磁碟存取效能與容錯處理方面取得了一個比其它等級要平衡的地位，所以非常適合實際上的應用。



RAID 6

- 基本上的做法是和 RAID 5 相同，只不過在容錯資料上有多了一份複本，以提高其容錯能力。
- 由於多了一份容錯資料又使得寫入時的效率降低，在加上價格較為昂貴，所以實際上使用的不多。
- 總括來說，常用的 RAID 技術是採用 RAID 0, 1, 3, 5 此四個等級。



異動記錄 (Transaction Log)

- 異動記錄中通常都會包含恢復 (Undo) 與重做 (Redo) 所需的相關資訊：
 - 異動編號 (Transaction id)
 - 該異動記錄的編號 (Log Record id)
 - 異動形式 (Action Type)
 - 異動前的資料值 (供 Undo 使用)
 - 異動後的資料值 (供 Redo 使用)
 - 其它相關的輔助資訊，例如：指標，用來指向同一個異動的前一筆異動記錄，以利搜尋



異動記錄 Commit/Abort

- 異動開始: `begin transaction transaction_id`
- 異動正常完成記錄 `commit transaction_id`
- 異動不正常完成則記錄 `abort transaction_id`
(或 `Rollback transaction_id`)
- 恢復 (Undo) 與重做 (Redo) 的運算則需具有 Idempotent 的特性：也就是執行數次與執行一次的結果應該是一樣的。



Log-write-ahead Protocol

- 為防止下述動作之間發生了系統錯誤，導致系統於恢復運作時無法正確判斷：
 - 寫入異動記錄的動作。
 - 將異動結果寫入資料庫中的動作。
- 通常資料庫管理系統都會採用 Log-Write-Ahead Protocol，先將異動記錄記錄完成後，再來將異動結果寫入資料庫中。



異動記錄上的注意事項

- 異動記錄一直增長，會使磁碟的空間達到飽和狀態，而使系統無法正常運作。
- 備份完畢後，要定期刪除異動記錄。
- 一個好的資料庫管理系統會提供使用者定義磁碟空間使用門檻 (Threshold) 的監督
- 例如：Sybase 中的儲存程序
sp_thresholdaction
- 或是下達 truncate log on checkpoint 等命令



系統當機後資料庫管理系統完成回復處理的步驟

- 如果系統只是單純的當機，磁碟資料仍然完整時，資料庫管理系統可以透過本節的步驟自動完成回復處理的工作
- 這項工作不須要由人來處理
- 要了解回復處理步驟必須先了解「檢查點」(checkpoint) 的觀念



檢查點 (Checkpoint)

- 「檢查點」(checkpoint)—在某固定的時間點(如：一秒鐘)由系統執行下列動作：
 - 將正在執行之異動的編號存到異動記錄中。
 - 寫下一筆「檢查記錄」(Checkpoint Record)到異動記錄中。
 - 將資料庫中應該反應到磁碟上的異動資料完全寫到磁碟中(包含異動記錄與運算資料，依照「異動記錄優先記錄協定」會先將異動記錄優先記錄完成後再來將異動結果寫入資料庫中)



「檢查記錄」的作用

- 讓資料庫管理系統在系統發生錯誤後：
 - 能有效地辨別那一個異動，事實上已經委任了（可從異動記錄中發現已經記錄了一筆 Commit Transaction_id），只不過資料尚未來得及反應到資料庫中（在更新資料庫中的資料之前，系統當機了！）
 - 或者是那一個異動在系統發生錯誤時還未完成 Commit，所以於系統重新啟動後，便要強制將它 Abort。



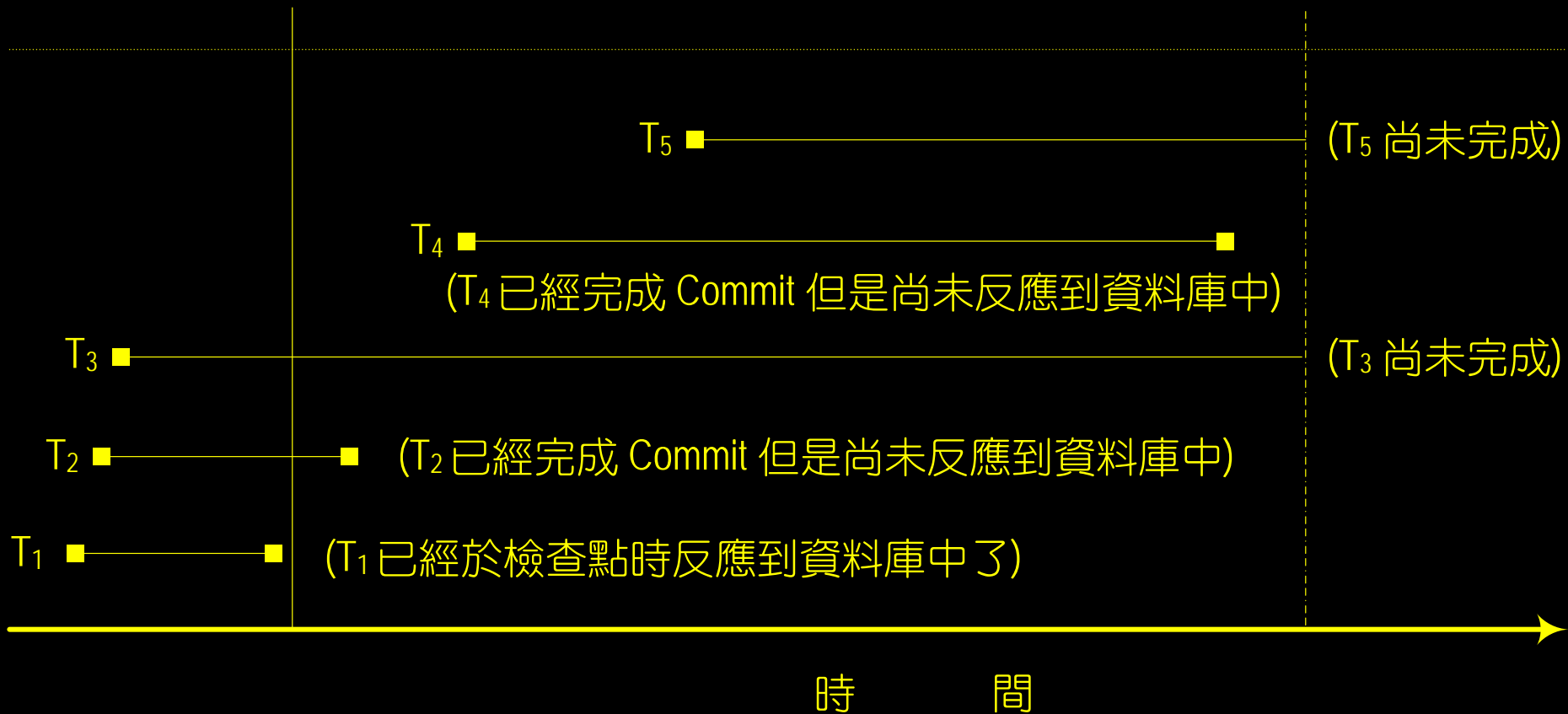
檢查點有兩種作法

- 非同步檢查點 (Asynchronous Checkpoint)：可隨時接受新來的異動。
- 同步檢查點 (Synchronous Checkpoint)：
 - 作法較為單純，在檢查點時，系統將不再接受新來的異動，一直等到所有目前正在執行的異動都完全做完為止，才執行上述的例行工作，
 - 效能上會較差，
 - 不過上述的第一點：“將當時都正在執行的異動之編號儲存起來” 在這種作法下是可以省略的。

非同步檢查點的作法

checkpoint

系統發生錯誤





非同步檢查點的作法 (續)

- 先求出一個 UNDO-List，以及一個 REDO-List。
- UNDO-List 存放的是尚未完成的異動，它們都會被強制撤回 (Abort)；
- REDO-List 存放的是已經完成的異動，只不過異動後的資料尚未反應到資料庫中，所以系統會根據異動記錄來重做 (Redo) 這些異動。



求取 UNDO-List 與 REDO-List

- UNDO-List = { } , REDO-List = { }
- 由檢查記錄 (Checkpoint Record) 往前搜尋，看看是否有某個異動在異動記錄中
 - 有一筆 begin transaction *transaction_id*
 - 但是卻沒有 commit *transaction_id* 或 abort *transaction_id*
 - 若有則放入 UNDO-List 中 ({T2, T3})
- *To Be Continued ...*



求 UNDO-List 與 REDO-List (續)

- 再由檢查記錄往後搜尋，看看是否有 `begin transaction transaction_id`，若有則加入 UNDO-List 中 ($= \{T2, T3, T4, T5\}$)
- 由檢查點往後搜尋，看看是否有 `commit transaction_id` 的異動，若有則由 UNDO-List 中將它移出後 ($= \{T3, T5\}$)，加入 REDO-List 中。 ($= \{T2, T4\}$)
- Undo-List 反向回復，Redo-List 正向重做



磁碟損毀後的回復處理步驟

- 如果系統的運算資料仍然存在，但異動記錄被損毀時，系統仍然可以正常運作
- 資料被損毀時
 - 如果異動記錄仍然存在，我們可以重新載入備份資料，讓資料庫管理系統依據異動記錄儘可能地重新執行自上次備份後所完成的異動，儘量將資料救回磁碟毀損前的狀態。
 - 如果異動記錄也損毀了則只能恢復到備份的時間點



回復處理步驟重點

- 將損毀前的備份重新載入，然後利用異動記錄重新將已完成，但是尚未備份的異動重做一遍。
- 不須要任何 Undo 的動作！
- 平時應該將資料與異動記錄分別存放在具有不同損毀機率或模式的磁碟上，以降低資料全毀的機率。
- 在分散式的環境中，回復處理更為複雜

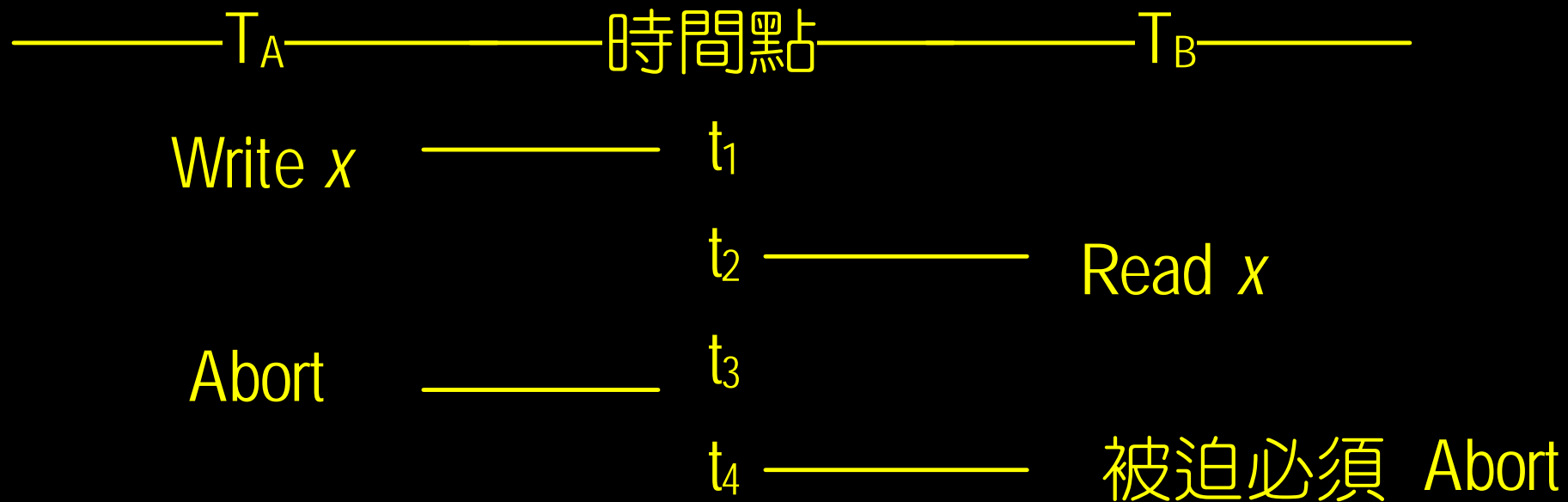


異動的並行控制

- 沒有並行控制所引發的問題
 - 相依於未委任的異動 (Uncommitted Dependency Problem)
 - 白做的更新 (Lost Update Problem)
 - 不相容的分析結果 (Inconsistent Analysis)

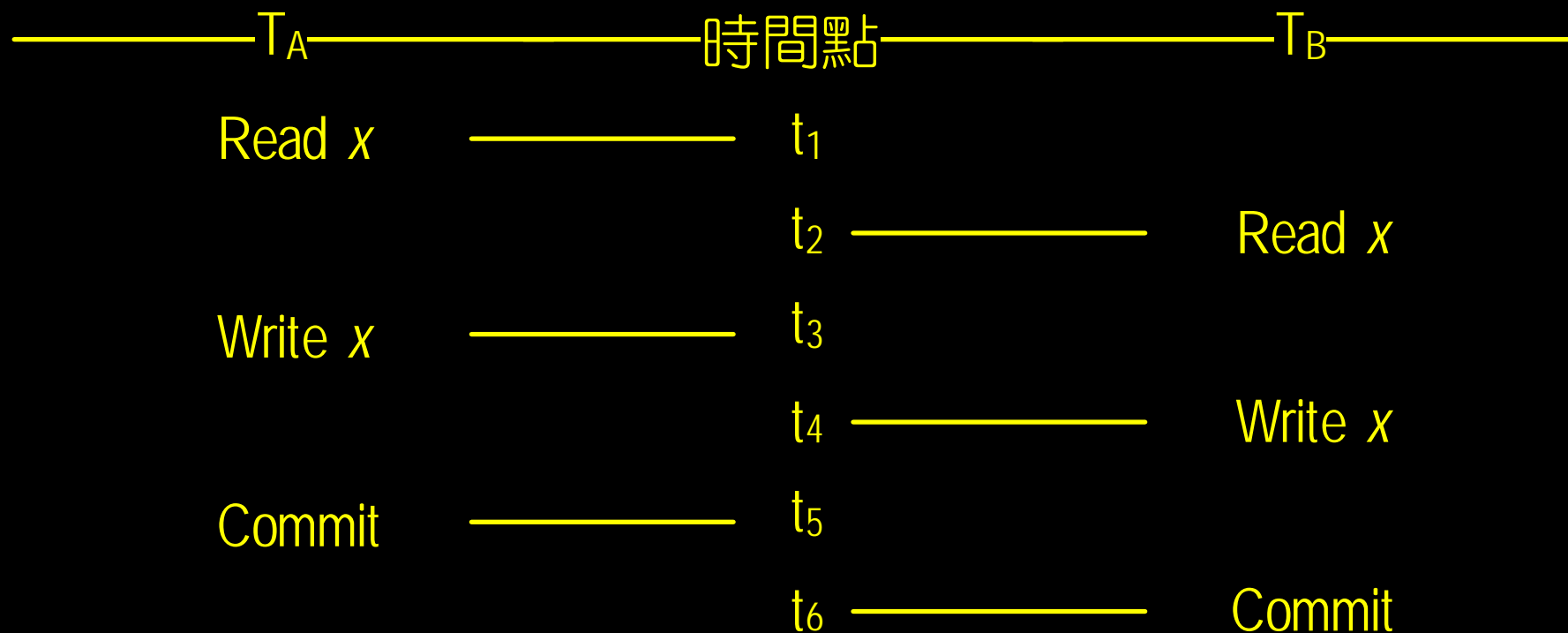


相依於未委任的異動

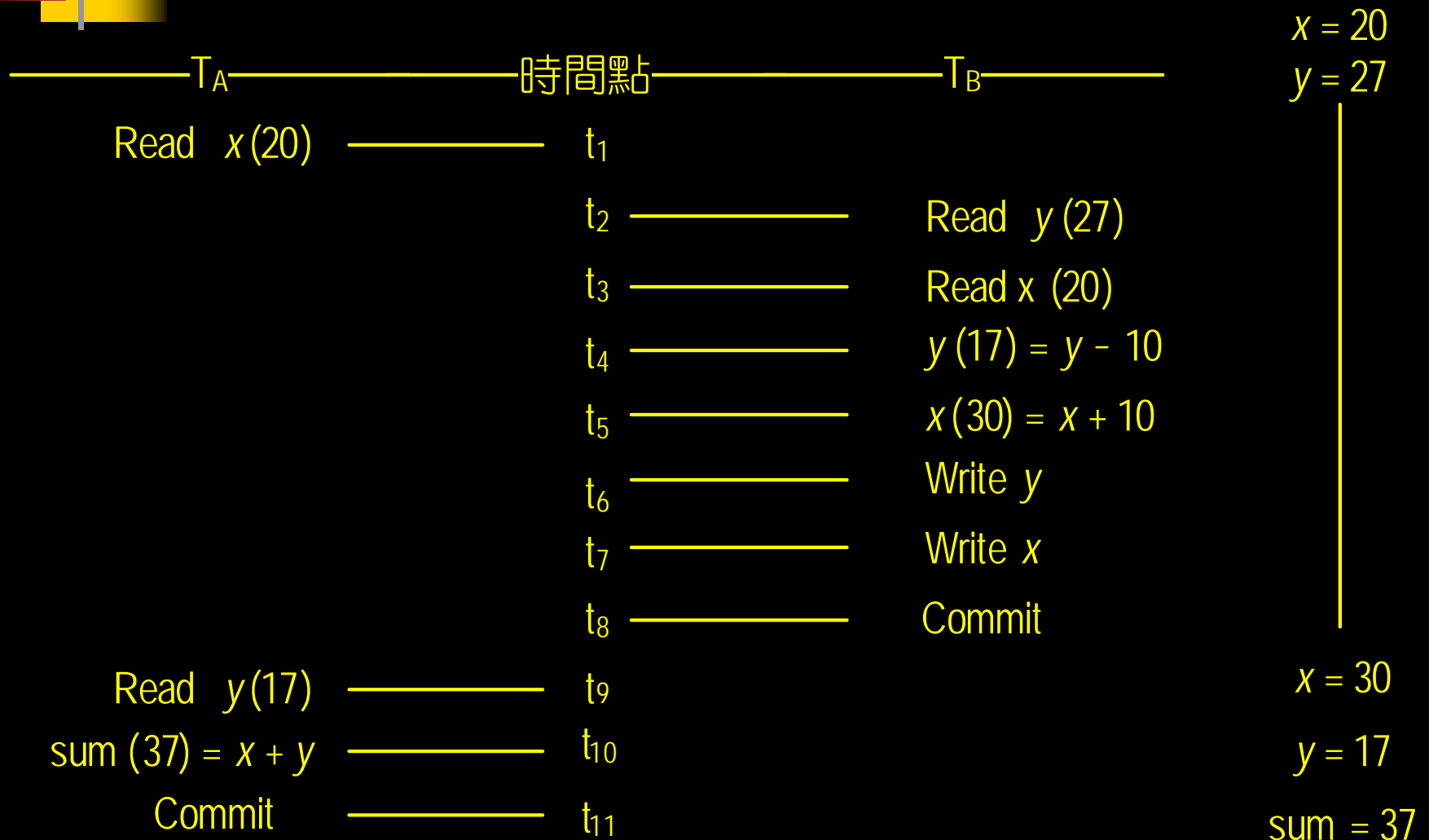




白做的更新 (Lost Update)



不相容的分析結果





異動的並行控制機制

- 兩個異動若要存取同一資料時
 - 兩個都只讀取 (Read)：不會有任何問題。
 - 其中一個讀取 (Read)，另外一個則要寫入 (Write)：此兩異動之間有「衝突」(Conflict) 發生，可能會造成上述三種問題
 - 兩個都要寫入 (Write)：在這種情況下也有衝突發生，可能造成上述三種問題。

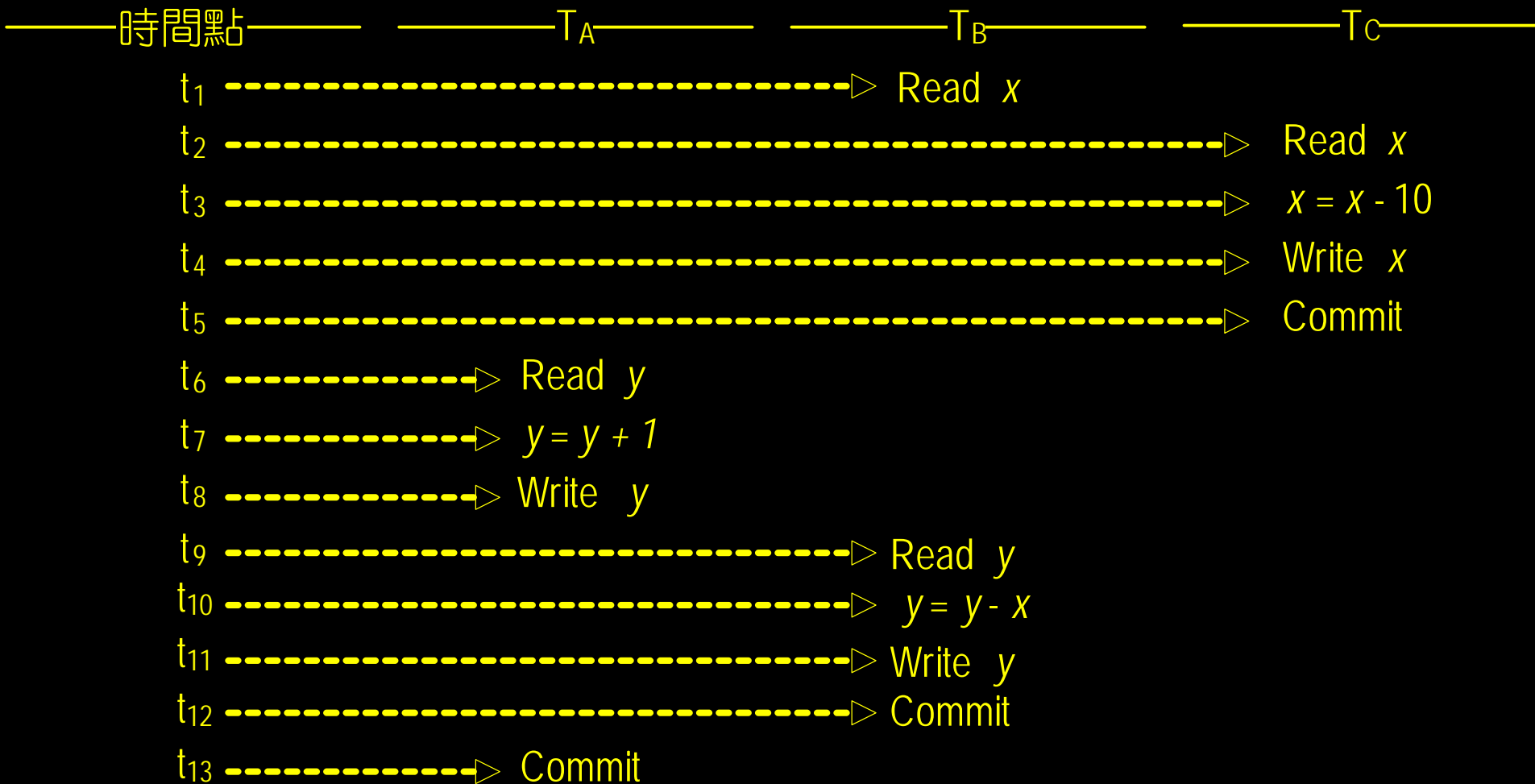


「循序性」 (Serializability)

- 讓異動都可以穿插執行 (Interleaving)。但是，其結果要確保與某一個按照順序來一一執行所有異動的結果一樣
- 符合 Serializability 條件的排程 (Schedule) 一定會保持資料的一致性
- 但「正確的排程」卻不一定符合 Serializability



說明範例 $T_A \rightarrow T_B \rightarrow T_C$





為什麼不是 $T_B \rightarrow T_C \rightarrow T_A$?

- $S = (R_B(x), R_C(x), W_C(x), R_A(y), W_A(y), R_B(y), W_B(y))$



衝突 ($T_B \rightarrow T_C$)



衝突 ($T_A \rightarrow T_B$)

- 所以應該是 $T_A \rightarrow T_B \rightarrow T_C$

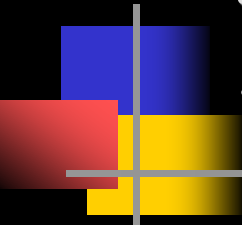
- 執行結果和下面的排程一樣

$(R_A(y), W_A(y), R_B(x), R_B(y), W_B(y), R_C(x), W_C(x))$



並行控制的處理方法

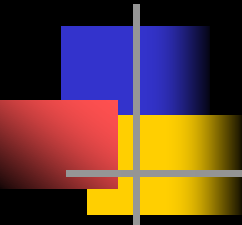
- 鎖定法 (Locking)
- 戳記順序法 (Time-Stamp Ordering)
- 積極進取控制法 (Optimistic Control)
- 日期值 (Value Date)：異動在開始前先宣稱
它希望完成的日期、時間
- 循序圖測試法 (Serialization Graph Testing)：分
散式資料庫系統上的並行控制方式



鎖定法 (Locking)

- 資料的鎖定分為兩種：
 - 讀取鎖定 (Read-Lock) ，
 - 寫入鎖定 (Write-Lock) 。
- 兩個不同異動有衝突 (Conflict) 的條件：

	Read-Lock	Write-Lock
Read-Lock	相容	不相容
Write-Lock	不相容	不相容



兩階段鎖定法 (2-Phase Locking)

第一階段

Lock D_1
Lock D_2
Lock D_3

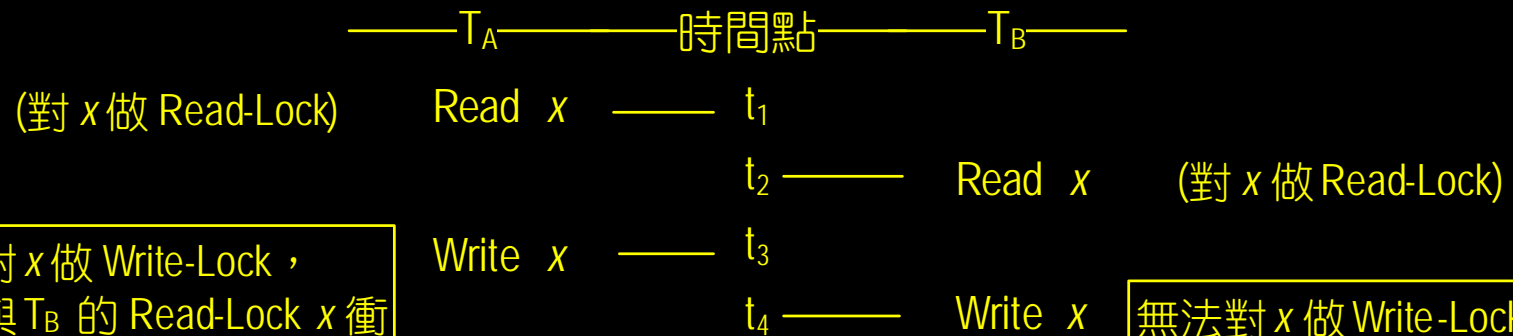
Commit

第二階段

Unlock D_1
Unlock D_2
Unlock D_3

- 處理異動時的
並行性
(Concurrency)
並不高
- 很容易造成
「死結」
(Deadlock)

死結的例子



無法對 x 做 Write-Lock，
因為與 T_B 的 Read-Lock x 衝突，所以要等待。

無法對 x 做 Write-Lock，
因為與 T_A 的 Read-Lock x 衝突，所以要等待。

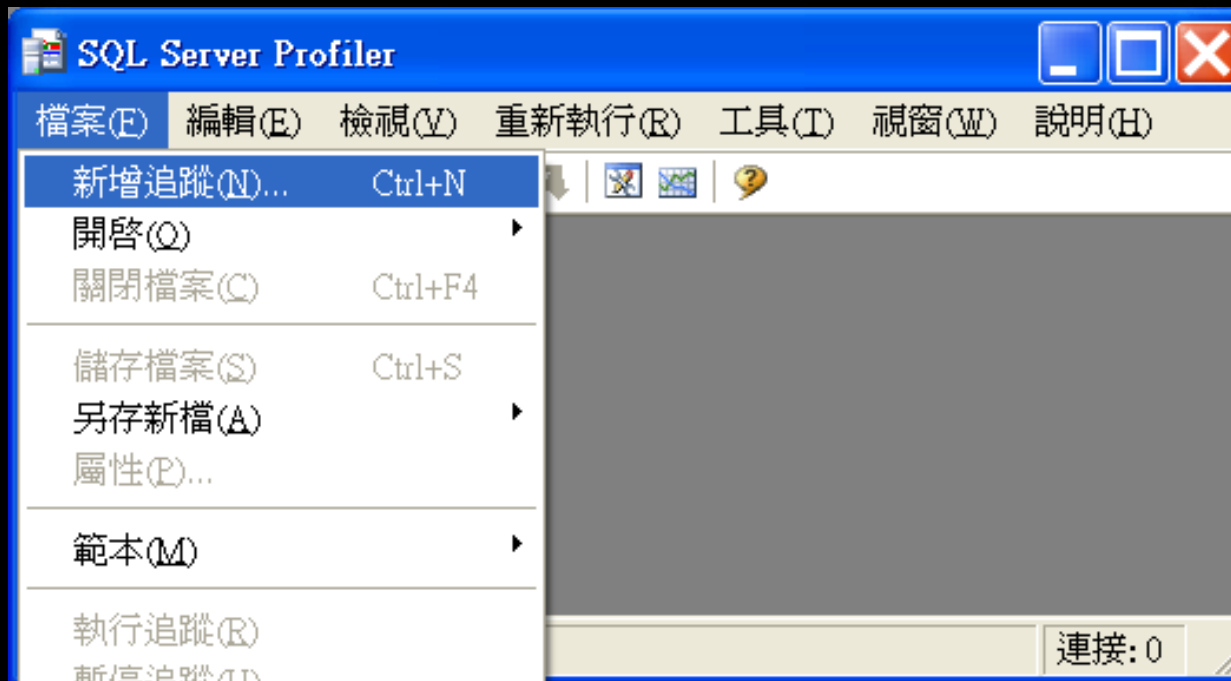


造成死結的四個條件

- 彼此互斥 (Mutual Exclusion)
- 鎖定並等待 (Lock and Wait)
- 不可以去鎖定已經被鎖定的資料 (No Preemption)
- 循環等待 (Circular Wait)
- 只要避免其中之一發生便可以避免死結

SQL Server 2008 的死結偵測

- 透過設定 SQL Server Profiler
- 當死結發生時，SQL Server 2008 可以偵測並顯示死結的細節。



SQL Server 2008 的死結偵測

追蹤屬性

一般 | 事件選取範圍

追蹤名稱(N): 未命名 - 1

追蹤提供者名稱: .

追蹤提供者類型: Microsoft SQL Server 2008 版本: 10.0.1600

使用範本(U): Standard (預設值)

☐ 儲存至檔案(S):

設定檔案大小上限 (MB)(I): 5

☒ 啟用檔案換用(N)

☐ 伺服器處理追蹤資料(E)

☐ 儲存至資料表(B):

設定最大資料列數 (單位: 千)(R): 1

☐ 啟用追蹤停止時間(Q): 2009/11/26 下午 11:41:55

執行 取消 說明

SQL Server 2008 的死結偵測

追蹤屬性

一般 | 事件選取範圍

檢閱已選取要追蹤的事件與事件資料行。若要查看完整清單，請選取 [顯示所有事件] 和 [顯示所有資料行] 選項。

Events	TextData	ApplicationName	NTUserName	LoginName	CPU	Reads	Writes	Duration	ClientProcessID
+ Broker									
+ CLR									
+ Cursors									
+ Database									
+ Deprecation									
+ Errors and Warnings									
+ Full text									
+ Locks									
+ OLEDB									
+ Objects									
+ Performance									

Broker
包括 Service Broker 所產生的事件類別。

ApplicationName (套用了 1 個篩選條件)
建立連接到 SQL Server 的用戶端應用程式名稱。會依應用程式所傳遞的值、而非由所顯示的程式名稱，來擴展這個資料行。

☒ 顯示所有事件(E)
☐ 顯示所有資料行(C)

資料行篩選(E)...
組織資料行(O)...

執行 取消 說明

SQL Server 2008 的死結偵測

追蹤屬性

一般 | 事件選取範圍 | 事件擷取設定

檢閱已選取要追蹤的事件與事件資料行。若要查看完整清單，請選取 [顯示所有事件] 和 [顯示所有資料行] 選項。

Events	TextData	ApplicationName	NTUserName	LoginName	CPU	Reads	Writes	Duration	ClientProcessID
+ Broker									
+ CLR									
+ Cursors									
+ Database									
+ Deprecation									
+ Errors and Warnings									
+ Full text									
- Locks									
<input checked="" type="checkbox"/> Deadlock graph	<input checked="" type="checkbox"/>			<input checked="" type="checkbox"/>					
<input type="checkbox"/> Lock:Acquired	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/> Lock:Cancel	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				<input type="checkbox"/>	<input type="checkbox"/>

Broker
包括 Service Broker 所產生的事件類別。

ClientProcessID (不套用篩選條件)
呼叫 SQL Server 之應用程式的處理序識別碼。

☒ 顯示所有事件(E)
☐ 顯示所有資料行(C)

資料行篩選(E)...
組織資料行(Q)...

執行 取消 說明

SQL Server 2008 死結測試實驗

- 將下列指令分別輸入不同的 Query 畫面
以製造死結的發生 (產生最下方的錯誤訊息)

```
begin transaction
update books set bookname = 'A'
where id = 1
waitfor delay '00:00:10'
select * from books where id = 2
```

```
begin transaction
update books set bookname = 'B' where id = 2
waitfor delay '00:00:10'
select * from books where id = 1
```

訊息1205，層級13，狀態51，行4

交易(處理序識別碼56) 在鎖定資源上被另一個處理序鎖死並已被選擇作為死結的犧牲者。
請重新執行該交易。

Microsoft SQL Server Management Studio

檔案(F) 編輯(E) 檢視(V) 查詢(Q) 專案(P) 偵錯(D) 工具(T) 視窗(W) 社群(C) 說明(H)

新增查詢(N) [Icons]

BOB [Execution Icons]

物件總管

- 連接(Q)
- SQL Server 10.0.1600 - KIDLAB\imfrank
 - 資料庫
 - 系統資料庫
 - 資料庫快照集
 - BOB

SQLQuery2.sql - (loc...IDLAB\imfrank (56))* SQLQuery1.sql - (loc...IDLAB\imfrank (55))*

```
begin transaction
update books set bookname = 'A' where id =1
waitfor delay '00:00:10'
select * from books where id = 2
```

已連接。 (1/1) (local) (10.0 RTM) KIDLAB\imfrank (55) BOB 00:00:00 0 個資料列

輸出

就緒 第 5 行 第 1 欄 字元 1 INS

Microsoft SQL Server Management Studio

檔案(F) 編輯(E) 檢視(V) 查詢(Q) 專案(P) 偵錯(D) 工具(T) 視窗(W) 社群(C) 說明(H)

新增查詢(N) [Icons]

BOB [Execution Icons]

物件總管

- 連接(Q)
- SQL Server 10.0.1600 - KIDLAB\imfrank
 - 資料庫
 - 系統資料庫
 - 資料庫快照集
 - BOB

SQLQuery2.sql - (loc...IDLAB\imfrank (56))* SQLQuery1.sql - (loc...IDLAB\imfrank (55))*

```
begin transaction
update books set bookname = 'B' where id =2
waitfor delay '00:00:10'
select * from books where id = 1
```

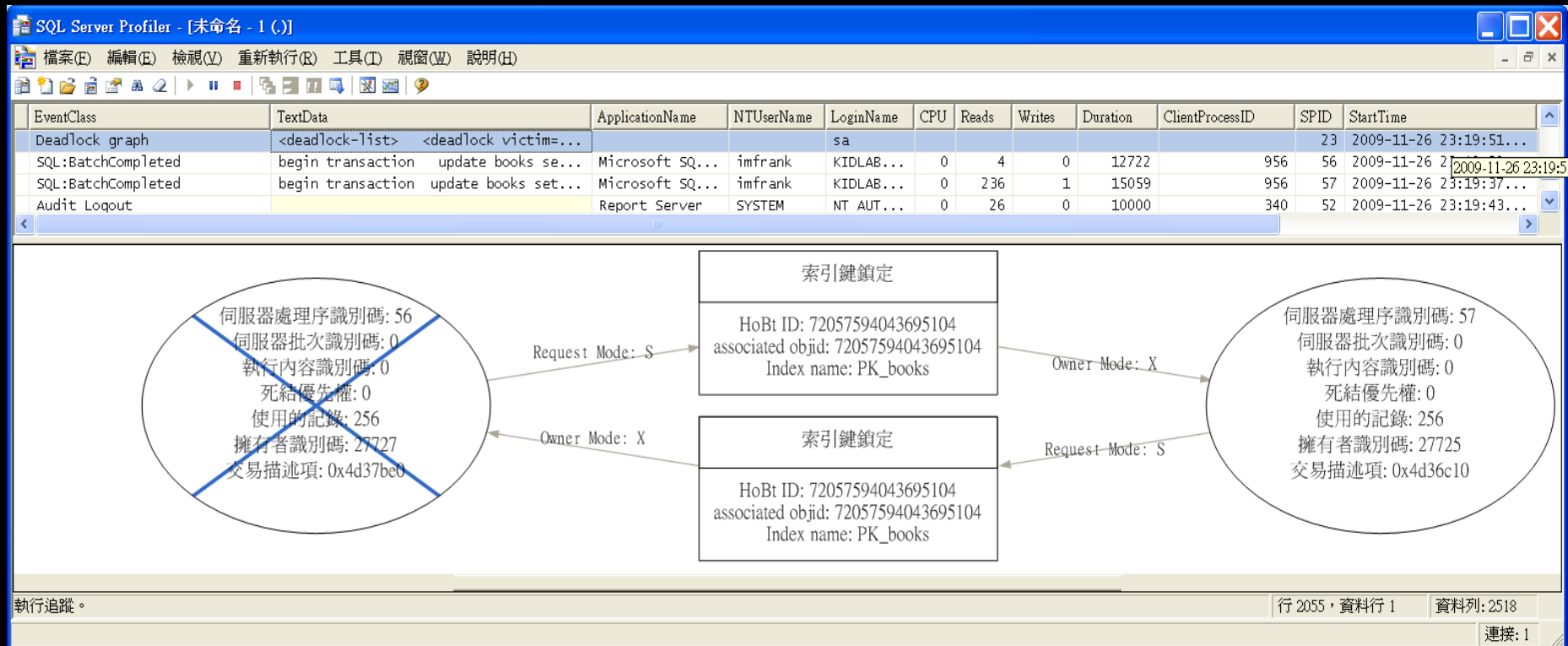
已連接。 (1/1) (local) (10.0 RTM) KIDLAB\imfrank (56) BOB 00:00:00 0 個資料列

輸出

就緒 第 5 行 第 1 欄 字元 1 INS

SQL Server 2008 的死結圖示說明

- 到 SQL Serve Profiler 中搜尋 deadlock 關鍵字





分散式資料庫系統上的鎖定法

- 一種做法是寫入時鎖定所有複本，讀出時僅鎖定一份複本即可 (Write-lock-all, Read-lock-one)
- 另一種做法則是規定某一複本是「主版本」 (Primary Copy)，鎖定時則都針對主版本做即可 (Primary Copy Locking)。



時間戳記法

- 將每一個異動 T_i 依其開始的先後順序，賦予一個時間戳記 (Time Stamp)
- 有異動 T 想存取某一資料，則要先將該資料蓋上該異動的時間戳記
- 系統便將欲存取此資料的所有異動依戳記上的時間先後依序予以執行，
- 絕對不會形成循環等待



時間戳記法的衝突定義

- 時間戳記法的衝突定義 ($T1$ 與 $T2$ 發生衝突):
 - 當異動 $T1$ 要求讀取一筆資料，而該筆資料曾被另一個較「年輕」(即時間戳記較大)的異動 $T2$ 更改過。
 - 當異動 $T1$ 要求寫入一筆資料，而該筆資料曾被另一個較「年輕」的異動 $T2$ 讀取或更改過，。
- 解決衝突的方式是將 $T1$ 重新啟動
- 不會有「撤回」(Rollback) 的情況發生
因為所有中間運算結果是放在記憶體中



每一筆資料的 RMAX 與 WMAX

- 依據每個異動 T 啟動的先後順序, 給予一個時間戳記 t 。
- $RMAX(x)$ 為讀取記錄 x 之最「年輕」異動 (Youngest Transaction) 的時間戳記。
- $WMAX(x)$ 為寫入記錄 x 之最年輕異動 (Youngest Transaction) 的時間戳記。
- 用下面的演算法來做並行控制...



時間戳記法的演算法

- 有異動 T 要讀取一筆記錄 R，假設其時間戳記為 t，則系統會做以下檢查

```
if ( $t \geq WMAX(x)$ ) then {  
    /* 允許讀取 */  
     $RMAX(x) = \max(t, RMAX(x));$   
} else {  
    /* 產生衝突 */  
    重新啟動 T;  
}
```

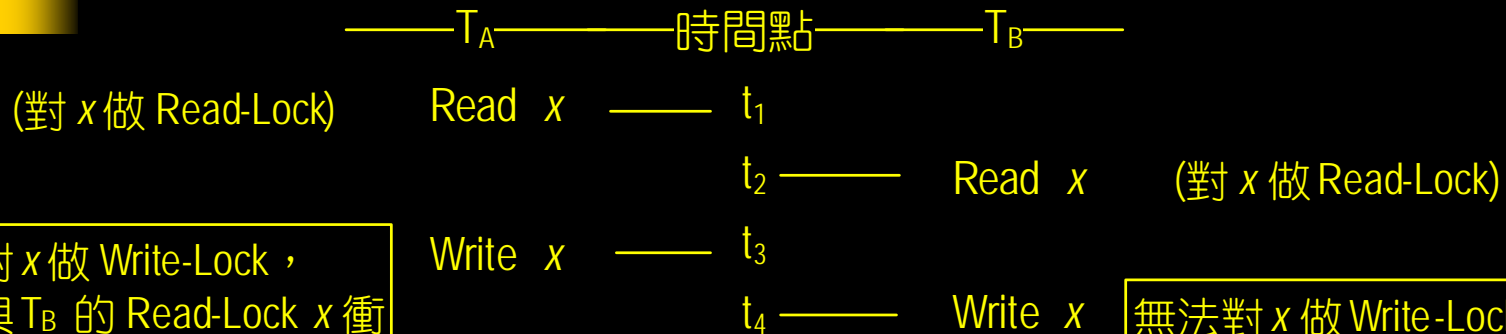



時間戳記法的演算法 (續)

- 如果一個異動 T 要寫入一筆記錄 R 的話，則系統會做以下檢查

```
if (( $t \geq RMAX(x)$ ) && ( $t \geq WMAX(x)$ )) {  
    /* 允許寫入! */  
     $WMAX(x) = t$ ;  
} else {  
    /* 衝突產生 */  
    重新啟動  $T$ ;  
}
```

不會發生 Deadlock 的範例說明



無法對 x 做 Write-Lock，
因為與 T_B 的 Read-Lock x 衝突，所以要等待。

無法對 x 做 Write-Lock，
因為與 T_A 的 Read-Lock x 衝突，所以要等待。

時間點	T_A 時間戳記 $t=100$	T_B 時間戳記 $t=120$	說明
t_1	Read x		$t(T_A)=100 > WMAX(x)=90$ \therefore 可以 Read，且將 $RMAX(x)$ 設定為 100。
t_2		Read x	$t(T_B)=120 > WMAX(x)$ \therefore 可以 Read，且將 $RMAX(x)$ 設定為 120。
t_3	Write x		$\therefore t(T_A)=100 < RMAX(x)=120$ $\therefore T_A$ 必須重新啟動，領取新的時間戳記重新來過。
t_4		Write x	$t(T_B)=120 \geq RMAX(x)$ 且 $t(T_B) > WMAX(x)$ \therefore 可以 Write，且將 $WMAX(x)$ 設定為 120。



時間戳記法的特性

- 異動的執行順序便如同依照其各異動真正完成時所取得的時間戳記由大至小來排列一般
- 可以配合使用「多重版本」(Multi-Version) 的做法，讓一筆記錄有多重版本



積極進取控制法

- 旅美的學者孔祥重教授 (Prof. H.T. Kung) 提出
- 原則：
 - 先讓每一個異動都放手一搏，儘量地往前大步邁進，直到要委任之前才由系統檢查 (Certify) 執行過程中是否有衝突情況發生？
 - 如果有的話，則由系統選擇一個影響最小的異動，命令它撤回。



系統要做檢查之前，必須要

- 在異動做完所有資料的讀取後，指定一個異動編號 (Transaction Number) 給該異動，
- 並對每一筆記錄記載一個 Read Set 以及一個 Write Set，存放曾經讀取或寫入該筆記錄的異動編號。
- 系統於是利用 Read Set 與 Write Set 中的異動編號來做檢查、驗證的工作



積極進取配合鎖定觀念的作法

- 我們可以引入 Strong Lock與 Weak Lock 的觀念，
- Strong Lock 可以蓋過 Weak Lock，
- 而且 Weak Lock 與 Weak Lock 之間是彼此相容的。
- 配合鎖定觀念的作法如下頁所示



配合鎖定觀念的作法 (續)

- 異動在存取資料時，求向系統求一個 Weak Lock，該 Weak Lock 在 Commit 後，會提升成為 Strong Lock。
- 如果異動在要求 Weak Lock 時，該資料已經有 Strong Lock 則此異動會被標示成「撤回」(Abort)，但並沒有真的撤回。



配合鎖定觀念的作法 (續)

- 在驗證時期，負責並行控制的模組會檢查：
 - 各個異動是否有被標示成「撤回」？
 - 若否，則該異動便可以 Commit，並且將它所要求的所有 Weak Lock 全變成 Strong Lock，所有尚未被驗證的異動先前如果也有要求這些剛變成 Strong Lock 的話，則標示成「撤回」



資料備份的建議

- 以會計年度為單元，分成兩個資料庫存放
 - 過期資料 (當時會計年度以前的資料) 與
 - 當期資料 (當時會計年度的資料)
- 資料庫在設計時對於有時效性的資料也要設計有保存期限的欄位



清除 SQL Server 2008 異動記錄

- 在 Checkpoint 時順便清除異動記錄的指令：
sp_dboption BOB, "trunc. log on chkpt.", true
go
checkpoint
go
- 僅下指令時清除異動記錄
Backup Log BOB with truncate_only -- 或
Backup Log BOB with no_log



備份資料庫與異動記錄

- 備份資料庫之前，最好先將 SQL Server 2008 啟動為 Single-User Mode：由 DOS 下執行 “sqlserver.exe -m”
- 也可以先使用下列指令檢查資料庫與異動記錄之間的相容性 (Database Consistency)：
dbcc checkdb -- 可能會很花時間
dbcc checkalloc --
dbcc checkcatalog
go
- 另外有檢查關聯表的指令 (比較不花時間)：
dbcc checktable (*tablename*) -- 或
dbcc checkalloc
go



備份資料庫與異動記錄

- 先使用 Backup Database 指令備份資料庫內容：
Backup Database BOB To Disk = 'C:\BOB1.bak',
DISK = 'C:\BOB2.bak',
DISK = 'C:\BOB3.bak'
Mirror To Disk = 'D:\BOB1.bak',
DISK = 'D:\BOB2.bak',
DISK = 'D:\BOB3.bak'

With Format

go

- 若不使用 With Format 則會接在原檔案之後



備份資料庫與異動記錄

- 接著用 Backup Log 指令備份異動記錄：
Use master
Go -- 修改 BOB 成為完整復原模式
Alter Database BOB Set Recovery Full
Go
Backup Log BOB To Disk = 'C:\BOB_Log.bak'
Go
- 通常資料庫預設使用簡單復原模式。若要建立 BOB 的異動記錄備份，那必須在完整備份之前，將它改成完整復原模式，才能如願



另一種備份資料庫的方式

- 透過 sp_addumpdevice 先建立備份裝置名稱：

Use master

Go

```
EXEC sp_addumpdevice 'disk', 'BOBData', 'C:\BOB_Data.bak'
```

```
EXEC sp_addumpdevice 'disk', 'BOB_Log', 'C:\BOB_Log.bak'
```

Go

- 然後再以相同指令備份到該裝置名稱上：

Backup Database BOB To BOBData

Go

Backup Log BOB To BOBLog

Go



還原資料庫與異動記錄

- 使用 Restore database 與 Restore Log 來達成：

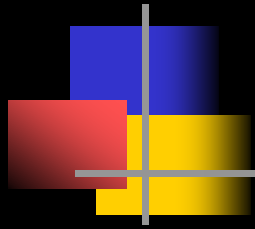
Use master

Go

Restore Database BOB From BOBData With Recovery

Go

- 若希望還原作業不要回復任何未委任的交易，則須將 With Recovery 改成 With NoRecovery，系統預設是 With Recovery。



本章結束
The End.