交易管理

- 導論
- 交易的執行
  - 並行處理的問題
  - 交易失敗的問題
- 交易和系統的概念
  - ACID的性質
  - 系統運作追蹤檔
- 正確的交易排程
  - 正確的交易復原
  - 正確的交易並行
- 利用鎖定來產生正確的交易排程
  - 嚴格兩階段鎖定法
  - 多層次鎖定
- SQL對交易的支援

導論

- 資料庫交易指的是將數個資料存取或更新的動作當成一個整體
  - 如果在執行過程中有任何的差錯，這些動作可以全部取消，好像從沒有發生過一樣
  - 如果沒有差錯，則這些交易的效果保證永久存在，即使將來系統當機亦然。
  - 範例

交易

```
1. INSERT INTO Transaction
   VALUES ('93000', 'b0922468', 'cart',…)
2. INSERT INTO Record
   VALUES ('93000', 'b30999', 1, 500);
3. INSERT INTO Record
   VALUES ('93000', 'd11222', 1, 300);
4. INSERT INTO Record
   VALUES ('93000', 'b10234', 2, 550);
```

還原

失敗

交易的執行

- 資料庫交易被定義成是「一個資料庫程式的執行」
  - 一次執行可能包括數個SQL敘述， SQL敘述的執行其實也就是對資料庫裡資料的讀或寫
  - 每次執行一個資料庫程式所產生的SQL敘述可能會不同，因為程式裡可能會有判斷式（如IF…THEN…ELSE)
  - 每次執行都被視為一個交易
- 觀念上，一個交易是由數個以下五種基本運算動作所組成：
  - begin(T)：表示一個交易T開始執行。
  - read(X, *x*)：表示從硬碟讀取資料項X到主記憶體變數*x*。
  - write(*x*, X)：表示將主記憶體變數*x*寫至硬碟資料項X。
  - commit(T)：表示一個交易T成功的結束。
  - abort(T)：表示一個交易T被駁回，其以前做過的動作因此全部還原。

交易的並行處理

- 一般DBMS會同時交錯處理數個交易的運算，稱為並行處理
  - 因為循序處理既沒效率也浪費資源
- DBMS如果任意交錯執行，有可能造成錯誤的執行結果
  - 更新遺失的問題（The Lost Update Problem）
  - 污染讀取的問題（The Dirty Read Problem）
  - 無法重複讀取的問題（The Non-repeatable Read Problem）
  - 幽靈資料的問題（The Phantom Problem）

更新遺失的問題

P1	P2

begin(T1)
read(X, *a*);
a=*a*+10;
write(*a*, X)
commit(T1)
	begin(T2)
	read(X, *b*);
	b=*b*+20;
	write(*b*, X)
	commit(T2)

一開始 X=100
P1和P2都執行後的
正確結果：X=130

交錯執行

	T1	T2
1	begin(T1)	
2	read(X, *a*);	
3		begin(T2)
4		read(X, *b*);
5	write(*a*, X)	
6	commit(T1)	
7		write(*b*, X)
8		commit(T2)

此更新的結果被蓋過去了！

X = 120

污染讀取的問題

P1	P2

begin(T1)
read(X, *a*);
a=*a*+10;
write(*a*, X)
abort(T1)
	begin(T2)
	read(X, *b*);
	b=*b*+20;
	write(*b*, X)
	commit(T2)

一開始 X=100
P1和P2執行後的
正確結果：X=120

交錯執行

	T1	T2
1	begin(T1)	
2	read(X, *a*);	
3	write(*a*, X)	
4		begin(T2)
5		read(X, *b*);
6	abort(T1)	
7		write(*b*, X)
8		commit(T2)

讀取到的資料後來被還原了！

X = 130

無法重複讀取的問題

P1	P2

begin(T1)
read(X, *a*);
a=*a*+10;
write(*a*, X)
commit(T1)
	begin(T2)
	read(X, *b*);
	…
	read(X, *b*)
	commit(T2)

交錯執行

	T1	T2
1	begin(T1)	
2		begin(T2)
3		read(X, *b*);
4	read(X, *a*);	
5	write(*a*, X)	
6	commit(T1)	
7		read(X, *b*);
8		commit(T2)

b = 100

b = 110

一開始 X=100
P2讀取X兩次的
結果應該相同

兩次讀取到的資料不一樣！

幽靈資料的問題

P1	P2
begin(T1) I1="**INSERT INTO** Product(pNo,unitPrice) **VALUES** ('b40000', 600); Exec(I1); commit(T1)	begin(T2) **Q1="SELECT * FROM** Product **WHERE** unitPrice > 500"; Exec(Q1); … Exec(Q1); commit(T2)

好像多了一筆幽靈記錄！

	T1	T2
1	begin(T1)	
2		begin(T2)
3		Exec(Q1); → *n*筆記錄
4	Exec(I1);	
5		Exec(Q1); → *n*+1筆記錄
6	commit(T1)	
7		commit(T2)

練習12-1：

- 請問右列交易的交錯執行會造成何種問題：

	T1	T2
1	begin(T1)	
2		begin(T2)
3		read(X, *b*); *b*=*b*+10
4	read(X, *a*); *a*=*a*+10	
5	read(Y, *c*)	
6		write(*b*, X)
7		write(*b*, Y)
8	write(*a*, X)	
9	read(Y, *c*)	
10	commit(T1)	
11		commit(T2) 或 abort(T2)

- Ans:
  - 若是(第11行)T2最後commit，則造成更新遺失的問題和無法重複讀取的問題
  - 若是(第11行)T2最後abort，則造成污染讀取的問題。

交易的失敗

- 交易處理的一個基本原則：全部或沒有（**All or Nothing**）
- 交易執行遭受失敗就需要復原，可能的失敗原因有：

DBMS處理交易時考量

  - **應用系統駁回**
    - 由應用系統主動提出的駁回，原因是繼續執行下去會導致現實世界裡錯誤的結果
  - **電腦系統駁回**
    - 電腦系統的軟硬體造成交易執行突然終止
  - **儲存媒體毀損**

交易執行所應具備的性質

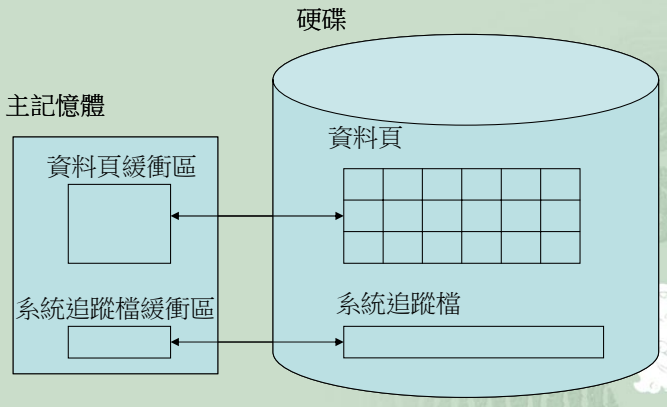
- ACID性質
  - Atomicity（單元性）：一個交易被視為一個不可分割的單元
  - Consistency（一致性）：一個交易裡的整體運算應該要滿足現實世界裡一致性的要求
  - Isolation（獨立性）：每一個交易在執行時，可將資料庫看成其專屬，而不用去考慮其他交易的存在
  - Durability（永久性）：一個交易一旦COMMIT後，其結果就該永久存在資料庫裡

系統追蹤檔（System log）

- 為了正確的處理交易COMMIT和ABORT，DBMS將交易的運算記錄存在系統追蹤檔 ，有以下五種記錄：
  - 交易開始記錄：格式為[start, 交易編號]
  - 交易資料項讀取記錄：格式為 [read, 交易編號, 資料項編號]
  - 交易資料項寫入記錄：格式為 [write, 交易編號, 資料項編號, 開始位置，寫入前的值，寫入後的值]
  - 交易COMMIT記錄：格式為 [commit, 交易編號]
  - 交易ABORT記錄：格式為 [abort, 交易編號]
- 範例
  - [start, t0001]
  - [read, t0001, p3]
  - [read, t0001, p15]
  - [read, t0001, p9]
  - [write, t0001, p9, 200, ‘英雄’, ‘狗熊’]
  - [commit, t0001]

資料和追蹤記錄

- 系統追蹤檔和資料頁分別在主記憶體裡都有緩衝區



運算動作

- 交易開始/讀取/寫入：執行運算動作後將相對的追蹤記錄寫入系統追蹤檔緩衝區裡
- 交易COMMIT：
  - COMMIT追蹤記錄寫入系統追蹤檔緩衝區
  - 將緩衝區裡所有的追蹤記錄**全部寫入硬碟**
- 交易ABORT：
  - 將ABORT追蹤記錄寫入系統追蹤檔緩衝區
  - 將其追蹤記錄的資料項寫入記錄反向執行一次，復原其寫入前的值
  - 將緩衝區裡所有的追蹤記錄**全部寫入硬碟**

系統復原動作

- 系統當機是無預警的
  - 已執行完畢（COMMIT）的交易資料必須保留
  - 正在執行中的交易被視為ABORT
- 復原動作如下：
  - 將硬碟裡系統追蹤檔裡記錄的運算動作從頭重作（Redo）一次
  - 然後再反向將被視為ABORT的交易動作還原（Undo）

系統復原動作（Cont.)

- 為避免系統追蹤檔太大和復原時間過長，DBMS可以在每隔一段時間設置一個檢查點（Checkpoint），動作如下：
  - 暫停所有交易的執行。
  - 將所有緩衝區裡的追蹤記錄寫入硬碟內。
  - 將所有更新過的資料頁從主記憶体寫入磁碟內
  - 加入一個追蹤記錄[checkpoint]，裡頭記載此時有哪些交易還在執行中，並將之寫入硬碟內。
  - 恢復交易的執行。
- 系統復原時，DBMS只要從最近的檢查點開始往後進行重作的動作

範例一

- 一開始,
  - X = 0
  - Y = 0
  - Z = 0
  - A = 0
  - B = 0
  - C= 0;
- 假設系統不主動將資料頁寫回硬碟

時間	T1	T2	T3
1	begin (T1)		
2	read (X, *x*)		
3	write (5, X)		
4		begin (T2)	
5		read (Y, *y*)	
6		write (10, Y)	
7	read (Z, *z*)		
8	write (15, Z)		
9	commit (T1)		
10		read (A, *a*)	
11		read (B, *b*)	
12		write (10, A)	
13		*checkpoint*	

範例一

- 還未執行時

	記憶體	硬碟
資料頁		X = 0, Y =0, Z = 0, A = 0, B = 0, C= 0
系統追蹤檔		

範例一

- T_1-begin, $r_1(X)$, $w_1(5, X)$, T_2-begin, $r_2(Y)$, $w_2(10, Y)$, $r_1(Z)$, $w_1(15, Z)$
- 執行commit$_1$前

	記憶體	硬碟
資料頁	X=5, Y=10, Z=15	X = 0, Y =0, Z = 0, A = 0, B = 0, C= 0
系統追蹤檔	T1-begin, r1(X), w1(0, X, 5), T2-begin, r2(Y), w2(0, Y, 10), r1(Z), w1(0, Z, 15)	

範例一

- T_1-begin, $r_1(X)$, $w_1(5, X)$, T_2-begin, $r_2(Y)$, $w_2(10, Y)$, $r_1(Z)$, $w_1(15, Z)$, **commit$_1$**
- 執行commit$_1$後

	記憶體	硬碟
資料頁	X=5, Y=10, Z=15	X = 0, Y =0, Z = 0, A = 0, B = 0, C= 0
系統追蹤檔		T1-begin, r1(X), w1(0, X, 5), T2-begin, r2(Y), w2(0, Y, 10), r1(Z), w1(0, Z, 15), **T1-commit**

範例一

- T_1-begin, $r_1(X)$, $w_1(5, X)$, T_2-begin, $r_2(Y)$, $w_2(10, Y)$, $r_1(Z)$, $w_1(15, Z)$, **commit$_1$**, $r_2(A)$, $r_2(B)$, $w_2(10, A)$, ***checkpoint***
- 執行checkpoint後

	記憶體	硬碟
資料頁	X=5, Y=10, Z=15, A=10	X=5, Y=10, Z=15, A = 10, B = 0, C= 0
系統追蹤檔		T1-begin, r1(X), w1(0, X, 5), T2-begin, r2(Y), w2(0, Y, 10), r1(Z), w1(0, Z, 15), **T1-commit** , r2(A), r2(B), w2(0, A, 10), ***checkpoint***

範例一

- T_1-begin, $r_1(x)$, $w_1(5, x)$, T_2-begin, $r_2(y)$, $w_2(10, y)$, $r_1(z)$, $w_1(15, z)$, **commit$_1$**, $r_2(a)$, $r_2(b)$, $w_2(10, a)$, ***checkpoint*** , $w_2(30, b)$, T_3-begin**,** $r_3(c)$, $w_3(40, c)$, **commit$_2$**, $r_3(a)$, $w_3(50, a)$ ||||||
- 執行到系統當機前 |||||| 時

	記憶體	硬碟
資料頁	X=5, Y=10, Z=15, A=50, B=30, C=40	X=5, Y=10, Z=15, A = 10, B = 0, C= 0
系統追蹤檔	r3(A), w3(10, A, 50)	T1-begin, r1(X), w1(0, X, 5), T2-begin, r2(Y), w2(0, Y, 10), r1(Z), w1(0, Z, 15), **T1-commit** , r2(A), r2(B), w2(0, A, 10), ***checkpoint*** , w2(0, B, 30) , T3-begin**,** r3(C), w3(0, C, 40) , **T2-commit**

範例一

- T_1-begin, $r_1(x)$, $w_1(5, x)$, T_2-begin, $r_2(y)$, $w_2(10, y)$, $r_1(z)$, $w_1(15, z)$, **commit$_1$**, $r_2(a)$, $r_2(b)$, $w_2(10, a)$, ***checkpoint*** , $w_2(30, b)$, T_3-begin**,** $r_3(c)$, $w_3(40, c)$, **commit$_2$**, $r_3(a)$, $w_3(50, a)$ ||||||
- 系統重新當機後並做完復原動作後

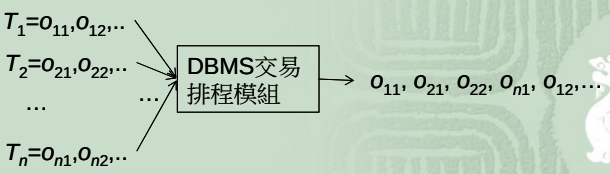
	記憶體	硬碟
資料頁	B = 30, C= 0	X=5, Y=10, Z=15, A = 10, B = 30, C= 0
系統追蹤檔	~~Undo:~~ ~~B=30~~ ~~C=0~~ Redo: B=30 C=40	T1-begin, r1(X), w1(5, X, 0), T2-begin**,** r2(Y), w2(0, Y, 10), r1(Z), w1(0, Z, 15), **T1-commit ,** r2(A), r2(B), w2(0, A, 10), ***checkpoint* ,** w2(0, B, 30), T3-begin, r3(C), w3(0, C, 40), **T2-commit**

練習12-2

- 考慮一個交易有以下的系統追蹤記錄：
  - [start, t0001]
  - [read, t0001, p3]
  - [write, t0001, p3, 135, ‘資料庫理論與實務’, ‘資料庫實務與理論’]
  - [read, t0001, p15]
  - [write, t0001, p15, 310, ‘5566專輯’, ‘7788專輯’]
  - [read, t0001, p9]
  - [write, t0001, p9, 200, ‘英雄’, ‘狗熊’]
  - [commit, t0001]
- 請問若要將這個交易所修改過的資料寫回硬碟，需寫入幾個硬碟頁？由此推論為何在COMMIT時，DBMS是選擇將該交易的系統追蹤記錄寫入硬碟，而不是將該交易所修改過的資料寫回硬碟。
- Ans:
  - 資料：3個硬碟頁
  - 系統追蹤記錄：1個硬碟頁

正確的交易交錯執行

- DBMS裡有一個交易排程模組（Transaction scheduler），交易排程模組決定交易運算的執行次序
- 但交易排程模組該根據什麼來調整交易運算次序？根據
  - 交易復原
  - 交易並行



根據交易復原來評判交易排程

- **可復原（Recoverable）的排程**
  - 一個排程S裡，對任兩個交易T_i和T_j且T_j讀取T_i的資料，若T_i的COMMIT或ABORT都在T_j的COMMIT或ABORT之前，則稱S是**可復原**的交易排程
  - 右例排程不可復原
    - T1被ABORT
    - T2應跟著被ABORT
    - 但T2已經被COMMIT

	T1	T2
1	begin(T1)	
2	read(X, *a*);	
3	write(*a*, X)	
4		begin(T2)
5		read(X, *b*);
6		write(*b*, X)
7		commit(T2)
8	***abort(T1)***	

根據交易復原來評判交易排程（Cont)

- **無連鎖駁回（Cascadingless）的排程**
  - 一個排程S裡，對任兩個交易T_i和T_j且T_j讀取T_i的資料，若T_j讀取T_i資料的運算動作都在T_i的COMMIT或ABORT之後，則稱S是**無連鎖駁回**的交易排程
  - 右例排程需連鎖駁回
    - T1被ABORT
    - T2會跟著被ABORT

	T1	T2
1	begin(T1)	
2	read(X, *a*);	
3	write(*a*, X)	
4		begin(T2)
5		read(X, *b*);
6		write(*b*, X)
7	***abort(T1)***	
8		…

根據交易復原來評判交易排程（Cont)

- **嚴格（Strict）的排程**
  - 一個排程S裡，對任兩個交易T_i和T_j且T_j讀取或寫入T_i的資料，若T_j讀取或寫入T_i資料的運算動作都在T_i的COMMIT或ABORT之後，則稱S是**嚴格**的交易排程
  - 右例排程不嚴格
    - T1被ABORT
    - 假設X的初值為100
    - X會被還原成100

	T1	T2
1	begin(T1)	
2	write(200, X);	
3		begin(T2)
4		write(300, X);
5		Commit(T2)
6	Abort(T1)	

練習12-3

- 請列出可復原的排程，無連鎖駁回的排程，和嚴格的排程之間的從屬關係
- Ans:
  - 嚴格的排程一定是無連鎖駁回的排程，而無連鎖駁回的排程也一定是可復原的排程

根據交易並行來評判交易排程

- 交易順序執行（Serial execution）便是正確的排程
- 如果交錯執行的結果可以等同於某一種順序執行的方式，應該也是正確的！這種交易排程就被稱為可順序的（Serializable）
- 兩個運算動作op1和op2如果滿足以下條件，則視為衝突：
  - op1和op2分屬不同的交易。
  - op1和op2作用在同一個資料項。
  - op1和op2至少有一個是寫入運算。

等同（Equivalent）交易排程

- 如果兩個交易排程S1和S2裡的每一對衝突運算次序都相同，則稱S1和S2為等同交易排程

	T1	T2
1	begin(T1)	
2	read(X, *a*);	
3	write(X, *a*)	
4		begin(T2)
5		read(X, *b*);
6		write(*b*, X)
7	write(*c*, Y)	
8		read(Y, *d*)
		commit(T2)
	commit(T1)	

=

	T1	T2
1	begin(T1)	
2	read(X, *a*);	
3	write(X, a)	
4		begin(T2)
5		read(X, *b*);
6	write(*c*, Y)	
7		write(*b*, X)
8		read(Y, *d*)
		commit(T2)
	commit(T1)	

可順序的（Serializable）排程

- 對於一個交易排程S，如果存在著另一個順序排程T，使得S和T為等同交易排程，則稱S為可順序的排程

	T1	T2
1	begin(T1)	
2	read(X, *a*);	
3	write(X, *a*)	
4		begin(T2)
5		read(X, *b*);
6		write(*b*, X)
7	write(*c*, Y)	
8		read(Y, *d*)
		commit(T2)
	commit(T1)	

=

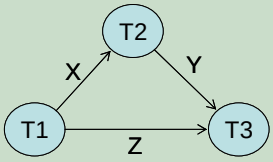
	T1	T2
1	begin(T1)	
2	read(X, *a*);	
3	write(X, a)	
4	write(*c*, Y)	
5	commit(T1)	
6		begin(T2)
7		read(X, *b*);
8		write(*b*, X)
		read(Y, *d*)
		commit(T2)

練習X-4

- 考慮*n*個交易的執行，請問有幾種順序執行的方式？
- Ans:共有*n*!個

驗證排程的可順序性

- 一個排程可表示成可順序圖，用來表示交易間的衝突運算的先後次序
  - 節點為交易
  - 有向邊表示交易的先後次序
- 若可順序圖裡不存在迴圈，該排程就是可順序的排程

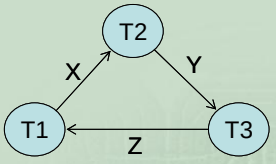


T1→T2→T3

	T1	T2	T3
1	begin(T1)		
2	read(X, *a*);		
3	write(*a*, X)		
4		begin(T2)	
5			begin(T3)
6	write(*c*, Z)		
7		read(X, *b*);	
8			read(Z, *d*);
9		write(*b*, Y)	
10		commit(T2)	
11			read(Y, *e*);
12			write(*e*, Z)
13			commit(T3)
14	commit(T1)		

驗證排程的可順序性（Cont.)

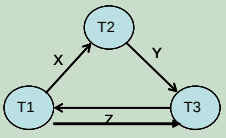
	T1	T2	T3
1	begin(T1)		
2	read(X, *a*);		
3	write(*a*, X)		
4		begin(T2)	
5			begin(T3)
6		read(X, *b*);	
7			read(Z, *d*);
8		write(*b*, Y)	
9		commit(T2)	
10			read(Y, *e*);
11			write(*e*, Z)
12	write(*c*, Z)		
13			commit(T3)
14	commit(T1)		



此排程為不可順序排程

練習X-5

- 請問右列交易排程是否是可順序的？
- Ans:否



	T1	T2	T3
1	begin(T1)		
2	read(X, *a*)		
3	write(*a*, X)		
4		begin(T2)	
5			begin(T3)
6		read(X, *b*);	
7			read(Z, *d*);
8	write(*c*, Z)		
9		write(*b*, Y)	
10		commit(T2)	
11			read(Y, *e*);
12			write(*e*, Z)
13			commit(T3)
14	commit(T1)		

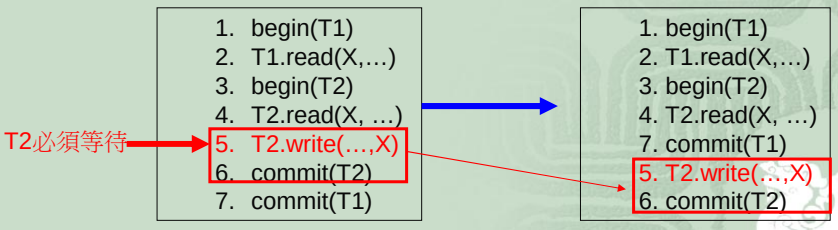
嚴格兩階段鎖定法

- 基本鎖定機制
  - 每一個資料項都有附屬一個鎖（Lock），有三種可能值
    - SHARED_LOCKED（**分享鎖定**）:讀取資料項前
    - EXCLUSIVE_LOCKED（**獨佔鎖定**）：寫入資料項前
    - UNLOCKED （**沒有鎖定**)：資料項處理完畢後

	UNLOCKED	SHARED_LOCKED	EXCLUSIVE_LOCKED
讀取	允許（並設成SHARED_LOCKED）	允許	不允許（並等待）
寫入	允許（並設成EXCLUSIVE_LOCKED）	不允許（並等待）	不允許（並等待）
解除鎖定	N/A	若已經沒有交易讀取，則設為UNLOCKED	設為UNLOCKED

嚴格兩階段鎖定法

- 嚴格兩階段鎖定
  - 遵守基本鎖定機制
  - 運算動作執行完後並不馬上解除相關資料項的鎖定，而是到最後COMMIT或ABORT時才一起解除



多層次鎖定

- 一個資料項可以是整個資料庫、一個檔案、一個資料表、一個硬碟頁、一筆記錄，甚至是一個欄位
  - 小單位的好處：可允許較多的交易並行處理，減少不必要的鎖定
  - 大單位的好處：減少鎖定的負荷
- 彈性的作法是將這些資料項的單位依大小視為一個階層
  - 資料庫 →檔案→資料表→硬碟頁→記錄
- 處理查詢時，視所需存取的資料量和位置，決定鎖定的單位和種類。

多層次鎖定（Cont.)

- 對於一個資料項X，有五種可能的鎖定方式
  - IS (Intentional Shared)：表示將讀取層次在X下的某個資料項
  - IX（Intentional Exclusive）：表示將修改層次在X下的某個資料項
  - S （Shared）：表示將讀取資料項X
  - X （eXclusive）：表示其將修改資料項X
  - SIX （Shared-Intentional-eXclusive）：表示將讀取資料項X並修改層次在X下的某個資料項。
- 依所要存取的記錄之存在位置和數量將鎖定 (S 或X)設定在適當的單位，而該單位的所有上級單位也要設成適當的鎖定 (IS、IX，或SIX)
- 鎖定解除的時機則如同兩階段鎖定法般是在交易COMMIT 或ABORT 時

多層次鎖定（Cont.)

- 修改位於(db, t1, p1)的一筆記錄r11和位於(db, t1, p2)的另一筆記錄r21
  - IX(db), IX(t1), IX(p1), IX(p2), X(r11), X(r21)
- 讀取位於(db, t2, p3)的一筆記錄r31和位於(db, t3, p4)裡的所有記錄
  - IS(db), IS(t2), IS(p3), S(r31), IS(t3), S(p4)
- 讀取位於(db, t5)裡的所有記錄，並修改(db, t5，p5)裡的所有記錄
  - IX(db), SIX(t5), X(p5)

鎖定狀態 鎖定要求	IS	IX	S	SIX	X
IS	允許	允許	允許	允許	不允許
IX	允許	允許	不允許	不允許	不允許
S	允許	不允許	允許	不允許	不允許
SIX	允許	不允許	不允許	不允許	不允許
X	不允許	不允許	不允許	不允許	不允許

多層次鎖定

- 考慮以下三個交易
  - T1: 修改位於(db, t1, p1)的一筆記錄r11
    - T1: IX(db), IX(t1), IX(p1), X(r11)
  - T2:讀取位於(db, t1, p1)的一筆記錄r12
    - IS(db), IS(t1), IS(p1), S(r12)
  - T3: 讀取(db, t1，p1)裡的所有記錄
    - IS(db), IS(t1), S(p1)
- 相容性
  - T1與T2沒有衝突
  - T2與T3沒有衝突
  - T1與T3衝突

SQL對交易的支援

- SET TRANSACTION
  - 不同的資料庫應用系統對於交易執行的正確性要求可能有別
  - 範例指令：
    - SET TRANSACTION REPEATABLE READ

	污染讀取的問題	無法重複讀取的問題	幽靈資料的問題
READ UNCOMMITTED	√	√	√
READ COMMITED	×	√	√
REPEATABLE READ	×	×	√
SERIAZABILITY	×	×	×

SQL對交易的支援

- COMMIT/ROLLBACK
  - SQL不用ABORT這個字，改用ROLLBACK
- SAVEPOINT
  - SQL允許在交易執行過程中設暫存點（Savepoint），當交易執行發生問題時，可以選擇讓資料庫回復到某一個暫存點的資料庫狀態

PL/SQL 範例

```
DECLARE
  tno  STRING;
  pno  STRING;
  ...
BEGIN
  ...
  SAVEPOINT do_product_insert;
  INSERT INTO Product VALUES (pno, ...);
  SAVEPOINT do_transaction_insert;
  INSERT INTO Transaction VALUES (tno, ...);
  …
  if salesprice < 100 ROLLBACK TO do_transaction_insert;
  INSERT INTO Record (tno, pno, ...);
  ...
  COMMIT;
EXCEPTION
  WHEN DUP_VAL_ON_INDEX THEN
    ROLLBACK;
  ...
END;
```