Database System

北京交通大学软件学院

王方石　教授

E-mail: fshwang@bjtu.edu.cn

Chapter 6 Database Security

6.1 Concepts

6.2 Privileges（权限）

6.3 Grant （赋予权限）

6.4 Revoke（收回权限）

6.1 Concepts

Security – **The mechanisms that protect the database against intentional or accidental threats.**

Threat – **Any situation or event, whether intentional or accidental, that may adversely affect a system and consequently the organization.**

e.g. attempts to steal, modify or destroy data

Authorization（授权）

- **Authorization is a mechanism that determines whether a user is who he or she claims to be.**
- **Authorization Controls are sometimes referred to as access controls.**
- **It is used to determine which objects (table, view) user may reference and what operations may be performed on those objects.**
- **Each object created in SQL has an owner, as defined in AUTHORIZATION clause of schema to which object belongs.**

CREATE SCHEMA student AUTHORIZATION wang;

- **Owner is the only person who create the object.**

6.2 Privileges

Actions that user are permitted to carry out on a given base table or view:

SELECT **Retrieve data from a table.**

INSERT **Insert new rows into a table.**

UPDATE **Modify rows of data in a table.**

DELETE **Delete rows of data from a table.**

REFERENCES **Reference columns of named table in integrity constraints.**

USAGE **Use domains, character sets, etc.**

Privileges

- **Privileges can restrict INSERT /UPDATE /REFERENCES to the named columns.**
- **The owner of a table must grant other users the necessary privileges using GRANT statement.**
- **To create view, a user must have SELECT privilege on all tables that make up the view and REFERENCES privilege on the named columns.**

6.3 GRANT

```
GRANT {PrivilegeList | ALL PRIVILEGES}
ON     ObjectName
TO     {AuthorizationIdList | PUBLIC}
[WITH GRANT OPTION]
```

- ***PrivilegeList*** **consists of one or more of above privileges separated by commas.**
- **ALL PRIVILEGES grants all privileges to a user.**

GRANT

- **PUBLIC allows access to be granted to all present and future authorized users.**

- ***ObjectName*** **can be a base table, view, trigger or character set.**

- **WITH GRANT OPTION allows privileges to be passed on.**

Example 6.1/6.2- GRANT

Give Manager full privileges to Staff table.

```
GRANT ALL PRIVILEGES
ON Staff
TO Manager WITH GRANT OPTION;
```

Give the users *Personnel* and *Director* SELECT and UPDATE on column *salary* of Staff.

```
GRANT SELECT, UPDATE (salary)
ON Staff
TO Personnel, Director;
```

Example 6.3 - GRANT Specific Privileges to PUBLIC

Give all users SELECT on Branch table.

```
GRANT SELECT
ON Branch
TO PUBLIC;
```

6.4 REVOKE

- **REVOKE takes away privileges granted with GRANT.**

```
REVOKE [GRANT OPTION FOR]
 {PrivilegeList | ALL PRIVILEGES}
ON ObjectName
FROM {AuthorizationIdList | PUBLIC}
   [RESTRICT | CASCADE]
```

- **ALL PRIVILEGES refers to all privileges granted to a user by user revoking privileges.**

REVOKE

- **GRANT OPTION FOR allows privileges passed on via WITH GRANT OPTION of GRANT to be revoked separately from the privileges themselves.**
- **REVOKE fails if it results in an abandoned object, such as a view, unless the CASCADE keyword has been specified.**
- **Privileges granted to this user by other users are not affected.**

Example 6.4/5 - REVOKE Specific Privileges

Revoke privilege SELECT on Branch table from all users.

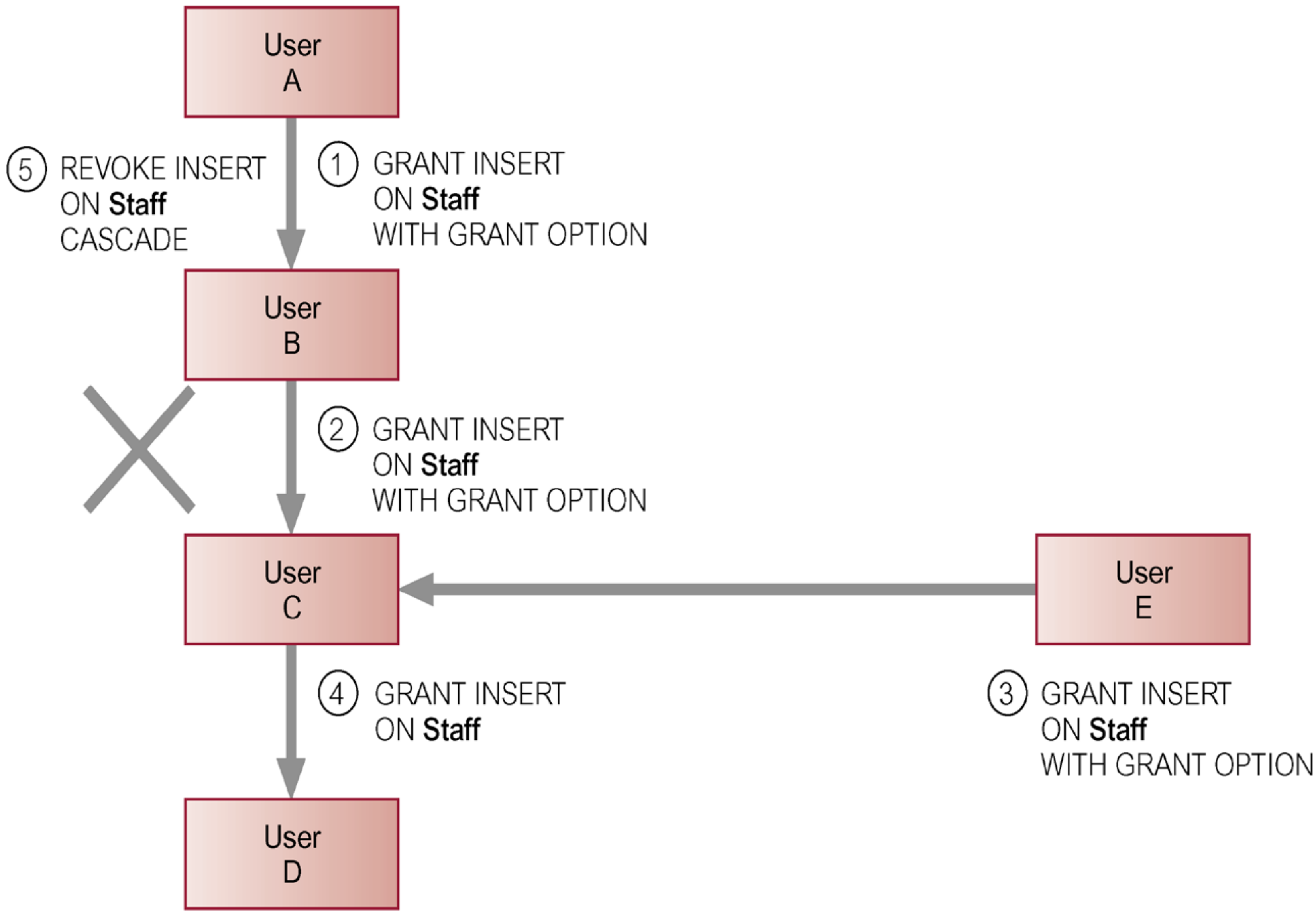
```
REVOKE SELECT
ON Branch
FROM PUBLIC;
```

Revoke all privileges given to *Director* on Staff table.

```
REVOKE ALL PRIVILEGES
ON Staff
FROM Director;
```

```
REVOKE  GRANT OPTION FOR ALL PRIVILEGES
ON Staff   FROM Director  CASCADE;
```

REVOKE


User A
5 REVOKE INSERT ON Staff CASCADE
1 GRANT INSERT ON Staff WITH GRANT OPTION
User B
2 GRANT INSERT ON Staff WITH GRANT OPTION
User C
User E
4 GRANT INSERT ON Staff
3 GRANT INSERT ON Staff WITH GRANT OPTION
User D

Chapter 7 Concurrency Control

7.1 Concept and Characteristics of Transaction

7.2 3 Potential Problems Caused by Concurrency

7.3 Serializability（可串行化）

7.4 Locking & 2PL

7.5 Isolation Levels

7.1 Concept and Characteristics of Transaction

7.1.1. Transactions

【Definition】

Action, or series of actions, carried out by a single user or application program,, which reads or updates contents of.

- **It is a logical unit of work on the database.**
- **Transaction should transform database from one consistent state to another, although consistency may be violated during transaction.**

7.1.1. Transactions

- **consistent state** means data to be “correct (correctness)”, ”valid (validity)” or “compatible (compatibility)” at all times.
- **Transaction are different from program.**
- **A transaction can be one SQL statement or a set of SQL statements.**
- **Generally, one program may include multiple transactions.**
- **Application program is series of transactions with non-database processing in between.**

- **In SQL, an transaction automatically begins with "BEGIN TRANSACTION" or *a transaction-initiating* SQL statement (e.g., SELECT, INSERT).**
- A transaction in SQL **ends** by:
  - **Commit** commits current transaction and begins a new one.

    The committed transaction cannot be aborted.

  - **Rollback** causes the current transaction to be aborted.

    The aborted transaction that is rolled back can be restarted later.

COMMIT & ROLLBACK

- **Transaction can have one of two outcomes:**
  - **Success - transaction has *committed* and database reaches a new consistent state.**
  - **Failure - transaction *aborts*, and database must be restored to consistent state before *aborting* started. Such a transaction is *rolled back* or *undone*.**
- **The committed transaction cannot be aborted.**
- **The aborted transaction that is rolled back can be restarted later.**

7.1.2 Properties of Transactions

Four basic (*ACID*) properties of a transaction are:

- **Atomicity** ‘**All** or **nothing**’ property.
- **Consistency** Must transform database from one consistent state to another.
- **Isolation** Transactions execute **independently** of one another. Partial effects of incomplete transactions should **not** be **visible** to other transactions.
- **Durability** (Permanence) Effects of a committed transaction are **permanent** and must **not** be **lost** because of later failure.

Notes

- **SQL transactions cannot be nested.**
- **Changes made by one transaction are not visible to other concurrently（并发地）executing transactions until transaction completes.**

7.2 Three Potential Problems Caused by Concurrency

The definition of Concurrency Control

It is the process of managing simultaneous operations on the database without having them interfere with one another.

- **It prevents interference when two or more users are accessing database simultaneously and at least one is updating data.**
- **Although two transactions may be correct in themselves, interleaving of operations may produce an incorrect result.**

7.2 Three Potential Problems Caused by Concurrency

- **Lost update problem.**
- **Uncommitted dependency problem.**
  i.e. Dirty Read
- **Inconsistent analysis problem.**
  i.e. Non-repeatable Read
  phantom read

(1) Lost Update Problem（丢失更新）

The successfully completed update is overridden by another user.

Time	T_1	T_2	bal_x
t_1		begin_transaction	100
t_2	begin_transaction	read(bal_x)	100
t_3	read(bal_x)	$bal_x = bal_x + 100$	100
t_4	$bal_x = bal_x - 10$	write(bal_x)	200
t_5	write(bal_x)	commit	90
t_6	commit		90

Loss of T_2's update can be avoided by preventing T_1 from reading bal_x until after update.

(2) Uncommitted Dependency Problem

Dirty Read（未提交依赖、读脏数据）

The problem occurs when one transaction can see intermediate results of another transaction before it has committed.

Dirty data：not committed and then cancelled data.

Time	T_3	T_4		bal_x
t_1		begin_transaction		100
t_2		read(bal_x)		100
t_3		$bal_x = bal_x + 100$		100
t_4	begin_transaction	write(bal_x)		200
t_5	read(bal_x)	⋮	Dirty data	$\boxed{200}$
t_6	$bal_x = bal_x - 10$	rollback		100
t_7	write(bal_x)			190
t_8	commit			190

This Problem can be avoided by preventing T_3 from reading bal_x until after T_4 commits or aborts.

(3) Inconsistent Analysis Problem

AKA Non-repeatable Read or phantom read

（不一致分析问题：不可重复读、幻读）

- **The problem occurs when the first transaction reads several values but the second transaction updates some of them during execution of the first.**

- **Include Two cases:**
  - ***Non-repeatable read* (updated by other T)**
  - ***phantom read* (inserted or deleted by other T) .**

(3) Inconsistent Analysis Problem

- when a transaction **T rereads** a data item it has previously read, but, in between, another transaction has **modified** it. Thus, T receives two different values for the same data item. This is sometimes referred to as a **nonrepeatable** (or **fuzzy**) **read**.
- If transaction T executes a **query** that retrieves a set of tuples from a relation satisfying a certain predicate,

  re-executes the query at a later time, but finds that the retrieved set contains an additional (**phantom**) tuple that has been **inserted** by another transaction in the meantime. This is sometimes referred to as a **phantom read**.

Example of Inconsistent Analysis Problem
Non-repeatable Read

Time	T_5	T_6	bal_x	bal_y	bal_z	sum
t_1		begin_transaction	100	50	25	$\boxed{175}$
t_2	begin_transaction	sum = 0	100	50	25	0
t_3	read(bal_x)	read(bal_x)	100	50	25	0
t_4	$bal_x = bal_x - 10$	sum = sum + bal_x	100	50	25	100
t_5	write(bal_x)	read(bal_y)	90	50	25	100
t_6	read(bal_z)	sum = sum + bal_y	90	50	25	150
t_7	$bal_z = bal_z + 10$		90	50	25	150
t_8	write(bal_z)		90	50	35	150
t_9	commit	read(bal_z)	90	50	35	150
t_{10}		sum = sum + bal_z	90	50	35	185
t_{11}		commit	90	50	35	$\boxed{185}$

This Problem can be avoided by preventing T_6 from reading bal_x and bal_z until after T_5 completed updates.

Example of Inconsistent Analysis Problem
phantom read

T_7 | T_8

Select * from student

Insert into student
Values('s3','wang',18,'f')

Select * from student

sno	sname	age	sex
s1	FENG	19	m
s2	LIU	20	m

sno	sname	age	sex
s1	FENG	19	m
s2	LIU	20	m
s3	wang	18	f

- **This Problem can be avoided by preventing T_8 from inserting a new tuple into table *student* until after T_7 finishes.**

Notes

- **The above problems are caused by that concurrent operations destroy isolation property of transactions.**

- **The objective of a concurrency control is to schedule transactions in such a way as to avoid any interference between them.**

- **One obvious solution is to run transactions serially, but this limits degree of concurrency in system.**

7.3 Serializability（可串行化）

Schedule **is a sequence of the operations by a set of concurrent transactions that preserves the order of the operations in each of the individual transactions.**

- a **schedule** for a set of transactions must **consist** of **all instructions** of the concurrent transactions.
- a **schedule** must **preserve** the **order** in which the instructions appear in each individual transaction.

Serial Schedules （串行调度）

【definition】A schedule where operations of each transaction are executed consecutively without any interleaved operations from other transactions.

There is no guarantee that results of all serial executions of a given set of transactions will be identical. But every serial execution is considered correct.

Time	T_1	T_2	bal_x
t_1		begin_transaction	100
t_2	begin_transaction	read(bal_x)	100
t_3	read(bal_x)	$bal_x = bal_x * 2$	100
t_4	$bal_x = bal_x - 10$	write(bal_x)	200
t_5	write(bal_x)	commit	90
t_6	commit		90

T1->T2 balx=180; T2->T1 balx=190

Nonserial Schedule （非串行调度）

A schedule where operations from a set of concurrent transactions are interleaved.

serializable schedule（可串行化调度）

- **Every serial execution is considered correct, although different results may be produced.**

- **If a nonserial schedule produces the same results as some serial execution, then the nonserial schedule is called *serializable* schedule（可串行化调度）.**

- **A serializable schedule is considered to be a correct schedule for the concurrent transactions .**

- **We want to find a serializable (correct) schedule for the concurrent transactions.**

The objective of serializability

- **The objective of serializability is to find nonserial schedules that allow transactions to execute concurrently without interfering with one another,** and thereby produce a database state that could be produced by a **serial execution**.
- **The above objective is same with the objective of concurrency control.**
- **Whether a concurrent schedule is correct (serializable) or not is depend on whether its result is same with that of a certain Serial Schedule.**

Example1: serial schedules

Let T_1 transfer $50 from A to B,

and T_2 transfer 10% of the balance from A to B.

The following is two serial schedules.

T_1	T_2
read(A)	
$A := A - 50$	
write (A)	
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)

A=100, B=200

Case 1: If T1->T2

A=45, B=255

Case 2: If T2->T1

A=40, B=260

Example2 : Concurrent Schedule 1

Let T_1 and T_2 be the transactions defined previously.

The following is concurrent schedule, not a serial schedule.

it is *equivalent* to Case 1 of Schedule 1.

Case 1: T1->T2

A=45, B=255

Case 2: T2->T1

A=40, B=260

T_1	T_2
read(A) $A := A - 50$ write(A)	
	read(A) $temp := A * 0.1$ $A := A - temp$ write(A)
read(B) $B := B + 50$ write(B)	
	read(B) $B := B + temp$ write(B)

A=100, B=200

Result:

A=45, B=255

same with case 1 of serial schedule

***serializable* schedule**

Example3 : Concurrent Schedule 2

The following concurrent schedule does not preserve the value of the sum *A* + *B*.

T_1	T_2
read(A)	
$A := A - 50$	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
write(A)	
read(B)	
$B := B + 50$	
write(B)	
	$B := B + temp$
	write(B)

A=100, B=200

Case 1: T1->T2
A=45, B=255

Case 2: T2->T1
A=40, B=260

Result:
A=50, B=210

not same with either case of serial schedules

Non-serializable **schedule**

7.4 Locking& 2PL

7.4.1 Locking

- **Basic concurrency control technique*: Locking***
- **A lock is a procedure used to control concurrent access to a data item.**
- **When one transaction T is accessing a data item A, a lock may deny modifying A by other transactions to prevent incorrect results.**
- **Before T release its lock on A, other transactions can not modify A.**
- **Locking methods are the most widely used approach to ensure serializability of concurrent transactions.**

Lock Types

Data items can be locked in two modes :

1. ***shared (S) lock*****(共享锁、读锁、S锁)**.

   If a transaction **T** has a shared lock on a data item **A**, it can read **A** but not update it. Other transactions can only have a S-lock on **A**, not a X-lock on it until T release the S-lock on **A**.

2. ***exclusive (X) lock*** **(排他锁、互斥锁、写锁、X锁)**.

   If a transaction **T** has an exclusive lock on a data item **A**, it can both read and update **A**. Other transactions can not have any type of lock on **A** until T release the X-lock on **A**.

Locking

- **A transaction must claim a *shared* (*read*) or *exclusive* (*write*) lock on a data item to concurrency-control manager before read or write.**

- **Transaction can proceed only after the requested lock is granted, otherwise the transaction must wait until the existing the lock is released.**

Lock-compatibility matrix

Reads cannot conflict, so more than one transaction can hold the shared locks simultaneously on the same item.

T_1 \ T_2	X	S	–
X	N	N	Y
S	N	Y	Y
–	Y	Y	Y

Y=Yes， compatible request
N=No， incompatible request

symmetric matrix

Exclusive lock gives transaction exclusive access to that data item.

Locking - Basic Rules

Some systems allow a transaction to upgrade a read lock to a write lock, or downgrade an exclusive lock to a shared lock.

Example – Two Transactions

initial values: $bal_x = 100$, $bal_y = 400$

T9	T10
$bal_x = bal_x + 100;$	$bal_x = bal_x * 1.1;$
$bal_y = bal_y - 100;$	$bal_y = bal_y * 1.1;$

Results of two serial schedules

– $bal_x = 220$, $bal_y = 330$, if T_9 executes before T_{10}
– $bal_x = 210$, $bal_y = 340$, if T_{10} executes before T_9

Example - Incorrect Locking Schedule

Time	T_9	T_{10}
t_1	WLOCK(bal_x)	
t_2	read(bal_x)	
t_3	$bal_x = bal_x + 100$	
t_4	write(bal_x)	
t_5	UNLOCK(bal_x)	
t_6		WLOCK(bal_x)
t_7		read(bal_x)
t_8		$bal_x = bal_x * 1.1$
t_9		write(bal_x)
t_{10}		UNLOCK(bal_x)
t_{11}		WLOCK(bal_y)
t_{12}		read(bal_y)
t_{13}		$bal_y = bal_y * 1.1$
t_{14}		write(bal_y)
t_{15}		UNLOCK(bal_y)
t_{16}		commit
t_{17}	WLOCK(bal_y)	
t_{18}	read(bal_y)	
t_{19}	$bal_y = bal_y - 100$	
t_{20}	write(bal_y)	
t_{21}	UNLOCK(bal_y)	
t_{22}	commit	

initial：
$bal_x = 100$, $bal_y = 400$

Results of serial schedules:

– **$bal_x = 220$, $bal_y = 330$**

– **$bal_x = 210$, $bal_y = 340$**

result：
$bal_x = 220$, $bal_y = 340$

It is not a serializable schedule.

Problem and solution

- **Problem is that transactions release locks too soon, resulting in loss of total isolation and atomicity.**
- **To guarantee serializability, need an additional protocol concerning the positioning of lock and unlock operations in every transaction.**
- **Solution：Two-Phase Locking protocol is introduced.**

7.4.2 Two-Phase Locking (2PL)

- **A transaction follows two-phase locking protocol if all locking operations precede the first unlock operation in the transaction.**

- **Two phases for transaction:**
  - **Growing phase （扩展阶段、加锁阶段）- acquires all locks but cannot release any locks.**
  - **Shrinking phase （收缩阶段、解锁阶段）- releases locks but cannot acquire any new locks.**

Notes

- **There is no requirement that all locks be obtained simultaneously.**
- **Normally, the transaction acquires some locks, does some processing and goes on to acquire additional locks as needed.**
- **It never release any lock until it has reach a stage where no new locks are needed. Rules:**
  - **A transaction must acquire a lock on an item before operating on the item. The type of the lock depends on the type of access needed.**
  - **It forbids a transaction to request a lock after it has unlocked anything, so unlock is implied in commit /rollback.**

Preventing Lost Update Problem using 2PL

Time	T_1	T_2		bal_x
t_1		begin_transaction	initial	$\boxed{100}$
t_2	begin_transaction	write_lock(bal_x)		100
t_3	write_lock(bal_x)	read(bal_x)		100
t_4	WAIT	$bal_x = bal_x + 100$		100
t_5	WAIT	write(bal_x)		200
t_6	WAIT	commit/unlock(bal_x)		200
t_7	read(bal_x)			200
t_8	$bal_x = bal_x - 10$			200
t_9	write(bal_x)			190
t_{10}	commit/unlock(bal_x)		100+100-10=	$\boxed{190}$

Preventing Uncommitted Dependency Problem using 2PL preventing Dirty read

Time	T_3	T_4		bal_x
t_1		begin_transaction		100
t_2		write_lock(bal_x)		100
t_3		read(bal_x)		100
t_4	begin_transaction	$bal_x = bal_x + 100$		100
t_5	write_lock(bal_x)	write(bal_x)	Dirty data	$\boxed{200}$
t_6	WAIT	rollback/unlock(bal_x)	Correct data	$\boxed{100}$
t_7	read(bal_x)			100
t_8	$bal_x = bal_x - 10$			100
t_9	write(bal_x)			90
t_{10}	commit/unlock(bal_x)			90

Preventing Inconsistent Analysis Problem using 2PL

Non-repeatable read

Time	T_5	T_6	bal_x	bal_y	bal_z	sum
t_1		begin_transaction	100	50	25	**175**
t_2	begin_transaction	sum = 0	100	50	25	0
t_3	write_lock(bal_x)		100	50	25	0
t_4	read(bal_x)	read_lock(bal_x)	100	50	25	0
t_5	$bal_x = bal_x - 10$	WAIT	100	50	25	0
t_6	write(bal_x)	WAIT	90	50	25	0
t_7	write_lock(bal_z)	WAIT	90	50	25	0
t_8	read(bal_z)	WAIT	90	50	25	0
t_9	$bal_z = bal_z + 10$	WAIT	90	50	25	0
t_{10}	write(bal_z)	WAIT	90	50	35	0
t_{11}	commit/unlock(bal_x, bal_z)	WAIT	90	50	35	0
t_{12}		read(bal_x)	90	50	35	0
t_{13}		sum = sum + bal_x	90	50	35	90
t_{14}		read_lock(bal_y)	90	50	35	90
t_{15}		read(bal_y)	90	50	35	90
t_{16}		sum = sum + bal_y	90	50	35	140
t_{17}		read_lock(bal_z)	90	50	35	140
t_{18}		read(bal_z)	90	50	35	140
t_{19}		sum = sum + bal_z	90	50	35	175
t_{20}		commit/unlock(bal_x, bal_y, bal_z)	90	50	35	175

Theorem

If the schedule follows 2PL protocol, it must be a *serializable* schedule;

Even though the schedule does not follow 2PL protocol, it may be still a *serializable* schedule.

Not following 2PL, still *serializable* schedule

Time	T_1 : H=F+1	F, G, H	T_2 : F=G+1
1		**0,0,0**	
2	**LOCK S(F)**		
3	**READ(F)**		
4	**A:=F**		
5	**UNLOCK(F)**		
6			**LOCK S(G)**
7			**READ(G)**
8			**B:=G**
9			**LOCK X(F)**
10			**F:=B+1**
11			**WRITE(F)**
12		**1,0,0**	**COMMIT**
13	**LOCK X(H)**		
14	**H:=A+1**		
15	**WRITE(H)**		
16	**COMMIT**	**1,0,1**	

Initial value： F=G=H=0

T1:H=F+1 T2:F=G+1

T1->T2, F=1,G=0,H=1

T2->T1, F=1,G=0,H=2

The result of concurrent schedule:
F=1,G=0,H=1

It is ***serializable* schedule.**

7.4.3 Deadlock

An impasse that may result when two (or more) transactions are each waiting for locks held by the other to be released.

Time	T_{17}	T_{18}
t_1	begin_transaction	
t_2	write_lock($\mathbf{bal_x}$)	begin_transaction
t_3	read($\mathbf{bal_x}$)	write_lock($\mathbf{bal_y}$)
t_4	$\mathbf{bal_x} = \mathbf{bal_x} - 10$	read($\mathbf{bal_y}$)
t_5	write($\mathbf{bal_x}$)	$\mathbf{bal_y} = \mathbf{bal_y} + 100$
t_6	write_lock($\mathbf{bal_y}$)	write($\mathbf{bal_y}$)
t_7	WAIT	write_lock($\mathbf{bal_x}$)
t_8	WAIT	WAIT
t_9	WAIT	WAIT
t_{10}	⋮	WAIT
t_{11}	⋮	⋮

7.4.3 Deadlock

- **Only one way to break deadlock: abort (撤销) one or more of the transactions.**
- **Deadlock should be transparent to user, so DBMS should restart the aborted transaction(s).**
- **Two general techniques for handling deadlock:**
  - **Timeouts (超时).**
  - **Deadlock detection and recovery (死锁检测和恢复).**

（1）Timeouts

- Transaction that requests lock will only wait for a system-defined period of time.
- If lock has not been granted within this period, lock request times out.
- In this case, DBMS assumes transaction may be deadlocked, even though it may not be, and it aborts and automatically restarts the transaction.

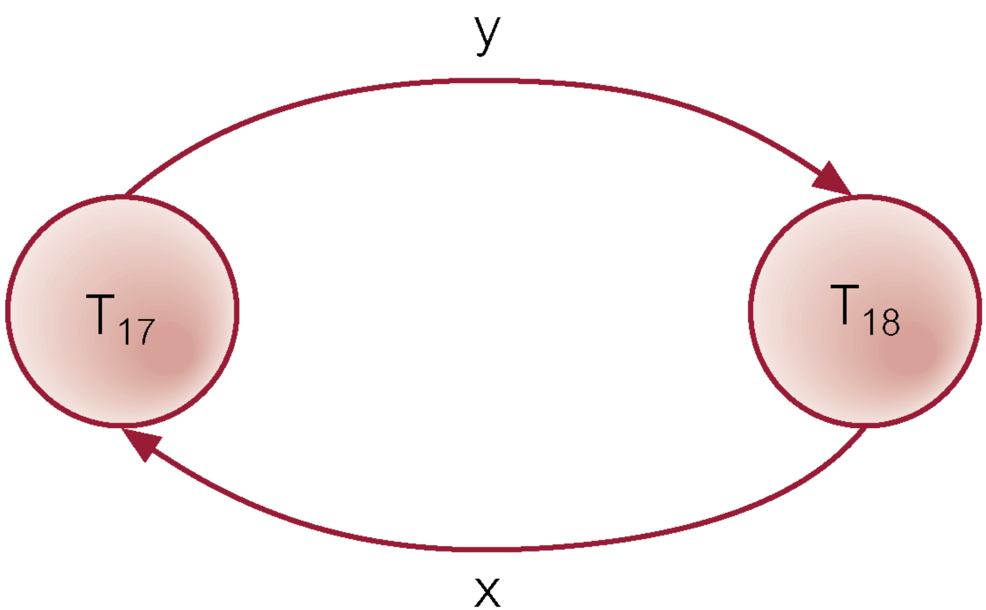
（2）Deadlock Detection and Recovery

- **DBMS allows deadlock to occur but recognizes it and breaks it.**
- **Deadlock Detection**
  - **Usually handled by construction of wait-for graph (WFG) showing transaction dependencies:**
    - **Create a node for each transaction.**
    - **Create edge T_i -> T_j, if T_i waiting to lock item locked by T_j.**
  - **Deadlock exists if and only if WFG contains cycle（回路，Topological Sorting）.**
  - **WFG is created at regular intervals and examines it for a cycle.**

（2）Deadlock Detection and Recovery

◆Recovery from deadlock detection

- **trOnce deadlock has been detected, the DBMS needs to abort one or more of the transactions.**
- **Abort the ansaction that incur the minimum cost**



Example - Wait-For-Graph (WFG)

a b h c d g f e

When implementing algorithm, the detail is as follows

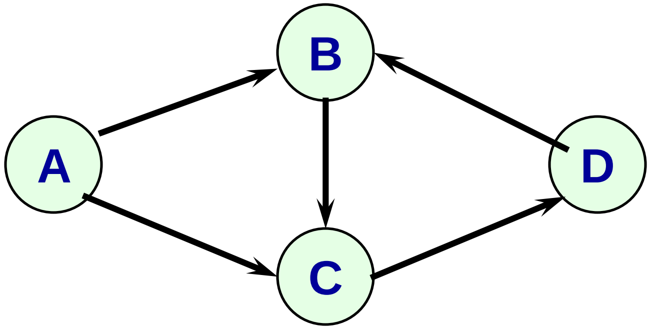
a vertex that has no incoming edge

≡≡ **a vertex with zero in-degree**

Delete this vertex and outbound edges from it

≡≡ **the in-degree of the vertex is decreased by one.**

example

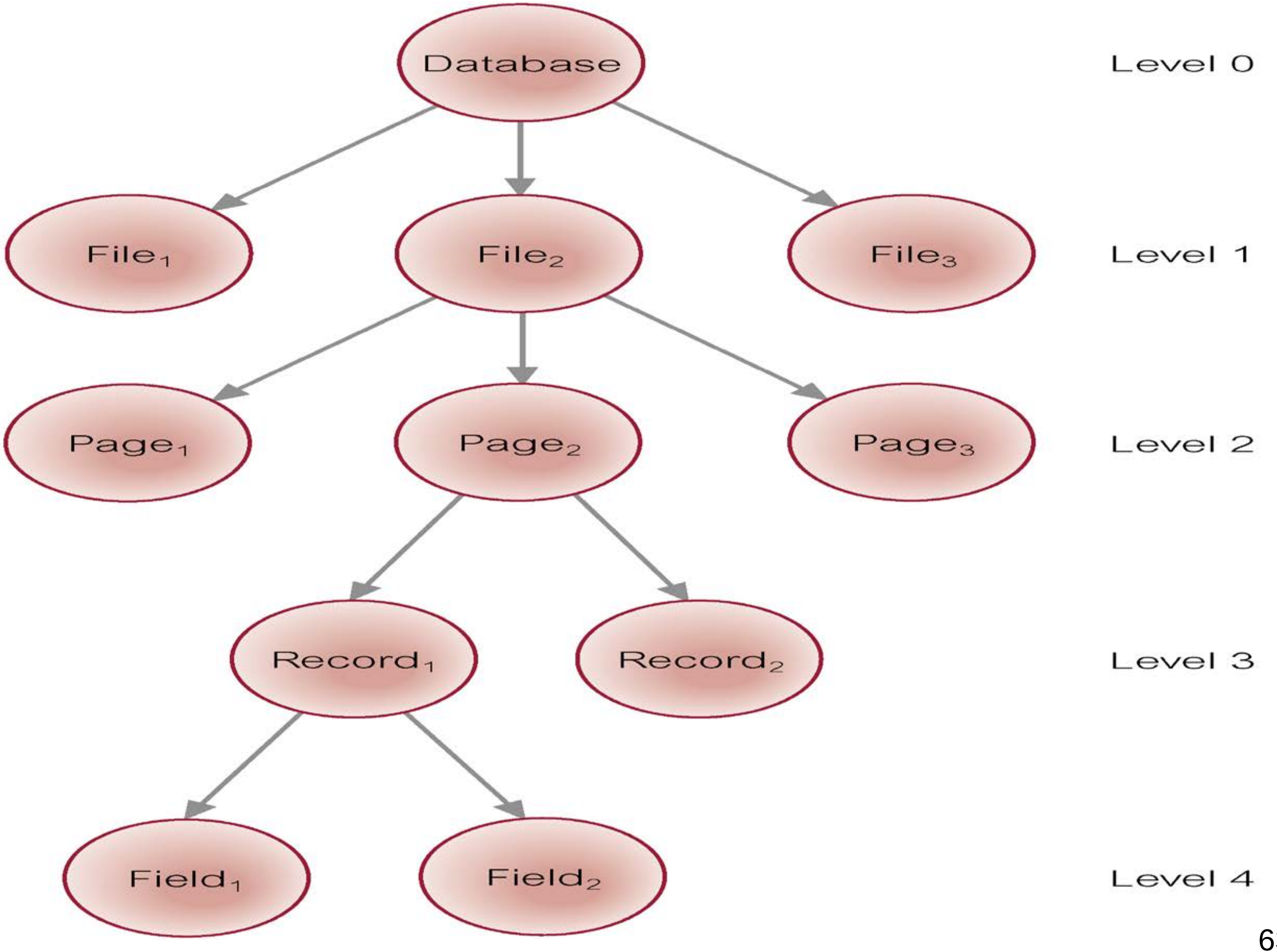


It has no topological order because there is a cycle {B, C, D} in it.

7.5 Granularity (粒度) of Data Items

- **The size of data items chosen as the unit of protection by concurrency control protocol.**
- **Ranging from coarse to fine:**
  - **The entire database.**
  - **A file.**
  - **A page (or area or database spaced).**
  - **A record (row, tuple).**
  - **An attribute (column) value of a record.**

Levels of Locking



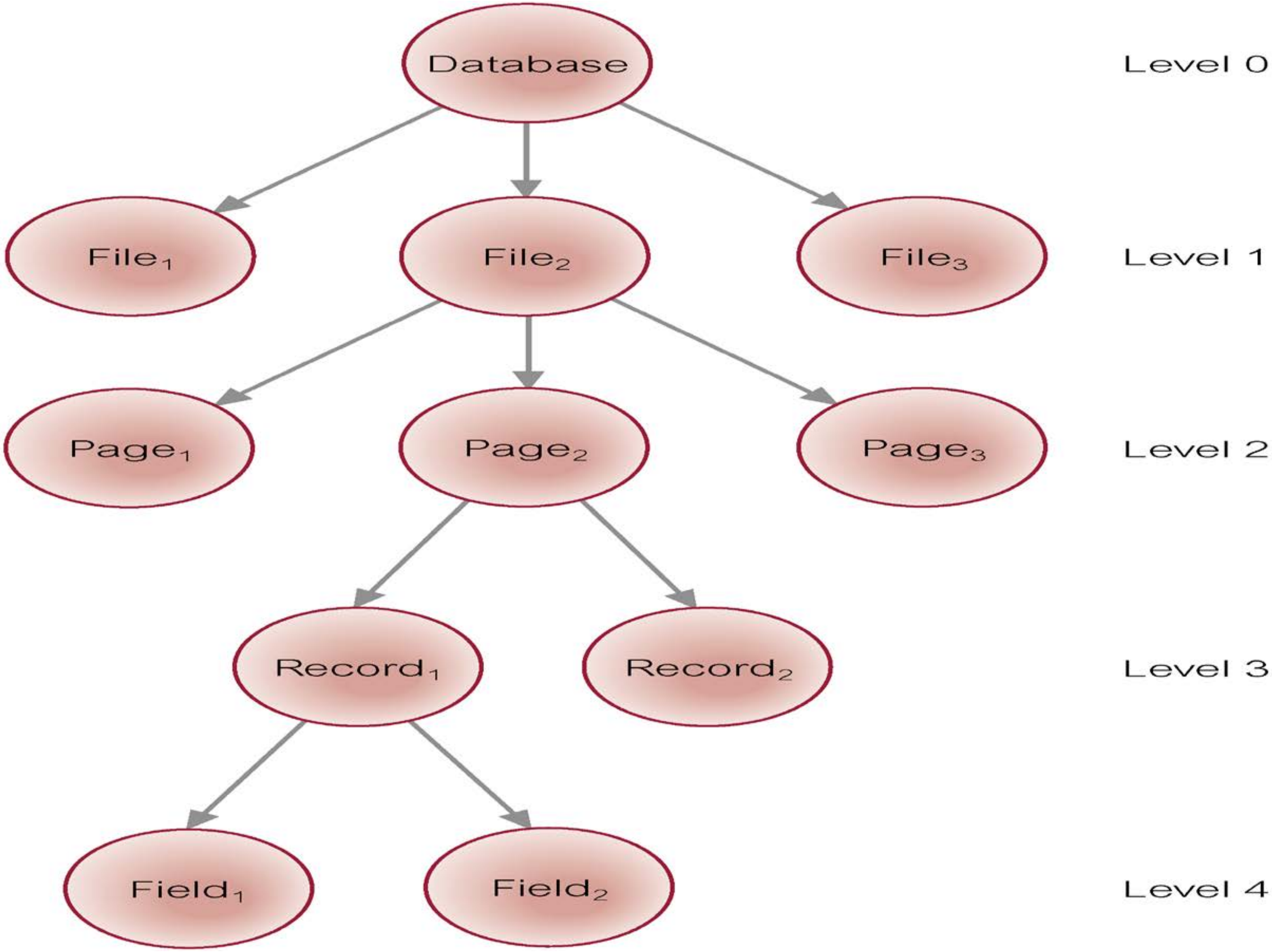
Granularity of Data Items

- **Tradeoff:**
  - **The coarser(粗) the granularity (the data item size) is, the lower the degree of concurrency is permitted;**
  - **The finer(细), the more locking information that is needed to be stored, the higher the concurrency degree.**
- **The best item size depends on the types of transactions.**
  - **If access a small number of records, better to have the granularity at the record level.**
  - **If access many records of the same file, better to have page or file granularity.**

Hierarchy of Granularity

- **It could represent granularity of locks in a hierarchical structure.**
- **The Root node represents entire database, level 1 nodes represent files, etc.**
- **When a node is locked, all its descendants are also locked.**
- **DBMS should check the hierarchical path from the root to the requested node to determine if any of its ancestors are locked before deciding whether to grant the lock.**

Levels of Locking



7.5 Isolation Levels (SQL 92)

- **ISOLATION LEVEL indicates the degree of interaction that is allowed from other transactions during the execution of the transaction.**

- **Four Levels of Isolation specified by SQL-92:**
  - **Serializable** — default
  - **Repeatable read**
  - **Read committed**
  - **Read uncommitted**

- **Only the Serializable isolation level is safe, which generates serializable schedule.**

Set Isolation levels of Transaction

SET TRANSACTION configures a transaction in the following format:

```
SET TRANSACTION
  [ READ ONLY | READ WRITE] |
  [ ISOLATION LEVEL   READ UNCOMMITTED
                     | READ COMMITTED
                     | REPEATABLE READ
                     | SERIALIZABLE ]
```

Levels of Isolation in SQL-92

Isolation level

high

- **Serializable** — default
- **Repeatable read** — — T obtains S lock before reading, and releases it until T ends. only committed records can be read, repeated reads of the same record must return the same value. However, a transaction may not be serializable – it may find some records inserted by a transaction but not find others (Phantom).
- **Read committed** — T obtains S lock before reading, but releases it immediately. only committed records can be read, but successive reads of record may return different (but committed) values i.e.Unrepeatable Read.
- **Read uncommitted** — T does not obtain S lock before reading data item, so even uncommitted records may be read. i.e. Dirty Read

Common property: T obtains X lock before writing, and releases it until the end of T.

low

Transaction Isolation Levels in SQL-92

Isolation level low → high

Concurrency degree high → low

level	Lost Update	Dirty Read	Unrepeatable Read	Phantom
READ UNCOMMITTED	No	Maybe	Maybe	Maybe
READ COMMITTED	No	No	Maybe	Maybe
REPEATABLE READ	No	No	No	Maybe
SERIALIZABLE	No	No	No	No

- **No** means this level of isolation can prevent this phenomena, there is no such a phenomena happened.
- **Maybe** means this level of isolation cannot prevent this phenomena, such a phenomena may happen.

Chapter 8 Database Recovery

8.1 Failure

8.2 Transactions and Recovery

8.3 Recovery Facilities（恢复机制）

Backup, Log files, Checkpoint

8.4 Recovery Techniques (恢复技术)

8.1 Failure

◆Failure is inevitable

- **System crashes, Soft Crash, resulting in loss of main memory.**

  due to hardware （CPU） or software （OS） errors

- **Media failures, Hard Crash, refer to hard disk failures, resulting in loss of parts of secondary storage.**

  such as a head crash or unreadable media

- **Application software error (e.g. Arithmetic Overflow)**

  such as a logical error in the program that is accessing the DB, which cause one or more transactions to fail.

- **Natural physical disasters**

  such as fire, floods, earthquakes, power failures.

- **Carelessness : unintentional destruction of data or facilities by DBAs or users.**

- **Sabotage: intentional destruction.**

8.1 Failure

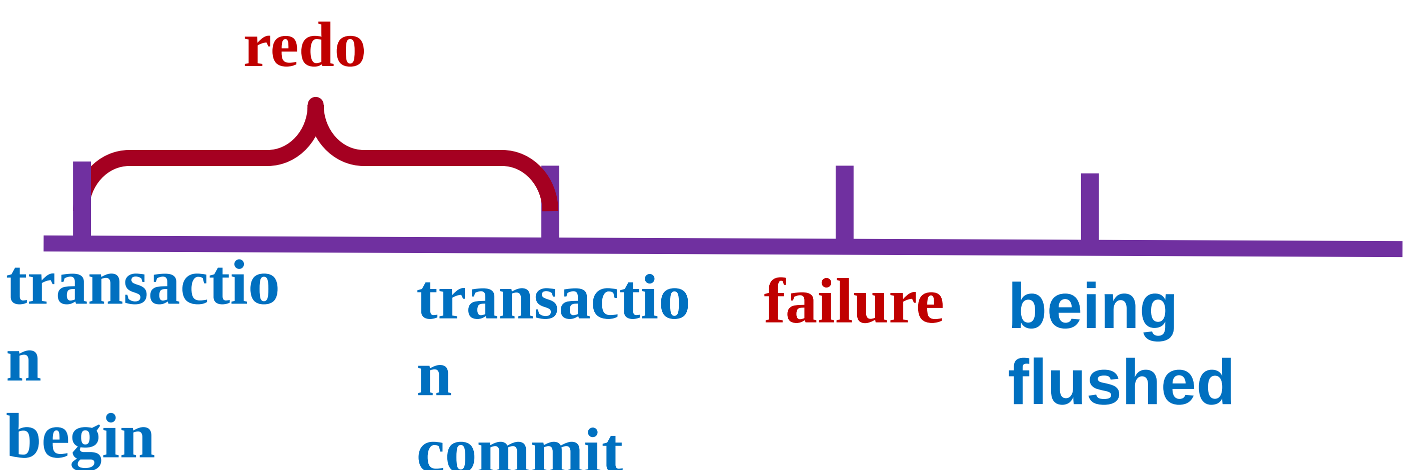
- **Consequence of Failures**
  - The loss of main memory, including the database buffer
  - The loss of disk copy of the database
- Whatever the underlying cause of the failure, the DBMS must be able to recover from the failure and restore the DB to a consistent state.
- **Definition of Database Recovery :**

  It is the process of restoring database to a correct state in the event of a failure.

8.2 Transactions and Recovery

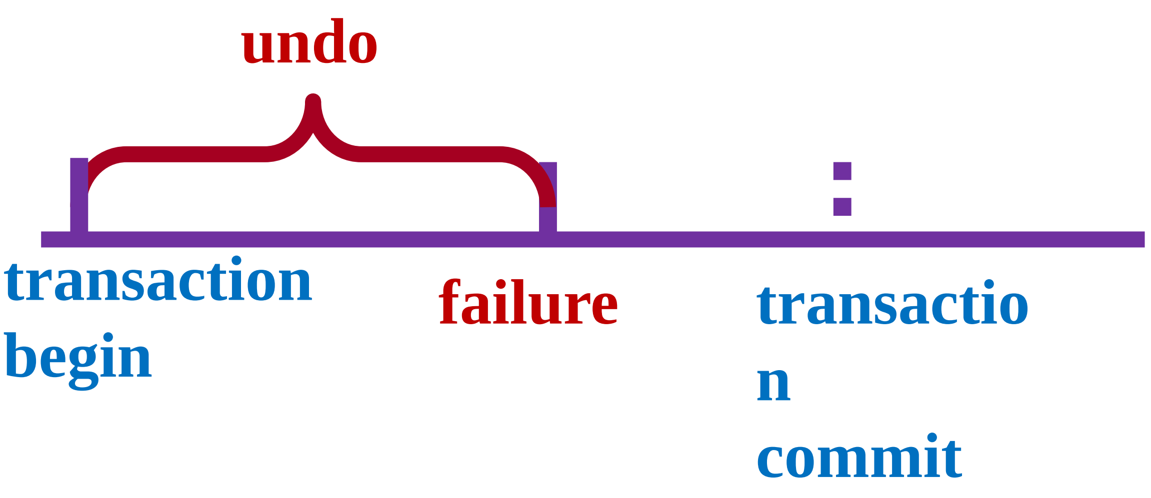
- **Transactions represent basic unit of recovery.**
- **Recovery manager is responsible for atomicity and durability.**

(1) If failure occurs between commit and database buffers being flushed to secondary storage then, to ensure durability, recovery manager has to *redo* (*rollforward*) transaction's updates.

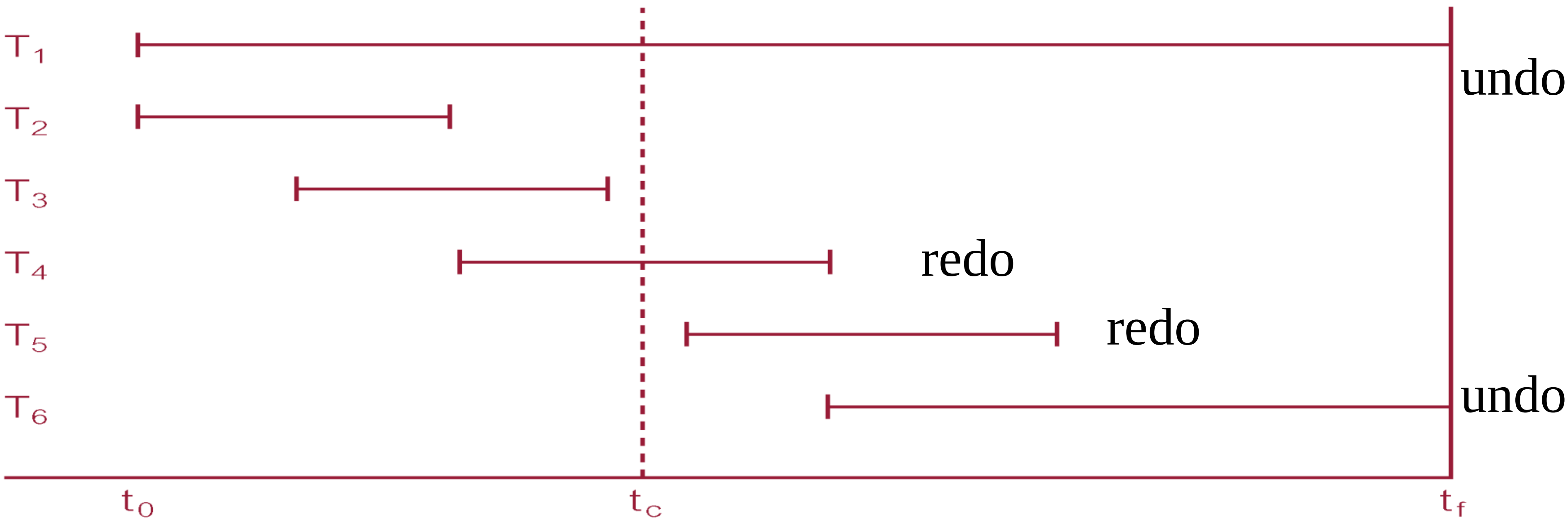


8.2 Transactions and Recovery

(2) If the transaction had not committed at failure time, then the recovery manager has to *undo* (*rollback*) any effects of that transaction for atomicity.



Example



- DBMS starts at time t_0, but fails at time t_f. Assume data for transactions T_2 and T_3 were written to secondary storage at time t_c.
- T_1 and T_6 have to be undone.
- Recovery manager has to redo T_4 and T_5.
- No need to doing anything for T_2 and T_3.

8.3 Recovery Facilities

Redundant data（backup / log file） are used to recover DB.

DBMS should provide following facilities to assist with recovery:

- **Backup mechanism, which makes periodic backup copies of database.**
- **Logging facilities, which keep track of current state of transactions and database changes.**
- **Checkpoint facility, which enables updates to the database that are in progress to be made permanent.**
- **Recovery manager, which allows DBMS to restore database to consistent state following a failure.**

（1）Backup mechanism

- The DBMS should provide a mechanism to allow backup copies of the database and the *log file* (discussed next) to be made at regular intervals without necessarily having to stop the system first.
- The backup copy of the database can be used in the event that the database has been damaged or destroyed.

（2） Log File

- **Contains information about all updates to database:**
  - **Transaction records.**
  - **Checkpoint records.**
- **Log file is often used for database recovery as well as other purposes (for example, performance monitoring and auditing).**

（2）Log File

- **Transaction records contain:**
  - **Transaction identifier.**
  - **Type of log record, (transaction start, insert, update, delete, abort, commit).**
  - **Identifier of data item affected by database action (insert, delete, and update operations).**
  - **Before-image of data item.**
  - **After-image of data item.**
  - **Log management information.**

A segment of a Log File

	Tid	Time	Operation	Object	Before image	After image	pPtr	nPtr
1	T1	10:12	START				0	2
2	T1	10:13	UPDATE	STAFF SL21	(old value)	(new value)	1	8
3	T2	10:14	START				0	4
4	T2	10:16	INSERT	STAFF SG37		(new value)	3	5
5	T2	10:17	DELETE	STAFF SA9	(old value)		4	6
6	T2	10:17	UPDATE	PROPERTY PG16	(old value)	(new value)	5	10
7	T3	10:18	START				0	11
8	T1	10:18	COMMIT				2	0
9		10:19	CHECKPOINT	T2, T3				
10	T2	10:19	COMMIT				6	0
11	T3	10:20	INSERT	PROPERTY PG4		(new value)	7	12
12	T3	10:21	COMMIT				11	0

pPtr points to its previous record in the same transaction.
nPtr points to its next record in the same transaction.

（2）Log File

- Log file may be duplexed or triplexed.
- Log file sometimes is split into two separate random-access files.
- Log file is a potential bottleneck and the speed of the writes to the log file can be critical in determining the overall performance of the DBS.
- We must follow "write- ahead logging rule".

先写日志文件

- 写数据库和写日志文件是两个不同的操作
  - 写日志文件操作：把表示这个修改的日志记录写到日志文件中。
  - 写数据库操作：把对数据的修改写到数据库中。

- 为什么要先写日志文件？
  - 在这两个操作之间可能发生故障
  - 如果先写了数据库修改，而在日志文件中没有登记下这个修改，则以后就无法恢复这个修改了
  - 如果先写日志，但没有修改数据库，按日志文件恢复时只不过是多执行一次不必要的**UNDO**操作，并不会影响数据库的正确性

（3） Checkpoint

Purpose for checkpoint:

To limit the amount of searching and subsequent processing that we need to carry out on the log file, we can use a technique called checkpointing.

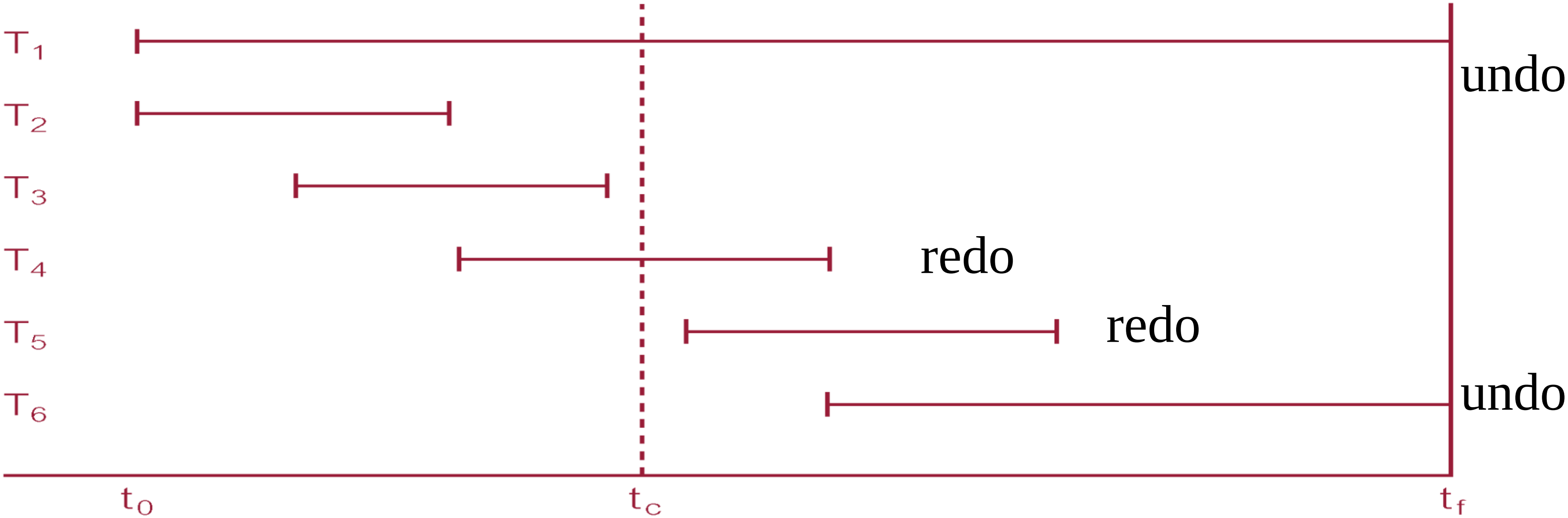
<u>Definition of Checkpoint</u>

It is the point of synchronization(同步) between the database and the transaction log file. All buffers are force-written to secondary storage.

Checkpoint are scheduled at predetermined intervals and involve the following operations:

- **Writing all log records in main memory to secondary storage;**
- **Writing the modified blocks in the database buffers to secondary storage ;**
- **Writing a checkpoint record to the log file. This record contain the identifiers of all transactions that are active at the time of the checkpoint.**
- **Checkpoint record is created containing identifiers of all active transactions.**
- **When failure occurs, redo all transactions committed since the checkpoint and undo all transactions active at time of crash.**

Example of Checkpoints



- DBMS starts at time t_0, but fails at time t_f .
- t_c Is checkpoint.
- Undo T_1 and T_6 .
- redo T_4 and T_5.
- Doing nothing for T_2 and T_3.

8.4 Recovery Techniques

The particular recovery procedure to be used is dependent to the extent of the damage (损坏程度) that has occurred to the database. We consider two cases:

- **hard crash: extensively damaged DB**
- **Soft crash: not been physically damaged database**

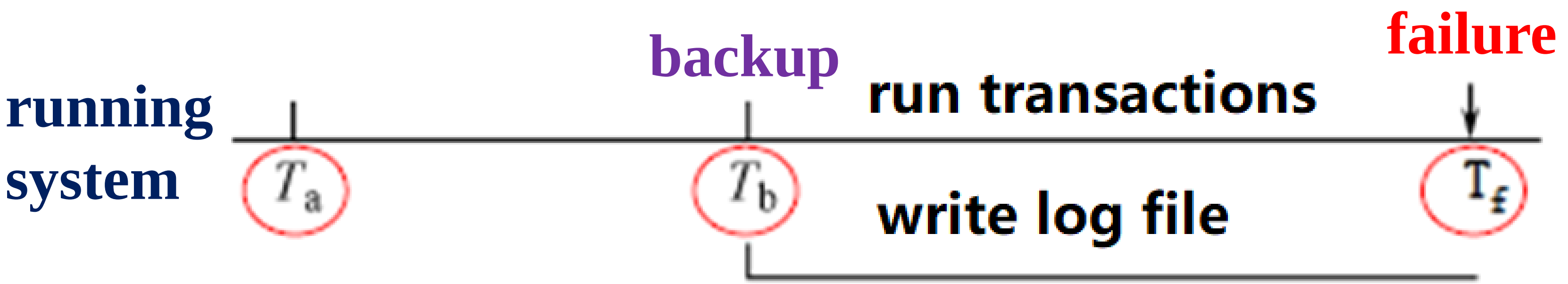
8.4 Recovery Techniques

- **hard crash**

If the database has been extensively damaged, for example a disk head crash has occurred and destroyed the database, then:

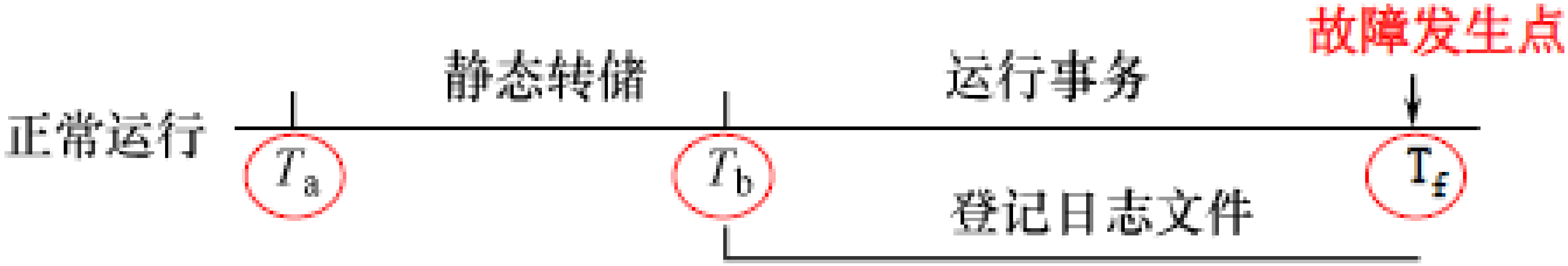
- **It needs to restore the last backup copy of database and reapply the update operations of committed transactions using the log file.**
- **This assumes that the log file has not been damaged as well.**
- **The log file should be stored on a disk separate from the main database files.**

Recover DB using Backup and log file



1. Load the latest backup of DB
2. Using log file, from the point of the latest backup
   (1) redo the committed transactions
   (2) and undo all the uncommitted transactions.

Recover DB using Backup and log file



1. **Load the latest backup of DB**
2. **Using log file, from the point of the latest backup** (利用日志文件，从最后一个备份点开始)
   (1) redo the committed transactions
   (2) and undo all the uncommitted transactions.

8.4 Recovery Techniques

- **Soft crash**

  If database has not been physically damaged but has become inconsistent, for example the system crashed while transactions were executing, then :

  - **It needs to undo the changes that caused the inconsistency. It may also need to redo some transactions to ensure that the updates have reached secondary storage.**
  - **We do not need to use the backup copy, but can restore database to a consistent state using before- and after-images held in the log file.**

Final Exam

- **I. single choice(20 pts)**
- **II. Fill blanks (10 pts)**
- **III. Brief questions(10 pts)**

  include chapter 1,2,6,7,8

- **IV. Problem Analysis (20 pts) Chapter 4-5**
- **V. SQL (20 pts) Chapter 3**
- **VI. DB Design (20 pts) Chapter 5**