13.4 Consider the relations $r_1(A, B, C)$, $r_2(C, D, E)$, and $r_3(E, F)$, with primary keys A, C, and E, respectively. Assume that r_1 has 1000 tuples, r_2 has 1500 tuples, and r_3 has 750 tuples. Estimate the size of $r_1 \bowtie r_2 \bowtie r_3$, and give an efficient strategy for computing the join.

We could use the associative role:

$$r_1 \bowtie r_2 \bowtie r_3 = ((r_1 \bowtie r_2) \bowtie r_3) \quad ①$$
$$= (r_1 \bowtie (r_2 \bowtie r_3)) \quad ②$$

Since Primary Keys are A, C, and E, it could be concluded that

$$\text{tuples} = \min\left(\max(\min(1000, 1500), 750),\ \max(1000, \min(1500, 750))\right)$$

Therefore, we could use solution ①, the size is 1000.

Primary Key C could be index for r_2.

Primary Key E could be index for r_3.

For each tuple in r_1, we could use index C of r_2 to match attribute C in r_1.

and we could use index E of r_3 to match attribute E in r_2.

13.5 Consider the relations $r_1(A, B, C)$, $r_2(C, D, E)$, and $r_3(E, F)$ of Practice Exercise 13.4. Assume that there are no primary keys, except the entire schema. Let $V(C, r_1)$ be 900, $V(C, r_2)$ be 1100, $V(E, r_2)$ be 50, and $V(E, r_3)$ be 100. Assume that r_1 has 1000 tuples, r_2 has 1500 tuples, and r_3 has 750 tuples. Estimate the size of $r_1 \bowtie r_2 \bowtie r_3$ and give an efficient strategy for computing the join.

For each tuple of r_1. $\frac{1500}{V(C, r_2)} = \frac{15}{11}$ tuples of r_2. will join it

For each tuple of r_2. $\frac{1000}{V(C, r_1)} = \frac{10}{9}$ tuples of r_1 will join it.

$\frac{750}{V(E, r_3)} = \frac{15}{2}$ tuples of r_3 will join it

For each tuple of r_3. $\frac{1500}{V(E, r_2)} = 30$ tuples of r_2 will join it.

Since

$$r_1 \bowtie r_2 \bowtie r_3 = (r_1 \bowtie r_2) \bowtie r_3 \quad ①$$
$$= r_1 \bowtie (r_2 \bowtie r_3). \quad ②$$

For solution ① $\text{total}_1 = 1000 \times \frac{15}{11} \times \frac{15}{2} = 10228$

For Solution ② $\text{total}_2 = 750 \times 30 \times \frac{10}{9} = 25000$, $10228 < 25000$.

Therefore, join r_1 and r_2 first, then join r_3.

14.12 List the ACID properties. Explain the usefulness of each.

①Atomicity (原子性): Couldn't be divided.
Either all the operations are conducted properly or none are.
Usefulness: Keeping consistency.

②Consistency (一致性): Execution of a transaction is isolation preserves the consistency of database.
Usefulness: Programmers should write SQL correctly.

③Isolation (孤立性): Each transaction is unaware of another transaction(s) while other transactions are acting concurrently.
Usefulness: Ensure that transaction won't be affected by wrong transactions.

④Durability (可持续性): After a transaction completes, the changes of the database consist even if the operation is wrong or system failures.

14.15 Consider the following two transactions:

T_{13}: read(A);
read(B);
if $A = 0$ **then** $B := B + 1$;
write(B).

T_{14}: read(B);
read(A);
if $B = 0$ **then** $A := A + 1$;
write(A).

Let the consistency requirement be $A = 0 \vee B = 0$, with $A = B = 0$ the initial values.

a. Show that every serial execution involving these two transactions preserves the consistency of the database.

b. Show a concurrent execution of T_{13} and T_{14} that produces a nonserializable schedule.

c. Is there a concurrent execution of T_{13} and T_{14} that produces a serializable schedule?

Table.

A	B	0 V B	
0	0	0	✓
0	1	1	✓
1	0	0	✓
1	1	1	X

a. ① T_{13}, T_{14}

	A	B	0 V B
Initially	0	0	0
After T_{13}	0	1	1
After T_{14}	0	1	1

$A=0 \vee B=0 \equiv T \vee F = T$

① T_{14}, T_{13}

	A	B	0 V B
Initially	0	0	0
After T_{14}	1	0	0
After T_{13}	1	0	0

$A=0 \vee B=0 \equiv F \vee T = T$

b.

T_{13}	T_{14}
read(A) A=0	
	read(B) B=0
	read(A) A=0
read(B) B=0	
if A=0: B:=B+1	
	if B=0: A:=A+1
write(B) B=1	write(A) A=1

$A=0 \vee B=0 \equiv F \vee F = F$

c. There exists no concurrent solution. Here is the reason:

Table.

A	B	O V B	
0	0	0	✓
0	1	1	✓
1	0	0	✓
1	1	1	X

Accroding to the table, only when A != B or A = B = 0 can be satified.

There exists only two ways:

① Start T_{14} after T_{13} is finished.

② Start T_{13} after T_{14} is finished.

The other circumstances, accroading to. ACID. Isolation will cause A=0 & B=0 all the time. when T_{13} is reading or T_{14} is reading which will cause A=1 & B=1 in the end.

Assignment

- A scheduled flight has 50 tickets. Agency one wants to book 10 tickets and agency two want to book 20 tickets. Please design a lock strategy to implement the concurrency control.

Agency one	Agency two
Read(A)	
	Read(A)
A=A-10	
Write(A)	
	A=A-20
	Write(A)

Agency One.	Agency Two.
lock-S(A)	lock-S(A)
lock-X(A)	Read(A)
Read (A)	A = A-20
A = A-10	Write(A)
Write (A)	unlock-S(A).
unlock-X(A)	
unlock-S(A)	

16.1 Explain why log records for transactions on the undo-list must be processed in reverse order, whereas redo is performed in a forward direction.

undo-list: undo is utilized to rollback the information.
Backward direction could meet the requirement.

redo-list: redo is utilized to backup the information.
Forward direction could meet the requirement.

If there exist an update of character 'a' → 'b' → 'c'

	backward	forward
undo	'c' → 'b' → '<u>a</u>' ✓	'a' → '<u>b</u>' ✗ initial value is 'a'. <u>false</u>
redo	'c' → '<u>b</u>' ✗ incorrect	'a' → 'b' → '<u>c</u>' ✓

16.18 Consider the log in Figure 16.5. Suppose there is a crash just before the < T_0 abort> log record is written out. Explain what would happen during recovery.

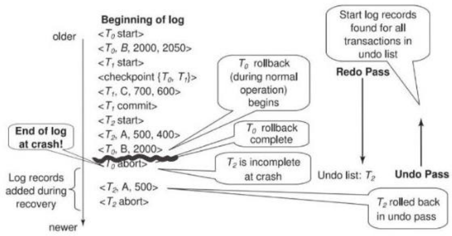

Figure 16.5 Example of logged actions, and actions during recovery.

① Redo Pass (备份)

a. Undo-List $T_0.T_1$
b. Start from <checkpoint {$T_0.T_1$}>
c. Redo T_1 C = 600
d. Undo-List T_0
e. Undo-List $T_0.T_2$
f. A = 400, B = 2000.

② Undo Pass (回滚).

a. Undo-List $T_0.T_2$
b. Rollback T_2
c. A = 500, output <T_2.A.500>
d. output <T_2.abort>
e. Roll back T_0.
f. B = 2000, output <T_0,B,2000>
g. output <T_0, abort>

Finally:

Result: A = 400
B = 2000
C = 600

Output.
<T_2.A.500>
<T_2.abort>
<T_0,B,2000>
<T_0, abort>