**Quiz 3 [LIVINGSTON]**

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ RUID: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. For each of the following schedules, draw the precedence graph and then state whether it is conflict-serializable. If yes, provide all equivalent serial schedules. If no, state why it is not conflict-serializable. (ri(X) denotes a read on object X for transaction Ti. wj(Y) denotes a write on object Y for transaction Tj.)
   1. r1(A), r2(B), r3(A), r3(B), r1(C), w3(A), r2(C), w1(A), w2(C), w1(C)
   2. r1(A), r3(A), r3(B), w3(B), r2(C), r2(B), r1(A), r3(C), w1(C), r3(A)

**Solution**

a) We start with an empty graph with three vertices labeled T1, T2, T3. We go through each operation in the schedule:

r1(A): A is subsequently written by T3, so add edge T1 -> T3;

r2(B): no subsequent operations to B, so no new edges;

r3(A): A is subsequently written by T1, so add edge T3 -> T1;

r3(B): no subsequent operations to B, so no new edges;

r1(C): C is subsequently written by T2, so add edge T1 -> T2;

w3(A): A is subsequently written by T1, so add edge T3 -> T1;

r2(C): C is subsequently written by T1, so add edge T2 -> T1;

w1(A): no subsequent operations to A, so no new edges;

w2(C): C is subsequently written by T1, so add edge T2 -> T1;

w1(C): no subsequent operations, so no new edges;

Therefore, we end up with the following precedency graph:

This graph has a cycle, so the original schedule is not conflict-serializable.

b) We start with an empty graph with three vertices labeled T1, T2, T3. We go through each operation in the schedule:

r1(A): no subsequent operations to A, so no new edges;

r3(A): no subsequent operations to A, so no new edges;

r3(B): no subsequent operations to B, so no new edges;

w3(B): B is subsequently read by T2, so add edge T3 -> T2;

r2(C): C is subsequently written by T1, so add edge T2 -> T1;

r2(B): no subsequent operations to B, so no new edges;

r1(A): no subsequent operations to A, so no new edges;

r3(C): C is subsequently written by T1, so add edge T3 -> T1;

w1(C): no subsequent operations to C, so no new edges;

r3(A): no subsequent operations, so no new edges;

Therefore, we end up with the following precedency graph:

This graph has no cycles, so the original schedule must be conflict-serializable. The equivalent serial schedule is T3-T2-T1.

1. Let T1, T2, T3 be the following transactions:

T1: r1(D), w1(B), w1(D)

T2: r2(D), w2(D), w2(B)

T3: r3(C), w3(B)

For each of the following schedules, state whether it is possible under 2PL protocol? Give your reasons.

* 1. r2(D), r3(C), w2(D), w3(B), r1(D), w2(B), w1(B), w1(D)
  2. r1(D), w1(B), r3(C), r2(D), w1(D), w2(D), w3(B), w2(B)

**Solution**

a) Let’s try to add the lock/unlock steps to the schedule:

l2(D), r2(D), l3(C), r3(C), w2(D), l3(B), u3(C), w3(B), u3(B), l2(B), u2(D), l1(D), r1(D), w2(B), u2(B), l1(B), w1(B), u1(B), w1(D), u1(D).

Pull out the lock/unlock operations of all the three transactions:

T1: l1(D), l1(B), u1(B), u1(D)

T2: l2(D), l2(B), u2(D), u2(B)

T3: l3(C), l3(B), u3(C), u3(B)

Which are consistent with the 2PL protocols.

b) Let’s try to add the lock/unlock steps to the schedule,

l1(D), r1(D), l1(B), w1(B), u1(B), l3(C), r3(C), …

The next operation is r2(D). T2 needs to lock D before this read, however, D is currently locked by T1. According to 2PL, T1 won’t unlock D until it has acquired all the locks on other resources. T1 won’t release the lock on D until it has finished all the operations on D (after w1(D)). In other words, in this case it is impossible for r2(D) to occur before w1(D). Thus, this schedule is not possible under 2PL protocol.

1. Let us begin with two bars: Cabana and Old Tavern. Cabana has local patrons A and remote patrons B, while Old Tavern has local patrons C and remote patrons D. Now a new bar New Tavern is opened in this area. Then Cabana and Old Tavern begin to lose clients. Suppose we have the followings two transactions:

T1: Cabana loses all patrons to Old Tavern. First remotes, then the rest

T2: Old Tavern loses all patrons to New Tavern. First remotes, then the rest

Given the following schedule S:

T2: Insert remote patrons of Old Tavern into New Tavern

T2: Delete remote patrons of Old Tavern

T1: Insert remote patrons of Cabana into Old Tavern

T1: Delete remote patrons of Cabana

T2: Insert local patrons of Old Tavern into New Tavern

T2: Delete local patrons of Old Tavern

T1: Insert local patrons of Cabana into Old Tavern

T1: Delete local patrons of Cabana

What patrons will each bar have after the execution of this schedule (in terms of A, B, C, D or empty)? Is the schedule serializable?

**Solution**

Let’s denote the local patrons and remote patrons of Cabana as C\_lp, C\_rp, respectively. Similarly, the local patrons and remote patrons of Old Tavern are OT\_lp, OT\_rp. For New Tavern, we have NT\_lp, NT\_rp for its local patrons and remote patrons. Then given the schedule S, we can get:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | C\_lp | C\_rp | OT\_lp | OT\_rp | NT\_lp | NT\_rp |
| Initial | A | B | C | D | empty | empty |
| T2 | A | B | C | D | empty | D |
| T2 | A | B | C | empty | empty | D |
| T1 | A | B | C | B | empty | D |
| T1 | A | empty | C | B | empty | D |
| T2 | A | empty | C | B | C | D |
| T2 | A | empty | empty | B | C | D |
| T1 | A | empty | A | B | C | D |
| T1 | empty | empty | A | B | C | D |

Then the results from the given schedule and two serial executions of T1, T2 are compared in the following table.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | C\_lp | C\_rp | OT\_lp | OT\_rp | NT\_lp | NT\_rp |
| Result | empty | empty | A | B | C | D |
| T1 -> T2 | empty | empty | empty | empty | AC | BD |
| T2 -> T1 | empty | empty | A | B | C | D |

The result is equivalent to the serial schedule T2 -> T1, so the given schedule is serializable.

1. Show all the conflict-serializable schedules of the following transactions T1 and T2. Explain why these are all such schedules.

T1: r1(Y), w1(Y), r1(A), w1(A), r1(Z), w1(Z)

T2: r2(Z), w2(Z), r2(B), w2(B), r2(Y), w2(Y)

**Solution**

We can see that the first operation of T1 is r1(Y) and the last operation of T2 is w2(Y). They conflict with each other since they contain the read and write operations on the same resource Y. Meanwhile, the first operation of T2 is r2(Z) and the last operation of T1 is w1(Z). They still conflict with each other since they contain the read and write operations on the same resource Z here. Therefore, there are only two possible conflict-serializable schedules for T1 and T2 here, which are T1, T2 (i.e. r1(Y), w1(Y), r1(A), w1(A), r1(Z), w1(Z), r2(Z), w2(Z), r2(B), w2(B), r2(Y), w2(Y)) and T2, T1 (i.e. r2(Z), w2(Z), r2(B), w2(B), r2(Y), w2(Y), r1(Y), w1(Y), r1(A), w1(A), r1(Z), w1(Z)).

1. Assume that 1% of the documents in the database contain the word "mongo". Our search results S on query "mongo" contain 0.5% of all database documents, but only 80% of S contain the word "mongo". What is precision and recall of our search results? The top document D in S has 10 occurrences of "mongo". What is TF-IDF score of "mongo" in D.

**Solution**

Precision is 80%, since only 80% of S contain the word "mongo".

Since 1% of the documents in the database contain the word "mongo" and our result only contain 0.5% \* 80% = 0.4%, recall is 0.4% / 1% = 40%.

TF(“mongo”, D) = 1 + log(10) = 2

IDF(“mongo”) = log(1 / 1%) = log100 = 2

TF-IDF(“mongo”) = 2 \* 2 = 4

1. Let T1 ... T100 be transactions

for i=1:100

Ti = LockAi, LockBi, UnlockAi, UnclockBi

T101 = LockA101, UnlockA101, LockB101, UnlockB101 (violates 2PL)

Can you build conflict non-serializable schedule from

a) Just T1 ... T100

b) T1 ... T101

In both cases either show such schedule or explain why you can't

**Solution**

Since all the transactions in this problem (T1, …, T100 and T101) have operations on different resources here (A1, …, A100 and A101; B1, …, B100 and B101), we cannot build any non-serializable schedule for both a) and b).

**Quiz 3 [BUSCH]**

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ RUID: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. For each of the following schedules, draw the precedence graph and then state whether it is conflict-serializable. If yes, provide all equivalent serial schedules. If no, state why it is not conflict-serializable. (ri(X) denotes a read on object X for transaction Ti. wj(Y) denotes a write on object Y for transaction Tj.)
   1. r1(A), r2(B), r3(A), r3(B), w2(B), w3(A), r2(C), w1(A), w2(C), w1(C)
   2. r1(A), r3(B), r3(A), r2(C), w3(B), r1(A), r2(B), r3(C), w1(C), r3(A)

**Solution**

a) We start with an empty graph with three vertices labeled T1, T2, T3. We go through each operation in the schedule:

r1(A): A is subsequently written by T3, so add edge T1 -> T3;

r2(B): no subsequent operations to B, so no new edges;

r3(A): A is subsequently written by T1, so add edge T3 -> T1;

r3(B): B is subsequently written by T2, so add edge T3 -> T2;

w2(B): no subsequent operations to B, so no new edges;

w3(A): A is subsequently written by T1, so add edge T3 -> T1;

r2(C): C is subsequently written by T1, so add edge T2 -> T1;

w1(A): no subsequent operations to A, so no new edges;

w2(C): C is subsequently written by T1, so add edge T2 -> T1;

w1(C): no subsequent operations, so no new edges;

Therefore, we end up with the following precedency graph:

This graph has a cycle, so the original schedule is not conflict-serializable.

b) We start with an empty graph with three vertices labeled T1, T2, T3. We go through each operation in the schedule:

r1(A): no subsequent operations to A, so no new edges;

r3(B): no subsequent operations to B, so no new edges;

r3(A): no subsequent operations to A, so no new edges;

r2(C): C is subsequently written by T1, so add edge T2 -> T1;

w3(B): B is subsequently read by T2, so add edge T3 -> T2;

r1(A): no subsequent operations to A, so no new edges;

r2(B): no subsequent operations to B, so no new edges;

r3(C): C is subsequently written by T1, so add edge T3 -> T1;

w1(C): no subsequent operations to C, so no new edges;

r3(A): no subsequent operations, so no new edges;

Therefore, we end up with the following precedency graph:

This graph has no cycles, so the original schedule must be conflict-serializable. The equivalent serial schedule is T3-T2-T1.

1. Let T1, T2, T3 be the following transactions:

T1: r1(D), w1(B), w1(D)

T2: r2(D), w2(D), w2(B)

T3: r3(C), w3(B)

For each of the following schedules, state whether it is possible under 2PL protocol? Give your reasons.

* 1. r1(D), r3(C), w1(B), w3(B), r2(D), w1(D), w2(D), w2(B)
  2. r3(C), r1(D), w1(B), w3(B), w1(D), r2(D), w2(D), w2(B)

**Solution**

a) Let’s try to add the lock/unlock steps to the schedule,

l1(D), r1(D), l3(C), r3(C), l1(B), w1(B), u1(B), l3(B), w3(B), u3(B), …

The next operation is r2(D). T2 needs to lock D before this read, however, D is currently locked by T1. According to 2PL, T1 won’t unlock D until it has acquired all the locks on other resources. T1 won’t release the lock on D until it has finished all the operations on D (after w1(D)). In other words, in this case it is impossible for r2(D) to occur before w1(D). Thus, this schedule is not possible under 2PL protocol.

b) Let’s try to add the lock/unlock steps to the schedule:

l3(C), r3(C), l1(D), r1(D), l1(B), w1(B), u1(B), l3(B), u3(C), w3(B), u3(B), w1(D), u1(D), l2(D), r2(D), w2(D), l2(B), u2(D), w2(B), u2(B).

Pull out the lock/unlock operations of all the three transactions:

T1: l1(D), l1(B), u1(B), u1(D)

T2: l2(D), l2(B), u2(D), u2(B)

T3: l3(C), l3(B), u3(C), u3(B)

Which are consistent with the 2PL protocols.

1. Let us begin with two bars: Cabana and Old Tavern. Cabana has local patrons A and remote patrons B, while Old Tavern has local patrons C and remote patrons D. Now a new bar New Tavern is opened in this area. Then Cabana and Old Tavern begin to lose clients. Suppose we have the followings two transactions:

T1: Old Tavern loses all patrons to New Tavern. First remotes, then the rest

T2: Cabana loses all patrons to Old Tavern. First remotes, then the rest

Given the following schedule S:

T2: Insert remote patrons of Cabana into Old Tavern

T2: Insert local patrons of Cabana into Old Tavern

T1: Insert remote patrons of Old Tavern into New Tavern

T1: Insert local patrons of Old Tavern into New Tavern

T1: Delete remote patrons of Old Tavern

T2: Delete remote patrons of Cabana

T1: Delete local patrons of Old Tavern

T2: Delete local patrons of Cabana

What patrons will each bar have after the execution of this schedule (in terms of A, B, C, D or empty)? Is the schedule serializable?

**Solution**

Let’s denote the local patrons and remote patrons of Cabana as C\_lp, C\_rp, respectively. Similarly, the local patrons and remote patrons of Old Tavern are OT\_lp, OT\_rp. For New Tavern, we have NT\_lp, NT\_rp for its local patrons and remote patrons. Then given the schedule S, we can get:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | C\_lp | C\_rp | OT\_lp | OT\_rp | NT\_lp | NT\_rp |
| Initial | A | B | C | D | empty | empty |
| T2 | A | B | C | BD | empty | empty |
| T2 | A | B | AC | BD | empty | empty |
| T1 | A | B | AC | BD | empty | BD |
| T1 | A | B | AC | BD | AC | BD |
| T1 | A | B | AC | empty | AC | BD |
| T2 | A | empty | AC | empty | AC | BD |
| T1 | A | empty | empty | empty | AC | BD |
| T2 | empty | empty | empty | empty | AC | BD |

Then the results from the given schedule and two serial executions of T1, T2 are compared in the following table.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | C\_lp | C\_rp | OT\_lp | OT\_rp | NT\_lp | NT\_rp |
| Result | empty | empty | empty | empty | AC | BD |
| T1 -> T2 | empty | empty | A | B | C | D |
| T2 -> T1 | empty | empty | empty | empty | AC | BD |

The result is equivalent to the serial schedule T2 -> T1, so the given schedule is serializable. (Notice that the T1 and T2 here are different from the T1 and T2 of the Livingston section. In this question, the operation sequence of T1 and T2 is [insert, insert, delete, delete], while the operation sequence of T1 and T2 of the Livingston section is [insert, delete, insert, delete]. They are different transactions from different questions.)

1. Show all the conflict-serializable schedules of the following transactions T1 and T2. Explain why these are all such schedules.

T1: r1(D), w1(D), r1(A), w1(A), r1(U), w1(U), r1(Z), w1(Z)

T2: r2(Z), w2(Z), r2(B), w2(B), r2(D), w2(D).

**Solution**

We can see that the first operation of T1 is r1(D) and the last operation of T2 is w2(D). They conflict with each other since they contain the read and write operations on the same resource D. Meanwhile, the first operation of T2 is r2(Z) and the last operation of T1 is w1(Z). They still conflict with each other since they contain the read and write operations on the same resource Z here. Therefore, there are only two possible conflict-serializable schedules for T1 and T2 here, which are T1, T2 (i.e. r1(D), w1(D), r1(A), w1(A), r1(U), w1(U), r1(Z), w1(Z), r2(Z), w2(Z), r2(B), w2(B), r2(D), w2(D)) and T2, T1 (i.e. r2(Z), w2(Z), r2(B), w2(B), r2(D), w2(D), r1(D), w1(D), r1(A), w1(A), r1(U), w1(U), r1(Z), w1(Z)).

1. Assume that 2% of the documents in the database contain the word "sql". Our search results S on query "sql" contain 1% of all database documents, but only 60% of S contain the word "sql". What is precision and recall of our search results? The top document D in S has 100 occurrences of "sql". What is TF-IDF score of "sql" in D?

**Solution**

Precision is 60%, since only 60% of S contain the word "sql".

Since 2% of the documents in the database contain the word "sql" and our result only contain 1% \* 60% = 0.6%, recall is 0.6% / 2% = 30%.

TF(“sql”, D) = 1 + log(100) = 3

IDF(“sql”) = log(1 / 2%) = log50

TF-IDF(“sql”) = 3 \* log50

1. Let T1 ... T100 be transactions

for i=1:100

Ti = LockA, LockB, UnlockA, UnclockB

T101 = LockA, UnlockA, LockB, UnlockB (violates 2PL)

Can you build conflict non-serializable schedule from

a) Just T1 ... T100

b) T1 ... T101

In both cases either show such schedule or explain why you can't

**Solution**

a) No. Since all the transactions here (T1, …, T100) follow 2PL, we cannot build any non-serializable schedule from them.

b) Yes. Since T101 violates 2PL, we can build non-serializable schedule from them (T1, …, T100, T101). An example schedule is T101(LockA), T101(UnlockA), T1, …, T100, T101(LockB), T101(UnlockB).