Concurrency Control

1

Transactions

- A **transaction** is a list of actions.
- The actions are reads (written $R_T(O)$) and writes (written $W_T(O)$) of database objects.

Example:

T_1: R(V), R(Y), W(V), W(C)

2

Schedules

- A **schedule** is a list of actions from a set of transactions and the order in which 2 actions of a transaction T appear in a schedule must be the same as the order in which they appear in T.

Example: T_1: R(V) W(V) T_2: R(Y) W(Y)

S_1: $R_{T1}(V)$ $R_{T2}(Y)$ $W_{T2}(Y)$ $W_{T1}(V)$ ← Yes

S_2: $W_{T1}(V)$ $R_{T2}(Y)$ $W_{T2}(Y)$ $R_{T1}(V)$ ← No

3

Complete Schedules

- For a schedule to be **complete**, each transaction must either commit or abort
- In this lecture we will assume that all transactions commit

Example:

T_1:R (V) W (V) C

T_2: R (Y) W (Y) C

4

Serializable Schedules

- A schedule is **serial** if the actions of the different transactions are not interleaved; they are executed one after another
- A schedule is **serializable** if its effect is the same as that of some serial schedule

- We usually only want to allow serializable schedules to be performed. Why?

5

WR Conflicts and Dirty Reads

- A WR conflict occurs when a transaction writes an object which is subsequently read by another transaction
- A **dirty read** occurs when a transaction reads an object that was written by a transaction that has not yet committed. Why is this a problem?

Example:

T_1:R(V) W(V) R(Y) W(Y) C

T_2: R(V) W(V) R(Y) W(Y) C

Which reads were dirty?

6

RW Conflicts and Unrepeatable Reads

- A RW conflict occurs when a transaction reads an object which is subsequently written by another transaction
- Suppose that T_1 reads an object A. Then, before T_1 commits, T_2 writes A. The read that T_1 did on A is **unrepeatable**. Why is this a problem?

Example:

T_1:R(V) W(V) R(Y) C

T_2: R(V) W(V) R(Y) W(Y) C

Which reads were unrepeatable?

7

WW Conflicts and Overwriting Uncommitted Data

- A WW conflict occurs when a transaction writes an object which is subsequently written by another transaction
- There can be a problem if T_1 overwrites the value of the object X which has already been changed by T_2, before T_2 commits. Why?

Example:

T_1:W(V) W(Y) C

T_2: W(V) W(Y) C

8

Conflict Serializable Schedules

- Two schedules are **conflict equivalent** if
  - they involve the same set of actions of the same transactions *and*
  - they order every pair of conflicting actions of two committed transactions in the same way.
- A schedule is **conflict serializable** if it is conflict equivalent to some serializable schedule.
- Conflict serializable schedules are also serializable.

9

Precedence Graph

- Given a schedule we can create a **precedence graph**
- The graph has a node for each transaction
- There is an edge from T_1 to T_2 if there is a conflict between T_1 and T_2 in which T_1 occurs first
- The schedule is conflict serializable if and only if there is no cycle in the precedence graph!!

10

Example

Which of the schedules are conflict serializable?

T_1: W(V) W(V) C
T_2: R(V) C

T_1: R(V) W(V) C
T_2: R(V) C

T_1: W(Y) C
T_2: R(V) R(Y) W(Z) C
T_3: W(V) C

11

Serializable vs. Conflict Serializable

Is the following schedule conflict serializable?

T_1: R(V) W(V) C
T_2: W(V) C
T_3: W(V) C

Note that it is serializable! The writes of T_2 and T_3 are called **blind writes**

12

View Serializable

- Two schedules S_1 and S_2 are **view equivalent** if
  - they involve the same set of actions of the same transactions *and*
  - if T_i reads the initial value of X in S_1 then it must also read the initial value of X in S_2 *and*
  - if T_i reads the value of X written by T_j in S_1 then it must also read the value of X written by T_j in S_2 *and*
  - For each data object X, the transaction (if any) that performs the final write on X in S_1 must also perform the final write on X in S_2
- A schedule is **view serializable** if it is view equivalent to some serializable schedule.

13

Example

Which of schedules are view serializable?

T_1: R(V) W(V) C
T_2: W(V) C
T_3: W(V) C

T_1: R(V) W(V) C
T_2: W(V) C
T_3: R(V) C

14

Serializable vs. View Serializable

Is the following schedule view serializable?

T_1: R(V) R(Y) C
T_2: W(V) W(Y) C

Note that it is serializable!

15

Locks

- We allow transactions to **lock** objects. Why?
- A **shared lock** is acquired on X before reading X. Many transactions can hold a shared lock on X.
- An **exclusive lock** is acquired on X before writing X. A transaction can hold a shared lock on X only if no other transaction holds any kind of lock on X.

16

Ensuring Serializable Schedules

- The following protocol ensures that only serializable schedules are allowed:

2 Phase Locking (2PL):

1. Each transaction must get an S-lock (shared lock) on an object before reading it
2. Each transaction must get an X-lock (exclusive lock) on an object before writing it
3. Once a transaction releases a lock it cant acquire any new locks

17

2PL implies Conflict Serialibility

- Every 2PL schedule is conflict serializable.
- Which of the following conform to the 2PL protocol?

T_1: X(Y) W(Y) U(Y) C
T_2: S(V) R(V) U(V) X(Y) W(Y) U(Y) C

T_1: X(Y) W(Y) U(Y) C
T_2: S(V) R(V) X(Y) U(V) W(Y) U(Y) C

18

Unrecoverable Schedules

Consider the following schedule, which follows 2PL:

T_1:X(V)S(Y)R(V) W(V)U(V) R(Y)U(Y)
T_2: X(V)R(V) W(V)U(V)

What happens if T_1 fails and is aborted?
What if T_2 commits and then T_1 fails?

19

Recoverable Schedules

- A schedule is recoverable if every transaction commits only after all the other transactions whose values it read have already committed.
- Strict 2PL: Same as 2PL, but a transaction releases its locks only after it has committed

==> Strict 2PL schedules are recoverable!

20

Phantom Reads

- A transaction re-executes a query and finds that another committed transaction has inserted additional rows that satisfy the condition
  - If the rows have been modified or deleted, it is called an unrepeatable read
- Example:
  - T_1 executes select * from Sailors where age < 25
  - T_2 executes insert into Sailors values(12,'Jim', 23, 7)
  - T_2 commits
  - T_1 executes select * from Sailors where age < 25

21

Levels of Isolation

- The SQL standard defines 4 levels of Isolation. Higher levels ensure greater "serializability", lower levels ensure greater concurrency

Level	Dirty Read	NonRepeatable Read	Phantom Read
Read Uncommited	Yes	Yes	Yes
Read Commited	No	Yes	Yes
Repeatable Read	No	No	Yes
Serializable	No	No	No

Yes means possible, No means not possible

22

Levels of Isolation in Oracle

- Oracle implements only 2 levels of Isolation
  - read committed (default)
  - serializable
- Oracle allows as much concurrency as it can,
  - readers don't wait for writers
  - writers don't wait for readers (i.e., Oracle assumes that if someone performs a select on a table, then he is only reading it and will not write it later on)

23

Example

- Suppose the database contains a table:
  - create table movie(seats number check(seats>=0));
  - movie has a single row with value 1
- Suppose we open two prompts on SQLPLUS, in two different windows. Now we do the following:
  - Prompt 1: select * from movie; (what happens?)
  - Prompt 2: update movie set seats = seats – 1; (what is the result?)
  - Prompt 2: commit;
  - Prompt 1: update movie set seats = seats– 1; (what is the result?)

24

Problem

- The movie seat seller of prompt 1 wanted to sell a seat, but by the time he tried to sell it, it disappeared. He actually needed a write lock on movie
- Solution: use select ... for update

25

Example Revisited

- Suppose we open two prompts on SQLPLUS, in two different windows. Now we do the following:
  - Prompt 1: select * from movie for update; (what happens?)
  - Prompt 2: update movie set seats = seats – 1; (what is the result?)
  - Prompt 1: update movie set seats = seats – 1; (what is the result?)
  - Prompt 1: commit; (what is the result?)

26