30/01/23

Unit - 5

Transaction Process System

Transaction is an Executing Program [process] form a logical large DB storage, majorly the transaction have 2 boundaries.

1) begin transaction
   * insert
   * delete
   * update.
2) end transaction

Any trans^n donot update the DB and only retrieves the data is called as "Read only transaction".

imp
* A DB is basically represented as a collection of named data items. the size of a data item is called its granularity.

The basic db access operations that transaction that include are.

1) Read_item (x)
   * finds disk block for x
   * buffer
   * program variable

2) write_item (x)
   * finds disk block for x
   * buffer
   * program variable

why recovery is required
(Or)
Different types of failure.

Because of failure need recovery.

1) s/m crash
2) A transaction / s/m error.
3) logical error
   local → low balance in account.

4) Concurrency control enforcement.
5) Disk failure
6) Physical problem and catastrophes.

Recovery from the failure.

5 operations

1) BEGIN_TRANSACTION
2) READ or WRITE
3) END_TRANSACTION
4) COMMIT —11— ——
5) ROLLBACK or ABORT.

transaction log / s/m log

5 types

1) [start_transaction, T]
2) [write_item, T, X, old_value, new_value]
3) [Read_item, T, X]
4) [Commit, T]
5) [abort, T] indicati T as fails.

Desirable properties of Transaction imp.

ACID properties.

↓

Automicity

1) A stands for automicity: A transaction is either performed in its entirety / not performed at all.
2) C stands for consistency preservation:
3) I stands for Isolation.
4) D stands for Durability / permanancy.

test.

Schedules of Transactions

order of Execution of operation from the various transaction forms.

A and B are Serial schedule
C and D are Interleaving operations.

$r_1(x)$ = read operation over X by Transaction 1.

a) $r_1(x)$; $w_1(x)$; $r_1(y)$; $w_1(y)$; $r_2(x)$; $w_2(x)$;
c) $r_1(x)$; $r_2(x)$; $w_1(x)$; $r_1(y)$; $w_2(x)$; $w_1(y)$;

* formal definition of a Schedule S of n transactions $T_1, T_2 \ldots T_n$
* Schedules can also be displayed in more compact notation.
* order of operation from left to right.

Characteristic Schedules based on Recoverability

classified two main class

* Recoverable Schedule :- one where no committed tra's
* Non-recoverable Schedule :- where a commited transaction

→ divide into
* cascadeless Schedule and Not cascadeless.
* Schedule requiring cascaded rollback.

→ divide into
* Strict Schedule
* Blind write

Characterizing Schedules based on Serializability

Schedule are guaranteed to give a correct result.

* Consistency preservation property of the ACID properties.
* Hence, each transaction is correct on its own.

* Serial Schedule
  * Serial → continuous without interaction
  * non serial.

* Serial schedule are not feasible for performance reason.
  * No interleaving of operations.
  * Long transactions force other transactions to wait.

* Serializable schedule.

Equivalance of Schedules

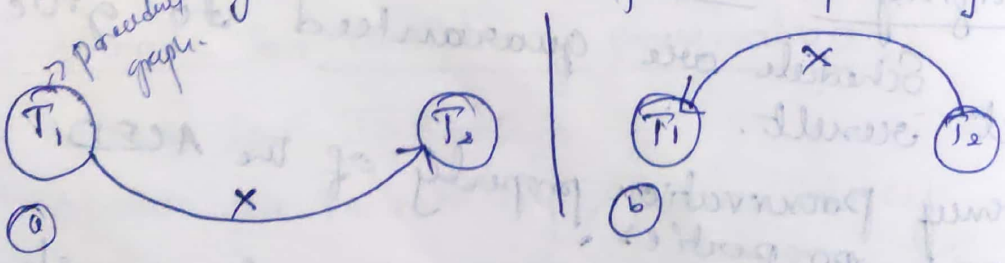
* Result equivalent : 2 schedules are called result equivalant.

* Conflict equivalent : 2 schedules are conflict equivalent.

Two operations are Conflicting
* Same data base
* 2 different transactions
* At least one is a write operation.

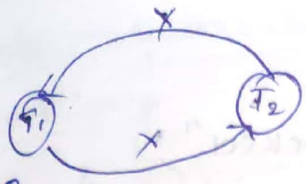
Testing for conflict serializability

Algorithm / precedence graph
* Looks at only r(x) and w(x) operations.
* Construct a precedence graph one node for each transaction.
* An edge is created from T_i to T_j.



SA: $r_1(x)\ w_1(x)\ r_1(y)\ w_1(y)\ r_2(x)\ w_2(x)$

SB: $r_2(x)\ w_2(x)\ r_1(x)\ w_1(x)\ r_1(y)\ w_1(y)$



(c)

sc: $r_1(x)\ r_2(x)\ w_1(x)\ r_1(y)\ w_2(x)\ w_1(y)$

2-phase locking technique for CC

LOCK : is a variable ite associate each one of its

lock & unlock operations for binary locks

lock_item(x):

B: if LOCK(x) = 0 item is unlocked
then LOCK(x) ← 1 lock the item
else begin
wait (until lock(x) = 0 and
the lock manager wakes up the transaction);
go to B
end;

unlock_item(x):

LOCK(x) ← 0; unlock the item
if any transactions are waiting
then wakeup one of the waiting transactions.

Shared / exclusive locks / multiple mode locks / read/write locks

3 types of operations.

1. read_lock(x)
2. write_lock(x)
3. unlock(x)

read_lock (x) / shared lock

```
B: if lock (x) = "unlocked"
   then begin lock (x) ← "read-locked";
      no_of_reads (x) ← 1
   end
   else if lock (x) = "read-locked"
    then no_of_reads (x) ← no_of_reads (x) + 1
     else begin wait (until lock (x) = "unlocked"
and the lock manager wakes up the transaction);
   go to B
   end;
```

write_lock (x) / exclusive lock

```
B: if lock (x) = "unlocked"
   then lock (x) ← "write-locked"
   else begin
      wait (until Lock (x) = "unlocked" and the
      lock manager wakes up the transaction);
      go to B
      end;
```

unlock (x)

```
if lock (x) = "write-locked"
 then begin LOCK (x) ← "unlocked;"
   wakeup one of the waiting transaction,
     if any
   end
  else if lock (x) = "read-locked"
   then begin
      no_of_reads (x) ← no_of_reads (x) - 1;
      if no_of_reads (x) = 0
      then begin lock (x) = "unlocked";
      wakeup one of the waiting transaction,
      if any
      end
    end;
```

Note :- guarantee Serializability by 2 phase locking

A transaction is said to follow 2-phase locking protocol if all locking operation (Read_lock, write_lock) precedes the 1st unlock operations in the transaction. Such a transaction is divided into 2 phase. an Expanding and growing phase during which new locks on items can be acquire but none can be released and a shrinking phase in which existing locks can be released but no new locks can be updated.

2-phase locking protocol

T_1	T_2
read_lock (Y);	read_lock(X);
read_item (Y);	read_item (X);
unlock (Y);	unlock (X);
write_lock(X);	write_lock (Y);
read_item (X);	read_item (Y);
X := X+Y;	Y := X+Y;
write_item (X);	write_item (Y);
unlock (X);	unlock (Y);

use of 2-phase locking Protocol

It can be prove that if every transaction in a schedule follows the 2-phase locking protocol. The schedule is guarantee to be serializable and there is no nessary protocol for serialisiability of schedules.

T_1'	T_2'
read_lock(Y);	read_lock (X);
read_item (Y);	read_item (X);
write_lock(X);	write_lock (Y);
unlock (Y);	unlock (X);
read_item (X);	read_item (Y);
X := X+Y;	Y := X+Y;
write_item (X);	write_item (Y);
unlock (X);	unlock (Y);

Deadlock Prevention

1) Every transaction lock all the items it need, in advance. and if any of the items cannot be obtained none of the items are lock.

2) wait_die : $TS(T_i) < TS(T_j)$ where TS stands for a time_stamp value assigned to a transaction based on the order in which the transaction are started. $TS(T_i)$ is smaller than $TS(T_j)$ then T_i is allowed to wait otherwise abort T_i and restart it later with the timestramp.

wait — contains

↓

unique identifier.

wound_wait.

3) If $TS(T_i)$ is smaller than $TS(T_j)$ then abort to T_j and restart it later with the same timestramp.

4) No waiting technique :- if a transaction is unable to obtain a lock, it is immediately abort it and then restart it after a some time delay without checking wheather a deadlock will actually occur (or) not.

5) Cautious waiting : if T_j is not block [not waiting for some locked item] then T_i is blocked and allowed to wait, otherwise abort T_i

$T_i \longrightarrow X \longleftarrow T_j$

Methods Deadlock Detection

1) Brute force
2) time out

Deadlock Detection

1) wait-for-graph

methods

1) Banker's
2) timeout.

Database Concurrency control

Timestamp based concurrency control algorithm

* Timestamp :

Basic timestamp ordering.

read_TS(x) = TS(T)

write_TS(x) = TS(T)

3 types

1) Basic timestamp ordering
2) Strict Timestamp ordering
3) Thomas' write Rule

Shadow Paging

ARIES algorithm.

ARIES stands for Algorithm for Recovery and Isolation Exploiting Semantics.

ARIES Algorithm

Algorithm for recovery & isolation
Exploiting Semantic, is
based on the write Ahead (WAL)
write a log record. Protocol.

Undo only log record : before Image is logged
Redo — " — : After Image logged
Undo-Redo : both before & after,

Every log assigned a unique & monotonically increasing log Sequence num (.

The recovery process consist of 3 phase

Analysis :
Redo —> Starting at the earliest LSN, the log is read forward & each update redone.
Undo :
The log is Scanned backward & updates corresponding to loser transaction undone.

ACID

Atomicity :- A transaction is an atomic unit of processing, it is either performed in its entirety & not part enforced by the recovery mthd

Consistency preserv :- A Correct Execution of the transaction must take the DB from & consist to another.

Isolation :- responsible for the Concurrency Control mthd.

Schedules of Transaction

when transaction are executing concurrently in an interleaved fashion, the order of execution of operation from the various transaction forms,

* possible shedules of 2 trans^{n} T_1 & T_2
* order of operation from top to bottom
* Each Schedule includes same operations.
* Different order of operation in each schedule.

* Schedule can also be displayed in more compact notation.
* order of operation from left to right.
* includes only read & write operation with
  - transaction Id & item name.
* can also include other operations begin, end, commit, abort.

Sha A :- $r_1(x)$; $w_1(x)$; $r_1(y)$; $w_1(y)$; $r_2(x)$; $w_2(x)$

Characterize Schedules based on Recoverability

Classified into 2 main classes.

Recoverable schedule: One where no committed transaction needs to be rolled back.

A Schedule S is recoverable, if no transaction T in S commits until all transaction T' that have written an

non-recoverable schedule:- A schedule where a committed transaction may have to be rolled back during recovery.

this violates Durability from ACID property.

Recoverable schedules

Cascadeless schedule

T_2 cannot read data X

Schedules requiring Cascade rollback.

uncommitted transaction T2.

Characteristic Specified by a set transaction. Syntax in SQL

* Access mode :- Read only / Read write. The default is Read write. unless the isolation level of read uncommitted, is specified.

* Diagnostic size :-

* Isolation level :- <isolation>, where <isolation> can be Read uncommitted, Read committed, ...

Potential problem with lower isolation levels.

* Dirty read :- transaction failed, reading a value that was written by it

* Nonrepeatable read :- Allowing another transaction to write a new value b/w multiple reads of a trans.

Marriage ~~hall~~ hall booking management. s/m

ER design. | strong, weak, r/s types, cardinal 1:1 1:M M:N etc. Quiz 3, 4 6c.

total part
multival
composite.

Queries.

| from.
each emp
group by, having.

Transaction

A transaction models the execution of a s/w procedure constituted by a set of instructio that may "read from" & write on db & that form a single logical unit.

Contains

1) one begin instruction
2) — " — end — " —
3) one among "commit" and rollback.

Set of properties

ACID

Atomicity: for each transaction execution. either All / none

Major operations
1) Read
2) write
3) commit.

Consistency: Before transaction started and After — " — computed, result. Should be same

Isolation: performs operations independently

Duration: - db values exist permanently