MASTER OF COMPUTER SCIENCE/
MASTER OF SCIENCE IN COMPUTER SCIENCE

Transaction Management – Part 1

Dr. Jeevani Goonetillake

UNIVERSITY OF COLOMBO SCHOOL OF COMPUTING

Introduction to Transaction Management

- Transaction is a logical unit of work that represents real-world events/user programs of any organization.
- A transaction consists of a sequence of database commands; disk reads and writes.

  example: Money Transfer

Concurrent execution of user programs is essential for good DBMS performance.

- Since disk accesses are frequent, and relatively slow, it is important to keep the cpu busy by working on several user programs concurrently.

Transaction Support

Transaction

Action, or series of actions, carried out by user or application, which accesses or changes contents of database.

Logical unit of work on the database.

Transforms database from one consistent state to another, although consistency may be violated during transaction.

Transaction Support

Can have one of two outcomes:

- **Success** - transaction *commits* and database reaches a new consistent state.
- **Failure** - transaction *aborts*, and database must be restored to consistent state before it started. Such a transaction is *rolled back* or *undone*.

Single User Vs Multiuser Systems

- **Single user** - at most one user at a time can use the system. Restricted to some PC DBMS.

- **Multi-user** -many users can use the system concurrently (at the same time). Most DBMSs are multi-user. E.g. Airline reservations systems, banks, Insurance agencies, stock exchanges are multi- user systems operated concurrently.

MultiProgramming

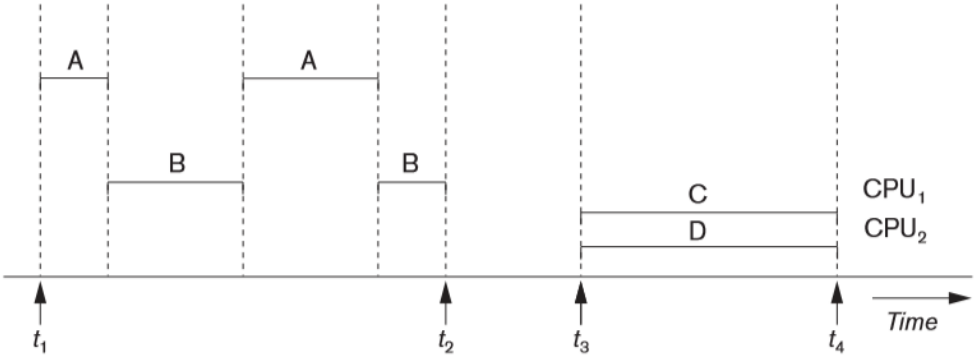
- Multiple users can use computer systems simultaneously because of the concept of multiprogramming.

- When only one CPU, the multiprogramming operating systems execute some commands from one program, then suspend that program and execute some commands from the next program and so on.

- A program is resumed at the point where it was suspended when it gets its turn to use the CPU again.

Interleaved Processing Vs Parallel Processing

- Hence, concurrent execution of the program is actually interleaved. Simultaneous processing of multiple programs are done with multiple CPUs.



Interleaved vs. Parallel processing.

Transaction Operations

- A transaction includes one or more database access operations
  - insertion, deletion, modification, or retrieval

- **Read-only transaction** - the database only retrieves data.
- **Read-write transaction**.

Transaction Operations

Operations	Descriptions
Retrieve	To retrieve data stored in a database.
Insert	To store new data in database.
Delete	To delete existing data from database.
Update	To modify existing data in database.
Commit	To save the work done permanently.
Rollback	To undo the work done.

Transaction Processing Systems

- Transaction processing systems execute database transactions with large databases and hundreds of concurrent users,

- Examples:
  - Airline reservations
  - Banking
  - Credit card processing
  - Online retail purchasing
  - Stock markets
  - Supermarket checkouts

Example Transaction

- E.g. Transaction Tl :No of reservations for airline A is X; No of reservation for airline B is *Y;* N reservation from A is cancelled and booked for B.
- Transaction T2 : M reservations to airline A.

T1	T2
read_item(X)	read_item(X)
X=X-N	X=X+M
write_item(X)	write_item(X)
read_item(Y)	
Y=Y+N	
write_item(Y)	

Database Access Operations

- Read(X)

- Reads a database item named X into a program variable.

1. Find the address of the disk block that contains item X.
2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
3. Copy item X from the buffer to the program variable named X.

Database Access Operations

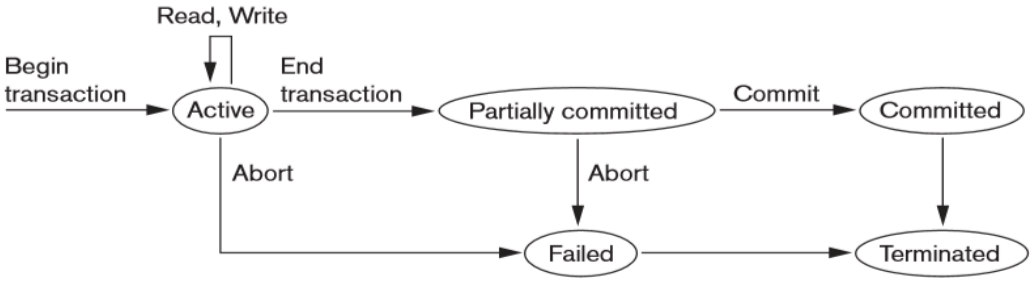
- Write(X)

  - Writes the value of program variable X into the database item named X.

1. Find the address of the disk block that contains item X.
2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
3. Copy item X from the program variable named X into its correct location in the buffer.
4. Store the updated block from the buffer back to disk (either immediately or at some later point in time).

Transaction States

- A transaction is an atomic unit of work that should either be completed in its entirety or not done at all.
- For recovery purposes, the system needs to keep track of its status.
- State Transition diagram shows how a transaction moves through its execution states.



State	Description
Active state	A transaction goes into an active state immediately after it starts execution, where it can issue READ and WRITE operations. A transaction may fail and be aborted when the transaction itself detects an error during execution which it cannot recover from. Example, a transaction trying to debit loan amount of an employee from his insufficient gross salary. A transaction may also be aborted before it is committed due to system failure or any other circumstances beyond its control.
Partially committed	A partially committed state appears to occur when all components of a database transaction have finished, and the DB has logically committed in persisting those changes to the database but has not yet actually persisted them. The word "logically" is used here because it is possible that after the work of a transaction has finished a failure could still occur. To take this possibility into account, the DBMS writes out enough information to disk to guarantee that, even if a failure were to occur, the result from the transaction could be recreated and the database could be updated appropriately.
Aborted	When the normal execution can no longer be performed. Failed or aborted transactions may be restarted later, either automatically or after being resubmitted by the user as new transactions.
Committed	After successful completion of transaction. A transaction is said to be in a committed state if it has partially committed and it can be ensured that it will never be aborted.

Transaction States

- **BEGIN**-This marks the beginning of transaction execution.
- **READ or WRITE**- These specify read or write operations on the database items that are executed as part of a transaction.
- **END -** This specifies that READ and WRITE transaction operations have ended and marks the end of transaction execution. (check whether the changes introduced by the transaction can be permanently applied to the database (committed) or whether the transaction has to be aborted).

Transaction States

- **COMMIT -** This signals a successful end of the transaction so that any changes (updates) executed by the transaction can be safely committed to the database and will not be undone.

- **ROLLBACK (or ABORT) -** This signals that the transaction has ended unsuccessfully, so that any changes or effects that the transaction may have applied to the database must be undone.

Properties of Transactions (ACID)

- **Atomicity** - A transaction is an atomic unit of processing; it should either be performed in its entirety or not performed at all.
- **Consistency** - A transaction should be consistency preserving, if it is completely executed from beginning to end without interference from other transactions, it should take the database from one consistent state to another.

Properties of Transactions

- **Isolation** - A transaction should appear as though it is being executed in isolation from other transactions, even though many transactions are executing concurrently. It should not be interfered with any other transactions executing concurrently.
- **Durability or permanency** - The changes applied to the database by a committed transaction must persist in the database. These changes must not be lost because of any failure.

Example

- e.g. Transfer 50 from account A to B

A=1000; B=2000;

```
T1:   Begin
      Read(a);
      A = A – 50;
      Write(a);
      Read(b);
      B = B + 50;
      Write(b);
      End;
```

Example

- Atomicity
  - Either performed in its entirety or not performed at all.
  - Transaction failure after WRITE(A), but before WRITE(B), then A=950; B=2000; i.e. 50 is lost Data is now inconsistent as A+B is now 2950.
- Consistency
  - Take the database from one consistent state to another.
  - Value of A+B (3000) should be same before transaction and after transaction

Example

- Isolation
  - updates not visible to other transactions until committed. Between WRITE(A) and WRITE(B) if second transaction reads A and B it sees inconsistent data as A+B = 2950
- Durability
  - changes must never be lost because of subsequent failures (e.g. power failure)
  - Recover database: remove changes of a partially done transaction (A=1000; B=2000);
  - reconstruct completed transactions (A=950; B=2050)

Concurrent Execution of Transactions

- The types of problems that may encounter with these two simple transactions if they run concurrently.
  - The Lost Update Problem
  - The Temporary Update (or Dirty Read) Problem
  - The Incorrect Summary Problem
  - The Unrepeatable Read Problem

The Lost Update Problem

- A lost update occurs when two different transactions are trying to update the same column on the same row within a database at the same time.

T1	T2
READ(X) X = X - N	
	READ(X) X = X + M
WRITE(X) READ(Y)	
	WRITE(X)
Y = Y + N WRITE(Y)	

X=80 ; Y=100 ; N=5 ; M=4;

T2 reads the value of X before T1 changes it in the database, and hence the updated value resulting from T1 is lost .

The Temporary Update (or Dirty Read) Problem

- This problem occurs when one transaction updates a database item and then the transaction fails for some reason. Meanwhile, the updated item is accessed (read) by another transaction before it is changed back to its original value.

X=80 ; Y=100 ; N=5 ; M=4;

T1	T2
READ(X) X = X - N WRITE(X)	
	READ(X) X = X + M WRITE(X)
READ(Y) ROLLBACK	

The value of item X that is read by T2 is called dirty data because it has been created by a transaction that has not completed and committed.

The Temporary Update (or Dirty Read) Problem

Transaction 1	Transaction 2
/* Query 1 */ **SELECT age FROM users WHERE id = 1;** /* will read 20 */	
	/* Query 2 */ **UPDATE users SET age = 21 WHERE id = 1;** /* No commit here */
/* Query 1 */ **SELECT age FROM users WHERE id = 1;** /* will read 21 */	
	ROLLBACK;

The Incorrect Summary Problem

- One transaction is calculating an aggregate summary function on a number of database items while other transactions are updating some of these items.
- The aggregate function may calculate some values before they are updated and others after they are updated.

The Incorrect Summary Problem

T1	T3
	Sum=0 READ(A) Sum+=A . . .
READ(X) X = X - N WRITE(X)	
	READ(X) Sum+=X READ(Y) Sum+=Y
READ(Y) Y = Y+ N WRITE(Y)	

X=80 ; Y=100 ; N=5 ; M=4;

T3 reads X after N is subtracted and reads Y before N is added; a wrong summary is the result.

The Unrepeatable Read Problem

- A transaction T reads the same item twice and the item is changed by another transaction T between the two reads. Hence, T receives different values for its two reads of the same item.

X=80

T1	T2
READ(X)	
	READ(X) X=X-5 WRITE(X)
READ(X)	

The Unrepeatable Read Problem

Transaction 1	Transaction 2
/* Query 1 */ **SELECT * FROM users WHERE id = 1;**	
	/* Query 2 */ **UPDATE users SET age = 21 WHERE id = 1;** **COMMIT;**
/* Query 1 */ **SELECT * FROM users WHERE id = 1;** **COMMIT;**	

Phantom Reads

- A *phantom* is a row that matches the search criteria but is not initially seen.
- The phantom reads anomaly is a special case of Non-repeatable reads.

Ex:

- Suppose transaction 1 reads a set of rows that satisfy some search criteria.
- Transaction 2 generates a new row (through either an update or an insert) that matches the search criteria for transaction 1.
- If transaction 1 re-executes the statement that reads the rows, it gets a different set of rows.

Phantom Reads

Transaction 1	Transaction 2
/* Query 1 */ **SELECT * FROM users WHERE age BETWEEN 10 AND 30;**	
	/* Query 2 */ **INSERT INTO users(id,name,age) VALUES (3, 'Bob', 27);** **COMMIT;**
/* Query 1 */ **SELECT * FROM users WHERE age BETWEEN 10 AND 30;** **COMMIT;**	

Transaction Isolation Levels

- These isolation levels describe to what extent the transactions are isolated from other transactions, and whether they can see the data inserted / updated by other transactions.
- Transaction isolation levels are defined by the presence or absence of the following phenomena
  - **Dirty Reads**
  - **Unrepeatable Reads**
  - **Phantom Reads**

Transaction Support in SQL

- A single SQL statement is always considered to be atomic; either it completes execution without an error or it fails and leaves the database unchanged.
- No explicit Begin_Transaction statement.
  - Transaction initiation is done implicitly when particular SQL statements are encountered.
- However, every transaction must have an explicit end statement.
  - Either a COMMIT or a ROLLBACK

Transaction Support in SQL

The following commands are used to control transactions.

- **COMMIT** – to save the changes.
- **ROLLBACK** – to roll back the changes.
- **SAVEPOINT** – creates points within the groups of transactions in which to ROLLBACK.
- **SET TRANSACTION** – Places a name on a transaction.
- **ROLLBACK TO SAVEPOINT**

Transaction Support in SQL

- Every transaction has certain characteristics **attributed** to it.
- Characteristics of a transaction are specified by a **SET TRANSACTION** statement in SQL.
- Example : SET TRANSACTION READ ONLY NAME ‘Ro_example';

  Only for data retrieval.

  SET TRANSACTION READ WRITE NAME 'RW_example';

Time	Session	Explanation
t0	`COMMIT;`	This statement ends any existing transaction in the session.
t1	`SET TRANSACTION NAME 'sal_update';`	This statement begins a transaction and names it `sal_update`.

t2	`UPDATE employees`
	`SET salary = 7000`
	`WHERE last_name = 'Banda';`
		This statement updates the salary for Banda to 7000.

t3	`SAVEPOINT after_banda_sal;`	This statement creates a savepoint named `after_banda_sal`, enabling changes in this transaction to be rolled back to this point.

t4	`UPDATE employees`
	`SET salary = 12000`
	`WHERE last_name = 'Greene';`
		This statement updates the salary for Greene to 12000.

t5	`SAVEPOINT after_greene_sal;`	This statement creates a savepoint named `after_greene_sal`, enabling changes in this transaction to be rolled back to this point.
t6	`ROLLBACK TO SAVEPOINT` `after_banda_sal;`	This statement rolls back the transaction to t3, undoing the update to Greene's salary at t4. The `sal_update` transaction has *not* ended.

Time	Session	Explanation

t7	`UPDATE employees`
	`SET salary = 11000`
	`WHERE last_name = 'Greene';`
		This statement updates the salary for Greene to 11000 in transaction `sal_update`.

t8	`ROLLBACK;`	This statement rolls back all changes in transaction `sal_update`, ending the transaction.
t9	`SET TRANSACTION NAME 'sal_update2';`	This statement begins a new transaction in the session and names it `sal_update2`.

t10	`UPDATE employees`
	`SET salary = 7050`
	`WHERE last_name = 'Banda';`
		This statement updates the salary for Banda to 7050.

t11	`UPDATE employees`
	`SET salary = 10950`
	`WHERE last_name = 'Greene';`
		This statement updates the salary for Greene to 10950.

t12	`COMMIT;`	This statement commits all changes made in transaction `sal_update2`, ending the transaction. The commit guarantees that the changes are saved in the online redo log files.

Isolation Level

ISOLATION LEVEL <isolation>

- READ UNCOMMITTED(most forgiving)
- READ COMMITTED (default for some)
- REPEATABLE READ
- SERIALIZABLE(Strict, highest isolation level, default)
  - No dirty read, unrepeatable reads, phantoms

Read Uncommited

- Database server process reads from the database table without checking for locks (let this process look at dirty data).

- This can be useful when 100% accuracy is not as important as speed and freedom from contention; and cannot wait for locks to be released.

SQL Syntax

- **SET TRANSACTION ISOLATION LEVEL READ UNCOMMITED;**

Question 1

Consider a table R(A) containing {(1), (2)}, and two transactions T1 and T2.

T1: update R set A = 2*A

T2: select avg (A) from R

If transaction T2 executes using *Read Uncommitted*, what are the possible values it returns?

a) 1.5

b) 1.5, 3

c) 1.5, 2, 3

d) 1.5, 2.5, 3

e) 1.5, 2, 2.5, 3

Question 1 - Explanation

- The update command in T1 can update the values in either order, and the select in T2 can compute the avg at any point before, between, or after the updates.

T2 1.5
T1 {(2),(4)} T2 3
T1 {(2),(2)} T2 2
T1 {(1),(4)} T2 2.5

e) 1.5, 2, 2.5, 3

Read Committed

- In this isolation level, read locks are acquired on selected data but they are released immediately whereas write locks are released at the end of the transaction.
- Data records retrieved by a query are not prevented from modification by some other transaction.
- This can be useful for lookups; queries; reports yielding general information (e.g. month-ending sales analyses).

SQL Statement

- **SET TRANSACTION ISOLATION LEVEL READ COMMITED;**

Question 2

Consider tables R(A) and S(B), both containing {(1), (2)}, and two concurrent transactions T1 and T2:

T1: update R set A = 2*A; update S set B = 2*B commit

T2: select avg(A) from R; select avg(B) from S

If transaction T2 executes using Read Committed, is it possible for T2 to return two different values?

a) Yes

b) No

Question 2 -Explanation

- T2 could return avg(A) computed before T1, and avg(B) computed after T1.

T2 1.5 1.5
T1 T2 3 3
T1 {(1),(2)} {(2),(4)} T2 1.5 3