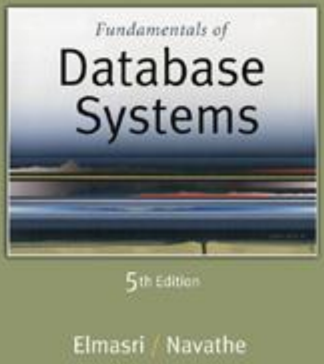
Lecture 10

Transaction Processing Concepts



Transaction processing systems

- They are systems with large databases and hundreds of concurrent users are executing database transactions. Examples of such systems include systems for reservations, banking, credit card processing, stock market, and other similar systems.
- They require high availability and fast response time for hundreds of concurrent users.

Transactions Read and Write operations

- A **transaction** is a logical unit of database processing that includes one or more database access operations. These can include insertion, deletion, modification, or retrieval operations.
- The database operations that form a transaction can either be embedded within an application program or they can be specified interactively via a high-level query language such as SQL.
- In an application program, we use two statements to illustrate the begin and end of transaction:

  BEGIN TRANSACTION

  END TRANSACTION

Transactions Read and Write operations

- If the database operations in a transaction do not update the database, but only retrieve data, the transaction is called a **read-only transaction.**
- A database is basically represented as a collection of named **data items**. **Data item** may be a field of some record, or a complete record or even a whole disk block.
- Whenever a transaction is submitted to a DBMS for execution, the system is responsible for making sure that either:

1. All the operations in the transaction are completed successfully and their effect is recorded permanently in the database, or
2. The transaction has no effect whatsoever on the database or on any other transactions.

Desirable Properties of Transactions (ACID)

1. **Atomicity**: A transaction is an atomic unit of processing; it is either performed its entirety or not performed at all.
2. **Consistency preservation**: A transaction is consistency preserving if its complete execution take(s) the database from one state to another.
3. **Isolation:** A transaction should appear as though it is being executed in isolation from other transactions. That is, the execution of a transaction should not be interfered with by any other transactions executing concurrently.
4. **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.

Data Protection

Data Protection is used to protect the database against a variety of possible threads.

There are four controls for database protection.

1. Recovery
2. Concurrency
3. Security
4. Integrity

Recovery

- Recovery in a database system means, primarily, recovering the database itself-that is, restoring the database to a state that is known to be correct after some failure.
- There are three types of recovery:

1. Transaction recovery
2. System recovery
3. Media recovery

Transaction recovery

- ***A transaction*** is a logical unit of work; it’s a sequence of several operations. It is also the unit of recovery.
- For example, if we have a transaction that adds a new shipment. This transaction consists of two updates, one to insert new values in SP table and the other to update the total quantity in P table. If a system crashes happen between the two updates. Those updates will be undone.

A pseudocode of such transaction will be as follow:

```
  BEGIN TRANSACTION;
    INSERT  ( {S#:'S5', P#:'P1',QTY:100})  INTO SP;
    IF any error occurred  THEN  GO TO UNDO;
    UPDATE P WHERE P#='P1' TOTQTY:=TOTQTY+1000;
    IF any error occurred  THEN  GO TO UNDO;
    COMMIT TRANSACTION;
    GO TO FINISH;
UNDO:    ROLLBACK TRANSACTION;
FINISH:  RETURN;
```

Transaction recovery

- The **COMMIT** operations signals ***successful*** end-of-transaction. It tells the transaction manager that a logical unit of work has been successfully completed and the database is in a consistent state.
- The **ROLLBACK** operation signals ***unsuccessful*** end-of-transaction. It tells the transaction manager that something has gone wrong and the database is in an inconsistent state, and the operations so far must be “rolled back” or undone.

Transaction recovery

- The system maintains a ***log*** with details of all update operations. The values of the updated objects before and after the update must be saved in the corresponding log entry to restore the updated object to its previous value.
- The COMMIT and ROLLBACK terminate the transaction, not the program.

System recovery

- **System failures** (such as Power failure), which affect all transactions currently in progress but do not physically damage the database. System failure is also known as ***soft crash***.
- When system failures occur, the contents of the main memory (the database buffers) are lost. So, the transactions that are completed before the system failures (COMMIT ones) must be redone after the restart time. While those transactions that are not completed before the system failures (ROLLBACK ones) must be undone after the restart time.

Checkpoint

- The system automatically takes a **checkpoint.**
- At checkpoint,

  1) Physically writing the content of the database buffers out to the physical database on the storage.

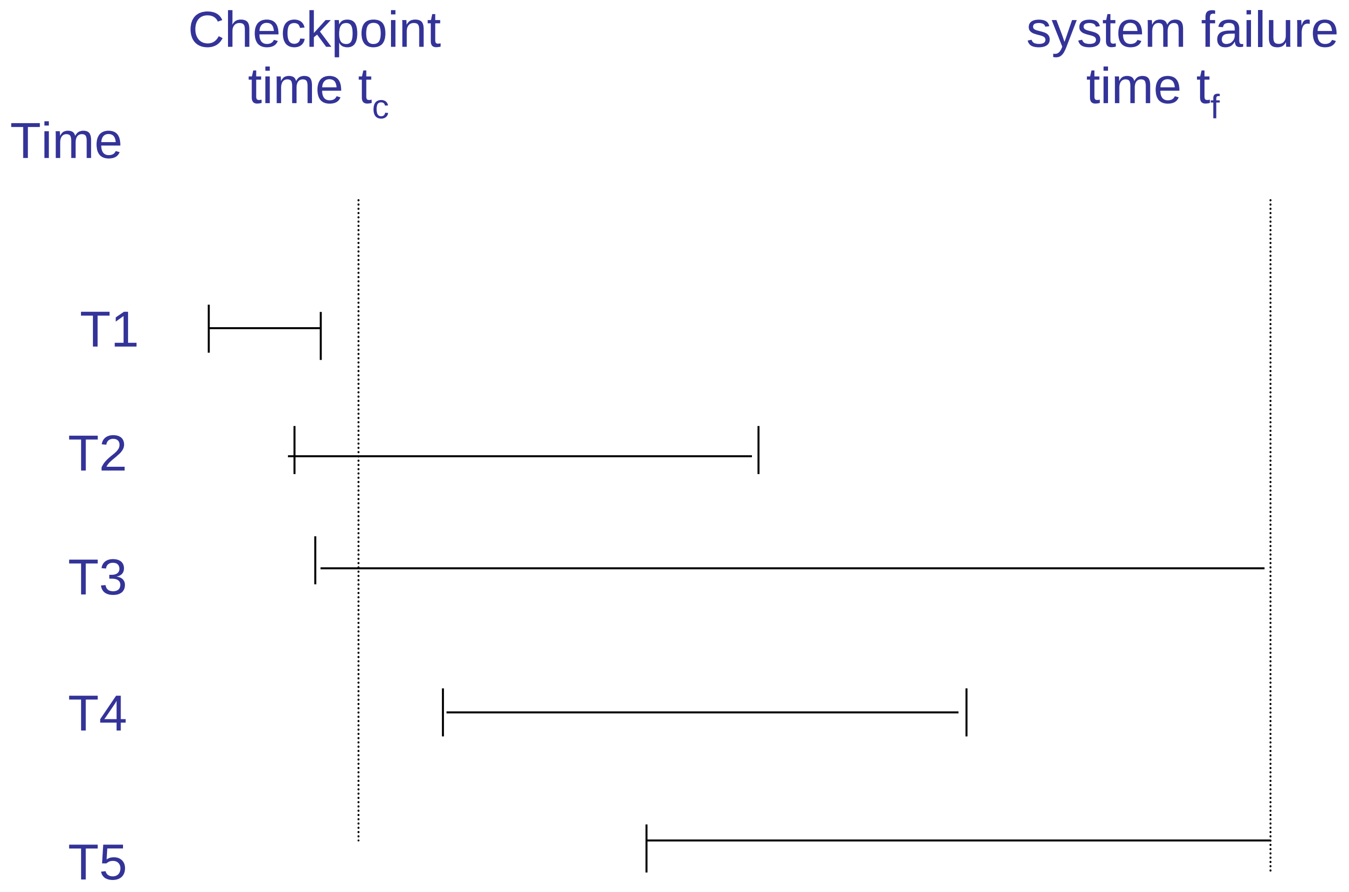
  2) Physically writing a special checkpoint record out to the physical log. The checkpoint record gives a list of all One) transactions that were in progress at the time this checkpoint was taken.

For example: Five transactions categories

- Consider a system has the following operation in sequence:

Begin(T1)- End(T1)- Begin(T2)- Begin(T3)- Checkpoint- Begin(T4)- End(T2)- End(T4)- Failure

For example: Five transactions categories



UNDO T3, T5   REDO T2, T4

For example: Five transactions categories

- A system failure has occurred at time t_f .
- A most recent checkpoint prior to t_f was taken at time t_c.
- Transactions of type T1 were completed prior to time t_c.
- Transactions of type T2 started prior to time t_c and completed after time t_c and before time t_f.
- Transactions of type T3 started prior to time t_c but did not complete by time t_f.
- Transactions of type T4 started after time t_c and completed before time t_f.
- Transactions of type T5 started after time t_c , but did not complete by time t_f.

At restart time

1. Start with two lists of transactions, the **UNDO** list and the **REDO** list.

Set the UNDO = list of all transactions given in the checkpoint record.

Set the REDO = empty.

2. Search forward through the log, starting from the checkpoint record.

3. If a "***begin transactions***", log entry is fount for transaction T, Add T to the UNDO list.

4. If a "***COMMIT***" log entry is found for transaction T, Move T from the UNDO list to the REDO list.

5. When the end of the log is reached, the **UNDO** list identifies transactions of types T3 and T5. While the **REDO** list identifies transactions of types T2, T4.

Media Failure

- **Media Failures** (such as head crash or a disk controller failure on the disk), which do cause damage to the database, or to some portion of it, and affect at least those transactions currently using that portion. Media failures are also known as ***hard crash***.
- Recovery from such a failure basically involves reloading the database from a backup copy, and then using the log to redo all transactions that completed since that backup copy was taken.

Concurrency

- DBMSs allow many transactions to access the same data at the same time. So there exists a concurrency control mechanism to ensure that concurrent transactions don't interfere with each other's operation.

Locking

Locking is a concurrency control technique.

The way locking works as follows:

1. There are two types of locking:

   exclusive locks (X locks) **shared locks (S locks)**

2. If transaction **A** holds an **X lock** on record **R**, then a request from transaction **B** for a lock of either type on **R** will cause **B** to go into a wait state. **B** will wait until **A**'s lock is released.

3. If transaction **A** holds an **S lock** on record **R**, then:

   - A request from transaction **B** for an **X lock** on **R** will cause **B** to go into a wait state.

   - A request from transaction **B** for an **S lock** on **R** will be granted.

The following table illustrates how locks are working.

	X_A	S_A	-
X_B	*N*	*N*	*Y*
S_B	*N*	*Y*	*Y*
-	*Y*	*Y*	*Y*

As	-	for no lock
	N	for conflict locks (i.e. not allowed)
	Y	for non conflict locks (i.e. allowed)

- When a transaction successfully **retrieves** a record, it automatically acquires an **S lock** on that record. When a transaction successfully **updates** a record, it automatically acquires an **X lock** on that record.
- X locks and S locks are held until the next COMMIT or ROOLBACK.

The Uncommitted Dependency Problem

Transaction A	Time	Transaction B
--		--
--	t_1	Update R (X lock on R)
Retrieve R (S lock on R)	t_2	--
wait	t_3	ROLLBACK (release lock)
Resume retrieve on R	t_4	

Transaction A is prevented from seeing or updating an uncommitted change at time t_2.

The Lost Update Problem

Transaction A	**Time**	**Transaction B**
--		--
Retrieve R (S lock on R)	t_1	--
--	t_2	Retrieve R (S lock on R)
Update R (X lock on R)	t_3	--
wait	t_4	Update R (X lock on R)
wait	t_5	wait **DEADLOCK**

A serious problem happens that is deadlock at time t_5.

How to overcome Deadlock?

- **Deadlock** is a situation in which two or more transactions are in a simultaneous wait state, each waiting for one of the others to release a lock to proceed.
- The system must detect deadlock and break it. One of the solutions is to choose a victim and cancel its transaction. Another solution is to cancel all the transactions that cause deadlock.

The following represents the sequence of events in the schedule involving transaction T_1, T_2 and T_3. Items used in the database are A, B, C, D and E. Assume S lock for retrieve operations while X lock for update operations. All locks are held until the end of the transaction. Are there any deadlocks at time t_n? If yes, state them. Suggest a way to overcome deadlock? After applying you suggestion, what will be the case?

Time			
	t_0		….
	t_1	T_1	retrieve A
	-	T_3	retrieve B
	-	T_2	retrieve C
	-	T_2	update C
	-	T_3	retrieve D
	-	T_2	retrieve D
	-	T_1	update E
	-	T_3	retrieve E
	-	T_1	retrieve C
	-	T_2	update D
	t_n	…..	

T_1	T_2	T_3
S_A		
		S_B
	S_C	
	X_C	
		S_D
	S_D	
X_E		
		S_E WAIT
S_C WAIT		
	X_D WAIT	

At t_n exist deadlock as:

$$\begin{tikzcd}
T_3 \arrow[r, "E"] & T_1 \arrow[d, "C"] \\
 & T_2 \arrow[ul, "D"]
\end{tikzcd}$$

To overcome deadlock, cancel one of the transaction T_1 , T_2 or T_3

Cancel T_2, i.e. T_2 is rollback, and cancel all its locks on C and D

T_1 can continue until end of its transaction, then T_3 continue .

T1	T2	T3
		SE
		WAIT
SC		
WAIT		
	SD	
	WAIT	
	ROLLBACK	
RESUME		
…		
COMMIT		
		RESUME

Security

- **Security** involves ensuring that users are allowed to do things they are trying to do.
- **Data object** is the unit of data for security purpose. It ranges from an entire collection of tables to a specific data value at a specific row and column.
- A given user will have different access rights or authorities on different objects. Different users may have different rights on the same object.

GRANT command

It is used to perform certain authorities on certain data objects for certain users. It format is:

```
GRANT  authority_type
ON  data_object
TO user(s)
[WITH GRANT option];
```

authority_type   may be SELECT, UPDATE, DELETE and INSERT.

User(s) is the user ID or ALL that stands for all users that are familiar to the system.

Option may be CASCADE or RESTRICTED.

Some of SQL command for GRAND uses the following option:

```
ON ATTEMPTED VIOLATION REJECT;
```

For example

```
CREATE SECURITY RULE SR3
GRANT      SELECT (S#, SNAME, CITY), DELETE
 ON        S WHERE S.CITY =’LONDON’
 TO        HODA, AHMED
 ON ATTEMPTED VIOLATION REJECT
```

REVOKE command

It is used to prevent user(s) to access a certain object. The general format is:

```
REVOKE        authority_type
ON            data_object
FROM           user(s)    [option];
```

Example:

```
REVOKE  SELECT
  ON    EX1
  FROM  HODA    RESTRICTED;
```