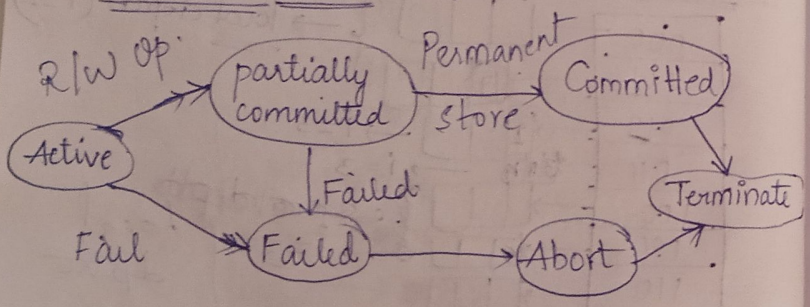
* Transaction Management:

→ Transactions are set of operations, used to perform a set of tasks.

↔ Access Operations:

* Read (Access Data): Read data from disk to memory variable.
* Write (Change Data): Write data item from memory variable to disk.
* Commit: Operation used to save the changes done.
* RollBack: If any of the step of the transaction is failed; we must go back and start from beginning.

→ Transaction States:



* Abort: If a transaction aborts; changes made are not visible

* ACID properties:

→ Atomicity: A transaction must occur completely or it should be roll backed completely if can error occurs.

↔ No partial execution of transaction.

→ Consistency: The changes made by the transaction on data must be seen in the database as well.

↔ Database should be either in prior stable state (or) the new stable state.

→ Isolation: If there 2 transactions performing data modifications on the same data; then if one transaction is using that variable; then another transaction cannot access that data item until first transaction ends.

↔ Taken care by concurrency control.

→ Durability: The data held in database is modified completely; if the transaction is successful. And if transaction is not successful; the old data may be misplaced or if any system failure occurs.

– which is not accepted.

↔ Recovery subsystem helps in holding this property.

* Transaction Isolation levels in dbms.

1) Read uncommitted: → allows dirty read
   - → lowest isolation level
   - → One transaction may read not yet committed changes made by other transactions
   - → transactions are not isolated

2) Read committed: → does not allow dirty read
   - → Any data read is committed at the moment, it is read.
   - → Transaction holds a read (or) write lock on current row & prevent other transactions to read, update or delete.

3) Repeatable read:
   - → restrictive isolation
   - → Transaction holds read locks on all rows it references and write locks on all rows it inserts, deletes or updates.
   - → avoids non-repeatable read.

4) Serializable:
   - → Highest isolation level
   - → Execution of operations in which concurrently executing transactions appears to be serially executing.

* Transaction failure classification:

1) Transaction failure:
   → Any transaction execution stopped abruptly due to any of the following reasons can be called transaction failure.
   * Logical Errors
   * Syntax Errors.

2) System Crash:
   → System failure due to power failure (or) other hardware or software failure.

3) Disk failure:
   → formation of bad sectors, head crash; unreachability to the disk.

* Serializability:

↳ way to check whether 2 transactions working on a database are maintaining database consistency or not.

1) Conflict Serializability:

* A schedule is called conflict serializable if it can be transformed into a serial schedule by swapping non-conflicting operations.

→ Conflicting operations:

* Conditions:

- They belong to diff. transactions
- They operate on same data item
- Atleast one of them is a write operation.

Ex: Pairs that are conflicting:

$(R_1(A), W_2(A))$; $(W_1(A), W_2(A))$, $(W_1(A), R_2(A))$.

* Conflict equivalent: Two schedules are said to be conflict equivalent when one can be transformed to another by swapping non-conflicting operations.

→ Advantages:

- consistency: results of transactions are consistent
- Correctness: order of execution of transactions is correct.
- Reduced overhead
- Improved Concurrency.

→ Disadvantages:

- Complexity
- Reduced performance
- Limited concurrency.

2) View Serializability:

* A schedule is said to be view serializable if it's view equivalent to a serial schedule

→ If the precedence graph does not have any cycle / loop then the schedule is said to be conflict serializable

* Schedule:

↳ process of lining the transactions & executing them one by one.

→ Serial Schedule
→ Non-serial schedule.

* Advantages of Transaction Management:

→ access data concurrently.
→ manage concurrency.
→ solve read/write conflicts.
→ used for locking the data.

* Disadvantages of TM:

→ difficult to change the information
→ Rollback the transaction when there is an error.

* Recoverability: Ensures that, in the event of a failure or error, the system can recover the database to a consistent state.

1) Cascading Rollback:

if T_2 is reading value updated by T_1 and commit of T_2 is delayed till commit of T_1, the schedule is called recoverable with cascading rollback.

2) Cascadeless Recoverable Rollback:

If T_2 reads value updated by T_1 only after T_1 is committed, schedule will be cascadeless recoverable.

* Concurrency Control:

→ Ensures the simultaneous execution (or) manipulation of data by several processes (or) users.

→ When one transaction completely executes before starting another transaction, the schedule is called serial schedule.

→ When operations of a transaction are interleaved with operations of other transactions of a schedule, the schedule is called concurrent schedule.

Serial Schedule	Serializable
→ transaction will be executed one after another	→ concurrent execution of transactions
→ less efficient	→ more efficient

→ Concurrency Control Protocols:

*→ Lock-Based Protocols:

→ Shared lock: Read lock which allows multiple transaction to read the data simultaneously. (No modification possible)

→ Exclusive lock: Write lock which allows the transaction to update a data item. While a transaction is holding an exclusive lock on a data item; no other transaction can acquire shared/exclusive lock.

→ 2 phase locking protocol: → Locks are released before transactions perform commits.

*→ ensures strict order of lock acquisition & release.

* Growing phase: Transaction starts acquiring locks before performing any modification. Lock acquired is released only when transaction is completed.

* Shrinking phase: Transaction releases all acquired locks once it performs the modifications on the data item.

→ Strict 2-phase locking protocol: Transactions are only allowed to release locks only when they perform commits.

*→ TimeStamp Based Protocols: C.W

* Advantages of Concurrency:

- waiting time, response time ~~resource utilization~~ is decreased
- Resource utilization is ↑sed.
- Efficiency is ↑sed.

* Disadvantages of Concurrency:

- Overhead ↑sed
- Deadlocks are developed
- Reduced concurrency
- complexity ↑sed
- Inconsistency.

30/05/2023

* Time Stamp-Based Protocols:

↳ time when the transaction starts.

TS(t) denoted $TS(T_i) < TS(T_j)$

$T_i \rightarrow$ older transaction

$T_j \rightarrow$ new transaction.

*→ The protocol manages concurrent execution such that the time stamps determine the serializability order.

2 types:

→ W-timestamp(Q): largest timestamp of any transaction that executed write(Q) successfully.

→ R-timestamp(Q): largest timestamp of any transaction that executed read(Q) successfully.

Ex: $TS(T_1) < TS(T_2) < TS(T_3) < TS(T_4)$

$\therefore$ W-timestamp(Q) = $TS(T_4)$

↳ this value keeps on changing as and when a new transaction updates the value of Q.
i.e. W-TS / R-TS takes the TS(Tlatest) timestamp of latest transaction).

→ The time-stamp ordering protocol ensures that any conflicting read & write operations are executed in timestamp order.

TS

* If transaction T_i issues a read(Q)

1) $TS(T_i) \leq$ W-Timestamp(Q); then T_i needs to read a value of Q that was already overwritten. Hence the read operation is rejected & T_i is rolled back.

2) $TS(T_i) \geq$ W-Timestamp(Q); then the read operation is executed and R-Timestamp(Q) is set to max(R-TS(Q), $TS(T_i)$).

* If transaction T_i issues write(Q).

1) If $TS(T_i) <$ R-TS(Q); then the value of Q that T_i is producing was needed previously & the system assumed that

value would never be produced.
Hence write(Q) is rejected.

2) $TS(T_i) < W\text{-}TS(Q)$; hence write(Q) is rejected and.

3) $TS(T_i) > R\text{-}TS(Q)$ & $TS(T_i) > W\text{-}TS(Q)$:
write(Q) is executed and
$W\text{-}TS(Q) = TS(T_i)$.

→ TimeStamp based protocols leads to conflict serializable schedule.
- → There are no cycles in the precedence graph.
- → Ensures freedom from deadlock

* Lock Based & TimeStamp based protocols ensures the property of isolation.

* Recovery System

→ Storage Structure:

* Volatile Storage:
  - → cannot survive system crashes.
  - → main and cache memory
  - → fast but can store only a small amount of information.

* Non-Volatile Storage:
  - → can survive system crashes
  - → huge data storage capacity
  - → slower accessibility
  - → Hard disks, flash memory.

* Stable Storage:
  - → multiple copies of a particular data

* Recovery & Atomicity:

→ When a DBMS recovers from a crash;

1) Check states of all transactions; which were being executed.
2) ensure atomicity of transaction
3) Check whether transaction can be completed or needs to be rolled back.
4) Inconsistent state of DBMS is not allowed

* Techniques that help a DBMS in recovering & atomicity.

→ Maintaining logs of each transaction

→ Maintain shadow paging; changes are done on a volatile memory.

* Log-Based Recovery:

→ Log is a sequence of records; which maintains the record of actions performed.

→ Logs are stored on stable storage.

Steps:

1) When a transaction enters the system & starts execution; log is maintained

   $< T_n, \text{start} >$.

2) When transaction modifies a data item X

   $< T_n, X, V_1, V_2 >$

3) When the transaction finishes

   $< T_n, \text{commit} >$

→ Techniques to avoid loss of data in volatile storage:

1) A state of active database in the volatile memory can be periodically

dumped into a stable storage.

2) <dump> can be marked on a log file: whenever database contents are dumped from a non volatile memory to a stable one.

* Remote Backup:
  - provides a sense of security in case the primary location where the database is located gets destroyed.
  - Online backup systems are more real-time for database administrators.

** Once the primary database storage fails, the backup system senses the failure & switches the user system to remote storage.

* Indexing & Hashing:

→ Indexing: data structure technique to efficiently retrieve records from the database files.

→ Types

1) Primary Index: data files are ordered based on key field (primary key)

2) Secondary Index: Generated from a field which is a candidate key & has a unique value.

3) Clustering Index: defined on an ordered data file and ordered based on a non-key field.

→ Ordered Indexing:
- Dense Index:
  - has an index record for every search key value.
  - Searching fast e.g.
  - more space to store index.
  - Index records contain search key value & a pointer to actual record.
- Sparse Index:
  - Index records are not created for every ~~record~~ key value.
  - Index record contains a search key and pointer to actual data.
- Multilevel Index:
  - helps in breaking down the index into several smaller indices.

* B^+ Trees:

→ Balanced Binary Search tree that follows multilevel index format.
→ Leaf nodes of B^+ trees denote actual data pointers.
→ Leaf nodes are linked using linked lists.
→ Supports random access as well as sequential access.

* Structure:

→ Every leaf node is at equal distance from root.
→ Internal non-leaf nodes contain atleast $\frac{n}{2}$ pointers except the root.
→ Leaf nodes contain
atleast $\frac{n}{2}$ record pointers & key values
atmost n " "

** Insertion:

→ Each entry is done at the leaf node & hence filled from bottom.
→ If leaf node overflows
  ↳ split node into 2 parts
  ↳ Partition (i) = $\lfloor (m+1)/2 \rfloor$
→ If non-leaf node overflows
  ↳ split node into 2 parts
  ↳ Partition = $\lceil (m+1)/2 \rceil$

* Hashing:

↳ Effective technique to calculate the direct location of a data record on the disk without using index structure.
→ hash functions with search keys as parameters.

* Static Hashing:

→ when a search-key value is given, hash fn always computes the same address.
→ Total no. of buckets provided is always fixed.
→ Bucket-overflow: also called collision.

1) Overflow Chaining / Closed Hashing:
- When buckets are full, a new bucket is allocated for same hash result & is linked after the previous one.

2) Linear Probing / Open hashing:
- When hash fn generates an address at which data is already stored, the next free bucket is allocated to it

** Dynamic Hashing: / Extended Hashing:

→ provides a mechanism in which data buckets are added and removed dynamically on demand.

* Deadlocks

→ State of dbms having 2 or more transactions ; when each transaction is waiting for a data item that is being locked by some other transaction

→ Detected using cycle in wait-for graph.

→ Wait-for graph: directed graph where vertices are transactions & edges are ~~of~~ waits for data items

→ Deadlock Handling:

1) Deadlock prevention

- → does not allow to acquire locks that leads to deadlocks.
- → One way to prevent deadlock is to acquire locks before the transaction starts and maintain till the end due to which other transactions ~~request~~ will not take the data.

2) Deadlock Avoidance:

- → handles deadlock before they occur.
- → 2 algorithms:
  - → Wait Die: If T_1 is older T_2, T_1 is ~~you~~ allowed to wait
  - → Wound-wait: If T_1 is older than T_2, T_2 is aborted & later restarted.

* Relational Database Design:

→ Relational Model:

A relational database consists of a collection of tables.

* Attribute: properties that define an entity
* Relation Schema: structure of relation & represents the name of the relation with its attributes.
* Tuple: Each row in relation is a tuple.
* Relation instance: Set of tuples of a relation at a particular instance of time.
* Degree: No. of attributes in relation
* Cardinality: No. of tuples in a relation is known as cardinality
* Column: Represents the set of values for a particular attribute.

→ Constraints in Relational Model:

1) Domain constraints:

↳ attribute-level constraints

2) Key integrity:

↳ Every relation in the database should have atleast one set of attributes that defines a tuple uniquely which are called keys.

* Key
  ↳ should be unique for all tuples
  ↳ can't have NULL values.

3) Referential integrity:

↳ one attribute of relation can only take values from another attribute of the same relation (or) any other relation.

→ Anomalies in the Relational Model:

1) Insertion Anomaly:

↳ We can't insert a row in referencing relation if referencing attribute's value is not present in the referenced attribute value.

2) Deletion / Updation Anomaly:

↳ We can't delete (or) update a row from referenced relation if the value of referenced attribute is used in the value of referencing attribute.

3) On delete Cascade:

↳ It will delete the tuples from referencing relation if the value used by referencing attribute is deleted from referenced relation.

4) On update cascade:

↳ It will update the referencing attribute in referencing relation if the attribute value used by referencing attribute is updated in referenced relation.

5) Super Keys:

↳ Any set of attributes that allows us to identify unique rows in a given relation.

Subset of super keys used Primary Key are known as candidate keys.

Combination of 2 (or) more attributes that are being used as primary key is called composite key.

* Advantages of RM:
  - Simple Model
  - Flexible
  - Secure
  - Data Accuracy
  - Data integrity
  - Operations can be applied easily

* Disadvantages of RM:
  - Not good for large databases.
  - Difficult to find relation blw tables
  - Complex structure; response time is more.

* Characteristics:
  - Data represented as rows & columns.
  - Stored as tables.
  - Supports data manipulation, data definition & transaction management.
  - Each row represents single entity.

ODBC	JDBC
→ Open database Connectivity.	→ Java database connectivity
→ By Microsoft	→ Sun Micro Systems
→ any language: C, C++, Java	→ only java languages
→ only windows platform	→ any platform
→ Procedural	→ Object oriented.

* Normalization (Defn & advantages)
  - Organizing the data in database
  - Minimize redundancy & also eliminate undesirable characteristics like insertion, update and delete anomalies.
  - Data consistency.

* Disadvantages:
  - Performance degrades when normalizing the relations to 4NF/5NF
  - Time consuming.
  - Careless decomposition leads to bad database design.

* Normal forms:

→ First Normal Form:

↳ Every attribute of the relation is a single valued attribute.

Ex:

Stud_No	Stud_Phone
100	9713545681, 7896354210
102	9785643201

↓ 1NF

Stud_No	Stud_Phone
100	
100	
102	

→ Second Normal form: → rigid

↳ must be in 1NF

↳ No partial dependency:

↳ If proper subset of candidate key determines non-prime attribute

* Prime attribute:

↳ which forms a super key

↳ must be a member of candidate key/primary key.

* Non-prime attribute:

↳ Cannot form super key

↳ not a member of any candidate key.

→ Every non-prime attribute is fully functionally dependent on primary key.

→ Third Normal form:

↳ Depends on transitive FD.

* Transitive FD: a F.D $x \rightarrow z$ that can be derived from 2 F.Ds $x \rightarrow y$ & $y \rightarrow z$.

→ must be in 2NF.

↳ No non-prime attribute A in R is transitively dependent on P.K.

→ Boyce-Codd NF:

↳ must be in 3NF

↳ X shd be superkey for every FD $X \rightarrow Y$ in a given relation

* Functional Dependency:

$\alpha \rightarrow \beta$ where $\alpha \subseteq R$ & $\beta \subseteq R$

if $t_1[\alpha] = t_2[\alpha]$ then

$t_1[\beta] = t_2[\beta]$

→ Let $\alpha \to \beta$ is a trivial functional dependency if $\alpha \subseteq \beta$.

* F : set of functional dependencies.
  F^+ : closure set of "

* Armstrong's axioms:
  1) If $\beta \subseteq \alpha \Rightarrow$ then $\alpha \to \beta$: Reflexivity
  2) $\alpha - \beta \Rightarrow \gamma\alpha \to \gamma\beta$: augmentation
  3) $\alpha \to \beta \Rightarrow \beta \to \gamma$ then $\alpha \to \gamma$: Transitive

* Union rule:
  $\{\alpha \to \beta, \alpha \to \gamma\} \Rightarrow \{\alpha \to \beta\gamma\}$

* Decomposition:
  $\alpha \to \beta\gamma \Rightarrow \alpha \to \beta$ & $\alpha \to \gamma$

* Pseudotransitivity:
  $\alpha \to \beta$ & $\gamma\beta \to \delta \Rightarrow \alpha\gamma \to \delta$

* Canonical Cover: (F_c)

→ If an attribute A is removed; functional dependencies does not change then such attribute is called extraneous attribute

→ F_c does not contain extraneous attribute.

Ways to find F_c:

Initially: $F_c = F$

1) Apply union rule if I can merge any FDs.
2) Update F_c
3) For any FD $\alpha \to \beta$ any attribute A be either on LHS/RHS is an extraneous attribute then
   $F_c = F_c - \{\alpha \to \beta\}$

* Rules:
  1) A is an attribute which is in LHS
     i.e. $A \subseteq \alpha$
     $\gamma = \alpha - \{n\}$
     find $(\gamma)^+$ from F; if γ^+ includes β then A is extraneous.
  2) if $A \subseteq \beta$

$F' = \{F - \{\alpha \rightarrow \beta\}\} \cup \{\alpha - (\beta - A)\}$

→ Find α^+ from F'; if α^+ includes A then A is extraneous attribute.

* If any common attribute form super key for $R_1 | R_2$; then decomposition is called lossless decomposition.

* Applications of NFs:

- → Data consistency
- → Data redundancy
- → Query performance.
- → Database Maintainence
- → Database design.