DBMS

19/01/24

Introduction :

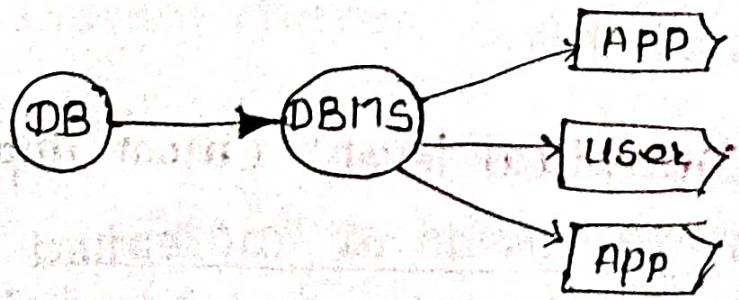
- **Data** :- Collection of raw and unorganized facts.
  - Data doesn't carry any specific purpose.
  - Types:
    - i) Quantitive : Numerical (eg. weight)
    - ii) Qualitative : Descriptive (eg. Name)

- **Information** :- Processed, Organized and structure data.
  - It makes sense.

- **Data Vs Information** :
  - Data is collection of fact, info. puts those facts into context.
  - Data does not depends on info., Info depends on data

- **Database**: Collection of related / similar data.

- **DBMS** :- Database management system.
  - Collection of data which is related and set of program to access those data.
  - DBMS is database itself along with s/w and functionality.



- DBMS advantages / File system disadvantages
  - Data redundancy : Duplication of data
  - Data inconsistency : data redundancy leads to it.
  - Difficulty in accessing data : data scattered in files.
  - Data isolation
  - Integrity problem : No automatic constraint check.
  - Security problem : unauthorized access.
  - Atomicity issue : Atomicity in transaction is difficult.

DBMS Architecture :

- View of schema :
  - major purpose of DBMS to provide an abstract view to users. (i.e hide details of how the data is stored & maintained).
  - Three schema architecture is provided.

(a) Physical level / Internal level :

- lowest level describes how data are stored.
- low-level data structures are used.
- describes physical storage structure of DB.

* storage allocation, Data compression & encryption, etc
  - Algorithm that allows efficient data access are defined here

(b) Logical level / conceptual level : (most imp)

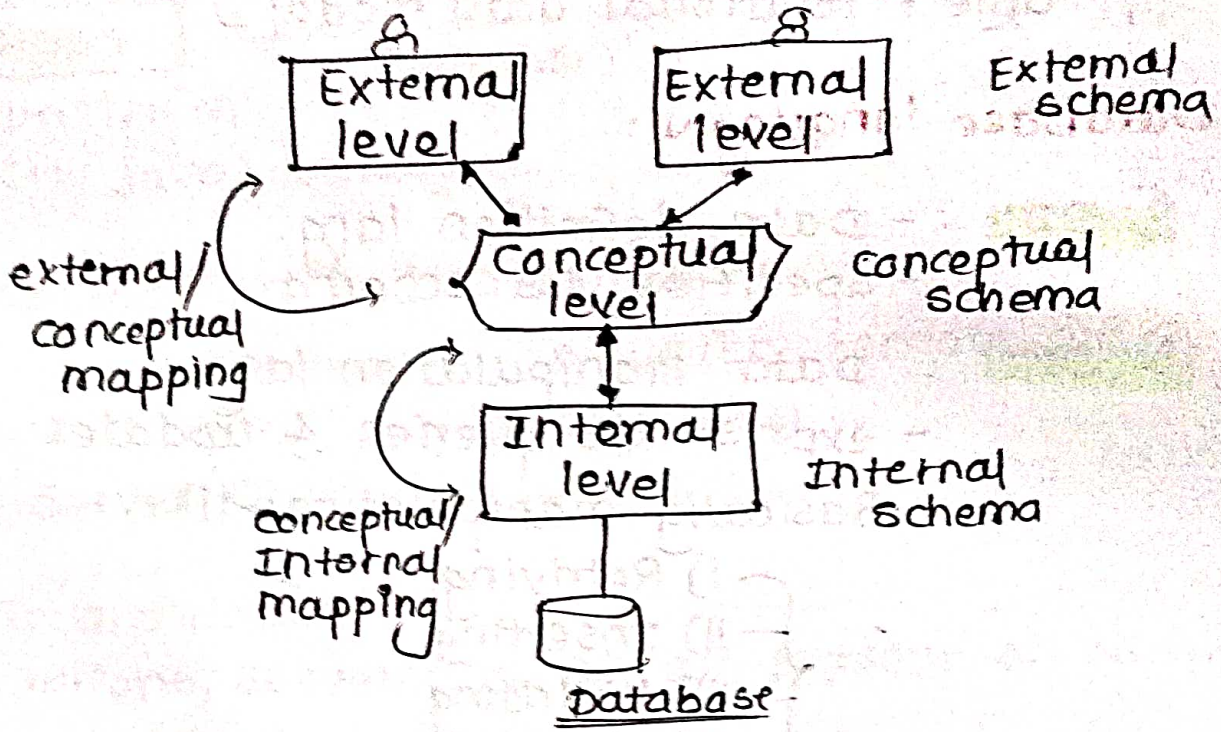
- describes DB design at conceptual level & describes what data are stored in DB,

and what relationship exist among them.

* DBA decides it.

⑤ View level / External level

- Also called user level
- describes the database part that a particular user group is interested and hides remaining db.
- contains several schemas (called as subschemas).



* Instances and Schema:
  - Collection of information stored in the DB at particular moment is called Instance of DB.
  - Design of DB is called schema.
  - schema is structural description of data which doesn't change frequently.
  - Three schema
    - physical schema
    - logical schema
    - View schema (subschema)
  - physical schema change should not affect logical schema (i.e physical data independence)

* Data model :

- Provides way to describe the DB design at logical level.
- Underlying DB structure is called data model.

Eg. ER model
Relational model
Object-oriented model
Object-relational data model

} describes data, rln, constraints, etc.

* Database languages :

(a) DDL : - Data definition lang.
- specifies DB schema.

(b) DML : Data manipulation lang.
- express db queries & updates
- Basically manipulations like
  - i) Retriving
  - ii) Inserting
  - iii) Updating
  - iv) Deleting
- DQL (Data query lang : for retriving)

(c) DCL - Data Control lang.

(d) TCL - Transaction control lang.

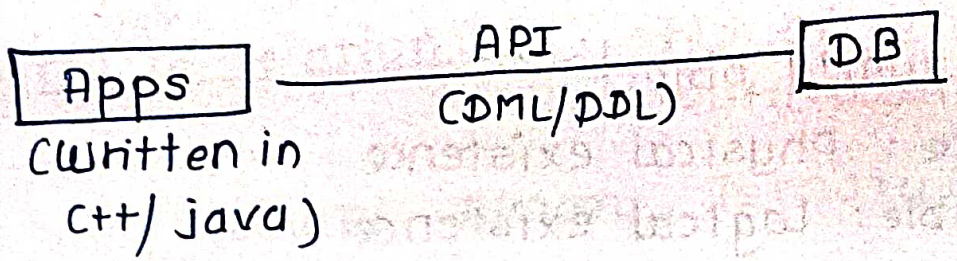
* DBA :

- Database Administrator
- Person who has control of both the data & the set of program that access those data.

DBA functions :

- Schema definition
- Storage structure & access methods
- Schema & physical organization modifications
- Authorization control
- Routine maintenance

* DB access ?



* DBMS Application architecture :

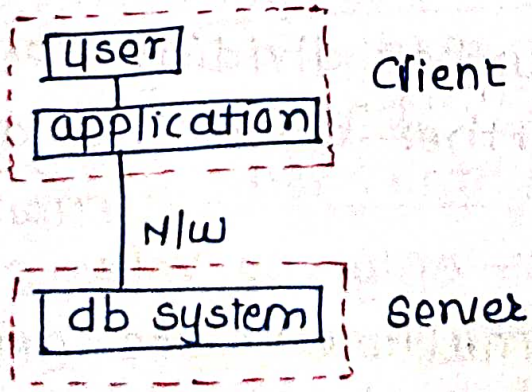
Client machines - on which remote DB users work.

Server machines - DB system runs here.

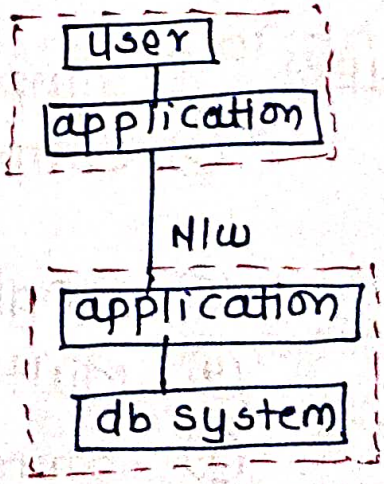
① T1 architecture :

Client, server and DB all present on same machine.

② T2 architecture :



③ T3 architecture



* Entity Relationship model :

- ER model :
  - It is high level data model
  - Consist of collection of basic objects : entities and Hnship among them
  - Graphical representation of ER model is ER diagram

- Entity : Real world object
  - Tangible : physical existence
  - Intangible : Logical existence

Types ① strong entity : uniquely identified.
② Weak entity : Can't uniquely identified.

- Entity set : - Collection of similar entities of same type. eg : student

- Attributes : - Properties of entity.
  - Attribute has domain (acceptable values)
  - each attribute has its value.
  - eg : Stud-ID
    Name

Types : ① simple : No further dividing (RollNo)
② Composite : further divided (Name)
FName LName
③ single valued : only one value (RollNo)
④ Multivalued : multiple values (MobileNo)
⑤ Stored : physically stored (DOB)
⑥ Derived : Derived framed stored (Age)
⑦ NULL value : Take NULL value, if there is no value for it

Relationships :

- links betn two or more entities.
- Association among entities.

eg - Person has car

Types : 1] strong rlnship : betn two independent entities.

2] Weak rlnship : betn weak entity

Degree of relationship : No. of entities participating in relationship

Unary, Binary, Ternary

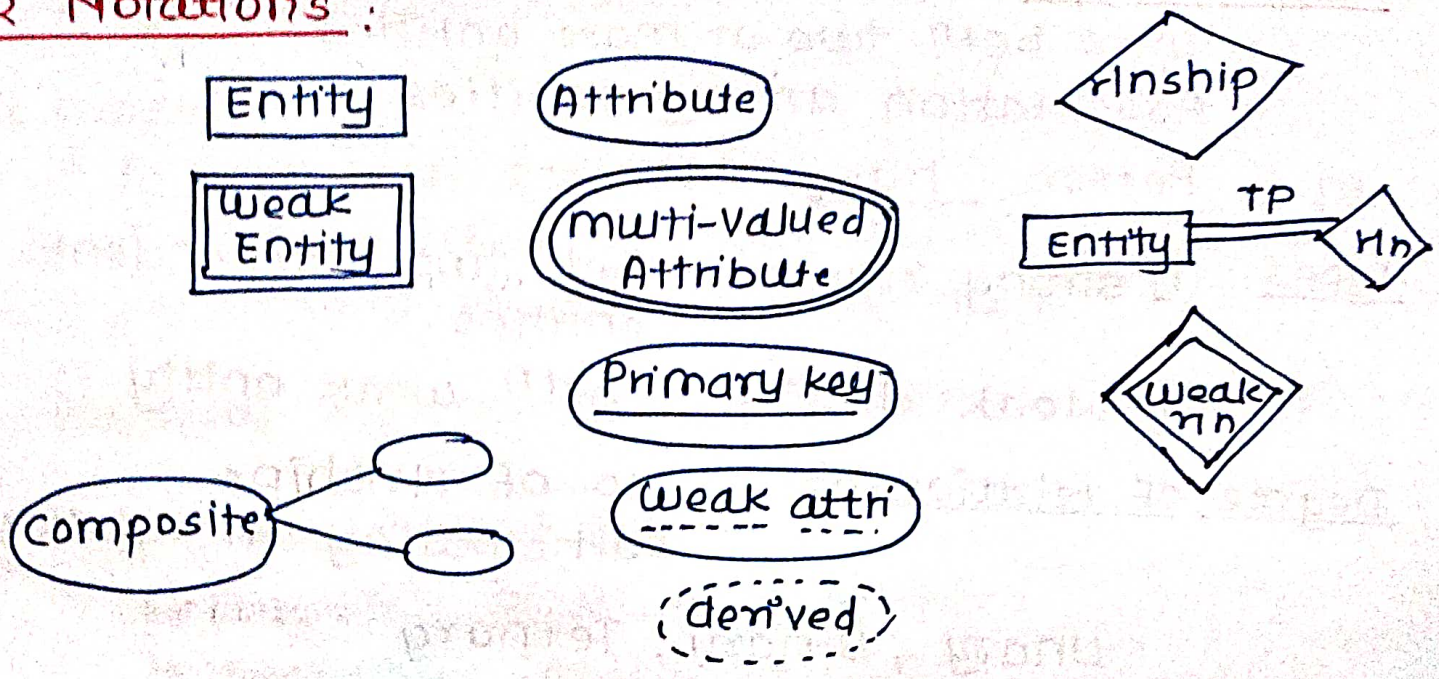
Relationship constraints :

① Mapping cardinality / cardinality ratio :
- No. of entities to which other entity can be related via rlnship.
  - one to one
  - one to many
  - many to one
  - many to many

② Participation constraints :
- Specifies whether the existence of an entity depends on its being related to another entity
- minimum cardinality
  - ① Partial participation (PP) : Not all entities are involved in rln
  - ② Total participation (TP) : each entity must be involved in at least one rln.

- weak entity has TP
- Strong entity may not have TP

ER Notations :



* Extended ER Features :

- **specialisation :**
  - splitting up entity set into further sub-entity sets on the basis of their functionality & features
  - It is top-down approach
  - depicted be triangle ▽
  - is-a rlnship betn superclass & subclass.
- **Generalisation :**
  - Reverse of specialisation
  - is-a rlnship betn subclass & superclass.
  - It is bottom-up approach.
- **Attribute inheritance :**
  - Attributes of higher level entity set is inherited by lower level entity set.
  - Both specialisation & Generalization has attribute inheritance.

- participation inheritance:
  - if parent entity set participates in rlnship then child entity set also.
- Aggregation:
  - Relationship among relationship is shown by it.

Relational Model:

- Organises data in the form of relations (tables).
- Collection of tables is Relational model.

Row: represents rln among a set of values.

Tuple: single row of table representing unique record.

Column: Represents attributes of the relation.

Relation schema: Defines design & structure of the relation, name of relation and all columns/attributes.

Degree of table: No. of columns in a table.

Cardinality: Total No. of tuples in a given relation

Relational key: set of attribute which uniquely identifies each tuple.

$2^n - 1$

① Super key (SK): - attribute or set of attributes which uniquely identifies each tuple.

② Candidate key (CK): SK whose proper subset is not a SK (super key).

③ Primary key:
- It is the candidate key with less no. of attributes

④ Alternate key: (AK)
- All CK except PK

⑤ Foreign key: (FK)
- use for creation of relation among two tables.
- FK in one table refers to PK in another table

⑥ Composite key:
- PK formed using at least 2 attributes

⑦ Compound key
- PK formed using 2 FK.

⑧ Surrogate key
- Generated automatically by DB, usually integer value.

• Integrity constraints:
- To avoid data corruption

⊙ Domain Constraints:
Data type restriction

⊙ Entity constraints:
PK != NULL

⊙ Referential constraints:
specified between two relations & help to maintain consistency among tuples of two relations.

- It requires that value appearing in specified column of any tuple in referencing relation also appear in the specified column of at least one tuple in referenced relation.
- FK must have matching PK.

⊙ Key constraint:

1) NOT NULL:
2) UNIQUE:
3) DEFAULT:
4) CHECK: checks data integrity should be maintained before and after CURD.
5) PRIMARY KEY: unique and not null values.
6) FOREIGN KEY: This will prevent every action which can result in loss of connection beth tables.

Transformation of ER model - Relational Model
(How to convert ER - RM steps are below)
ER diagram notations to relations:

1) Strong entity:
   - Convert to individual table and attributes and names will become columns for table.
   - Entity PK' to Relation/Table PK
   - FK are added for establishing rlnship.

2) Weak entity:
   - Convert to individual table.
   - PK from its corresponding strong entity will be added as FK.
   - PK will be composite.

8] Aggregation :

- Table for the relationship set is formed.

SQL : structured Query Language :

- SQL is used to access and manipulate data.
- mainly designed for RDBMS.
- SQL uses CRUD for communicating with DB
  - Create : create and insert data to table.
  - read : Read from table
  - update : Modify
  - delete : Remove tuple / row
- SQL is not DB, it is query language.

RDBMS :

- slw that enable us to implement designed relational model.
- eg. Ms SQL, MySQL, IBM, etc.
  (open source)

- mySQL is RDBMS
- SQL is query lang

- SQL data types :
  - Nature of data to be stored into table.

⊙ String
- CHAR
- VARCHAR
- BINARY
- TEXT
- BLOB

⊙ Numeric
- BIT
- INT
- FLOAT
- DECIMAL
- BOOL
- DOUBLE

⊙ Date & Time
- DATE
- YEAR
- DATETIME
- TIMESTAMP

- SQL Commands :

① DDL (Data Definition lang)

- Related to schema designing
- commands :
  - create - Creation of table, DB
  - alter - alter / modify table structure
  - drop - delete structure & data of table
  - truncate - only deletes records from table
  - rename - modification in name

② DML (Data Manipulation Lang)

- for modification
  - insert - data insertion to table
  - update - modify data
  - delete - Removes one or more row.

③ DQL / DRL (data retrieval Lang / Data query Lang)

- select : used to fetch data

④ DCL (Data control Lang)

- Used to provide and take back authority.
- commands
  - Grant : gives permission
  - revoke : takes back permission

⑤ TCL (Transaction control Lang)

- start transaction
- commit : Save all transaction
- rollback : undo transaction that have not save.
- savepoint : rollback upto savepoint

managing DB : queries related to db.

Creation of DB :

Query – Create database dbname;

Check DB :

Query – show databases;

Delete DB :

Q – DROP database dbname;

Select DB :

Q – use dbname;

select tables from db.

Q – show Tables;

2) queries related to table

Create table

Q – create table tbName
(
col1 datatype,
col2 datatype
:
);

delete records + data /table structure

Q – Drop table tbName;

delete rows from table

Q – delete from tbName [Where condition]

Truncate table : only deletes records

q - truncate table tbName;

Rename table

q - rename old_Name to New_Name,

copy one table data to other

q - select * into Newtb from oldtb;

Alter table :

(i) add column :

q - alter table tbName add colname datatype;

(ii) modify

q - alter table tbName modify col1 datatype;

(iii) Drop column

q - alter table tbName drop column col1;

(iv) Rename

q - alter table tbName rename column
old to New;

* Select data from table :

- select * from tbName;
- Select col1, col2 from tbName;
- select * from tbName [where condition]

* where : reduces row based on given condition.

- select * from tbName where age >18;

* between : inclusive selection

- select * from tbName where age

between 0 and 19 ;

* IN : reduces or conditions
  - select * from tbName where age IN (10,20);

* and/or/Not : combining of conditions
  - ——"——— where cond1 and cond2
  - ——"——— where cond1 or cond2
  - ——"——— where age Not IN (10,20)

* is null :
  - select * from tb where age is null;

* pattern searching / Wildcard (%, _)
  - % can be any no. of char
  - _ only one char.

  - select * from tbName where name like '%';

* Order by
  - using ASC / DESC / or by default ascending.
  - select * from tbName order by Name DESC (by default ascending)

* Group by
  - It is used to collect data from multiple rows and group the result by one or more column.

SQL clause:

① **where** : - used for filtering records.
- used in select, update, delete statement
- select * from tbName [where condition];

② **AND** - used with select, update, delete, insert
- Combines two conditions
- select * from tbName where cond1 AND cond2

③ **OR** - Combines two conditions.
- select * from tbName where cond1 OR cond2;

④ **AS** (Alias) - To assign temp name to table's column.
- select col1 AS temp1 from tbName;

⑤ **SQL having** - we can't use where clause with aggregate functions thus having clause is used.
- select col1, col2, aggregatefun(col3) from tbName group by col1 having condition

Having	vs.	where
- executed with group by clause		- executed without group by
- aggregate function		- can't use them
- use with select		- use with update, delete, select
- Group filtering		- row filtering

6] order by :

Group by	vs	Order by
- Groups the rows		- sorts in ascending / descending order.
- used before order by		- used after group by
- mandatory aggregate functions		- Not mandatory

7] Group by :

- organize similar data into groups
- used with order by
  where
  select

- select col1, aggregatefun(col2)
  from tbName
  group by col1;

- If want filtering using having clause after group by.

- Aggregate functions :
  - count()
    - count(*) — Null + Not Null
    - count(col1) — only not Null
    - count(distinct col1) — unique not null
  - sum()
  - min()
  - max()
  - AVG()

Constraints:

1) primary key:

```
Create table tbName(
    id INT PRIMARY KEY
    PRIMARY KEY (id));
```

2) foreign key:

```
Create table tbName (
    id INT PRIMARY KEY
    id2 INT
    FOREIGN KEY (id2) References
    tbName2 (id));
```

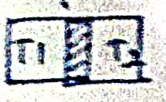
3) UNIQUE / CHECK / DEFAULT

```
Create table tbName(
    email varchar(50) UNIQUE,
    CONSTRAINT age_chk Check (age>12),
    Weight INT NOT NULL DEFAULT 20,
);
```

Joining tables:

① Inner Join:

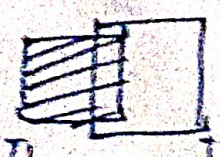
- returns resultant tables that has matching values from both the tables



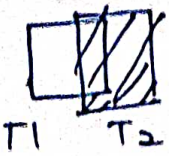
- Select col1, col2 from table1 INNER JOIN table2 ON condition;

② Outer Join:

i] Left Join: all data from left table and matching data from right table.



ii] Right JOIN :
- all the matched data from left table and all data from right



iii] Full join:
- all the data from right and left table is returned.

Websites

Last min : JavaTpoint ←
GFG ←
InterviewBit ←

• Normalization

- Reduces data redundancy and improves data integrity

what if there is redundant data?

— Insertion/ Deletion/ updation anomalies arises

functional dependency: It in betn two set of attributes.

$A \rightarrow B$

① Trivial FD:

$AB \rightarrow A$

(determinant) (dependent)

Here dependent A is subset of determinant AB

② Non-trivial

$AB \rightarrow C$

Here determinant is not a subset of AB.

Types of normal forms:

① 1 NF:

- Every relation cell must have atomic value
- Relation must not have multivalued attributes.

② 2NF

- Relation must be in 1NF
- There should not be any partial dependency

Partial FD → non-prime attributes determined by part of candidate key.

- If BCD is candidate key then B, C, D are prime attributes.
- BCD → A Here A is non prime attribute.

③ 3NF:
- Relation must be in 2NF
- No transitivity dependency.
  (Non-prime determines non prime)

④ BCNF
- Relation must be in 3NF
- left side i.e determinant should be a super key.

Concurrency Control :

- procedure required for controlling concurrent execution of the operations that take place on database

Concurrent execution:

- multiple user can access and use the same database at one time, which is known as concurrent execution.

→ Goal is to develop concurrency control protocol to ensure serialbity.
→ To develop a schedule which is serializable.

Protocols

① Lock-Based protocol :

- Lock mechanism is used to control concurrent access of data item.

Two modes in which data items can be locked.

Exclusive (X) mode

→ Transaction can perform read and write operations.
→ lock-x(Q)
→ Any other transaction can't obtained either exclusive/shared lock.

shared (S) mode

→ can perform read operation
→ lock-s(Q)
↳ transaction.
→ we can apply same lock on same data item at same time in any other transaction

* Compatibility

	S	X
S	T	F
X	F	F

T1 :
lock-S(A)
read(A)
unlock(A)
lock-S(B)
read(B)
unlock(B)
display(A+B)

} Not sufficient to guarantee serializability

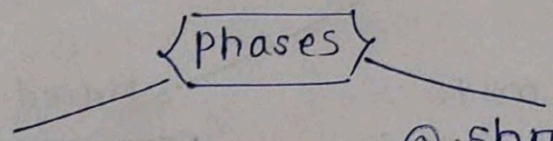
⇓

T1
lock-S(A)
lock-S(B)
Read(A)
read(B)
display(A+B)
unlock(A)
unlock(B)

} Serializability achieved

2) Two-phase locking (2PL)

- Protocol that ensures conflict serializable schedules.
- Requires both lock and unlock being done in two phases.



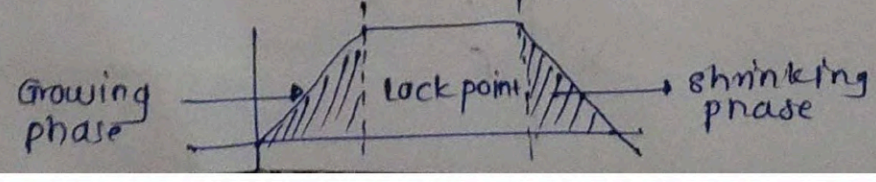
① Growing phase

- New lock on items can be acquired
- But no lock can be released.

② Shrinking phase

- Release existing locks
- But no new lock can be acquire.

② Lock point :- point at which growing phase acquires its final lock and then shrinking phase can be started



Variations of 2PL locking protocol :

① Conservative (static) 2PL :
- Acquire all locks before it starts
- Release all locks after commit
- It is deadlock free
- Avoids cascading rollback.

② strict 2PL :
- Transaction holds all exclusive locks till commit / abort
- avoids cascading rollback.
- deadlock may occur.

③ Rigorous 2PL :
- In regrous 2PL both shared & exclusive locks till commit
- avoids cascading rollback
- deadlock may occur.

Implementation of Locking :
- Separate process is runned by lock manager
- send Lock / Unlock request
- Lock manager will Grant / reject request (Transaction will wait for answer)
- Lock table will be maintained by Lock manager.

3) Graph Based protocol :
- Alternative for 2phase locking protocol.
- Additional info about transaction i.e "how each transaction will access data item" will be required.
- There are various method to get that additional information.

- one method can be having prior knowledge about the order in which database items will be accessed.

- i.e partial ordering
  if data items $D = \{d_1, d_2 \dots d_n\}$
  and if $d_i \rightarrow d_j$
  then d_i must access first then access d_j

- Set $D \Rightarrow$ directed acyclic graph called as database graph

- It is free from deadlock
- No roll back required
- ensures conflict serializability

4) Time-stamp based protocol:

- It is tag that can be attach to any data item or transaction, which denotes specific time when transaction or data itemes are activated
  - unique timestamp
  - Newer transactions timestamp strictly < older —11—
  - Timestamp order = execution order.

methods → system clock
→ Logical counter

→ W-TS(Q) = write timestamp of transaction
→ R-TS(Q) = Read —11—

Time-stamp ordering protocol:

- Ensures that conflicting Read & write operations are executed in time-stamp order.
- Free from deadlock
i) Ti issues Read(Q):

Cases :

(i) If $TS(T_i) < W(Q) - TS$, T_i is an older transaction than last transaction that wrote the value of Q.

→ Request failed

(ii) If $TS(T_i) \geq W(Q) - TS$

→ Here Request granted
→ T_i is allowed to read updated value of Q.

ii) T_i issues write(Q) :

Cases

[i] If $TS(T_i) \geq W(Q) - TS$ and $TS(T_i) \geq R(Q) - TS$

→ then T_i is allowed to modify/write value of Q
→ and $TS(T_i)$ will become current value of W(Q)

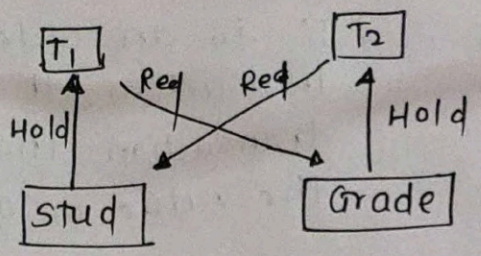
[ii] If $TS(T_i) < R(Q) - TS$, Younger transaction is already using value of Q

→ updation not allowed

[iii] if $R(Q) - TS \leq TS(T_i) \leq W(Q) - TS$, Younger transaction has updated Q.

Deadlock in transaction :

- conditions where two or more transactions are waiting indefinitely for one another to give up the locks
- None task ever gets finished and is in waiting state forever



deadlock

- T1 is waiting for T2 to release lock & vice versa for T2
- All activity comes to halt until deadlock detected & resolved.

Deadlock Avoidance :

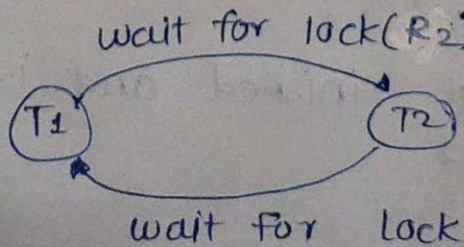
- when DB stuck in deadlock state, then it is better to avoid the database rather than aborting / Restarting
- for smaller databases = wait for graph method is used

  larger databases = deadlock prevention method is suitable.

Deadlock Detection :

- If transaction waits indefinitely to obtain a lock, then DB should check transaction is in deadlock state or not?

- **wait for graph method :**
  - Graph is created, based on transaction & their lock.
  - If resultant graph has cycle then deadlock is present.
  - It is maintained by system for transactions waiting for data.
  - Regularly check for cycles.



Deadlock prevention :

- used for large databases.
- DBMS analyses transactions and never executes those transactions that are creating deadlocks.

Two methods

(1) wait - Die scheme

- If transaction request for resource which is already held with a conflicting lock by another transaction then DBMS simply check for timestamps of both transactions

i) if $TS(T_i) < TS(T_j)$

then Older transaction waits until resource becomes available

ii) if $TS(T_i) \geq TS(T_j)$

T_i aborted and rolled back

(2) wound - wait scheme

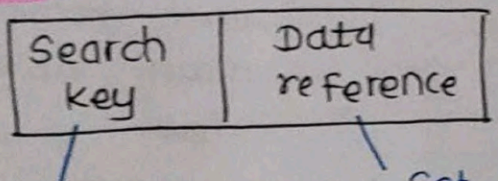
- If older one request resources held by younger one, the older forces younger to abort.
- If older one holds resources requested by younger one, then younger one must wait until it is released.

Indexing and Hashing

Indexing :

- used to optimize the DB performance by minimizing no. of disk access required while query processing
- Index is type of data-structure
- used to locate & access data in DB table quickly.

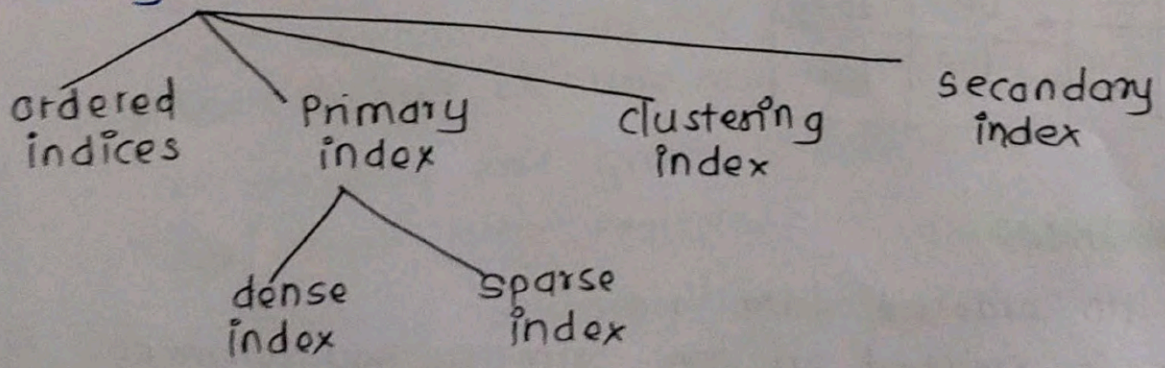
Index structure :



- Copy of primary key or candidate key
- In sorted form

Set of pointers holding the address of the disk block where value of particular key can be found.

Indexing methods :



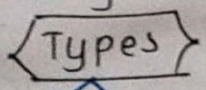
① Ordered Indices :

- are sorted indices
- which makes searching faster

② Primary index :

- Index is created on the basis of primary key of the table

- As primary keys are unique and stored in sorted order the searching will be faster.



Dense Index

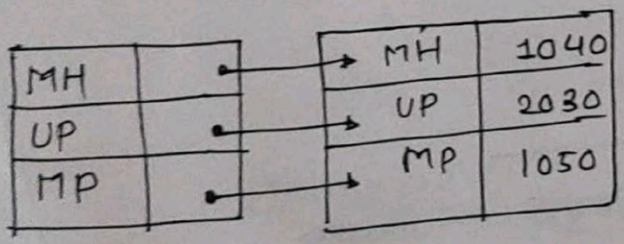
I] Contains an index record for every search key value in the data file.

II] Thus searching is faster.

III] no. of record in index table = no. of record in main table.

IV] Needs more space for storing record itself.

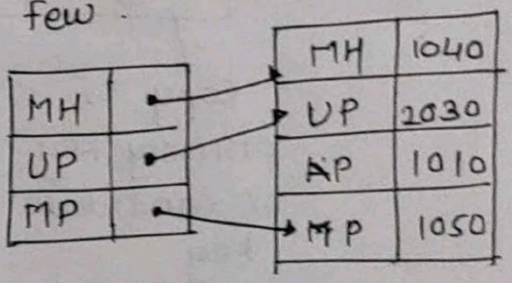
V] Index records have search key & a pointer to actual record.



Sparse Index

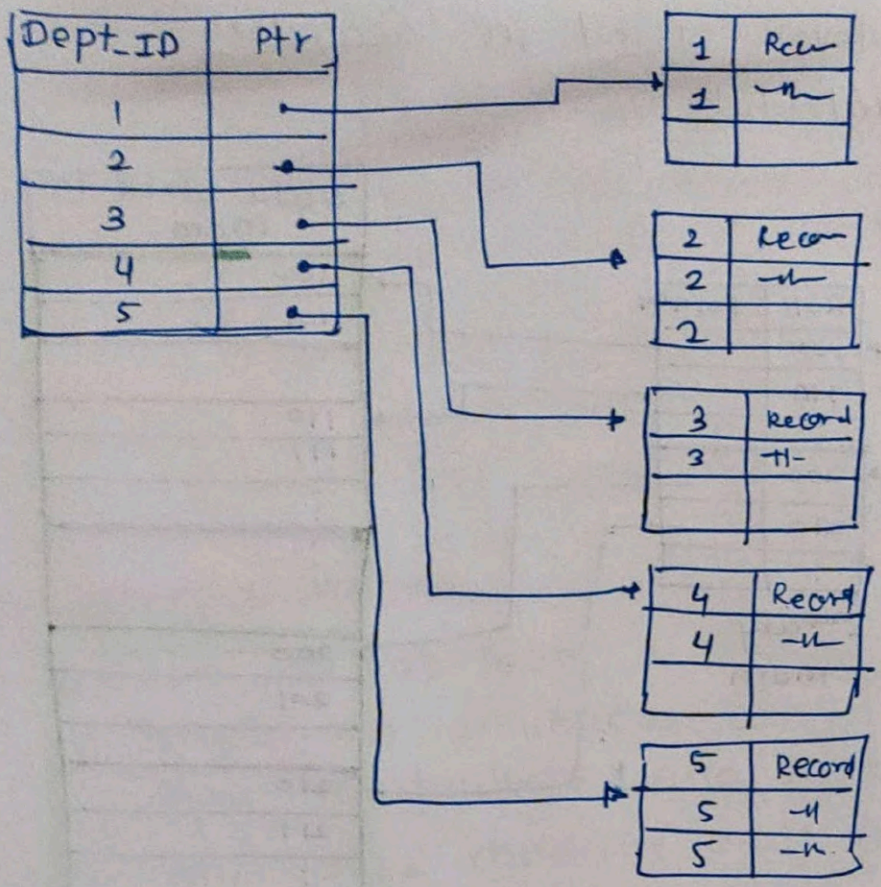
I] In data file, index record appears only for a few items.

II] Instead of pointing to each record in the main table, points to few.



3) Clustering Index :

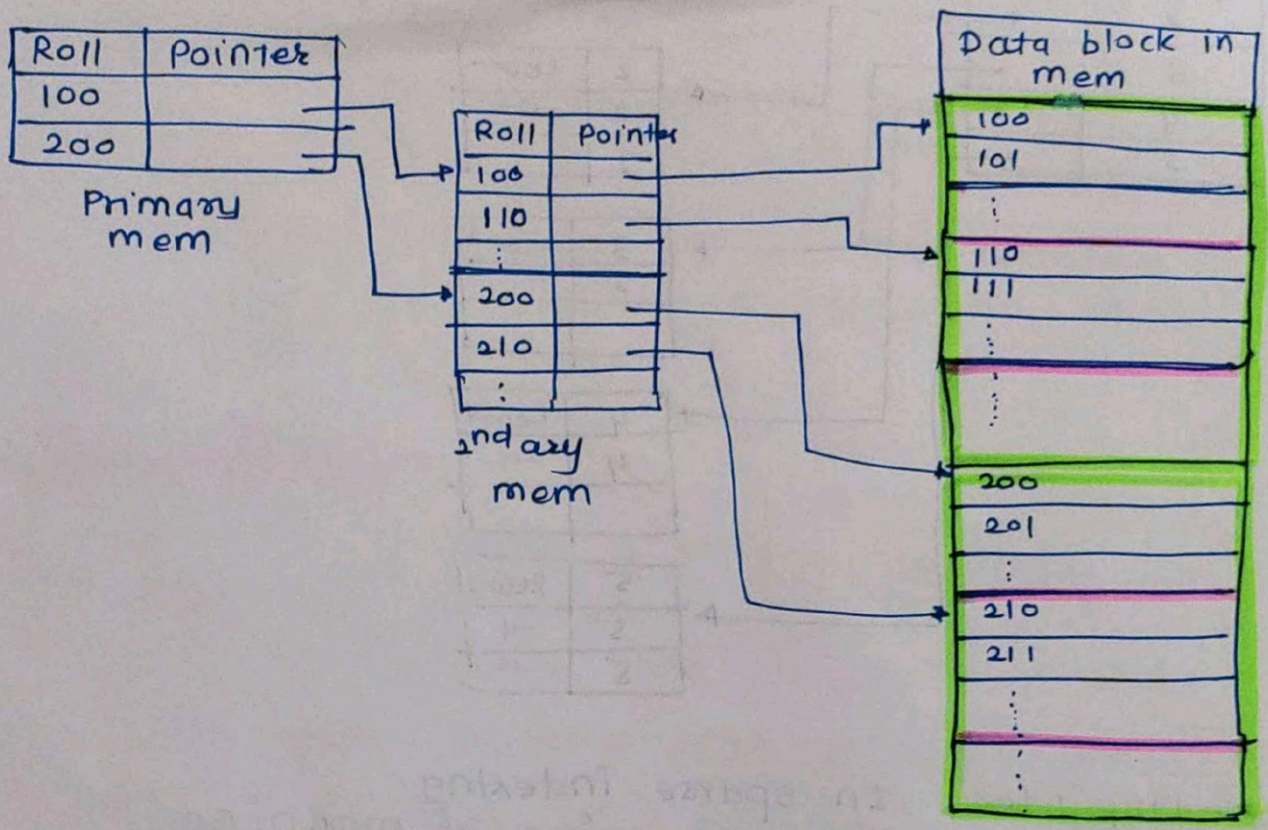
- It is an ordered data file
- Index is created on non-primary key columns which may not be unique.
- To identify record faster, we will group two or more columns to get unique value & create index from them.



4) Secondary index : In sparse indexing
- Size of table grows size of mapping also grows
- These mappings are kept in primary memory so that address fetch should be faster.
- Then secondary memory searches actual data based on address got from mapping
- If mapping size grows fetching time increases. (ie more time requires)

To overcome these prblms
- In this method
  Large no of columns are selected initially so that mapping size of first level becomes small.
- Then each range is further divided into small ranges.
- mapping of first level stored in primary memory.

- mapping of 2nd level stored in secondary memory with actual data.



B + Tree :

- B+ tree is a balanced binary search-tree
- follows multilevel index format
  - leaf nodes = actual data pointers
  - B+ ensure that all leaf nodes remain at same height.
  - leaf nodes are linked using linkedlist.
  - B+ support random and sequential access.

Structure of B+ Tree :

- every leaf node is at equal distance from root node
- It contains internal node & leaf node.

Internal node —— internal node can contain at least n/2 record pointers except root node

—— At most internal node of tree contains n pointers.

leaf node — contain at least $n/2$ record pointers & $n/2$ key values

- At most n records and n key pointers values.
- Every leaf node of B+ tree contains one block pointer P to point to next leaf node.

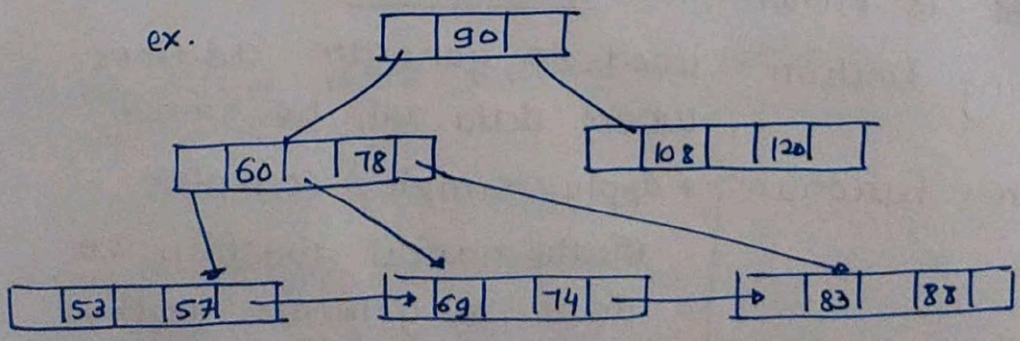
Properties of B+ tree :

- There are at least 2 children for root
- except root all nodes can have max m childrens and max 'm-1' keys
- min m/2 childrens and $(m/2 - 1)$ keys

In B+ tree

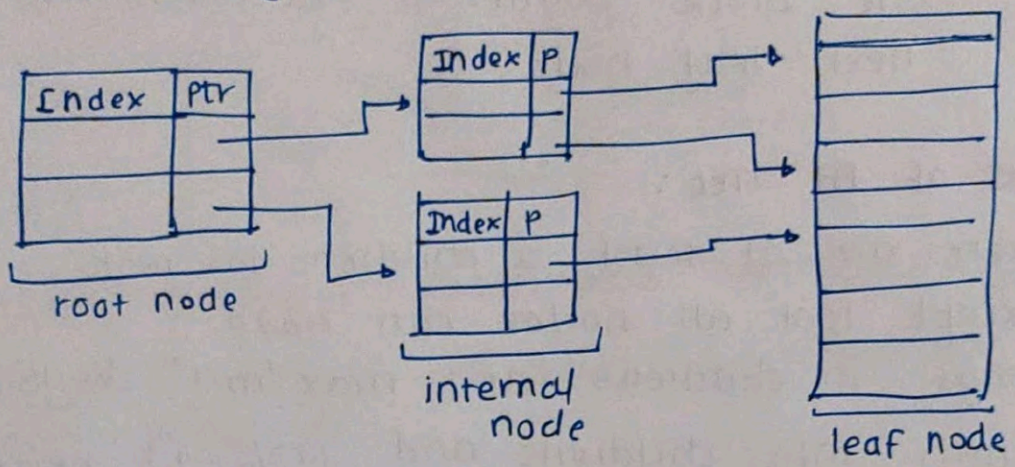
- key value are placed in internal node
- Record and data placed in leaf node.
- leaf nodes are connected as singly linked list for fast accessment
- internal nodes → In main memory
  Leaf nodes → In secondary memory

ex.



* Multilevel Indexing :

- refers to a hierarchical structure of Indexes.
- Each level of index provides a more detailed reference to the data.
- allows faster data retrieval.
- B+ tree is type of multilevel Indexing



Hashing

- In huge DB it is very difficult to / inefficient to search all index values and reach to desired data
- So, Hashing is the technique used to calculate the direct location of data record on the disk without using index structure
- memory location where these records are stored is known as data bucket.
- Hashing function :- used to generate address where data will be stored.
- Hash function
  - apply simple / complex mathematical function on data to generate address
  - Any column value can be choose to generate address.
  - most of the time primary key

→ Types of Hashing :

① Static Hashing

② Dynamic Hashing

① Static Hashing

- Resultant data bucket address is always same.
- Ex.

Hash function is mod 5

suppose, Stud_ID = 103

∴ 103 % 5 = (3) —— always same.

Here, no. of data bucket in memory are constant i.e 1-5 —— (for mod 5)

▶ operation of static hashing

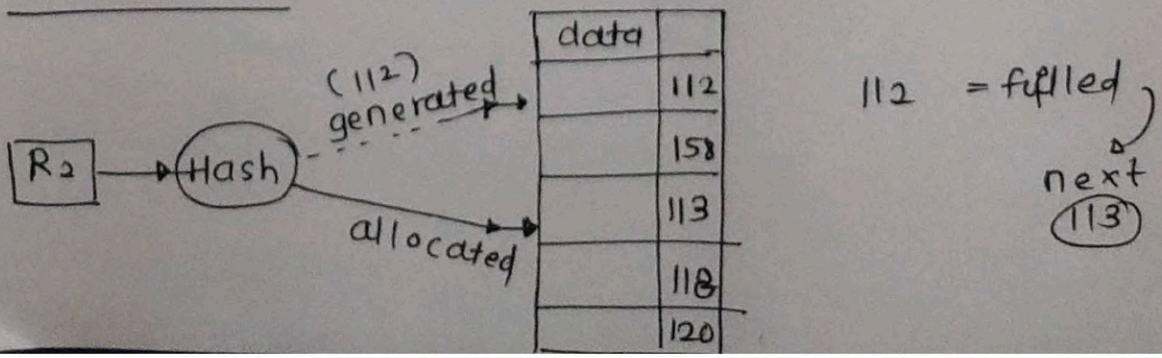
- searching
- Insert
- Delete
- Update

If we want to insert new record, but address generated by hash function is not empty or data already exists there, this situation is bucket overflow.

To overcome,

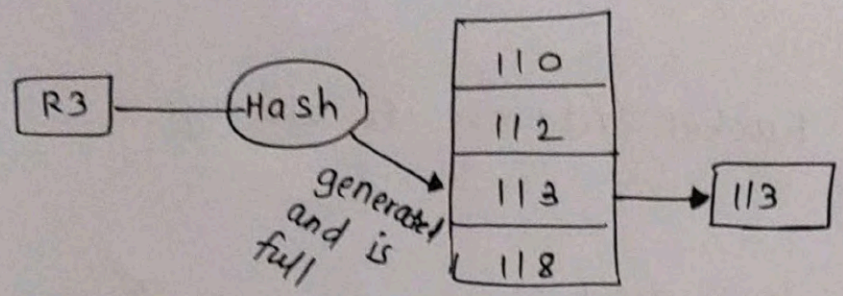
① Open hashing (linear probing) :

- when hash function generates an address at which data is already stored, then next bucket will be allocated.



② close hashing (overflow chaining)

- When buckets are full, ~~same~~ then new data bucket is allocated for the same hash result and is linked after previous one.



② Dynamic Hashing (Extendable hashing):

- Overcome problem of bucket overflow of static hashing
- Here data buckets grows & shrinks as per records increases / decreases.
- Insertion and deletion without resulting in poor performance.

Adv
- performance doesn't decrease with data growth
- memory is well utilized
- good for dynamic DB.

Disadv
- maintaining data bucket is complex

Transaction

- Properties and states
- Concurrent execution
- Serializability
- concurrency control
  - Lock-Based protocols
  - 2 phase locking protocol
  - Graph based protocol
  - Time stamp based protocols
  - Deadlock handling.

Transaction :

- set of logical work done on data of database is called **transaction** :
- Work can be - inserting
  - updating
  - deleting

  } data from current db.
- Three steps in DB transaction
  - Read data
  - Write data
  - Commit

Transaction Property : maintains consistency in db before and after transaction.

ACID

- **Atomicity** - either all successful or none
- **Consistency** - Brining the db from one to another consistent state.
- **Isolation** - Ensures that transaction is isolated from other transaction.
- **Durability** - means once a transaction has been committed, it will remain so.

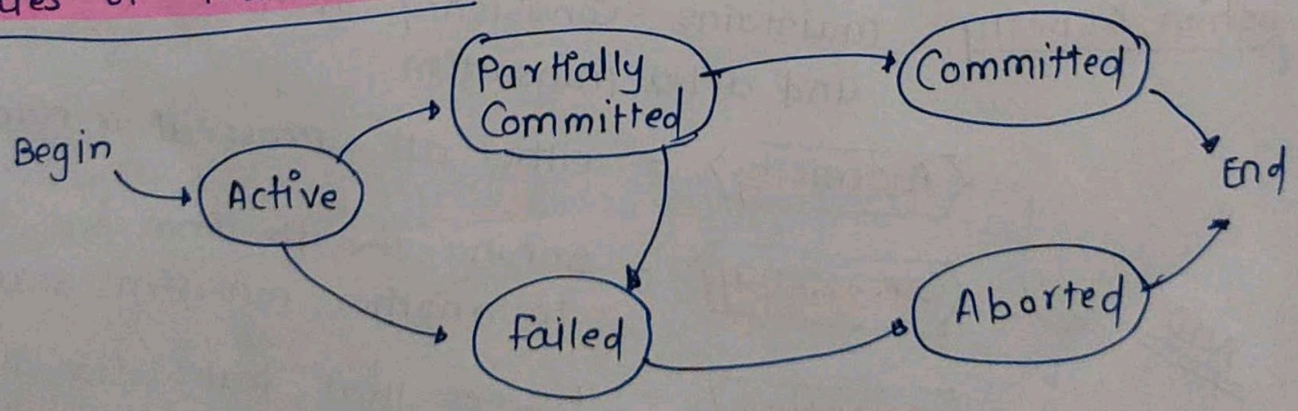
Atomicity :- all operations of transaction take place at once if not, then transaction is aborted.
- transaction can't occur partially.
- Involves
  ① abort : If abort, then all changes made are not visible.
  ② Commit : If commits, then all changes made are visible.

Consistency :- DB should be consistent before & after transaction.
- Execution of transaction will leave db in either its prior stable state or a new stable state.

Isolation : - data used at the time of execution of transaction can't be used by 2nd transaction until first one completed.

Durability : - states, transaction made the permanent changes.
- The changes can't be lost.

States of transactions :



Active State : - First state
- In this state, transaction is being executed

Partially committed : - transaction executes its final operation, but the data is still not saved to db.

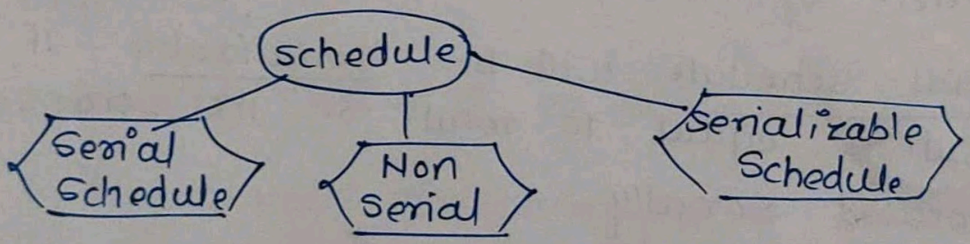
Committed : if all operations of transaction are executed successfully.

failed state : If any of checks made by database recovery system fails, then transaction is in failed state.

Aborted : - when transaction fails db recovery system will make sure that the db is in previous consistent state.
- If not then it will abort or roll back transaction to bring the db into consistent state.
- After abort :
  i) Restart the transaction.
  ii) kill the transaction.

Schedule :

- Series of operation from one transaction to another is known as Schedule.



(1) Serial schedule:

- Here one transaction is executed completely before starting another transaction starts.
- One schedule is followed by other.

T_1	T_2
Read(x)	
write(x)	
read(y)	
write(y)	
	read(x)
	read(y)
	write(y)

T_1 followed by T_2

(2) Non-serial Schedule :

- If interleaving of operation is allowed, then there will be non-serial schedule.

eg

T_1	T_2
redd(A)	
	read(A)
write(A)	
read(B)	
	write(A)
write(B)	

chapter VI
महत्वपूर्ण

(3) Serializable Schedule :

- Used to find non-serial Schedules that allow transaction to execute concurrently without interfering with one another
- Identifies which schedules are correct when executions of transaction have interleaving their operations.

* Non-serial schedule will be Serializable if its result is equal to result of its transactions executed serially.

Serializability :

- db consistency is preserved by serial execution of set of transaction.
- Schedule is serializable if it is equivalent to serial schedule.

(1) Conflict serializability :-

Schedule is called conflict serializability if after swapping of non-conflicting operations, it can transform into serial schedule.

Conflicting actions

i) → IF both action belongs to separate transaction.
ii) → They have same data item.
iii) → Contain at least one write operation/ action

Conflict equivalent :

Two schedules are said to be conflict equivalent iff

① They contain the same set of transaction
② IF each pair of conflict operations are ordered in same way.

* Eg. Non serial schedule

T_1	T_2
Read(A)	
Write(A)	
	Read(A)
	Write(A)
Read(B)	
Write(B)	
	Read(B)
	Write(B)

S_1 →

Serial schedule

T_1	T_2
Read(A)	
Write(A)	
Read(B)	
Write(B)	
	Read(A)
	Write(A)
	Read(B)
	Write(B)

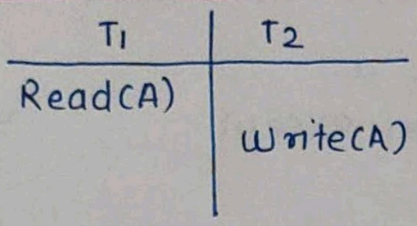
∴ S_1 is conflict equivalent / Serializable

② View serializability :

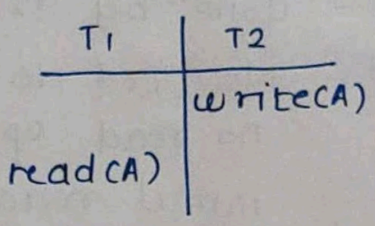
- It is view serializable, if it is view equivalent to serial schedule.
- If a schedule is conflict serializable, then it will be view serializable.

conditions for view equivalent :

1] Initial Read :



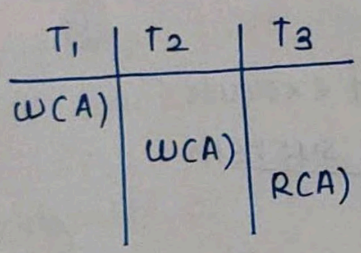
S1



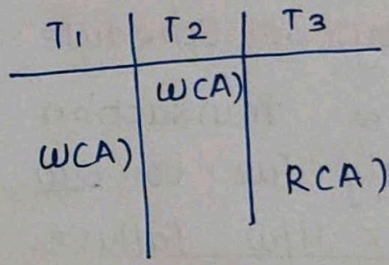
S2

- Initial read of both schedules must be the same.

2] Update Read :



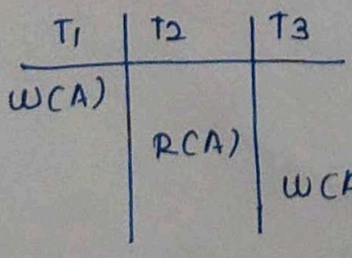
S1



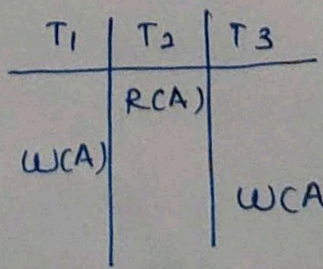
S2

∴ it is not view equivalent because in S1 T3 is reading A updated by T2
and in S2 T3 is reading A updated by T1

3] final write :



S1



S2

∴ Here final write must be done by same transactio

Example

T_1	T_2	T_3
R(A)		
	W(A)	
W(A)		
		W(A)

S_1

T_1	T_2	T_3
R(A)		
W(A)		
	W(A)	
		W(A)

S_2

① Initial Read - done by T_1 in both schedules ✓

② Update Read - No need to check as no read operation except initial read ✓

③ Final write - final write done by T_3 in both schedules ✓

∴ as three conditions statisfied S_1 and S_2 are view equivalent

→ Recoverability of Schedule:

- Sometimes transaction may not execute completely due to s/w issue, system crash or H/w failure.
- In that case, failed transaction has to be rollback.
- some other transaction may also have used value produced by failed transaction. So rollback those transactions too.

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
	Commit
Failed	
Commit	

Here,
- T_1 reads & writes the value of A and that value is read and write by T_2.
- T_2 commits but, T_1 fails lateron.
- due to fallure we have to rollback T1 and T2 (But T_2 can't be rollback as it is committed already)

∴ It is **irrecoverable schedule**.

Irrecoverable Schedule

→ Schedule will be irrecoverable if T_j reads updated value of T_i. And T_j committed before T_i commit.

Recoverable with cascading rollback :

→ schedule will be recoverable with cascading roll back if T_j reads the updated value of T_i.
Commit of T_j is delayed till commit of T_i.

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
Failed	
Commit	
	commit

→ Here, we have to rollback T1.
→ T_2 should be rollback becaz T_2 has read value written by T1.
→ Here roll back of T_2 is possible becaz T_2 is not committed before T_1

∴ It is recoverable with cascade rollback.

Cascadeless recoverable schedule

T_1	T_2
R(A)	
W(A)	
commit	
	R(A)
	W(A)
	commit

Here A is read and write by T_1 and committed.
Similarly after commit of A T_2 reads and write A

∴ Here cascadeless recoverable schedule

* Concurrency :

- multiple transaction at a time
- Adv : - waiting time is less
  - Response time is high
  - Resource utilization is more
  - Efficiency
- disAdv
  - Inconsistent system
  - Difficult to manage

- Problems on concurrent transaction
  1. Dirty Read problem : (W-R problem)
  2. Read - Write conflict : (phantom Read problem)
  3. Write - Write problem : (Blind Write)

* write - Read problem (dirty Read)

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
fail()	

i.e transaction reads the data written by other transaction before committing.

* Read - Write problem

T_1	T_2	
R(A)		Transaction T2 is writing data which is previously read by T1.
	R(A)	
	W(A)	
	commit	
R(A)		
W(A)		
↳ fail()		

* Write - Write problem (Blind write)

T_1	T_2	
W(A)		T2 writes data which is already written by T1. ∴ T2 overwrites the data written by T1.
	W(B)	
W(B)		
comm.		
	W(A)	
	Commit	

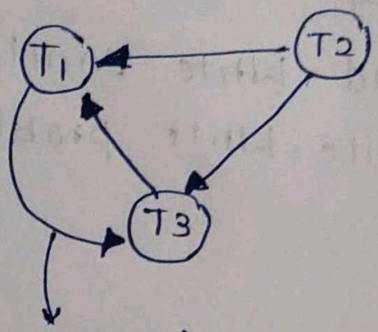
Conflict serializable by precedence graph

- check conflict pairs in other transactions and draw edges from that transaction to current
- If cycle formed then not seriable not possible
  If no cycle —11— Seriable possible.

↳ Now to print order of execution
→ Start with vertex with no incoming edge

EX 1

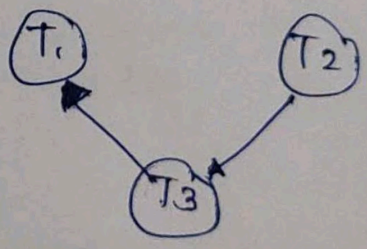
T_1	T_2	T_3
R(x)		
		R(z)
		W(z)
	R(Y)	
R(Y)		
	W(Y)	
		W(x)
	W(z)	
W(x)		



cycle is formed
∴ can't be serializable

EX 2

T_1	T_2	T_3
R(x)		
	R(Y)	
		R(Y)
	W(Y)	
W(x)		
		W(x)
	R(y)	
	W(x)	



Here no cycle
∴ can be serializable.

→ Order
$T_2 - T_3 - T_1$

∴ solution

T_1	T_2	T_3
	R(Y)	
	W(Y)	
	R(Y)	
	W(x)	
		R(Y)
		W(x)
R(x)		
W(x)		