Date: 01/09/2021 TOPIC: DBMS. GFG Placement 100 course.

DATABASE Management System.

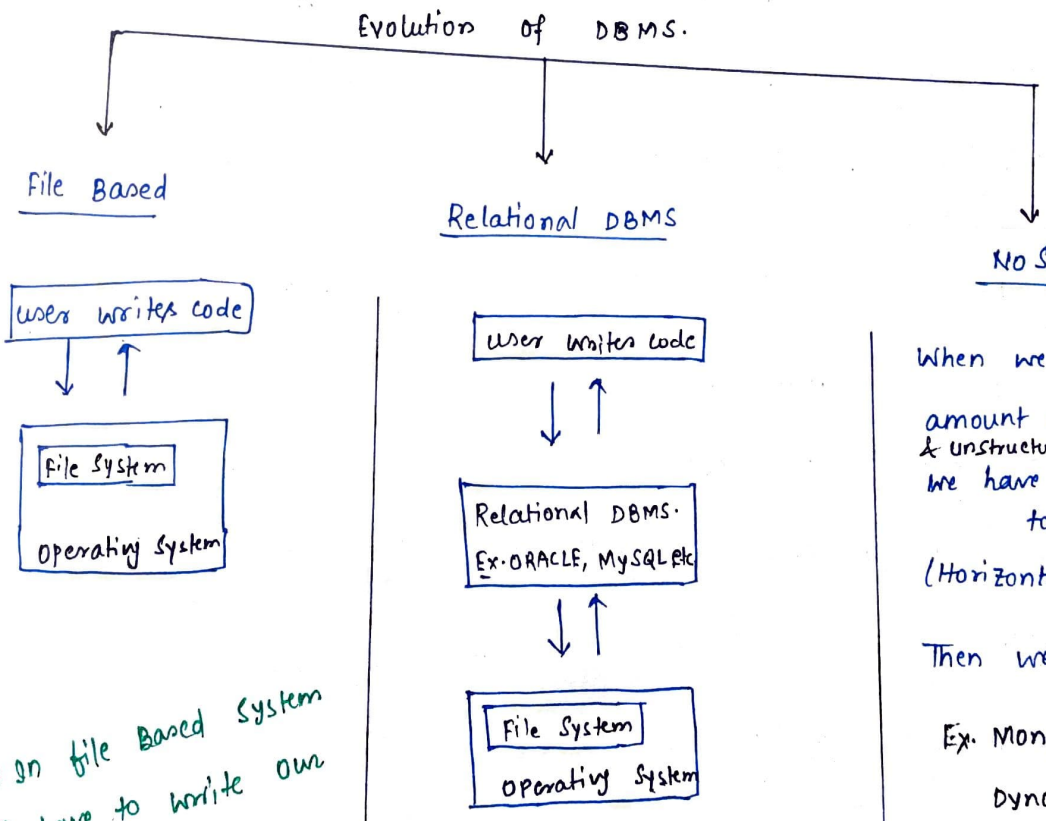
01. DBMS: Introduction.

Most of the websites like Amazon, IRCTC, and many more use database in their backend to store information.

* Database Management system is a collection of software that provides you the quick way to access and modify the data.

Ex. Oracle, MySql, Microsoft sql server and many more.

02. Evolution of DBMS.



* In file Based System we have to write our own DBMS code to access the data.
But via SQL. we don't need to worry just we have to write the Query and it will give us the data.

Uses: Before 1960.

* Relational DBMS provides
  - SQL (Structured Query Lang.).
  - It provides security, concurrency, and many more that is not provided in file Based System.

Uses: 1960 to till now.
(most used and famous DATABases).
i.e. Relational DBMS.

When we have large amount of data and & unstructured data. then we have ~~the~~ scalability to maintain (Horizontal Scalability).

Then we use NoSQL.

Ex. MongoDB,
DynomoDB by Amazon

Uses: Recent Days but not too much.

03. Entity Relationship Model.

Relational DBMS: A Relational DBMS stores data in the form table. that is how we implement Databases.

But before implemention of Databases we must first design the Databases.

* In Databases we have ER (Entity Relationship) Model to Design the Databases.

ER Model.

The ER Model contains 3 things. mainly:

1) Enitity Set.	: Basically Noun.	: Ex: Student, Teacher, ...
2) Relationship Set	: Basically Verb.	: Ex: teaches, gives, join...
3) Attributes.	: Attributes of Entity & Relationship	: Ex: Students ... course joining ... etc.

1) Entity set:

These are basically Nouns.

Ex. (Teacher) teaches a subject.

Here teacher is a noun. so. it is Entity Set.

2) Relationship set:

These are basically verb.

Ex. Teacher (teaches) a subject.

Here teaches is a verb so. it is Relationship Set.

3) Attributes:

Attributes are those who are attributes of Entity and Relationship

Ex: A student's phone no, Address, etc.
course's joining date, ending date etc.] → These are Attributes

04. ER Diagrams:

ER Diagrams are those diagrams which represents the ER Model of the database.

1) Entity Set: It is defined by  or 

Rectangle.

Double Rectangle for weak Entity set.

2) Relationship Set: It is defined by  or 

Diamond.

Double Diamond for weak Relationship set.

ATTRIBUTES

3) Attributes: There are different types of Attributes.

i) Normal Attributes: These Attributes contains single field.

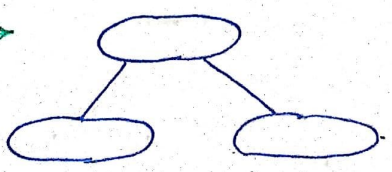
Ex. ~~Depart~~ Department. (Single value).

defined by → 

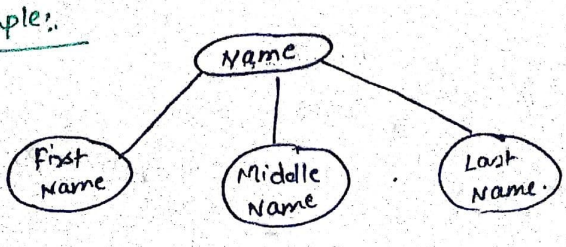
Ex :- Department

ii) Composite Attributes: These Attributes contains multiple field.

Ex. Name → (first name, middle name, Last name).
Address → (Street No, city, District, State, country).

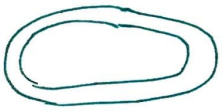
defined by → 

Example:



iii) Multiple Valued Attributes : Attributes those have multiple values for their entity. Ex. An entity may have multiple phone no. multiple Address and many more.

Represented by : Double ellipse,



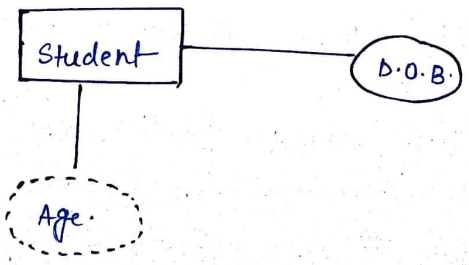
Ex:

iv) Derived Attributes : Attributes those can be derived from a given attributes.

Ex. If a attribute is given Date of Birth of an entity then its age can be derived from that Date of birth attributes.

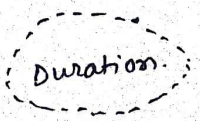
Represented by. Dotted ellipse.

Ex:



Other example: If a course have its Begin date and end as an attributes then we can derive its duration of course.

So, Duration is the derived attributes

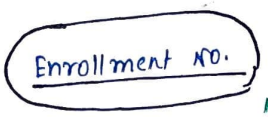


v) Key Attributes: Key attributes are those attributes that defines a particular entity set.

Ex. A student can be defined by it's Enrollment no.

Represented by: Ecllipse with underline.

Example:



↖ This is the key Attributes of a student.

---

05. Relationship sets.

1) Degree of Relationship sets.

i) unary

ii) Binary

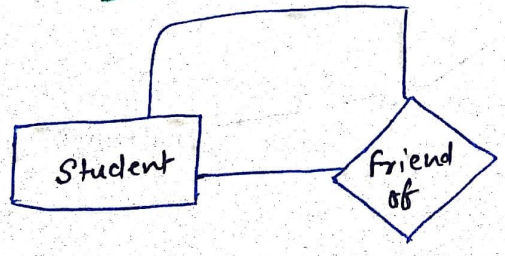
iii) n-Ary

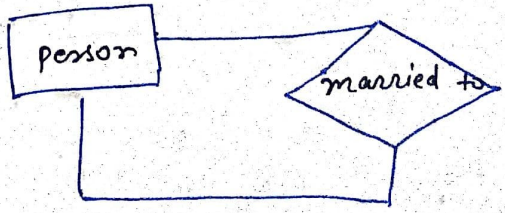
i) Uniary Relationship set: One Entity set related to itself. (Not very common).

Ex. student is a friend of itself.

Ex. A person is marry to a person.

Represented by.





ii) Binary Relationship Sets: (Very common).

These are the (Most common) Relationship Set.

An entity is related with other entity.

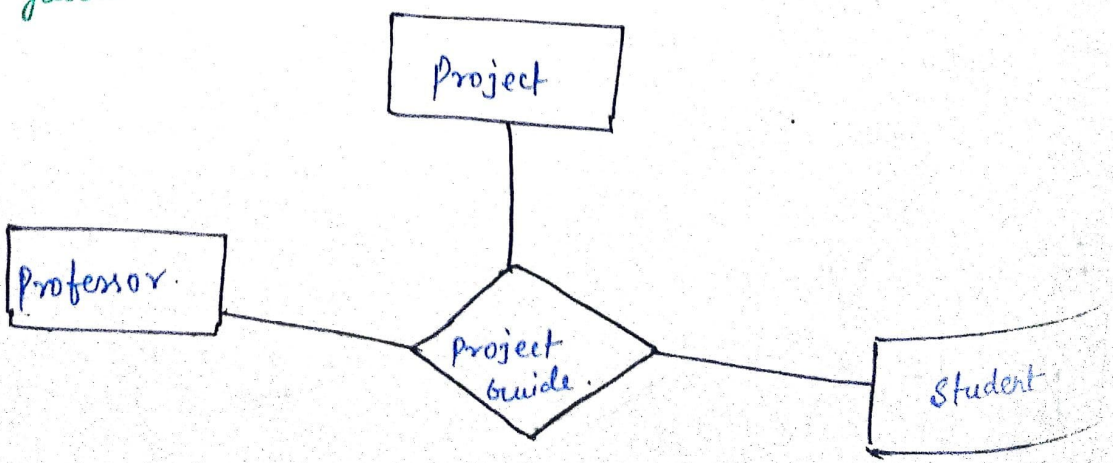
Ex. A student attends the course.

Ex: Supplier supplies items.

iii) n- ary Relationship sets: (Not very common).

Where two or more entity are participating in a relation are called n- ary Relationship sets.

Ex: Proffesor, project and student are connected with Project guide.



2). cardinality.

cardinality: How many entities of one side participates in a relationship.

These are of 3 types.

1) One to one
2) One to many (or many to one).
3) many to many.

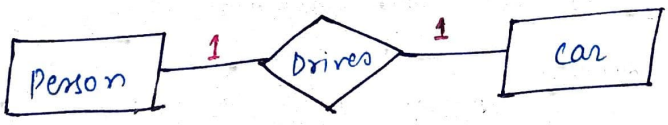
i) **one to one:** When only one entity of each side participates

ii) **one to many:** When one entity of one side participates an many entities of other side participates.

iii) **Many to many:** When many entities of both side participate in the relationship.

Examples of one to one Relationship.

Ex. A person drives a car.

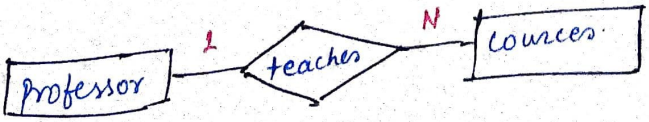


Ex. A person has Addhar card

Or, one Aadhar card is connected with one person

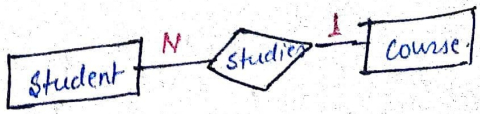
Examples of one to many Relationship.

Ex. 1 professor teaches N courses.



(1 to many)

Ex. N Student studies 1 course



(Many to one).

Both are same.

Example of Many to Many Relationship.

Ex. A student may studies many subjects.

Or, A subject may be studied by many students.

Ex. A customer orders many products.

Or A product is ordered by many customers.

---

06. ER Participation and weak Entity set

ER Participation.

Two types of participation are there.

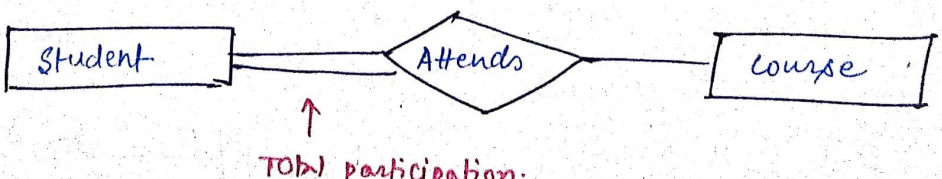
i) Total participation
ii) partial participation.

i) Total participation. Every entity of one side participates in relationship set.

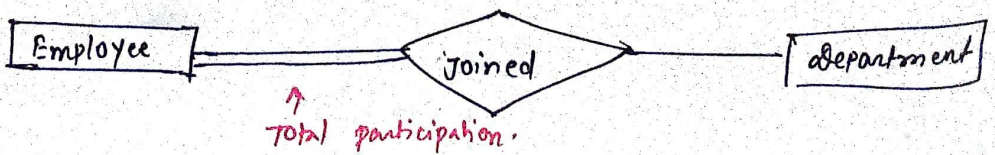
shown using ═══ (double)

Ex: We are assuming that,

A student must attainds one course.



Ex. Every Employee must be joined in a Department.



Weak Entity Sets.

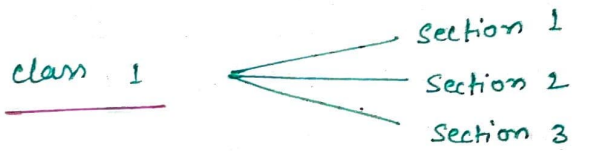
* Do not have their own key. are called weak entity and such sets are called weak Entity sets. Represented by 

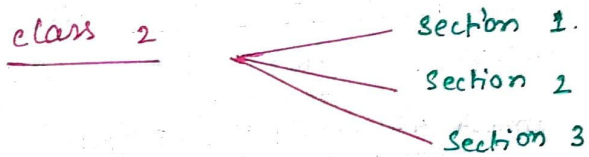
* weak entity sets always have total participation.

Ex. In a class their are different sections.

Every section is defined by a class (unique).

But in every class a section can't defined a particular class.



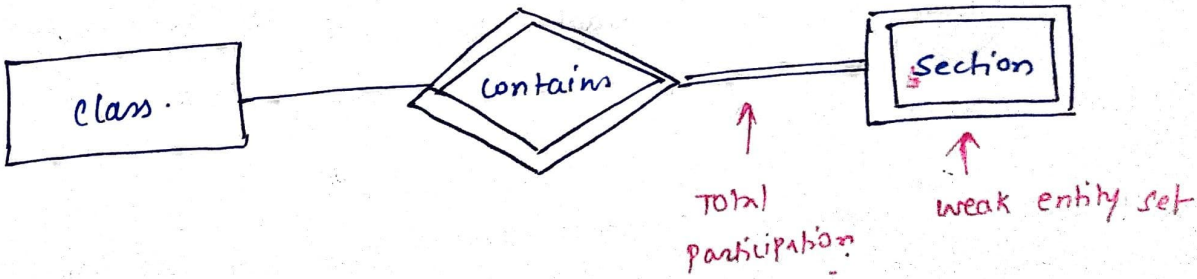


i.e. If we called only section-2.

class 2's section 2 then its a meaning.

But if we call section-2. Then it has no meaning

Which class's ??



Ex. Employee has dependents.

if the employee exits then only its dependents are valid otherwise - dependents are of no use.

Ex. hosts contains log ins.

Login exits only if a host exits.

Date: 02/09/2021 DBMS. 07. Keys.

Written by Abhishek Sharma. from GFG Placement course.

In this, we will talk about different keys on databases. There are various keys that are used in Databases. such as: candidate key, super key and primary key.

candidate key: A candidate key is the minimal set of attributes that derives all other attributes.

Example.

Enroll. No.	Student Name	AAdhar No.	Address	pan. no.	contact.
101	Ram	101---9	ABC	AEJ0---9	62---- 09
102	Ramesh	201---8	BCD	BJK--- 8	62 --- 94
103	Suresh	501---6	EFG	CKS---6	98--- 50
104	Sunil	809---1	GHI	MIK----5	93---- 62
105	Mohan	725---6	IJKL	KLM---4	95----- 68
106	Kamlesh	421---5	MNOS	MNO----6	63----95
107	Ritesh	015---6	PQR	STV----5	98---96

* Name is not a candidate key bcz. Another person with the same name may be there.
* Aadhar No. is a candidate key. bcz. it is unique
* Enroll No. is a " " " " "
* Address is not a cantlidate key. bcz two students they might be brother with the same address.
* Pan no. is a candidate key. (Assuming it is unique).
* Contact may or may not be candidate key. bcz. it is dependent upon business logic. If it is allowed to give phone no. only once then it is a candidate key otherwise not.

So, Enrollment NO, Addhar NO, Pan no. → candidate Key.

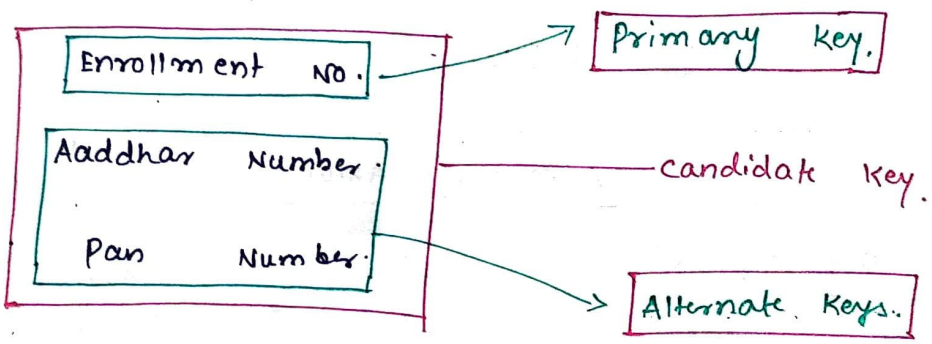
We only pick one out of above candidate key that is called Primary key. (chosen by the Database Designer).

Ex. Enrollment NO. → Primary Key.

Primary Key: The one key that is chosen from multiple candidate key is called Primary Key.

and other keys are called alternate Keys.

Ex.



** candidate key must be distinct and not null. (unique).

Properties ↗

Super key: : Greater than or equal to Super key.

↳ Any set of Attributes that uniquely identifies all the rows is Super key.

Ex. (Enrollment NO, Student Name) :- ↘ This is a key but This can not be a candidate key bcz. candidate Key must be minimal.

It is a Super Key.

* If we minimise Super key, it we remove unnecessary Attributes out of it then it becomes the candidate key.

* Example to find Keys (candidate Key, Super Key, Primary Key)

Ex: 1) R_1 (A, B, C, D)

(A → B means A derives B)

A → B , A → C   C → D

Here A derives B, A derives C and C derives D. and A derives itself.

$A^+ = \{ A\ B\ C\ D \}$ ⇝ clouser of a candidate Key.

That means 'A' is the candidate Key. and A, AB, ABC, AC, and many more are super key.

The Rule that Derives A → C and C → D that means A → D is called Armstrong Axions.

Armstrong Axions.

1) Reflexivity : A Derives A (itself).

2) Transitivity : A → C and C → D then A → D.

3) Augmentation : It says :- if X → Y then XZ → YZ.

These are The Three Armstrong Axions.

using these Three Basic Rules we can find colouser of a candidate key.

Example : 2. R_2 (A, B, C, D)

$AB \rightarrow CD$

Here, AB Derives CD so AB is the candidate key.

Example : 3. R_3 (A, B, C, D)

$B \rightarrow AC$, $C \rightarrow D$

using Reflexivity Rule we can say that B derives A, B, C and using tranivity we can say B derives C. and C derives D so. B derives D.

$B^+ = \{A\ B\ C\ D\}$

So, B is the candidate key.

for super key, we can add anything in B like..

BA, B, BAC, BAD, BD and many more are super keys.

Super keys : $2^n - 1$

$2^4 - 1 = 15$ ↑ in the above 3 example.

* Candidate Keys and super keys does not having NULL value.

that means: Every Table does have atleast one super key Which means atleast one candidate keys.

So, you always have one Super key, one candidate key and one primary key as well.

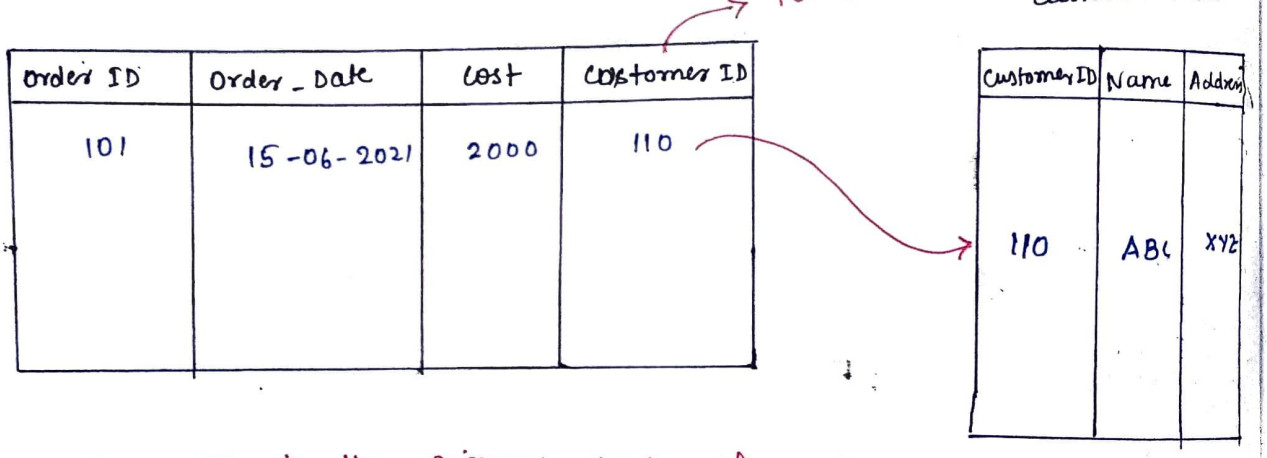
08. Foreign Keys.

Foreign keys refers to primary key of some other table. It may refers primary key of same table. but it most of the time refering to some other table.

It doesn't have to be primary key all time, it might be unique key as well. but we prefer to take primary key.

Example of Foreign keys.

Order Table of a E-commerse website.



* Customer ID is the primary key of order table. So we will took it and refrencing to another table that represents the customer table of that same customer ID.
* This is the function of ~~Primary~~ Foreign Key.
* In code the Example s will be. like this

```
Order Table    Order (
    Order-id    INTEGER Primary key,
    order-Date    Date,
    cost    Integer,
    Customer-ID    REFERENCES    Customer (Customer ID))
```

// ~~Primary~~ Foreign key //

* If we want to delete that information from the customer table (110) then we will use "ON DELETE CASCADE"

their are many ↓ methods that are provided by databases. like default, NULL and many more more.

and this integrity which is maintained by databases (to delete the data if the customer wants) is called Referential Integrity.

---

09. DATABASE Normalisation.

We do Database Normalisation to ensure that data Redundancy is minimum (↓) and Data Integrity is maximum (↑)

DATA
Redundancy : having the same data in multiple places specially string type of data not ID type of DATA.

DATA Integrity : Insuring that their is no arrouneous data inside your Database. , maintainers of data, follows the Busines Rule.

Data Base Normalisation.

Data Redundancy (↓). Low

Data Integrity (↑). high.

* Objectives of Good Database Design.

* No updatation, insertion, deletion anomalies.
  ↳ नियमहिनता (Doesn't follow the rules)
* Early Extendible.
* Good Performence for all query sets.
* More Informative.

1) Updatation Annomaly

If we want to update something in our database table. Updatation Anamoly occures when we have Redundancy in our database.

How we will remove these ??

Ans:- we will split the database table in two or more tables.

Example: 1

Student ID	Student Name	Subject ID	Subject Name.
1001	ABC	CS101	DataBase Management Syst
1002	MNO	CS101	DataBase Management Sys
1001	PQR	CS102	Operating System
1003	ABC	CS102	Operating System
1002	XYZ	CS102	DataBase Managmnt System

If we want to update Here DBMS.

So we have to work more upon the all Query.

Feasible way: Split the table into 2 table and update.

ii) Insertion Anomaly.

We can't insert only student name or student ID if there is already said that you have to fill all the column. This is called insertion Anomaly.

Example. 2

Student ID	Student Name	Stubject ID	Subject Name.
1004	JKL	??	??

We have to fill all the columns. This is called. Insertion Anomaly.

iii) Deletion Anomaly.

If we delete some record but the deleted record which was required to be their even after deletion then it is Deletion Anomaly.

Example: 3

Suppose student 1001 and 1002 Leaves the college and we Deleted both the entries from our Database.

(Example 1)

So, the subject Database Management System also deleted from our record. But we don't want to delete it

This is called Deletion Anomaly.

All these Anomaly which we learnt happens in our Databases bcz. we have Data Redundancy. for get rid of them we need to make seperate tables.

Keep it in mind :- Breaking into tables in more tables we also have caused problems. bcz. if we do this we have many many tables and which is not easy to handle.
⟹ So break the tables but not in too many tables.

10. Functional Dependency.

We studies functional dependency for normalisation of our Databases.

Q) Why we study normalisation ?

Ans: TO Avoid Redundandary in our Databases.

Q) Why we avoid Redundandary in Databases ?

Ans: TO Avoid Anomalies in our Databases like insertion, deletion, updation Anomalies.

Now we will see Functional Dependency.

$$A \longrightarrow B$$

i.e. B is functionally dependent on A.

Example. 1

Enroll No.	Name	Address	phone number

Enroll No. $\longrightarrow$ Name.

Enroll No. $\longrightarrow$ Address

Enroll No. $\longrightarrow$ phone No.

i.e. Name, Address and phone no. is dependent on Enroll no.

Or Enroll No. uniquelly defines Name, Address and phone no.

Enroll No is the Functional Dependency Here.

Example: 2

Subject ID	Student ID	Marks	Grade.

Subject ID, Student ID → Marks.

Subject ID, Student ID → Grade.

Or we can write : Subject ID, Student ID → Marks, Grade.

Only subject ID or Student ID can't define marks bcz.

A student have more than one marks he can be inrolled in many subjects. that why we have to take both.

Here. Subject ID & Student ID is functional Dependency.

Ex: 3

A	B
x	1
y	1
z	2

Here:

A → B ✓✓ (True).

B → A ✗✗ (Not True)

Bcz. if we say '1' it may be x or y.

Functional Dependency.

For a given attribute, if you can uniquely define the other attributes then there is a functional dependency.

Example: 4

Enroll No.	Name	Address
101	Raj	ABC
102	Ravi	ABC
103	Raj	XYZ

Enroll No → Name, Address. ✓ (True)

Name → Address ✗ (False)

Address → Name. ✗ (False)

Q) Why do we study Functional Dependency ??

Soln If we are given a database like this

Student ID	Name	Department ID	Depart. Name.
101	ABC	10	CS
102	BCD	11	ECE
103	ABC	10	CS
104	XYZ	11	ECE
105	CDE	10	CS.

Now from the given above table first we will find the functional dependency. i.e.

Depart. ID ⟶ Depart. Name.

Now we will delete that column (Department Name) (X) and create a seperate table. and link with that table.

Depart - ID	Depart. Name
10	CS
11	ECE

That's How we reduce Avoid. Data Redundancy.

Two Types.

Functional Dependency.

Trivial

Ex. AB → A

A → A

ABC → AC

Attributes that defines itself.

Non Trivial.

Ex. A → B

AB → C

BC → DEA.

Attributes that gives some info

1. First Normal Form (1NF)

Abhishek Sharma Notes.

Rule: Every Attribute should contain only single value (Atomic).

Rule: Every Attribute should contain only single value. i.e. Atomic

Example:

Customer ID	Name	Mobile No.
101	ABC	6290903490, 6291005 70
102	BCD	9598462 7,
103	XYZ	9586756 7,
104	PQR	9354 32 01.

This table is not in INF bcz. as the defⁿ says every Attribute should contain only single value, but the Attribute Mobile No. containing 2 values.

⇓ Conversing the table in 1NF.

1st possible way.

We can add a new column for mobile No. 2. like this

customer ID	Name	Mobile No. 1.	Mobile No. 2
101	ABC	6290903490	6291 00570
102	BCD	9598 46 27	—
103	XYZ.	9586 7526	—
104	PQR	93543201.	—

Theoritically it is in 1NF but the Problem with this table is that there are too many Empty fields.

⇓ Now we will further optimise this table for better 1NF.

Customer ID	Name.	Mob No.
101	ABC	6290903490
102	BCD	95984627
103	XYZ	9586 7526
104	PQR	9394 3201
101	ABC	6291 00570

* Now '101' has two entry. This is in 1NF and Better than previous one bcz. previously we are wasting too much empty spaces. We are repeating Here. in the same table.
This is Better 1NF than Previous one.

* But This table has its own demerit though it is in 1NF. We already know that customer ID should be unique but here it is repeating.

We will Remove this problem in 2NF and 3NF.

As of now this is in 1NF.

⇓

We can modify it via breaking it in two tables. like this

⇓ Better solution

Customer ID.	Name.
101	ABC
102	BCD
103	XYZ
104	PQR

ID.	Customer ID	Mobile No.
1	101	6290903490
2	102	95984627
3	⋮	⋮
⋮	⋮	⋮

* Now customer ID is unique.

Best Idea

We will see further in 2NF & 3NF.

Rule: i) No partial functional dependency.

(i.e. NO non-prime Attribute should depends on partial candidate key.)

Prime Attribute

ii) It should be in 1NF (1st Normal Form).

candidate key : Minimal key that defines all attributes.

Prime Attribute : Any Attribute which is part of any candidate key.

Non-Prime Attribute : Any Attribute which is not the part of any candidate key.

Example: Let's understand via examples.

User ID	Course ID	Course Fee.
1	CS101	5000
2	CS102	2000
1	CS102	2000
3	CS101	5000
4	CS102	2000
2	CS103	7000

Here, user ID alone is not a candidate key bcz. One user can took many courses.

So (user ID, course ID) = candidate key.

* course Fee is not ~~depen~~ in the candidate key so it is non-prime Attribute.

So, user ID, course ID = candidate key.

course Fee = Non-Prime Attribute.

Here, course fee → course ID (course fee depends upon course ID)

But as the defⁿ says no non prime Attribute should depend upon partial candidate key. So Here the 2NF Rule Breaking.

So this table is not in 2nd Normal Form (i.e. 2NF).

Q How will we convert the table in 2NF ??

A we will create another table. We will seperate the table in two parts. like this

user ID	Course ID
1	CS101
2	CS102
1	CS102
3	CS101
4	CS102
2	CS103

Course ID	Course Fee
CS 101	5000
CS 102	2000
CS 103	7000

In this setup we haven't store course Fee in multiple places like we previously did. So, this avoid Reduntancy and anomaly in our databases.

So, that's How we convert the table in 2nd NF. (2NF).

---

13. Thrid Normal Form (3NF)

Rule: i) NO non prime attribute should Depend upon another non prime Attribute.

ii) It should be in 2NF. (2nd Normal Form).

iii) It should be in 1NF (1st Normal Form).

* Let's see and understand this via help of Example

Student NO.	Student Name	Student State	Student Country.
101	Ram	Haryana	India.
102	Ramesh	punjab.	India.
103	Suresh	punjab.	India.

Now, First we will check that whether the table is in 2NF or not. and 1NF or NOT also.

* No Attribute contain multiple value, so it is in 1NF.

Student ID → Student name, Student state, Student country.

Student State → Student Country.

candidate key

Functional Dependency.

As the Def^n of 2NF says no non prime Attribute should depend upon partial candidate key.

Here Non Prime Attribute = Student name, Student state, Student country.

Here no partial candidate key, candidate key is only one (i.e. student ID) so, Here NO non prime Attribut should depends upon the partial candidate key.

So the following function follows 2NF.

Now we will check Rule No. 1. of 3NF i.e. no non prime Attribute should Depend upon another Non - Prime Attribute.

Here we saw,

Student state → student country.

i.e. Non prime Attribute depends upon another non prime Attributs. So, the 3NF Rule is Breaking.

How to convert the table in 3NF. ??

We Split the table in 2 parts.

table 1 : student ID, Student Name, Student state.

table 2: Student state, Student Country.

In General: If a table is not in 3NF we split the table in the following way so that if there is a non prime attributes defining another non prime Attributes.

then we took these two non prime Attribute together in a table and we make a seperate table and that non-prime Attribute which is defining the other non-prime Attribute should be kept in the original table also.

Ex. from our previous Example

Table t_1 : Student ID, student Name, Student State.

Table. t_2 : Student state, student country.

Example : 02.

Exam Name	Exam Year	Topper Name	Topper DOB.
ABC	2016	Abhishek	07-08-2000
BCD	2017.	Anurag	28-09-2000
EFG	2017	Sunil	03-05-2000
ABC	2017	Rahul.	25-07-2000

We have to make the table in 3NF.

Rule : 1. First check Whether it is in 1NF ??

Ans Yes the table is in 1NF bcz No attribute contain multiple values.

Rule : 2 : check whether it is in 2NF ??

Ans (Exam Name, Exam year) → Topper Name, Topper DOB.

(Exam Name, Exam year) ← candidate key.

TOPPER Name → TOPPER DOB.

2NF says it should not happen that a non prime Attribute is derived by part of a candidate key.

and from the above condition 2NF Rule is not violated.

So the table is in 2NF.

Rule : 3. Check for 3NF if it is yes then it's OK Otherwise make it in 3NF. (table).

TOPPER Name → TOPPER DOB.

Ans: Topper Name and TOPPER DOB is a non prime attribute and a non prime attribute depends On another non prime Attribute. So Here 3NF Rule is violated.

Q How do we fix it ??

Ans We will make two tables.

Table 1 will contain: Exam name, Exam Year, Topper Name.

Table 2 will contain: TOPPER Name, Topper DOB.

That's How we will fix the table in 3NF.

Abhishek Sharma
Notes

Alternative Definition of 3NF.

i) Should be in 2NF and 1NF.

ii) Non-Prime Attributes are not transitively dependent on Prime Attribute.

This means

from ~~Armo~~ Armstrom Axioms, Rule of Transitivity is that

> if $A \rightarrow B$ & $B \rightarrow C$
> Then $A \rightarrow C$

means if a ~~Pri~~ prime Attribute —defines a→ non prime Attribute, then Non prime Attribute —defines a→ non-prime Attribute is not allowed.

Same definition in another way,

Summarising Here All 3 Normal Forms.

1 NF: Any attribute should ~~Pos~~ contain only single value.

2 NF: Any Non Prime Attribute should not be a part of any ~~Pa~~ partial prime Attribute.

Not allowed. ✗ $P \rightarrow NP$ ✗ Not Allowed.

3NF: No non prime Attribute should not define any another non prime Attribute.

Not Allowed ✗ $NP \rightarrow NP$ ✗ Not allowed.

~~BCNF~~
BCNF ✗ $(P/NP) \rightarrow P$ ✗ Not Allowed (Study further).

14. BCNF (Boyce - codd Normal Form)

Rule: i) Both prime and Non-prime attributes they do not define a prime Attribute.

Not Allowed. ✗ | P / NP → P | ⇒ Not Allowed ✗.

ii) The table must follow 1NF, 2NF and 3NF. for BCNF.

According to BCNF

for any functional Dependency in your table

Let's suppose $x \rightarrow y$

conditions for BCNF.

i) Trivial OR.

ii) X is a Superkey.

(~~Left~~ hand side (only Superkeys on the left Hand side).

Let's understand through example.

Student ID	Subject	Proffesor ID.
1001	DBMS	103
1001	OS	110
1002	DBMS	111

Here, Student ID, subject → Proffesor ID.

Or Student ID, Proffesor → subject.

Proffesor ID → subject

In this example:

Candidate Key : i) student ID, Subject

ii) proffesor ID, student ID.

So, Here No non prime Attribute present.

So, 1NF Rule Followed.

So, 2nd NF Rule also followed

bcz. No non prime Attribute Here so we don't care further.

So 3rd NF Rule Also followed.

" " " " "

" "

Now we will check this table follows BCNF or not ??

No, It is not following BCNF bcz

BCNF says Prime or Non-prime attribute they do not define a prime Attribute.

But Here

Professor ID → Subject

↑ Prime ↑ Prime.

So, BCNF Fails.

How to convert the table in BCNF form. ??

We will split the table in 2 tables such that.

table 1 : Student ID , Professor ID.

table 2 : Proffesor ID, Subject.

Stud. ID	Proff. ID
1001	103
1001	110
1002	111

Proff. ID	Subject
103	DBMS
110	OS
111	DBMS.

Q) What is the Advantage of doing so ??

A) Suppose in the 1st table there are thousands of rows and Subject (string) written in thousand of time. So we have anomaly in Database.

But after BCNF we split the table in 2 table and Subject (String) type written only once.

Remember: BCNF decomposition (split into more table) might not be acceptable. bcz. it may loose functional dependency.

So decomposition of table to get the BCNF is not a good idea.

---

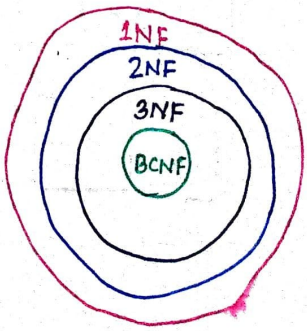
* BCNF is the most strict form

then 3NF then 2NF then 1NF

* Whatever in BCNF will also be in 3NF

Whatever in 3NF will also be in 2NF

Whatever in 2NF will also be in 1NF



15. Indexing in Database.

The whole idea of indexes is to improve performance of your Query.

Let's Learn Indexing through an example:-

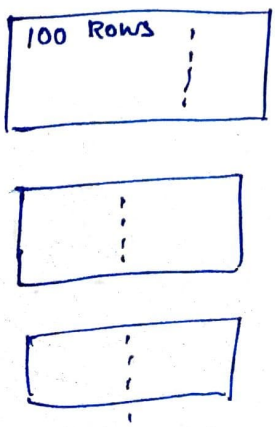
Suppose we have a E-commerse website Database. like this :—

Order ID	order date	cost	Customer ID.
101	15-06-19	2000	102
⋮	⋮	⋮	⋮

If we have minimum amount of order then we have a small database and we can handle easily but if we have a thousands of hundereds of orders then it will be really difficult to maintain it.

* All the data is saved in our Hard disk. like this in hard Disk we have blocks of rows. Like these are many blocks and each block can contain only 100 rows.

Disk block.



and so on.

If we search a particular customer ID. then we don't know where it is. So, we have to traverse through all the Disk block.

So Basic idea of indexing is to put an order in the data in your harddisk. and this particular indexing is called clustered indexing.

Clustered indexing :- To put order in your database storage in files.
↳ Data stored in increasing order in your disk block.
Examp: if I have to store data in the table and we have n orders then it will store like that.

The smallest order id will apper first like that.

Order ID.
101
102
103
⋮
L

This will give some Advantage. if I want to search any order id then we will do Binary search and find it.

Example.

Order ID	order date	Cost	customer ID.
101	15-06-19	2000	102
102	15-06-19	5000	103
105	15-06-19	6000	101
101	15-06-19	8000	102

↳ clustered indexing.

* If we don't do clustered index in our table then these all order. id can be stored in any order in table. and This type of organisation of your data is called Heap organisation. bcz. the tables are act like Heap.

Abhishek Bhaiya's Notes

* Most of the DATABASE serverses they Bydefault create clustered index on your primary key.

Prime Index: When primary key is used as an index

* prime index is also a clustered index.

* Remember we can create only one clustered index. bcz. there is only one way to ~~order~~ D. organise data in physical order.

Let's see one more example.

Order ID	order Dat	cost	customer ID.
101	15-06-2021	2000	102
102	15-06-2021	5000	103
105	15-06-2021	6000	101
106	15-06-2021	8000	102

This table is in clustered index. along order id.

But if we want to Arraye this table along customr id also then we can't do via clustered index.

Bcz. a table has only one culustered index.

So How can we arraye customr id table also.

This arraymment is called Non-Clustered indexing. Or Secondary indexing.

* The idea of clustered index and non clustered index is understood via example of Book.

Book Example.

We saw the book. in book

i) All the chapters are assigned in the increasing order

like : chapter 1, chapter 2 ----- chapter n.

This is clustered indexing.

ii) But If we want to index a particular item of the book or topic of the book. we found it in the end of the book with their respective page no.

This is secondary index or non - clustered index.

So, chapter in the book are example of clustered indexing. and particular topic that is present in the book's last page with their page no. is the example of secondary indexing or non - clustered indexing.

That's How indexing Helps us to find a particular item in the table.

Day-3 Notes ended.

Abhishek Sharma.

02/09/2021

Date: 04/09/2021 DBMS 16. clustered Index Abhishek Sharma Notes. By GFG placement course.

* Now we will understand How clustered index are implemented by DBMS servers:

DBMS server Maintain an Extra Index file.

That file has entries in this form:

Search_key	Pointer

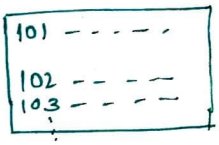
→ stores reference to disk block containing given key.

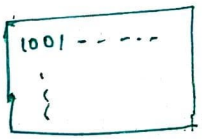
Example:

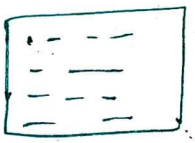
Table

Order ID	order Date	cost	CostumerID
101	– –	–	–
102	–	–	–
103	–	–	–
104	–	–	–
1001	–	–	–

Disk block







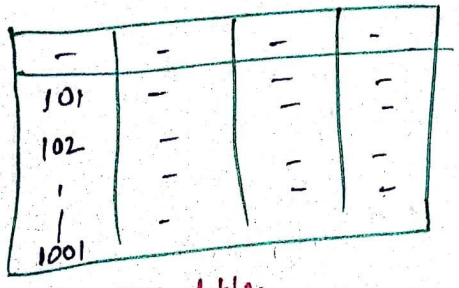
index file.

It contains search key along with the reference (Address of that block).

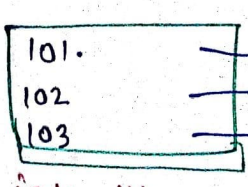
Let's say,

If we find Key 101

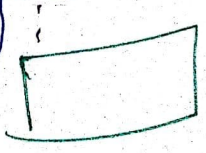
then it will come to the index file.



our table

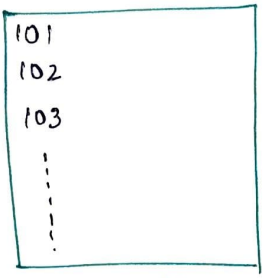


index file.



This is how we implement clustered key in our DBMS.

we store search key and we store disk block address where the search key items are present.



index file.

⇒ This index files are called Ordered index file. Or Sequencial index file organisation

↓

in this type of index file all the keys are present in Sequencial or ordered mamner.

like 101 - 102 - 103 then 104 and so on.

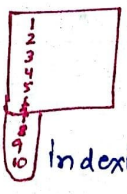
* Ordered index file or sequencial index file may be sparse or Dense.

Dense indexing: Every key has an entry in the index file and for every key value you have the disk Address.

So, index file might be big in dense index file.

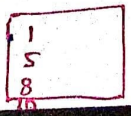
bcz every key value will be present.

Ex. (if table में order id में 1 से 10 तक Number है तो index file में भी 1 से 10 तक key होगी और उसका corresponding Address).



(Dense index file).

Sparse indexing: we can skip some Indexes in the index file like if we have 10 items (1 to 10) are in table then we can add in the index file like that



(Sparse index file)

Ex. (Only we can add 1, 5, 8 10)

In sparse indexing the index file is small as compared to Dense indexing.

bcz. in Dense indexing we have to add all the keys but in sparse indexing we can skip some keys (why we skip some key ??)

⇓

bcz all are in ~~sorted~~ increasing order.

if 1 comes then 2 will come after 1. that why we can skip some keys.

* Which is fast ?? sparse or Dense index ?? in searching a key

Ans :- Dense index is fast in searching a key bcz. All the keys are present there and we can search easily from there.

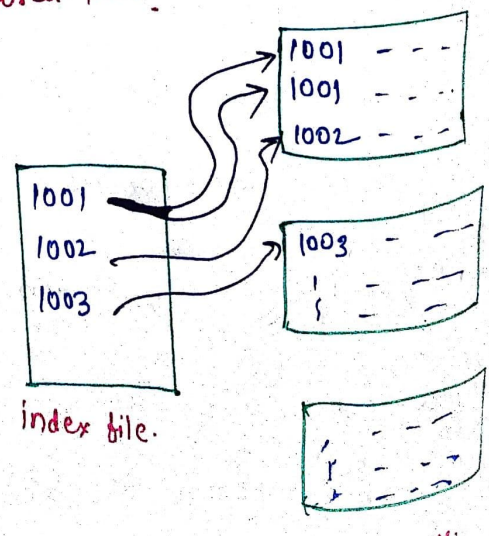
sparse index as it contain only some keys so it might be slow compare to Dense index.

Let's see an Example.

clustered index.

Order ID	Order Date	Cost	Customer ID
1	-	-	1001
2	-	-	1002
3	-	-	1003
4	-	-	1001
.	-	-	.
.	-	-	.
.	-	-	.

Our table.



Explanation: We want that customer ID as a clustered index

So, we found that there are one key is appearing more than one time (multiple time) "(1001)" So, in index file we will Right only once.

* in index file every entry has to be present exactly once. (in Dense file system).

But in disk block we will give the space for both the entry (1001) twice. how many times it may appear. it will be shown in disk block.

But in index file it will be only once.

17. Non clustered Index.

Q. How non-clustered index are implemented by DBMS Server ?

A. we will learn it via example.

clustered index →

Order ID	order date	cost	Consumer ID.
101	15-6-19	2000	1005
102	15-6-19	5000	1001
103	15-6-19	300.	1005
104	15-6-19	1000	1002
105	15-6-19	2000	1003
106	15-6-19	5000	1005
~~109~~			
⋮	⋮	⋮	⋮

→ Non clustered index.

we have a clustered Index. = Order ID.

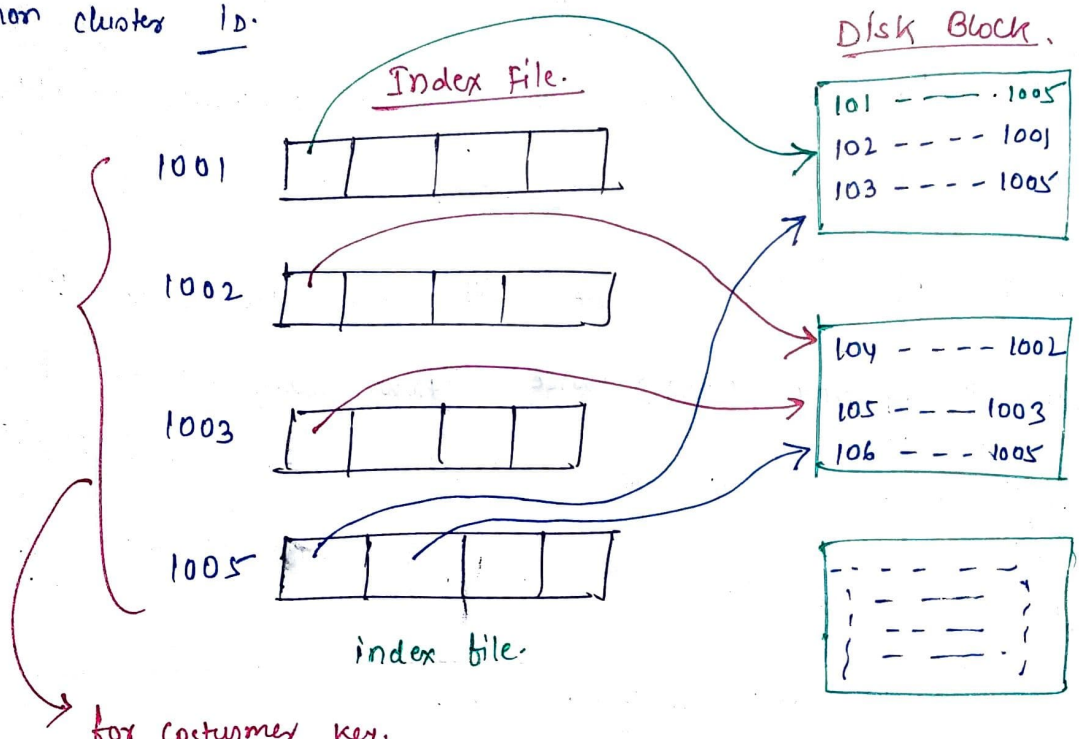
we " " non " " = Consumer ID.

We want to implement in our DBMS.

So DBMS server does the following task.

DBMS server makes an index file & like that for Disk block.

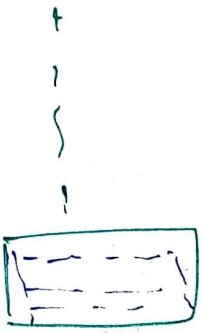
non cluster ID.



for costumer key.

it will be in increasing manner.

This is how secondary indexes are created for every Attribute.



Disk Block.

* non clustered indexing always be Dense.

bcz there have to present all the keys.

* They are not dense in implementation.

* we should not create many secondary indexes we ~~for~~ Should create for that attribute only who is most Searched Attribute.

18. Multi Level Indexing.

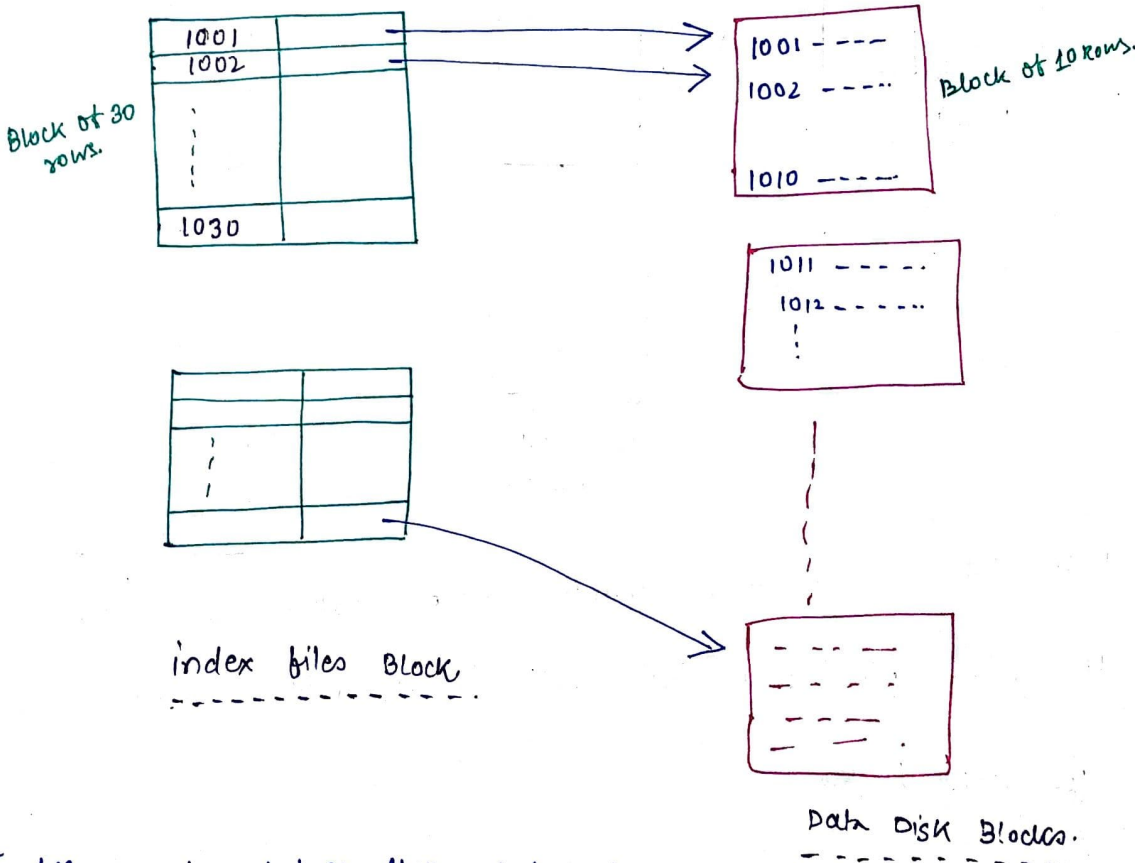
Abhishek sharma Notes.

Multilevel Indexing are used when our Database has very large number of Records.

As the time increases the databases ~~increase~~ grow with the time so the index files also grows with time.

So this index files may not fit in one single block. They might be spreading along multiple disk blocks.

Let's see an example.

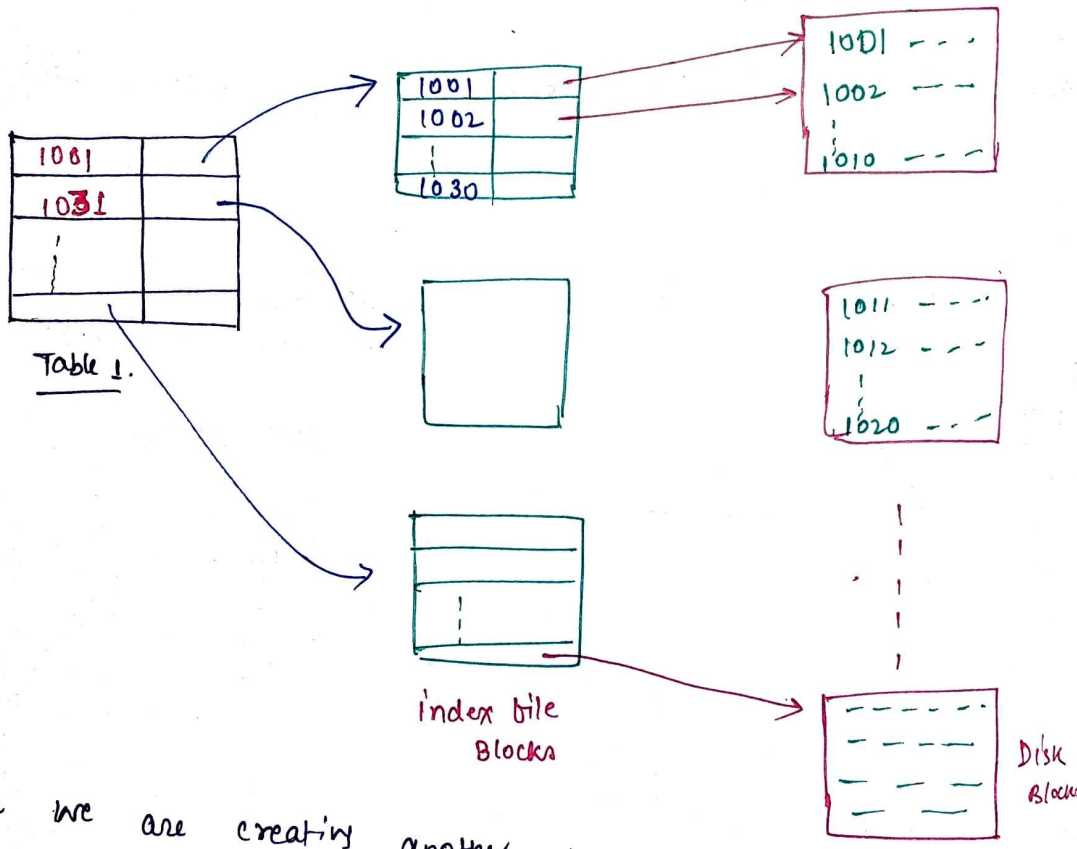


If we go in future then index files grow with time that the cause of Disk Blocks also grow with time in our databases.

So how can we handle this. If we search here any key in index file then it will take O(n) times. at max. (worst case). we have to search from the full index.

So, from get rid of this we will use Multilevel indexes. such that if we search a key then it will take minimum time.

we will Create Multilevel indexes like this:

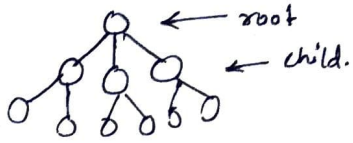


Here we are creating another table which maintain the index file. and index file blocks will maintain the disk blocks.

if we want to search for (1032) then we will find it in 2 comparsion of blocks. (so the time Reduces from O(n).

So, it is more like a tree structure if we rotate it.

The table which we created will act as root node then it's child (i.e. index files) and index files child are disk blocks.



Advantages of Multilevel indexing.

if we want to find last key then we have to go through only 2 blocks. that is Table 1 and index file's block. So we found the last key in minimum no. of blocks. But in earlier we should have to go till the last block in the index file. i.e. (O(n)) time earlier we should take in the worst case. So via using of Multilevel indexing we find the record in minimum no. of times.

* Problems in Multiplelevel indexing.

As per the time grows we need to modify our databases as these are huge no of records. if a person leaves the table then we should have to shrink the table also. So to get rid of these problems the idea of B tree and B^+ Tree. comes.

* How to get rid of problems in Multi level indexing.

using B and B^+ tree. (They modify themselves. grows and shrink when the data coming in and deleted)

These are the Balanced Binary Tree * like Data Structures.

Date: 05/09/2021 — 19. B and B+ Tree — DBMS Notes Written by Abhishek Sharma (GFG course)

Q) What are B and B+ Trees are ??

Ans :. They are n ways search Trees.

* Every Node of a Tree has some no. of values. and the no. of values that a node can contain is defined by **Branch factor**.

*

If Branch factor is b, then maximum b children are in the node and minimum $\lceil b/2 \rceil$ children.

Ex. if Branch factor = 4.

then minimum no. of children = $\lceil \frac{4}{2} \rceil$ = 2 children.

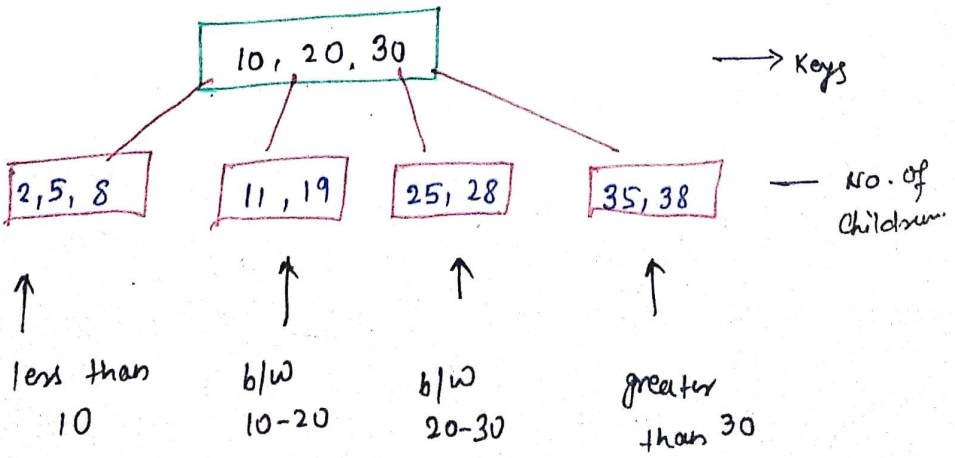
maximum no. of children = 4 children.

* No. of keys present in ~~a node~~ in n ways search Tree node = **(no. of children − 1)**

Example:

if a Trees has 4 children then No. of keys = (4−1) = 3

Ex.



* B and B^+ Trees are self Balanced Search Trees.

Ex. If There are millions of Record and millions of hight then they manage them by themselves.

* They make sure that hight never goes beyond $O(\log n)$ if there are n records to store.

Q) How B and B^+ Trees maintain their Height ??

Ans we will see via this example.

Given: 10, 30, 25, 80, 90, 100

15, 18, 20, 60, 110, 120.

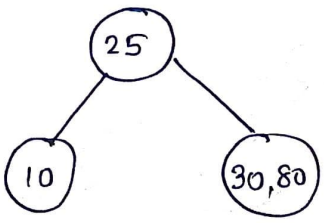
Branch factor = 4.

We have to insert these values in the Tree. Let's see.

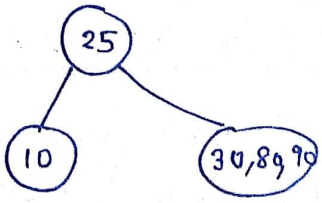
$\lceil b/2 \rceil$ minimum children formula is not applicable for root.

Now if we try to insert Here 80, then it violents the maximum children property. it has $b = 4$ children now.

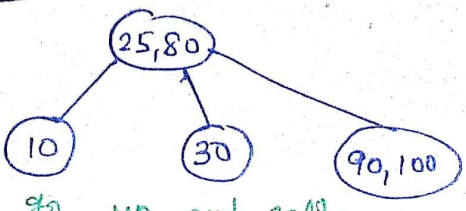
so we will split it.



Now we will insert 90

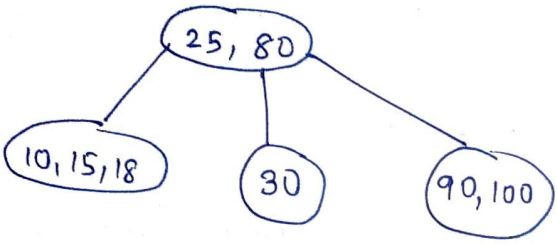


if we add 100 same problem



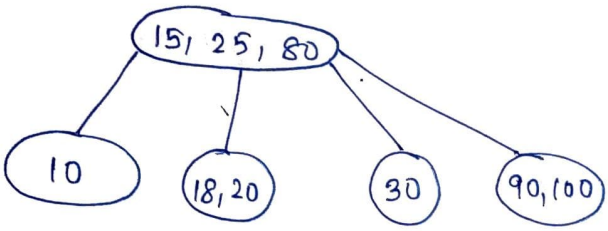
middle key will go up and split.

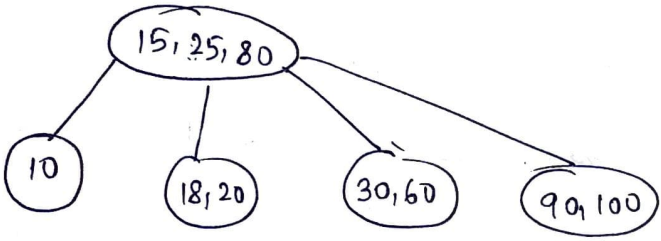
Now if we add 15, 18



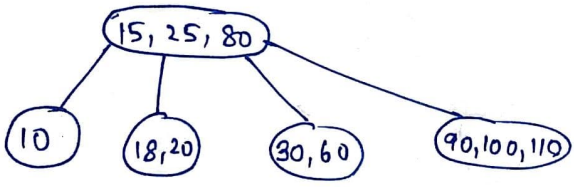
Now to add 20. and 60

// If we insert 20 then Rules violeted so we split it and 15 (middle key) go upward and splitting done.





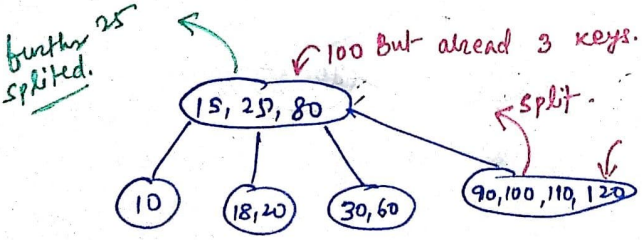
Now we have to add 110.



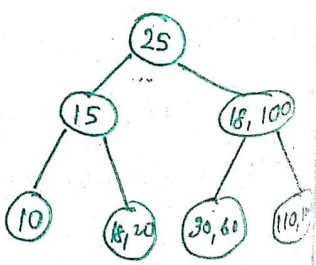
Now we have to add 120.

Mechanism · How we will add 120.

Finally Tree will be



Finally.



This is Our B Tree.

Abhishek Sharma Notes.

Note: B Trees Height grows upward. unlike the normal Binary Search Trees where Height grows downward.

in BST we ~~add~~ insert always from the leaf and that's why height always grows downwards.

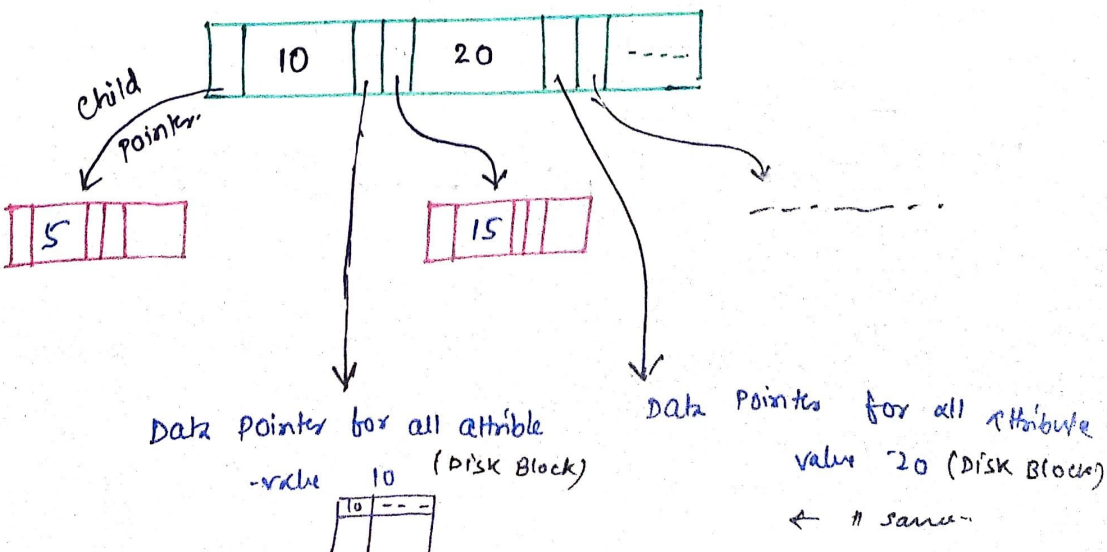
But Here in B and B+ Trees we always insert from root and Height growing from the root ↑

In the previous examples we saw that if the ~~Height~~ no. of keys increases Height grows upward.

* Similarly Deletion Operation, it (B and B+ Trees) shrinks the Height when no. of keys removed or deleted.

* B Trees are used To represent index files.
  - So They have pointers of actual records. actual ~~Records~~ of the disk where the records saves. and blocks
  - They also have pointers to children nodes.
  - So B Trees has 2 Pointers. i) Child pointer and ii) Data pointer for (Disk Block)

Q. How the B Trees store data. ??



Abhishek Sharma notes.

B+ Tree.

It is the improved version of B Tree.

It is the most used Tree in database for indexing like Oracle, Microsoft Sql Server. uses this (B+ Tree)

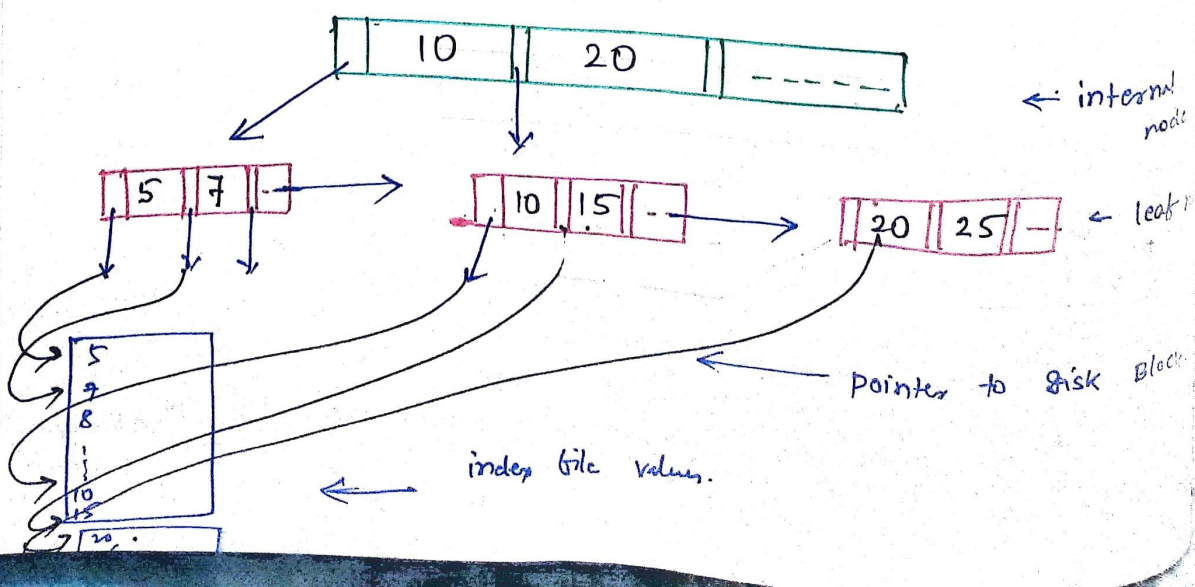
Idea of B+ Tree :-

All the leaf nodes are connected with each other and Leaf nodes allows you to sequencially access the data.

Since leaf nodes allows us to sequencially access the data so, every data item has to be present in the leaf nodes. Whatever present in the internal node is also to be present in the leaf nodes. so that we can access all the table entries in a sequencial manner.

* B+ Trees are index blocks.

B+ Trees.



* Advantages of B^+ Trees over B Trees.

you do not need 2 pointers. you only need one pointer with every key attributes also we have sequencial Access.

* Drawbacks of B^+ Trees.

This might happen that key value might be present Two times. (i.e. all the key value that is present in the internal nodes will be also be present in the leaf nodes also).

* B^+ Tree is the fantastic Data structure that works upon multilevel indexing. They grow and shrinks automatically. They have only one pointer. bcz. your internal nodes have block pointers pointing to disk blocks. which have index file values. (as we saw in the diagram).

* Both B and B^+ Trees insure that every leaf node present at the same level.

* Height of these (B and B^+) Trees is actually $\log n$.

Abhishek Sharma Notes.

20. ACID properties.

Transactions and concurrency control (ACID property).

Transaction: A set of logical instructions for the Database that should happen together. Either all of them Happen or non of them will happen.

Example: In a Bank data of Two people x and y

x has initial balance 500 Rs. and y has 200 Rs.

Our task is to send 100 Rs. from x to y.

So the instructions are.

Before Transactions:

x : 500
y : 200

Read (x)

x = x - 100

Write (x) : 400

Read (y)

y = y + 100

write (y) : 300

After Transaction.

x : 400
y : 300

This is the whole process for Sending 100 Rs. from x to y.

So Transaction says either all steps will happen or nothing will happen.

Bich में stop नहीं करेंगे Step.

* Most of the databases (DBMS) have the syntax available for Transaction

You have to give instructions to DBMS that this is the Transaction, either it should happen completely or should not happen at all.

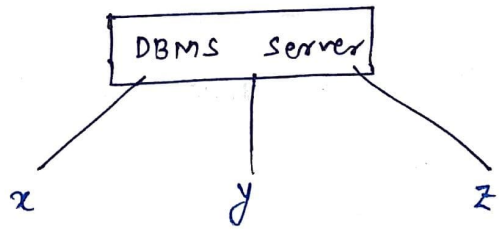
Transaction is one problem that your DBMS should solve

Another problem that comes with Transaction is concurrency.

Concurrency: concurrency says that if a DBMS server is doing multiple transactions. Then they all should be doing in concurrent way.

Same as operating system i.e. when a process that is doing IO, you bring another process that can use CPU.

Example.



⇐ DBMS Server

Multiple.

⇐ Transactions

Transaction 1.

x = 500 Rs.

y = 200 Rs.

Transfering 100 Rs. from x to y.

Transaction 2.

z = 200 Rs.

Transfering 200 Rs to x account.

Let's Discuss if a Transaction is happen in Transaction 1. and it stops at Deduction Rs 100 from x account and at the same time z's account Transfer the money to x account

So, we might not get our Desireed Result.

i.e. These are the problems happen in Transaction and concurrency.

To get Rid of these types of problem we have ACID properties. in our DBMS.

These are 4 certain things that will insure Transaction and Concurrency in the DBMS.

These are. ACID properties.

1) Atomicity
2) Consistency
3) Isolation
4) Durability.

Atomicity: Either the whole Transaction happen or nothing will happen.

Consistency: your Database should be in consistent state After the Transaction has happened.

Example: In a Bank, two people x and y have money 500 and 400 respectively. After any Transaction happen. then the Sum of them remain Constant. It should not change.

x = 500	= Sum = 900
y = 400	

At Befor Transaction.

if any Transaction Happen

Sum will be 900.

This is consistency says.

Money should not go anywhere. It should be constant before and after the Transaction. This is consistency says. your DBMS should be in consistent state.

Isolation: There are multiple transaction that is doing by your DBMS server. Isolation says that every Transaction feel like that it is running alone. Any Transaction should not be impacted by another Transaction. All the Transaction should be running in alone ~~seperately~~ and in a Sequencial manner.

Ex: If you are doing Internet Banking. and at the same time anyone sending you money. So Here are Two Transaction is happening at the same time. But your internet Banking Transaction will be alone and safe. It will not an effect of anyone sending you money. at the same time.

Duarability: Duarability says that once your transaction is done, whatever will be the changes. it should be in ↓ Stay your DBMS. It should not be lost. Changes should be durable. It should not be lost.

Abhishek Sharma. 21. ~~conflict~~ View Serializability.

Serializability: If there are many transaction take place at a same time. Let's say for Transaction T_1 and Transaction T_2. if T_1 T_2 or T_2 T_1 condition is satisfied then it is called serializabiles schedule.

Example.

Transfering 10 Rs from x to y (T_1)	Transfering 10 Rs from y to z (T_2)
Read (x)	
x = x - 10	
write (x)	
	Read (y)
	y = y - 10
	write (y)
Read (y)	
y = y + 10	
write (y)	
	Read (z)
	z = z + 10
	write (z).

Schedule.

* The term T_1 T_2 means that if T_1 being Run Before T_2 and T_2 T_1 means that if T_2 being Run Befor T_1

कोई भी एक Transaction पूरा हो जाए तो ही next कोई दूसरा start हो। This is the meaning of Serializbigel Schedule.

* Serializability is used to validate an interlived schedule.

We have serializability of Two Types.

Serializability.

i) view serializability

ii) conflict serializability.

* View serializability.

It states that if their are 2 transaction happen T_1 and T_2 the both should follow $T_1 T_2$ or $T_2 T_1$ and the both conditions ($T_1 T_2$ and $T_2 T_1$) should follow view equivalent of that schedule.

View equivalent:

from the previous page x, y, z example (T_1 and T_2)

Read (x)		Read (Y)
x = x - 10		Y = Y - 10
write (x)	=	write (Y)
Read (Y)		Read (Z)
Y = Y + 10		Z = Z + 10
write (Y)		write (Z).

This is view stable.

for

* View serializability we need to validate every item in the entry. So it is most difficult to find view Serializability.

View Seriazability. follows 3 rules.

i) Initial Read

ii) updated Read

iii) Final write.

* View Seriazability is used to validate wheter the interlived transaction is seriazible or not.

It is a very big problem not easy to find view Seriazability. So we will furthr see the next type of Seriazability i.e. conflict Serializability.

It is easy to compute.

---

22. conflict Serializability.

As we talked about in view serializability that we have a schedule which have interlived multiple Transactions two or more than two. We just need to validate that Schedule of interlived transactions whether it is equal to any of the serial schedule or not.

We discuss that view serializability is not easy to check. we need to consider each permutations. and for every permutations we need to check 3 Things.

i) initial read
ii) updated read
iii) Final write.

$\Longrightarrow$ view Serializability.

So we have a new Serializability which is easy to evaluate this is called conflict Serializability.

conflict Serializability: This serializability talks about conflicting operations. That's why this serializability is called conflict Serializability.

conflicting operation:

Two operations from two different transactions are such that they are on same data item and one them being "write". Then they are conflicting operations.

Example of conflicting Serializability:

Given Schedule.

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

our task is to check whether the given schedule is in conflict serializability or not ??

Serial Schedule. (T_1 T_2)

T_1	T_2
Read (A)	
write (A)	
Read (B)	
write (B)	
	Read (A)
	write (A)
	Read (B)
	write (B)

// क्योंकि पहले T_1 का Transaction पूरा होगया Then T_2 का हो रहा। order change नहीं हुआ। ∴ This is conflict Serializability.

One simple Method to check whether the given schedule is conflict Serializability or not.

The Method is called (Graph based) Method.

and the Graph is called **precedence Graph.**

*

Given Schedule

T_1	T_2
Read (A)	
	Read (A)
	write (A)
write (A)	
Read (B)	
write (B)	
	Read (B)
	write (B)

Q. How we say this given schedule is in conflict serializability or not by the help of graph method. ??

Method to check the conflict Serializability via presedence graph.

Rules: i) First we will draw how many transactions.

Like There are two transaction so we will write this

ii) For a given transaction we will follow only one Read and write operation.

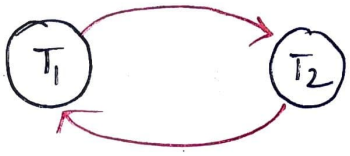
i.e. in the given schedule (T_1) A is Read. then Again in (T_2) A is read then after A is write.

so we will ignore T_2's A is read.

only we will follow T_1's A read and T_2's A write. and we will draw like this in graph.

iii) After this, in T_2, A is read and in T_1, A is write.

So again T_2 to T_1 in the graph there will be edge.



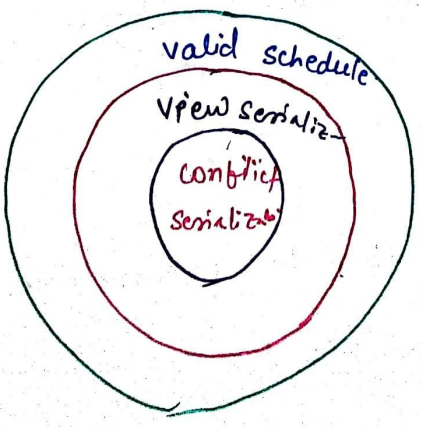
So if the circle completes (🔄) in the graph then the given schedule is not conflict serializbile.

* If we draw a graph and there if we fond any type of cycle then the give schedule is not in conflict Serializable.

* conflict Serializability is very strict.

There might be some schedule which may not be in Conflict Serializability.

* view serializability is slightly less strict compared to conflict serializability.



These are some valid schedule. Then we will check view Serializability out of some valid schedule. and then out of these view Serializable schedule some of them are conflict Serializablity. that we can check.

23. Recoverable, cascadeless and Strict Schedule.

Sometimes it may happen that transactions might role back (stop) in the middle. and when the Transactions role back none of the changes should be reflect in Databases.

Example:

T_1	T_2
Read (A)	
Write (A)	
	Read (A)
	Write (A)
Read (B)	
	Commit.
Write (B)	
Roll Back	

→ Dirty Read.

Before T_1 (B) Rolling Back T_2 Transaction is commited. Once the Transaction is commited, you can't do anything. commited means changes are reflected.

This is called Irrecoverability.

* It is not allowed your schedule to irrecoverede.
* your schedule must be recoverable.

The above example is doing Dirty Read.

Dirty Read: When a Transaction is doing irrecoverable. Then it is doing Dirty Read. It is called Dirty Read.

Rule: If a Transaction is doing Dirty Read then ~~it is~~ this transaction must commit after the transaction from which it has read.

* If a Transaction is irrecoverable then we will find the Dirty Read and do ~~the~~ follow the rule.

Dirty Read: When you read something from another transaction which is not commited yet.

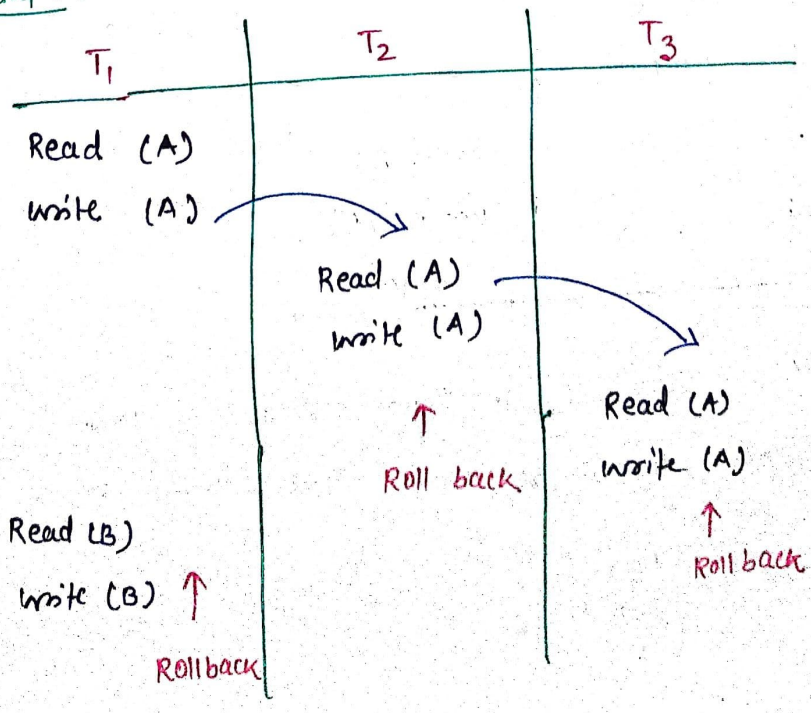
By following the Rule we can make a irrecorable transaction to Recoverable Transaction..

* cascadeless *

* One Rollback is causing other transactions to roll back is called cascading.

we want our schedule to be cascadelessness.

Example:



* Cascadelessness is not the mandetory Requirment. It only causes some performence issues.

* Recoverability is mandetory. your Transaction, your schedule must be recoverable.

Q) How to insure cascadelessness ?! that it shouldn't happen ??

Ans:- Rule: DO not Read the item from the transaction which is not commited yet.

Example. (Related to previous ~~Problem~~ Example).

T_1	T_2	T_3
Read (A)		
write (A)		
Read (B)		
write (B)		
Commit		
	Read (A)	
	write (A)	
	Commit.	
		Read A)
		write (A)
		Commit.

* Fisrtly we will find ~~Dirty~~ Read and then we will say do not Read the item from the Transaction which is not commited yet.

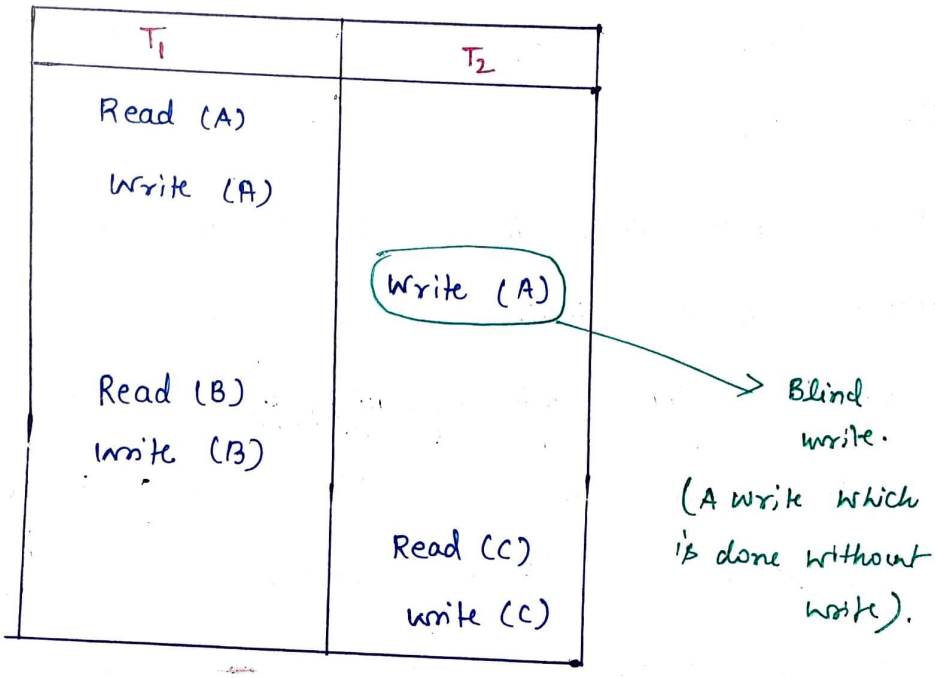
T_1 पहले commit होगा Then T_2 में Read होगा (A) thenn T_2 commit होगा Then T_3 में A Read होगा and so on. This is called cascadless.

* Strict Schedule *

Abhishek Sharma Notes.

The idea of strict schedule is we are not only handle Dirty Read. we also ~~Bottom~~ handle Blind Write.

Blind Write : A write which is done without write.



* In a Nutshell. *

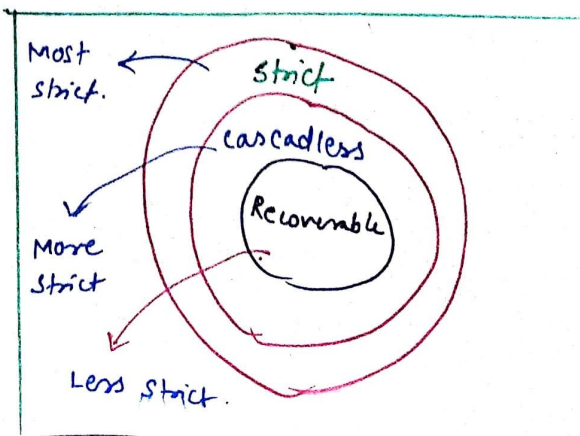
* Recoverable : Recoverable schedule are least strict.

It says that order of transactions should be according to Dirty reads.

* cascadless : It says that if a transaction is doing dirty read, ~~does / does not dirty / read only / if the transaction~~ do not read the item from the Transaction which is not commited yet.

* cascadlessness is more strict than Recoverable.

* Strict schedule :- It says that, not only dirty read, even write operation should wait other Transactions to commit. that means no blind write.



So far, in the Transactions and Concurrency Control, we have talked about ACID properties, we have talked about Steriazibility, Recoverablity, cascadlessness and strict schedule.

This things are there to insure that whether the schedule is fine or not.

* Now we will see that How can we generate the schedule in our DBMS server. so that so that it doesn't violate steriazibility, and it make sure that transactions are Recoverable and may be insures other things (like, cascadless, strict) as well.

---

24. Two phase Locking.

Lock: We use Lock to lock something in our Database. (in OS Lock will be understood easily). We lock something so that no other transaction can work on those things.

We have two types of Lock.

i) Shared Locks.

ii) Exclusive Locks.

* Lock is a basic Mechanism to implement two phase Locking protocols.

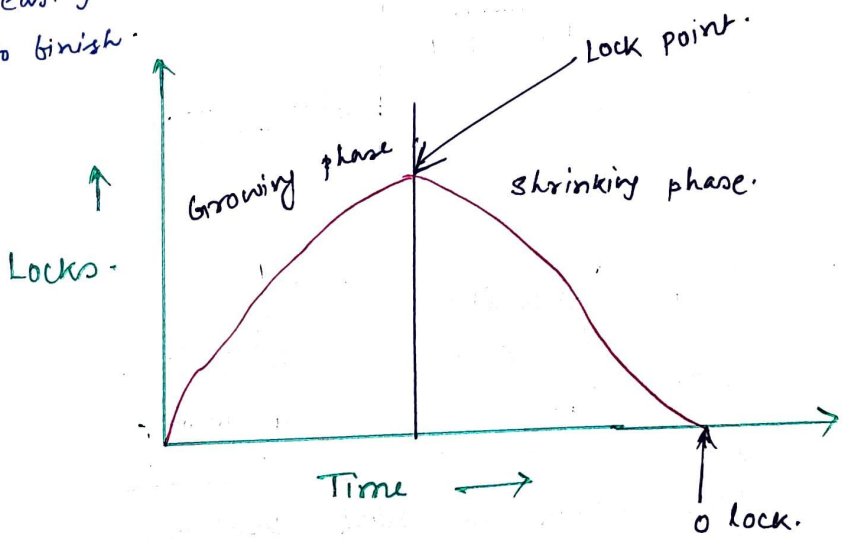
Idea of Two phrase Locking protocol:

we have two phases in a Transaction.

i) Growing phase

ii) Shrinking phase.

we will do that over the time, we have locks for every Transactions. There is an phase in which it keeps accuring the locks. and after that it starts releasing the locks. it goes to 0 locks when it is about to finish.



keeps on accuring the locks = Growing phase.
Releasing the lock = Shrinking phase.

* Note: once a Transaction Releases a lock it should not again accuring the locks. That means. if the Transaction start in shrinking phase then it will not go again in growing phase. we accure all the locks at ~~one time~~ all together then we start releasing the locks.

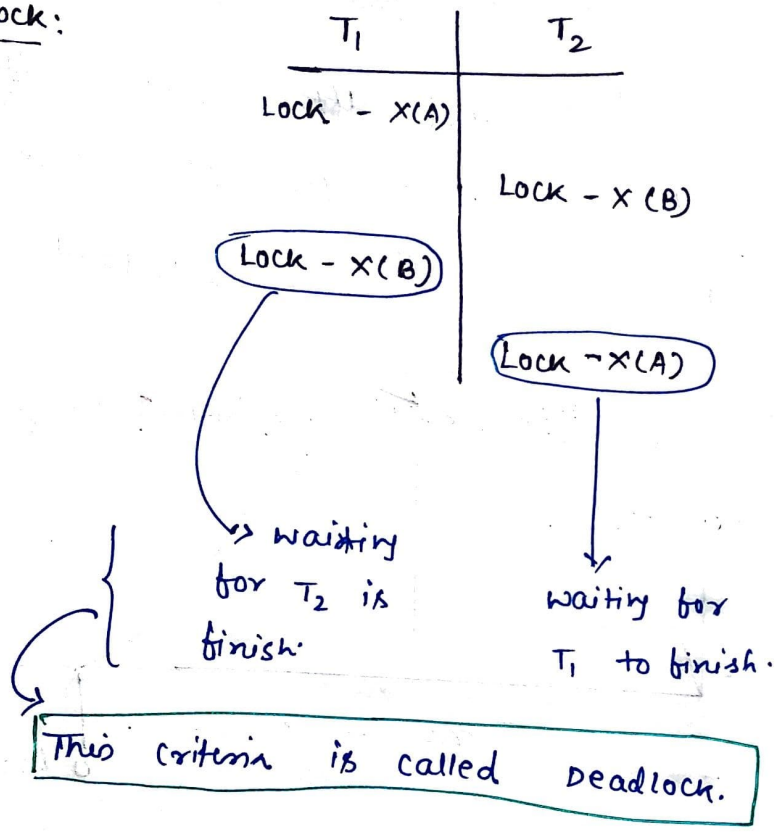
Lock point: we the transaction finishes growing phases and at the point when transaction enters in the shrinking phase is called lock point.

* Two phase locking protocol ensures conflict serializability as well as view serializability.

25. Problems with Basic 2-phase Locking protocol. (2PL)

1) 1st problem is Deadlock.

Dead lock:



2) 2nd problem is Recoverability.

If a schedule is not recoverable then it is not going to cascadlessness also.

Example:

* The Basic 2 phase Locking protocol that we have discussed is not Recoveral. and also not deadlock free.

Abhishek sharma notes.

26. Conservative, Strict and Rigorous 2 Phase Locking protocols. (2PL)

We have seen problems in 2PL. Here we will find solution for the problem that arrises in 2PL.

Conservative 2PL:- It handels Deadlock.

Idea: Remove Hold and wait. Accuires all the lock first.

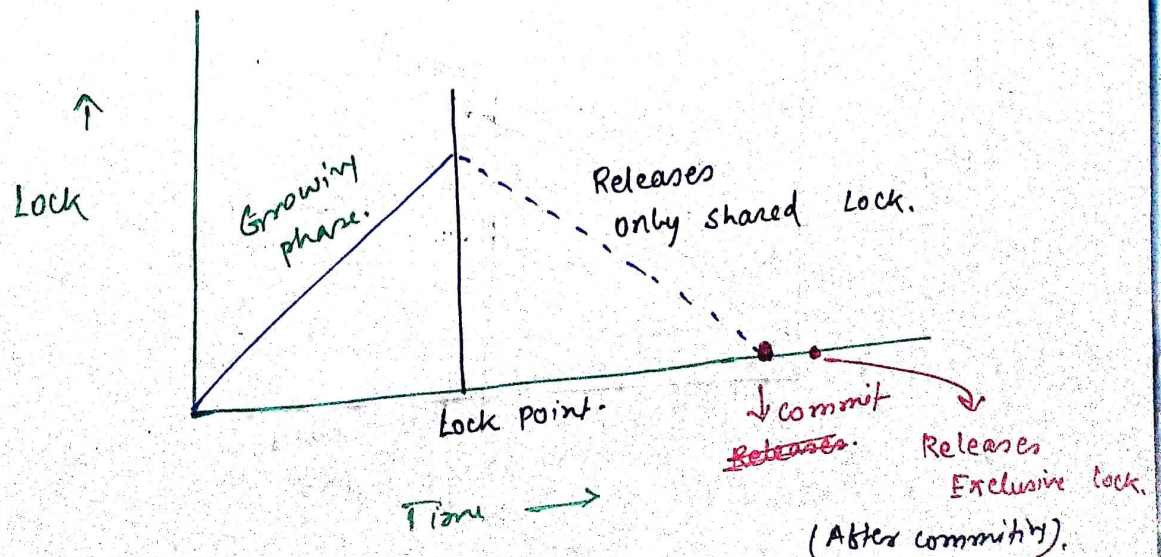
* Deadlock is not a big-big issue, as it looks like. Most of the DBMS Systems ignores the Deadlock. It happens rearly. So this Conservative 2PL is not used as much in practice.

* It only Handles Deadlock ← (conservative 2PL). → It does not ensure ~~Concurrency~~ recoverability.

Strict 2PL:- It ensures that your schedule is recoverable. It says that Release the exclusive lock only when you have commited. It means Dirty reads are not possible at all in Strict 2PL.

It releases Exclusive locks after commit.

* Strict 2PL is Recoveral and it is cascadlessness also.



Rigorous 2PL: It does not allows any locks to free untill unless all the transactions are commited.

So these are no shrinking phase.. we seem to have Less concurrency in Rigorous 2PL.

* Rigorous 2PL is the most used Locking Protocol among all these (conservative, strict and Rigorous).

It has been used since 1970's.

Reasons for Most Using Rigorous 2PL.

i) It is really simple to implement.

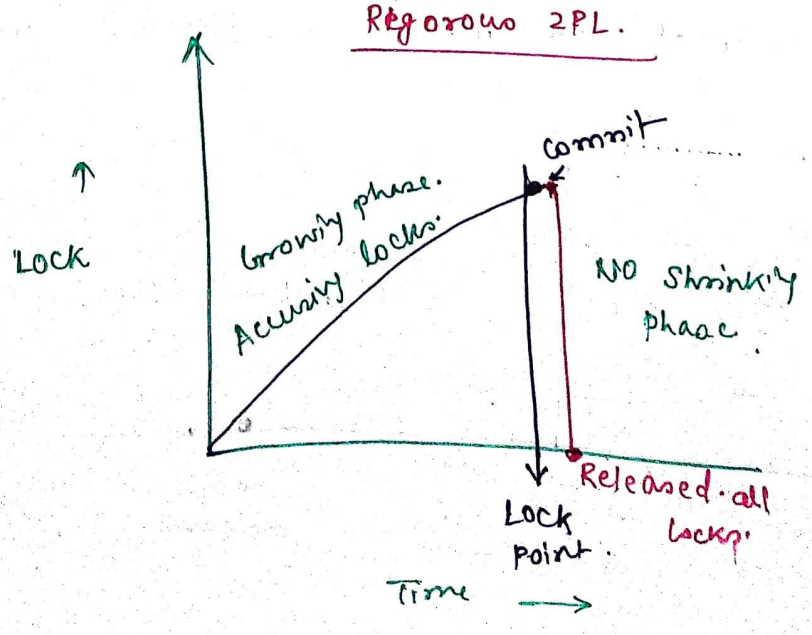
ii) In conservative and strict 2PL we have to find the lock point and after that we need to release the locks.

But Here in Rigorous 2PL, a transaction simply keeps on Accuring the locks. and ONCE it is commited, it releases all the locks.

That's why it is really Simple to implemnt.

No need to find Lock point in Rigorous 2PL.

Rigorous 2PL.



In a NUTSHELL all the 3, 2PL. Abhishek Sharma notes.

Conservative, Strict and Rigorous (2PL).

Conservative 2PL.	Strict and Rigorous 2PL.
i) Deadlock Free.	i) Deadlock possible.
ii) Recoverability and cascadelessness not gurranteed.	ii) Recoverability and cascadlessness gurranteed.

Rigorous Vs. Strict 2PL.

i) Strict Allows more concurreny.

ii) Rigorous is simple to implement and most used.

iii) Rigorous also suitable in distributed enviornment.

---

27. Timestamp Based Protocols.

We studied Two phase protocols.

After that This is new type of protocoles that is commonly used. and we will study now. i.e. Timestamp based Protocols.

Timestamp based Protocols.

i) This protocol is dead lock free.

ii) It ensures. conserstarisblity.

iii) It doesn't gurrenty Recoverability and cascadlessness.

~~Timebased~~ .

Timestamp Based protocols: Every Transaction that enters into your system has a timestamp. and this timestamp is the time when it enters ~~your~~ into the system. and every object on your transaction being Read or write. They also have a read timestamp and also have a write timespamp.

Read Timestamp: Read timestamp of an object is the timestamp of the latest transaction that performed a read operation on ~~this~~ this object.

Write Timestamp: write timestamp of an object is the timestamp of the transaction that performed the latest write operation on this object.

Example of timestamp based protocols.

If there are 3 transaction is happening T_1, T_2 and T_3. at time 10 sec, 30 sec, 20 sec. Respectively; Then their Serial Schedule will be T_1 T_3 T_2

T_1	T_2	T_3
Time : 10	Time : 30	Time : 20

~~Serial~~ Serial schedule will be T_1 T_3 T_2.

It will be increasing order of their Timestamp.

Example of Timestamp based protocols:

$T_s(T_i)$ → Timestamp of Transaction T_i

RTS (x) → Max Timestamp of a Transaction that read x.

WTS (x) → " " " " " " write x.

Given schedule.

T_1 (10)	T_2 (20)
Read (A)	
write (A)	
	Read (A)
	write (A)
	Read (B)
write (B) ↑ Roll back	

$T_s(T1) = 10$

$T_s(T_2) = 20$

RTS (A) = 10

WTS (A) = 10

NOW,

RTS (A) = 20

WTS (A) = 20

RTS (B) = 20.

NOW WTS (B) = 10 नहीं होगा XX

यहाँ से Roll back हो जाएगा Transaction.

According to Timestamp Based Protocol.

होगा increasing order में।

* Rules followed by Timestamp Based protocols:

Abort = roll back the Transaction.

(i) Rules for Read (x) in T_i

if $TS(T_i) < WTS(x)$ then Abort (roll back) T_i

else Allow Read (x)

$RTS(x) = \max(RTS(x), TS(T_i))$.

(iii) Rules for write (x) in T_i

if $TS(T_i) < RTS(x)$ or $TS(T_i) < WTS(x)$ then

Abort T_i.

else

Allow write (x).

$WTS(x) = TS(T_i)$

* **Schedular**: Like in operating system, there is a Schedular present in your DBMS Server also. The job of the schedular is to Schedule Transactions in interlived Manner.

DBMS finished.

Abhishek Sharma Notes.

Time: 11:50 Pm.

Date: 05/09/2021

* If you read these notes plear share among your friends and college group the group link.

*

"Thank you."

for supporting me.

Abhishek Sharma.

Currently A undergraduate from NIT Durgapur.

05/09/2021