Database :- A database is collection of inter-related data.

DBMS :- The DBMS is a software that provides an ~~environmental~~, i.e, convenient and efficient for the user's first storage and accessing of data from the database. In addition to creating the database, it should also provide us with some security features (unauthorized access and a recovery from failure).

Disadvantages of TPS (Transaction Processing System or File Processing System or Traditional Processing System) over DBMS :-

I) Data Redundancy and Inconsistency

II) Difficulty in accessing data

III) Difficulty in specifying integrity constraints

IV) Atomicity Problem (Atomic → Indivisible)

It states that either all the steps of a transaction should be executed or none of them should execute.

→ Transaction - Any operation in the database is called as Transaction.

Ex:- Banking

A B

Transaction → Transfer 100 Rs from A to B.

A = 500, B = 500

```
            r(A)
            A = A-100
      400   w(A)
            r(B)
            B = B+100
            w(B)
```

All steps should be at one time.

V) Concurrent Access Anomaly

Series Execution :- One process ends then another one starts.

Parallel Execution :- Simultaneously two processes run.

Concurrent Execution :- Other process execution started, even before the first process end.

vi) Database Security.

It states – "all the users of the database shouldn't be disclosed with the entire database".

Levels of Abstraction

Major advantage of database is that it provide the users a certain levels of abstraction where it hides certain details of implementation from the user. The DBMS package provides us with three levels of abstraction.

i) Physical Level   ii) Logical Level   iii) View Level

Physical Level

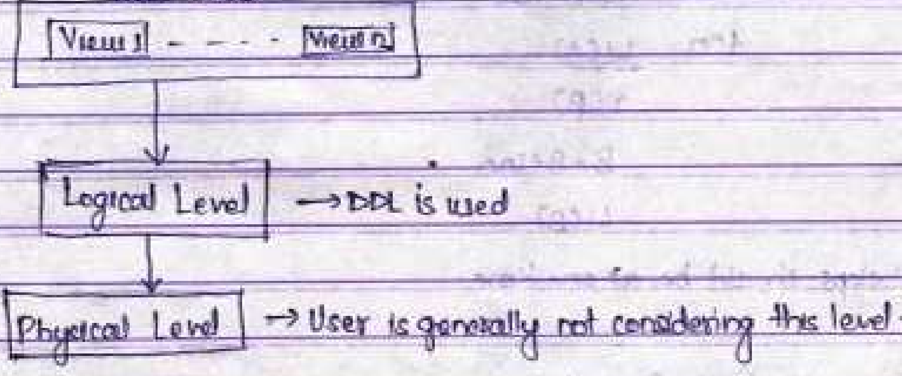
It is the lowest level of abstraction that tells us how data is stored.

Logical Level

It is the next higher level of data abstraction that tells us what data is to be stored.

View Level

It is the highest level of data abstraction where a part of the database is disclosed to the users.



Three-tier Architecture or three-Schema Architecture or ANSI/SPARC Model.

Lecturenotes.in

Instance and Schema

Instance :- The contents of the database at a specific instant of time is called as Database Instance. It is also known as snapshot of database.

Schema :- The overall structure of the database is called Schema. The instance of database changes frequently but schema changes very rarely.
The overall structure of a database at the physical level is called as physical schema, at the logical level called as logical schema and at the view level, Subschema.
For a database, there will be one physical schema, one logical but ~~an~~ more than one sub-schemas.

Data Independence
The ability to modify the schema definition at one level without causing the application programs to be rewritten at the next higher level is called as Data Independence. DBMS provides us with two types of data independence :-
I> Physical Data Independency
II> Logical Data Independency

Physical Data Independence :- The ability to modify physical schema definition without causing the application programs of logical level and view level to be rewritten is called Physical Data Independence.

Logical Data Independence :- The ability to modify logical schema definition without causing the application programs of view level to be rewritten is called Logical Data Independence.
Logical Data Independence is more difficult to achieve than physical one because the application programs are heavily dependent on logical level.

Data Models

The underline structure of database is a Data Model. It is a conceptual tool to specify data, data relationships, data semantics and the consistency constraints. Data Models are of three types:-

1> Object Based Logical Data Model
11> Record Based Logical Data Model
111> Physical Data Model → Not included in physical level.
[illegible]

Object Based Logical Data Model
a) E.R. (Entity Relationship) Model
b) Object Oriented Model
c) Semantic Model
d) Functional Data Model

Record Based Logical Data Model
a) ~~Relationship~~ Relational Model
b) Network Model
c) Hierarchial Model

Lecturenotes.in

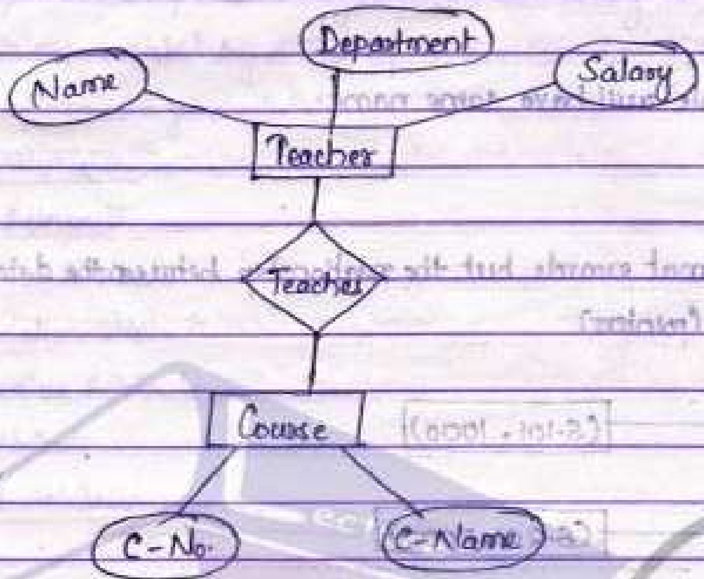
E-R Model
It is based on perception of real world. An entity is an object i.e. distinguishable from other objects by a set of properties. These properties are called attributes. The entities of the same type are grouped together to form an entity set. The entities of various entity sets can be associated through relationship sets. The E.R Model is represented pictorially through an E-R Diagram. The notations used are:-

1> $\boxed{\text{Rectangle}}$ → Entity Set

2> (Ellipse) → Attributes

3> ◇ → Relationship Set

4) ——————— → Link (attributes to entity sets, then entity sets to relationship sets).



Physical Data Model
1) Unifying Model
2) Frame-Memory Model

~~Relational Model~~
Record Based Logical Data Model
Here data is stored in the form of fixed format records. So it is named.
Each record has several columns, each of which are of fixed length.
The Relational Model
Here data is ~~s~~ and the relationship among the data is represented by a collection of tables. Each table will have a unique name. The tables have multiple columns and each column is given a unique name

Ex:- Customer

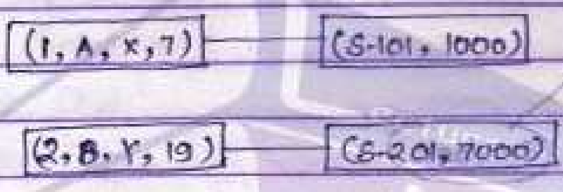
ID	C-Name	Address	Tel-No.	AC-No
1	A	X	7	S-101
2	B	Y	19	S-201

Account

Account No.	Balance
S-101	1000
S-201	7000

No two columns in a same table will have same name.

Network Model
Data is stored in fixed format records but the relationship between the data is represented through links (pointers)



The arrangement of the nodes (records) are such that, it forms a graphical structure.

Hierarchial Model
• Data is arranged in fixed format records.
• Relationship represented through links (pointers).
• The arrangement seems to be like that of a tree structure.

Q:- Role of DBA (Data Base Administrator).
Q:- Types of DataBase Users.

ER MODEL

18/2/11

An entity is an object that is distinguishable from other objects by a set of properties.
Collection of entities of a same type forms an Entity Set.
An entity is represented by a set of attributes. The attributes are the descriptive properties of an entity.

Types of attributes

I> Simple Attribute
II> Compound Attribute
III> Single Value Attribute
IV> Multi-value Attribute
V> Null Attribute
VI> Derived Attribute

I> Simple Attribute :- An attribute whose value can't be sub-divided further is called as a Simple Attribute.
Ex:- Roll No.

II> Compound Attribute :- Can be subdivided further are Compound Attributes.
Ex:- Name, Telephone No.

III> Single Valued Attribute :- Those attributes that can take only one value is called Single valued Attribute
Ex:- Gender, Date of Birth, Reg. No., Roll No.

IV> Multiple valued Attribute :- Can take more than one value at a certain instant of time is Multi-valued Attribute.
Ex:- Tel. No.

V> Null Attribute :- Those attributes that can take null value are called as Null Attributes. Null indicates that value for that attribute is currently not available. Null is not saying as 'zero'.
Primary Key Attribute can't contain null value.

VI> Derived Attributes :- Those attributes whose value can be derived from other attributes is called Derived attributes.
Ex:- Age is a derived attribute provided DOB is your base attribute.

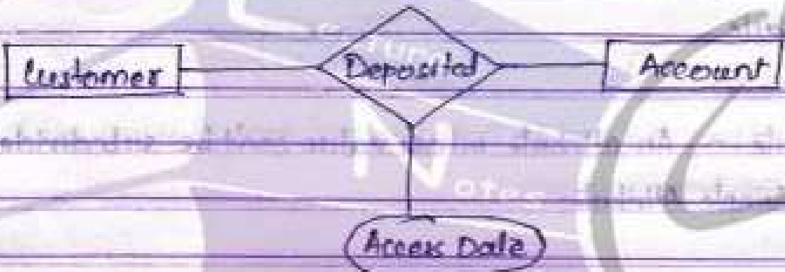
15|2|11

Relationship

A relationship is defined as the association of entities of several ~~times~~ types. Relationships of the same type are grouped together to form a relationship set. The association between the entity sets to create a relationship set is called as participation. The participation of the entity sets in creating a relationship set can be either total or partial. An entity set is having total participation if all the entities of that entity set are involved in creating the relationship set. Otherwise, the participation of the entity set will be partial.

A relationship set can have its own descriptive attributes.

Consider an entity set Customer (ID, Name, Address) and Account (Account No., Balance). Let the relationship sets betn these two entity sets is 'deposited'.



Cardinality Mapping / Cardinality Ratio

It defines how many entities of one entity sets can be associated with how many entities of other entity set through a relationship.

There are of four types:-

I> One to one

II> One to Many

III> Many to one

IV> Many to Many

I> One to one:- One entity of entity set A is associated with ~~one ent~~ at most one entity of entity set B and one entity of B is also with at most one entity of A.

II> One to Many:- One entity of entity set A can be associated with one or more entities of set B and one ~~or more~~ entity of B is associated with atmost one

Lecturenotes.in

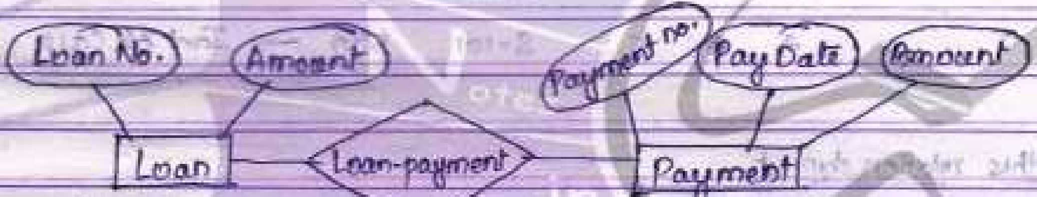
entity of A.

III> Many to one :- More than one entity of A → B, and one entity of Set B is associated with more than one entity of A.

IV> Many to Many :- More than one of A → More than one of B.
More than one of B → More than one of A.

Existential Dependency

If the existence of an entity set 'X' depends on existence on existence of entity set 'Y', then 'X' is said to be existential dependent on 'Y'. If 'Y' is deleted then automatically 'X' is deleted, here 'X' is called subordinate entity of 'Y'

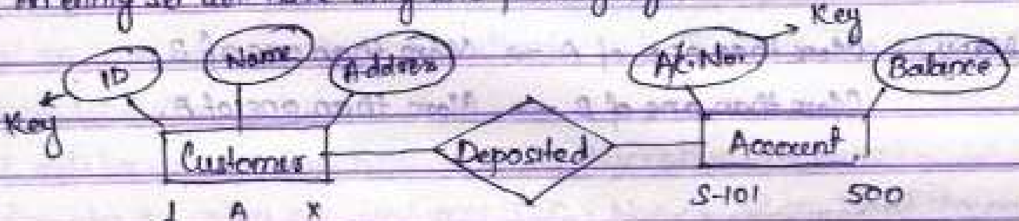


Here the 'Payment' entity-set is existential dependent on the 'Loan' entity set because once the loan entity set is deleted, then automatically all the payment details of that loan are deleted.
'Y' is called as the Dominant Entity.

KEY (Super Key)

A set of attributes that helps to uniquely identify an entity within an entity set is called as Super Key.
Consider 'Student' entity set with these attributes:-
R.No. Name Branch Address Ph.No.
Here (R.No., Name, Branch, Address, Ph.No.) is a super key.
Super key may contain extraneous attributes, by removing these attributes we can get Minimal Super Key which contain keys called Candidate Keys.

Minimal Super Key may contain any no. of attributes.
An entity set can have more than one candidate key.
~~But However~~ Whichever candidate key is selected by ~~database des~~ database designer as a means of distinguishing $\overset{one}{\wedge}$ entity from other entity key is called Primary Key.
An entity set will have only one primary key.



1	A	X	S-101	500
2	B	Y	S-201	700

Deposited

1	A	X	S-101	500	
2	B	Y	S-201	700	
2	B	Y	S-101	500	→ Joint A/c case

For this relationship set.

Primary Key of Deposited = (Primary Key of Cust.) ∪ (Prim. Key of Acc)
= (Customer ID, A/c No.)

There are certain entity sets which don't have any primary key, these entity sets are Weak Entity Sets. These don't have independent existence. These entity sets depend on strong entity sets for their existence.

Loan ($\overset{P.K.}{\text{Loan-No.}}$, Amount)

L-101	50,000	] taken on 21/2/11
L-207	200000	

Payment (Payment No., Pay Date, Pay Amt.)

1	21/3/11	5000
1	21/3/11	5000
2	21/4/11	5000

2   21/4/11   5000

This is not containing any key, so it is a weak entity set.

22/2/11

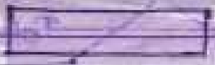
The weak entity sets have some attributes which partially behave as a key, these attributes are known as partial keys or discriminators.
In the weak entity set 'payment', 'PayNo.' attribute is a discriminator.

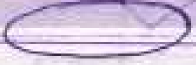
P.K of weak entity set = P.K (strong entity set) U Discriminator

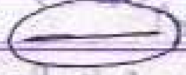
P.K (Payment) = {Loan No., Pay No.}

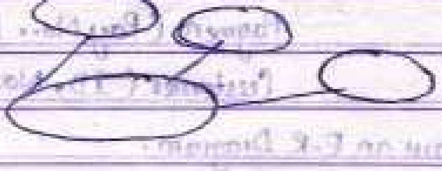
A weak entity set is always a part of one-many relationships.
The relationship between a strong entity set and a weak entity set is called as an Identifying relationship. An Identifying relationship cannot have any attributes.

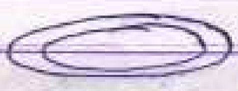
NOTATIONS USED IN E-R DIAGRAMS

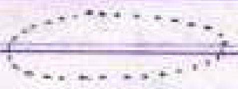
1> Entity Set →


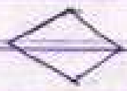
2> Attributes →

(Simple) & (Single-Valued)

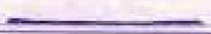
3> Primary Key Attribute →


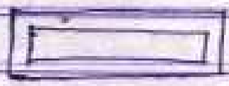
4> Compound Attribute →


5> Multivalued Attribute →


6> Derived Attribute →


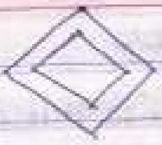
7> Relationship Set →


8> Links →


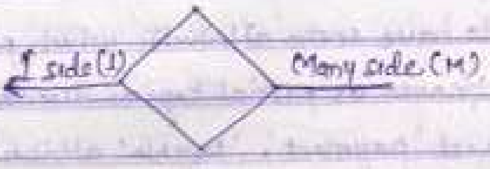
9> Weak Entity Set →


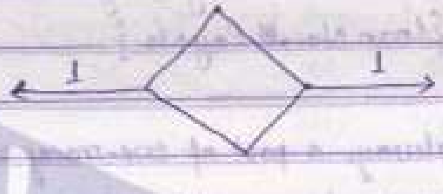
Lecturenotes.in

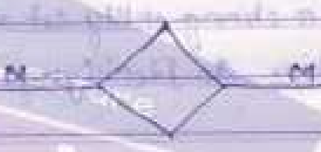
10> Identifying Relationship Set :-



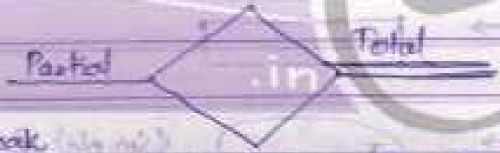
11> Cardinality Ratio :-







12> Total and Partial Participation :-

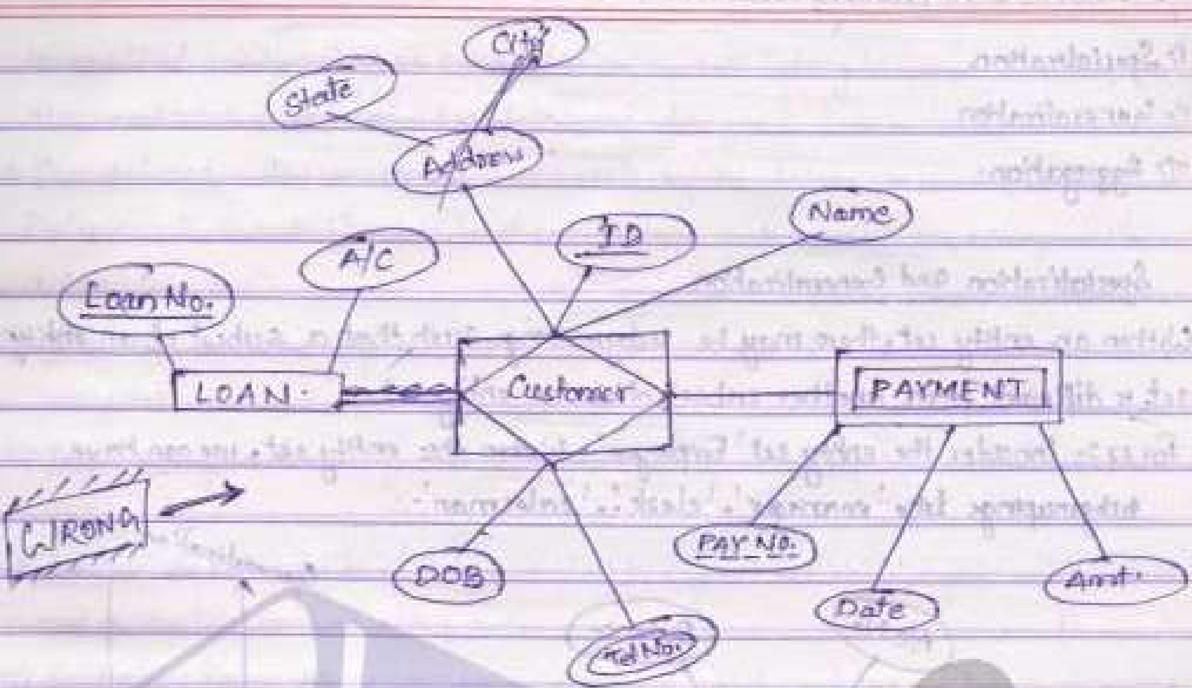


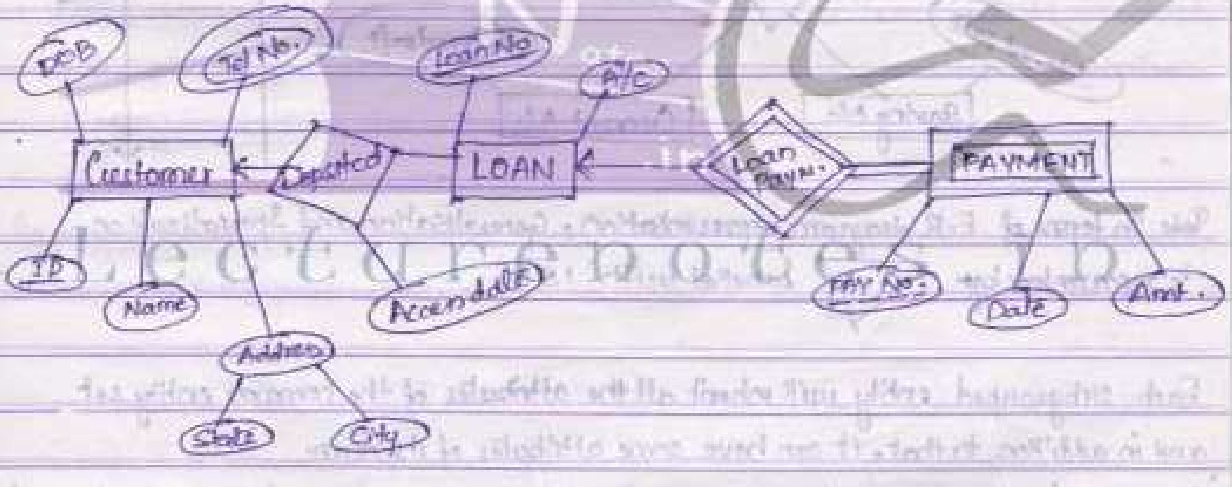
Always in terms of strong & weak entity.

Consider :- Loan (Loan No., A/c)
Payment (Pay No., Pay-Date, Pay Amount)
Customer (ID, Name, Address, DOB, Tel. No)

Draw an E-R Diagram.

lecturenotes.in





lecturenotes.in

25|2|11

Extended E-R Features (EER Features)

i> Specialization

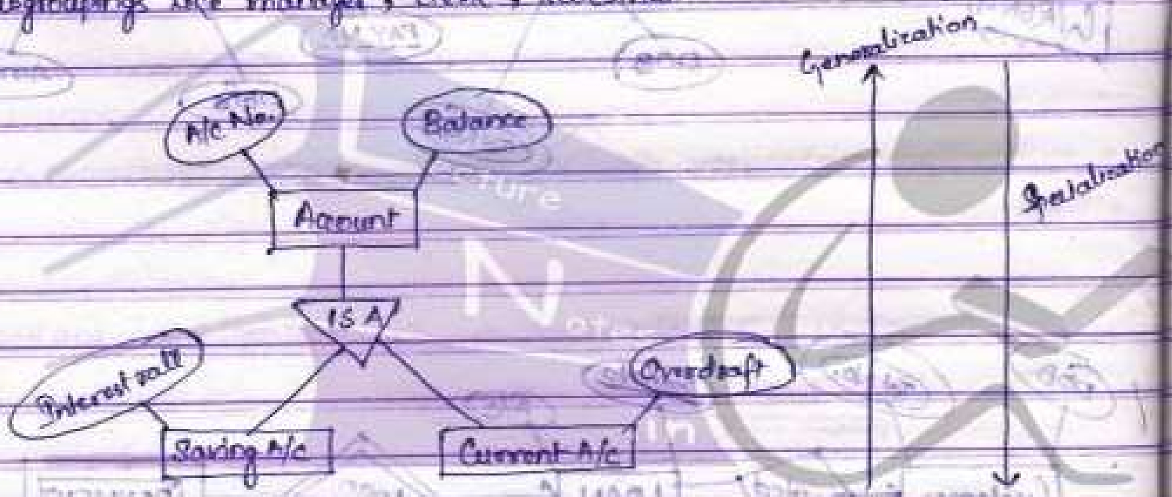
ii> Generalization

iii> Aggregation.

Specialization and Generalization

Within an entity set, there may be subgrouping such that a subset of an entity set is different from another subset of same entity set.

for ex:- Consider the entity set 'Employee'. Within this entity set, we can have subgroupings like 'manager', 'clerk', 'salesman'.



Note In terms of E-R diagram representation, Generalisation and Specialization is represented by labelled with 'ISA'.

Each subgrouped entity will inherit all the attributes of the common entity set and in addition to that, it can have some attributes of its own.

'Saving A/c' entity set will 've the attributes A/c No., Balance and interest rate and similarly entity set will've A/c No., Balance, overdraft.

Generalization - Specialization provides a hierarchial structuring among the entity sets that are derived from basic entity set. This hierarchial structuring produces a tree. If we traverse the tree from the top to bottom it is called as Specialization while traversal in reverse order is called as Generalization.

Note In Generalization - Specialization, we can enforce some design constraints:-

i> Condition defined :- during the creation of the entity, a predicate will be checked depending upon whether the predicate is true or false, it will be decided, to which

lower level entity set that ~~particular~~ entity will belong to.
~~The~~ constraint is enforced by database designer.

ii) User defined :- The constraint is enforced by the database user (DBA). He'll decide to which lower level entity set, that particular entity will belong to.

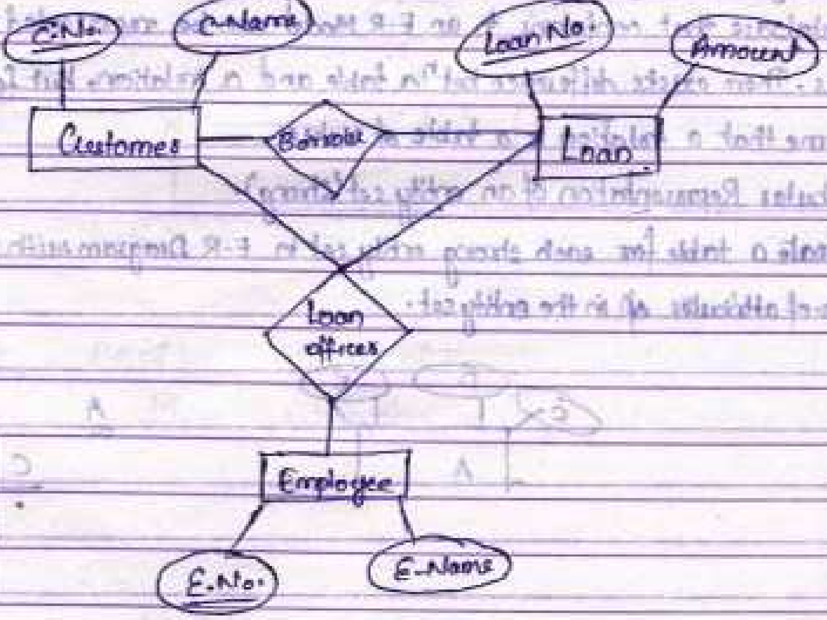
iii) Disjoint :- An entity ~~set~~ can't belong to more than one lower level entity set.

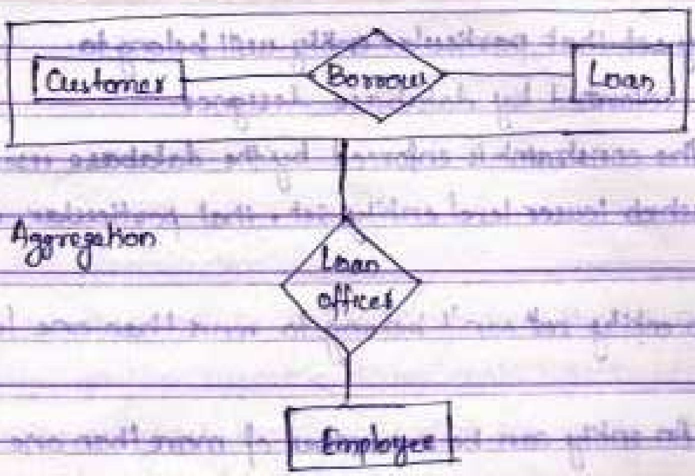
iv) Overlapping :- An entity can be a member of more than one lower level entity set.

v) Total :- All the entities of a higher level entity set must belong to at least one of the lower level entity sets.

vi) Partial :- ~~Some~~ All entities in higher level entity set couldn't be a member of lower level entity sets.
Suppose the higher level entity set is 'sportman' and two lower level entity sets are 'Play basketball' and 'Play cricket', then there may be some possibility that there may be some sportspersons who are neither playing basketball nor cricket.

Aggregation 26/2/11
In simple E-R diagram, it is difficult to express relationship within a relationship.





Aggregation is a process by which a relationship set is treated as a higher level entity set.

Steps in Database Design

i) User Requirements

ii) Conceptual design → choose a data model

iii) Specify the functional requirements (operations on the database).

iv) Design of database.

→ Logical design (Mapping of Data Model to Relational Model)

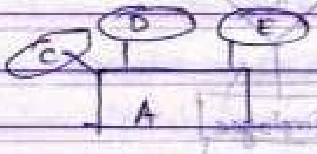
→ Physical Design (Implementation)

Mapping E-R Diagram to Tables

A database that conforms to an E-R Model can be represented as a collection of tables. There exists difference betⁿ a table and a relation, but for the time being we'll assume that a relation is a table of values.

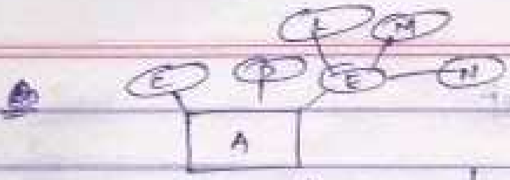
i) Tabular Representation of an entity set (strong).

• Create a table for each strong entity set in E-R Diagram with no. of columns equal to no. of attributes of in the entity set.



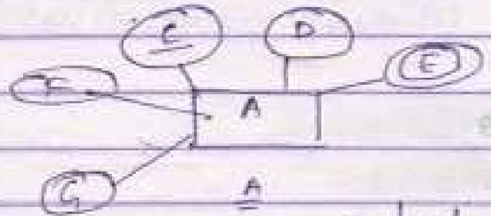
A

C	D	E



A

C	D	L	M	N

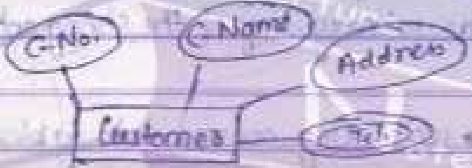


A

C	D	F	G

A

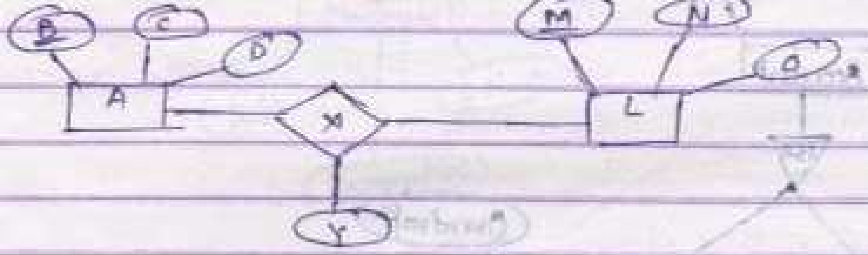
C	E



C-No.	C-Name	Address
1	A	X

C-No.	Tel.N
1	7
1	9

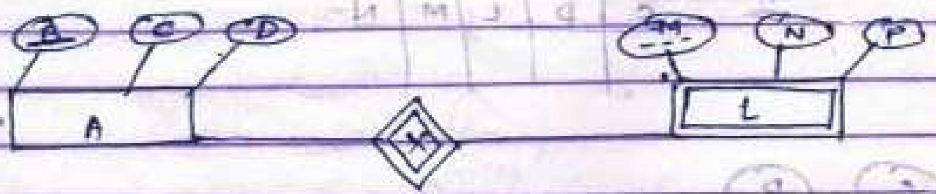
1) Tabular Representation of a Relationship set.



P.K of A	P.K of L	Attribute of X
B	M	Y

4/3/11

iii) Tabular Representation of a Weak Entity Set



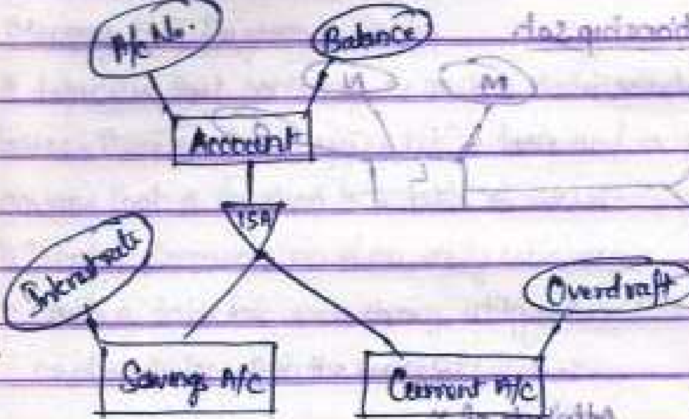
L

M	N	P	B

In the tabular representation of a weak-entity set, it will contain all the attributes of weak entity set as well as primary key of the strong entity set on which it is dependent.

NOTE:- An identifying relationship sets isn't being convert into tables. The information that will be reflected by the identifying relationship set will be present either in strong or weak entity set.

iv) Tabular Representation of Generalization & Specialization



We'll consider only two design constraints.

Lecturenotes.in

1) Total

a) Savings
(A/c No., Balance, Interest rate)

b) Current
(A/c No., Balance, Overdraft)

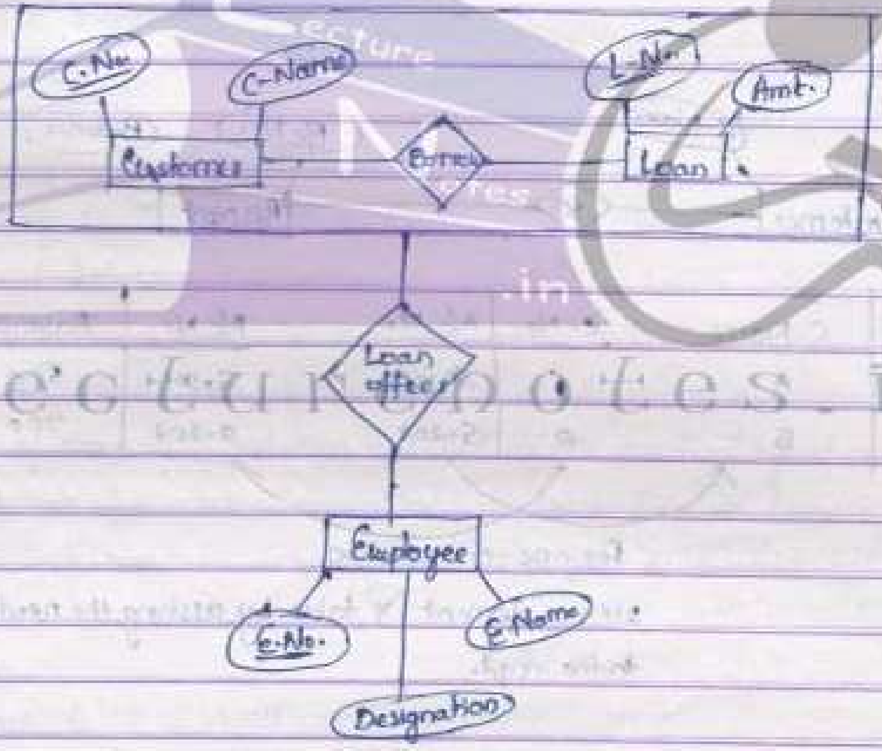
2) Partial

a) Account
(A/c No., Balance)

b) Savings
(A/c No., Interest rate)

c) Current
(A/c No., Overdraft)

v) Tabular Representation of Aggregation



In this aggregation's we'll have the tabular representation of three entity sets as well as two relationship sets.

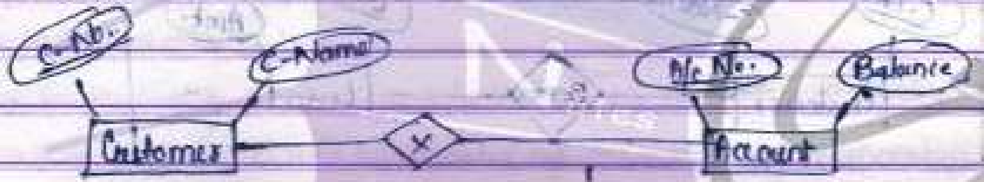
Lecturenotes.in

1) Customer (C-No., C-Name)
2) Loan (L-No., Amount)
3) Employee (E-No., E-Name, Designation)
4) Borrow (C-No., L-No.)
5) Loan officer (C-No., L-No., E No.)

} Primary Key is all the attributes

VI) Minimization of Tables

In the mapping of E-R diagrams to tables, we'll have two types of tables:- one reflecting entity set and other one for relationship set.
In minimization of tables, we try to eliminate some of the tables that reflect a relationship set.



C.No.	C-Name	E-No.	A/c No.	A/c No.	Balance
1	A	1	S-101	S-101	500
2	B	2	S-202	S-202	700

For one-one relation,
we can element 'x' table by pushing the needful table to the reqd.

a) For one-one relationship set:
This table can be eliminated by associating the primary key of one entity set with other entity set.

b) For one-many:

Lecturenotes.in

C-No.	A/c No.
1	S-101
2	S-202
1	S-202

ONE-MANY OR MANY-ONE

A/c	Balance	C-No.
S-101	500	1
S-202	700	2
S-202	700	1

This can be eliminated by transfering primary key of one side to many side.

c) Many-Many relationship set table:
This table can't be eliminated.

THE RELATIONAL MODEL 5/3/11

Proposed by Dr. E.F.Codd in 1970.
A relational database consists of a collection of tables each with a unique name. Tables have attributes and each attribute has a unique name. The total no. of attributes in a relation is called as degree of a relation and the total no. of rows in a relation is called as the cardinality of a relation.
Each attribute of a relation can obtain its values from its domain.
Each attribute can take only atomic values from its domain.

Relation:-
Condition: A to B ⊆ A×B
In general, a relation schema ~~are~~ R(A₁, A₂, ..., Aₙ) is represented by like this.
And the domain is represented by dom(Aᵢ).
Henceforth, the tables will be named as relations and the rows will be termed as tuples.

Characteristics of a Relation:-

i) Each relation will have a uniquename.
ii) Each attribute will have a uniquename.
iii) Each attribute will take atomic values from its corresponding domain.
iv) Ordering of tuples doesn't affect the relation.
v) Change in attribute values for a tuple isn't permitted within a tuple.
vi) No tuples can be identical.
vii) Change in ordering of attributes doesn't affect the relation.

The Relational Schema is also called as <u>Intension</u>.
The values within a database is called as <u>instance</u> of the database. It is also known as <u>relation state</u>, <u>snapshot</u> and <u>Extension</u> of a database.

- $r(R)$
  - R → name of Relationship Schema (always written in upper case)
  - r → Relational Instances (represented by lower case)

Relation state of R is represented by $r(R)$.
- If 'R' is given only → table is empty.
- If '$r(R)$' is given → some tuples are provided, table isn't empty.

KEYS

A set of attributes that helps to distinguish one tuple from another tuple within a relation is called as ~~Keys~~ SuperKey.
If K is a superkey, then any superset of K is a superkey.
We're often interested in finding out Minimal Super Keys. These Minimal Super Keys are called candidate Keys.
Any selected candidate key is the Primary Key. If S.K. is the superkey of a relation state $r(R)$ then for any two tuples t_1 and t_2:

$$t_1[S.K] \neq t_2[S.K]$$

7/8/11

Constraints in a Database

* While designing a database, we can enforce some restrictions. The restrictions can be of following types:-

i) Domain Constraint

ii) Key Constraint

iii) Entity Integrity Constraint

iv) Referential Integrity Constraint

Domain Constraint

An attribute can take only atomic values from its corresponding domain.

Key Constraint

$t_1[SK] \neq t_2[SK]$

Basis is no two tuples must be identical, in that relation.

Entity Integrity Constraint

The primary key of a relation can't contain NULL values.

Referential Integrity Constraint

This constraint is used to link two different relations in a relational database. Informally, referential integrity constraint (RIC) states that whenever a tuple in one relation is referencing to a tuple in another relation, it must refer to an existing tuple in that relation.

To understand this link, we've to first define a foreign key.

(FK) Foreign Key:- A set of attributes FK is a foreign key of a relation R_1, if it references to the primary key PK of the relation R_2 and satisfies the following two conditions:-

a) Dom(FK) in R_1 is same as Dom(PK) in R_2.

b) The value of FK must either be a subset of PK of R_2 or NULL.

In this case R_1 is called as Referencing Relation and R_2 is called as Referenced Relation.

Foreign Key (F.K)

Employee (Referencing) → Dept (Referenced)

(P.K) E-No.	E-Name	Dept No.
1	A	NULL
2	X	5
3	Y	3

(P.K) Dept No.	D-Name
1	CS
2	IT
3	MCA
4	ET
5	EEE
6	AEI

*F.K must be a subset of PK or NULL.

Sometimes a foreign key can refer to the same relation

E-No.	E-Name	Dept No.	Dept HOD No.
1	A	NULL	4
2	X	5	
3	Y	3	
4	B	6	NULL

Lecture Notes .in

QUERY LANGUAGES

Any statement that deals with retrieval of information from database is called as Query Language.

Query Languages are of two types:-

- → Procedural → specify sequence of operations.
- → Non-Procedural (SQL) → What is needed from database?

~~Procedural~~ Procedural Query Language:-

i) Relational Algebra

ii) Tuple Relational Calculus (TRC)] Non-Procedural

iii) Domain Relational Calculus (DRC)]

• Non-Procedural is converted first to procedural query languages.

iii) Query-By Example.

~~Relational~~ Relational Algebra

i) Selection	iii) Union	v) Cartesian Product	vii) Intersection
ii) Projection	iv) Set difference	vi) Rename	viii) Join Operation
ix) Division	x) Assignment		

i) to vi) are fundamental Operation and vii) ~~to~~ to (x) are derived operations. ~~All~~ derived ~~op~~ operations can be expressed only in terms of fundamental operations. The operands to these operations are Relation names. Some of these operations will take a single relation ~~or~~ name as operand. These are called as Unary Operations. Ex:- Selection, Projection and Rename. and the rest are the Binary Operations.

8/3/11

The Relational Algebra

Fundamental or Basic Operations

1> Selection

Syntax:

$\sigma_{condition}$ (Relation Name)

Ex:

- Select * from emp where sal>5000; $\equiv \sigma_{sal>5000}(emp)$
- Select * from emp where sal>5000 and deptno=15 $\equiv \sigma_{sal>5000 \wedge deptno=15}(emp)$

$\equiv \sigma_{deptno=15}(\sigma_{sal>5000}(emp))$

~~While~~ ~~tor~~ The selection operation is represented by σ. Syntax is given above. Example is also provided.
Once this statement is executed, all the tuples satisfying the condition is given as output relation table. The degree of the output relation is equal to degree of base relation.
'n' no. of conditions can be written as a subscript to σ using logical connectives 'or', 'and' and 'not'.

2> Projection

Denoted by Π. This operation is used to project some of the attributes of the base relation in the output relation.

Syntax:

$\Pi_{a_1, a_2, \ldots, a_n}$ (Relation name)

Ex:- Select ename from emp; $\equiv \Pi_{ename}(emp)$

$\Pi_{ename, job, sal}(emp)$

Select ename from emp where job like 'clerk' and deptno = 10;

$\equiv \Pi_{ename}(\sigma_{job\ like\ 'clerk' \wedge deptno=10}(emp))$

Use of more than one relational algebra condition in a statement. it is called Composition.

3> Union :

$$E_1 \cup E_2 = \{x \mid x \in E_1 \text{ or } x \in E_2\}$$

To find out the ~~relation~~ union betn two relations, the degree of both the relations must be same (or it is said to be Union Compatible).

Union Compatability Conditions:-

i) degree of E_1 = degree of E_2.

ii) domain of i^{th} attribute of E_1 must be same as domain of i^{th} attribute of E_2.

If we take the Union between two Union compatible relations, then degree of output relation will remain same.

4> Set Difference

$$A - B = \{x \mid x \in A \text{ and } x \notin B\}$$

The difference betn two sets can be calculated provided they are union compatible.

5> Cartesian Product

This operation is used to combine the information from two different relations. Denoted by 'X'.

$$Deg(R_1 \times R_2) = Deg(R_1) + Deg(R_2)$$

Cardinality ← $|R_1 \times R_2| = |R_1| * |R_2|$

R_1

E-No.	E-Name	D-No.
1	A	10
2	B	20

R_2

Dep-No.	D-Name
10	MCA
20	CS

$R_1 \times R_2$

E-No.	E-Name	D-No.	Dep-No.	D-Name
1	A	10	10	MCA
1	A	10	20	CS
2	B	20	10	MCA
2	B	20	20	CS

Name of employee studying in CS dept ?

Ans. will be of that employee for whom D-No. = Dept No.

So Ans. will be B.

$\sigma_{D\text{-}name = CS}(R_1 \times R_2)$ [∵ Tuple I & III will be omitted]

$\sigma_{D\text{-}no. = Dept\,No}(\sigma_{D\text{-}name = CS}(R_1 \times R_2))$ [∵ Only Tuple IV will left]

$\Pi_{ename}(\sigma_{D\text{-}no. = Dept\,No.}(\sigma_{D\text{-}name = CS}(R_1 \times R_2)))$ [∵ o/p = B]

$\equiv \Pi_{ename}(\sigma_{D\text{-}name = CS}(R_1 \underbrace{\bowtie}_{\text{JOIN OPERATION}} R_2))$

∵ We have used '=' operator for join operation, so it will be called <u>Equi-Join Operation</u>.

Except '=' operator, all other comparison operator are used, it will be called <u>'θ join'</u> operation.

To remove the duplicacy of deptno. and D-No., if one attribute is removed, say, DeptNo. then it will be called <u>Natural Join Operation</u>.

<u>Relational Algebra Operations</u> 12/3/11

i) Set Intersection : $R \cap S = \{x \mid x \in R_1 \text{ and } x \in S\}$

ii) Rename :- Operator is ρ.

iii) Assignment

<u>Rename Operation</u>

It is used to give a name to the output of a relational algebra expression.

A relation-name by itself is a relational algebra expression

lecturenotes.in

Student

Roll No.	Name	Marks
1	A	10
2	B	15
3	C	12

Q:- Display the max ~~[illegible]~~ marks.

Select Max(Marks) from Student;

Steps:-

1) ρ_d (Student)

SELF JOIN

2) Student × ρ_d (Student)

Stud. Roll No.	Stud.Name	Stud.Marks	d.Roll No.	d.Name	d.Marks
1	A	10	1	A	10
1	A	10	2	B	15
1	A	10	3	C	12
2	B	15	1	A	10
2	B	15	2	B	15
2	B	15	3	C	12
3	C	12	1	A	10
3	C	12	2	B	15
3	C	12	3	C	12

3) $\sigma_{student.marks < d.marks}$ (Student × ρ_d (Student))

Student. Marks
10
[illegible]
12

4) $\Pi_{student.marks}$ ($\sigma_{student.marks < d.marks}$ (Student × ρ_d (Student)))

10
12

Lecturenotes.in

5) Π_{Mark} (Student)

10

15

12

6) 5) - 4)

15

The rename operators are useful in Self-join operations.

ASSIGNMENT OPERATION

Represented by $\leftarrow$

Ex:- $r_1 \leftarrow$ Student $\times$ ρ_d (Student)

$r_2 \leftarrow (\sigma_{stud.marks > d.marks}(r_1))$

$r_3 \leftarrow \Pi_{marks}$ (Student)

result $\leftarrow r_3 - r_2$

Lecturenotes.in

Extended Relational Algebra Operation

i) Generalized Projection

$$\Pi_{\underbrace{A_1, A_2, E_1}_{\text{Arithmetic Expression or Attributes}}} (\text{Relation Name})$$

Q1- Display the name and total annual income of each employee.

Sol^n:-

$$\Pi_{ename,\ sal*12+comm}(emp) -$$

ii) Outer-Join Operation

This operation is used to deal with missing information

R_1

E-Name	Street	City
A	X	L
B	Y	M
C	Z	N

Lecturenotes.in

R_2

E-Name	DeptNo	Sal
A	10	10,000
B	15	15,000
D	10	12,000

$R_1 \bowtie_{R_1.ename = R_2.ename} R_2$

O/p: $\Rightarrow$

E-Name	Street	City	DeptNo	Sal
A	X	L	10	10,000
B	Y	M	15	15,000

There is a loss of information as C & D are missing.

14/3/11

To counter-attack this loss of information, Outer-Join Operation is used. There are three types of Outer-join operation:-

i) Left Outer-Join ⟕
ii) Right Outer-Join ⟖
iii) Full Outer-Join ⟗

Left Outer-Join
The left outer-join will produce a result that will contain all the tuples of the relation i.e. present on the LHS of the join operation.

Assuming R_1 as left relation:-

$R_1 ⟕ R_2$

R_1.ename	R_1.street	R_1.city	R_1.deptno	R_1.sal
A	X	L	10	10,000
B	Y	M	15	15,000
C	Z	N	NULL	NULL

~~Assuming R_2 as~~

Right Outer-Join
The right outer join will produce a result that will contain all the tuples of relation present on RHS of join operation, placing NULL at appropriate places.

$R_1 ⟖ R_2$

Ename	Street	City	deptNo	Sal
A	X	L	10	10,000
B	Y	M	15	15,000
D	NULL	NULL	10	10,000

lecturenotes.in

Full-outer Join

It will contain all the tuples from both the relations

$$R_1 ⟗ R_2 = (R_1 ⟕ R_2) \cup (R_1 ⟖ R_2)$$

Ename	Street	City	deptNo	Sal
A	X	L	10	10,000
B	Y	M	15	15,000
C	Z	N	NULL	NULL
D	NULL	NULL	10	10,000

AGGREGATE OPERATION

i> SUM	iv> AVG
ii> MIN	v> COUNT
iii> MAX	vi> COUNT DISTINCT

These operations take a set of values as input and produce a single valu as output.

Ex:- Set -> <1, 2, 3, 4, 5>

SUM <1, 2, 3, 4, 5> = 15

MIN <1, 2, 3, 4, 5> = 1

MAX < " > = 5

AVG < " > = 3

COUNT < " > = 5

COUNT DISTINCT < " > = 5

From EMP Table:-

Min_{sal} (emp);

Sometimes, we may have to use these aggregate func^n onto a group of data these cases, we've to use the aggregate operators in addition to these func^n s. The aggregate operators can be represented as

$\mathcal{G}$ or $\mathcal{F}$

Calligraphic 'G'   Script F

Q:- Display department-wise min^m salary.

A:- Attribute used for grouping will appear as a left substript to $\mathcal{G}$.

$${}_{deptno}\mathcal{G}_{Min(sal)}(emp)$$

Q:- Job-wise max^m salary and department-wise min^m sal.

A:-

General Syntax:

$$\text{Group-by clause} \leftarrow {}_{G_1, G_2, \ldots G_n}\mathcal{G}_{F_1, F_2 \ldots F_n}(\text{Relation name})$$

$${}_{deptno, job}\mathcal{G}_{Min(sal), Max(sal)}(emp)$$

Relational Algebra ~~Expression~~ Operation for Modification of Database

The ~~schema~~ instance of database is modified if we perform insertion, deletion and updation.

i) Insertion

$$r \leftarrow r \cup (E)$$

where E -> Relational Algebra Exp.

We can insert a tuple into a relation by directly mentioning the tuple or sometimes a tuple can be selected from a particular table and inserted into another table.

R

ename	deptno	sal
A	10	500

insert

$$r \leftarrow r \cup ('A', 10, 500)$$

ii) Deletion

$$r \leftarrow r - (E)$$

Q: Delete the information about all the employees who are working as a clerk.

$$emp \leftarrow emp - (\sigma_{Job=clerk}(emp))$$

iii) Updation or Modification

a) Updating the Database

To perform this operation we have to use the generalized projection operator.

Syntax:

$$r \leftarrow \Pi_{F_1, F_2, \ldots F_n}(r)$$

where each F_i is an attribute if it isn't modified, otherwise it is an expression.

Depositor

C-Name A/C No. Bal.

$$Depositor \leftarrow \Pi_{C\text{-}Name,\ A/C No.,\ Bal \leftarrow Bal * 1.05}(\sigma_{Bal > 5000}(Depositor))$$

$$\cup$$

$$\Pi_{C\text{-}Name,\ A/C No.,\ Bal \leftarrow Bal * 1.04}(\sigma_{Bal <= 5000}(Depositor))$$

b) Outer-Union

It helps to find the union of two relations which are not fully union-compatible (Partially Compatible Relations).

$R_1(A, B, C)$

$R_2(A, B, D)$

Output Relation:- A B C D

Represented as R_1 outerunion R_2.

vi) Semi-join

Symbol :- $\ltimes$

It is used in Distributed Query Processing

R_1 R_2

$$\begin{tikzcd} \text{E-No.} & \text{E-Name} & \text{Dno.} \arrow[r, "\text{Join}", "\text{operation}"'] & \text{Depno} & \text{D-name} \end{tikzcd}$$

$R_1 \ltimes_{\substack{Dno. = \\ Depno.}} R_3$ $R_3 \leftarrow \pi_{depno}(R_2)$

$$\pi_{R_1}(R_1 \ltimes_{A=B} R_2) = \pi_{R_1}(R_1 \bowtie_{A=B} R_2)$$

• $\ltimes$ is not commutative. $R_1 \ltimes R_2 \neq R_2 \ltimes R_1$

$R_1 \ltimes R_2$ means joining attribute from R_2 is take to the side where R_1 is present. The join is peformed and the attributes of R_1 are projected in the o/p. Similary in $R_2 \ltimes R_1$ means joining attribute from R_1 is transfered to side where R_2 is present. The join is peformed and attributes of R_2 are projected in o/p.

Relational Calculus

The Relational Calculus is a Query language that helps in retrieval of information using a declarative statement (Predicates). Any query that can be expressed in relational algebra can also be expressed in Relational Calculus. So, we say that the expressive power of both the languages are same. The concept of expressive power leads to a concept called as Relationally Complete Language. A language 'L' is said to be Rellationally complete if any query that can be expressed through relational calculus can also be expressed by language 'L'.

The Relational Calculus Languages are :-

i) Tuple Relational Calculus

ii) Domain Relational Calculus

lecturenotes.in

19/3/11
D.E

Functional Dependencies and Normalization of Relational Databases

Aim:- To design good relational databases
The goodness of a relation can be define at two levels :-
i) Conceptual Level
ii) Physical Level (Implementation Level)

At the conceptual level, the goodness of a relation specifies that the attributes that have been grouped to form a relation has a well-defined meaning and the user has understood the meaning properly inorder to formulate correct queries.
At the implementation level, goodness defines how the data can be stored at and updated efficiently.
There are two different approaches to design a relational database:-
i) Bottom-up approach / Design by synthesis
ii) Top-down approach / Design by analysis

In bottom-up approach, the individual attributes are identified and then grouped to form a relation. This method isn't very popular.
The very popular approach for designing of relational databases is the Top-down approach, where we start with a relation (Base / Universal Relation) and then analyse that relation which results in the decomposition of the relation to satisfy the goodness properties.
Characteristics of a good relation/Informal parameters for a quality relation:-
1) Semantics of the attributes
2) Reducing redundancy in values of tuples
3) Reducing null values in tuples
4) Guarantee that no spurious tuples are generated.
irrelevant

Guidelines for a good database design:-
1) The attributes that have been grouped to form a relation should've a well defined meaning. We shouldn't mix the attributes from multiple entity sets ~~with~~ or relationship sets.

2) Try to design a relation so that there is min^m redundancy in the tuple values and at the same time, there shouldn't be insertion, deletion and update anamoly.
Insertion anamoly is a tuple can't be inserted in a table, because primary key field values aren't available.
Deletion anamoly is, if a tuple is deleted, then there is loss of information.
Update anamoly is, if the data isn't updated at all the places then it may lead to inconsistency.

3) Do not place the attributes in the base relation whose values are very often NULL
When an attribute contains NULL values, then applying aggregate func^n's onto these attributes may result in incorrect o/p.

4) Design the relations in such a way that they should be joined by the primary key - foreign key attributes in a way that guarantees no spurious tuples will be generated.

Functional Dependency 21/3/11.

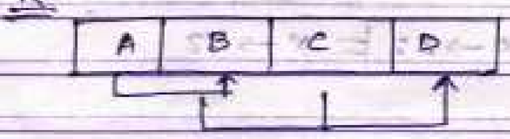
Design by Analysis

The basis of design by analysis approach to design a relational database is the functional dependency (fd).
Consider a Universal Relational Schema R with attributes: $A_1, A_2 \ldots A_n$.
A functional dependency (fd) denoted by $x \rightarrow y$ (x determines y) where X and Y are the subsets of attributes of R, defines a constraint on the tuples of R, such that for any two tuples t_1 & t_2, if $t_1[x] = t_2[x]$ then $t_1[y] = t_2[y]$.

- $x \rightarrow y$ (y is functionally dependent on x) where x is called LHS and y is called RHS of functional dependency.
- If x is the candidate key of 'R', then $x \rightarrow y$ holds True.
- Functional Dependency is the property of a relation schema.

$R = (A, B, C, D)$

$F = \{A \rightarrow B,\ BC \rightarrow D\}$



Inference Rules

Given a relation schema, the domain experts provide us ~~with~~ those functional dependencies that are logically obvious. The set of functional dependencies that are applicable on a particular relation schema is represented by upper case alphabets (F). Given F, we can derive new functional dependencies out of F using the inference rules popularly called as Armstrong's Axioms.

$$F \xrightarrow{A.A} f_1 = \{\text{New f.d's}\}$$

$$F \cup f_1 = F^+ \rightarrow \text{closure of } F.$$

Armstrong's Axioms

I) Reflexive Rule

$\uparrow$ R

$$X \supseteq Y \; ; \vDash X \rightarrow Y \qquad \text{Trivial Fd.}$$

such that

containing or super-set.

II) Augmentation Rule

$\uparrow$ A

$$X \rightarrow Y ; \vDash XZ \rightarrow YZ$$

Here Z is augmented both the sides.

III) Transitivity Rule

$\uparrow$ T

$$X \rightarrow Y;\ Y \rightarrow Z\ ; \vDash X \rightarrow Z$$

These three are basic axioms and others are derived from these 3 axioms. These 3 axioms are called RAT Rule.

IV) Decomposition Rule

$X \rightarrow YZ ;\ \models X \rightarrow Y$ or $X \rightarrow Z$

V) Union Rule

$X \rightarrow Y$ or; $X \rightarrow Z ;\ \models X \rightarrow YZ$

VI) Pseudo transitive Rule

$X \rightarrow Y ;\ WY \rightarrow Z ;\ \models WX \rightarrow Z$

Imp

Q:- State and prove Armstrong's Axioms.

Q:- Prove that A.A are sound and complete.

A:- Only justify giving defn of sound and complete.

Relational Algebra Continued ...

Division Operation

- Denoted by ÷, suited for queries that include phrase 'for all'.

Ex:-

$r_1 = \Pi_{ename} (\sigma_{deptno = 20} (emp))$

$r_1 = \Pi_{branch\text{-}name} (\sigma_{branch\text{-}city = \text{"Brooklyn"}} (branch))$

branch name
Brighton
Downtown

$r_2 = \Pi_{c\text{-}name,\ branch\text{-}name} (depositor \bowtie account)$

c-name	branch-name
H	Perryridge
J	Downtown
BJ	Brighton
K	Brighton

lecturenotes.in

branch	account	depositor
branch-name	acc-no.	c-name
branch-city	branch-name	acc-no.
assets	balance	

customer
- c-name
- c-street
- c-city

We need to find customers who appear in r_2 with every branch-name in r_1.

$\Rightarrow \Pi_{c\text{-}name,\ branch\text{-}name}$ (depositor $\bowtie$ account) $\div$

$\Pi_{branch\text{-}name}$ ($\sigma_{branch\text{-}city = Brooklyn}$ (branch

So, o/p will be ~~[illegible]~~ schema (c-name) and tuple (J)

- Let $r(R)$ and $s(S)$ be relations, such that $S \subseteq R$, then $r \div s$ is a relation on schema $R-S$ (schema contain all attributes of R that are not in schema S)
- A tuple t is in $r \div s$ if:-

I) t is in $\Pi_{R-S}(r)$

II) for every t_s in s, there is t_r in r satisfying:-

a) $t_r[S] = t_s[S]$

b) $t_r[R-S] = t$

$$\boxed{r \div s = \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))}$$

Ex:-

P(P)

A	B
a_1	b_1
a_1	b_2
a_2	b_1
a_3	b_1
a_4	b_2
a_5	b_1
a_5	b_2

Q(Q)

B
b_1
b_2

Closure of X under F 22/3/11

Find out F^+ from F by using inference rule is a time consuming and tedious process. So instead of applying IR's we'll try to find out the closure of each attribute that occurs as a LHS in F.

Algorithm:-

```
1) $X^+ = X$;
2) Repeat
      for each fd $Y \to Z$ in F
      if $X^+ \supseteq Y$, then $X^+ = X^+ \cup Z$
   until there are no changes to $X^+$;
```

Ex:- F = {SSN → Ename ∵ Y → Z

Pno. → (Pname, Plc) ∵ Y → Z

(SSN, Pno) → Hours ∵ Y → Z }

All is treated separately and taken into consideration.

Soln:- SSN^+ = {SSN, Ename}

Pno^+ = {Pno, Pname, Plc}

$(SSN, Pno)^+$ = {SSN, Pno, Hours, Ename, Pname, Plc}

Lecturenotes.in

Closure of an attribute or closure of combination of attributes that contains the entire attributes of relation schema R, is the key of the relation.

Ex:-

$R = (A, B, C, D, E)$ $\{CD = \{C, D\}$

$F = \{AB \rightarrow C, CD \rightarrow E, DE \rightarrow B\}$

Find Key?

Sol^n:- $(AB)^+ = \{AB, C\}$ or $\{A, B, C\}$

$(CD)^+ = \{C, D, E, B\}$

$(DE)^+ = \{D, E, B\}$

Key is ACD or ABD.

Equivalence of two sets of fd's

Two sets of functional dependencies F and G are equivalent if $F^+ = G^+$.

Practically, $F \equiv G$ if F covers G, and G covers F.

F covers G :- F is said to cover G if for each functional dependency $X \rightarrow Y$ in G, find X^+ with respect to F and check if X^+ contains Y. If yes, then F is said to cover G.

Ex:- $F = \{A \rightarrow C, AC \rightarrow D, E \rightarrow AD, E \rightarrow H\}$

$G = \{A \rightarrow CD, E \rightarrow AH\}$

Check if $F \equiv G$.

Sol^n:- $A^+ = \{A, C, D\}$

$E^+ = \{E, \{A, D, H\}, C, D\}$

Thus F covers G

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

$(AC)^+ = \{AC, D\}$

$E^+ = \{E, A, H, C, D\}$

~~$E^+ = \{E,$~~

Thus G covers F

So $F \equiv G$

∵ F covers G and G covers F, both hold F and G are equivalent.

Minimal Set of Functional Dependency.

Given set of fd 'F' is minimal if it satisfies the following 3 conditions:-

I) In each fd, RHS must contain only one attribute

By decomposition Rule $\left\{ \begin{array}{l} A \to CD \times \\ A \to C \text{ or } A \to D \checkmark \end{array} \right.$

II) A fd X determines A ($X \to A$) can't be replaced by ($Y \to A$) where $Y \subseteq X$. and still have the set of fd $\equiv$ original set.

• on LHS, we should've @least two attributes.

$F = \{AC \to D\}$

$A \to D$ or $C \to D$ ×

III) We can't ~~replace~~ eliminate $X \to A$ from F and still have set of fd $\equiv F$.

$F = \{A \to C, AC \to D\}$ ✓

We can't delete any one fd like $F_1 = \{AC \to D\}$ ×

Everytime we must check $F_1 \equiv F$.

If $F_2 = \{A \to C, A \to D\}$
or $F_3 = \{A \to C, C \to D\}$
| Equivalency is checked, then it is possible.

After performing these steps, we get Reduced Set of Functional Dependency.

25/3/11.

Normalization Process

Normalization is to enforce a relation to certain test to certify in which normal form it is in. The Normalization procedure was proposed by Codd. He suggested three normal forms:-

i) 1st Normal Form (1NF)

ii) 2nd Normal Form (2NF)

iii) 3rd Normal Form (3NF)

Later on a stronger form of 3rd Normal Form was proposed by BOYCE and CODD, and that normal form is called Boyce Codd Normal Form (BCNF).

Normalization is the process of analyzing a relation schema wrt its set of functional dependencies associated with it so as to have minimum redundancy and no insertion, deletion and updation anomalies.

The process of Normalization is correct if it satisfies the following three properties:-

i) Non-Additive Join Property (Lossless Join Property)

ii) Dependency Preservation Property

iii) Attribute Preservation Property

lecturenotes.in

Normal Forms

1st Normal Form (1NF)

A relation schema 'R' is in 1st Normal Form if every attribute contains only atomic values

2nd Normal Form (2NF)

It is based on concept of full functional dependency (FFD).

A fd $X \to Y$ is a ffd if the removal of any attribute 'A' from X the fd doesn't hold.

$(X - \{A\}) \to Y$ doesn't hold.

Ex:- (SSN, Pno.) → Hours

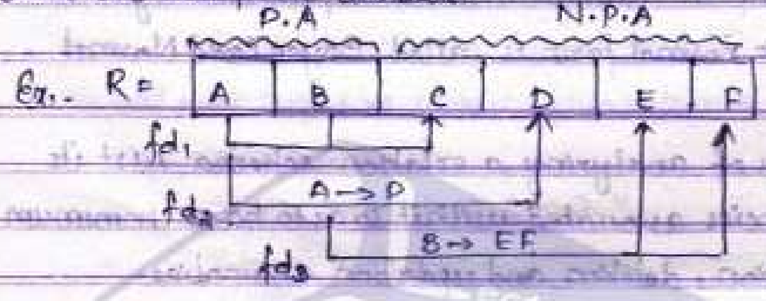
(SSN, Pno. − {Pno.}) → Hours won't hold.

• On LHS of fd, there must be @least two attributes.

If on removal of any attribute 'A' from X, the fd still exists, then it is called as Partial Function dependency (PFD).

2NF:- A relation schema 'R' is said to be in 2NF if no non-prime attribute is partially dependent on the key of 'R'.

Any attribute which is a part of a candidate key is a prime-attribute and others are non-prime attribute.



AB → C

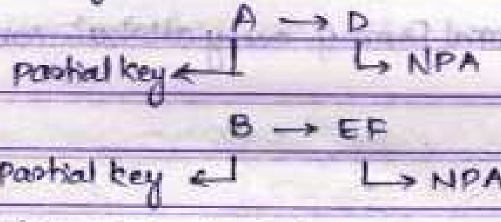
$(AB)^+ = A, B, C, D, E, F$

A → D

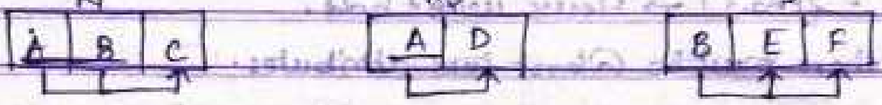
$(A)^+ = A, D.$

$(B)^+ = B, E, F.$

AB is the key.



fd2 and fd3 violate the definition of 2NF because in fd2, a non-prime attribute (D) is partially dependent on the key-attribute (A) and similarly two non-prime attributes (EF) are partially dependent on key attribute (B) in fd3. Hence relational schema isn't in 2NF. So we decompose R to R_1, R_2 and R_3.



26/3/18

3rd Normal Form (3NF)

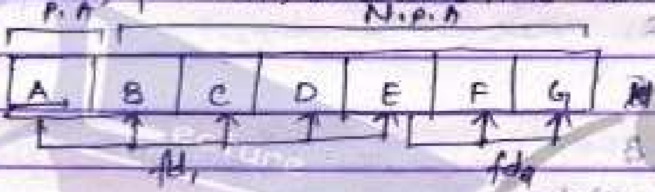
The 3NF is based on transitive dependency. A functional dependency (X→Y) in a relational schema 'R' is said to possess transitive dependency if there exists an attribute set Z (neither the key nor prime attribute) such that two fd's X→Z and Z→Y hold in R.

A relation schema 'R' is said to be in 3NF if no non-prime attribute is transitively dependent on key of R.

OR

A relation schema 'R' is said to be in 3NF if for every non-trivial fd X→Y either X is the key of R or Y is a prime-attribute.

• A fd X→Y is trivial, if X ⊇ Y, otherwise it is non-trivial.

Ex:-


$\boxed{A} \rightarrow \boxed{BCDE}$ must be prime.

Candidate key or super key.

$A^+ = A, B, C, D, E, F, G$

$E \rightarrow FG$

$E^+ = E, F, G$

A will be the key.

∵ In the key, there is no question of partial dependency. So, R is in 2NF already.

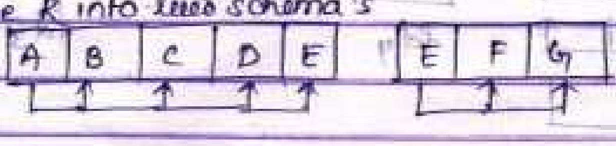
$A \rightarrow E$

$E \rightarrow F$

∴ $A \rightarrow F$, where A is transitively depending upon F.

So, given schema isn't in 3NF.

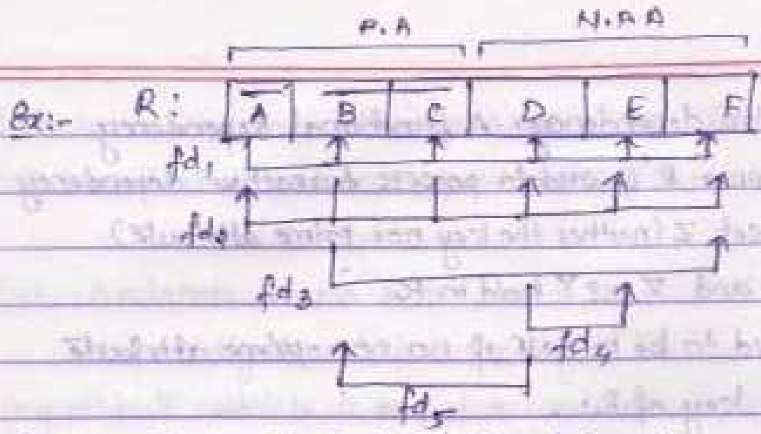
So, we decompose R into two schema's



Lecturenotes.in

11/4/26

Ex:-



Decompose R into 2NF, 3NF and BCNF.

fd1 : A → BCDEF

fd2 : BC → A

BC → DEF

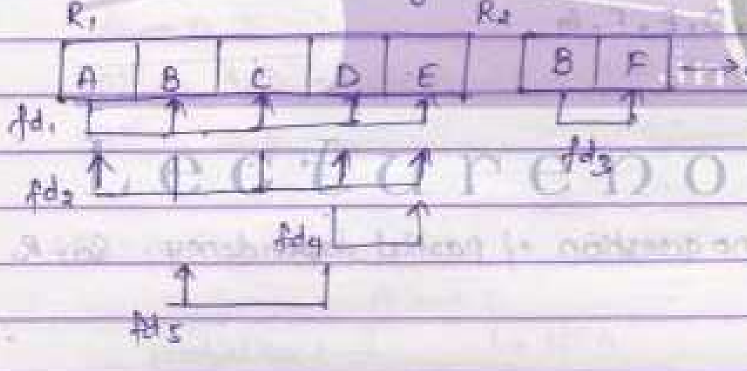
fd3 : B → F

fd4 : D → E

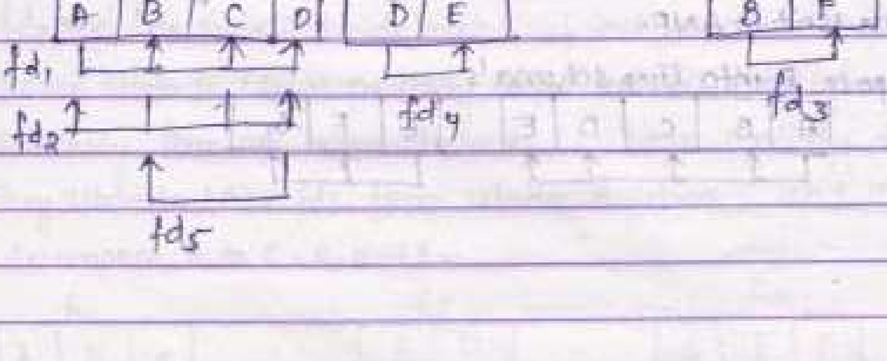
fd5 : D → B

Solⁿ: A and BC is the key

B → F violating 2NF property.



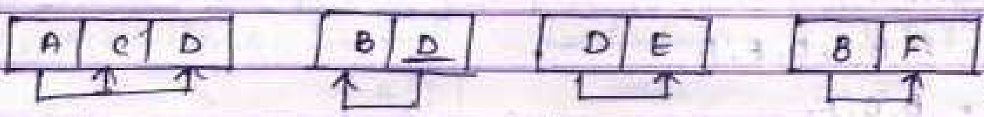
fd4 is violating the 2nd definition of 3NF.



BCNF

It is the stronger form of 3NF. A relation schema 'R' is said to be in BCNF if for a non-trivial fd $X \to Y$, X is the key of R.

The relation schema R_{11} isn't in BCNF because in fds where $D \to B$, D isn't the key of R_{11}.



Every relation in BCNF is in 3NF but every relation in 3NF isn't in BCNF.

At the BCNF level, some dependencies are lost. So, if we roll back from a higher normal form to a lower normal form, it is called as Denormalization.

28/8/11

Q:- R = {A, B, C, D, E, F, G, H, I, J}

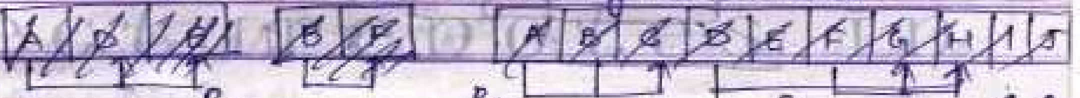
F = {$AB \to C$, $A \to DE$, $B \to F$, $F \to GH$, $D \to IJ$}

Decompose R into 2NF, 3NF and BCNF.

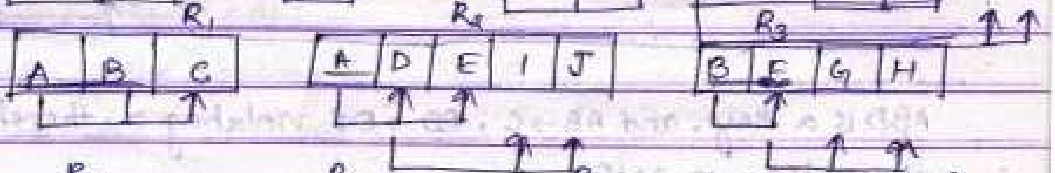
Sol^n:- $(AB)^+ = A, B, C, D, E, F, G, H, I, J$

AB is the key.

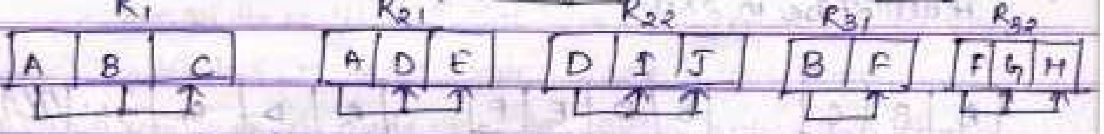
$A \to DE$ and $B \to F$ violating the 2NF rule.



2NF:



already in 3NF



Already in BCNF

For previous ex:-

Check for dependency preservation :-

G = { A → CD, D → E, D → B, B → F }

F = { A → BCDEF, BC → A, BC → DEF, B → F, D → E, D → B }

G covers F

A^+ = A, B, C, D, E, F

BC^+ = B, C, F.

So, G isn't covering F, then F also will n't cover G.

As BC doesn't containing A, D, E, F

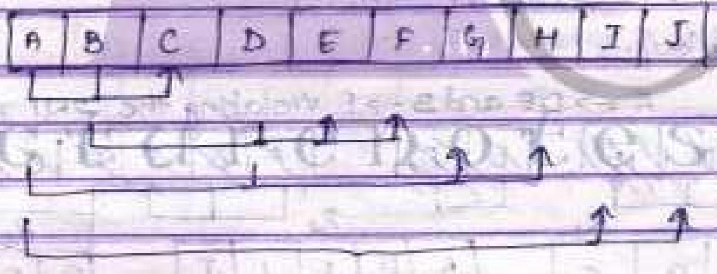
Thus F ≢ G.

Therefore, dependency preservation property isn't preserved.

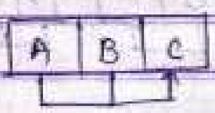
Q:- R = { A, B, C, D, E, F, G, H, I, J }

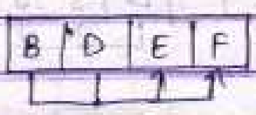
F = { AB → C, BD → EF, AD → GH, A → I, A → J }

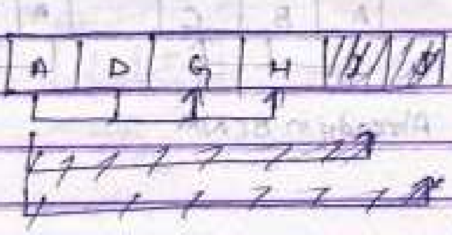
Sol^n :- Key will be ABD.



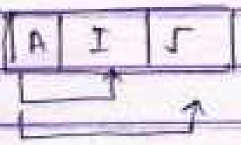
ABD is a key, and AB → C, BD → EF violating it, therefore it wouldn't be in 2NF.







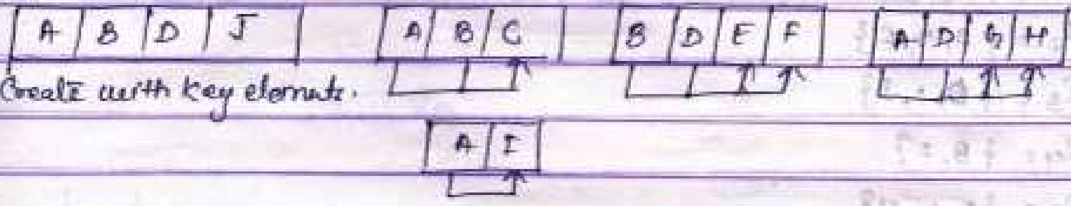
Also in 3NF.



Now If F = {AB → C, BD → EF, AD → GH, A → I}

Key will be ABDJ

R_1



Leaving R_1, discarding J find out key and find relations for the remaining attributes.

29/3/11

Algorithm to check loss-less Join / Non-additive join property.

Input : Orignal Relational Schema 'R', the set of dependencies F applicable on R and the decomposition set D(R_1, R_2 ..... R_n).

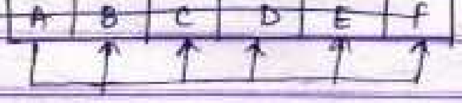
Step I : Create the initial matrix 'S' with no. of rows equal to no. of elements in set D and no. of columns equal to the no. of attributes in R.

Step II : For each row i, representing the relation R_i, for each column j, representing the attribute A_j if R_i contains the attribute A_j, then set $S(i,j) = a_j$;

Step III : Repeat the following steps until there are no changes to S.
For each fd X → Y in F
for all the rows in S that have the same values for the attribute X, make the Y values same in those rows

Step IV : If a row is entirely made up of 'a' symbols only, then that decomposition is lossless otherwise it is lossy.

Q:- ~~R~~



Q: $R = \{A, B, C, D, E, F, G, H, I, J\}$

$F = \{AB \to C, A \to DE, B \to F, F \to GH, D \to IJ\}$

$D = \{R_1, R_2, R_3, R_4, R_5\}$

$R_1 = \{A, B, C\}$

$R_2 = \{A, D, E\}$

$R_3 = \{D, I, J\}$

$R_4 = \{B, F\}$

$R_5 = \{F, G, H\}$

	1	2	3	4	5	6	7	8	9	10
	A	B	C	D	E	F	G	H	I	J
R_1	$\underline{a_1}$	$\underline{a_2}$	a_3	a_4 ↑	a_5 ↑	a_6 ↑	a_7 ↑	a_8 ↑	a_9 ↑	a_{10} ↑
R_2	$\boxed{a_1}$			a_4	a_5				a_9 ↑	a_{10} ↑
R_3				$\boxed{a_4}$					a_9	a_{10}
R_4		$\boxed{a_2}$				a_6	a_7 ↑	a_8 ↑		
R_5						$\boxed{a_6}$	a_7	a_8		

lecturenotes.in

Since Row R_1 is consisting of 'a' symbols only, the given decomposition is lossless.

Q:- $R = \{A, B, C, D, E, F, G, H, I, J\}$

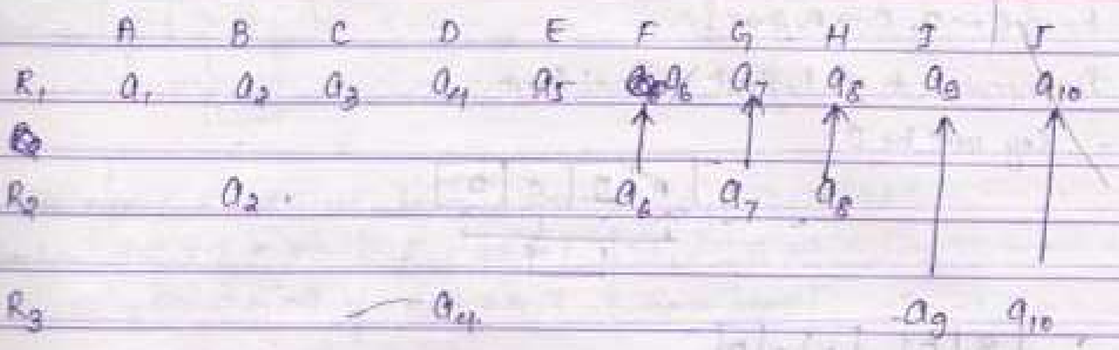
$F = \{AB \to C, A \to DE, B \to F, F \to GH, D \to IJ\}$

$D = \{R_1, R_2, R_3\}$

$R_1 = \{A, B, C, D, E, \bullet\}$

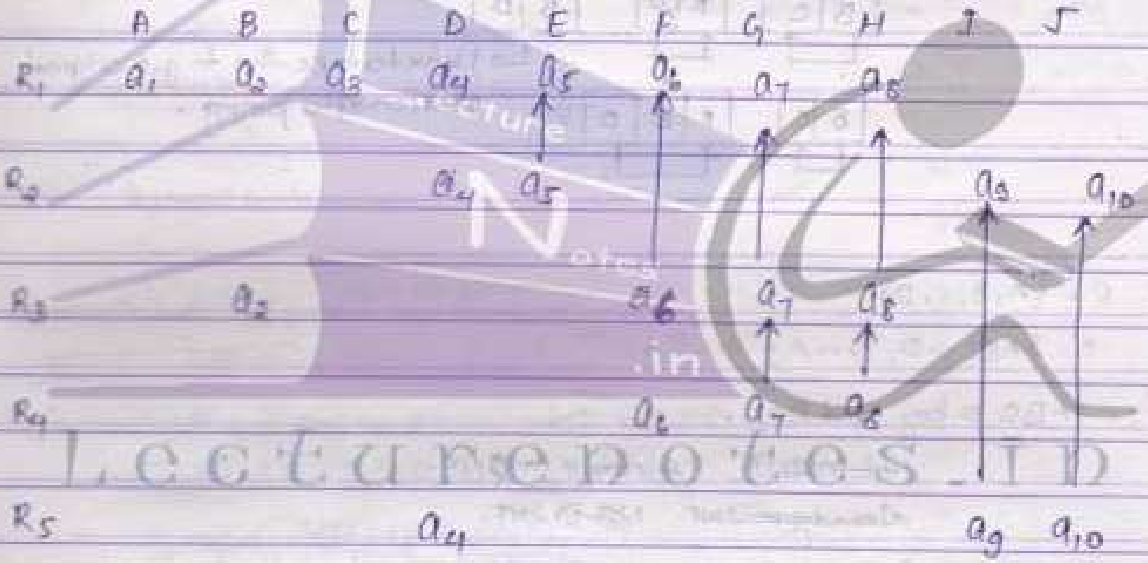
$R_2 = \{B, F, G, H\}$

$R_3 = \{D, I, J\}$



Lossless

Q:-



Lecturenotes.in

Lossy

Q:-

A	B	C	Table
10	b_1	C_1	1
10	b_2	C_2	2
11	b_4	C_1	3
12	b_3	C_4	4
13	b_1	C_1	5
14	b_3	C_4	6

Which of these are correct.

$A \to B$, $B \to C$, $C \to B$, $B \to A$, $C - A$

✗ ✓ ✗ ✗ ✗

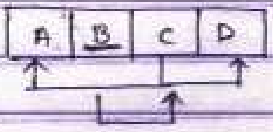
1/4/11

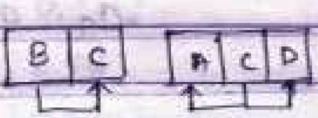
Q:- R = {A, B, C, D}

F = { C → D, C → A, B → C }

Decompose R to its highest normal form.

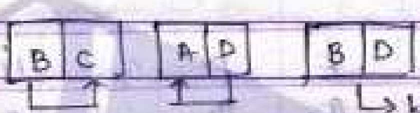
Sol^n:- Key will be B.



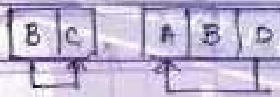


Q:- F = { B → C, D → A }

Sol^n:- BD



↳ We've to take this to get origina
relation after joining.



Q:- R = {A, B, C, D}

F = { ABC → D, D → A }

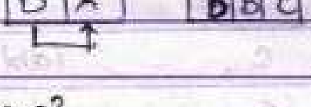
Sol^n:- ABC is key and BCD.

~~Not in 2NF~~ Already in 2NF.

~~already in 2NF~~. ~~Not in 2NF~~

ABC → D, D → A, So already in 3NF

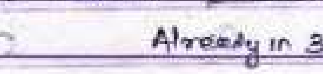
D → A violating BCNF.



Q:- R = {A, B, C, D}

F = { A → B, BC → D, A → C }

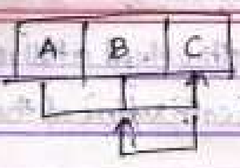
Sol^n:- A ~~[illegible]~~ is the key



Already in 2NF

Already in 3NF

Lecturenotes.in

Q:- 

Decompose R such that lossless join property is satisfied?

Sol^n:- $R_1 = \{A, B\}$, $R_2 = \{A, C\}$ | $R_1 = \{B, A\}$, $R_2 = \{B, C\}$ | $R_1 = \{C, A\}$, $R_2 = \{C, B\}$

	A	B	C
R_1	a_1	a_2	[illegible]
R_2	a_1		a_3

~~lossless~~ Lossy

	A	B	C
R_1	a_1	a_2	[illegible]
R_2		a_2	a_3

~~lossless~~ Lossy.

A	B	C
a_1	a_2	a_3
	a_2	a_3

Lossless

D_3 will be lossless.

Q:- $R = \{A, B, C, D\}$

$F = \{AB \to C, AB \to D, C \to A, D \to B\}$

Sol^n:- ~~AB is the key and CD, AD, BC~~

AB, BC, CD, AD will be the key

already in 2NF.

R_1 $\boxed{A \mid \underline{C}}$ R_2 $\boxed{B \mid \underline{D}}$ R_3 $\boxed{C \mid D}$ $\longrightarrow$ BCNF

$$\underbrace{\underset{R_4}{\boxed{A \mid B \mid C}} \quad \underset{R_5}{\boxed{A \mid B \mid D}}}_{\text{Optional}}$$

If the dependency preservation property has to be satisfied, then we've to add two more relations R_4 & R_5

4/4/11

Multivalued Dependency (MVD) and Fourth Normal Form

Whenever a relation schema R contains two independent multi-valued attributes, then as a consequence of 1NF, we've to create one tuple for each value of the attribute.

A multivalued dependency represented as $X \twoheadrightarrow Y$ (X multideterminer Y) where X and Y are sets of attributes of relation schema R, defines a constraint on the tuples of R such that for any two tuples t_1 and t_2, if $t_1[x] = t_2[x]$, then two other tuples t_3 and t_4 exists on r(R) such that following conditions are satisfied :-

i) $t_1[x] = t_2[x] = t_3[x] = t_4[x]$

ii) $t_1[y] = t_3[y]$ and $t_2[y] = t_4[y]$

iii) $t_1[z] = t_4[z]$ and $t_2[z] = t_3[z]$

where $Z = R-(X \cup Y)$

①

X	Y	Z
1	A	17
1	A	15
1	B	17
1	B	15

X	Y	Z
1	A	17
1	B	17
1	A	15
1	B	15

X	Y	Z
1	A	17
1	B	15
1	A	15
1	B	17

$\therefore X \twoheadrightarrow Y$

Ex:- ②

X	Z	Y
1	17,15	A,B

X	Z	Y
1	17	A
1	17	B
1	15	A
1	15	B

X	Z	Y
1	17	A
1	15	B
1	17	B
1	15	A

$\therefore X \twoheadrightarrow Z$

Whenever $X \twoheadrightarrow Y$, by symmetry we can prove that $X \twoheadrightarrow Z$. In general,
$X \twoheadrightarrow Y/Z$

- A Multivalued Dependency $X \twoheadrightarrow Y$ is said to be trivial, if Y is the subset of X or $(X \cup Y) = R$. If none of the above conditions are satisfied, it is called Non-trivial MVD.

4NF

A relation schema R is said to be in 4NF wrt a set of dependencies F (which contains both fd and MVD) if for any non-trivial MVD in F^+, X is the superkey of R.

For above ex② XYZ is the key
$X \twoheadrightarrow Y/Z$ is non-trivial.
and also X is not the key of R here.
So, $X \twoheadrightarrow Y/Z$ isn't in 4NF.

To bring it into 4NF,

$R_1 = (X \cup Y)$

$R_2 = (R - Y)$

$R_1 = (1, A, B) \longrightarrow$

X	Y
1	A
1	B

$R_2 = (1, 15, 17) \longrightarrow$

X	Z
1	15
1	17

Both are trivial, but X is not the key. So, this will be in 4NF.

Decomposing R into R_1 and R_2 converts the non-trivial MVD to trivial MVD. Hence R_1 and R_2 are in 4NF and this decomposition preserves the Lossless Join Property.

lecturenotes.in

5/4/11

TRANSACTION CONCEPT

A transaction is a collection of operations that defines a logical unit of work which accesses and possibly updates the data items present in the database.

main()
Begin transaction → { → delimiters
} → End transaction

Every transaction starts with delimiters begin transaction and terminates with end transaction delimiters. The set of operations within these two delimiters constitute one transaction.

Properties of a Transaction :-

i) Atomicity
ii) Consistency
iii) Isolation
iv) Durability

} ACID Property

Atomicity

Either all the steps of transaction should execute till completion or none of them should execute.

Consistency

The execution of the transactions in isolation should preserve the database consistency.

Isolation

For a pair of transactions t_i and t_j, t_i assumes that it has started its execution only after t_j has completed and similarly t_j assumes that it has started only after t_i is complete.

Durability

The changes made by a transaction to the database persists even if there is a failure. i.e, changes made by a transaction are permanent.

A transaction can access a data-item 'x' by executing read(x) and similarly the transaction can update the value of X by executing write(X).

Transaction States,

A transaction passes through various states during its lifetime. The state of a transaction is defined by the current activity it is performing. At a particul instant of time, a transaction can be in ~~one~~ one of following states:-

i> Active State
ii> Partially Commited state
iii> Failed State
iv> Commit State
v> Abort State

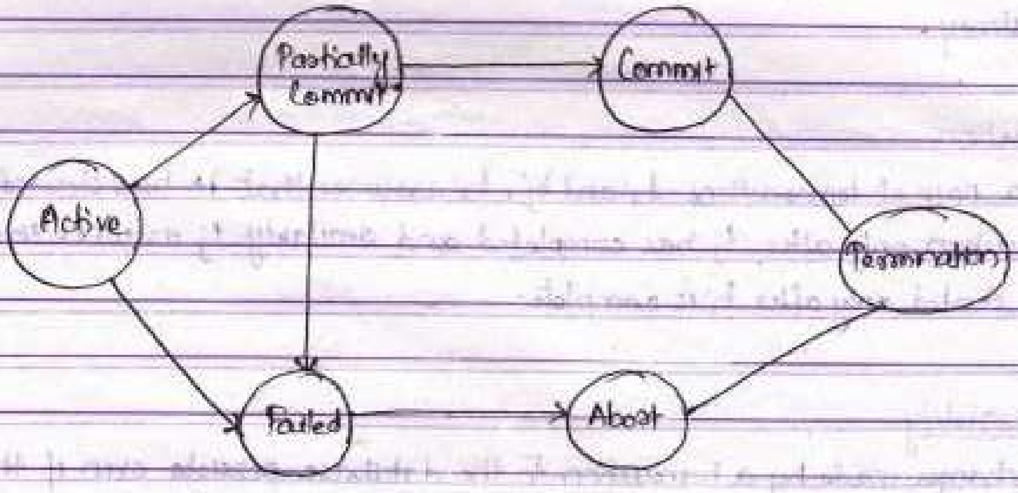
Active State : Transaction is under execution.

Partially Committed Set : When the last step of transaction is executed, the transaction enters the partially Commit State.

Failed State : When the transaction can't be continued with its normal execution, then it enters Failed State.

Commit State :- Transaction has successfully completed its execution.

Abort State : When the transaction fails, we forcefully terminate the transaction and the changes made by failed transaction on the base are re-initialized to its original state.



Transaction States Transaction Diagram.

A transaction will terminate either when it commits or when it is aborted.

Transaction Executions

There are three possible ways in which a transaction can be executed:-

i) Serial Execution

ii) Parallel Execution

iii) Concurrent Execution.

In Serial execution, 2nd transaction can begin its execution only after first transaction has completed. This is possible on a Uniprocessor System.

In parallel execution, two transactions can start its executions exactly at same instant of time. For this, we require more than one processor.

In concurrent execution, execution of the second process can begin even before the process has completed its execution. Let us consider two transactions t_1 and t_2. In concurrent execution, the CPU executes some instructions of transaction t_1, then move to second transaction and execute its instructions for some-time and again it comes back to first transaction. This cycle is repeated until all the instructions of both the transactions are executed. This process is called as Context Switching

Advantages :-

i) Optimum utilization of resources.

ii) Average waiting time is less.

iii) Response time is shorter.

iv) Through put is more (No. of processes executed in one unit of time).

Let us consider two transactions t_1 and t_2 where t_1 performs transfer Rs.50 from A/c A to A/c B and similarly t_2 transfers 10% of balance from A to B.

T_1	T_2
read (A)	read (A)
A = A-50	temp = 0.1 * A
W(A)	A = A - temp
r(B)	W(A)
B = B+50	r(B)
W(B)	B = B+ temp
	W(B)

The order in which the instruction of transaction t_1 and t_2 are executed is called as <u>Schedule</u>. The possible serial schedules are :-

Let A=100, B=100

S_1

T_1	T_2
r(A) : 100	
A = A-50 : 50	
W(A) : 50	
r(B) : 100	
B = B+50 : 150	
W(B) : 150	
TIME GAP	
	r(A) : 50
	temp = 0.1 * A : 5
	A = A - temp : 45
	W(A) : 45
	r(B) : 150
	B = B+temp : 155
	W(B) : 155

S_2

T_1	T_2
	r(A)
	temp = 0.1 * A
	A = A - temp
	W(A)
	r(B)
	B = B + temp
	W(B)
r(A)	
A = A-50	
W(A)	
r(B)	
B = B+50	
W(B)	

Schedule S_1 keeps the database in consistent state

Schedule S_2 also keeps the database in consistent state.

In general, if system consists of n no. of transactions, then we can generate n! no. of valid Serial Schedule

Concurrent Schedule

S_3

T_1	T_2
r(A) : 100	
A = A-50 : 50	
w(A) : 50	
	r(A) : 50
	temp = 0.1 * A : 5
	A = A-temp : 45
	w(A) : 45
r(B) : 100	
B = B+50 : 150	
w(B) : 150	
	r(B) : 150
	B = B + temp : 155
	w(B) : 155

Consistent

S_4

T_1	T_2
r(A) : 100	
A = A-50 : 50	
w(A)	
	r(A) : 50
	temp = 0.1 * A : 5
	A = A-temp : 45
	w(A) : 45
	r(B) : 100
w(A) : 45	
r(B) : 100	
B = B+50 : 150	
w(B) : 150	
	B = B+temp : 155
	w(B) : 155

Not Consistent

All concurrent Schedules don't keep the database in the consistent state. The concurrent execution of the transactions have to be carried out in a controlled environment. If a system consists of 'n' no. of transactions, we'll have more than 'n!' no. of ~~other~~ concurrent schedules, out of which only a few of them are keeping the database in Consistent state. So, our next aim is to determine which concurrent Schedules will ~~give~~ keep the database in consistent state. We can determine whether a concurrent schedule will keep the database in the consistent state can't through ~~Serialization~~ Serializability. There are two forms of Serializability:-

i> Conflict Serializability

ii> View Serializability

Conflict Serializability

Serializable :- A concurrent schedule S is serializable if it produces the same o/p as that of a valid serial.

Obtained from S_3 by considering only the read and write opⁿ. Consider the consecutive instructions I_i and I_j of transactions T_i and T_j respectively. If the consecutive instructions are the operations on two different data items, then we can swap the order of execution of I_i and I_j, but if I_i and I_j are two operations on to the same data item then four possibilities arise:-

i> $I_i = r(Q)$, $I_j = r(Q)$ we can swap the order of execution.

ii> $I_i = r(Q)$, $I_j = W(Q)$ order of execution can't be swap.

iii> $I_i = W(Q)$, $I_j = r(Q)$ order of execution can't be swap.

iv> $I_i = W(Q)$, $I_j = W(Q)$ order of execution can't be swap.

Two consecutive instrⁿ I_i and I_j are conflicting if @least one of the instrⁿ is a write instrⁿ.

Defn:-

A concurrent schedule S is conflict Serializable, if it is conflict equivalent to a serial schedule S'.

$$\begin{tikzcd}
\text{(C.S)}S \arrow[rr, dash, bend right, "\text{Conflict Equv.}"'] \arrow[u, phantom, "\uparrow\ \text{Conflict Serializability}"] & & S'\text{(S.S)}
\end{tikzcd}$$

Conflict Equivalent :- A concurrent Schedule S is conflict equivalent to a serial schedule S', if we can obtain S' out of S by swapping the order of execution of the non-conflicting instructions.

Even if the schedule S_6 keeps the database in consistent state, we can't convert it into a serial schedule and hence we conclude that the schedule isn't conflict equivalent to any of the serial schedules. So, instead of considering only the read and write operation, we'll also consider the intermediate operations which will result in a new form of seriali-zability known as View Serializability

S_6 is conflict equv. of S_7.

Lecturenotes.in

S_6

T_1	T_2
r(A)	
A = A-50	
w(A)	
	r(B)
	B = B-10
	w(B)
r(B)	
B = B+50	
w(B)	
	r(A)
	A = A+10
	w(A)

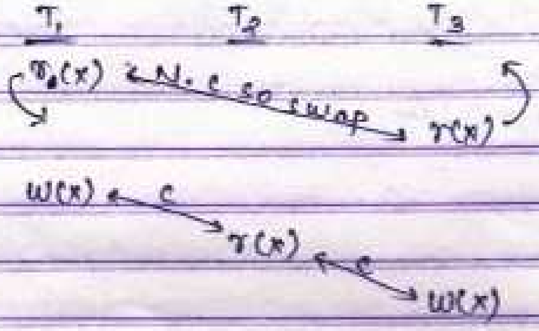
S_7

T_1	T_2
r(A)	
A = A-50	
w(A)	
r(B)	
B = B+50	
w(B)	
	r(B)
	B = B-10
	w(B)
	r(A)
	A = A+10
	w(A)

15/4/11

Determine if the following Schedules are conflict Serializable :-

1) $r_1(x), r_3(x), w_1(x), r_2(x), w_3(x)$



No, this can't be converted into serial schedule.

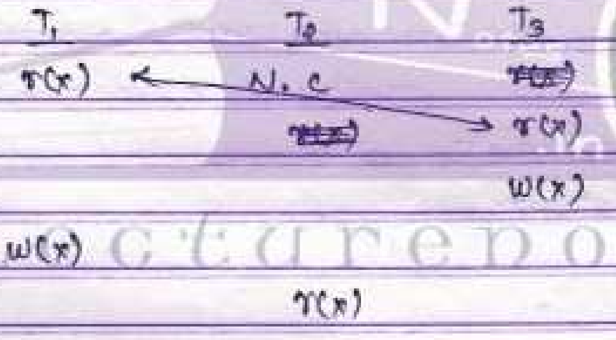
Non-conflict serializable.

ii) $r_3(x), r_3(x), w_3(x), w_1(x), r_2(x)$

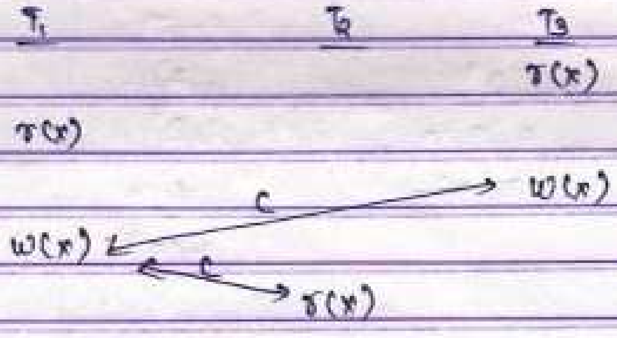
iii) $r_3(x), r_2(x), w_3(x), r_1(x), w_1(x)$

iv) $r_3(x), r_2(x), r_1(x), w_3(x), w_1(x)$

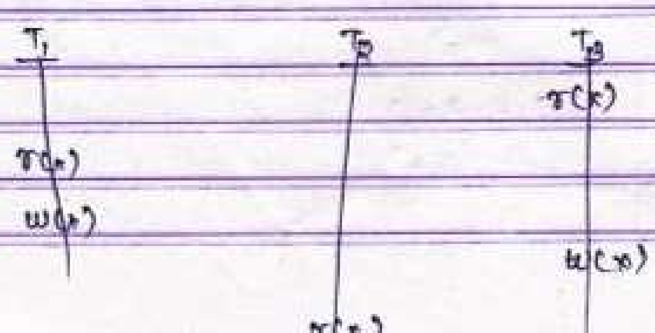
ii)



lecturenotes.in



Non-Conflict Serializable



~~Non-~~ Conflict Serializable

T1	T2	T3
		r(x)
	r(x)	
		w(x)
r(x)		
w(x)		

N.C

Conflict

T1	T2	T3
	r(x)	
		r(x)
		w(x)
r(x)		
w(x)		

T1	T2	T3
		r(x)
	r(x)	
r(x)		
		w(x)
w(x)		

N.C

Confl

T1	T2	T3
	r(x)	
		r(x)
r(x)		
		w(x)
w(x)		

N.C

Non-Conflict

T1	T2	T3
	r(x)	
r(x)		
		r(x)
		w(x)
w(x)		

View Serializability

A concurrent schedule 'S' is view serializable if it is view equivalent to a serial schedule S'.

Two schedules S and S' are view equivalent if they satisfy the following three conditions:-

i) If a transaction Ti is reading the initial value of data item 'q' in the schedule S, then the same transaction should read the initial value of q in S'.

ii) If the transaction Tj reads a value of 'q', i.e. produced by Ti in S, then the same transaction Tj must read the value, i.e. produced by Ti in S'.

Ti	Tj
w(A)	
	r(A)

iii) If Ti performs the last write operation in S, then Ti must also perform the last write operation in S'.

S

T1	T2	T3
r(A)		
	w(A)	
w(A)		
		w(A)

S'

T1	T2	T3
r(A)		
w(A)		
	w(A)	
		w(A)

The given schedule S is view equivalent to ~~no~~ serial schedule S'(T1, T2, T3)
The given schedule S is view serializable but it isn't conflict serializable.
All Conflict Serializable schedules are view serializable but all view serializable schedules are not conflict serializable.
A view serializable schedule S won't be conflict serializable if it contains "Blind Writes".
• The write operations without their appropriate read operations are called as blind write operations. So a concurrent schedule containing a blind write operation

Lecturenotes.in

will never be conflict serializable schedule.

RECOVERABLE SCHEDULES

13/4/11

T_1	T_2
r(A)	
w(A)	
	r(A)
r(B)	
¦	

A partial schedule

Suppose in this schedule, we allow the transaction t_2 to commit first, then because of some reason or the other, let us assume that transaction t_1 can't continue with its normal execution. So transaction t_1 is to aborted (rollback) and the changes made to the database are rolled back. Since t_2 is dependent on t_1, we should also roll back t_2 as t_1 is being aborted but this isn't possible since t_2 has committed. Hence, in this scenario, we can't recover from failure properly. Schedules of such type are called Non-Recoverable Schedules.

We need the schedules to be recoverable. In a recoverable schedule, the commit operation of independent transaction should appear before the commit operation of the dependent transaction.

CASCADELESS SCHEDULES

T_1	T_2	T_3
r(A)		
w(A)		
¦	r(A)	
	W(A)	
	¦	r(A)
		¦
¦	¦	¦

T_2 is dependent on T_1 and T_3 is dependent on T_2. If T_1 fails, then we've to roll back T_2 and as T_2 is being rolled back, we've to roll back T_3 also. This phenomenon is called Cascading Rollback, in which failure of one transaction leads to a series of roll backs of dependent transactions.

We need our schedules to be Cascadeless. In a Cascadeless schedule, for every pair of transaction t_i and t_j such that t_j reads the value produced by t_i, then the commit operation of t_i should appear before the read operation of t_j.

Testing for Conflict Serializability (Graph Based Approach)

Graph (V, E)

V = { No. of transactions given in the schedule } $\textcircled{T_1}$

E = { $T_i \rightarrow T_j$ }

Conditions are:-

i> read (Q) of T_i appears before write(Q) of T_j.

ii> w(Q) of T_i appears before r(Q) of T_j

iii> w(Q) appears before w(Q) of T_j

iv> If resulting graph doesn't contain a cycle, then it is conflict serializable.

Ex:-

T_1	T_2	T_3
r(x)		
		r(x)
w(x)		
	r(x)	
		w(x)

E = $(T_1, T_2), (T_2, T_3), (T_3, T_1)$

$$\begin{tikzcd} & T_1 \arrow[dl] \arrow[dr, bend left] & \\ T_2 \arrow[rr] & & T_3 \arrow[ul] \end{tikzcd}$$

It is a cycle so it is not conflict serializible.

Q:- $r_3(x), r_2(x), w_3(x), r_1(x), w_1(x)$

T_1	T_2	T_3
		r(x)
	r(x)	
		w(x)
r(x)		
w(x)		

$$\begin{tikzcd} & T_1 & \\ T_2 \arrow[ur] \arrow[rr] & & T_3 \arrow[ul] \end{tikzcd}$$

Lecturenotes.in

22/4/10

Con~~cu~~rrency Control

The Concurrency Control Schemes are necessary to keep the database in the consistent state. All the concurrency control schemes are based on the property of serializability. The Serializability property can be ensured, if the data items are accessed in a mutually exclusive manner.

Mutual Exclusion means that if a transaction Ti is ~~reading~~ accessing the data item Q, then no other transaction Tj is allowed to modify that data item. The Mutual exclusion condition can be ensured by implementing LOCKS, i.e, whenever a transaction Ti needs to access the data item Q, it has to lock Q in the appropriate mode.

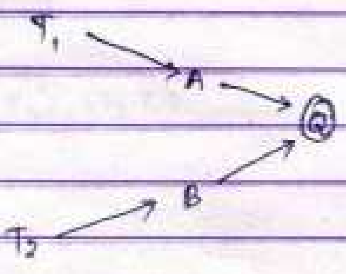
Basically, a transaction can lock the data item, in two different modes:-

i) Shared mode (S-Mode)

ii) Exclusive Mode (X-Mode)

When a transaction locks the data item Q in the shared mode, it can only read the value of Q but it can't updated. Similarly, when the transaction locks the data item Q in exclusive mode. It can read as well as update the value of Q. The transaction can lock the data item in the shared mode by executing LOCK-S(Q). Similary a transaction can request for exclusive mode by executing LOCK-X(Q). All the lock requests are made to the concurrency control manager or to the lock manager. If the lock requests are granted, then only the transaction can use the data-item, otherwise the transaction has to ~~wa~~ wait. After completing its operations on Q, the transaction will unlock the data item by executing UNLOCK(Q) instruction.

In a concurrently executing environment, there are several transactions that want to lock the data item in different lock modes. So, in this case, we've to define the LOCK COMPATIBILITY MATRIX.



LOCK-B(Q) → Lock Manager

Suppose a transaction Ti has locked the data item Q in LOCK MODE A, and transaction Tj wants to lock the data item Q in MODE B. If the lock request by Tj can be granted immediately, then we say that the LOCK MODES A & B are compatible, otherwise they are incompatible.

Ti → / Tj ↓	Lock-S	Lock-X
Lock-S	TRUE	FALSE
Lock-X	FALSE	~~TRUE~~ FALSE

Ex:-

T1	→ T1'
r(A)	Lock-X(A)
A = A-50	r(A)
w(A)	A = A-50
r(B)	w(A)
B = B+50	UNLOCK(A)
w(B)	LOCK-X(B)
	r(B)
	B = B+50
	w(B)
	UNLOCK(B)

T2	T2'
r(A)	LOCK-S(A)
r(B)	r(A)
display(A+B)	UNLOCK(A)
	LOCK-S(B)
	r(B)
	UNLOCK(B)
	DISPLAY (A+B)

Lecturenotes.in

Ex:-

	T_1	T_2		
	LX(A)			A=100
100	r(A)			B=100
50	A=A-50			
50	W(A)			
	U(A)			→ If this will be removed, it will become consistent. T_2 wouldn't be considered.
		LS(A)	50	
		r(A)	50	
		U(A)	50	
		LS(B)	100	
		r(B)	100	
		U(B)	100	
		display(A+B)		
100	L-~~S~~X(B)			
100	r(B)			
150	B=B+50			
150	W(B)			
	Unlock(B)			

Inconsistent

The schedule is keeping the database in inconsistent state because of UNLOCK-A instruction, if we can defer U(A) till the end then my schedule is keeping the database in consistent state. This shows that every transaction has to following a set of rules for locking and unlocking data items. This set of rules are called Locking and Unlocking Protocol.

Lecturenotes.in

23/4/11

LOCKING PROTOCOLS

The Locking and Unlocking of the data items should be done in such a way that there is no inconsistency, deadlock and no starvation.

→ 2-Phase Locking Protocol (2PL):

• Every transaction will lock and unlock the data item in two different phases :- (a) Growing Phase
(b) Shrinking Phase

In previous ex:- Lock and unlock request are in same transaction T_1, So it is not in 2-Phase Locking Protocol.

Growing Phase :- The transaction can make LOCK requests only. It cannot UNLock any data item.

Shrinking Phase : When a transaction unlocks a data item, it enters into Shrinking Phase. In this phase, a transaction can't make any new Lock requests.

The 2-Phase Locking Protocol ensures Conflict Serializability but it can't ensure freedom from deadlock.

```
        T1
     L-X(A)
     r(A)
     A = A-50
     W(A)
     L-X(B)   -> Time where the transaction executes its
                 last lock request is LOCK POINT
     r(B)
     B = B+50
     W = B
     UNLOCK(A)
     UNLOCK(B)
```

Let for T_2 : L.P = 1~~0~~1:00
" T_3 : L.P = 12:00
" T_1 : L.P = 10:00

We'll execute T_1 first, then T_2 and T_3.

It doesn't ensures freedom from deadlock and cascading rollback is also possible under 2-phase locking protocol.

Cascading rollbacks can be avoided by using a variant of 2-Phase locking protocol known as Strict 2-Phase Locking Protocol. Acc. to this protocol, all exclusive-mode Locks acquired by a transaction should be held by the transaction until that transaction commits.

Another variant of 2-Phase Locking Protocol exists called Rigorous-2 Phase Locking Protocol. Acc. to protocol, all the LOCKs acquired by the transaction (both S & X) are held by the transaction until the transaction commits. This restriction will generate only Serial Schedules and hence there is no deadlocks.

```
        T1              T2
      r(a1)           r(a1)
      r(a2)           r(a2)
        |               |
        |
                      display(a1+a2) ──┐
                                       S-Mode
      r(an)
   ┌─ w(a1)
X- ~~Shared~~ Mode
```

Lecturenotes.in

```
            T1                  T2
          L-S(a1)
           r(a1)
                              L-S(a1)
                               r(a1)
          L-S(a2)
           r(a2)
                              L-S(a2)
                               r(a2)
              |
              |
          L-S(an)
           r(an)
        Upgrade(a1)  ────────→  Convert S-Mode to X-Mode.
           W(a1)
```

In Downgrade, X-Mode is converted into S-Mode Lock.

Acc. to 2-Phase Locking Protocol, T_1 will acquire X-Mode LOCK on a_1 where T_2 will acquire S-Mode LOCK on a_1. This will result in a serial schedule only. If we want a concurrent schedule, we'll allow T_1 to lock a_1 in the shared mode and actually when T_1 writes a value to a_1, we'll change S-Mode to X-Mode LOCK. This is called LOCK Conversion.

All upgrades are carried out in Growing Phase and downgrade are carried out in Shrinking Phase.

Time-Stamp Based/Ordering Protocol 25/4/11

$T_i \rightarrow TS(T_i)$

Every transaction T_i is assigned a unique TS value by the database system denoted by $TS(T_i)$ prior to the start of execution of T_i.

If T_i has entered the system and has started its execution and then T_j enters and begins its execution, then $TS(T_i) < TS(T_j)$. Here T_j is called the younger transaction and T_i is called as Older Transaction.

There are two ways in which a transaction can be assigned a Time-stamp value:-

i) Using the System Clock

ii) Using a logical counter which is incremented everytime a new transaction enters into the system.

Apart from time-stamping a transaction, there are two other time-stamp values that are associative with any data item Q:-

i) Read Time-Stamp (Q):-

ii) Write Time Stamp (Q)

Read Time-Stamp(Q):- It is the time-stamp of largest transaction which has performed the Read(Q) operation successfully.

T_1 T_2 T_3

$TS(T_1) = \alpha$

$TS(T_2) = \beta$

$TS(T_3) = \gamma$

$R\cdot TS(Q) = \cancel{\alpha\beta\gamma}\ max(\gamma, \beta, \alpha)$

Write TS(Q) :- Time stamp of largest transaction which has performed w(Q) operation successfully.

Time-Stamp Ordering Protocol

This protocol ensures that any conflicting read and write operations are carried out in Time-stamp order.

1) Suppose T_i issues $r(Q)$

a) if $TS(T_i) < W\text{-}TS(Q)$

then the read opⁿ is rejected and T_i is rolled back.

b) if $TS(T_i) \geq W\text{-}TS(Q)$

then read opⁿ is allowed and $r\text{-}TS(Q) = MAX\{r\text{-}TS(Q), TS(T_i)\}$.

2) Suppose T_i issues $w(Q)$

a) if $TS(T_i) < r\text{-}TS(Q)$

then write opⁿ is rejected and T_i is rolled back.

b) if $TS(T_i) < W\text{-}TS(Q)$

then write opⁿ is rejected and T_i is rolled back.

c) Otherwise $w(Q)$ is executed and $W\text{-}TS(Q) = TS(T_i)$

Lecturenotes.in

Q:-

	T_1	T_2
	r(B) ;	r(B)
		B = B - 50
		w(B)
	r(A)	
		r(A)
	disp(A+B)	
		A = A+50
		w(A)
		disp(A+B)

Check if the schedule is legal under the TS Ordering Protocol.

Ans:-

TS(T_1)=0   TS(T_2)=0

r.TS(A)=0   w.TS(A)=0

r.TS(B)=0   w.TS(B)=0   1   2

Let TS(T_1)=1   Let TS(T_2)=2   1   2

1   0   0   0

2   2   0   0

→ 1	2	→ 1	2		1	2
2	0	2	2		0	0
2	2	2	2		1	0

All are updated, so it is legal.

<u>29/4/1</u>

Q.

T_1	T_2
r(A) ✓	
	r(A) ✓
	w(A) ✓
	r(B) ✓
obsolete write ← ~~w(A)~~	
r(B)	
w(B)	

1   2

0   0

~~2~~   ~~0~~

2   0

1   2

0   0

2   2

TS(T_1)= ~~0~~ 1   TS(T_2)= ~~0~~ 2

r.TS(A)= ~~0~~ ~~1~~ 2   w.TS(A)= ~~0~~ 2

r.TS(B)= ~~0~~ 2   w.TS(B)= ~~[illegible]~~ 0

∵ W(A) for T_1 is rejected and T_1 is rolled back, so the given schedule isn't legal under the TSOP.

THOMA'S WRITE RULE

It is a modification to the Time-Stamp-Ordering Protocol. Acc. to TSOP, any transaction that wishes to perform an obsolete write operation, has to be rolled back.

But acc. to Thoma's Write Rule, rather than rolling back the entire transaction the obsolete write op^n should be ignored.

Deadlock

Methods to handle Deadlock

The deadlock is a situation where there is a set of ~~situations~~ processes and each process in the set is waiting for an event that can be caused by another process within the set.

Let us consider a set of processes :-

$S = \{T_0, T_1, \ldots T_n\}$

where T_0 is waiting for an event that can be caused by T_1. T_1 is waiting

$T_0 \xrightarrow{event} T_1$

for an event that can be caused by T_2 ..... T_{n-1} is waiting for an event that can be caused by T_n and T_n is waiting for an event caused by T_0.

In this situation, none of the processes execute and the system is said to be in the deadlock state.

Conditions for deadlock to occur :-

i> Mutual Exclusion

ii> Hold and Wait

iii> No preemption

iv> Circular Wait

When all the four conditions occur simultaneously then only the system enters into deadlock state. There are three different strategies to deal with deadlock :-

i> Deadlock prevention

ii> Deadlock Avoidance

iii> Deadlock Detection & Recovery

LectureNotes.in

Deadlock Prevention

I) The methods that are used for Deadlock Prevention are based on Time Stamp.

i) The Wait Die Scheme

ii) The Wound Wait Scheme

Wait Die Scheme

This scheme is non-preemptive in nature. (Preemption → forceful removal

If a transaction T_i is waiting for a data-item, i.e. currently held by

then T_i will wait if Time Stamp of T_i, $TS(T_i) < TS(T_j)$.

But if $TS(T_i) > TS(T_j)$, then T_i will be rolled back.

$$\begin{tikzcd} T_1 \arrow[r, bend left, "request"] & T_2 & T_3 \arrow[l, bend right] \end{tikzcd}$$

15 > 10, T_3 is rolled back.

TS → 5   TS → 10   TS → 15

5 < 10, so T_1 will wait

Wound-Wait Scheme

Preemptive in nature. If T_i is waiting for a data item, i.e. currently

by T_j then the transaction T_j will be rolled back if $TS(T_i) < TS(T_j)$.

But if $TS(T_i) > TS(T_j)$, then T_i will wait.

$$\begin{tikzcd} T_1 \arrow[r, bend left, "T_2 \text{ will roll back}"] & T_2 & T_3 \arrow[l, bend right, "T_3 \text{ will wait}"'] \end{tikzcd}$$

5   10   15

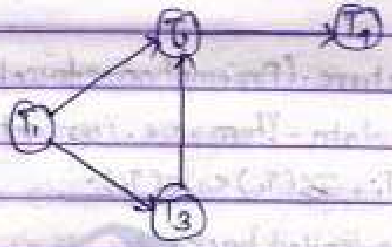
T_1 has wounded T_2.

II) Another scheme is Time Out Based Scheme.

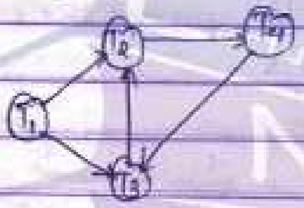
Whenever a transaction request a lock onto a data-item, it waits for a finite amount of time. If within that time frame, if the lock request isn't granted, we say that the time quantum or time slice has expired and the requested/waiting transaction is rolled back.

Deadlock Detection and Recovery

In deadlock detection, we construct the Wait-For Graph.
The vertices of graph corresponds to transactions present in the system.
We draw an edge from $T_i \rightarrow T_j$ if T_i is waiting for a data-item, i.e., held by T_j.
This edge is called as Wait-for-Edge. If the resulting wait-for graph contains a cycle, then the system is in Dead-lock state.



It doesn't contain any cycle, so no deadlock.



It contains a cycle, so deadlock state occurs
Once the system enters into deadlock state, then we've to recover from deadlock.
There are two methods to recover from deadlock:-

i> Resource Preemption: Forcefully take out the resources by O.S
ii> Transaction Rollback: Select one transaction, roll it back and check whether the deadlock is occuring yet or not. The deadlock cycle is broken after rollback, the transaction that is selected is called ~~[illegible]~~ Tx. Victim Transaction. Care must be taken to ensure that starvation condition do not arise.

Starvation:→ Always anyone transaction isn't be selected as a Victim Transaction.

Recovery System

Like any electromechanical device, the computer system is prone to failure. There are three types of failures:-

i) Transaction Failure

ii) System Crash

iii) Disk Failure.

Any modifications to the data item is carried out, provided it is prese main memory. If the data item isn't available, we've to bring the dat item, say x, to main memory. Instead of bringing x alone, we trans the entire block of data containing x to the main memory. The block transfer can be carried out by executing two operation:-

a) Input (B):- Transfer block B from secondary storage to main me

b) Output (B):- Transfer block B from main memory to secondary sto

Whenever the system fails, there are two different approaches to reco from failure:-

a) Log-based Recovery :- A log contains records pertaining to updatio data-item in the database. A log record contains following fields:-

i) Transaction Identifier (ID)

ii) Data-item identifier

iii) Old Value of a data item

iv) New Value of data item

A log record can be contain following types of statements

i) <Ti, Start>:- Transaction Ti has started its execution.

ii) <Ti, x_j, V_1, V_2>:- Transaction Ti wants to perform a write opera on the data-item x_j. V_1 is the value of x_j prior to the write ope or before the write op^n and its value will be V_2 after the write is complete.

iii) <Ti, Commit>:- Transaction Ti has committed.

iv) <Ti, Abort>:- Transaction Ti is aborted.

Based on these log record, we've to determine, which are the tranc that needs to execute Redo and Undo op^n after the system fails.

A Redo operation ~~perform~~ permanently reflects the changes by a transaction onto the database.

The Undo op^n will bring the database to its Original state.

Deferred Modification

A=100, B=100

		Deferred
Ex:-	r(A)	1. <T₁, Start>
	A = A-50	2. <T₁, A, 50>
	w(A)	3. <T₁, B, 150>
	r(B)	4. <T₁, commit>
	B = B+50	
	w(B)	

Redo op^n is applied, then A=50, B=150

If 4th statement isn't there, then we'll not execute the commit op^n. If we'll have all the four statements, then only the transaction will be executed successfully.

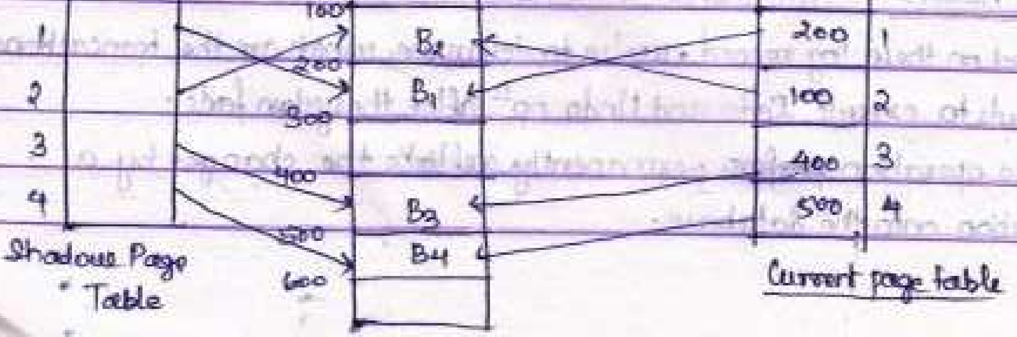
For any transaction T_i, if the log record doesn't contain <T_i, commit>, redo op^n can't be executed.

Immediate Database Modification

1. <T₁, Start>
2. <T₁, A, 100, 50>
3. <T₁, B, 100, 150>
4. <T₁, commit>

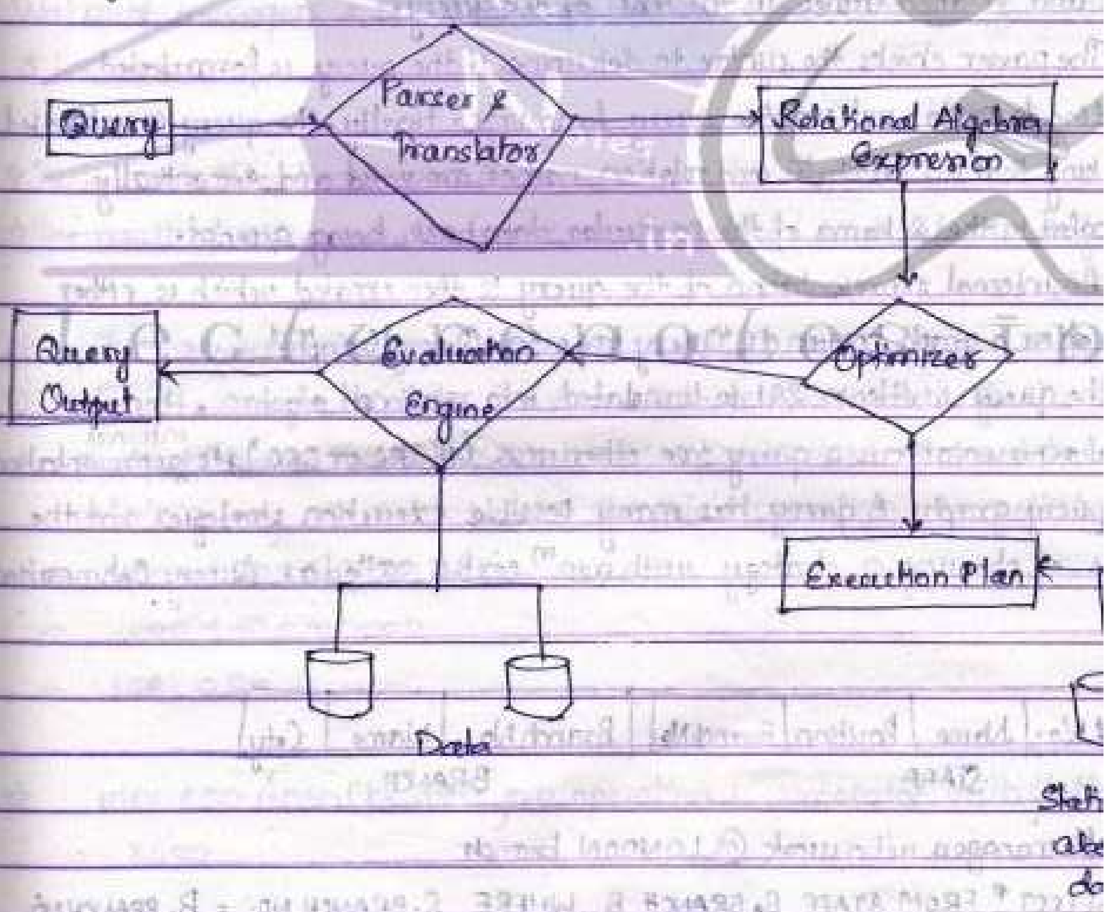
If the log record doesn't contain, the commit op^n we've to execute UNDO op^n. The Undo op^n will reinitialize the database to its original state.

b) ~~Shadow~~ Shadow Paging:- The database which is stored onto the secondary storage is divided into fixed size blocks, called pages. The database mayn't be stored contiguous blocks. There is a page table which gives information about location of blocks in secondary storage.



Rest of the blocks are free blocks.
Suppose a transaction wants to modify a data-item present in a
specific block (B_2), at the very first step, we try to find a free p
and load the contents of desired blocks onto that particular free pag
and update the current page table accordingly.
Allow the transaction to perform its updations onto B_2 i.e., load
upto the free page. If the transaction successfully commits, the
current page table will be the new shadow page table for succe
transactions. But if the transaction fails, then the current page tabl
discarded and shadow page table is treated as new current page ta

<u>Query Processing and Optimization</u> 30/4/1



Query processing refers to range of activities that are involved in extracting data from a database. The basic steps are, parsing and translating, optimization, evaluation.

A sequence of primitive operations that can be used to evaluate a query is called as Query Execution Plan or Query Evaluation Plan.
The Query Execution Engine takes a Query Evaluation Plan, executes that plan and produces the desired output. The different execution plans for a given query can have different costs (No. of disk access). It is the responsibility of the system to construct a query evaluation plan that minimizes the cost of Query evaluation. This task is called as Query Optimization.

A query which is expressed in High Level Query Language is scanned, parsed and validated. The scanner identifies the SQL keywords, attribute names and relation names in the text of the query.

The parser checks the syntax to determine if the query is formulated according to syntax rules of the query language. Finally, the query is validated by checking that all attributes and relation names are valid and semantically meaningful in the Schema of the particular database being queried.

An internal representation of the query is then created which is either a tree or a graph known as Query tree or a Query Graph.

If the query written in SQL is translated into relational algebra, then its internal representation is a query tree otherwise (if TRC or DRC) its internal representation is a query graph. A query has many possible execution strategies and the process of choosing a strategy with min^{m} cost is called as Query Optimization.

Q:

Staff No.	Name	Position	Branch No.

STAFF

Branch No.	Name	City

BRANCH

Find all managers who work @ LONDON branch

Sol^{n}. SELECT * FROM STAFF S, BRANCH B WHERE S.BRANCH NO. = B.BRANCH NO.
AND S.POSITION = 'MANAGER' and B.CITY = 'LONDON';

Relational Algebra (R1)

$$\sigma_{(Position = 'Manager') \wedge (Branch.city = 'London')} \left(\sigma_{Staff.Branchno. = Branch.Branch No.} (Staff \times Branch) \right)$$

(R2) $$\sigma_{(Position = 'Manager') \wedge (Branch city = 'London')} (Staff \bowtie Branch)$$

(R3) $$\left. \begin{array}{c} \sigma_{Position = 'Manager'} (Staff) \\ \bowtie \\ \sigma_{City = 'London'} (Branch) \end{array} \right\}$$ First unary then binary operation

Suppose the 'Staff' relation 1000 tuples 'Branch' relation contains 50 tuples. Total no. of managers = 50 (1 for each branch) and London City contains 5 branches.

Assumptions:-

i) Tuples are accessed one @ a time.

ii) The result of intermediate operations are stored back into the disk

2/5/11

No. of disk accesses:-

1) Total no. of tuples (Staff) + Total no. of tuples (branch) + {Cartesian product (Branch × Staff)} ×2 → because again we've to read the whole query, so ~~set~~ to perform the select operation, we multiply 2.

$\Rightarrow$ 1000 + 50 + 50000 × 2

$\Rightarrow$ 1000 + 50 + 100000

$\Rightarrow$ 1001050

ii) 1000 + 50 + {1000 (because join operation is associated with staff) ×2}

$\Rightarrow$ 3050

iii) 1000 + 50×2 + 50 + 5×2

1000 + 100 + 50 + 10

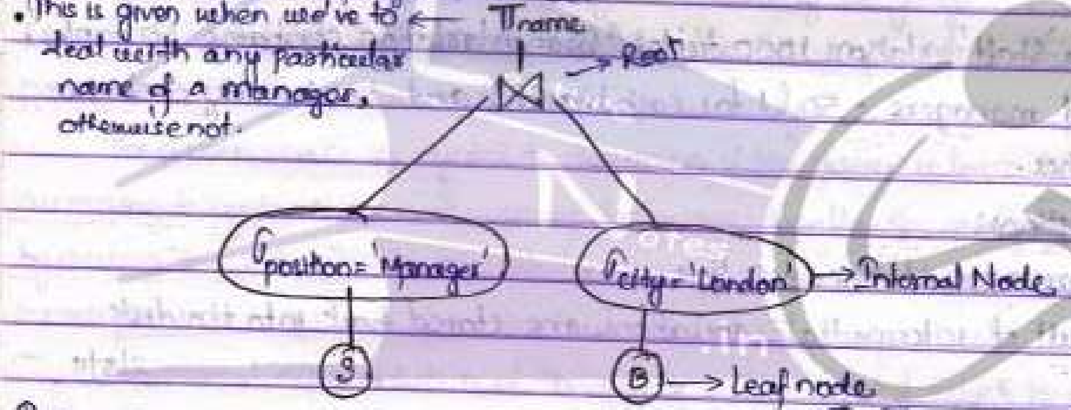
. 1160

Hence, one of the fundamental strategies in query-processing is to perform the unary operations like selection and projection as early as possible

Heuristic Approach of Query Optimization

Heuristics are the rules that have been formulated, tested and found efficient in most of the cases.

For Heuristic approach to optimization, the relational algebra expression for the given SQL statement has to be represented in the form of a Query Tree. The tree represents the input relations of the Query as a leaf node and the relational algebra operations are represented as the internal nodes and the result is displayed @ the root.



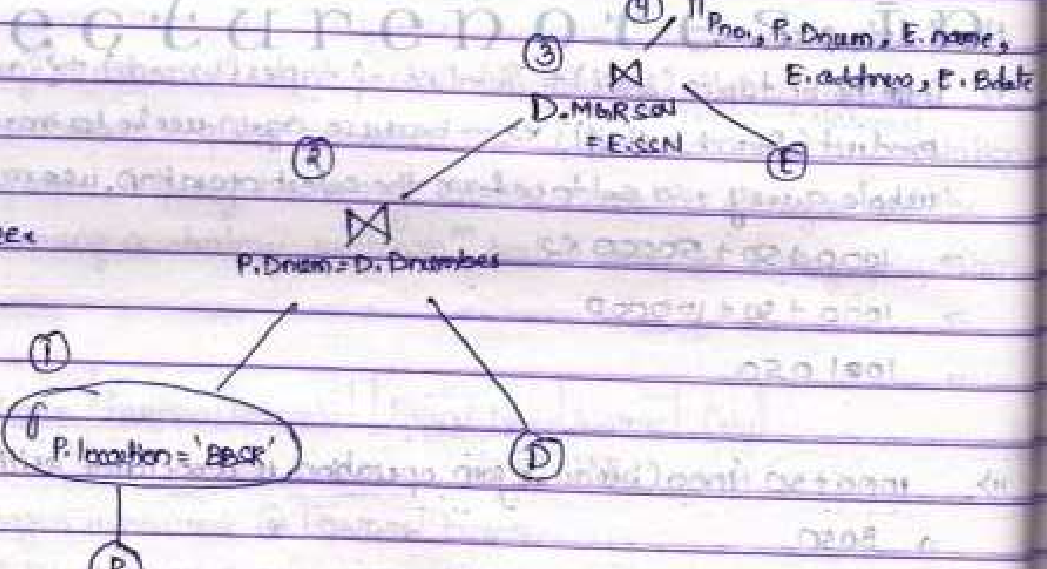
This is given when we've to deal with any particular name of a manager, otherwise not.
π_{name}
Root
$\bowtie$
$\sigma_{position='Manager'}$
$\sigma_{city='London'}$
Internal Node
S
B
Leaf node

Que

P – Project

D – Dept.

E – Employee.



4
$\pi_{Pno, P.Dnum, E.name, E.address, E.Bdate}$
3
$\bowtie$
D.MGRSSN = E.SSN
E
2
$\bowtie$
P.Dnum = D.Dnumber
1
$\sigma_{P.location='BBSR'}$
D
P

In this query tree, the three relations project, dept. and employee are represented as the leaf node P, D and E respectively.

While the relational algebra operations are represented by internal nodes.

When the tree is executed, the node marked as ① must begin execution before node ② and similarly node ② must begin execution and start producing the results before node ③ starts its execution. The result is displayed at the root.

Like for a single SQL statement, there are many equivalent relational algebra expressions. Similarly, there can be several query trees which are equivalent to one another. So, an optimizer must include the rules of equivalence among the relational algebra expressions which can be applied to the initial query tree to transform it into a final optimized query tree.

Rules of Equivalence

I) $\sigma_{p \wedge q \wedge r}(R) = \sigma_p(\sigma_q(\sigma_r(R)))$

II) $\sigma_p(\sigma_q(R)) = \sigma_q(\sigma_p(R))$

III) $\pi_L \pi_M \ldots \pi_N (R) = \pi_L(R) \quad \because L \subset M \subset N$

$\pi_A(\pi_{A,B}(\pi_{A,B,C}(R))) = \pi_A(R)$

IV) $\pi_{A_1,A_2,\ldots A_n}(\sigma_p(R)) = \sigma_p(\pi_{A_1,A_2 \ldots A_n}(R))$

$\pi_{Eno.,Ename,sal}(\sigma_{sal>25000}(R)) = \sigma_{sal>25000}(\pi_{Eno.,Ename,sal}(R))$

σ and π are commutative provided π contains the attributes on which the selection is to be performed.

V) $R \bowtie_p S = S \bowtie_p R$

$R \times S = S \times R$

VI) $\sigma_p(R \bowtie_R S) = (\sigma_p(R)) \bowtie_R S$

OR

$\sigma_{p \wedge q}(R \bowtie_R S) = (\sigma_p(R)) \bowtie_R (\sigma_q(S))$

vii) Same rule as vi for projection.

viii) $R \cup S = S \cup R$
$R \cap S = S \cap R$

ix) $\sigma_p(R \cup S) = \sigma_p(R) \cup \sigma_p(S)$
$\sigma_p(R \cap S) = \sigma_p(R) \cap \sigma_p(S)$
$\sigma_p(R - S) = \sigma_p(R) - \sigma_p(S)$

x) $\pi_L(R \cup S) = \pi_L(R) \cup \pi_L(S)$

xi) $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$
$(R \times S) \times T = R \times (S \times T)$
$(R \cup S) \cup T = R \cup (S \cup T)$
$(R \cap S) \cap T = R \cap (S \cap T)$

LectureNotes.in

Steps in Heuristic Optimization 06/05/11

Ex:-

SSN	Name	BDate	Address

Employee

ESSN	Pno

Works ON

Pnumber	Pname

Project

Q. Find the name of employees born after 1972 who work on the project named as 'Networking'.

Sol:- Select name from employee E, works_on W, Project P
where P.Pname = 'Networking' and P.Pnumber = W.Pno and
W.ESSN = E.SSN and E.Bdate > '31-DEC-1972';

———— → C_1
^^^^^^^ → C_2
∿∿∿∿∿ → C_3
— — — — → C_4

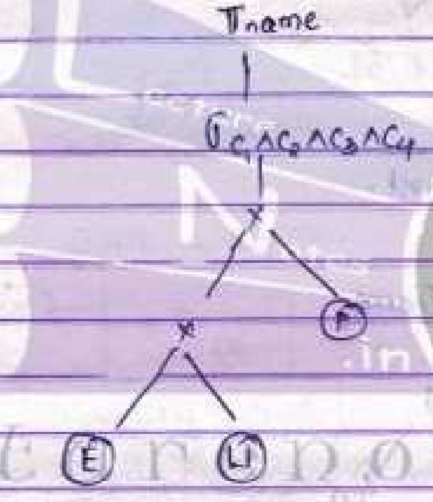
Steps

I) Perform the selection operations as early as possible. Use Rule I to cascade the selection operation and Rules IV, VI, VII and IX to move the selection operations as far down the tree as possible. Keep the selection predicates on the same relation together.

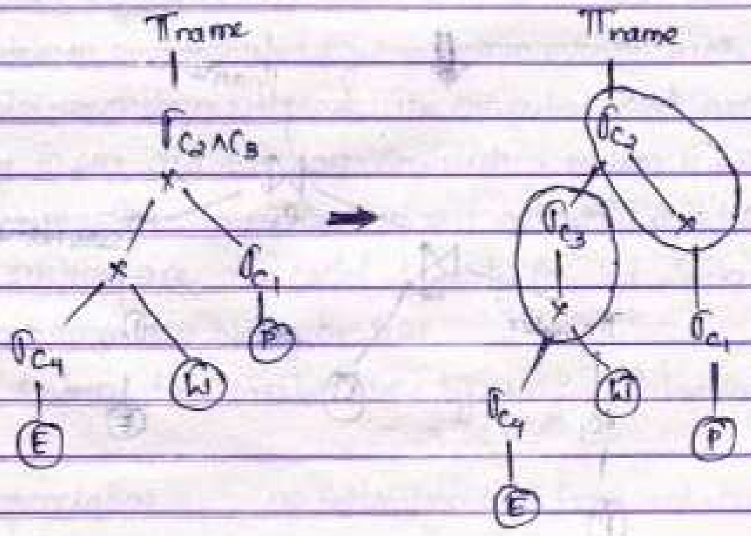
II) Combine the Cartesian product with a subsequent selection operation, who's predicate represents a join condition into a join operation.

III) Use associativity of Binary Operations to rearrange the leaf nodes so that the leaf nodes with the most restrictive $\overset{selection}{\wedge}$ operations are executed first.

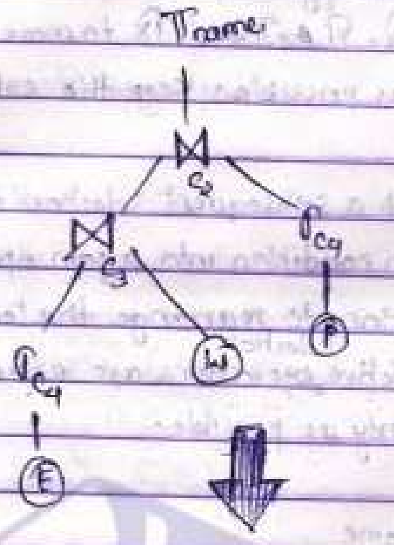
IV) Perform the project operation as early as possible.



Initial Query Tree



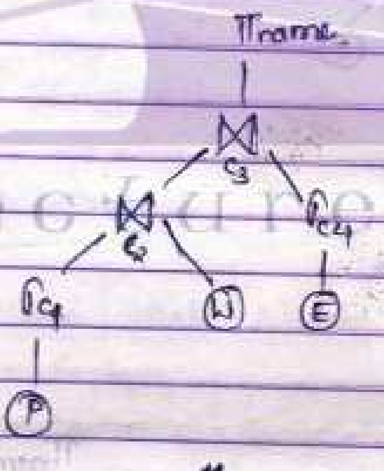
Lecturenotes.in

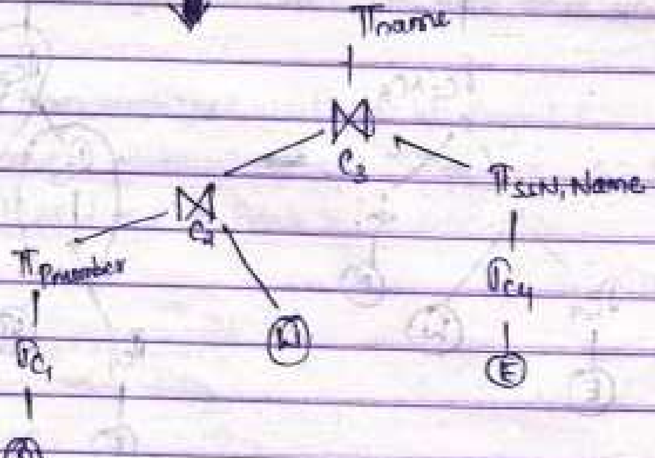


Total tuples $E \times W = 8$

" $W \bowtie P = 4$

So, $W \bowtie P$ will be done first.





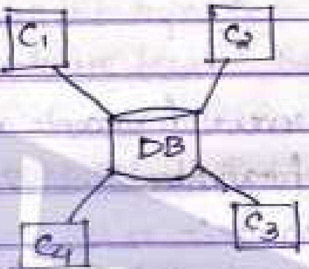
Lecturenotes.in

Ex:- SQL: Select P.PropertyNo, P.Street from Client C, Viewing V, Property for rent P where C.proptype = 'Flat' and C.Client no = V.client no. and V.Propertyno = P.Propertyno and C.Maxrent >= P.rent and C.proptype = P.type and P.ownerno = 'CO 93';

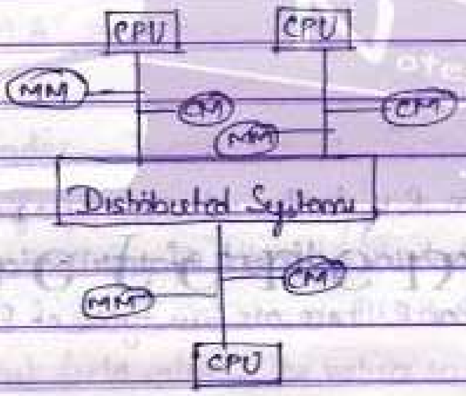
DISTRIBUTED SYSTEMS

7/5/11

Centralized



Distributed



CM -> Cash Memory
MM -> Main Memory

In a distributed database systems, the database is stored on several computers. The computers in a distributed system communicates with one another through the inter-connection network. The computers that are connected may vary in size and function. So, such a system is called as a Heterogeneous System. The computer that are attached don't share memory. So, these systems are also called Loosely Coupled Systems. The computers are also named as Nodes or Sites.

In a distributed system, data can be stored by using the following three approaches:-

i) Replication ii) Fragmentation iii) Replication and Fragmentation

Replication

The system maintains several identical replicas (copies) of the relation. Each replica is stored at a different site resulting in Data Replication.

Fragmentation

The relation is partitioned into several fragments and each fragment is stored at different site.

Replication and Fragmentation

The relation is partitioned into several fragments and then the system maintains several replicas of each fragments.

The advantages of Replication are:-

i> improves reliability

ii> high Availability

iii> increased parallelism

Fragmentation

If a relation R is fragmented, then R is divided into no. of fragments like $R_1, R_2 \ldots R_n$, these fragments contain sufficient information to allow the reconstruction of the original relation R. There are two types of Fragmentation:-

i> Horizontal

ii> Vertical

Horizontal Fragmentation split the relation by assigning each tuple of R, to one or more of the fragments. In this case, a fragment can be defined as a selection on the global relation R.

$$R_i = \sigma_{P_i}(R)$$

We can obtain the original relation R by taking union of all these fragments

$$R = R_1 \cup R_2 \cup \ldots \cup R_n$$

Vertical fragmentation is same as decomposition. It involves the definition of several subsets of attributes of R. Each fragment is defined as

$$r_i = \Pi_{R_i}(r)$$

We can reconstitute the original relation by taking Natural Join.

$$r = r_1 \bowtie r_2 \bowtie \cdots \bowtie r_n$$

INDEX STRUCTURE OF A TABLE

The index structures provide the ways of accessing the records without affecting the physical placement of records on a disk. To find a record in the file based on a certain selection criteria on an indexing field, one has to initially access the index which points to one or more blocks in the file where the reqd. records are located. Basically, there are two types of indices :-

i) Single level Index ii) Multilevel Index (Tree Data Structure)

Under Single Level Index, we've :-

a) Primary Index b) Secondary index c) Clustering Index

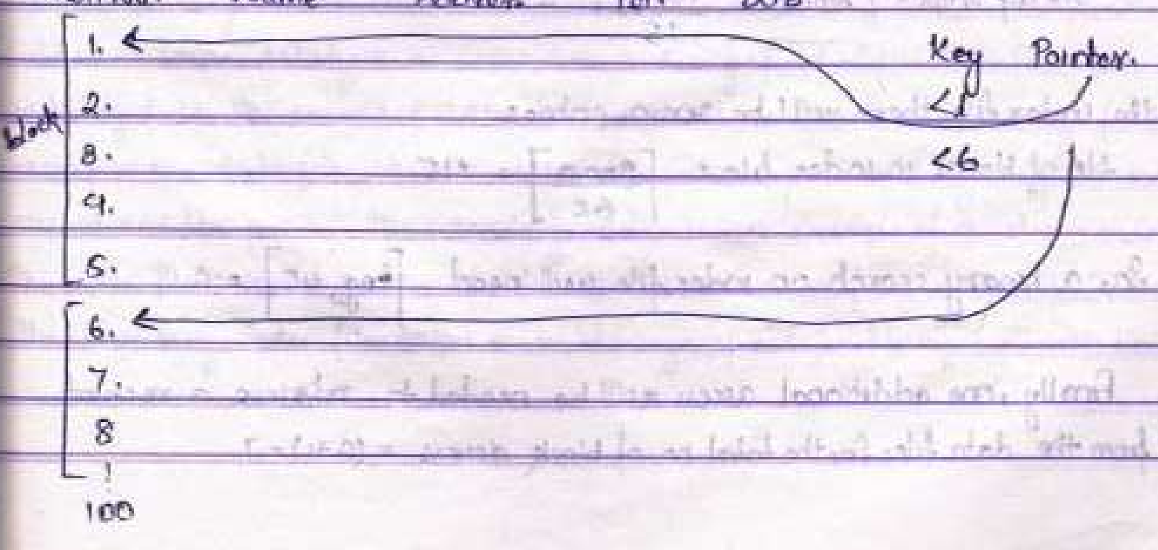
Under Multilevel Index, we've :-

a) B-Tree b) B^+ Tree

Single Level Index

In Single Level Index, one or more fields of the data file are treated as an index field or an indexing attribute. The index stores each value of the index field with a list of pointers to all the disk blocks that contains records with that field values. The values in the index are ordered so that we can perform a Binary Search.

A) Primary Index



A primary index is an ordered file whose records are of fixed length of with two fields:- i) Primary key of data file.

ii) Pointer to a disk block.

There is one index entry in the index file for each block in the datafile. Each index entry has the value of the primary key field for the first record in a block and ~~th~~ pointer to that block. An index entry is of the form:

<key, pointer>

Total no. of entries in the index file is same as total no. of blocks in the datafile. The first record in each block is called as Anchor Record or Block Anchor. The index can be Dense or Sparse. An index is dense if it contains an index entry for each record in the datafile otherwise it is called as Sparse. The primary index is an example of a Sparse Index.

Ex:- Consider an ordered file with 30,000 records. Let the block size = 1024 bytes

Let the record size = 100 bytes

So, no. of records/block = $\lfloor 10.24 \rfloor$ (taking integer, i.e, 10)

Blocking ~~Ration~~ Factor = 10

No. of blocks need to store the whole record = $\lceil 30000/10 \rceil$

= 3000

To search a record (binary search), no. of block access needed =

$\lceil \log_2 3000 \rceil = \lceil 11.55 \rceil = 12$

Suppose the key field is 9 bytes long and the pointer is 6 bytes long.

So, the size of record in index file = 9+6 = 15 byte

No. of entries / block = $\frac{1024}{15} = 68$

In the index file there will be 3000 entries.

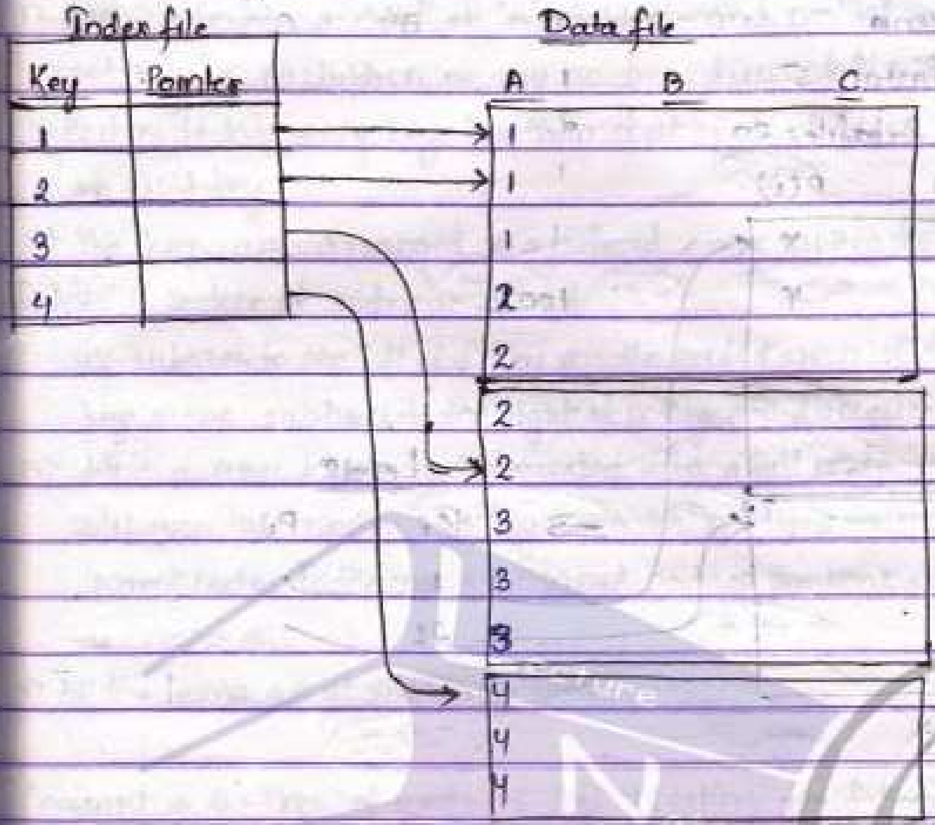
No. of blocks in index file = $\left\lceil \frac{3000}{68} \right\rceil = 45$

So, a binary search on index file will need $\lceil \log_2 45 \rceil = 6$

Finally, one additional access will be needed to retrieve a record from the data file. So, the total no. of block access = (6+1) = 7

13/5/11

B> Clustering Index



If the records of a file are physically ordered on a non-key field which doesn't have a distinct value for each record, that field is called Clustering Field. A Clustering Index is an ordered file with two fields:- i> Key ii> Pointer. There is one entry in the clustering index file for each distinct value of the Clustering Index. A Clustering Index is an example of a Sparse Index.

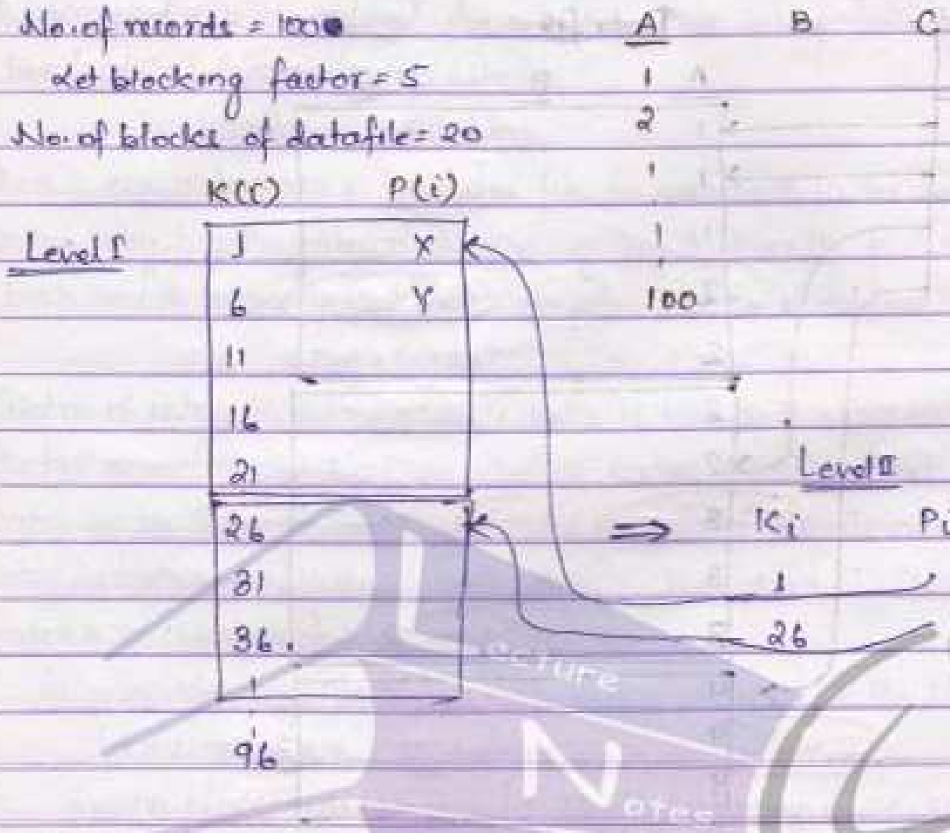
C> Secondary Index

The Secondary Index File is an ordered file with two fields. The first field is of same datatype as some non-ordering field of the datafile, i.e. an indexing field. The second field is a pointer. There is one index entry for each record in the data-file which contains the value of secondary key for the record and the pointer. The secondary index is an example of a dense index. The index file is sorted on key-field so that we can perform a Binary Search. The secondary index file takes more storage space and longer search time.

12/5/21

MULTILEVEL INDEX

No. of records = 100
Let blocking factor = 5
No. of blocks of datafile = 20



• Data are too much.
• Time taking is more
So, multilevel index is used.

Lecture Notes .in

14/5/11

Dynamic Multilevel Index - B-Trees

i> Each node has a max^m of 'n' children and min^m of n/2 children. The root may've no children or any no. from two of the max^m.

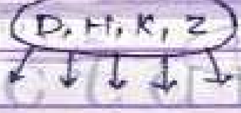
ii> Each node has only key less than that of its children with a max^m of (n-1) key

iii> The keys are arranged in a defined order within the node. All keys in
~~Def^n: A B-tree of order 'n' is a t~~
the subtrees to the left of a key are the predecessors of the key and all keys in the subtrees to the right of a key are successors of the key.

iv> When a new key is to be inserted into a full node, then the node is split into two nodes and key with the median value is inserted in the parent node. In the case, the parent node is the root, a new root node is created.

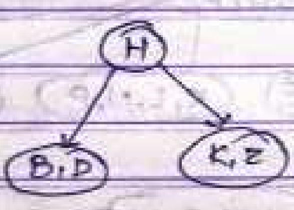
v> All the leaves are at the same level.

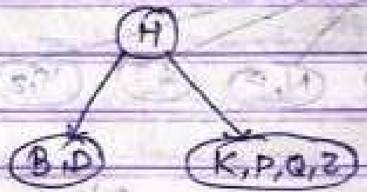
Construct a B-Tree of order 5 by inserting the following datas:
Σ D, H, K, Z, B, P, Q, E, A, S, W, T, C, L, N, Y, M

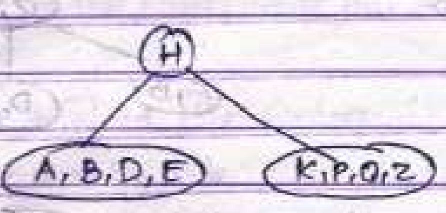
i>


ii> B, D, H, K, Z

Median = H

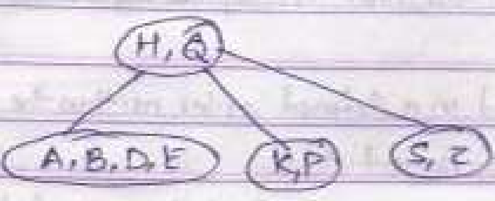


iii>


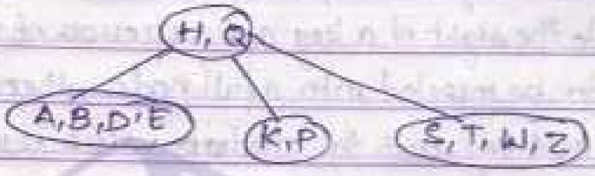
iv>


Lecturenotes.in

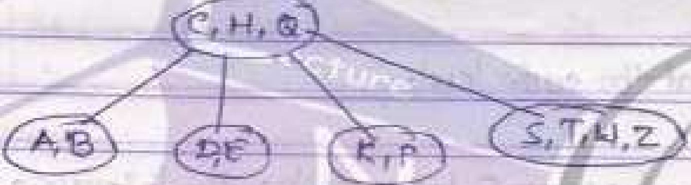
iv> When we encounter S ; it is to be inserted on right sub-tree, but the node is already full.



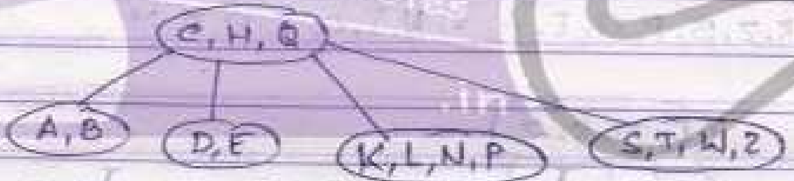
v>



vi>

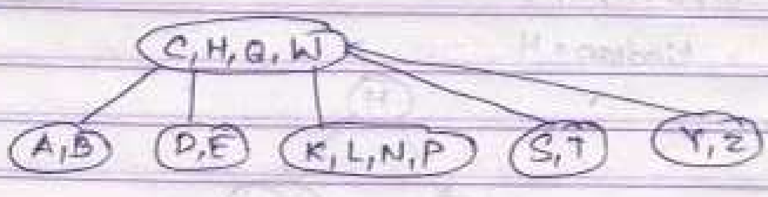


vii>

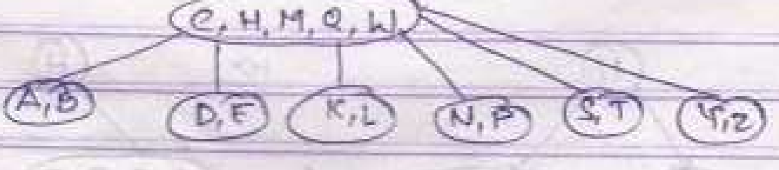


Lecturenotes.in

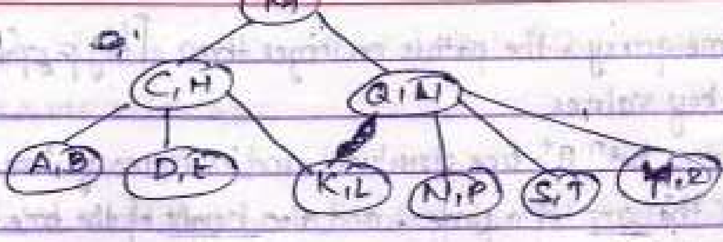
viii>



ix>



x> Here we find that ~~K, H~~ root node is already full. So, we take the median of data values contained in the root and split the node.



B$^+$ Tree Index Files

25/5/11

The main disadvantage of index-sequential file organization is that performance degrades as the file grows, both for index lookups and for sequential scans.

→ A B$^+$ Tree index takes the form of a balanced tree in which every path from the root of the tree to a leaf of tree is of same length.

- Each non-leaf node in the tree has between $\lceil n/2 \rceil$ and n children, where n is fixed for a particular tree.

P_1	K_1	P_2	K_2	- - ---	P_{n-1}	K_{n-1}	P_n

It contains upto $n-1$ search key-values $K_1, K_2 - - - K_{n-1}$
and n pointers $P_1, P_2 - - P_n$

$K_i < K_j$, ~~i~~ i.e. search key values are kept in sorted order.

Points

- Each leaf can hold upto $n-1$ values.
- We allow leaf nodes to contain as few as $\lceil (n-1)/2 \rceil$ values. The ranges of values in each leaf don't overlap.
- If a B$^+$ tree index is a dense index, every search key value must appear in some leaf node.
- The structure of non-leaf nodes is same as that for leaf nodes, except that all pointers are ~~pointed~~ pointers to ~~leaf~~ tree nodes.
- A Non-leaf node may hold upto n, and must hold @least $\lceil n/2 \rceil$ pointers.

The number of pointers in a node is called Fanout of the node.

B in B$^+$ tree stands for "Balanced". It is due to this property that ensures good performance for look up, insertion and deletion.

In processing a B^+ tree query, the path is no longer than $\left[\log_{\lceil n/2 \rceil}(K)\right]$
where K = search key values
An important difference bet^n B^+-tree structures and in memory tree structures like binary trees is the size of a node, and also height of the tree. ~~The binary~~

Binary Trees	B^+ Trees.
i) Each node is small	i) Each node is large
ii) Each node has atmost 2 pointers.	ii) A node can have large no. of pointers.
iii) Path can be of length $\log_2(K)$	iii) It is of length $\left[\log_{\lceil n/2 \rceil}(K)\right]$

Insertion
If the root itself gets splitted, entire tree becomes deeper.

• B^+ tree file organizations can be used to store large objects, such as SQL clobs and blobs which may be larger than a disk block and as large as multiple gigabytes.

Static Hashing
One disadvantage of sequential file organization is that we must access an index structure to locate data, or must use binary search that results in more I/o operations. Hashing allows us to avoid accessing an index structure.
Bucket :- It denotes a unit of storage that can store one or more records.
A bucket is a disk block, but could be chosen smaller or larger than a disk block.
Let K denote set of all search keys and B denote a set of all bucket addresses. Then a hash func^n h is a func^n from K to B.
To insert a record
Compute $h(K_i)$ which gives the address of bucket for that record.
To perform lookup
Compute $h(K_i)$ then search for bucket with that address.
Suppose $h(K_i) = h(K_j)$

lecturenotes.in

If we perform lookup on K_1, then bucket $h(K_1)$ contains records with K_1 and records with K_3.

Deletion

Compute $h(K_i)$, search for corresponding bucket and delete the record.

Hashing can be used for two different purposes:-

1) Hash File Organization :- We obtain the address of disk block containing a desired record directly by computing a funcⁿ.

2) Hash Index Organization :- We organize search keys, with their associated pointers into a hash file structure.

Hash Function

An ideal hash funcⁿ distributes the stored keys uniformly across all the buckets, so that every bucket has the same no. of records.

Uniform distribution

Hash funcⁿ assigns each bucket, same no. of search-key values from the set of all possible search-key values.

Random Distribution

- Each bucket will have nearly same no. of values, regardless of actual distribution of search key values.
- Hash value will not be corelated to any alphabetic or ordering by length on the search key values.

The following hash funcⁿ can be used to hash a string in Java implementation:

$$s[0] * 31^{(n-1)} + s[1] * 31^{(n-2)} + \cdots + s[n-1]$$

- A bad hash funcⁿ may result in lookup taking time $\propto$ no. of search keys in a file.
- A well designed funcⁿ gives an average-case lookup time that is constant, independent of no. of search keys in the file.

Handling of Bucket Overflows

Bucket Overflow may occur for several reasons:-

i) Insufficient buckets

$$n_B > \frac{n_r}{f_r}$$

$n_B \rightarrow$ no. of buckets

$n_r \rightarrow$ total no. of records stored

$f_r \rightarrow$ no. of records that will fit in a bucket.

Designation assumes that total no. of records is known when the hash func^n is chosen.

ii) Skew

Some buckets are assigned more records than others, so a bucket may overflow. This condition is Skew. It can occur for two reasons:-

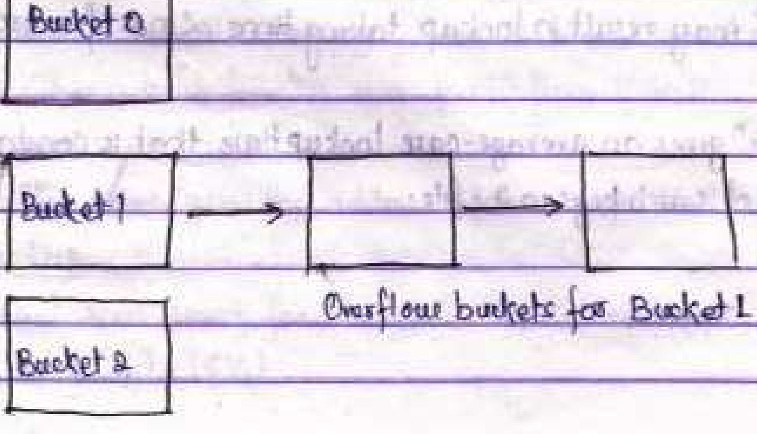
a) Multiple records may have the same search key.

b) Chosen hash func^n may result in non-uniform distribution of search keys.

So, that the probability of bucket overflow is reduced, no. of buckets is chosen to be $\left(\frac{n_r}{f_r}\right) * (1+d)$, where d = fudge factor, around 0.2

Handling by Overflow buckets

If a record is inserted into bucket 'b' and b is already full, system will provide an Overflow bucket for b and insert the record into b. If it is also full, then system will provide another overflow bucket and so on. All the overflow buckets of a given bucket are chain together in a linked list. This is Overflow Chaining.



lecturenotes.in

The system must examine all the records in bucket b to see whether they match the search key, as before. If bucket b has overflow buckets, the system must examine the records in all the overflow buckets. This form of hash structure is Closed Hashing.

Open Hashing :- The set of buckets is fixed, there are no overflow chains. Instead, if a bucket is full, the system inserts the records in some other bucket in the initial set of bucket B. One policy to use the next bucket (in cyclic order) that has space ; this policy is Linear Probing.

Closed & Open Hashing

Open Hashing is used to construct Symbol Tables for Compilers & Assemblers but Closed Hashing is used for database system ;

Reason :- Deletion under open hashing is difficult.

Important drawback of Hashing

Hash func^n can't be changed easily, if the file is being indexed, grows or shrinks.

Hash Indices

A Hash index organizes the search keys with their associated pointers, into a hash file structure.

Consider a secondary hash index on A/c file, for search key A/c No.

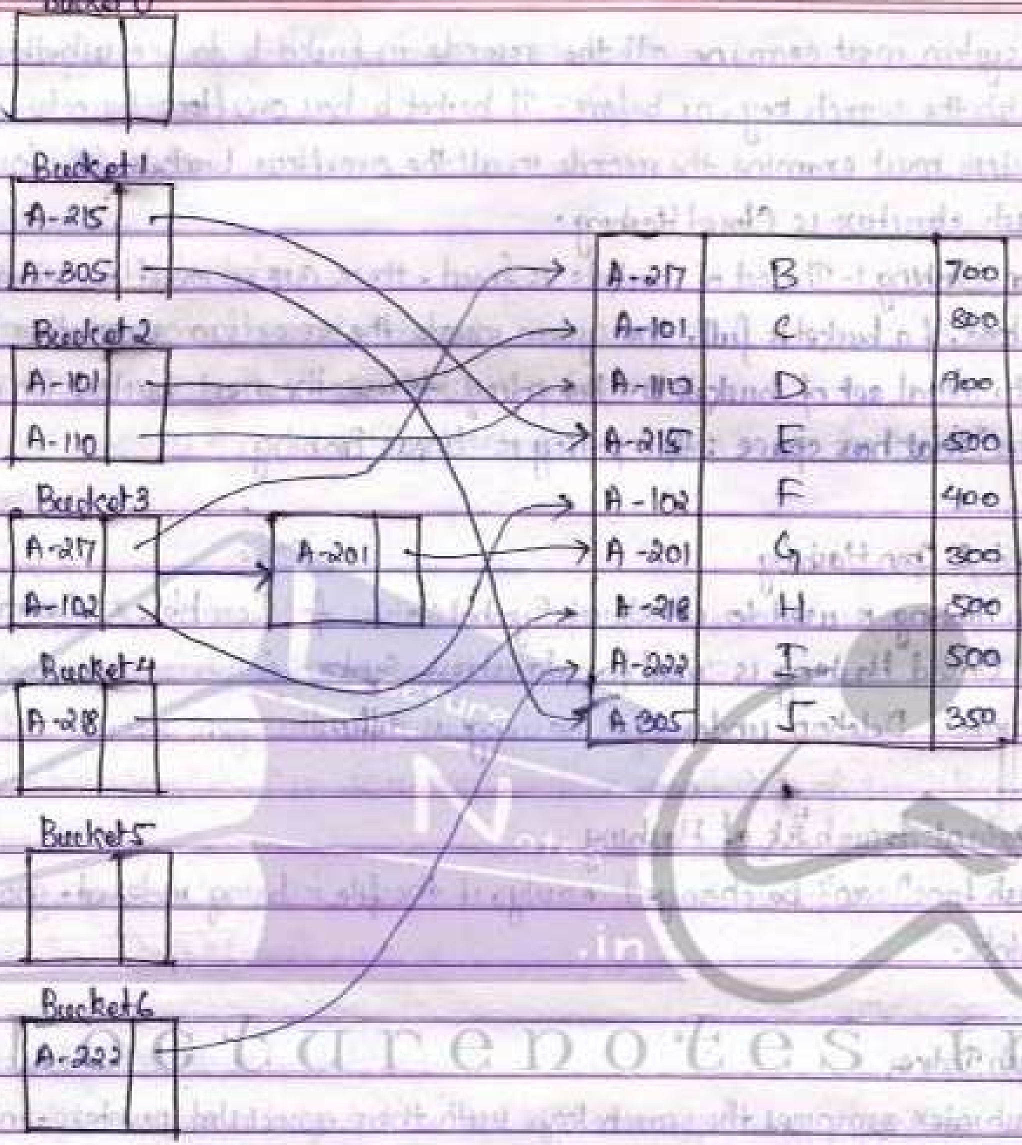
No. of buckets = 7

Bucket size = 2

One of the bucket has 3 keys, so it has an overflow bucket

A/c No. is primary Key, so each search key has only one associated ptr.

lecturenotes.in



If a file is itself organized by hashing, there's no need for separate has index structure. Therefore, it ~~never~~ never needs a Clustering index structure.

Dynamic Hashing:-

If we are to use Static hashing for such a database, we have three options:-
i) Choosing a hash func^n based on current file size, result in performance degradation of a database.
ii) Choosing a # func^n based on anticipated file size, performance degradation is avoided, but it results into wastage of space.
iii) Choosing a # func^n recomputing it on every record in file. This is a massive time computer consuming operation.

We'll study one form of dynamic hashing called Extendable Hashing.

Data Structure

We choose a # func^n h with a desirable property of uniformity and randomness. The # func^n generates large range values - b-bit binary integers. Value for b = 32.

2^{32} = 4 billion.

We don't use b bits initially, at a point we use i bits, $0 \le i \le b$.

Value of i grows & shrinks with size of database.

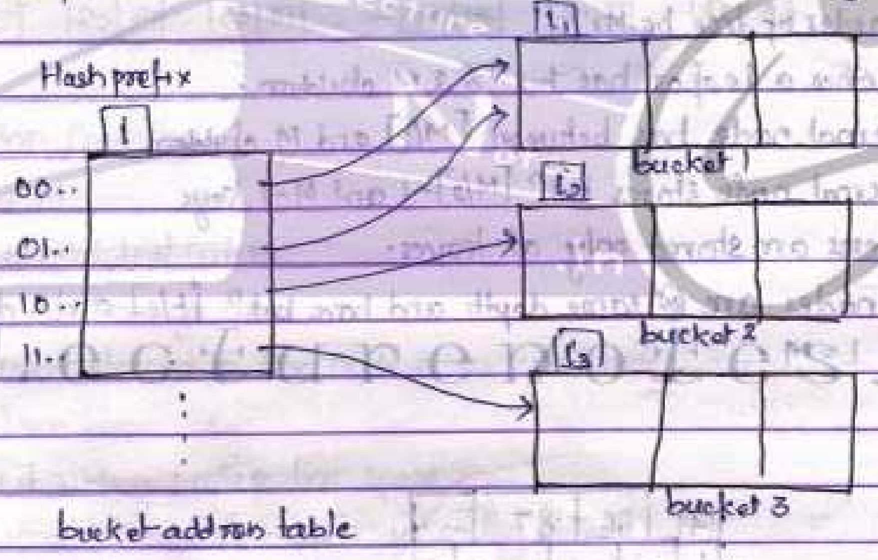
Consider a general extendable hash structure.

i = i bits of h(K) are reqd. to determine correct bucket for K.

Several consecutive entries may point to same bucket. All such entries will have a common hash prefix, but length of prefix may be < i.

Integer associated with bucket j is i_j.

The no. of bucket-address table that points to a bucket j is $2^{(i-i_j)}$.



Static v/s Dynamic Hashing

- Extendable Hashing, no buckets need to be reserved for future growth, bucket can be allocated dynamically.

A disadvantage of the hashing is that lookup involves an additional level of indirection, since the system accesses the bucket address table before accessing the bucket itself.

Another form of dynamic hashing, Linear Hashing avoids the extra level of indirection associated with extendable hashing at a possible cost.

Comparison of Ordering Indexing and Hashing

Select $A_1, A_2 \ldots A_n$
from r
where $A_i = C$

Here hashing is preferable

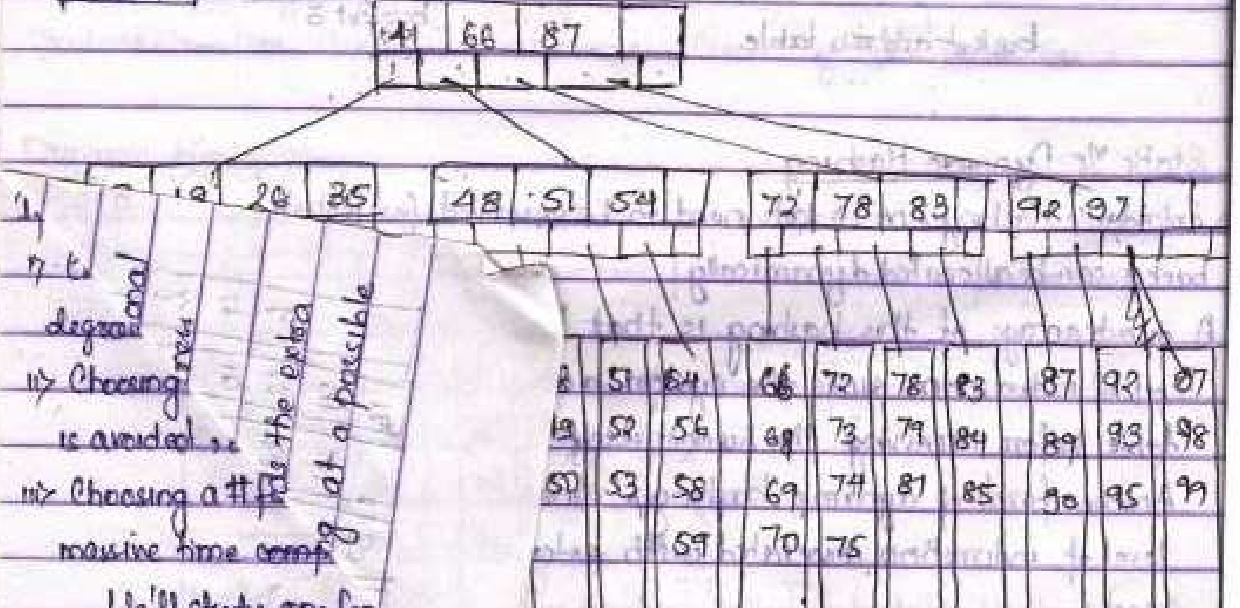
Select $A_1, A_2 \ldots A_n$
from r
where $A_i \leq C_2$ and $A_i \geq C_1$

Here ordering indexing techniques are preferable.

Simple Explanation of B^+ trees:
Let the order of tree be M.

i) Root is either a leaf or has bet^n 2 & M children.
ii) Each internal node has between $\lceil M/2 \rceil$ and M children
iii) Each internal node stores bet^n $\lceil M/2 \rceil - 1$ and M − 1 keys
iv) Data items are stored only at leaves.
v) All leaf nodes are @ same depth and have bet^n $\lceil L/2 \rceil$ and L data items, L = M.

For M = 5



1)
7   and
degree
ii) Choosing   even
is avoided.   the extra
iii) Choosing a H fn   at a possible
massive time comp
We'll study one for

Satisfying all conditions i.e,

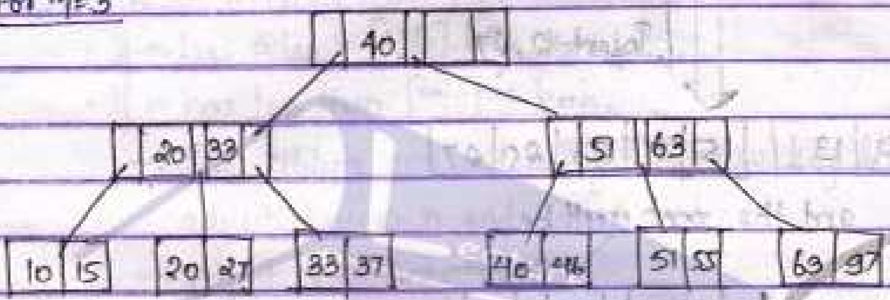
i) Root node containing betn 2 and 5 children, i.e, 4.

ii) Each internal node has betn 3 and 5 children.

iii) Each internal node stores betn 2 and 4 keys.

iv) All leaf nodes are of same depth and have betn 3 and 5 data items.

For M=3



Insertion Operation

Order M

1) Find correct leaf node n.

2) If n has enough space,

a) insert element in the node n in sorted order

b) Done

3) Else if sibling node p has space

a) insert element in node n in sorted order

b) Move last item of n to p.

c) Update the corresponding key in parent node so that first key of p is in the parent

4) Else

a) Split n into n and a new node n2.

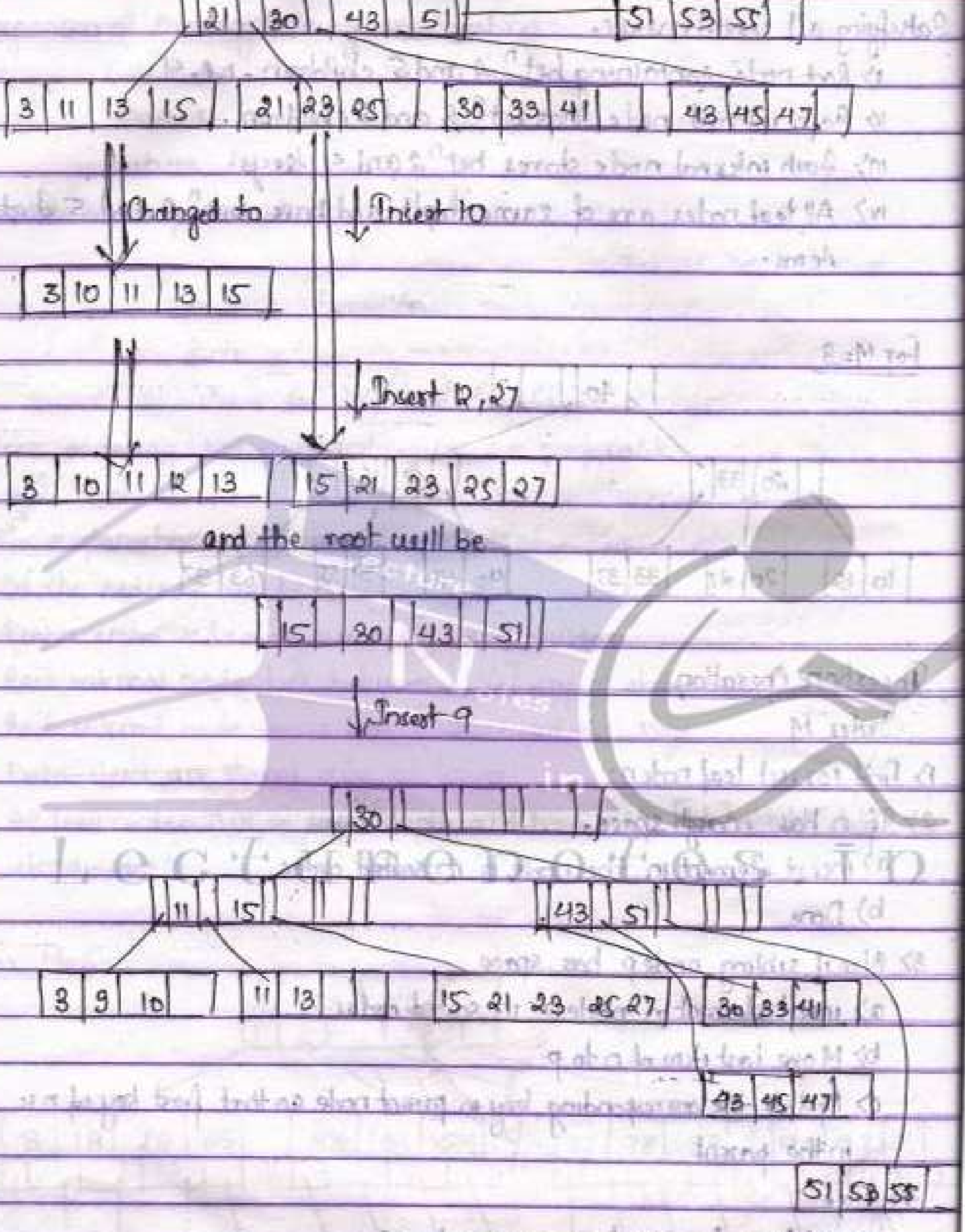
b) Redistribute element into n & n2.

c) Insert key of first element of n2 into parent of n.

d) If parent of n ~~keys~~ has keys less than M-1, done.

e) Else repeat a) to d).

Ex:- M=5


21
30
43
51
51
53
55
3
11
13
15
21
23
25
30
33
41
43
45
47
Changed to
Insert 10
3
10
11
13
15
Insert 12, 27
3
10
11
12
13
15
21
23
25
27
and the root will be
15
30
43
51
Insert 9
30
11
15
43
51
3
9
10
11
13
15
21
23
25
27
30
33
41
43
45
47
51
53
55

Deletion Operation

Find node n on the path from root to leaf node containing K.

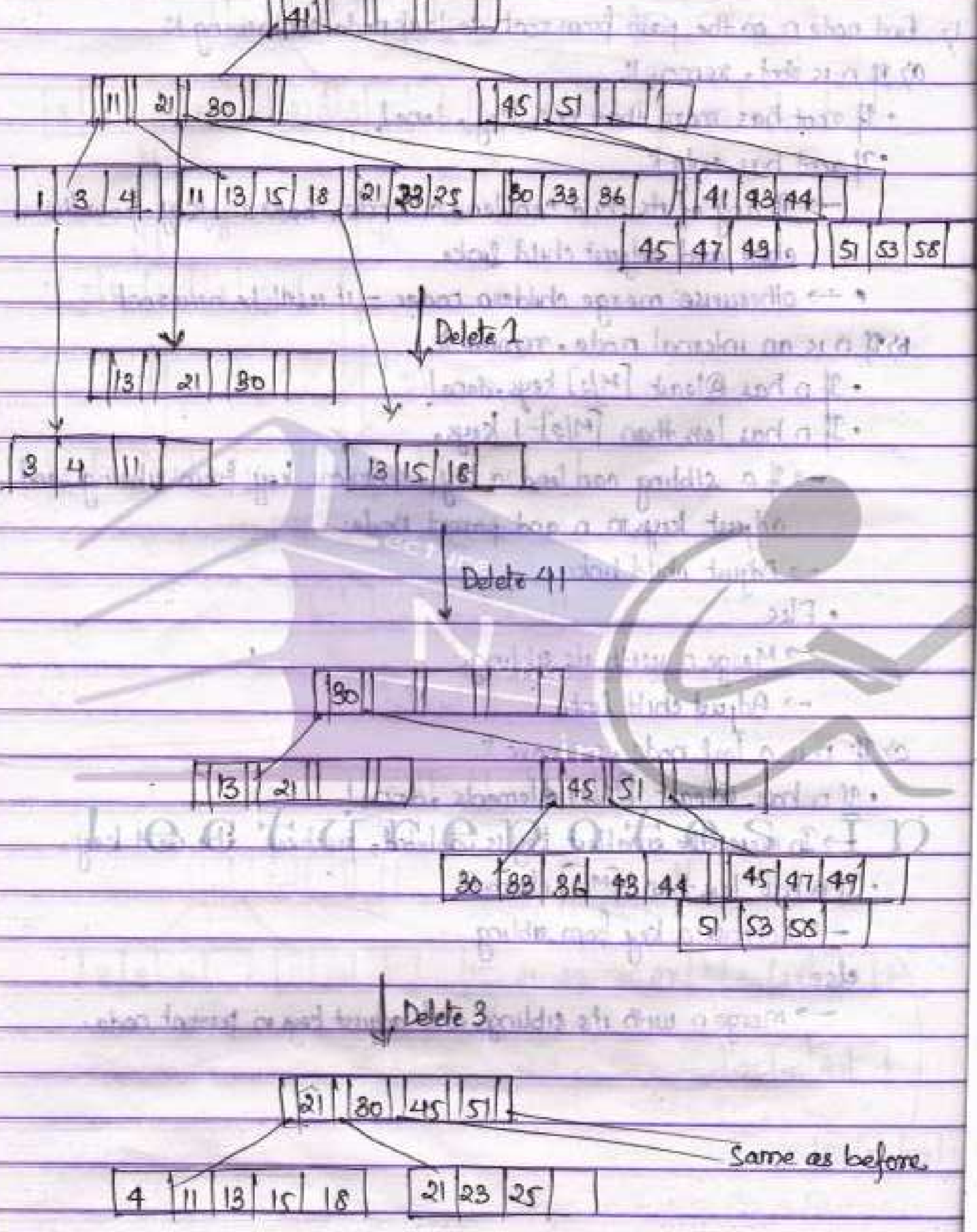
a) If n is root, remove K.
- If root has more than one key, done!
- If root has only K
  - → If any of its child can lend a node, borrow key from the child and adjust child links
  - → otherwise merge children nodes - it will be new root.

b) If n is an internal node, remove K.
- If n has @least ⌈M/2⌉ keys, done!
- If n has less than ⌈M/2⌉-1 keys,
  - → If a sibling can lend a key, borrow key from sibling and adjust keys in n and parent node.
  - → Adjust child links.
- Else
  - → Merge n with its siblings.
  - → Adjust child links.

c) If n is a leaf node, remove K.
- If n has @least ⌈M/2⌉ elements, done!
  - → In case the smallest key is deleted, push up the next key.
- If n has less than ⌈M/2⌉ elements
  - → Borrow a key from sibling

else
  - → merge n with its siblings and adjust keys in parent node.

M=5



Lecturenotes

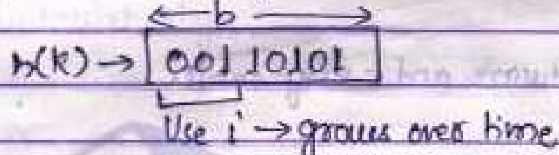
Hash Function

Example :-

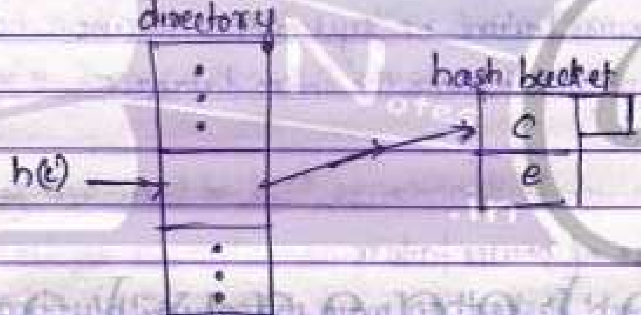
- Key = '$x_1 x_2 \ldots x_n$' n byte char string
- Have b buckets
- $h : (x_1 + x_2 + x_3 \ldots + x_n) \bmod b$

Extendible hashing

a) Use of i of b bits output by # func^n.



b) Use directory that maintains pointers to hash buckets (indirection)



Bucket merge Condition

- Bucket i's are the same.
- First (i-1) bits of the # key are same

Directory shrink Condition

All bucket i's are smaller than directory i.

STORAGE & FILE STRUCTURE

Overview of Physical Storage Media

Storage media are classified by:-

i) Speed with which data can be accessed.

ii) cost/unit of data to buy the medium

iii) medium's reliability.

Typically available medias are:-

I) Cache

- Fastest and most costly
- Memory is used by hardware and is very small.

II) Main Memory

- Storage medium for data available to be operated.
- General Purpose Machine instructions operated.
- Contents lost if power failure or system crash occurs.
- too small & too expensive to store the entire database.

III) Flash Memory

- Data survives in case of power failure
- Data reading is much faster than main memory but writing is complicated.
- Drawback → It can support only a limited no. of erase cycles ranging from 10000 to 1 million. To overwrite the data, we've to erase the whole bank of memory
- It is a form of EEPROM (Electrically Erasable Programmable ROM).
- It has found popularity in use of digital cameras, USB, etc.

IV) Magnetic Disk Storage

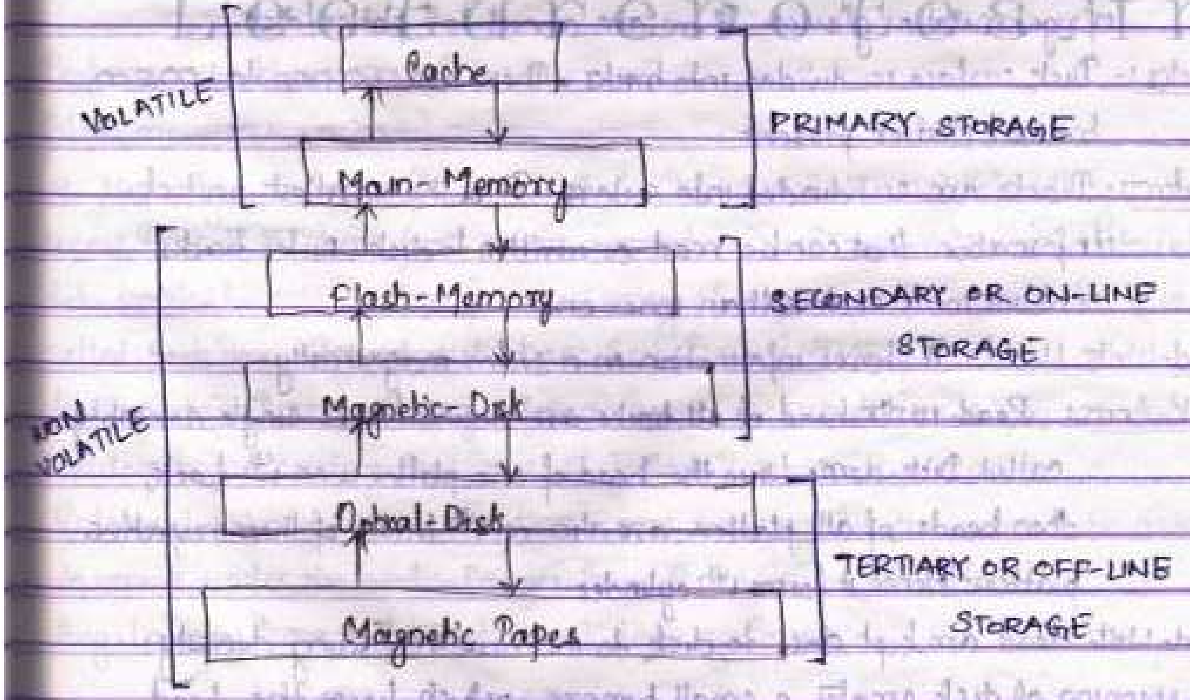
- Medium for long term storage of data
- Datas must be moved from disk to main memory to get accessed.
- They also survive powerfailures and system crashes.
-

V> Optical Storage.

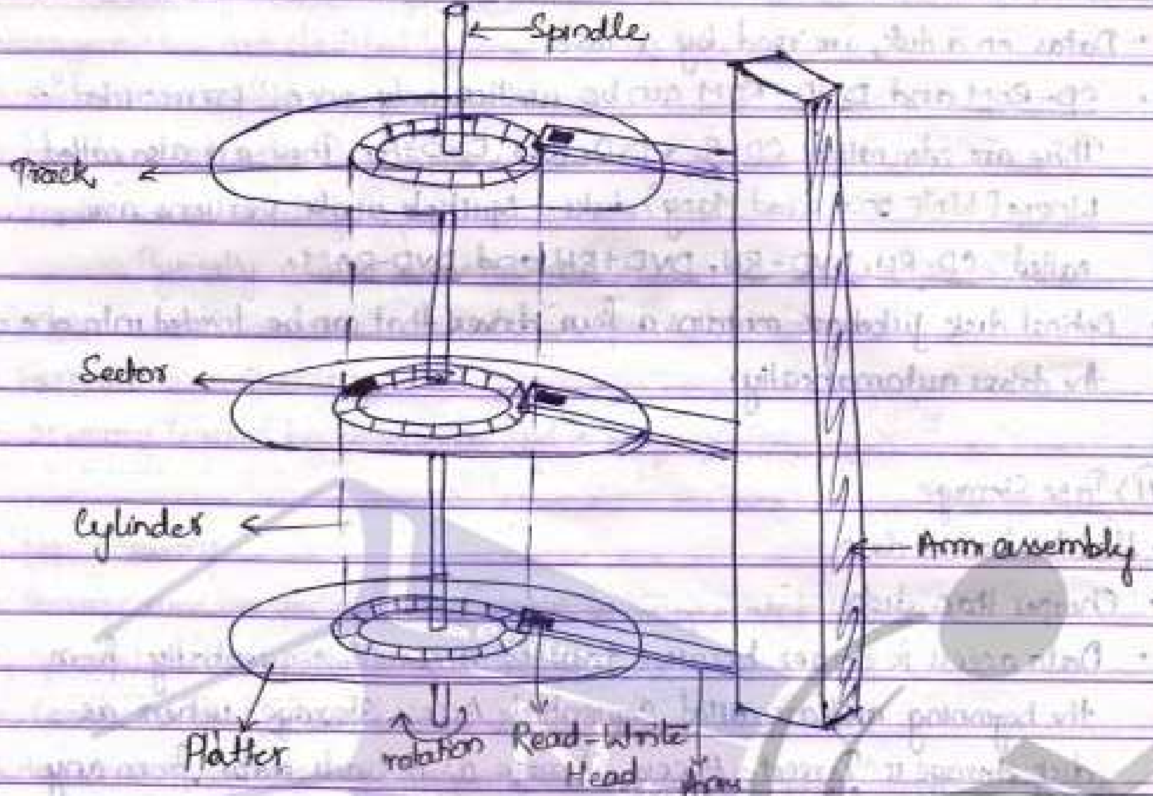
- Most popular forms CD (700MB - 80 minutes), DVD (4.7 or 8.5 GB).
- Data on a disk, are read by a laser.
- CD-ROM and DVD-ROM can be written only once "prerecorded". These are also called CD-R, DVD-R or DVD+R. These are also called WORM (Write once Read Many) disks. Multiple write versions are called CD-RW, DVD-RW, DVD+RW and DVD-RAM.
- Optical disk jukebox contains a few drives that can be loaded into one of the drives automatically.

VI> Tape Storage

- Used for backup data
- Cheaper than disks.
- Data access is slower because tape is accessed sequentially from the beginning and are called Sequential-Access Storage, where as disk storage is Direct-Access Storage as it reads data from any location.
- Tape libraries are used to hold exceptionally large collections of data like of Terabyte or even petabytes.



Magnetic Disks



Platter: Flat circular shape, its two surfaces are covered with a magnetic material and information is stored on surfaces. These are of rigid metal or glasses. There are about 1-5 platter/disk.

Tracks :- Disk surface is divided into tracks. There are 50,000 to 1,00,000 tracks/platter present.

Sector :- Tracks are subdivided into sectors. It is the smallest unit of information that can be read or written to disk. Outer tracks contains more sectors than inner one.

Read-Write Head : It stores information on a sector magnetically.

Disk-Arm : Read-write head of all tracks are mounted on a single assembly called Disk-arm. When the head of one platter is on ith track, then heads of all platters are also on ith track of their respective platters. Hence it forms ith cylinder.

Read-Write Head are kept close to disk to increase recording density. The spinning of disk creates a small breeze, which keeps the head floating above the disk surface.

If the head contacts the disk surface, head can scrape the recording medium off the disk, destroying the data.
Disk Controller interfaces betn computer system and hardware of the disk. It accepts the high level command to read or write a sector and initiate actions such as moving of disk arm to the right tracks.
Remapping of bad sectors :- Disk controller also logically map the sector to a different physical location, if it detects that the sector is damaged at the time of reading or writing.
There are a no. of common interfaces for connecting links to PC :-
1> ATA → AT Attachment
II> SATA → Serial ATA
III> SCSI → Small Computer System Interconnect

SAN :- Storage Area Network, large no. of disks are connected by a high speed N/W to a no. of server.
NAS :- N/W attached Storage, provides a file system interface using networked file system instead of networked storage appearing to be a large disk.

Performance Measures of Disks
Main measures are :-
1> Capacity II> Access Time III> Data Transfer Rate IV> Reliability
Access Time is the time from when a read or write request is issued to when data transfer begins. The time for positioning of arm over the correct track is called Seek Time. Typical disks have Seek Time 2-30 msec.
Average Seek Time is the avg. of Seek Times measured over a sequence of random requests. Avg. Seek Time $\approx \frac{1}{2}$ Seek Time = 4-10 msec.
Rotational Latency Time is the time spent waiting for the sector to be accessed to appear under the head. Ranges from 4-11.1 msec/rotation.
Avg. Latency Time = $\frac{1}{2}$ × Rotational Latency Time

Access Time = Seek Time + Latency Time = 8-20 msec.

Data-Transfer Rate :- Rate at which data can be retrieved from or stored to the disk. Current disk system accepts 25-100 MB/sec.

MTTF (Mean Time to Failure) is a measure of disk reliability. It is the amount of time that, on avg, we can expect the system to run continuously without any failure.

SATA interface allows only one disk to be connected to each interface.

Optimization of Disk-Block Access

Each request on the disk is in the form of a Block No., which is a logical unit consisting of a fixed no. of contiguous sectors. Datas are transferred betn disk & Main Memory in units of blocks.

Techniques to speed up the access of data on disk are:-

1> Scheduling :- Disk-Arm Scheduling algorithms attempt to order accesses to track in a fashion that increases no. of accesses that can be processed. There is a special algorithm called Elevator Algorithm which works in the same way as that of an elevator, checks at every step for any requests and then reaches a places where there is no any request and then come in reverse direction.

II> File Organization :- To reduce block-access time, we must organize block on disk so that we can expect data to be accessed.

Windows & Unix O.S hide the disk organization from users and manage allocation internally.

III> Non-Volatile Write Buffers :- ~~Purpose of~~ performance of update-intensive database applications, such as transaction-processing systems heavily dependent on speed of disk writes.

Non-Voltatile (NV-RAM) is used to speed up disk write drastically. Their contents aren't get lost in Power failure. A common way to implement NV-RAM is to use battery-backup RAM.

IV> Log-disk :- A disk devoted to writing a sequential log - in the same way as NV-RAM buffer does. File System that supports log-disk are called Journaling File Systems. It can be implemented without a separated log disk, keeping data & log on same disk, reduces cost.

It allows quick restart without the need for such file system consistency checks after system crash.

RAID

Redundant Arrays of Independent Disks proposed to achieve improved performance and reliability. At first, Independent was not there in RAID for I, but it represents Inexpensive.

Improvement of Reliability via Redundancy

The solⁿ to improve reliability is to introduce Redundancy, i.e., we store extra information that is needed normally but that can be used to rebuild the lost information.

The simplest approach to introduce redundancy is to duplicate every disk and techique is called Mirroring or Shadowing.
Logical disk contains 2 disks & every ~~rea~~ write is carried on both disks.
If one disk fails, datas can be read from other. Data will only be lost if the 2nd disk fails before the 1st disk is repaired.
Mean Time to Repair (MTTR) is the time it takes to replace a failed disk and restore data on it.

Suppose:- failure of two disks are independent,
MTTF of single disk is 100,000 hrs and MTTR is 10 hrs, then
Mean Time To Data Loss (MTTDL) of a mirrored disk system is

$$\frac{(100,000)^2}{2 \times 10} = 57,000 \text{ yrs. or } 500 \times 10^6 \text{ hrs.}$$

Mirrored disked system offer much reliability than single-disk system.

Two ~~blocks~~ disks written → consistent
Two disks unwritten → inconsistent
One disk unwritten + One disk written → consistent

} Power Failure Cases.

Improvement in Performance via Parallelism,

With multiple disks, we can improve the transfer rate as well by stripping data across multiple disks. Data stripping consists of splitting the bits of each byte across multiple disks, this is called Bit-level Stripping. If we use an array of 4 disks, bits i and 4+i of each byte go to disk i.

Block Level Stripping treats the array of disks as a single large disk. It gives blocks logical no.s starts from 0. With an array of n disks, B-L-stripping assigns logical block i of disk array to disk (i mod n) + 1; it uses $\lfloor i/n \rfloor$ th physical block of disk to store logical block i. When a single block is read, data-transfer rate is same as on one disk, but the remaining n-1 disks are free to perform other ~~[illegible]~~ action.

Two main goals of parallelism:-

i) Load-Balance Multiple small accesses, so that throughput of such accesses increases.

ii) Parallelize large accesses, so that response time of large accesses is reduced.

RAID Levels

Mirroring provides high reliability but it's expensive. Stiping provides high data transfer rates but doesn't improve reliability.

In the given figures.

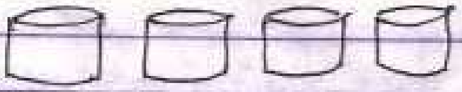
P → error correcting bits

C → second copy of data

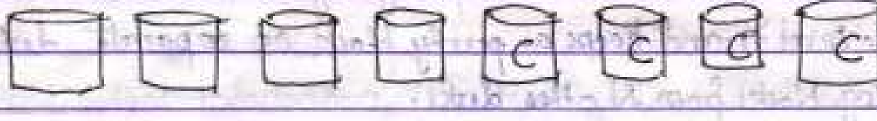
four disks are data worth and the extra disks are used to store redundant information.

• RAID Level 0: An array of size 4.

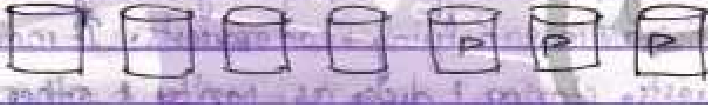
It refers to disk arrays with striping at level of blocks but without any redundancy.



• RAID level 1: It refers to disk mirroring with block striping. It shows a mirrored organization holding 4 disks data worth.



• RAID level 2: Called as Memory-style Error-Correcting-Code (ECC) organization. Memory system have long used parity bits for error detection & correction. Each byte have a parity bit that records whether the no. of bits in byte that are set to 1 is even (parity = 0) or odd (parity = 1). If one of the bits in byte gets damaged, i.e, 0 changes to 1 or 1 changes to 0, parity of the byte changes and willn't match with computed parity. Thus all 1-bit errors will be detected by memory system. ECC stores 2 or more extra bits, and can reconstruct the data.



Level 2 requires 3 disks overhead for four disks of data.

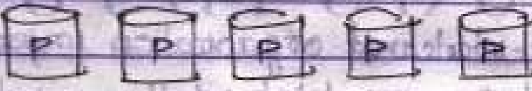
• RAID Level 3: It is bit-interleaved organization improves level 2 by the fact that disk-controllers can detect whether a sector has been read correctly, so a single parity bit can be used for error-correction. If one of the sector gets damaged, if the parity of the remaining bits = stored parity, the missing bit = 0, otherwise 1.
It is less expensive level as only one disk extra is used as overhead. So, it is beneficiary over level 2. Also in level 1, it needs one mirror disk for every disk
With N-way striping of data, transfer rate for reading or writing a single block is N times faster than level 1. Level 3 supports a lower no. of I/O opⁿ/sec since every disk has to participate in every i/o request.

Raid Level 4: It is block-interleaved parity organization uses block-level striping like level 0 and keeps a parity block on separate disk for corresponding blocks from N other disks.

Multiple read accesses can proceed in parallel, leading to a higher overall I/O rate. The transfer rates for large reads & writes is high, since all the disks can be read and can be written also in parallel.

A single write requires 4 disk accesses, 2 to read the old 2 blocks and 2 to write the 2 blocks.

RAID Level 5: It is block-interleaved distributed parity organization. It improves level 4 by partitioning data & parity among all N+1 disks. All disks can participate in satisfying read requests. It increases the total no. of requests, storing 1 disks as parity & other N for blocks.



Ex:- for 5 disks and parity block Pk for logical blocks 4k, 4k+1, 4k+2, 4k+3 is stored in disk (k mod 5)+1 : Following tables indicate 20 blocks numbered 0-19.

P0	0	1	2	3
4	P1	5	6	7
8	9	P2	10	11
12	13	14	P3	15
16	17	18	19	P4

4th level is not used in practice as 5th level offer better read-write performance at same cost.

RAID Level 6: P+Q Redundancy scheme just like level 5 but stores extra redundant information. Instead of using parity, it uses ECC. ~~In~~ the fig., 2 bits of redundant data are stored for every 4 bits of data - and system can tolerate 2 disk failures.

Choice of RAID Level
Factors taken into account in choosing RAID level:-
i) Monetary Cost of Extra-disk storage requirements.
ii) Performance requirements in terms of I/O op^n.
iii) Performance when a disk has failed.
iv) Performance during rebuild on a new disk.
Raid level 0 is used in high performance applications where data safety isn't critical. Bit striping (3) is inferior to block striping (5) since it has good data transfer rates for large transfers.
Level 6 isn't supported currently by many RAID implementations but it offers better reliability than level 5.
Read frequently and write safely → level 5
Storage of log files → level 1

Application of RAID
Array of tapes, broadcast of data over wireless systems.

Magnetic Tapes

These are limited to sequential access. They can't provide random access for secondary-storage requirements.

i> Used for backup

ii> storage of infrequently used information.

iii> as offline medium for transferring information from one system to another.

iv> Storing large volm of data like video, audio, images, etc.

Currently available tape capacities range from a few GB with Digital Audio Tape (DAT) format, 10-40GB with Digital Linear Tape (DLT) format, 100 GB and higher with Ultrium format, to 330 GB with Ampex Helical Scan format with a data transfer rate of 10 MBPS.

Good tape-drive systems perform a read of just-written data to ensure that it has been recorded correctly.

Storage Access

Assumption is that, no data items span 2 or more blocks. Major goal of database system is to minimize the no. of block transfer betn disk & memory. One way to reduce the no. of disk access is to keep as many blocks in main memory.

Buffer is that part of main memory available for storage of copies of disk blocks. The subsystem responsible for allocation of buffer space is called Buffer Manager.

Buffer Manager

When programs in database needs a block from disk, they make request to buffer manager.

If block is already in buffer → address of block passed to main memory.

If block isn't in buffer → allocates some space for new block throwing out some other block.

To serve the database system well, it must use techniques :-

i> Buffer Replacement Strategy

Least Recently Used (LRU) Scheme → block referenced least recently is written back to disk and is removed from buffer.

The goal is to minimize accesses to the disk.

Most Recently Used (MRU) strategy chooses the most recently used blocks (that are not eligible for replacement while they are being used).

ii) Pinned blocks :- A block i.e, not allowed to be written back to disk is said to be pinned. Such a feature is essential for a database system, resilient to crashes.

iii) Forced o/p of blocks :- Situation in which it is necessary to write back the block to disk, even though the buffer space isn't needed. This write is called forced o/p of blocks.

Introduction to Database

A database system provides a Data-Definition Language (DDL) to specify the database schema and a Data-Manipulation Language (DML) to express database queries and updates.

DML

It is a language that enables the user to access or manipulate data as organized by appropriate data model. The types of access are:-

- Retrieval of information stored in DB.
- Insertion of new information into DB.
- Deletion of information from DB.
- Modification of information stored in DB.

Two types of DML's :-

i) Procedural DML :- it requires a user to specify what data are needed and how to get those data.

ii) Declarative DML (Non Procedural) :- it requires a user to specify what data are needed without specifying how to get those data. It is more easier to learn than Procedural DML.

Query :- It is a statement requesting the retrieval of information. The portion of DML that involves information retrieval is Query Language.

DDL

We specify the storage structure and access methods used by DB system by a set of statements in a special type of DDL called Data Storage & Definition language. These define the implementation details of DB schemas usually hidden from users.

A constraint can be an arbitrary predicate pertaining to database. DB System concentrate on integrity constraints that can be tested with minimal overhead :-

i) Domain Constraints :- A domain of possible values must be associated with every attribute. It is the most elementary form of integrity constraint. They are tested easily by the system whenever a new data item enters into DB.

Introduction to Database

A database system provides a Data-Definition Language (DDL) to specify the database schema and a Data-Manipulation Language (DML) to express database queries and updates.

DML

It is a language that enables the user to access or manipulate data as organized by appropriate data model. The types of access are:-

- Retrieval of information stored in DB.
- Insertion of new information into DB.
- Deletion of information from DB.
- Modification of information stored in DB.

Two types of DML's :-

i) Procedural DML :- it requires a user to specify what data are needed and how to get those data.

ii) Declarative DML (Non Procedural) :- It requires a user to specify what data are needed without specifying how to get those data. It is more easier to learn than Procedural DML.

Query :- It is a statement requesting the retrieval of information. The portion of DML that involves information retrieval is Query Language.

DDL

We specify the storage structure and access methods used by DB system by a set of statements in a special type of DDL called Data Storage & Definition language. These define the implementation details of DB schemas usually hidden from users.

A constraint can be an arbitrary predicate pertaining to database. DB System concentrate on integrity constraints that can be tested with minimal overhead :-

i) Domain Constraints :- A domain of possible values must be associated with every attribute. It is the most elementary form of integrity constraint. They are tested easily by the system whenever a new data item enters into DB.

ii) Referential Integrity : There are cases that ensure a value that appears in one relation for a given set of attributes also appears for certain set of attributes in another relation.

iii) Assertion :- It is any condition that DB always satisfy. The above two constraints are special form of it.

iv) Authorization :- We want to differentiate among users as far as type of access they are permitted on various data values in DB, these differentiation are expressed in terms of authorization.

- Read authorization :- allows reading, not modification of data.
- Insert authorization :- allows insertion, not modification
- Update " :- allows modification, not deletion.
- Delete " :- allows deletion

O/P of DDL is placed in Data Dictionary, which contains Metadata - data about data. It is special type of table which can only be accessed and updated by DB system itself.

Database Access from Application Programs :

Application Programs are programs used to interact with DB in languages such as C, C++, Cobol or Java, There are 2 ways in which DML statements needs to be executed from host language :- $\rightarrow$ host lang.

i) providing an Application Pgm Interface used to send DDL & DML to DB to retrieve the results.

ODBC (Open DB Connectivity) standard started by Microsoft using C and JDBC (Java DB Connectivity) using Java are two such examples

ii) extending host language syntax to embed DML calls. Usually DML precompiler is used which converts DML statements to normal procedural calls in host language.

Data Storage and Querying.

Functional components of DB system can be broadly divided into :-
Storage Manager and Query Processor.

$$\begin{tikzcd} \text{Storage Manager} \arrow[d, Rightarrow] & \text{Query Processor} \arrow[d, Rightarrow] \\ \text{require due to large amount of storage space.} & \text{require due to simplify \& facilitate access to data.} \end{tikzcd}$$

Storage Manager

It is a program that provides interface bet^n low-level data stored in DB and application pgm and queries submitted to system. It translates the DML statements into low-level file system commands. It includes:-

i) Authorization and Integrity Manager :- test for satisfaction of integrity constraints and checks the authority of users to access data.

ii) Transaction Manager :- ensures DB remains in a consistent state despite system failures.

iii) File Manager :- manages the allocation of space on disk storage.

iv) Buffer Manager :- responsible for fetching data from disk into main memory and deciding what data to cache in main memory.

It implements several data structures :-

i) Data files :- Stores the DB.

ii) Data Dictionary :- stores the metadata.

iii) Indices :- provides fast access to Data items.

The Query Processor

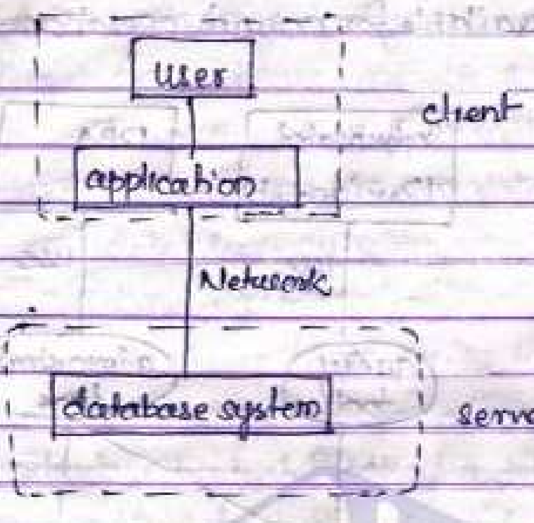
i) DDL interpreter :- interprets DDL statements and records definition in dictionary.

ii) DML Compiler :- translates DML statements in a query lang. It also perform Query optimization.

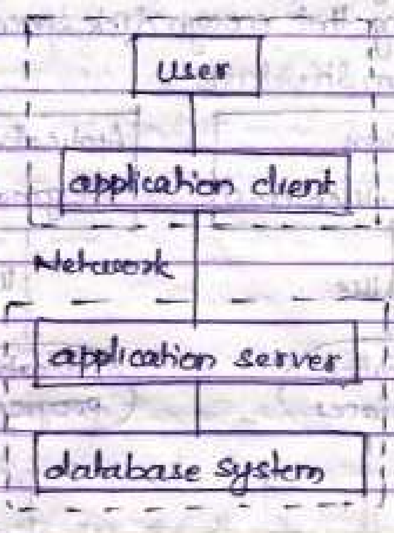
iii) Query Evaluation Engine :- executes low-level instructions generated by DML compiler.

Database Architecture

Two-tier architecture



Three-tier architecture



Database Users and User Interfaces

Four types of DB Users, differentiated by the way they expect to interact with the system:-

i) Naive Users :- Unsophisticated users interact with the system by invoking one of application pgm that is written previously.

ii) Application Programmers :- Computer professionals who write application pgm's. Rapid Application Development (RAD) tools are used to construct forms & reports with min^m programming effort.

iii) Sophisticated Users :- interact with the system without writing programs. They form their request in a Query language.

iv) Specialized Users :- Sophisticated users write specialized DB applications that don't fit into traditional data-processing framework.

Database Administrator (DBA)

A person having ~~control~~ central control over system. Functions of DBA:-

i) Schema definition

ii) Storage access and method definition.

iii) Schema & Physical organization modification

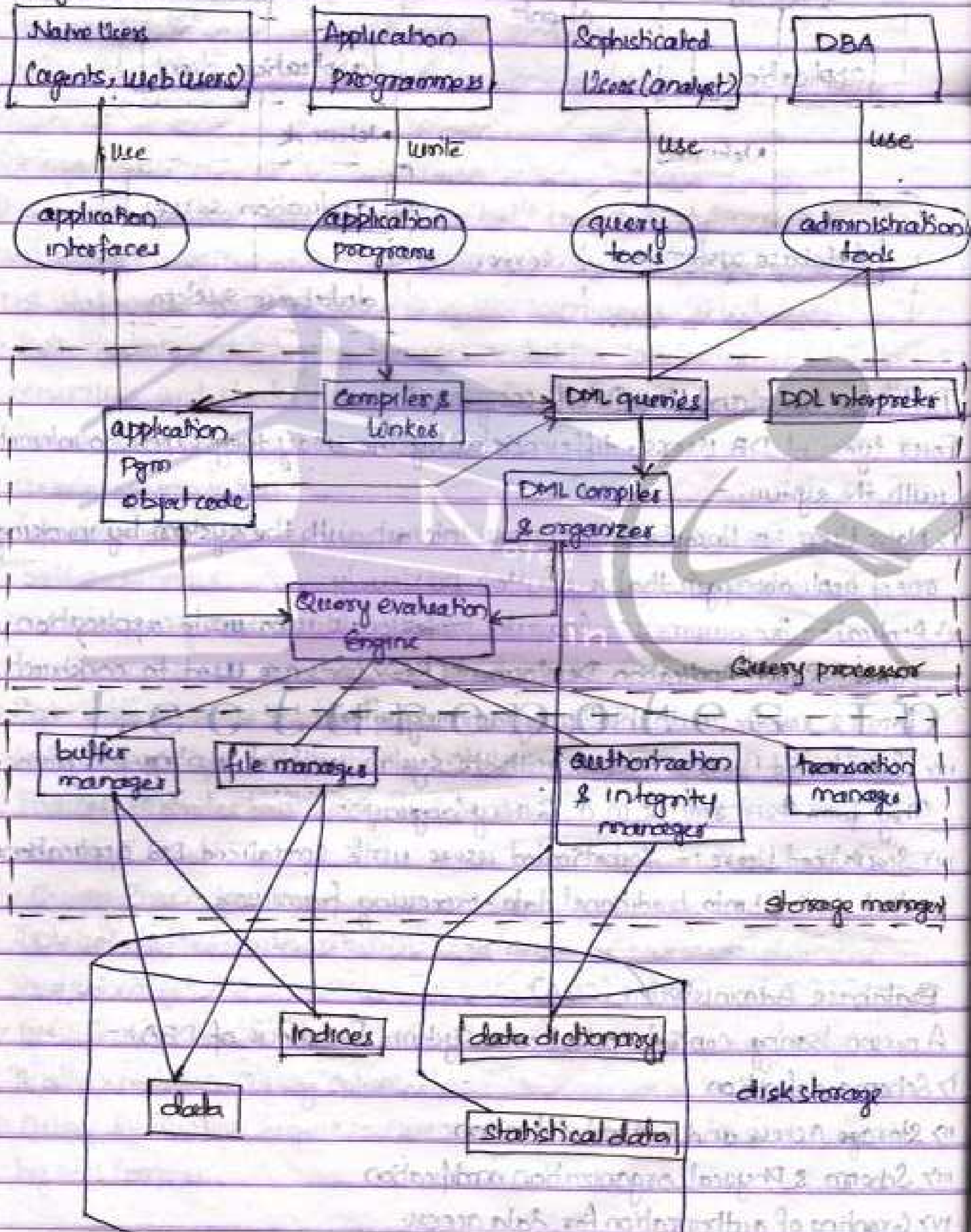
iv) Granting of authorization for data access.

v) Routine Maintenance

• Backup of DB into tapes & disks to prevent loss of data.

• ensuring that performance isn't degraded by very expensive tasks.
• ensuring that enough disk space is available for normal operations.

System Structure



ADVANCED SQL.

Built-in Data Types in SQL

i) date :- containing 4 digit yr., month, day.

ii) time :- time in hours, minutes & secs. A variant time(p) is used to specify no. of fractional digits for seconds.

iii) timestamp :- a combination of date & time

date '2001-04-25'

time '09:30:00'

timestamp '2001-04-25 10:29:01.45'

To extract individual fields of a date or time valued, we can use extract (field from d), where field can be yr., month, day, hour, minute, sec.

localtime → current local time without time zone.

current_time → current time with time zone.

User-defined Types

i) distinct types

ii) structured data types

both allows the creation of complex data types with nested record structures, arrays and multisets.

drop types and alter types clauses are used to drop or modify types that have been created earlier.

Large-Object Types

For character data → clob (10 KB)

For binary data → blob (10MB)

lob refers to "Large OBject".

Ex:- book-review ~~clt~~ clob(10 KB)

image blob(10 MB)

movie blob(2 GB)

Schemas, Catalogs & Environments

Contemporary DB system provide a 3-level hierarchy for naming relations. The top level consists of catalogs, each of which contain schemas.

Ex:- catalog5. bank_schema. account

Let catalog5 be the default catalog, we can use bank_schema. account to identify the same relation uniquely.

We can use account if the default catalog is catalog5 and default schema is bank_schema.

The default catalog and schema are part of an SQL environment. The environment additionally contains user identifier or authorization identifier.

Embedded SQL

A programmer must've access to a DB from a general purpose programing lang for @least two reasons:-

i) Not all queries can be expressed in SQL. Queries that can be expressed in C, Java can't be expressed in SQL. To write such queries, Embed SQL within a powerful lang.

ii) Printing a report, user interaction, sending the results of a query to GUI can't be done from within SQL.

A language in which SQL queries are embedded is called Host language. To identify embedded SQL requests to preprocessor, we use :-

EXEC SQL <embedded SQL statement> END-EXEC

; Is also used instead of END-EXEC to end the statement in C.

For JAVA embedding of SQL (SQLJ):-

SQL { <embedded SQL statement > };

Before executing SQL statements, pgm must connect to DB.

EXEC SQL connect to server user user-name END-EXEC

Dynamic SQL

The Dynamic SQL component of SQL allows pgm to construct and submit SQL queries at run time, but embedded SQL statements must be completely present at compile time.
It contains a "?", which is a placeholder for a value, i.e., provided when SQL pgm is executed.

ODBC

ODBC standard defines a way for an application to communicate with a DB server. It defines an API (Application Pgm Interface) that applications can use to open a connection with DB, send queries and updates and get back results.
Applications like GUI, statistics package, spreadsheets can make use of same ODBC API.

Other Relational Languages

Tuple Relational Calculus (TRC)

It is a non-procedural query language. It describes the desired information without giving a specific procedure.
A query in TRC is expressed as:-
$\{t \mid P(t)\}$
It is a set of all tuples t such that P is true for t.

Ex:- Find branch-name, loan-no. & amount for loans over \$1200.
$\{t \mid t \in loan \wedge t[amount] > 1200\}$

Ex:- Find the loan no. for each loan of an amount > \$1200.
$\{t \mid \exists s \in loan\ (t[loan\_no.] = s[loan\_no.] \wedge s[amount] > 1200)\}$

Ex:- Find the names of all customers who have a loan from Perryridge branch.
$\{t \mid \exists s \in borrower\ (t[customer\_name] = s[customer\_name] \wedge \exists u \in loan\ (u[loan\_no.] = s[loan\_no.] \wedge u[branch\_name] = \text{"Perryridge"}))\}$

Que:- Find all customers who have an a/c at the bank but don't have a loan from bank.

$\{t \mid \exists u \in depositor\ (t[customer\_name] = u[customer\_name])$
$\wedge \sim \exists s \in borrower\ (t[customer\_name] = s[customer\_name])\}$

Que:- Find all customers who've an a/c at all branches located in Brooklyn.

$\{t \mid \exists r \in customer\ (r[customer\_name] = t[customer\_name]) \wedge$
$(\forall u \in branch\ (u[branch\_city] = \text{"Brooklyn"} \Rightarrow$
$\exists s \in depositor\ (t[customer\_name] = s[customer\_name] \wedge \exists w$
$\in account\ (w[account\_no.] = s[account\_no.] \wedge w[branch\_name] = u[branch\_name]))))\}$

Formal Definition

A tuple variable is said to be a free variable unless it is quantified by a $\exists$ or $\forall$.

$t \in loan \wedge \exists s \in customer\ (t[branch\_name] = s[branch\_name])$

where $t \rightarrow$ free variable
$s \rightarrow$ bound variable

A TRC formula is build of atoms. An atom has one of following forms:-

i) $s \in r$, where s is a tuple variable and r is a relation (we don't allow $\notin$ operator).

ii) $s[x]\ \theta\ u[y]$, x and y are attributes on which s and u are defined. θ is a comparison operator ($<, \leq, =, \neq, >, \geq$)

iii) $s[x]\ \theta\ c$, c is a const. in the domain of attribute x.

We build formula from atoms by following rules:-

i) an atom is a formula.

ii) If P_1 is a formula, then so are $\sim P_1$

iii) If P_1 & P_2 are formula, then $P_1 \vee P_2$, $P_1 \wedge P_2$ and $P_1 \Rightarrow P_2$.

iv) $P_1(s)$ is a formula and r is a relation,

$\exists s \in r(P_1(s))$ and $\forall s \in r(P_1(s))$

TRC Rules

- $P_1 \wedge P_2 = \sim(\sim(P_1) \vee (\sim(P_2)))$
- $\forall t \in r (P_1(t)) = \sim \exists t \in r (\sim P_1(t))$
- $P_1 \Rightarrow P_2 = \sim(P_1) \vee P_2$

$\{t \mid P(t)\}$ is safe if all values that appear in the result are values from dom(P).

DRC

It uses domain variables that take on values from an attributes domain, rather than values for an entire tuple.

It is of the form:-

$$\{<x_1, x_2, \ldots, x_n> \mid P(x_1, x_2, \ldots, x_n)\}$$

$x_1, x_2 \ldots$ domain variables or domain const.

An atom in DRC has following forms:-

i> $<x_1, x_2 \ldots x_n> \in r$, where r is a relation on n attributes

ii> $x \Theta y$, x & y are domain variables.

iii> $x \Theta c$

As a shorthand, we write $\exists a, b, c (P(a,b,c))$ for
$\exists a (\exists b (\exists c (P(a,b,c))))$.

Some ex:- for DRC

1> $\{<l,b,a> \mid <l,b,a> \in loan \wedge a > 1200\}$

2> $\{<l> \mid \exists b, a (<l,b,a> \in loan \wedge a > 1200\}$

3> $\{<c,a> \mid \exists l (<c,l> \in borrower \wedge \exists b (<l,b,a> \in loan \wedge b$
$=$ "Perryridge"))}

5> $\{<c> \mid \exists s, t (<c,s,t> \in customer) \wedge$
$\forall x,y,z (<x,y,z> \in branch \wedge y =$ "Brooklyn" $\Rightarrow$
$\exists a, b (<a,x,b> \in account \wedge <c,a> \in$
depositor))}

An expression $\{<l,b,a> \mid \sim(<l,b,a> \in loan)\}$
is unsafe, because it allows values in the result that aren't in the domain of expression.

An expression $\{<x_1, x_2, \ldots, x_n> \mid P(x_1, x_2 \ldots x_n)\}$ is safe if :-

i> all values that appear in tuples of exp. are values from dom(P).

ii> for every '$\exists$' of form $\exists x (P_1(x))$, subformula is true $\Leftrightarrow$ there is a value x in dom (P_1) such that $P_1(x)$ is true.

iii> for every '$\forall$' of form $\forall x (P_1(x))$, subformula is true $\Leftrightarrow$ if $P_1(x)$ is true for all values x from dom (P_1).

QBE

It has two distinctive features:-

i> It has 2-D Syntax. Queries look like tables.

ii> Queries are expressed "by example".

Two flavours of QBE are:-

i> Original text-based version

ii> Graphical version

Ex:- to find all loan no.'s at Perryridge branch.

loan	loan no.	branch_name	amount
	P._x	Perryidge	

It prints the value of variable x.

Again, if a variable doesn't appear more than once in a query, then.

	P.	Perryidge	

To suppress duplicate elimination,

	P.ALL.	Perryidge	

.in

To display entire loan relation.

P.				

Find the loan no. of all loans with an amount >$700

	P.		>700	

QBE supports =, <, ≤, >, ≥ and ¬.

Find the names of all branches that aren't located in Brooklyn.

Branch	branch_name	branch_city	assets
	P.	¬Brooklyn	

Find loan no.s of all loans made jointly to Smith and Jones.

Borrower	customer_name	loan_no.
	Smith	P._x
	Jones	_x

Find all customers who live in same city as Jones.

customer	customer_name	customer_street	customer_city
	P._x		_y
	Jones		_y

Lecturenotes

Queries on Several Relations

Find the names of all customer who've both an a/c and a loan @ the bank.

depositor	customer_name	account_no.
	P._x	

borrower	customer_name	loan_no.
	_x	

The Condition Box

QBE allows logical expressions to appear in a condition box, like, and, or, "&", "|".

Find all a/c no. with a balance betn $1300 & $1500.

a/c	a/c_no	branch_name	balance
	P.		_x

conditions
_x ≥ 1300
_x ≤ 1500

with a balance betn $1300 & $2000, but not exactly $1500.

conditions
_x = (≥1300 and ≤2000 and ¬1500)

The Result Relation

Find the customer_name, a/c-no. and balance for all a/c @ Perryridge branch.

a/c	a/c-no	branch-name	balance
	-y	Perryidge	-z

depositor	customer name	a/c-no
	-x	-y

result	customer_name	a/c_no.	balance
P.	-x	-y	-z

lecturenotes.in