11/1/19

→ We come to DBMS from file system because ~~becaus~~ it takes more memory and uses many resources which make other processes slow, so that other processes get enough room for its processing in main memory, we store data in secondary memory. This is the main reason we move to DB.

→ Traditional Database Model

→ ER model (Entity - Relationship model)

21/1/19

DB → Empty book (Raw data)
DBMS → Pen (How to write, what will b its contents, etc)

→ Images, videos, etc form of data → multimedia D
→ GIS based data (maps, etc) → spatial DB (postgr SQL
→ Real time data (~~not~~ efficient or not, but we need in real time) → real time DB.

→ Create conceptual view :-
Before designing a database a model is constructed. Eg., before construction of a building :-
① A chart is constructed
② Its real model

③ Conceptual Implementation

In DBMS 3 steps of modelling is done.

1. Conceptual view
2. Representational view
3. Logical level or physical data models

① Conceptual or high level view

To explain to naive users, here we used diagram for entities, attributes and relationship between entities.

Eg:- Student · course

Entities → noun

Relationship → verb.

E-R model

→ It is the most popular & conceptual model because it is easy & clear to naive user & easy to understand

② Representational view (Implementational view)

→ Relational model

→ It is used by developers

→ It tells how SQL query runs or any query runs on programming language.

Here we
→ Generally use tables.

③ Logical level or physical data models.
→ It tells how the tables are stored in the memory device.
→ So to deal about the structure of records i.e, how to store the data type, how to place the record, how many bytes are required to retrieve the data.

Assignment :-

① Create an E-R diagram for the cookie database.
② Name the entities and relationship along with their attributes using the ER notations.
③ Define the relationships between various entities (one-one, one-many, many-many).
By "entity" one should understand the table names in the cookie db (ie, BIB FILE, PUBFILE, et
④ Indicate minimum & maximum cardinalities for each entity.

I A company db needs to store info about employees (ssn, salary, phone); departments (dno, budget, dname) and children of employees (name, age). Draw an ER diagram

Set name → Entity type
Set elements (instance) → Entity

Employee → Entity type
Employee name → instance (Entity)

Thing → Entity
Properties → attributes
Relationship → Association

① Difference b/w Entity and Entity type
→ Entity type is the schema which is going to represent the type of entity.
→ Entity is a person who is going to be represented in entity type itself

② → Entity type → Heading
→ Entity → instance of that heading

③ → Entity type → Intension
→ Entity → Extension

④ Entity might get added or deleted

⑤ With respect to set theory,
Set name → Entity type
Set elements → Entity

25/1/19

Attribute -

↳ single valued & Multivalued. (that can have more than one values)

↓ Age.

↓ account numbers
phn no.

↳ simple v/s Composite (which can further be divided).

↳ Age
↳ Gender

↳ Name
Phno.
~~Address~~

→ Stored & Derived

↓ We have to store it in ~~database~~
+ Gender.

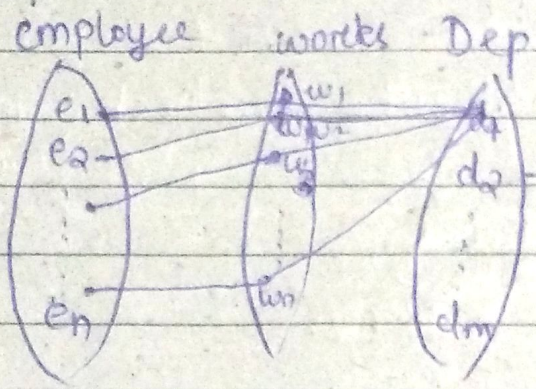
↓ We can derive from the values stored in DB.
+ Age (from DOB).

Q Employee → 2 headquaters

Name → Single, Composite, stored
Address → Multivalue, Composite, stored
Phno → Multi, Composite, Stored.
Age → Single, Simple, Derived
DOB → Single, composite, stored.

Relationships:

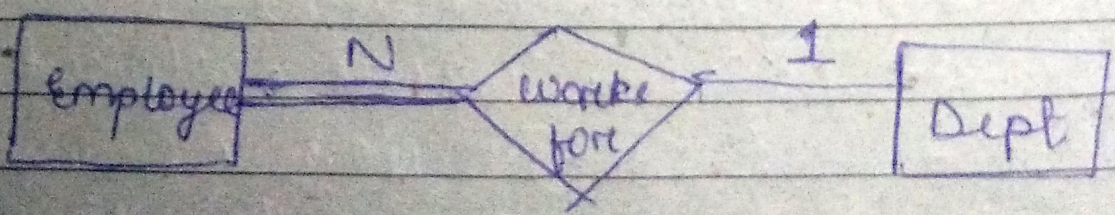
Q. Every employee works for a Dept. and every dept. can have many employees and new dept. need not have any employee.



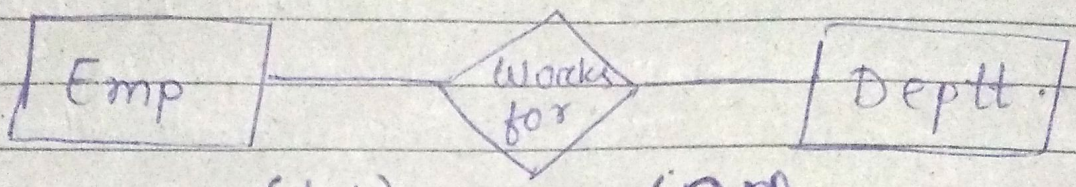
Degree:- 2. (How many entities are there for that relation)

Cardinality: Employee → N
Dept → 1
{ ~~One to one~~ ~~one to many~~ many to one }

Participation / Existence / Structural constraint:
Employee → total.
Dept → partial.



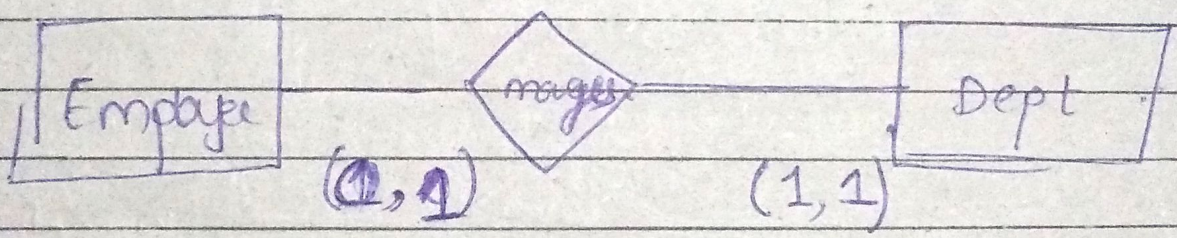
Min-Max Representation

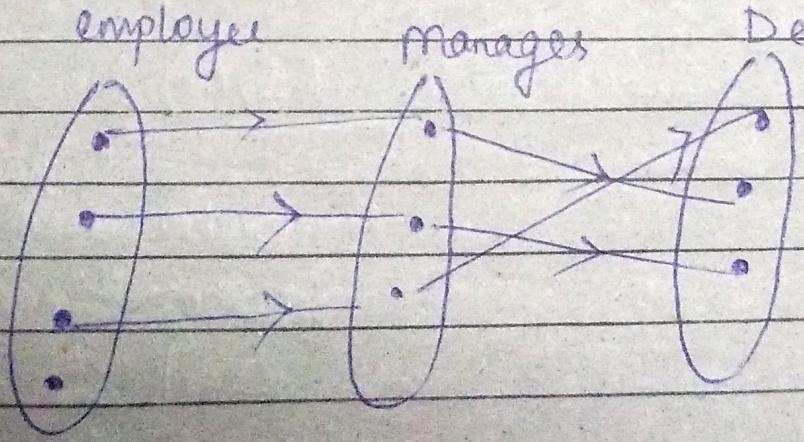


(1,1) (0,N)

min max {first participation and cardinality}.

Q Every deptt. should have a manager and only one employee manages a deptt. and 1 employees can manage only one dept.

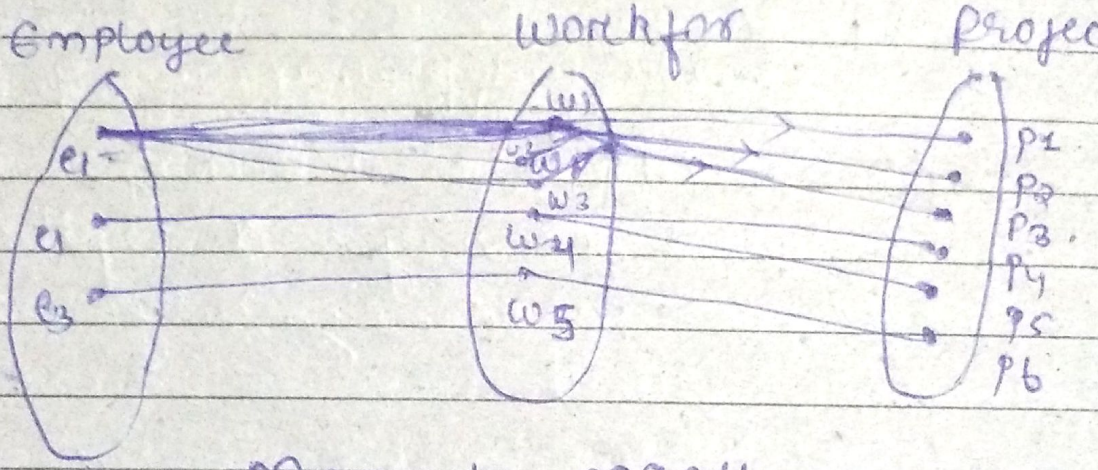




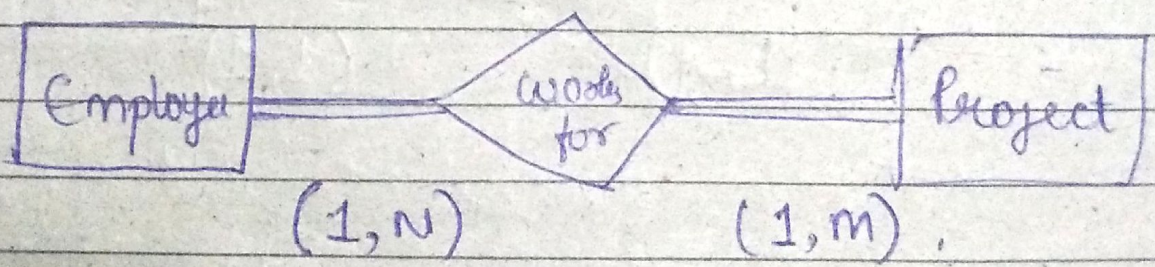
{one to one}.

Degree = 2.

(3) Every employee is supposed to work for atleast 1 project.
An employee can work on many projects
A project can have many employees and every project is supposed to have atleast one employee.



Many to many

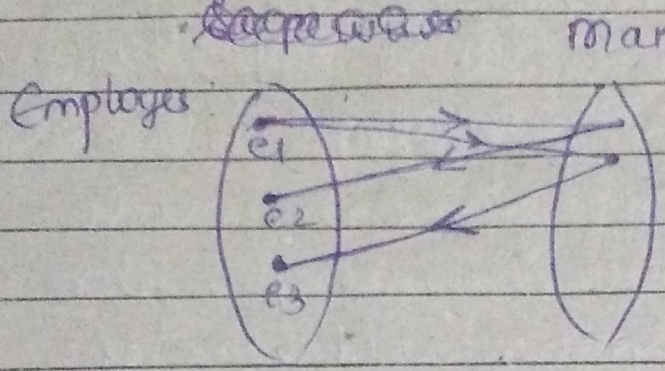


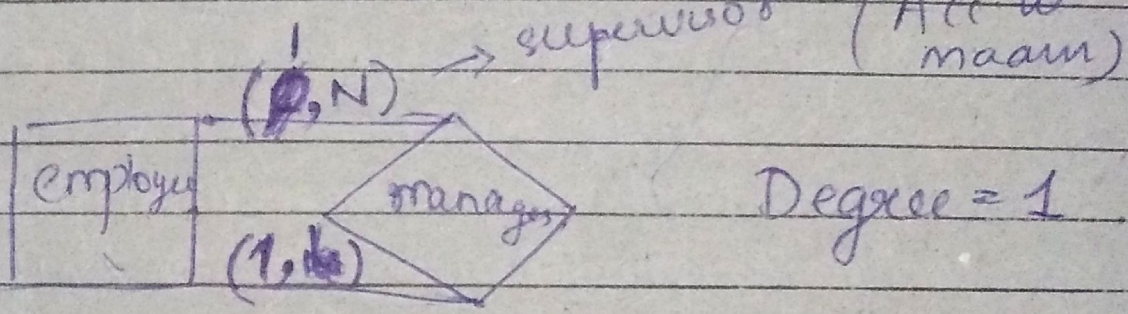
Degree = 2.

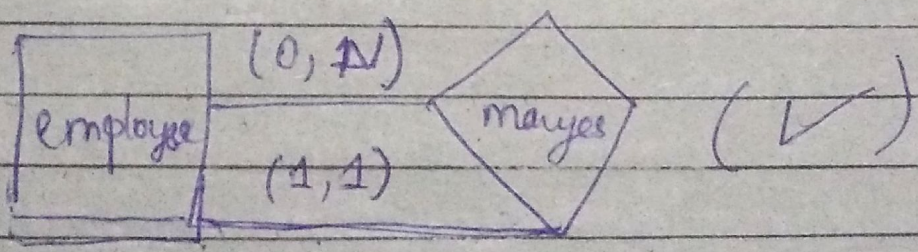
{Both min-max ~~relationship~~ and structural representation}

Recursive Relation

~~Employee~~ Supervisor manages Supervisee.



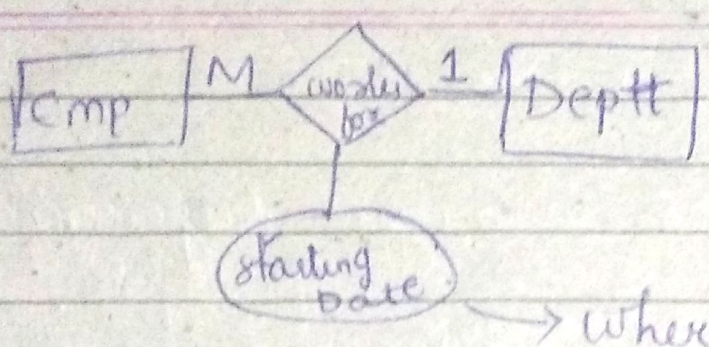




(✓)

28/1/19

Whenever we have many to one relationship, we will shift the descriptive attributes towards the many side.



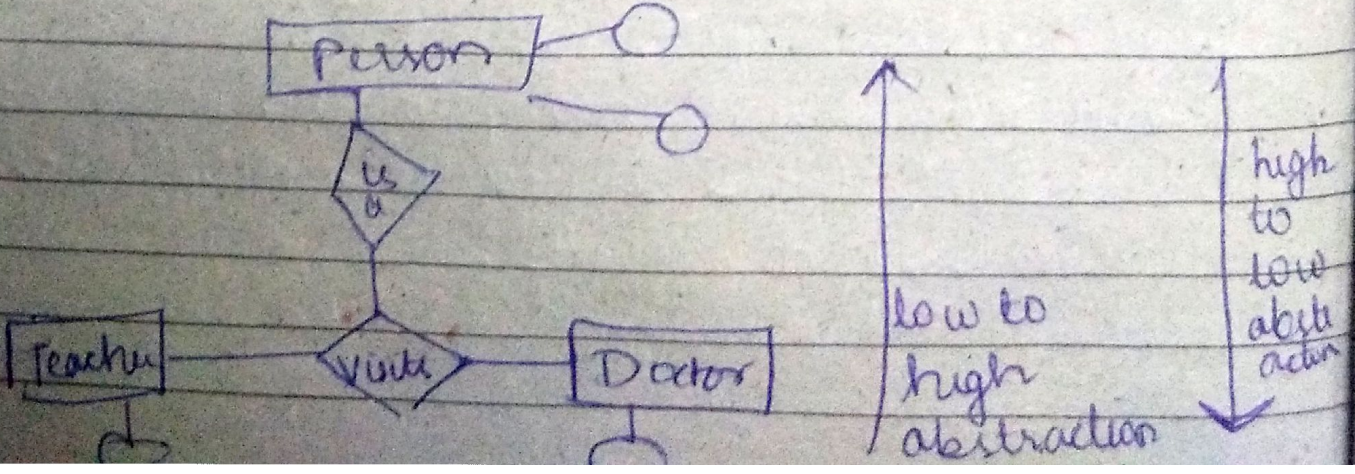
→ where to shift this descriptive attribute

Then we will shift towards many side.
If we have one-to-one relationship, then we can shift the descriptive attributes towards any side.

If we have many to many relationship, then we can't decide which side to shift.
For this we have;

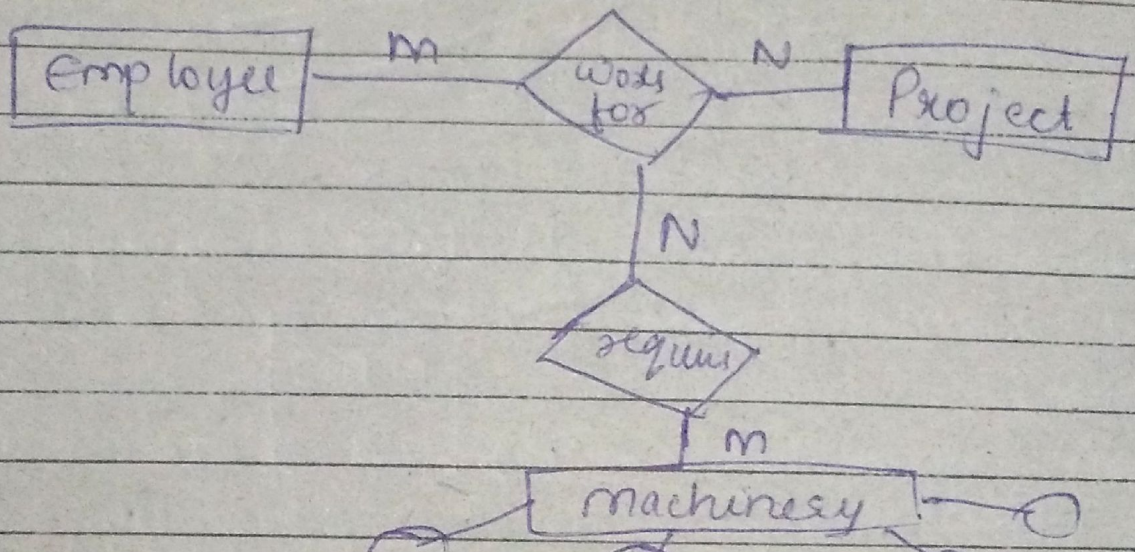
① Generalization - (moving from lower abstraction level to high abstraction level)
② Specialization.
③ Aggregation

Specialization - moving from higher abstraction level to lower abstraction level.



Aggregation

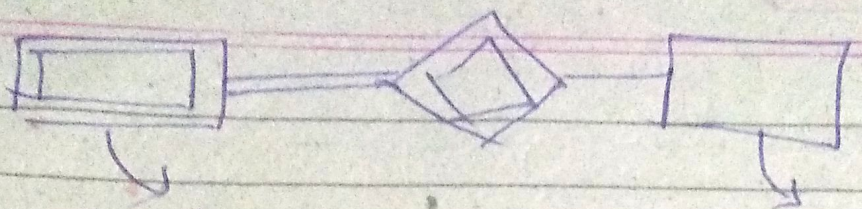
→ To understand better, we further divide relation into attributes and entities, to understand the detail better.



→ Designing is very important part. So ER diagram is necessary for this.

→ First we choose database models and then we choose DBMS (software).
Relational model is chosen most in the market because it is standardized.
So for RDMS, we have many DBMS.

→ If our model is standardized, then we need not have to train our employee (as if technology changes, model remain same).



→ If there is a weak entity, then there will be a total participation with relation
→ But vice-versa is not true.

Various database models.
① Relational model.
② Object-oriented model
③ Hierarchical model
④ Network model.
⑤ E R model.

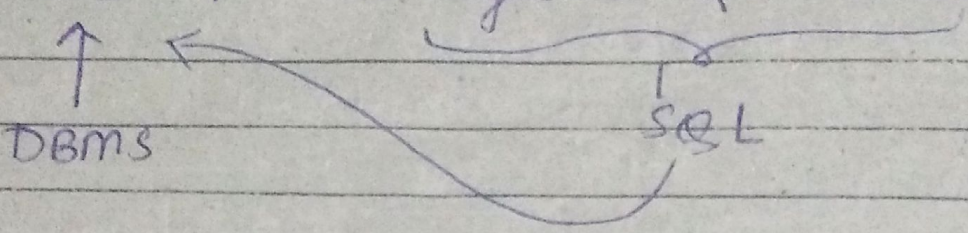
Properties of relational model
① Tabular form
② Rows → tupple columns - attribute.
③ Use SQL queries to retrieve, access, change, dat
④ Composite attributes are not allowed
⑤ ~~conceptual level~~ ~~database~~ Implementation level mo

During design time also we can set constraints
domain constraints, key constraints,
authorization " , referential integrity
assertion " constraint

Relational database can be commercially used.

Relational Model

RDB + Rel algebra & calculus.



→ Relation - Table

⟶ Tupples / Rows / Records & Attributes / fields / cols.

Domain → All attributes are defined over a specific domain.

Relational schema

↳ $R \rightarrow (A_1, A_2, A_3 \cdots A_n)$

$A_1 \uparrow D_1$, $A_2 \uparrow D_2$, $A_3 \uparrow D_3$, $A_n \uparrow D_n$

Size = Cross product of all the domains

$D_1 \times D_2 \times D_3 \times D_4 \cdots D_n$

Constraints → Integrity constraint

- Entity Integrity Constraint (Primary Key) — (Every record is identified by a unique identification number)
- Referential Integrity constraint (Foreign key).

29/1/19

Constraints

→ Domain constraints (constraint on Domain of attributes)
→ Key constraints
→ Entity constraints (In the primary key, value shall not be null)
→ Referential Integrity constraints (reference should be integral)

Key constraints

→ There are some attributes with key associated to it.

Super key :- set of all attributes that uniquely define a tupple

Candidate key :- minimum set of attributes that uniquely define a tupple
↳ Minimal super key

If we get a key by extraction to smaller units → minimal
If we choose minimum count then it is minimum key

eg: SK: A_1A_2 $A_1A_3A_4$

Two superkeys
minimal → both
minimum → A_1A_2

Semantic constraints during
→ which you apply ~~after~~ the implementation of the database.
→ You cannot apply during the designing of the database.
eg:- Salary should be greater than 15000.

Violations

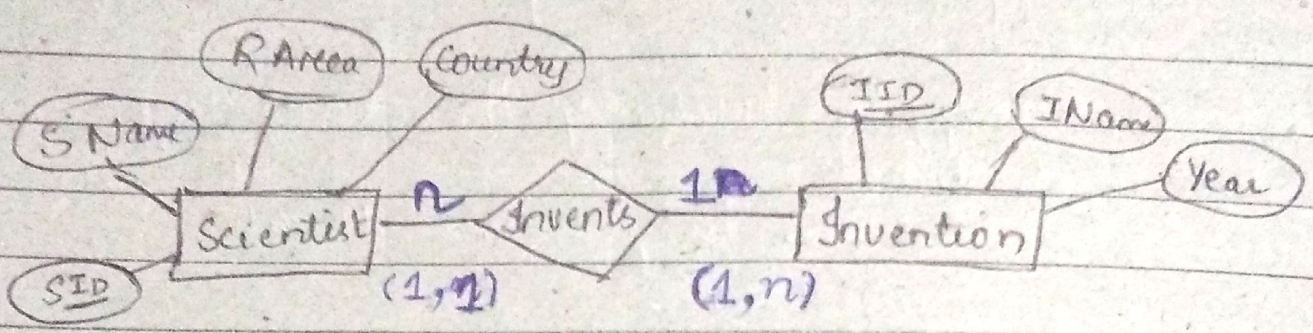
→ Insertion
→ Update (Del + insertion)
→ Delete

→ Insertion can violates domain constraint (like instead of int, it inserts varchar).
→ Key constraint → Should be unique while inserting.
→ Entity constraint → should be unique while inserting.
→ Referential Integrity → should be there in the referred table while insertion.

Delete

DC → No effect
KC → "
EC → "
RIC → {If we delete value in referred table, then we should delete or set null in refer table}

Q. Convert the given ER diagram :-



Include IID of invention as the foreign key of scientist

Scientist (SID, SName, RArea, Country, IID)

Invention (IID, IName, Year).

(OR)

Include SID of scientist as foreign key of invention.

Here considering SName as composite attribute and RArea as multi valued attribute so many to many relationship. Considering

① Scientist → strong entity

② Invention → " "

Scientist (SID, SName, RArea, Country, IID)

Invention (IID, IName, Year)

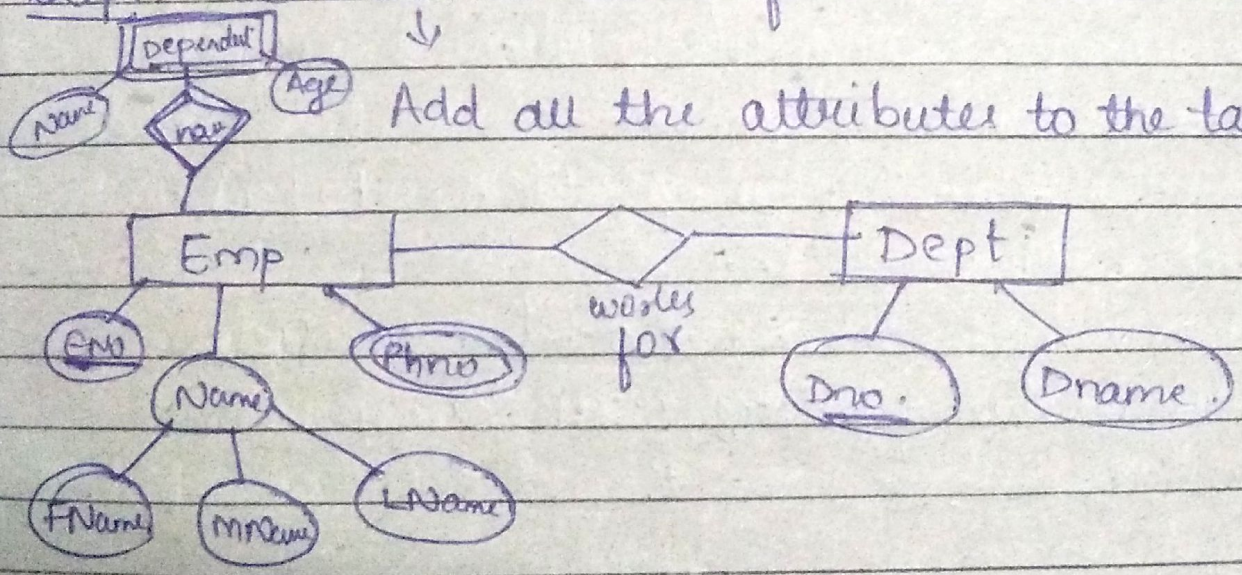
Considering many to many relationship.

Scientist (SID , SName , RArea , country)
Invention (IID , IName , Year)
Invents (IID , SID) & this is because we are considering many to many relationship.

01/2/19

Converting the ER Diagram/Model to Relational Schema.

Step1:- Create the table for all entities
↓
Add all the attributes to the table.



Eno	fname	lname	Mname

Dno	Dname

Step 2 :- Dealing with weak entities.

foreign key Dependants

ENO	Name	Relation

Here name is the partial key

Step 3 :- Converting the relationships to table :

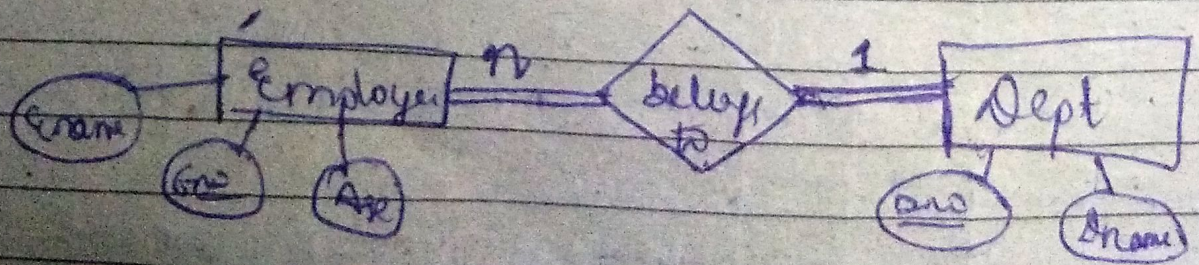
(i) Lets talk about 1:1 relationship. only one

eg :- Only one employee can manage dept. and each dept is managed by an employee.

→ Add foreign key towards the total participation side.

→ There is no problem to add towards other side but there will be space wastage.

(ii) 1 : m relationship (one to many relationship)



employee

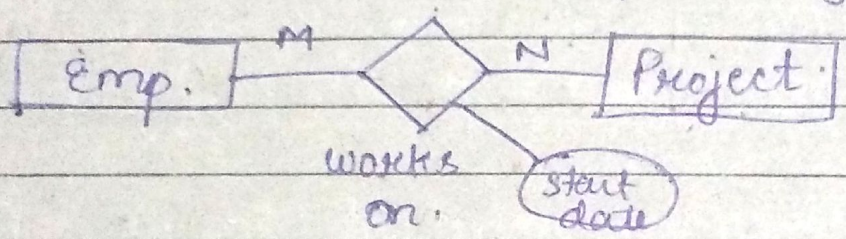
Eno	Ename	Age	Dno

Department

Dno	Dname

We will add foreign key towards the employee side. or we can say many side.

(iii) M:N relationship (many to many relationship)



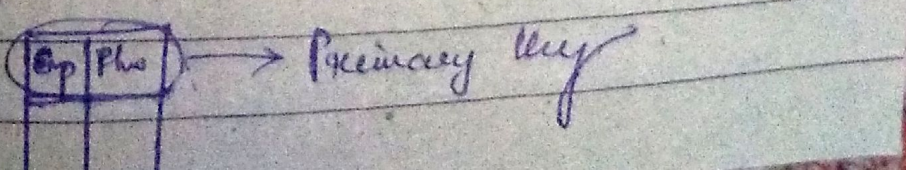
Here we will make a separate table with primary key as Empno and Project no combined.

Eno	Dno	Startdt

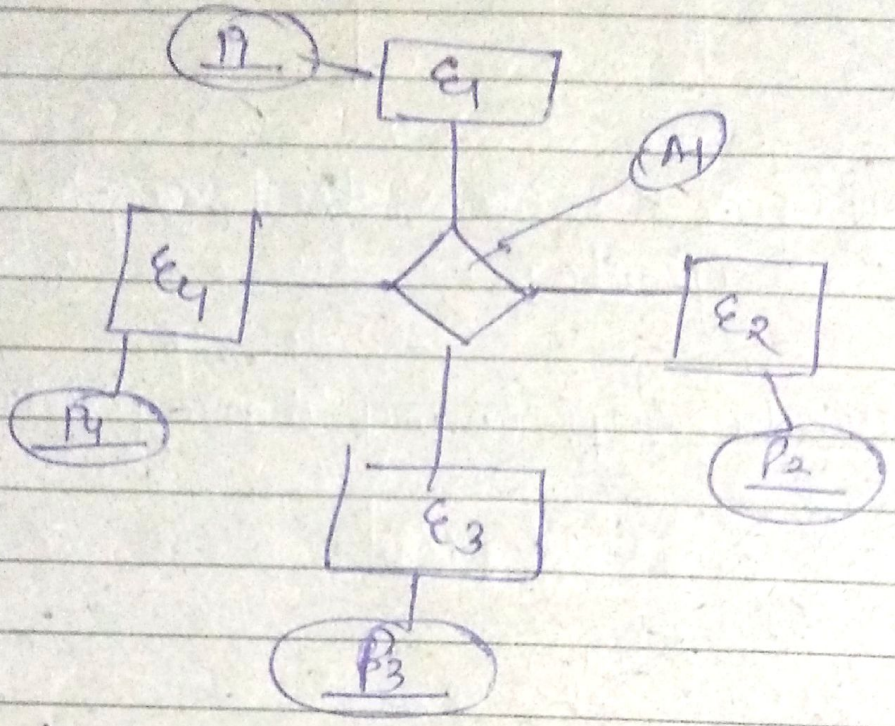
The attributes of the relation will be added to that table.

Step 4. Multivalued attribute

We will another table for multi-valued attribute



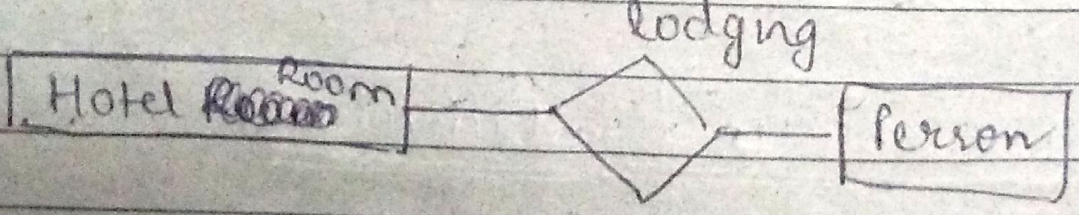
Step 5: m-ary relationships.



We will make a separate table with P_1, P_2, P_3 and P_4 and also attributes of that relation.

P_1	P_2	P_3	P_4	A1

9.



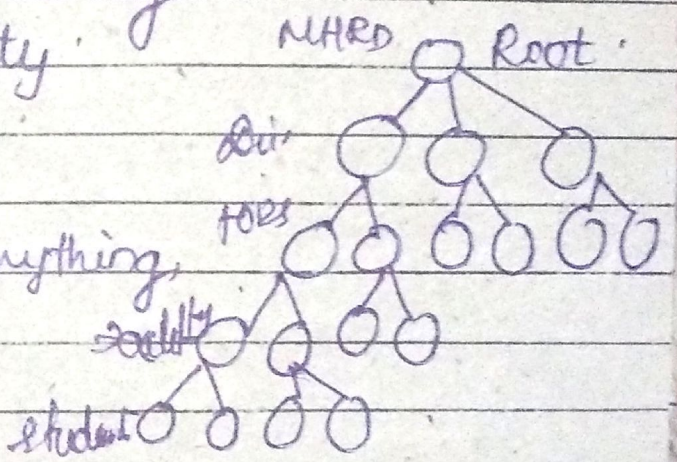
Lodging is a m:m relationship.
Rent is a payment to be made by a person occupying different hotel rooms,

should be added as an attribute to
① Hotel Room ② Lodging ③ Person ④ Null

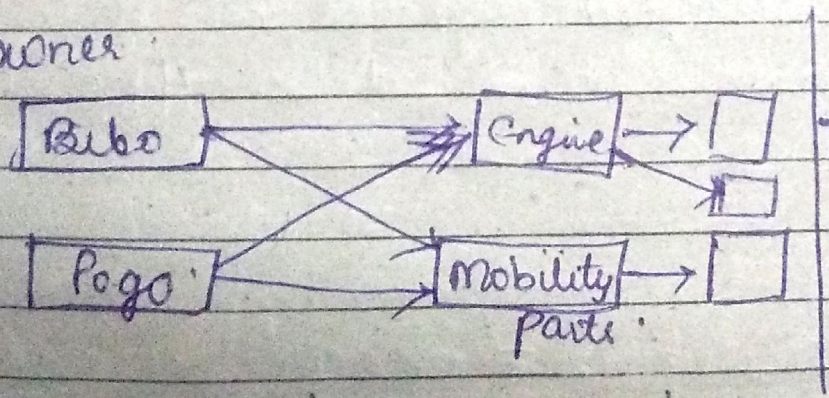
01/02/19

Hierarchical Model

↳ Data have been classified ~~in some kind of~~ hierarchy system.
→ This is do because when data becomes large it is very difficult to manage.
↳ Adv → Data integrity
→ Search is easier
Disadv → Less flexible.
→ We cannot add anything anywhere



Network model



Adv → Flexibility got increase

Disadv → It got complicated.
→ Slow down the database.

PAGE NO.
DATE

Object oriented model

↳ Extensions to ER model
→ Entities and relationship.
↳ Similar kind of objects are put in one class.

5/8/19

Varchar2 → range is larger than varchar

CHECK → check values that should be in list

Alter → change in schema, we can say (not the tupple value)

Restruct → if we are deleting column, then other should not be affected thats why we use restruct

Auto increment
↳ Max value = 10000
↳ There should not be cycle, after value reaches 10000 the value automatically should not be incremented to 1

Data dictionary :- data about data.

↳ All kind of info is stored is data dictionary because schema is entered in data dictionary.

↳ This is stored in hard disk.

Aliasing

Select Cust.CustomerName As Name from Customer As Cust

- As Name → Alias for column name.
- As Cust → Alias for table name

→ AND should be evaluated before OR in any boolean expression.

CHECK ⟹ See on your own.

Customer IN 'A' OR Customer IN 'B' OR - - - -

↓

Can be written as :-

Customer IN ('A', 'B', 'C' - - -)

h

→ Scalar aggregate
Vector aggregate

→ Views. (Read on ur own)
→ Can there be nested views?
→ can we create a view from view

→ Can view reside in database.
→ Advantages & Disadv. of views.
→ If we make changes in view, will it be reflected in table?

—x—

08/02/19

Views

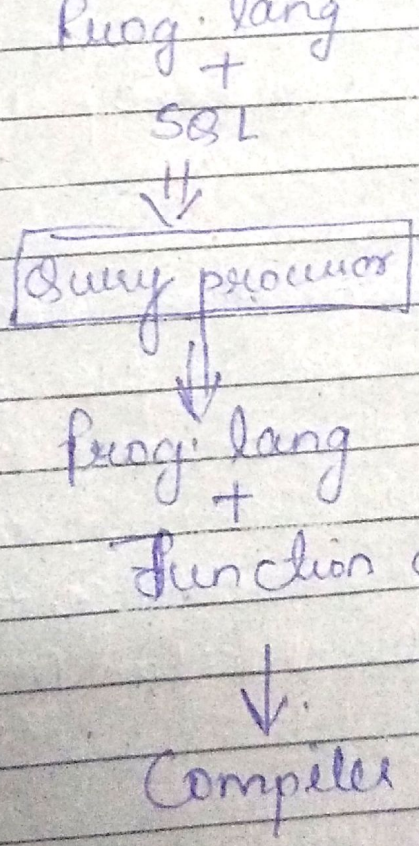
→ To hide some details, we go for views
→ Views does not get saved in the memory, rather query which is processed is saved in memory
→ Materialised view → views which get saved in the memory in some applications. It is get updated after some period of time.
→ We can create view from view.

Embedded SQL

→ We apply SQL directly to database.
→ General applications require many components.

Impedence Mismatch:-
Compiler of our lang. does not or is not able to compile or run our SQL language.

→ We introduce our prog lang (C) + SQL input into a preprocessor.



// Program

EXEC SQL <SQL Query> → to execute SQL Query in b/w any prog. language

In database, if we want to execute a loop or get multiple tupples at a time, this is done by cursor

→ Helps to pick values from database and then get it to the prog. lang
→ Cursor is bit slower.

→ Embedded SQL is an approach where, language is extended to allow SQL in program blocks.
→ There can be a specialized compiler that handle the extended language or there can be a preprocessor that converts x + SQL into plain x using call level interface. → prog. lang

x + SQL

//

x.

Impedence mismatch

Procedural languages have a strict set of data types. They support mutable variables, loops, if statement, etc. They work with one data at a time, to process n data items it uses loops. Whereas, DB works on many data items at one time. It don't support if state or loops, etc so the challenge is to make 2 diff. prog. models to co-exist together.

- EXEC SQL BEGIN DECLARE SECTION
  int presentworth;
  char StudioName[50];
  char SQLSTATE[6];
  ↳ special kind of variable that tells us about the SQL status.

* If SQSTATE returns:
  00000 ← no error.
  02000 ← No more results to display

- EXEC SQL END DECLARE SECTION
- strcpy(studioName, "xyz");
- EXEC SQL SELECT networth INTO :presnetworth
  FROM
  use to fetch value of that variable
- FROM Studios JOIN MOVIES ON Pres.c = ~~ctt#~~
  WHERE Studios name = :studioName;

Cursor:
Cursors are used to return the collection of results.

Step 1:-
→ Declare the cursor (ie, associate the cursor to the query)

Step 2:- Open the cursor. (~~Sending~~ Initializing the query to the DB)

Step 3:- To fetch the rows.

Step 4:- Close the cursor when finished.

SYNTAX:-

→ Declaring the cursor

- EXEC SQL DECLARE cursor_name CURSOR for Query.

- EXEC SQL OPEN Cursorname;

- EXEC SQL FETCH FROM Cursor-name INTO :var1, :var2;

- EXEC SQL CLOSE Cursor-name;

Eg:-

```
EXEC SQL BEGIN DECLARE SECTION
    int rec_number;
    Char title[100];
    int year;
    Char SQLSTATE[6];
EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE C-mov CURSOR FOR
  SELECT title, year FROM Movies where
    rnum = :rec-number ORDER by year
```

→ During fetching also we can modify the data

→ first declare the shared variable and then execute the SQL Query in Embedded SQL

Eg:

```
EXEC SQL OPEN C_movies;
for(;;) {
EXEC SQL FETCH FROM C_movies INTO
    :title, :year;
if (!strcmp(SQLSTATE, "02000"))
    break;
printf("%d, %s\n", year, title);
}
EXEC SQL CLOSE C_movies;
```

Modifying the rows :-

11/2/19

→ All the data are stored in the memory in the form of blocks.

→ File structure :-

→ Ordered way

(→ we will choose one attribute and will do that sorting according to that attribute

→ Insertion is difficult.

→ Deletion is easy.

→ Searching is easy (binary search).

→ Unordered way

Adv

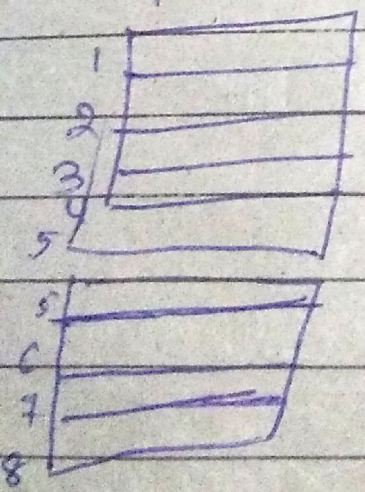
→ Insertion is efficient.

Disadv

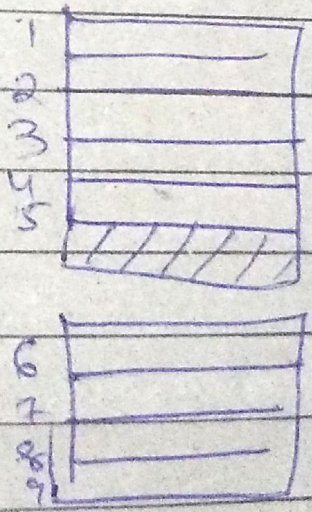
→ Linear searching, so it is costlier.

Data can be stored in 2 ways :-

① Spanned.



② Unspanned.



Memory usage is efficient.

→ Memory wastage is there.

→ Fixed data uses unspanned strategy.
Variable length data uses spanned strategy
(like content of website)

→ Generally, we convert our variable data into fixed data.

→ In ordered way, complexity of searching is $\log_2 B + 1$

→ Indexing
- sparse (Only key value have pointer).
- Dense. (Every record have a pointer record)

→ Indexed file must always be ordered and unique.

→ Indexes
- Single level indexed (Primary, Secondary, clustered)
- Multi-level indexing (B, B^+ trees)

→ Primary Index:
↳ Ordered + primary key.

Clustering
→ Ordered + non-key

Secondary
→ considering any key other than pk
→ $\log_2$ bit

Q Suppose that we have an ordered file of 30,000 records on disks whereas having a block size of 1024 bytes. The record strategy is fixed & unspanned of size 100 bytes. Suppose we have created primary index on key field on file size = 9 bytes and BP = 6 bytes.

$$BF = \left\lfloor \frac{1024}{100} \right\rfloor = 10$$

$$\text{No:- of blocks} = \left\lceil \frac{30,000}{10} \right\rceil = 3000$$

Without indexing searches $= \log_2 3000 + 1$ → Ans

$$\text{BF in index file} = \left\lfloor \frac{1024}{15} \right\rfloor = \lfloor 68.2 \rfloor = 68.$$

$$\text{No:- of blocks} = \left\lceil \frac{3000}{68} \right\rceil = 45.$$

Searches $= \log_2 45 + 1$ → Ans

Q2 Suppose there is an ordered file of 30 000 records, BS = 1024 bytes.
Rs = 100 bytes.
Secondary index on key field of file of size 9 bytes and BP = 6 bytes.

Multi-level indexing

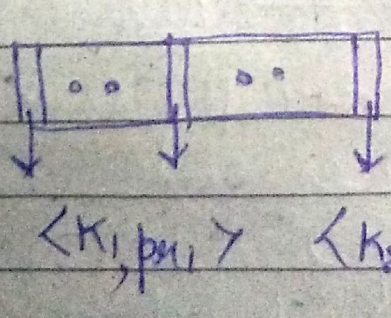
→ Flat file structure in the secondary disk.
→ This is done to reduce the access time and mapping time of the data from secondary memory.
→ B and B+ trees are dynamic in nature

B-trees (Characteristics)

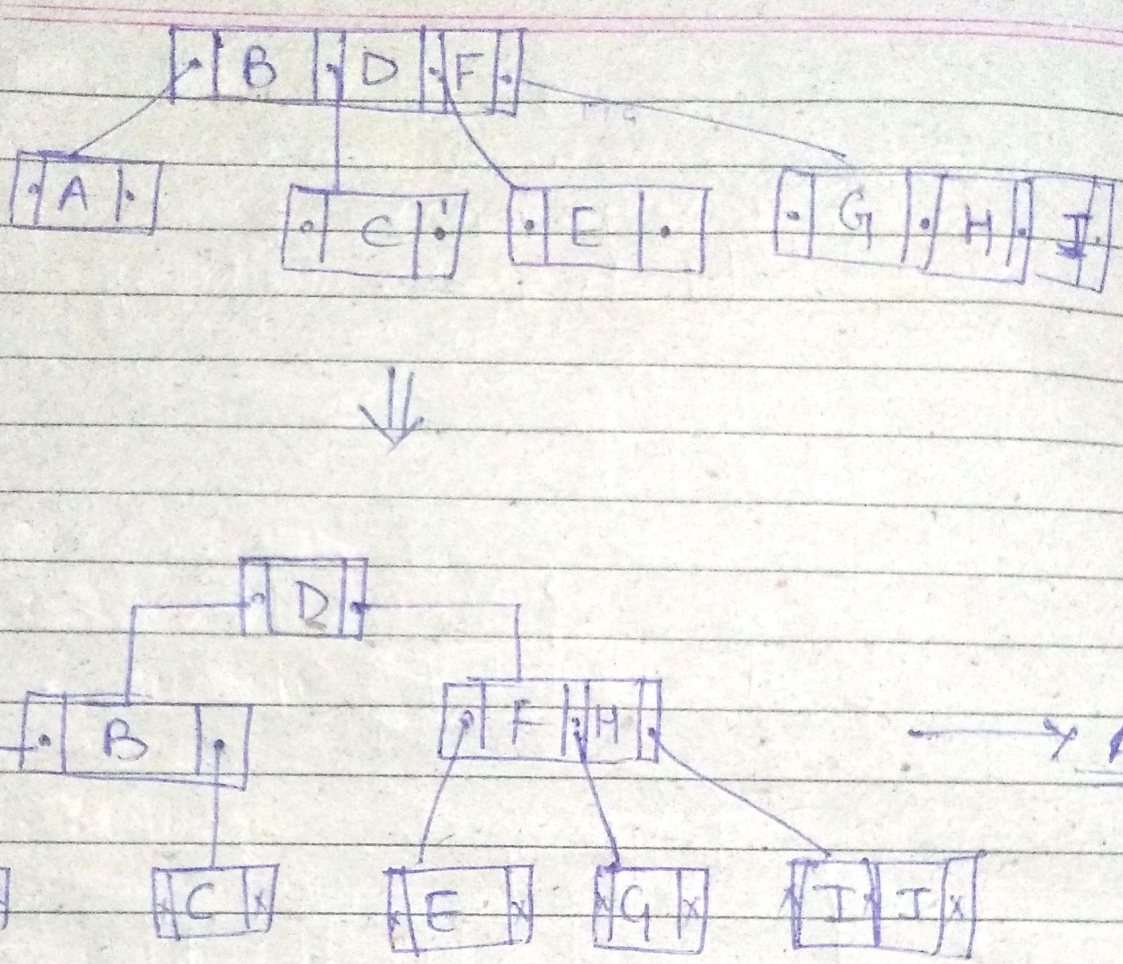
In B trees at every level we are going to have key and data pointers and that data pointer actually pointing to either block or record.

Properties

① A root of B tree can have children between 2 and p, where p is called as order of the tree.
→ Order of tree means maximum no. of children a node can have.



Q P=4 ; A B C D E F G H I J (Insert)



Overflow :- If there is no space for another key; then there is a problem of overflow.
To overcome this, we split the nodes.

Underflow :- When order is less than desired order then there is a problem of underflow.
To overcome this we do borrow and merge.

Deletion

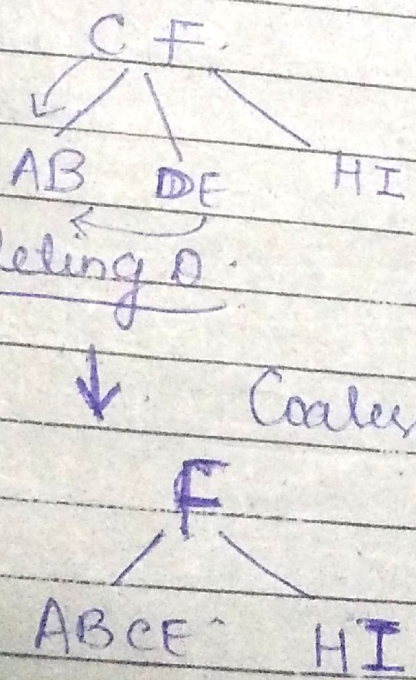
If underflow occurs then we have to adopt two methods

1. Borrow.
2. Coalesce.

Deletion:-

Removing a key from leaf node leaving l keys in leaf node. If $l >= $ min no. of nodes, then we can stop else rebalance.

Coalesce



→ The internal nodes in B+ trees, does not have record pointers unlike B trees.

B+ child node structure

$\langle K_1 P_{H1} \rangle, \langle K_2, P_{H2} \rangle \cdots \langle K_n P_{Hn} \rangle \langle P_R \rangle$

P_R → next leaf node.

—— x ——

12/2/19

ISAM Organization:-

I: Indexed

S: Sequential

A: Access

M: Method

It was introduce by IBM. In this we maintain two level of indexes. Indexing takes place (2 level) based upon primary indexing and ordered file. In this case, insertion was costly and searching was easy.

This is static and also costly, so we use dynamic hashing which use B and B+ trees

Buffer Management:-

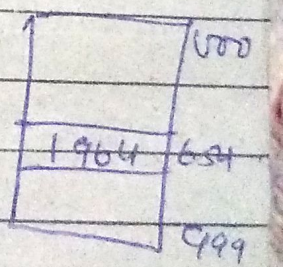
We have record that lies in hard disk when a command is to be processed, the pages are transferred from 2° memory to 1° memory. This work is done by buffer manager.

It divides 1° memory into section which is called as memory pool. In this, pages which are more frequently used are kept in buffer pool. When we require a page then that page is brought to the buffer pool.

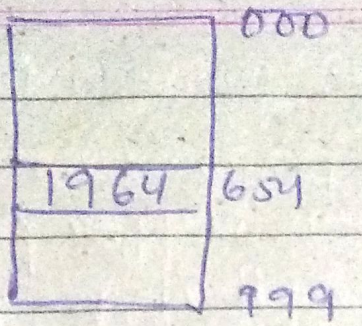
→ We maintain two flags / pointer →
count → (+, -)
dirty → (when page is not in memory).

→ When we want to ~~address~~ access small number of addresses from a large amount of address space to avoid wastage of memory, we use hashing.

Methods of hashing :-
① Division modulo
② Mid square 1964 → 96 6543 6
③ Folding.



Mid Square
Squaring the value that we want to access
e.g. 1964 = 9665496
if we assume value to be stored as
000 - 999 i.e, 1000 values
i.e, 3 digit number



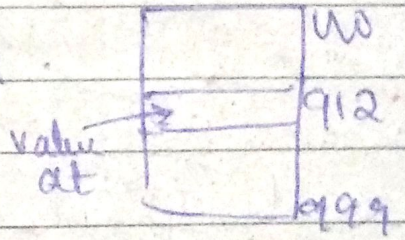
Folding

- Boundary folding
- Fold shifting

Eg 123 456 789

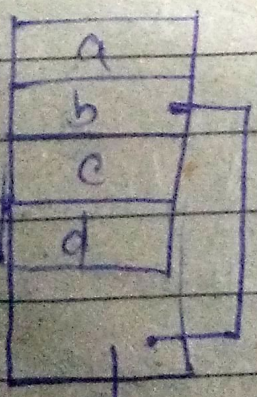
If we want to store values in 3 digit number address

store at (123 + 789) = 912.



Collision Removal Method

① Open addressing
- Linear Probing
- Quadratic Probing



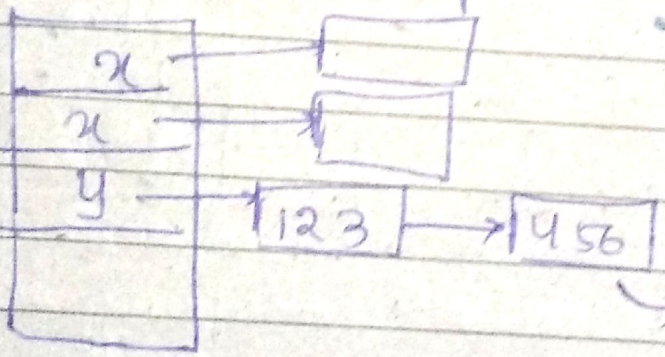
~~Insert~~ here ← Linear Probing

* Here is a limit on the no of attempts we can make to find free space.

Hashing — Internal (on Buffer)
\ External (on 2° memory)

② Double Hashing
→ Apply hashing on the collision.
(Another Best method)

③ Chaining
In this linked list is created when more than one value occurs at a particular address.



→ It can grow to any level.
(Here it is 2)

load factor = no of entries in one place (n) / total memory space (m)

It tells whether the entries are evenly distributed or not. Used to check the cost of collision, if collisions are more then we

Should change our method for hashing.

Example -

10, 25, 48, 20, 55, 69, 41, 99, 108

$h(key) = key \bmod m$

$m = (0-9)$ so total value = 10

$LP(key, i) = (key + i) \bmod m$

$LP(10,0) = 10 \bmod 10 = 0$

$LP(25,0) = 25 \bmod 10 = 5$

$LP(48,0) = 48 \bmod 10 = 8$

$LP(20,0) = 20 \bmod 10 = 0$

Already filled ←

$LP(20,1) = 21 \bmod 10 = 1$

$LP(55,1) = 56 \bmod 10 = 6$

$LP(69,0) = 69 \bmod 10 = 9$

10	0
20	1
41	2
99	3
108	4
25	5
55	6
	7
48	8
69	9

$LP(41,0) = 1$ (x) $LP(41,1) = 2$

$LP(99,0) = 9$ (x), 0 (x), 1 (x), 2 (x), 3 (✓)

$LP(108,0) = 8$ (x)

9 (x) 0 (x) 1 (x) 2 (x) 3 (x) 4 (✓)

Q. Quadratic hashing

$h(k,i) = (h(k) + c_1 i + c_2 i^2) \bmod M$

$m = 10, c_1 = 6, c_2 = 3$

$h(10,0) = 10 \bmod m = 0$

$h(25,0) = 5$

$h(20,0) = 0$ (already filled)

$h(20,1) = (20 + 6 + 3) \bmod m$

$= 29 \bmod m$

$= 9$

$h(55,0) = 5$ (x)

$h(55,1) = (55 + 6 + 3) \bmod 10$

$= 64 \bmod 10 = 4$

10	0
41	1
	2
69	3
64	4
25	5
99	6
108	7
48	8
20	9

$h(69,0) = 0 \bmod 10 = 0$

$h(69,1) = (69 + 6 + 3) \bmod 10 = 78 \bmod 10 = 8$ (x)

$h(69,2) = (69 + 12 + 12) \bmod 10$

$= 93 \bmod 10 = 3$

$h(41,0) = 1$

$h(99,0) = 0 \bmod 10 = 0$; $h(99,1) = (99+9) \bmod 10 = 8$ (x)

$h(99,2) = (99 + 24) \bmod 10 = (123) \bmod 10 = 3$ (x)

$h(99,3) = (99 + 18 + 27) = 144 \bmod 10 = 4$ (x)

Q Double Hashing

25, 77, 54, 95, 16, 5, 22, 99, 61

m = 10

$h_1(K,i) = K \bmod m$

$h_2(K) = (1 + (key \bmod (m-2)))$

$DH(Key,i) = (h_1(Key) + i(h_2(key))) \bmod m$

95 + 1 x 8

(5 + 8) mod m

13 ... 3

DH (95, 1) =

(5 + 1 x $h_2$key) mod 10

↓

1 + 95 mod 8

= 8

= (5 + 8) mod 10 = 3

$h_1(5,0) = 5$

$h_2(5) = 6$

DH (5, 1) = (5 + 6) mod 10 = 1

	0
5	1
22	2
95	3
54	4
25	5
16	6
77	7
	8
99	9

$h_1(61,0) = 1$

$h_2(61) = 6$

DH(61, 1) = 7 mod 10 = 7 (X)

DH(61, 2) = (1 + 12) mod 10 = 3 (X)

DH(61, 3) = (1 + 18) mod 10 = 9 (X)

DH(61, 4) = (1 + 24) mod 10 = 5 (X)

DH(61, 5) = (1 + 30) mod 10 = 1 (X)

DH(61, 6) = (1 + 36) mod 10 = 7 (X)

DH(61,7) = (1+42) mod10 = 3(X)
DH(61,8) = (1+48) mod10 = 9(1)
∴ Cycle come, so this won't come at any position.

Codds Rule:-

① Information Rule:- Everything in a database must be stored in a table format.

② Guaranteed Access Rule:-
Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column name).

③ Systematic Treatment of NULL Values.
The NULL values in a database must be given a systematic and uniform treatment.
This a very important rule because a NULL can be interpreted as one of the following - data is missing, data is not known, or data is not applicable.

④ Active Online Catalog
The structured description of the entire DB must

be stored in an online catalog, known as data dictionary.

(5) Comprehensive Data sub-language Rule
A DB can only be accessed using a language having linear syntax that supports data def, data manipulation and transaction mgmt operations. If DB allows access to data without any help of this language, then it is consider-ed as a violation.

(6) View Updating Rule
All the views of a DB, which can theoretically be updated, must also be updatable by the system.

(7) High-Level Insert, Update and Delete Rule
This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.

(8) Physical Data Independence
Any change in the physical structure of the DB must not have any impact on how the data is being accessed by external applications.

(9) Logical Data Independence
The logical data in a DB must be independent of its user's view (application).

(10) Integrity Independence
A DB must be independent of the application that uses it. This rule makes a DB independent of the front-end application and its interface.

(11) Distribution Independence
The end-user must not be able to see that the data is distributed over various locations.

(2) Non-Subversion Rule.
If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.

ule.

(2) The Foundation Rule
For a system that is advertised as, or claimed to be, a RDBMS, the system must be able to manage data base entirely through its relational capabilities.

19/8/19

Q. Indices speed query processing, but it is usually a bad idea to create indices on every attribute, and every combination of attributes that is a potential search keys. Explain why?

Ans - ① Every index requires additional CPU time and disk I/O overhead during inserts and deletions.

② Indices on non-primary keys might have to be changed on updates, although an index on the primary key might not.

③ Each extra index requires additional storage space.

④ For queries which involve conditions on several search keys, efficiency might not be bad even if only some of the keys have indices on them. Therefore, DB performance is improved less by adding indices when many indices already exist.

Q. When it is preferable to use a dense index rather than a sparse index?

It is preferable to use a dense index instead of a sparse index when the file is not sorted on the indexed field (such as when the index is a secondary index) or when the index file

is small compared to the size of memory.

Q3: What is the difference b/w a clustering index and a secondary index.

Ans - Secondary index may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.
Clustering index is defined on an ordered data file. The data file is ordered on a non-key field.

Q4. Construct a B+ tree for the following set of key values 2, 3, 5, 7, 11, 17, 19, 23, 29, 31. Assume that the tree is initially empty and values are added in ascending order. Construct B+ trees for the cases where the number of pointers that will fit in one node is as follows:-

1) 4 2) 6 3) 8

Q5. Explain the distinction between closed and open hashing. Discuss the merits of each technique in database application.

Q6. What are the causes of bucket overflow in a hash file organization? What can be done to reduce the occurance of bucket overflows?

Q7. Why is the hash structure not the best choice for a search key on which range queries are likely?

Q8. Suppose a relation is stored in a B+ tree file organization. Suppose 2° indices stored record identifier that are pointer to record on disk, then

(a) What would be the effect on the 2°

indices of a node split happen in the file organization.

b) What would be the cost of updating all affected record in a secondary index.

c) How does using a search key of file organization as logical record identifier solve this prob.

d) What the extra cost due to the use of such logical record identifier?

Extended Hashing or Dynamic Hashing:

Eg: data

Kartik - 00000
Sneha - 00001
Sharan - 01010
Varun - 10011
Imran - 01111
Ravi - 01100

Buckets are used to store the data. We can store the above data by taking last two or three, etc. The total number of buckets $2^{(\text{checking bits})}$ can be used.

→ It eliminates overflow chains by splitting a bucket when it overflow
→ Range of hash function has to be extended to

accomodate additional bucket.

→ Family of hash function based upon h
$h_k(v)$ mod 2^k (use last k bits of h(v))
↑
search value

→ At any given time a unique hash is used depending upon the number of times buckets have been split.

→ Extendable hashing using i directory to accomodate family of hash functions.

Transaction processing

A – Atomicity
C – Consistency
I – Isolation
D – Durability

Q Show how the extendable hash structures of the following figure changes as a result of each of the following steps :–

1) Delete 11
2) Delete 31
3) Insert 1
4) Insert 15.

PAGE NO.
DATE / /

Syllabus :-

① Overview of database models.
(only Introduction)

② Conceptual DB design
(General ER diagram concepts)

③ Logical DB design
(Relational model (no calculus))
Converting ER to relational model.

④ Physical DB design.
(Hashing, B BT tree (insertion & deletion))

⑤ Normalization

⑥ Transactions
(Properties, Schedules, serializability, Consistency, transaction performance
↙
Eg :- (if trans. is interleaving then what will be its effect on performance)

22/02/19

→ To improve the CPU utilization, we introduce the concept of concurrency

→ Problems → lost update (write some uncomitted trans.)
→ dirty read (read some uncomitted trans)
→ unrepeatable read.

→ $\frac{(n+m)!}{n!\,m!}$ = Total schedules if a trans have m operation and other have n ".

Total serial schedule = $n!$
↳ where n is no. of trans.

Q Find total number of schedule with number of operation wrt each trans are:- 2, 3 and 4.

Total schedule = $\frac{(2+3+4)!}{2! \times 3! \times 4!}$

$= \frac{9!}{2 \times 6 \times 24} = \frac{9 \times 8 \times 7 \times 6 \times 5 \times 4!}{2 \times 6 \times 24}$

$= 9 \times 4 \times 7 \times 5$

Total non-serial schedule = Total schedule
- Total serial schedule

Here, = $9 \times 4 \times 7 \times 5 - 3!$

Q

T_1 :-	T_2 :-
read (P) ;	read (Q)
read (Q) ;	read (P)
if P = 0 then Q = Q + 1 .	if Q = 0 then P = P + 1
write (Q) .	write (P) .

What does Any non-serial interleaving of T_1 and T_2 for concurrent execution leads to ?

(1)

T_1	T_2
read (P)	
read (Q)	
	read (Q)
	read (P) .
write (Q)	
	write (P) .

(2)

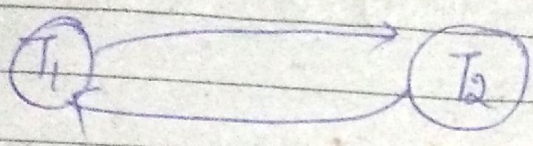
	read Q
	read P .
read P	
read Q	
	write (P)
write (Q)	

T_1	T_2
③ read P	
	read Q
read Q	
	read P.
write Q.	
	write P.

T_1	T_2
④ read (P).	
	read (Q)
read (Q)	
	read (P)
	write (P)
write (Q)	

T_1	T_2
⑤	
	read (Q)
read (P)	
	read (P)
read (Q)	
write (Q)	
	write (P)

There can be many more non-serial schedules.



This is not conflict serializable.

Q

S_1 : $R_1(A)$, $W_1(A)$, $R_2(A)$, $W_2(A)$, $R_1(B)$, $W_1(B)$, $R_2(B)$, $W_2(B)$.

S_2 : $R_1(A)$, $W_1(A)$, $R_1(B)$, $W_1(B)$, $R_2(A)$, $W_2(A)$, $R_2(B)$, $W_2(B)$.

Whether it is conflict serializable or not.

S1

T_1	T_2
R(A)	
W(A) →	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B) ↓
	W(B)

S2

T_1	T_2
R(A)	
W(A)	
R(B)	
W(B)	
	R(A)
	W(A)
	R(B)
	W(B)

It is conflict serializable.

Q T_1 :- $r_1(x)$; $r_1(z)$; $w_1(x)$; $w_1(z)$.

T_2 :- $r_2(y)$; $r_2(z)$; $w_2(z)$.

T_3 :- $r_3(y)$; $r_3(x)$; $w_3(y)$

S_1 : $r_1(x)$; $r_3(y)$; $r_3(x)$; $r_2(y)$ $r_2(z)$; $w_3(y)$; $w_2(z)$; $r_1(z)$; $w_1(x)$, $w_1(z)$.

S_2 : $r_1(x)$; $r_3(y)$; $r_2(y)$; $r_3(x)$; $r_1(z)$ $r_2(z)$; $w_3(y)$; $w_1(x)$; $w_2(z)$; $w_1(z)$

S_1

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

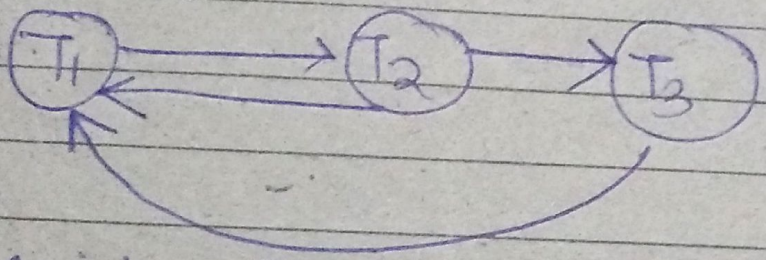
$T_1 \leftarrow T_2$ T_3

It is conflict serializable

$T_3\ T_2\ T_1$ or $T_2\ T_1\ T_3$

S2

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

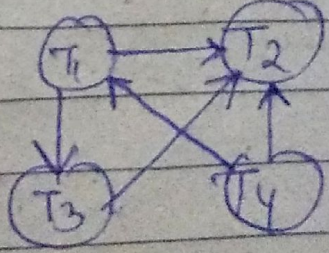


Cycle exist, it is not serializable

Q

T_1	T_2	T_3	T_4
			R(A)
	R(A)		
		R(A)	
W(B)			
		W(A)	R(B)
	W(B)		

conflict serializable



logical schedule: T3 T4 T1 T2

→ Every conflict serializable is also view serializable.

→ If a schedule is not conflict serializable it may be view serializable (in case of blind write).

— x —