UNIT-I :-

DBMS :- A s/w package / system to facilitate the creation and maintenance of a computerised database

Database Systems:

The DBMS s/w together with the data itself. Sometimes the applications are also included.

DBMS Functionality:

To Defines a particular database interms of its data types, structures and constraints.

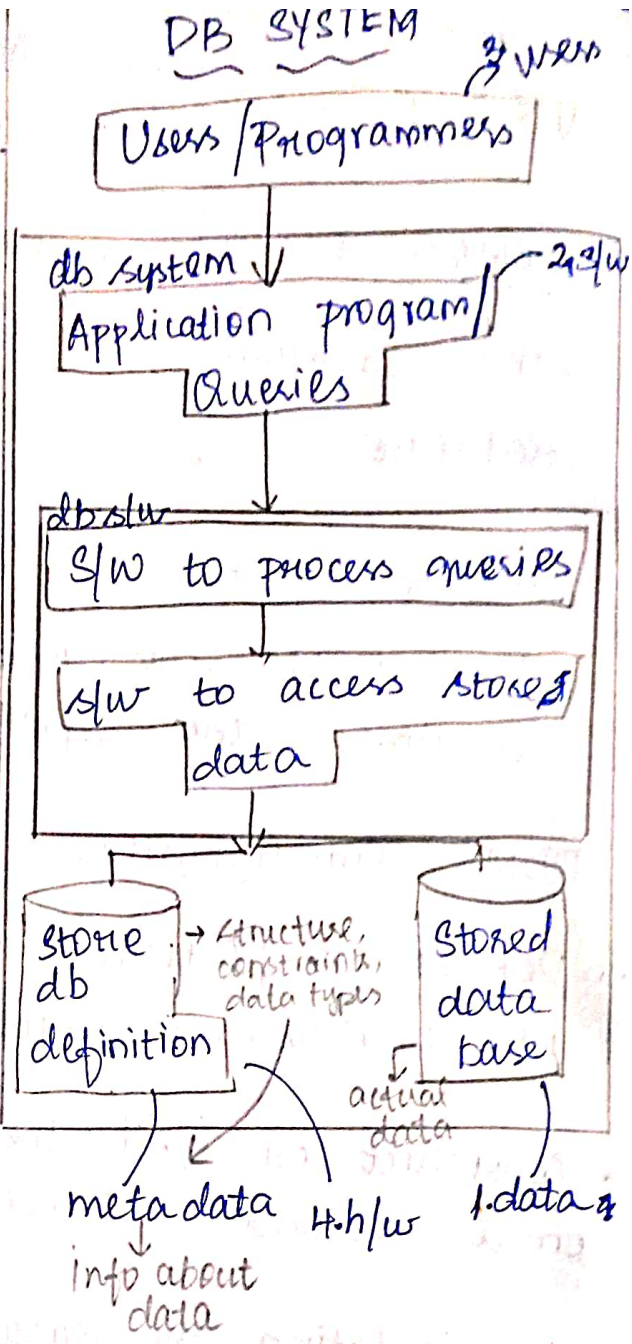
- Construct or load the initials database contents on a secondary storage medium.
- Manipulating the database
  - → Retrieval - querying, generating reports
  - → Modification - Insertion, deletion and update to its contents
  - → Accessing the db through applications.
- Processing and sharing: by a set of ~~construct~~ concurrent (simultaneous) users and application programs yet keeping all ~~the~~ data valid and consistent.

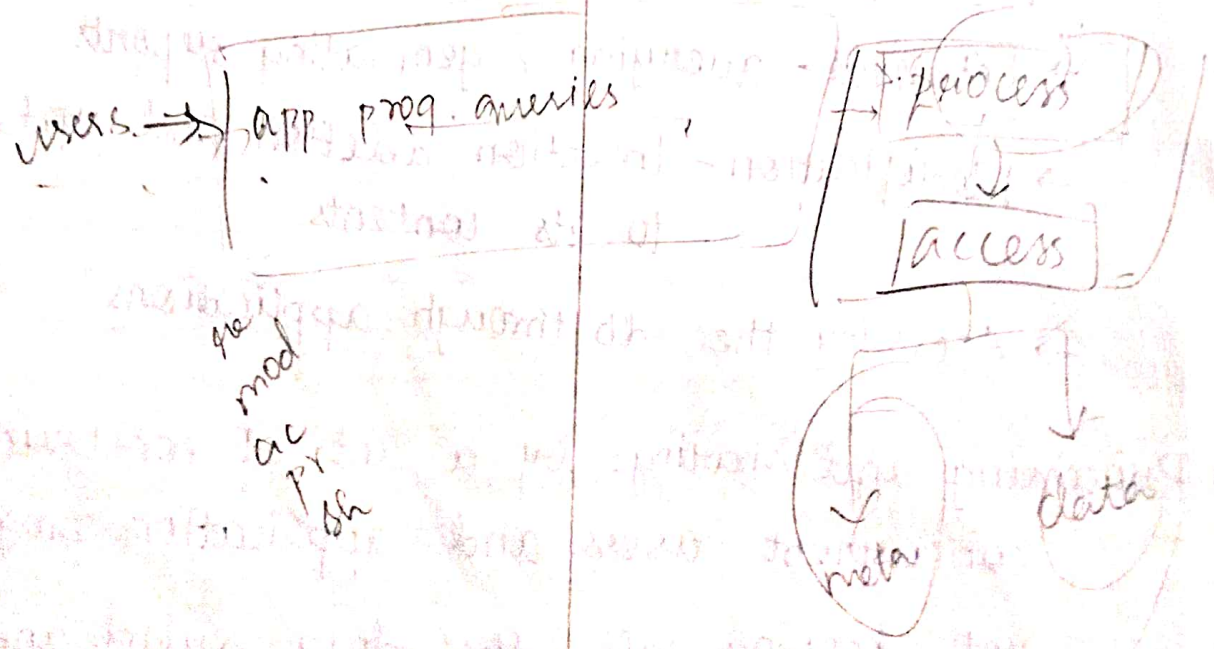
Database Implementation:

- → Data Types
- → Structures
- → Constraint
- → Sharing the data
- → Querying

Database Applications:

- → Numeric, Textual (notepad)
- → Multimedia dbs (Youtube audio/video)
- → GIS (Geographical info system)
- → Data ware houses
- → Real time and active db





eff:

Data Models: How we store data.

↳ visual representations of an enterprise's data elements and the connections b/w them.

↳ FMS: File management system
→ can store any no. of data
→ stored in sequential order
draw back → retrieval also begins from the begin

ROFL

↳ HM: Heirarchical model
→ based on one to many relations → Tree-like structure
→ less time consumption than FMS
→ Cant jump from one data to another (has to follow branches of a tree)

↳ NM: Network model
→ faster than both because we use pointers to process data
→ Too many pointers might increase the complexity in searching data. → so cant use for large datasets

1970 : EF. Kod

↳ RM: Relational model
→ Table form (data storage)
↳ rows and columns.

Any database satisfying max of 12 rules of KOD → perfect RDBMS

↳ Oracle satisfies 11/12 rules

ORACLE LAB:

↳ Oak Ridge Arithmetic Calculating Logical Electronic device

↳ got popular in 1977

DDL: Create, Alter, Drop, Truncate, Rename

(data definition Lang)

Alter → add, modify

Drop → delete table

Truncate → delete data alone not structure

SYNTAX:

- Create table Stu (Sno number, Sname varchar(20), S_sex char(1));
- Alter table Stu drop column Sno;
- Alter table Stu add (Saddr varchar(30));
- Alter table Stu modify (Saddr Varchar(40));
- Rename Stu to Student;
- Truncate table Student; → removes all rows but the structure, col, const, indexes remain same
- Drop table Student;

Date

DML : Insert, Update, Select, Delete, Merge

- Insert into Student values (1, 'Ram', 'M', 'Hosur');

- For multiple insertions:
  Insert into Student Values (&Sno, '&Sname', '&S_sex', '&Saddr');

```
Insert into all
Insert into Student (        );
Insert into Student (        );
select * from dual;
```

dual → dummy table

- Update Student Set S_name = "Krishna" where S_name = "Ram";

- Delete from Student; → delete all rows
- Delete from Student where Sname = "Krishna";

DCL
Data control commands

SOAH :)

Delete contents of a column

boys only want love if its fortune

(1.)

Main characteristics of the db approach:

→ self describing nature of a db system
- A DBMS catalog (how to view) stores the description of a particular db (eg. types, structures, constraints)
- the description is called meta-data

→ Insulation between programs and data
- Called program-data independence
- allows changing data structures and storage organization without having to change the DBMS access program.

→ Data abstraction
- A data model is used to hide storage details and present the users with a conceptual view of the db.

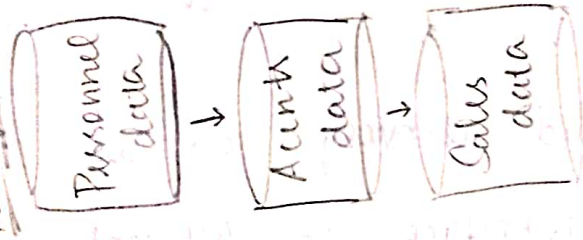
→ Support of multiple views of the data
- Each user may see a different view of the db, which describes only the data of interest to that user.

→ Sharing of data and multiuser transaction processing
- allows a set of concurrent users to retrieve from and to update the db.

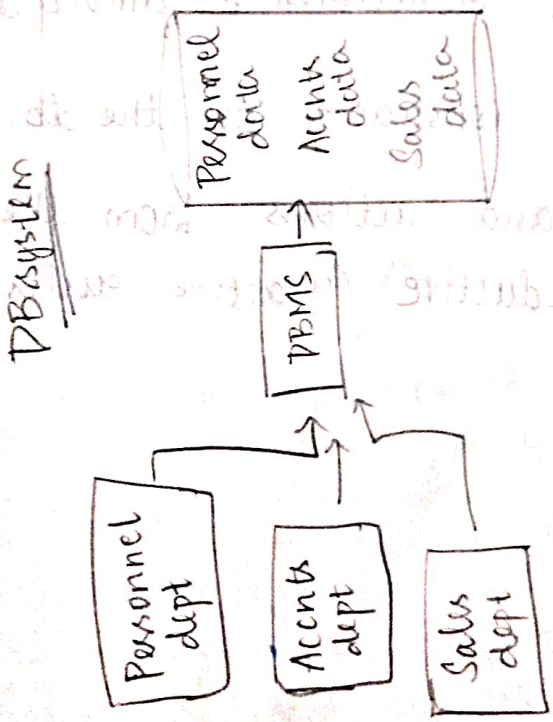
- Concurrency control with the DBMS Guarantees that each transaction is correctly executed or aborted
- Recovery subsystem ensures each completed transaction has its effect permanently recorded in the db.

select * from tab;

File System



DBsystem



Advantages of DB System

1. efficient memory
2. Easy access
3. EZ sharing of data
4. Analyze
5. Managable
6. Security
7. Avoids duplicates of dat
8. Easy recovery of data

Advantages of using of the db approach:

- controlling redendancy in data storage and in development and maintanance effort
- sharing of data among multiple views
- Restricting un authorized access to data → DCL Grant Revoke
- providing persistent storage for program objects
- Providing storage structures for efficient query processing.
- Providing backup and recovery services
- providing multiple interface to different classes of users.

+ ~~Refreshing~~

- Representing complex relationship among data
- Enforcing integrity constraints on the db.
- Drawing inference and actions from the stored data using deductive & active rules

Data Abstraction :-

Level of abstractions :-

⇒ **Physical level / Internal level :**

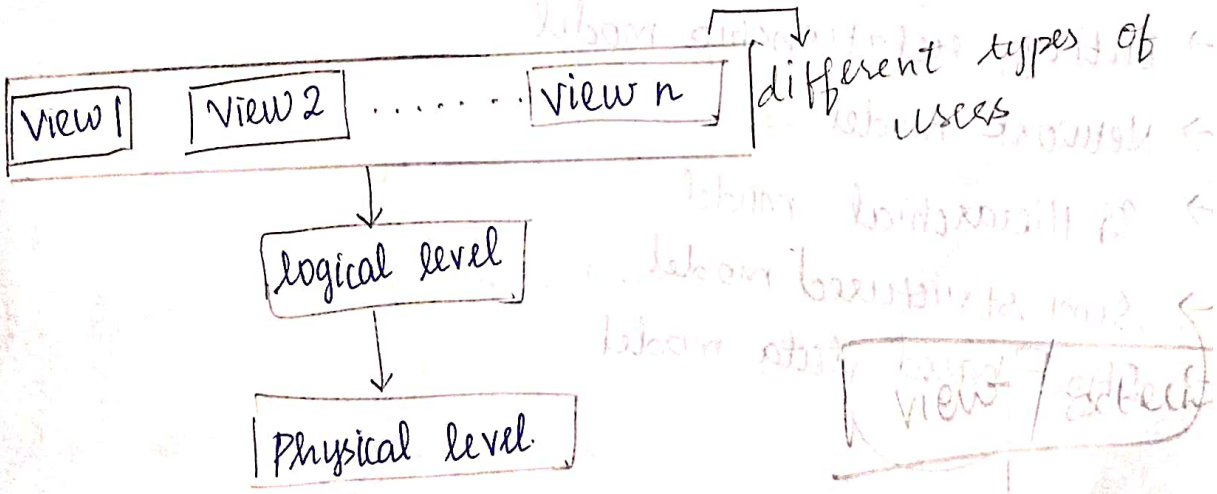
- describes how the data are actually stored

⇒ **Logical level / Conceptual level :**

- Describes what data are stored in the db and what relationship exists among the data

⇒ **View level / External level : ??**

- Highest level of abstraction describes only part of the entire db.



Level of data abstraction.

sft.

Instance: ·X·
the collection of info stored in the db at a particular moment is called an instance. ↳ the value of variables @ a particular moment.

Schema: ·X·
The overall design of the db is called db schema
↳ blueprint of a database which describes how the data may relate to other tables / data models

Application program: to exhibit physical data independence if they do not depend on the physical schema
↳ is a comprehensive, self-contained program that performs a particular function directly for the user.

Data models: → describes the structure of the db

→ relational model
→ entity - relationship model
→ Network model → star, bus, ring
→ Eg Hierarchial model
→ Semi structured model
→ Object based data model

comprehensive, self contained program that performs a particular function

Sql:

LAB:

1. Select * from tab; → contains all tables available

Constraints:

1. Key constraints
   - Primary key → not null always (col) → to maintain uniqueness
   - Foreign key → shud be PK of another table
2. Default constraints
3. Not Null constraints.
4. Check constraints
5. Unique Constraints → allows null values

→ these come under DDL

2. Alter table <table_name> modify <col_name> notnull;

3. <col_name> varchar(10) notnull; → while defining

4. Alter table ~~Product~~ <table_name> add constraint (→ keywords) P_info_PK primary key (model_no);
   - ⇓ only 3/or key constraints or check
   - P_info_PK → user gn name of constraint.
   - (model_no) → col name

5. Alter table PC add constraint PC_pk primary key (model_no);

6. Alter table pc add constraint PC-FK foreign key (model_no) references product_info (model_no);

7. Alter table laptop add constraint Lap-Fk foreign key (model_no) references Product_info (model_no on delete cascade;

if this col is deleted from parent table it will be deleted from this table also

8. Alter table printer add printcode varchar(10);

9. Alter table laptop modify HDD number(4);

10. select * from user_constraints;

11. Select table_name, constraint_type, Constraint_name from user_constraint where table_name = 'PC';

eg:
→ referential constraint (foreign key)

⇒ PC R PC-FK

PC P PC-PK

PC C PC-CK

one table can have only one primary key

12. Alter table PC ~~add~~ modify speed default 2;

13. Alter PC add ~~column~~ constraint PC-CK check price > 0;

alter table < > drop constra < >;

14. Alter table product-info add constraint P_info-ck check (type in ('LP', 'PC', 'PR'));

15. Create table printerinfo as select * from printe
model-no, price.

modify speed not null;
modify speed unique;
modify speed default 2;

16. Create table printerinfo as select * from printer where 1>2;
only structure will be copied

17. Delete from printerinfo;
only data deleted

18. drop table printerinfo; → including structure

Sol:

19. Update PC set price = price - 10;

20. Update PC set price = price + 10 where ~~model = 1122~~ model_no = 'PC111';

21. Delete from PC where model_no = 'PC112';

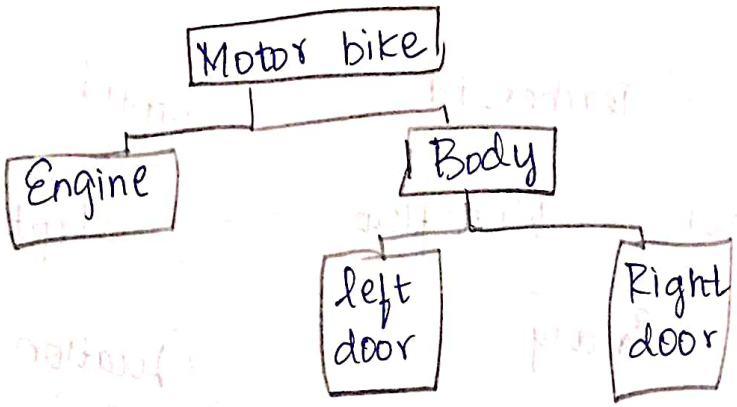
22. Select * from product_info where model_no = (Select model_no from PC where price = 500);

DATA MODELS :-

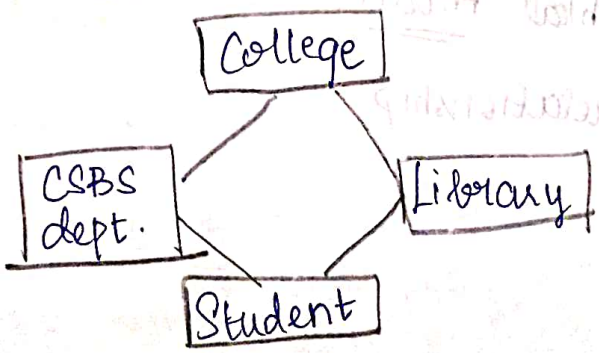
Relational model: → tabulated form

Empid	Ename	Salary	Did
101	Ram	45000	10
102	Renu	42000	11

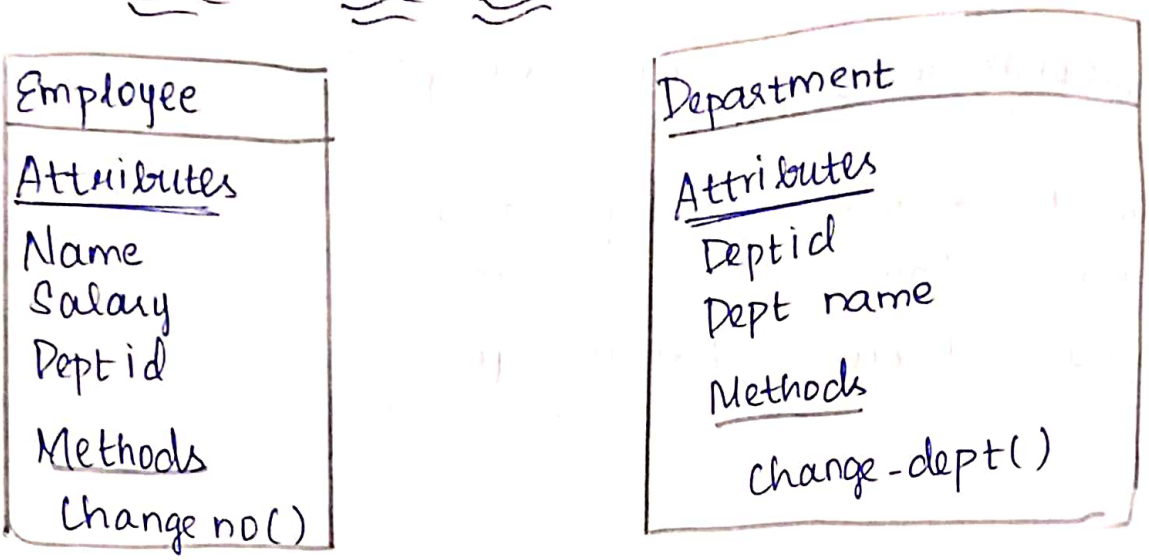
Hierarchial mode: → tree structured



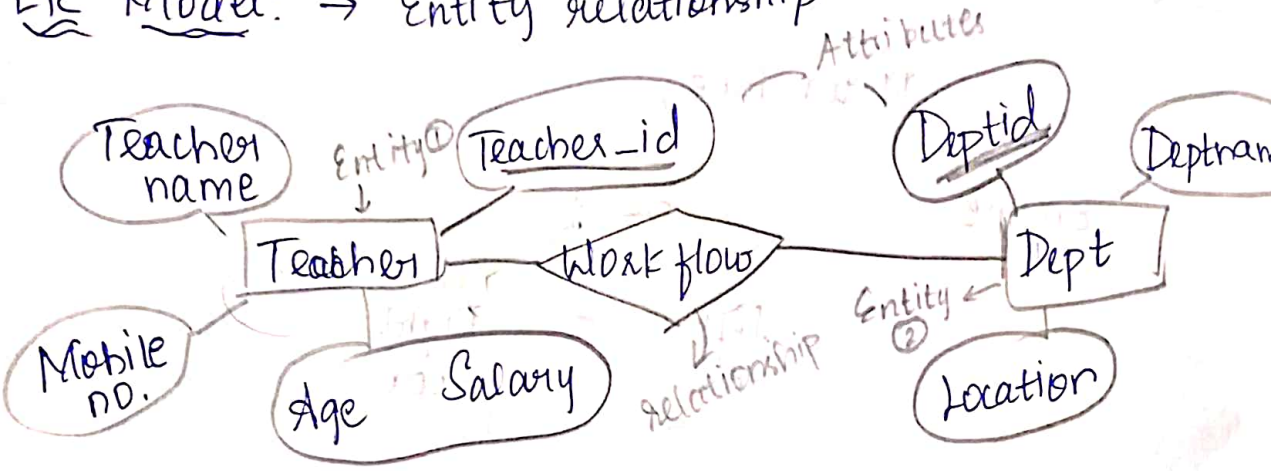
Network model: → diff types of topology



Object Oriented data model:



ER Model: → Entity relationship



Features Of Hierarchical model:

* one-to-many relationship
* parent-child " "

Advantages:

→ very simple & fast to traverse through a tree like structure

→ Any change in the parent node is automatically reflected in the child node

Disadvantage:

→ Complex relationships are not supported

→ parent node is deleted, the child also gets deleted.

Features Of Network model:

→ ability to merge more relationship

→ many paths

→ Circular linked lists

Advantages:

→ the data can be accessed faster as compared to hierarchical model

→ Parent-child relationship

Disadvantages:

→ Change like updation, deletion and insertio very complex

Features Of ER:

Entities: real-world things

Attributes: real-world property

Relationship: Tells how 2 attributes are related

DBMS vs RDMS

Advantages:

→ Simple
→ Effective communication tool
→ Easy conversion to any model

Disadvantages:

→ Hidden info → no dml available for ER
→ No industry standard for notation

Features of Relational model:

→ Tuples
→ attribute or field

Advantages:

→ simple
→ scalable
→ structural Independence

Disadvantages:

→ H/w overhead??
✓ requisites

① Presentation layer:
→ client layer
→ purpose is to communicate with the application layer

② Application layer:
→ business logic layer
→ middle layer b/w 1 & 3 layers
→ transmit partially processed data

③ Database layer:
→ stored info
→ insert, update & delete

Features of ER model: → it's a diagram → describes interrelated things of interest in a specific domain of knowledge

→ is an abstract of our DB

→ high level or conceptual model

Relation:

→ format and relations in a way a db could understand

→ Implementation of our model

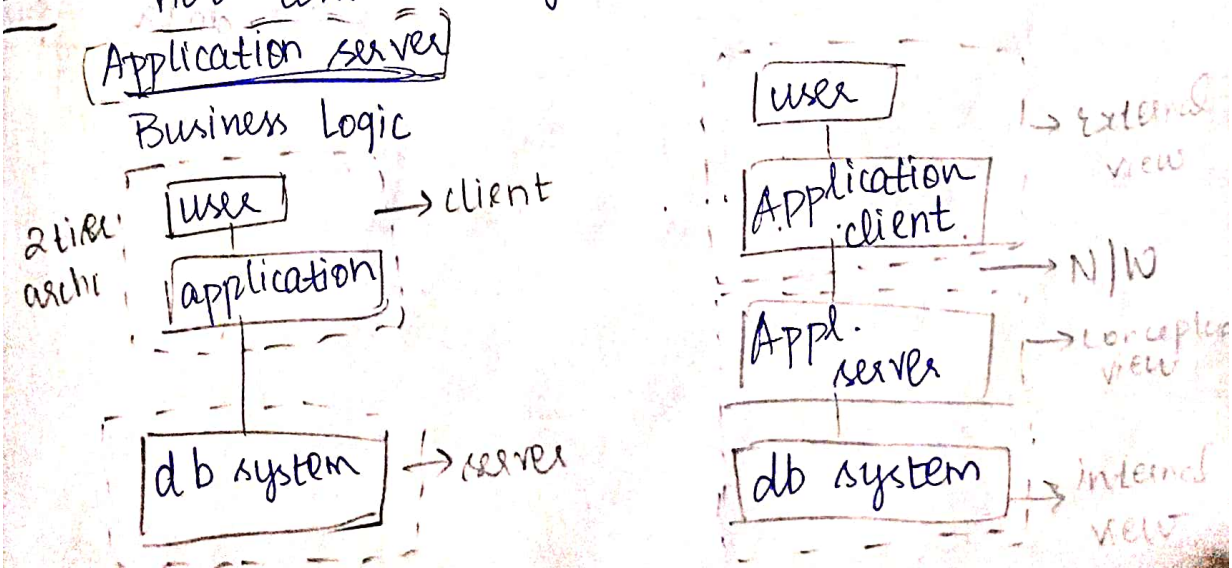
Two-Tier architecture: → ex: client/server

→ the application resides at the client machine m/c, where it invokes db system functionality at the server m/c through query language.

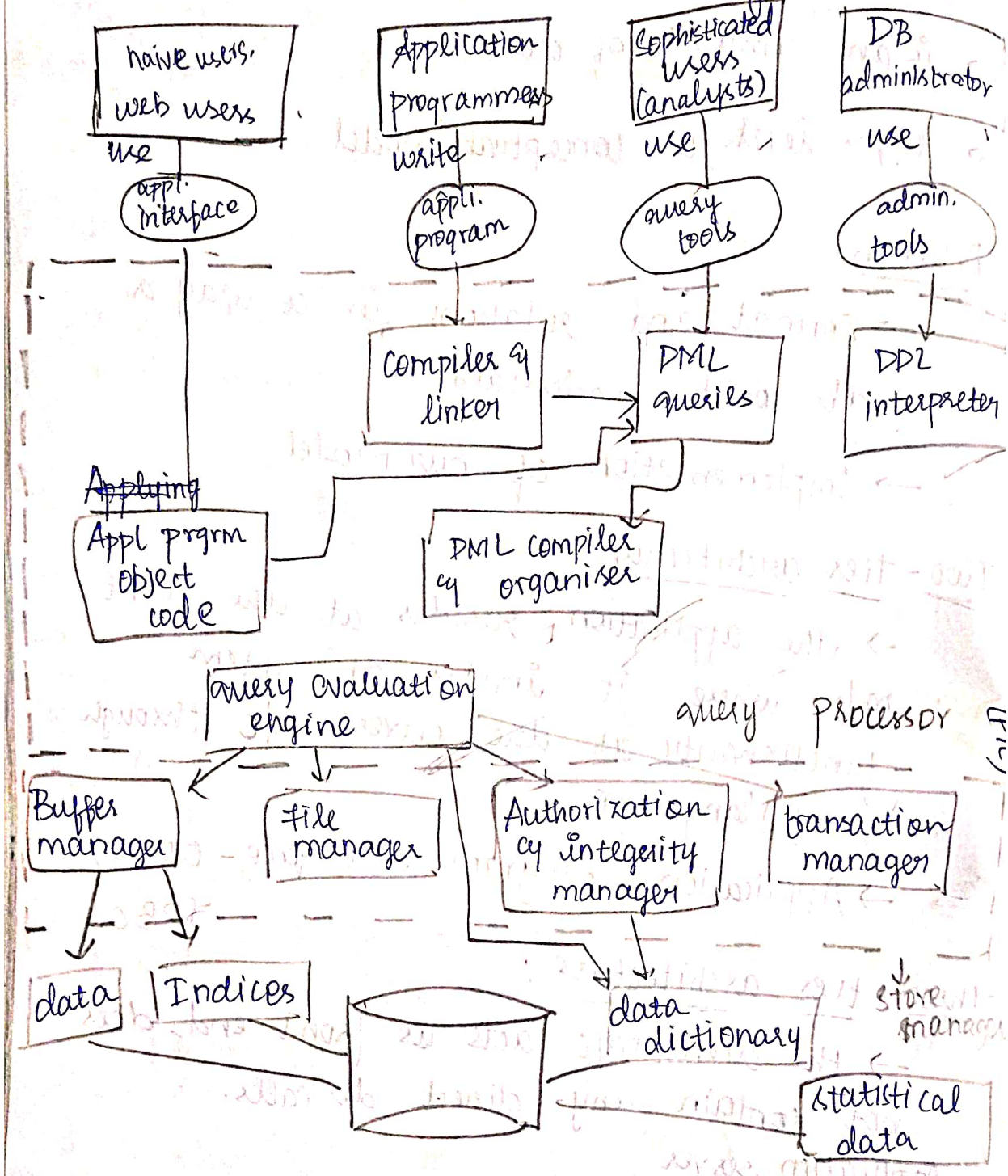
→ Application program interface - ODBC, JDBC

Three tier architecture:

→ the client m/c acts as front end, does not contain any direct db calls.



Database System architecture: (X)



Ett.

Query Processor:

⇒ DDL interpreter:

- which interprets DDL statements and record the definitions in the data dictionary.

⇒ DML Compiler:

- which translates DML statements in a query language into an evaluation plan consisting of low level instructions in the query evaluation engine understands.

⇒ Query evaluation Engine:

- which executes low-level instructions generated by the DML compiler.

⇒ Data files:

- which store the db itself

⇒ Data dictionary:

- which stores the metadata about the structure of the db, in particular the schema of the db.

⇒ Storage manager:

→ Authorization and integrity manager:

- which tests for the satisfaction of integrity constraints and check that authority of user of access data

→ Transaction manager:

- the db remains in a consistent (correct) state despite system failures and the concurrent transaction executions proceed w/o conflicting

→ File manager:

- manages the allocation of space on disk storage and data structures used to represent info stored on disk

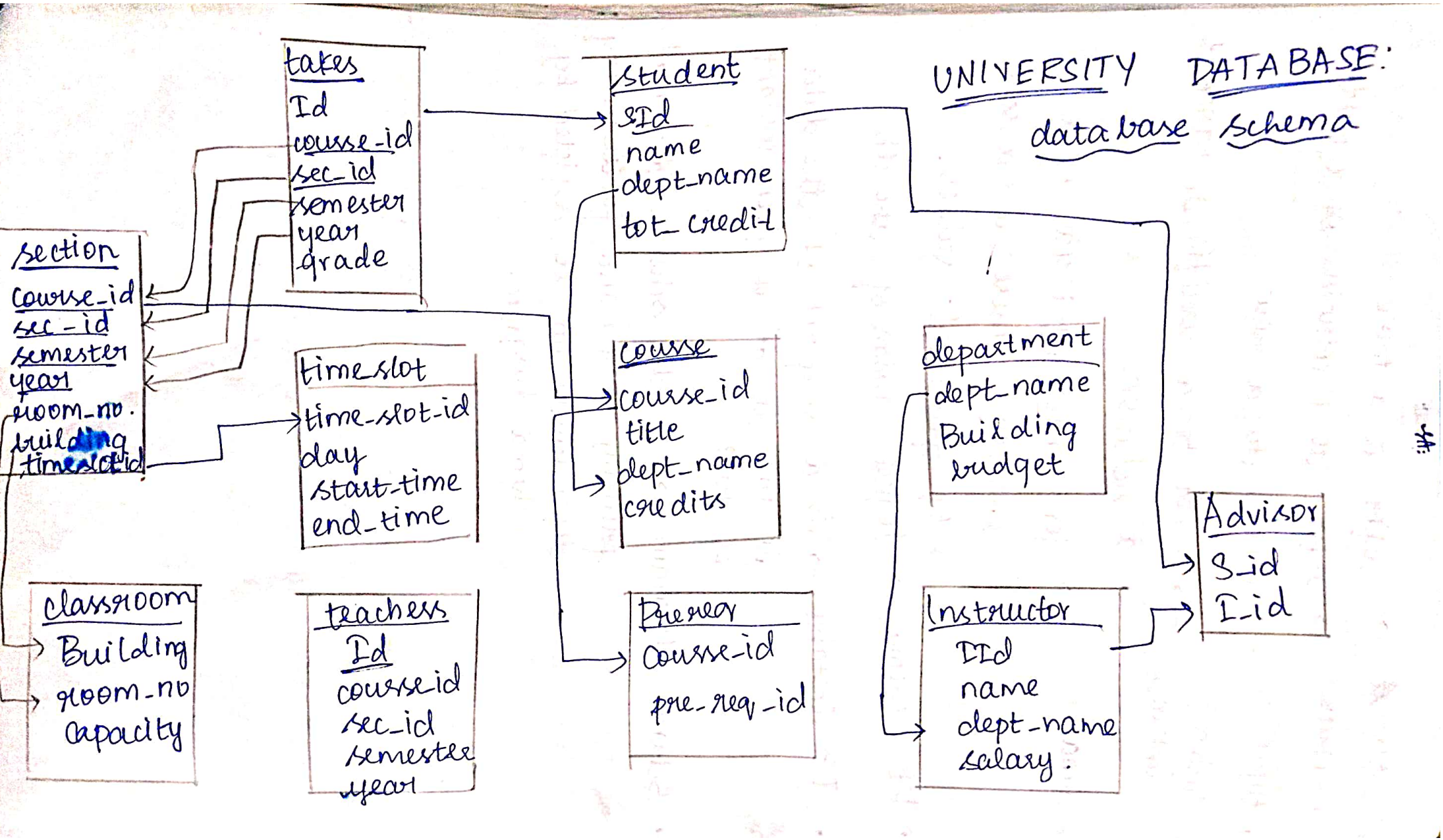
⇒ Buffer manager:

- responsible for fetching the data from disk storage ~~units~~ into main memory and deciding what data to access in main memory

- Relation → refer to a table
- attribute → refer to a column of a table
- tuple → refer to a row
- relation instance → refer to a specific instance of a relation
- domain → for each attribute of a relation there is a set of permitted values.
- DB schema → logical design of the db
- DB instance → is a snapshot of the data in the db at a given instant of time.
- Schema diagram → A db schema along w/ primary key and foreign key dependencies can be depicted by schema diagram.

UNIVERSITY DATABASE:

database schema



KEY CONSTRAINTS:

→ Primary key → unique / not-null
→ Foreign key → PK of another table
→ Candidate key → can be PK (minimal set of super key)
→ Super key → combination of columns in a table to maintain uniqueness
→ Unique key → unique

ER model:

- Entities:
  - Ex: Employee, Department, Project
- Attributes:
  - Employee: Name, SSN, DOB, Address
- Types of Attributes:

(1)

→ Simple:
- Single atomic value for the attribute

→ Composite: ()
- Several components

Ex: • Address (Apt #, House #, St., City)
• Name (first name, last name, middle name)

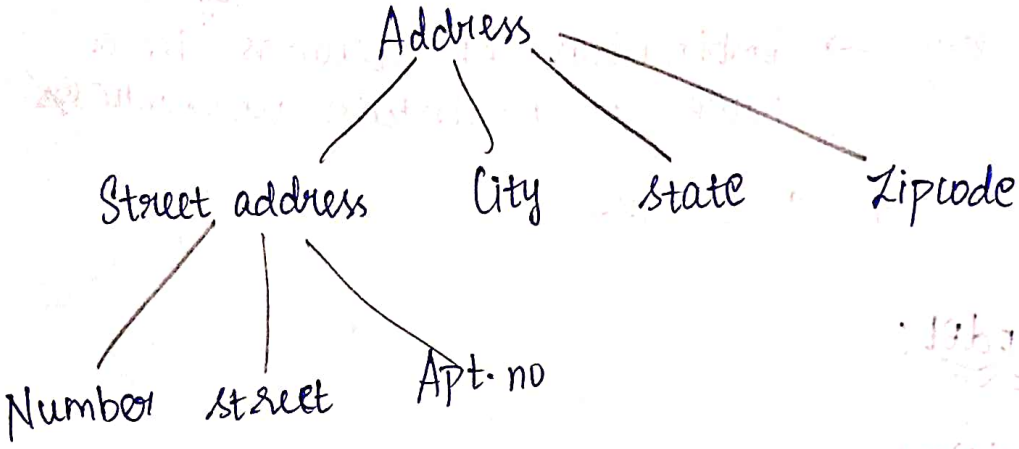
→ Multivalued: {}
- Multiple values for that attribute

Ex: • color of a car
• previous degrees of a student.

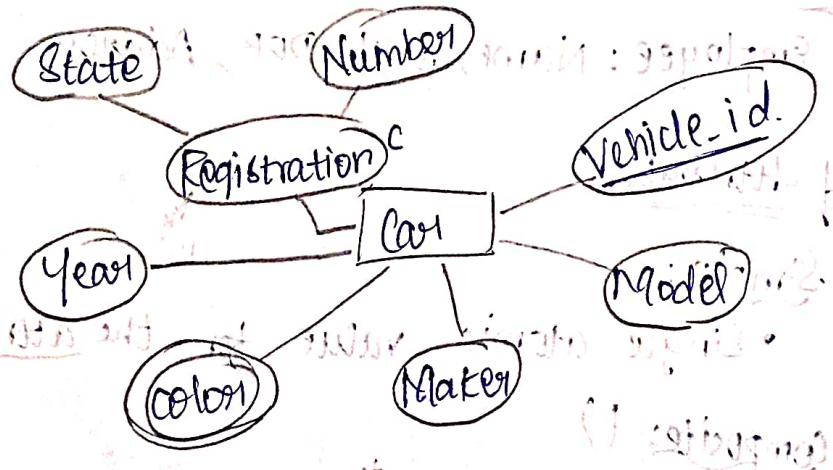
⇒ Composite ~~valued~~ multivalued:

{previous degrees(college, year, degree, field)}

⇒ Composite:



Vehicle tag:



Entity set:

Registration (Number, state), vehicle_id, make, model, year, {color}.

CAR 1:

(ABC123, Texas), TK629, FORD, 4 door, 2004, {red, black}

CAR 2:

(VSY720, Texas), TD729, NISSAN, 4 door, 2002, {blue, red})

Relationship:

- when an attribute of one entity type refers to another entity type.
- Relationship type R among 'n' entity types $E_1, E_2 \ldots E_n$
- Defines a set of associations among entities from these entity types

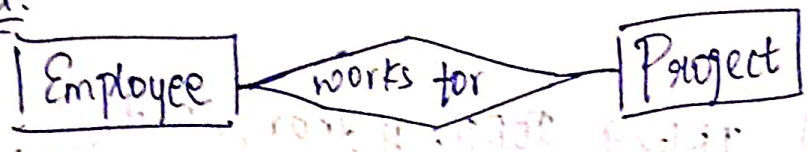
Relationship instances r:

Each r_i associates n individual entities $(e_1, e_2 \ldots e_n)$ Each entity e_j in r_i is a member of entity set E_j.

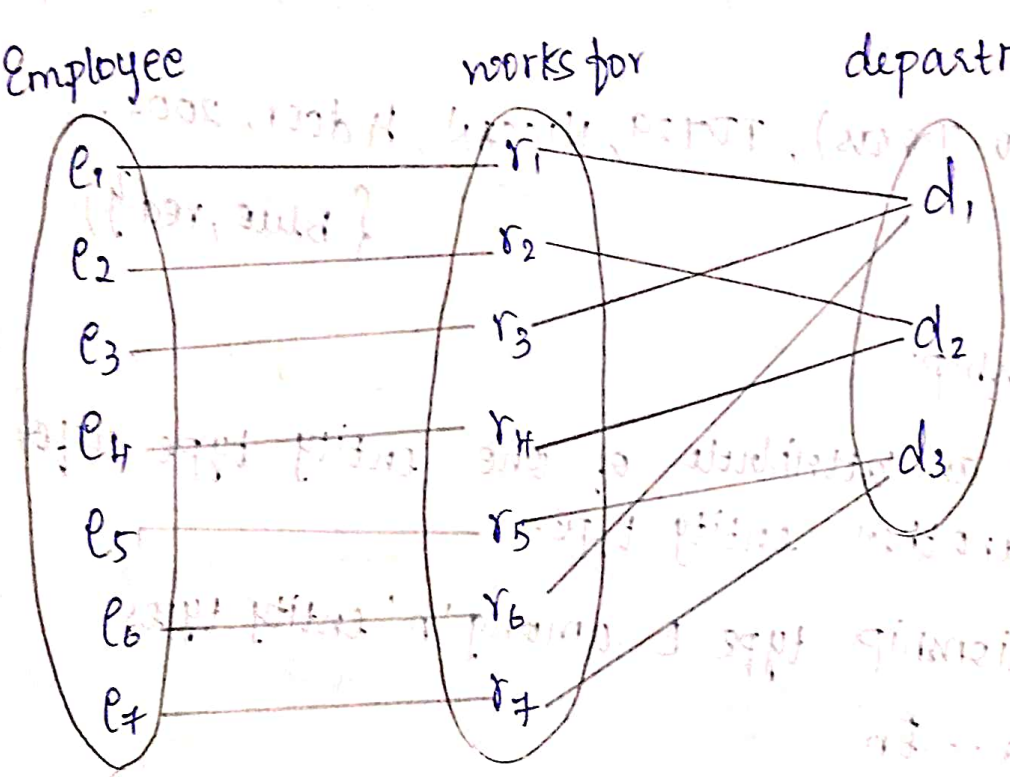
Degree of relationship type:

No. of participating entity types binary, ternary

Ex:

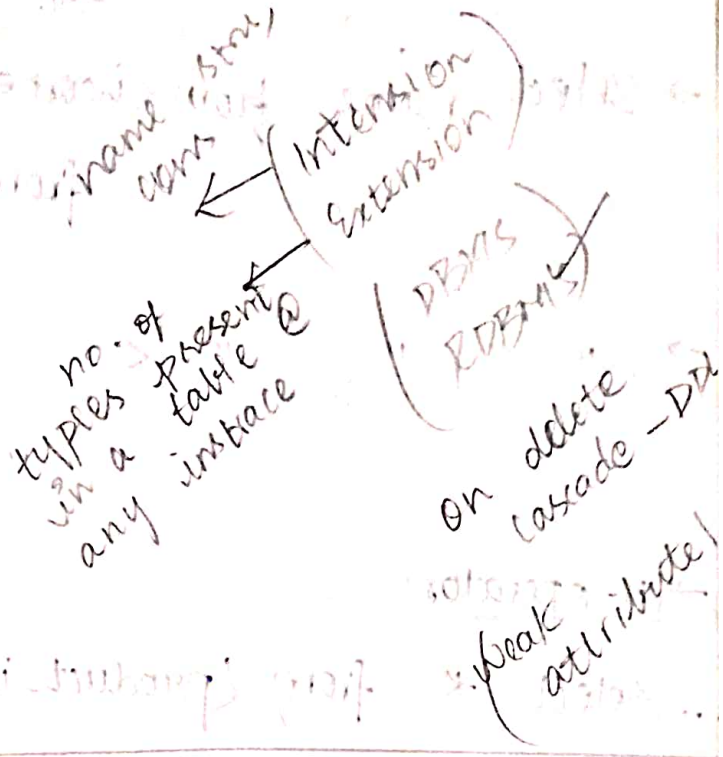


Relationship instances:



Some instances of the work for relationship b/w employee and dept.

SQL
EPdia
Archi



LAB:

Arithmetic, Logical, set operations:

- Arithmetic: + - / *
- Logical: OR, AND
- set operations: Union all, union, Intersection, minus

A = {1, 2, 3, 4}

B = {3, 4, 5, 6}

A union B = {1, 2, 3, 4, 5, 6}

A union all B = {1, 2, 3, 4, 3, 4, 5, 6}

A Intersection B = {3, 4}

A minus B = {1, 2}

B " A = {5, 6}

eg:

⇒ Select name from <worker> union select name from <worker skills>;

⇒ " " " < > (Intersect / minus) union all " " " < > ;

⇒ In operator:

→ select * from <product_info> where type in ('PC', 'PR');

→ " " " " where type = 'PC';

→ " " " " " type not in ('PC', 'PR');

→ update <pc> set price = price+1000 where model_no = 'PC112';

→ select 2 * RAM from <PC>;

→ select 100 price - 1000 from <laptop>;

→ select * from <pc> where price > 5000 and price < 10000; (or)

→ select * from <pc> where price between 5000 and 10000;

→ select * from <laptop> where model_no = 'LP112'
or model_no = 'LP113';

⇒ Aggregate functions:
Sum, average, max, min, count

→ select max(price) from <pc>;
→ select min(salary) from <emp>;
→ select count(*) from <product_info>;

⇒ Group By function:
→ select dept_id, count(*) from <student>
group by dept_id;

→ select dept_id, sum(salary) from <employee>
Group by deptid;
→ select RAM, count(*) from <pc> group by RAM;
⇒ GroupBy - having :- condition for groupby:
→ select RAM, count(*) from <pc> group by RAM
having count(*) > 1;

⇒ Order by:

→ select * from <pc> order by price;

→ " " " " " desc;

Translate ('3*[2+1]/{8-4}', '[]{}', '()()')

⇓

gives

3*(2+1)/(8-4)

Primary key

- unique
- Not null

Superkey

- ~~It has all the capabilities to be a PK~~
- A combination of attributes to maintain uniqueness

comp. PK

eft:

Indexing:

→ is a data structure technique which allows you to quickly retrieve records from a db file.

→ An index is a small table having only two columns:

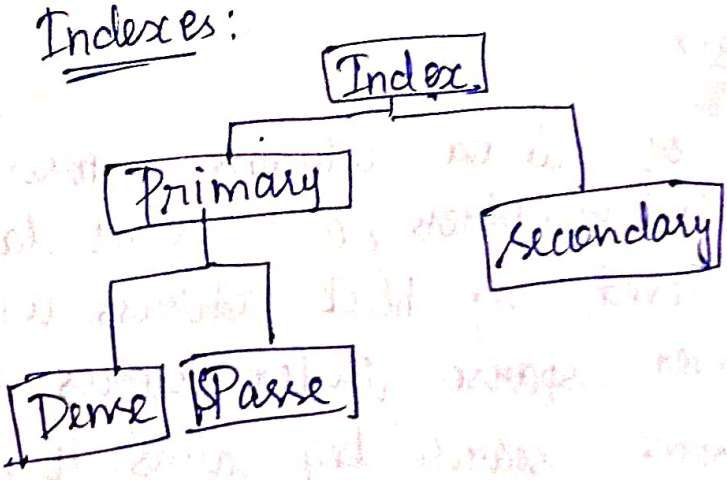
→ 1st column comprises a copy of the primary (or) candidate key of a table

→ 2nd column contains a set of pointers for holding the address of the disk block where that specific key value is stored.

Advantages:

* takes a search key as input
* efficiently returns a collection of matching records

Types of Indexes:

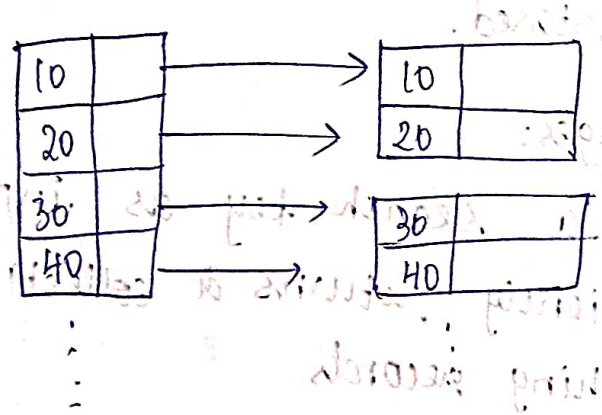


Primary Index:

→ is an ordered file which is fixed length size with two fields. The first field is the same as PK and and field pointed to the specific data block - one to one relationship.

→ Two type:
- * Dense
- * Sparse

→ Dense index: a record is created for every search key in the db, this helps you to search faster but needs more space index records.

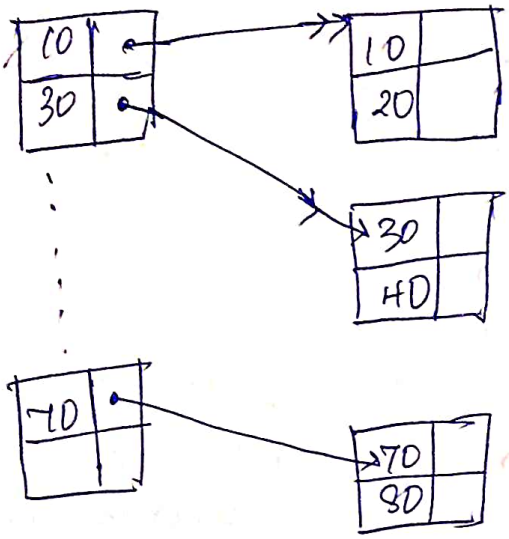


→ Sparse Index:

A range of index columns stores the same data block address, and what data needs to be retrieved, the block address will be fetched. However sparse index stores records for only some search key values. It needs less space, less maintainance over read.

(or):

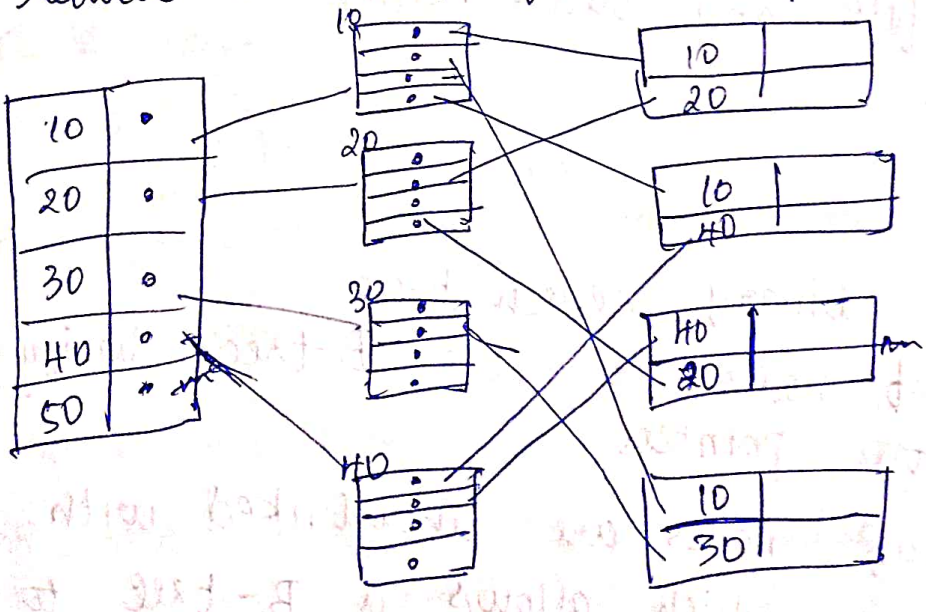
It is slower compared to the dense index, for locating records



128
45
85

Secondary index: Generated by a feild which has a unique value for each record and it should be a candidate key called non-Clustered index.

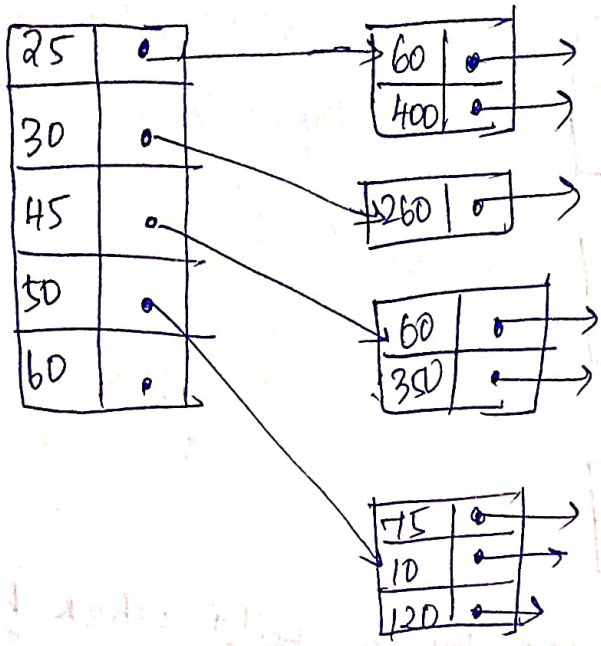
→ Two levels db indexing technique is used to reduce the mapping size of the first level.



B
S

Eg:

Multilevel Index:



The outer blocks are divided into inner blocks which in turn point to the data disk.

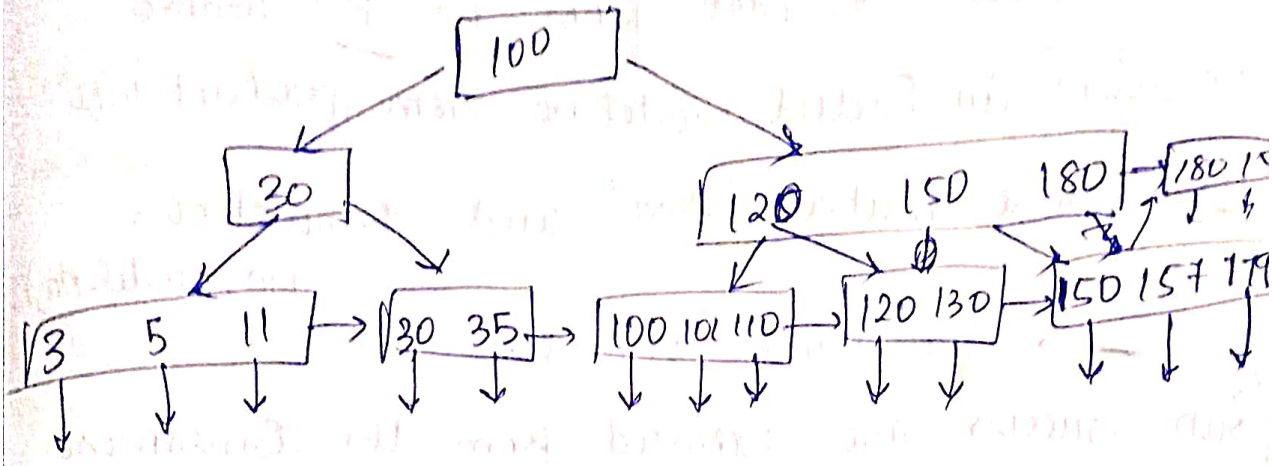
Multi level indexing in db is created when a primary index does not fit in memory.

You can reduce the no. of disk accesses to sort any record and kept on a disk as a sequential file and create sparse base on that file.

segregates the main block into various smaller blocks so that the same data can be stored efficiently

B-Tree:

- Balanced binary search tree
- All leaf nodes of the B-tree signify actual data pointers.
- All leaf nodes are interlinked with a linked list, which allows a B-tree to support both random & sequential access.



Create index in_pc_mo on pc (model_no);

LAB:

Sub Query / Nested Query:

→ Outer Query

→ Inner Query

select salary from employee where salary > (select salary from employee where empid = 700);

- select salary from employee where salary > (select salary from employee where empid = 700);
- select salary from employee where salary >= (select avg(salary) from emp); ⇒ These 2 return single value. so relational
- select model_no, price from pc where model_no in [select model_no from product_info where maker = 'IBM'];

⇓ referential integrity works here

⇓ more than 1 table involved

⇒ This has to return a set of values so 'in'.

Eg:

- Select model-no, price from pc P where model_no in (select model-no from product_info pin where maker = 'IBM' and p.model_no = pin.model_no)

— → alias name for table

(Sub queries are executed from the innermost query.)

Sub query:

→ Correlated subquery

→ Non-correlated subquery.

Non-correlated: The inner query is executed only once and then sends it to the outer loop.

Correlated: The inner query is executed for every record of the outer query.

Examples:

- select salary from employee where salary > (select avg(salary) from employee);

- select empid from employee where empid in (select mgr_id from employee);

JOINS:

- Inner Join
  - Equi Join
  - Non Equi Join
  - self Join → In a single table
- Outer Join
  - Left
  - Right
  - full

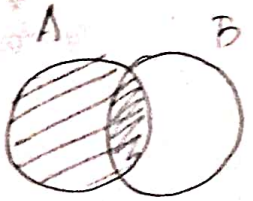
For n tables we can have a max of n-1 joins

```
select model_no from pc p, product_info pin
where p.model_no = pin.model_no;
```

↳ Inner Joint

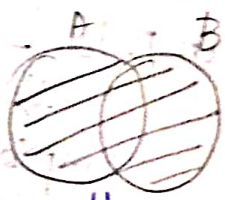
```
Select * from pc, product_info;
```

↳ Cartesian product (no condition joining)



⇓ Left outer join

⇓ right outer join



⇓ Full outer join

SYNTAX:

Select w.name, wk.name from worker w, workerskill wk
where w.name = wk.name; → equi joint

select name from worker w, workerskill wk
where w.name = wk.name;

select w.name, wk.name from worker w,
workerskill wk where w.name <> wk.name;
non equi joint not equal or !=

select w.name, wk.name from worker w,
workerskill wk where w.name = wk.name and
w.age >30; → non-equi joint

select e1.emp_id, e2.mgr_id from employee e1, e2
where e1.emp_id = e2.mgr_id;
employee
↓
employees working as manager

select w.name, wk.name from worker w,
workerskill wk where w.name <> wk.name(+);
(or)
select w.name, wk.name from worker left outer
join workerskill on w.name = wk.name;

eg:

select w.name, wk.name from worker w, workerskill wk where wk.name(+) = wk.name

↳ right outer join

Views: is a logical rep of physical table

- Since DBMS is a multiuser accessible model

ID name no. ... (25 columns)

— — — —

— — — —

— — — —

View 1 :

ID	name
—	— —
—	— —
—	— —

View 2:

ID	no.	—
—	—	— —
—	—	— —
—	—	— —

→ changes in the view, will be reflected in the actual table and vice-versa.

→ there is no separate memory for view table.

Create view <view_emp> as select (emp_id, ename)) from employee;

Create view emp_view_sum as select sum(salary), avg(salary) from employee where dept_id = 7;

Delete from view_emp where e_name = 'RAM';

Drop view emp_view;

view_emp as select fro

Indexes:

Create index index_emp on employee (ename);

Create index index_dept on dept (dname);

Create synonym <empl> for employee;

alias name

create index ie on empl

Sequence:

create sequence <sequence_name>
increment by n
start with n
max value n
min value n
cycle / no cycle
cache / no cache;

inc by
stat with
max value
min value

Ex:

- create sequence dept-deptno
  increment by 1
  start with 91
  max value 100
  no cycle
  no cache;
- select sequence_name, minvalue, max-value increment-by, last-number from user-sequences;
- insert into dept (deptno, dname, loc) values (dept-deptno.nextval, 'marketing', 'New delhi');
- select dept-deptno.currval from ~~dept~~; dual;
- alter sequence dept-deptno.currval
  increment by 1
  maxvalue 9999
  No cache
  No cycle;
- drop sequence dept-deptno;

PL | SQL: Procedural language:-

→ An extension to SQL with design features of programming languages (procedural & oops)

→

PTO

Advantages:

⇒ Avoids network traffic → n no. of SQL statements can be executed @ a time

⇒ Pre compiled

⇒ Portable (can work on any platform)

Declaration:

⇒ v_first_name VARCHAR(35);
⇒ v_count NUMBER:=0;

Executable section:

```
BEGIN
  SELECT first_name, last_name
      INTO v_first_name, v_last_name
      FROM student
      WHERE student_id =123;
  DBMS_OUTPUT.PUT_LINE
   ('student name:' || v_first_name || ' ' || v_last_name);
  END;
```

Exception Handling section:

⇒ these statements are executed when a runtime error occurs

```
EXCEPTION
      WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE ("Nope studen");
END;
```

Reference Variables:

(OR data type)

→ It directly references specific database column or row

→ It assumes data type of associated column or row

⇒ %.TYPE:

variable_name tablename.colname %.TYPE;

→ %.ROWTYPE:

variable_name tablename %.ROWTYPE;

```
DECLARE
    sname    sailors.sname%.TYPE;
```

type of column sname in sailors

```
DECLARE
    reserves_rec  reserves%.ROWTYPE;
```

accessing fields in reserves_rec:

reserves_rec.sid := 9; → (will it change?)

Creating a Record: → a user defined data type like struct

```
TYPE  sailor_record_type IS RECORD
    (sname      VARCHAR2(10),
     sid        VARCHAR2(9),
     age        NUMBER(3),
     sailor_record  sailor_record_type;);

BEGIN
    sailor_record.sname := 'peter';
    sailor_record.age := 45;
```

eff.

To get user input:

Set Serveroutput on

Accept p-price Prompt 'Enter:'

Comments: /* */

Normalisation:

①

Stu-id	Proj_id	stu-name	Proj-name

②

Stu-id	St-name	City	Zip

③

S-id	Prog-lang	Stud-age
101	Compiler N/w	20
101	JAVA	20
102	DBMS	20
103	S/w Engg	21
103	Compiler	21

All these tables are in first normal form

Convert to ① 2NF ② 3NF ③ 2NF

①

Stu-id	Stu-name

Proj-id	Proj-name

already in 2NF

② ~~Stuname~~

→ Candidate key

Stu-id	Stu-name	Zip

Zip	City

(3)

S_id	S_age

S_id	Program

B

1NF → repeating groups are non existent

2NF → nonkey attributes are fully dep. on key attributes

3NF → transitivity is removed

BCNF → Every determinant is a candidate key

4NF →

5NF →

Emp-id	Emp-nationality	emp-dept	dept-id	dept-no-emp
1001	UK	Production	D001	200
1002	USA	Sales	D002	300

① Student

Sid	Sname	Flag
S1	Alex	A
S2	Henry	B
S3	Michel	B
S4	Charlie	D
S5	Ace	D
S6	HIngo	C

Team

Team-name	Flag
Highlighter	A
warriors	B
Supercool	C
Beast	D

Formal Lang Query:

select Salary from emp where empid = 123;

Salary ↓ projection

where empid = 123; ↓ selection

$\sigma_{A=B \wedge D>5}(r)$: Select Operation :

- $\sigma_p(r)$ where p is the selection predicate.
- $\wedge$ (and), $\vee$ (or), $\neg$ (not)

Cartesian join vs natural join. (HW) CIA 2

2nd maximum Salary: select max(salary) from details where salary < max(salary)

Ummm..

P

Ref:

select max (salary) from employee where (salary in (select salary from employee where salary < max (salary)));

↓ Selection

Select : σ
Project : Π
Union : $\cup$
Set diff : $-$
Cartesian : $\times$
rename : ρ

$\sigma_{p}(r)$ → relation

↓ selection predicate

(b) $\Pi_{A_1, \ldots}(emp)$

replace ("Sam

translate ("

$\Pi_{course}(\sigma_{sem = "fall" \wedge yr = 2009}(sec)) \cup$

$\Pi_{ID}(\sigma$

The lost Update Problem:

↳ Some operation is interleaved inc it makes the value of some db

The Temporary Update:

↳ failure after update of then another trans accesses it before rollback

Incorrect Summ

↳ one trans is agg another as update

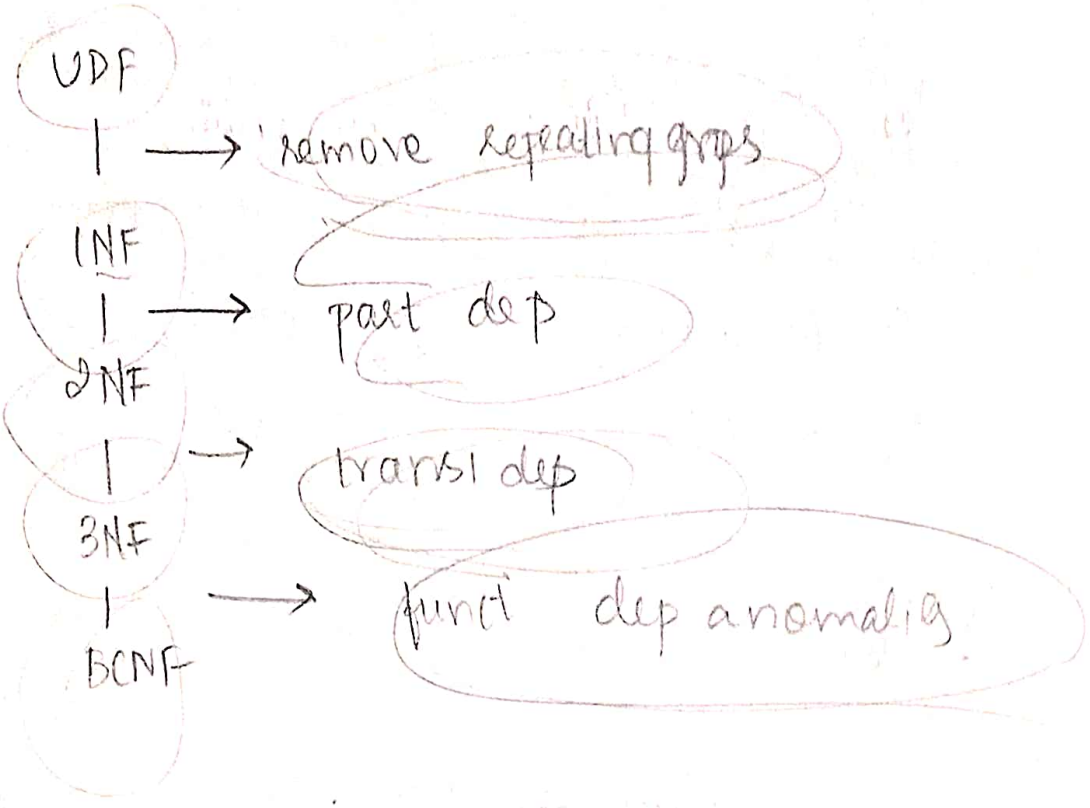
misc out on the updated values

```
Sum = 0
Read A
Sum = Sum + A
```

```
Read X
X : X - N
write X
```

```
Read X
Sum = Sum + X
Read Y
Sum = Sum + Y
```

```
Red Y
Y : Y + N
Write Y
```




```
select  w.name , wk.name from
        worker w, workcenter wk
where   w.name = wk.name.
```

round (price, 2)

length (name)

substr (phone, 9, 13)

instr (name, 'a')

soundex (city) = so

```
select *  from  printer where
price in (select price from
printer where model no in
(select model no from PC
where make
```

Serializability:

A schedule is serialable if it is equivalent to a serial schedule

1. Conflict serializable
2. View "

Conflict Serializable:

⇒ if a schedule S can be transformed into a schedule S' by a series of swaps of non-conflicting instructions, we say that S and S' are conflict equivalent.

⇒ Schedule S in Conflict seriable if it is Conflict equi. to a serial schedule

View Serializable:

A schedule is called view Serializable if it a view equal to a serial schedule

Conflicting operation:

* They belong to diff. trans
* They operate on the same data item
* Atleast one of them is a write operation.

Precedence Graph for testing Conflict Serializability

Graph (V, E) consisting of a set of nodes $V = \{T_1, T_2, T_3 \ldots T_n\}$ and set of directed edges $E = \{e_1, e_2 \ldots e_n\}$.

- the graph contains one node for each transaction T_i
- An edge e_i is of the form $T_j \rightarrow T_k$ where T_j is the starting node of e_i and T_k is the ending node of e_i.
- An edge e_i is constructed between node T_j to T_k if one of the operations on T_j appears in the schedule before some conflicting operation in T_k.

Algorithm:

1. Create a node T_i in the graph for each participating transaction in the schedule.
2. For the conflicting operation read_item and a write_item if a transaction T_j executes a read_item(x) after T_i executes a write_item(x) draw an edge from T_i to T_j in the graph.
3. For the conflicting operation write_item(x) and read_item(x), if a transaction T_j executes a write_item(x) after T_i executes a read_item(x) draw an edge from T_i to T_j in the graph.

that is so rude of U WOW!

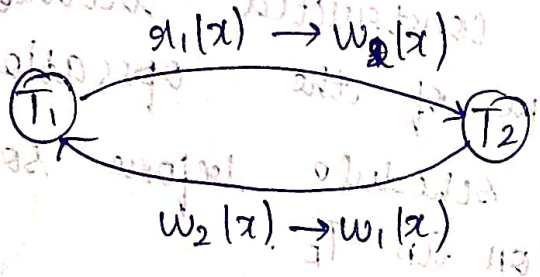
eff:

4. for each conflicting operation write_item(x) and write_item(x) of a transaction Tj executes a write_item(x) after Ti executes a write_item(x), draw an edge from Ti to Tj in the graph.

5. The schedule is serializable if there is no cycle in the precedence graph.

Examples:

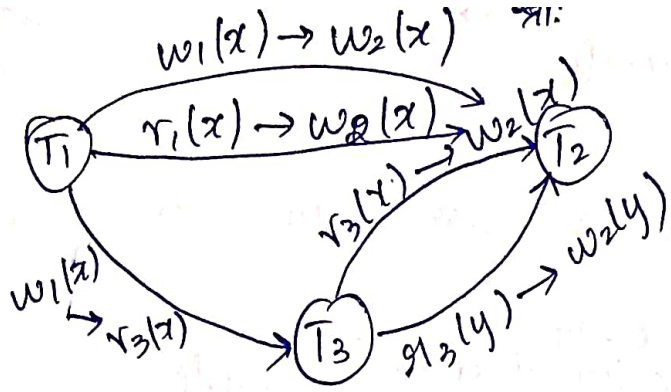
1. S: $r_1(x)$ $r_1(y)$ $w_2(x)$ $w_1(x)$ $r_2(y)$



→ not conflict serializable

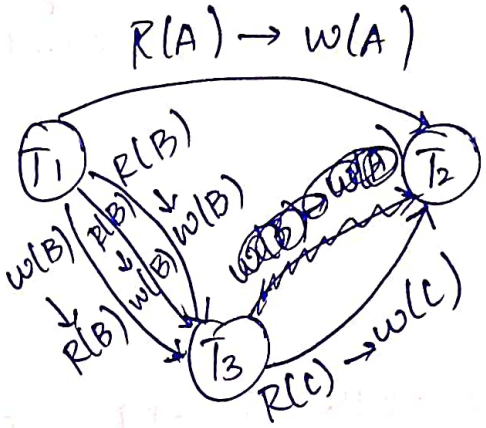
2. S: $r_1(x)$ $r_3(y)$ $w_1(x)$ $w_2(y)$ $r_3(x)$ $w_2(x)$

T_1	T_2	T_3
$r_1(x)$		
		$r_3(y)$
$w_1(x)$		
	$w_2(y)$	
		$r_3(x)$
	$w_2(x)$	



3.

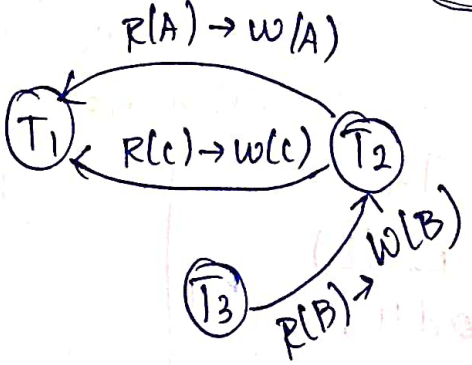
T_1	T_2	T_3
R(A)		
R(B)		
	R(A)	
	R(C)	
W(B)		
		R(B)
		R(C)
		W(B)
	W(A)	
	W(C)	



⇓ its conflict serializable

4.

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



~~No cycle~~ | ~~no overall loop~~ →

⇓

$T_3 \rightarrow T_2 \rightarrow T_1$

if loop → not conf ser.

if cycle → not

Lock based Protocols:

⇓

a mechanism to access data item

Modes:

→ shared lock (even tho its locked other users can access (ex: read))

→ exclusive locks

↳ unless the lock is released nobody can access (ex: write)

Example:

```
Lock_s(A)
read(A)
unlock(A)
Lock_s(B)
read(B)
unlock(B)
display(AB)
```

- Two phase locking protocol (2PL)

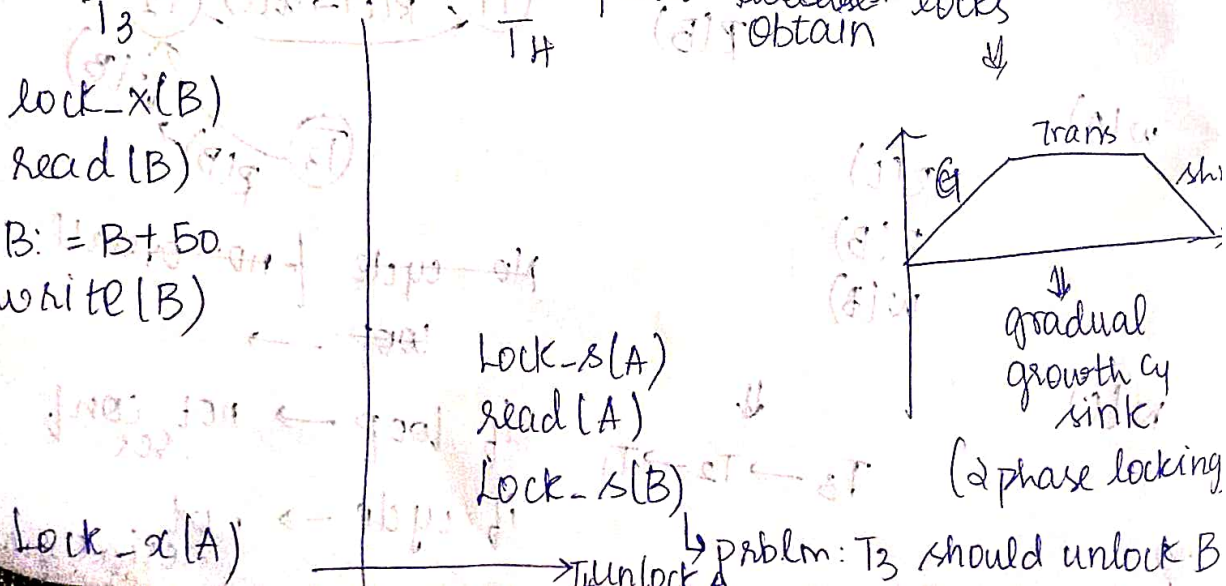
Phase 1: growing phase

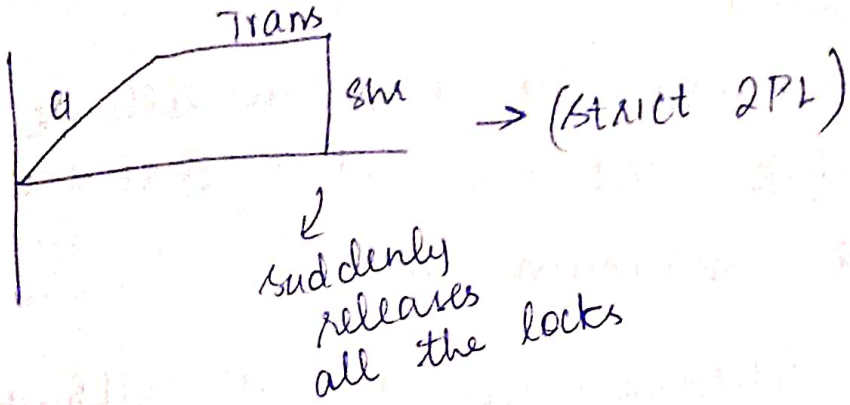
- transaction may obtain locks
- transaction may not release locks

phase 2: shrinking phase

- transaction may release locks
- transaction may not ~~release~~ obtain locks

⇓





the problem in T_3 & T_4 → deadlock

↓

One transaction waiting for another transaction to release a lock on an item.

(Prevention & disadv of dead lock)

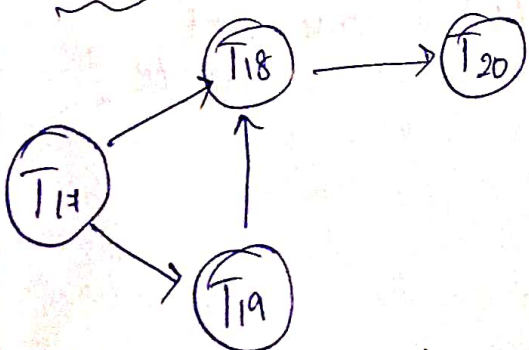
Deadlock handling:

- deadlock prevention:

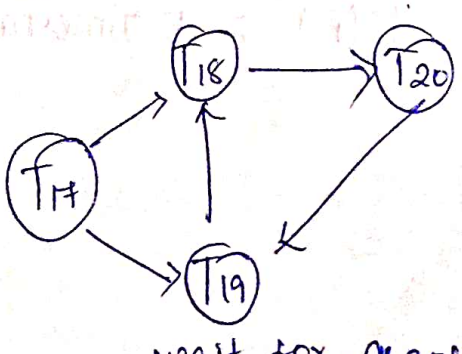
→ requires each transaction to locks all its items before it begins executions.

→ impose partial ordering of all data items and require that a transaction can lock data items only in the order specified by partial order (Graph based).

- deadlock detection algorithm:



wait for graph w/o cycle



wait for graph with a cycle

~~eft~~

Deadlock recovery:

→ Some transactions will have to rollback to break deadlock. Select the victim that will incur the minimum cost.

→ Rollback - determine how far to rollback transaction

→ Total rollback: Abort the transaction and then restart it.

Timestamp based protocol:

→ Each transaction is issued a timestamp when it enters the system. If an old transaction T_i has timestamp $TS(T_i)$ a new transaction T_j is assigned timestamp $TS(T_j)$ such that $TS(T_i) < TS(T_j)$

-9A:

Ex1:

T_1	T_2
r(A)	
w(A)	
	r(A)
	w(A)
r(B)	

step1:

T_1	T_2
r(A)	
w(A)	
	r(A)
r(B)	
	w(A)

step2:

T_1	T_2
r(A)	
w(A)	
r(B)	
	r(A)
	w(B)

$\Rightarrow$ serial

Ex2:

T_1	T_2
r(A)	
w(A)	
	r(A)
w(A)	
r(B)	

There is no adjacent non conflict pairs exist

This schedule cant ~~to~~ be converted into serial.

~~eg:~~

Data Warehousing and Data mining:

⇒ Decision support systems: used to make business decisions often based on data collected by on-line transaction-processing systems.

⇒ Data analysis, Statistical analysis, Data mining
↓
discover meaning, knowledge automatically from data in the form of statistical rules & patterns

⇒ Data Warehouse: is a repository or an archive of information gathered from multiple sources, stored under a unified schema.

⇒ Imp. for large businesses that have various data inputs and outputs

⇒ Greatly simplifies querying,

⇒ permits study of historical trends

⇒ shifts decision support query load away from transaction processing systems.

Design Issues:

→ When and how to gather data
  * Source driven architecture (source to wh periodically)
  * Destination " " (req from wh to source)

→ keeping wh exactly synchronised with uptodate data is costly.

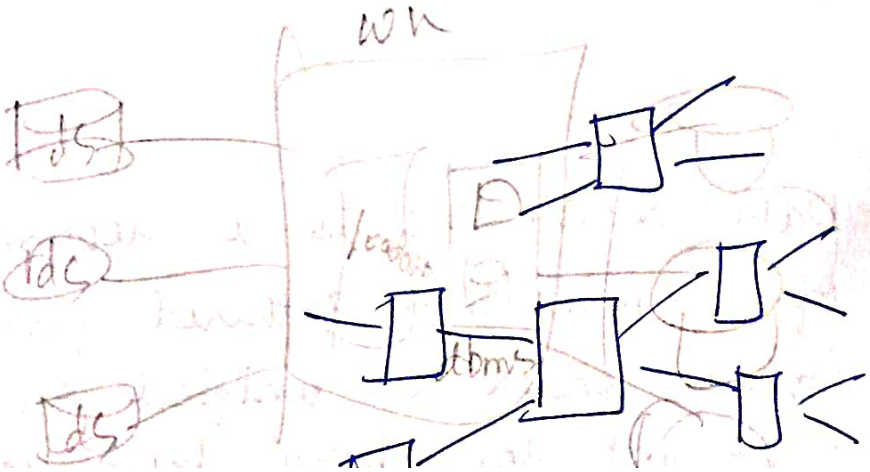
→ What schema to use

→ Data cleansing

→ What data to summarize

WH Schemas:

→ Star schema
→ Snowflake "
→ constellation "



*11:

Data mining: process of semi automatically analyzing large databases to find useful patterns.

→ predictions based on past history
  * classification
  * Regression

→ Descriptive patterns
  * Associations (causations)
  * Clusters

Classification rules:

→ Summarised using a decision tree

Decision Tree:

→ each internal node partitions the data into groups based on a partitioning attributes ay a conditioning for the node

→ leaf node:
  * all items @ the node belongs to the same class
  * all attributes have been considered ay no further partitioning is possible.

~~If~~ no. of classes = k
no. of instances = |S|

Best Split:

$$Gini(S) = 1 - \sum_{i=1}^{k} p_i^2$$

p_i = fraction of inst in class i

- When all instances are in same class = 0
- reaches maximum if each class has the same no. of instances.

$$Entropy(S) = -\sum_{i=1}^{k} p_i \log p_i$$

When a set S is split into multiple sets $S_i, i = 1, 2, .. r$ we can measure the purity of the resultant set of sets as:

$$purity(S_1, S_2, \ldots S_r) = \sum_{i=1}^{r} \frac{|S_i|}{|S|} purity(S_i)$$

Information gain due to particular split of S into $S_i, i = 1, 2 .. r$

$$\text{Info gain} = purity(S) - purity(S_1, S_2 .. S_r)$$

$$\left[\text{Information gain ratio} = \frac{\text{Info gain}}{\text{Info content}}\right] \uparrow \rightarrow \text{best split}$$

$$\text{Info content} = -\sum_{i=1}^{r} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

Other types of classifiers:

→ Bayesian classifiers

→ Regression

→ clustering

Clustering: → unsupervised learning

→ inter cluster & intra cluster distance

→ hierarchical and partitional sets → nested clusters as a tree → non overlapping subsets

→ Agglomerative & Divisive clustering

↓ small cl to big

⇓ big cl to small

Distributed Databases:

→ consists of loosely coupled sites that share no physical component

→ 2 or more database files @ diff locations in the network.

→ They run independently but consistently

$$1 - \sum_{i=1}^{K} P_i^2$$

loosely coupled sites

Homogeneous :-

⇒ All sites have identical s/w

⇒ Are aware of each other & agree to cooperate in processing user requests

⇒ Each site surrenders a part of its autonomy

⇒ appears as a single system to user

Heterogeneous:-

⇒ diff sites may use diff schemas & s/w
↓ problm for query processing
↓ Problem for trans. processing

⇒ may not be aware of each other & may provide only limited facilities for cooperation

Distributed Data Storage:

⇒ Replication:
→ maintains multiple copies of data, stored in diff sites, for fault retrieval and fault tolerance

→ Adv:
→ Availability
→ Parallelism
→ Reduced data transfer

dis:
→ Increased cost of updates
→ ↑ complexity of concurrency control

~~Str~~:

⇒ Fragmentation:

→ division of relation r into fragments $r_1, r_2, \ldots r_n$ which contain ~~suff~~ sufficient information to reconstruct relation r

→ Horizontal & vertical

Horizontal ↓
each tuple is assigned to one/more fragments

vertical ↓
split into several smaller schemas
→ all schemas must contain a common candidate key (or super key) to ensure lossless join property
→ a tupple-id attribute may be added to each schema to serve as cand. key

→ Adv:

Hori:
* PP on fragments of a relation
* tuples are located where they are most frequently accessed

Vert:
* PP on a relation
* each part of tuple is located where they are most frequently used

diff b/w distributed & centralised

dis: * very high speed access sometimes
* recursive fragm → very expensive

Availability
parallelism
reduced data transf

~~11~~

Web databases: peh

↳ access the database on the internet

Information Retrieval: locates relevant documents on the basis of user input such as keywords or example documents.

Ex: Web search engines

Diff from database systems:

⇒ IR systems dont deal w/ transaction updates

⇒ DBMS deals w/ structured data

⇒ IRS deals with some querying issues not generally addressed by DBMS

* Approx searching by keywords
* Ranking of retrieved ans by estimated degree of relevance.

Keyword Search: done by first calculating relevance → Term frequency
→ Inverse document frequency
→ Hyperlinks to documents

Relevance Ranking:

⇒ TF-IDF: $n(d)$ → no. of terms in the document
$n(d,t)$ → no. of occurrences of term t in the document d

Relevance of a document d to a term t:

$$TF(d,t) = \log\left(1 + \frac{n(d,t)}{n(d)}\right)$$

Relevance of a document to query Q

$$r(d,Q) = \sum_{t \in Q} \frac{TF(d,t)}{n(t)}$$

- Words that occur in title, author list, section headings are an imp.
- Words whose first occurrence is late in the document → less imp.
- 'a', 'an', 'the', 'it' → eliminated → Stop words
- Proximity: If keywords in query occur close together in the document, → high imp

Similarity based Retrieval: retrive documents similar to a given document

* Relevance feedback: user selects a few documents from those retrieved by keyboard query & returns similar docs

* vector space model:
  - vector for document d goes from origin to a point whose ith coordinate is $TF(d,t)/n(t)$
  - cosine of angle b/w vectors → measures of similarity

Relevance using Hyperlink: ipeh!

- connections to social networking theories
- Hub & authority based ranking

what what

DB security:

⇒ Types of security issues:
- Legal and ethical issues
- policy issues
- System related issues
- The need to identify multiple security levels

⇒ Threats to databases:
- Loss of integrity
- Loss of availability
- Loss of confidentiality.

⇒ Countermeasures:
- Access control → ~~by creating user acc & pass~~
- Inference "
- flow "
- Encryption

⇒ Types of db security mechanisms:
- Discretionary " "
- mandatory " "

- Access control:
  - → for restricting access to unauthorised users
  - → done by creating passwords & user accounts to manage logins

- Inference control:
  - → controlling the access to statistical database that provide statistical data & summaries
  - → to safeguard such dbs we use inference control measures

- flow control:
  - → way to prevent information from flowing in such a way that it reaches unauthorised users.
  - → channels that are pathways for info to flow implicitly in ways that violate the security policy of an organisation are called covert channels.

- data encryption:
  - → used to protect sensitive data transmitted via some comm. network
  - → data is encoded using some encoding algorithm.
    - unauthorised users will find it difficult to decipher it.
    - authorised users will be given decoding algorithms

eA:

DBA : Database administrator:
→ central authority for managing a dbs
* granting privileges
* classifying users

Page (12)

- DBA has a DBA account in the DBMS.
  - called a system or superuser acc
  - ⇒ Acc creation → access control
  - ⇒ Privilege granting ⇒ } discretionary
  - ⇒ " revocation }
  - ⇒ security level assignment → mandatory

① account creation
② system log
③ db audit

Discretionary Access Control Based on Granting & Revoking Privileges:

→ Account level : privilege that each account holds independently of the relations in the db.

→ relation level: privilege to access each individual relation or view in the db.

Types of privileges: (account level)

- create schema / create table
- create view
- Alter
- drop
- modify
- select

Pg(19)
(~~relation~~ ~~proof~~).

follows access matrix model

rows → subjects (users, acc, prgrms)

columns → objects (relation, record, col, views)

each position M(i,j) → type of privilege that i holds over j

Pg(20)