Database Management Systems

Module – I :- Database Concepts :-

Data :- Consists of raw facts, & when organized may be transformed into information.

Database :- A collection of data organized to meet user's needs.
ie, Collection of table (in short).

Database Management System (DBMS) :- A group of programs that manipulate the database & provide an interface between the database & the user of the database or other application programs.

i) A collection of programs that enables you to store, modify & extract info. from a database.

ii) There are many diff. types of DBMS, ranging from small systems that run on personal computers to huge systems, that run on mainframes.

iii) Examples of database applications :-
1) Computerized library systems
ii) Automated teller machines
iii) Flight reservation systems
iv) Computerized parts inventory systems.

Basics of Data Arrangement & Access

Data Form
Google Form
→ Google it

→ Data Hierarchy

i) Field – a logical grouping of characters into a word / group of words / complete number.

ii) Record – a logical grouping of ~~reade~~ related fields.

iii) File – a logical grouping of related records.

iv) Database – a logical grouping of related files.

Eg:

Name : —
Addres : —
City : —
State : —

Fields → Record → File / Table

Collection of many files / tables → database (DB)

Database — Personal file
Department file
Payroll file.

Files — 005-10-6321 Manuel
549-77-1001 Rehan
098-40-1370

Entity :-

:) A generalized class of people, places, or things (objects) for which data are collected, stored & maintained.

Eg :- Customer, Employee, student

Attribute :-

:) A characteristic of an entity; something the entity is identified by

Eg :- Customer name, employee name, student roll no.

Keys :-

:) A field or set of fields in a record that is used to identify the record.

Eg :- A field or set of fields that uniquely identifies the record.

Traditional Approach :- (File system)

* Used different files

↳ separate files are created to store each applications.

Database Approach :-

Single set of data connected to each other

↓

DBMS

↓

Taken by several application programs

↓

which is taken by the user.

Purpose of Database Systems :-

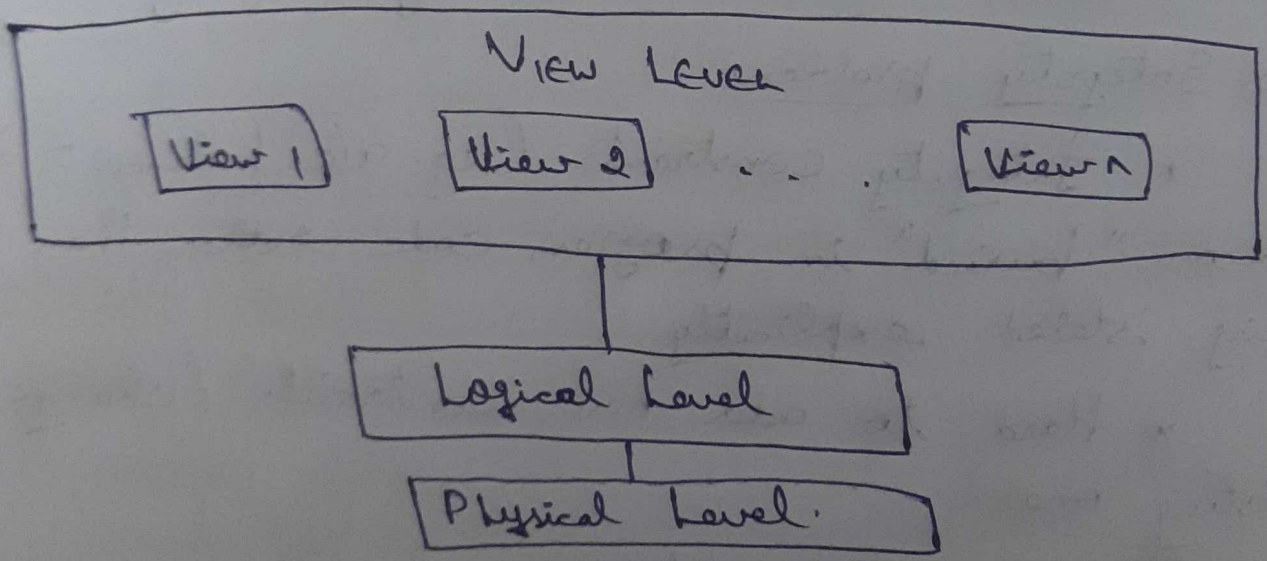
* In early ~~dab~~ days, database apl. were built directly on top of file systems.
* Drawbacks :- of using file systems to store data :-
  → Data redudancy & inconsistency
    * xle file formats, duplication of info. in diff. files.
  → Difficulty in accessing data
    * Need to write a new prg. to carry out each new task.
  → Data isolation – xle files & formats.
  → Integrity problems :-
    * Integrity constraints (eg. a/c balance > 0) become "buried" in program code rather than being stated explicitly
    * Hard to add new constraints / change existing ones.
  → Atomicity of updates :-
    * Failures may leave database in an inconsistent state with partial updates carried out.
    * Eg :- Transfer of funds from one account to another should either complete or not happen at all.

→ Concurrent access by the users:-
* Concurrent accessed needed for performance
* Uncontrolled concurrent accesses can lead to inconsistencies.
* Eg:- 2 people reading a balance & updating it at the same time.

→ Security problems
* Hard to provide user access to some, but not all, data.

Database systems offer solutions to all the above problems.

VIEW OF DATA :- / LEVELS OF ABSTRACTION :-



Physical Level :- (lowest level)
does describes how a record (eg. customer) is stored.

Logical Level :- describes data stored in database & the relationships among the data.

View Level :- Application prgs. hide details of

data types. Views can also hide info.

Instances & Schemas:-

* Similar to types & variables in prg. lang.
* Schema - the logical structure of the DB.
  Eg - The db consists of info. about a set of students & results & the relationship between them.
* Analogous to type info. of a variable in a prg.
* Physical Schema:- DB design at physical level
* Logical Schema - " " at logical level.

→ Instance:- the actual content of the db at a particular pt. in time.

* Analogous to the value of a variable

* → Physical Data Independence:- the ability to modify the physical schema without changing the logical schema.

* Appl. depend on the logical schema.

* In general, the interfaces b/w the various levels & components should be well defined so that the changes in some parts do not seriously influence others.

Data Models:-

Collection of tools for describing:-

→ Data
→ Data relata relationships
→ Data semantics
→ Data constraints

Types of Data Models :-

* Relational Model
* Entity - Relationship model (mainly for db design)
* Object based (Obj oriented
* Semistructured data model
* Network model (Other Network Models)
* Hierarchical model

SQL (STRUCTURED QUERY LANG.)

Data Manipulation Language (DML)

→ Lang. for accessing & manipulating the data organized by the appropriate data model.

→ DML also known as query lang.

Data Definition Lang (DDL)

→ Specification notation for defining the ~~dta~~ db. schema. / Types of SQL query where you create/ define database.

Eg :- create table account (
acc-no char(10),
balance integer);

Two classes of DML :-

1) Declarative (non procedural) — user specifies what data is required without specifying how to get those data.

2) Procedural — user ~~sp~~ specifies what data is req. & how to get those data.

SQL is the most used query lang.

SAMPLE SQL CODE :-

```
create table student ( Sno integer,
                       sname varchar(20)); ] → DDL
insert into student values (1, 'Antony');
select * from student;
```

RELATIONAL MODEL :-

Eg. of tabular data in the relational model. It has a concept of DB which consists of several tables which has a no. of ~~rows~~ rows & columns.

Rows → Tuple / Entity

Column → Attributes

Redundancy (Duplication of Data's)

To avoid redundancy & to make the query lang easy } → Relational Model is used.

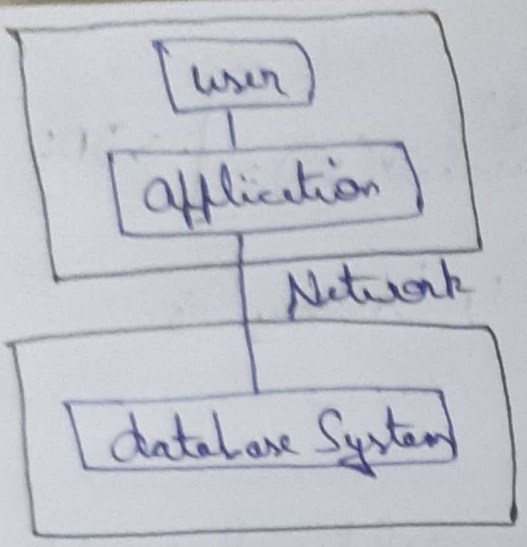
Any question asked in Relational Model can be answered using query lang. via SQL

DATABASE ARCHITECTURE :-

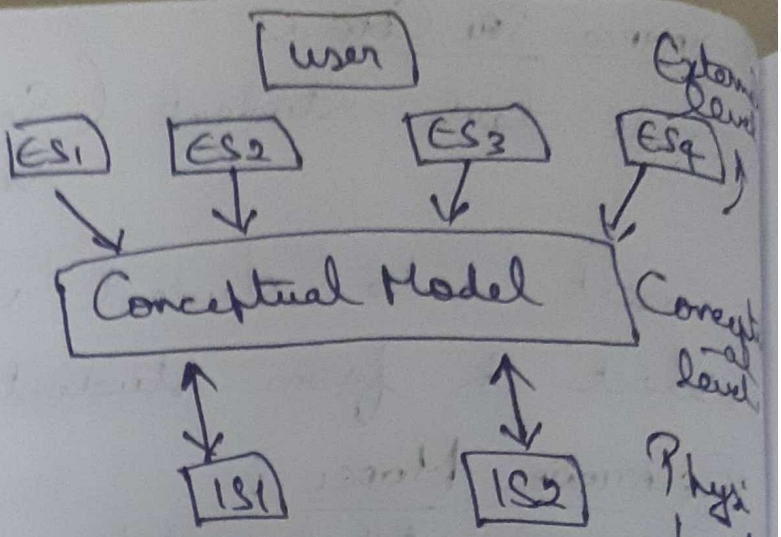
The arch. of a db system is greatly influenced by the underlying computer system on which the database is running:-

* Centralized
* Client-server
* Parallel (Multi-processor)
* Distributed

* 2-tier arch of db
* 3-tier arch. of db



2 tier



3 tier

DATABASE USERS:-

. Users are differenciated by the way they expect to interact with the system.

* Application Programmers :- interact with system through DML.
* Soplisticated users :- form requests in a db query ℓ lang.
* Specialiyd users :- write specialized db. appl. that do not fit into the traditional data processing framework.
* Naive users :- invoke the permanent

Database Administrator :- (DBA)

Coordinates all the activities of the db system; the DBA has a good understanig of the enterprise info resources & needs.

Duties include :-

* Schema definition
* Storage structure & access mthod defintion

* Schema & physical organization modification
* Granting user authority to access the db
* Specifying integrity constraints
* Acting as liaison with users
* Monitoring performance & responding to changes in requirements.

STORAGE MANAGEMENT :-

Storage manager - is a program module that provides the interface between the low-level data stored in the db & the appln. prgs. & queries submitted to the system.

Responsible for the foll. tasks :-

* File Manager
* Authorization & integrity manager
* Transaction manager
* Buffer Manager.

Implements several data structures :-

* Data files → db which contains all the tables.
* Data Dictionary - deals with meta data or stores all meta data
  → data about data
* Indices

Query Processor :-

1) DDL Interpreter

→ Interprets DDL statements & records the definitions in the data dictionary

2) DML Compiler :-

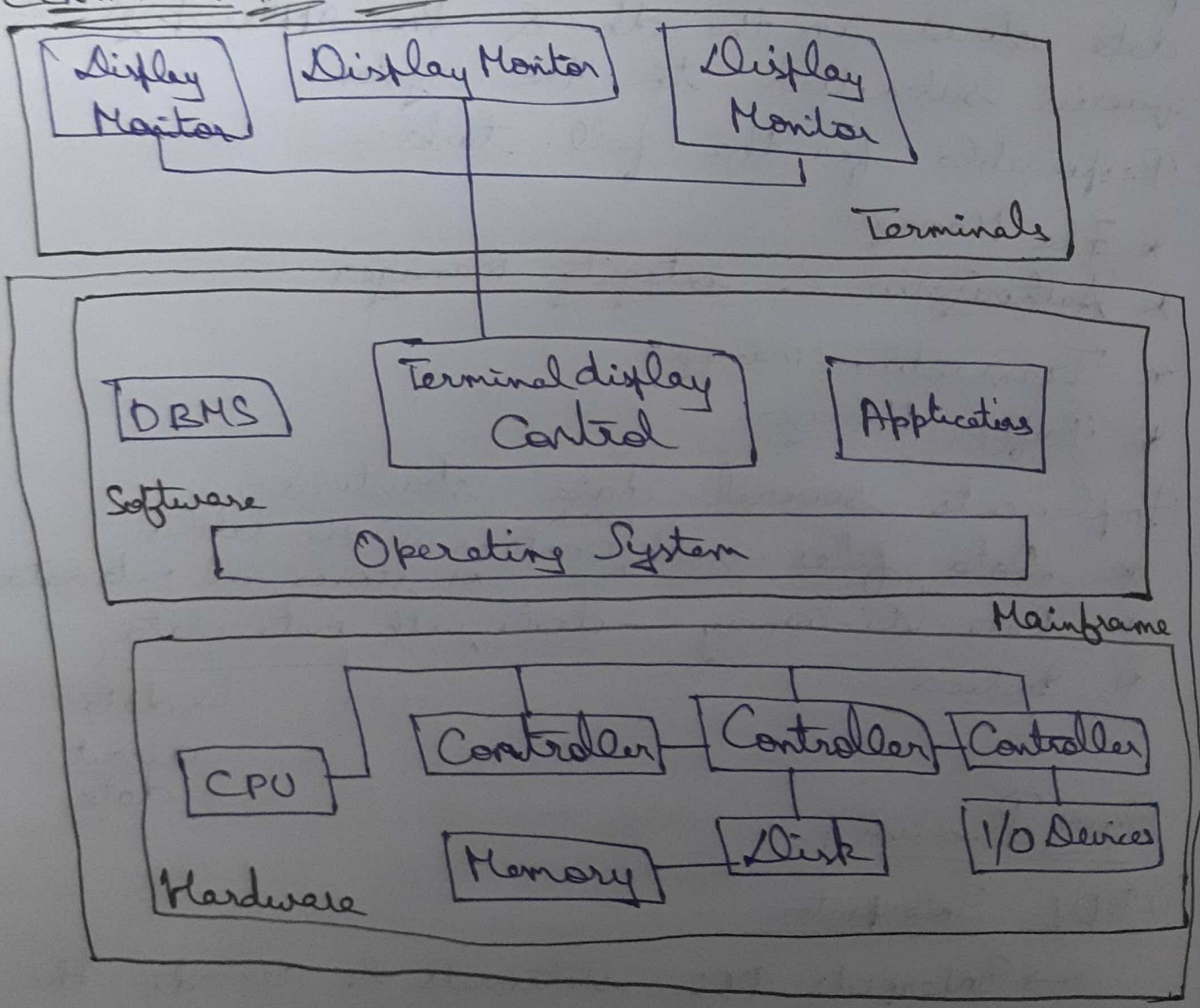
→ Translates DML commands to low-level file system commands

* Evaluation plan
* Query optimization optimization

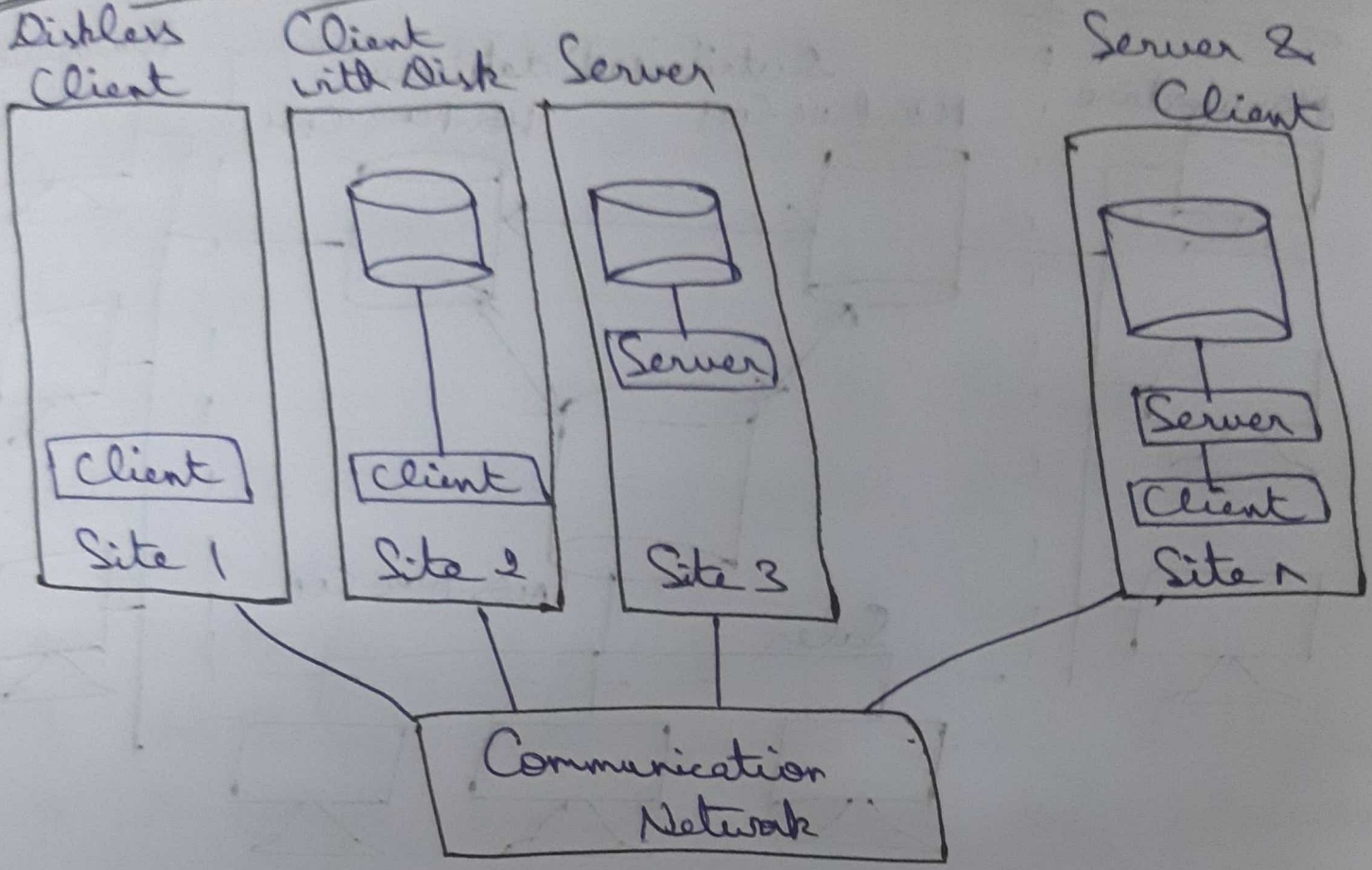
→ Query Evaluation Engine

* Executes low-level instructions generated by the DML compiler.

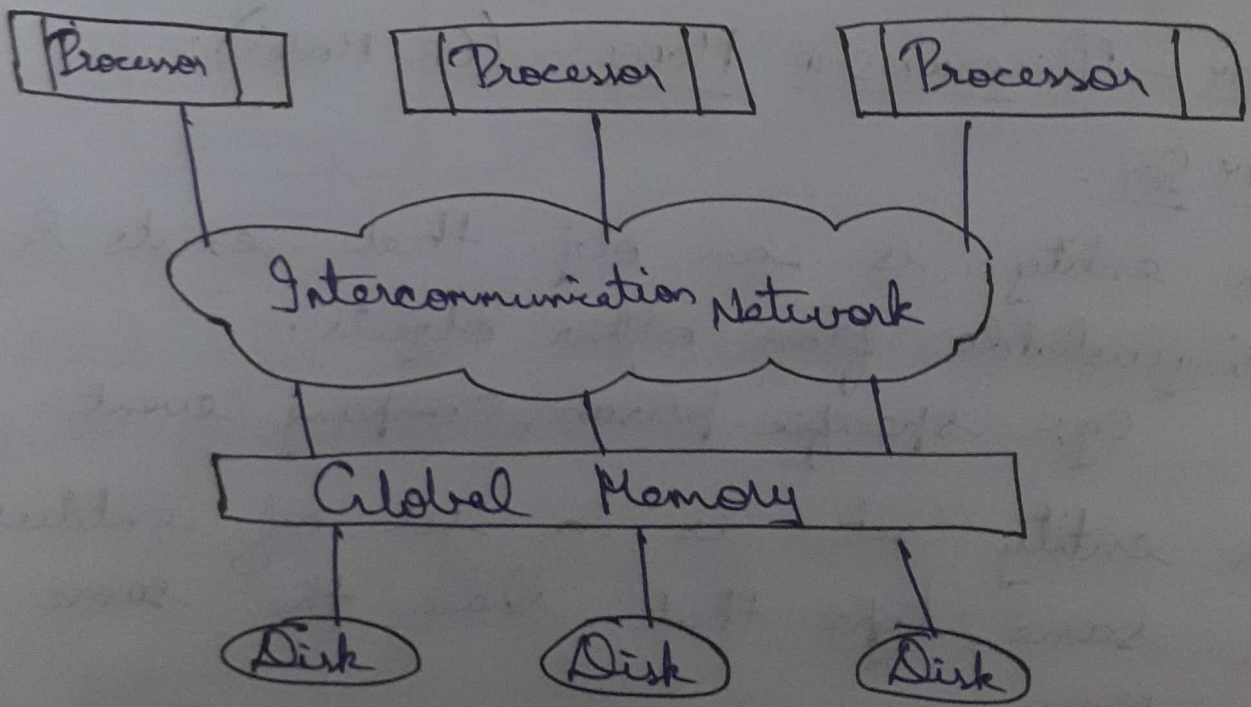
CENTRALIZED DB ARCH :-



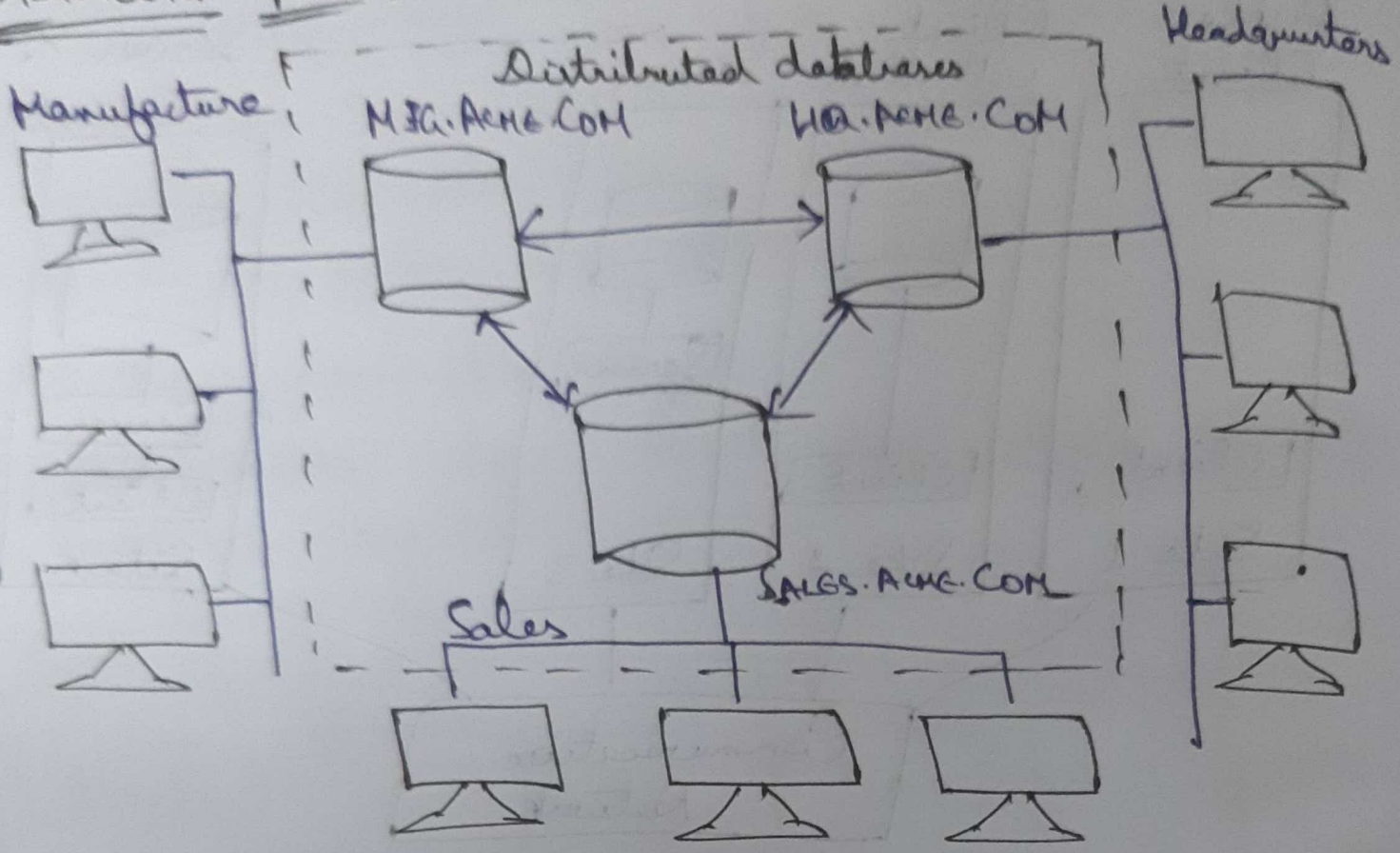
CLIENT - SERVER ARCH :-



PARALLEL ARCH :-



DISTRIBUTED ARCH :-



DATA MODELING :-

ENTITY - RELATIONSHIP MODEL :- (ER Model) :-

ENTITY SET :-

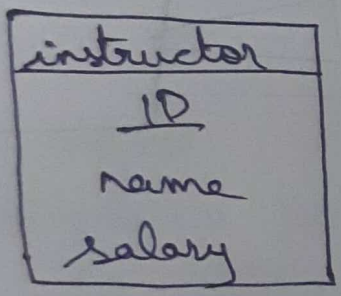
* An entity is an obj that exists & its distinguishable from other objects.
  Eg:- specific person, company, event.
* An entity set is a set of entities of the same type that share the same properties.
  Eg:- set of all persons, companies, trees, holidays.
* An entity is represented by a set of attributes ie, descriptive properties possessed by all members of an entity set.

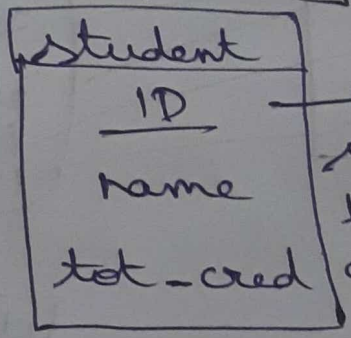
Eg:- instructor = (ID, name, salary) → attributes
course = (course_id, title, credits) ↓ members of the entity set

* A subset of the attributes form a primary key of the entity set i.e, uniquely identifying each member of the set.

REPRESENTING ENTITY SETS IN ER DIAGRAM

* Entity sets can be represented graphically as follows:-
  * Rectangles represent entity sets
  * Attributes listed inside entity rectangles
  * Underline indicates primary key attributes



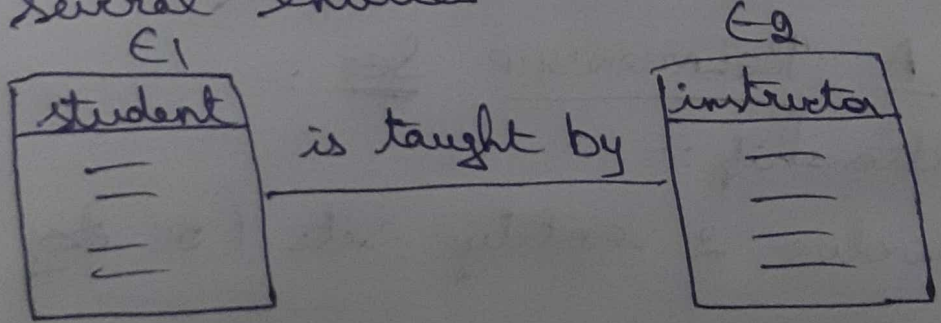


→ primary key should always be the first attribute

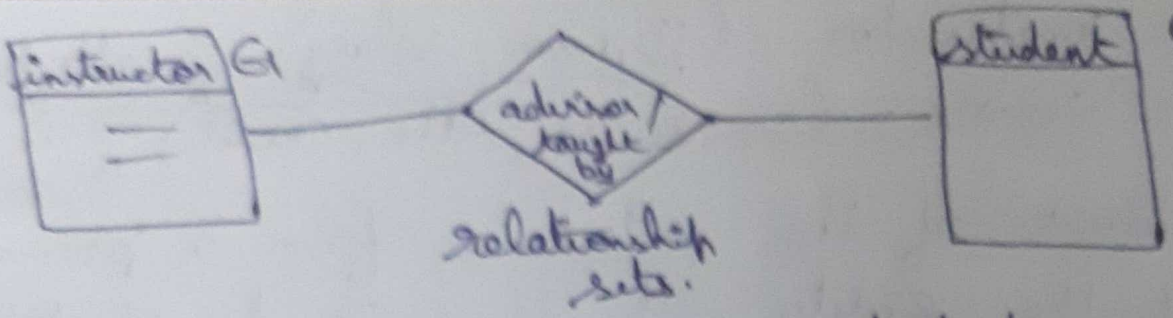
RELATIONSHIP SETS:-

* A relationship is an association / connection among several entities.

Eg:-

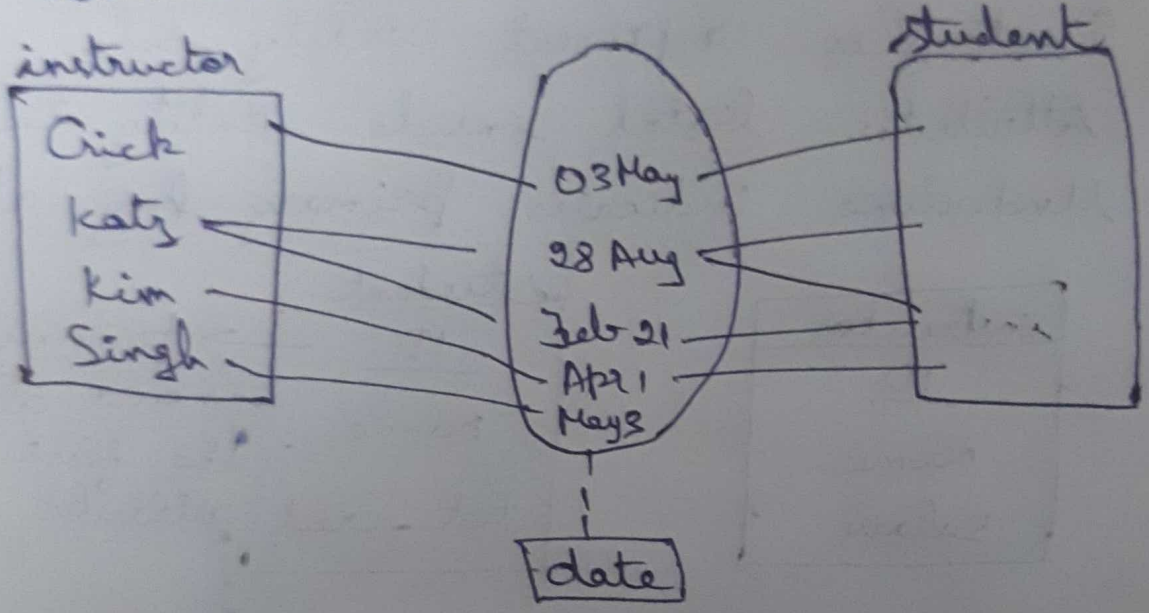


* A relationship set is a mathematical relation among $n \geq 2$ entities, each taken from entity set.
* Diamonds represent relationship sets.



Instructor is advisor to student.

* An attribute can also be associated with relationship sets.
* For eg, the advisor relationship may have the attribute date.



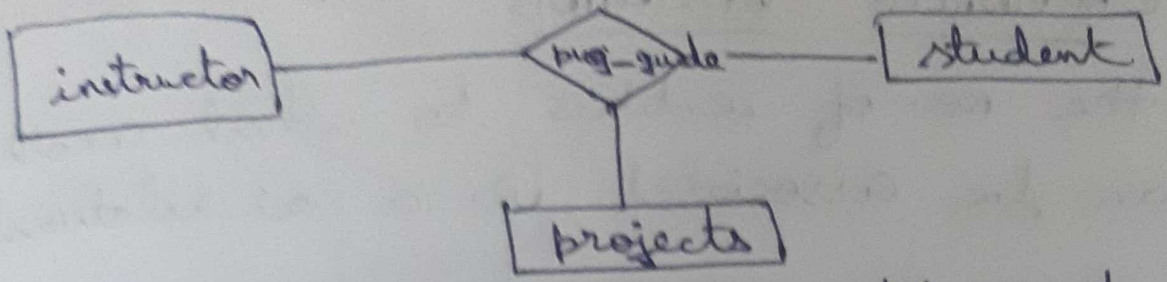
* Relationship sets with attributes are denoted by broken lines in a rectangle.

Degree of a Relationship Set :-

1) Binary relationship :-
   * Involve 2 entity sets (or degree 2)

Non-Binary Relationship Sets :-

1) Students work on research projects under the guidance of an instructor.
   * E-R diagram with a ternary relationship

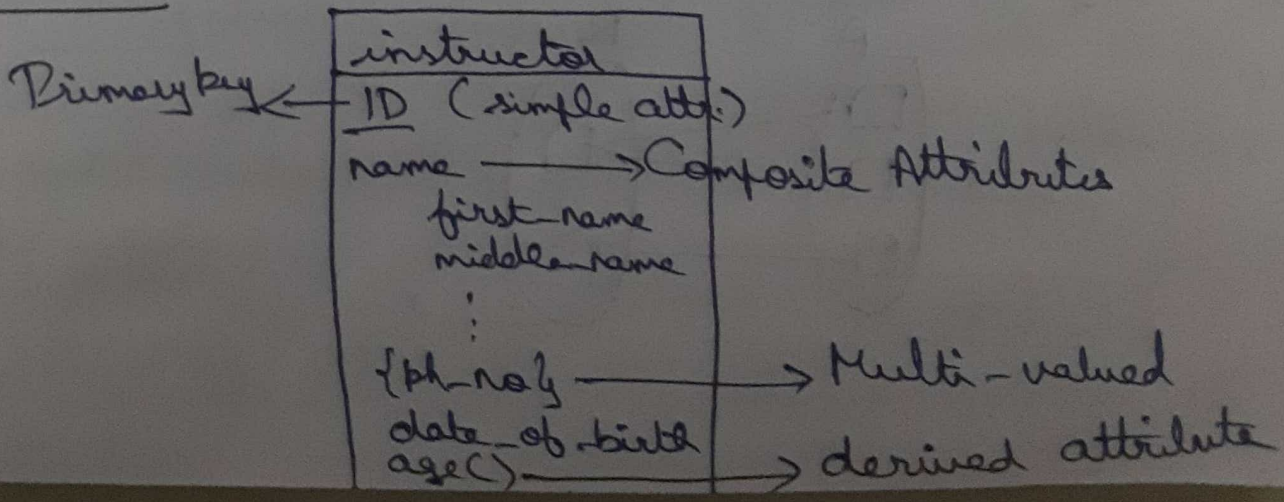


Ternary relationship — 3 entity sets.

TYPES OF ATTRIBUTES :-

1) Simple — which can no more be splitted.
   Eg:- Phno, age, mail id
2) Composite — they are attributes which can be splitted.
   Eg:- address — street
   — pincode
3) Single valued — Attributes with only a single value.
4) Multi-valued — Attributes with more than 1 value attributes are called multi-valued attributes. Eg- address
5) Derived — Can be composed from other attributes. Eg:- age, given date_of_birth.; total mark
6) Domain — the set of permitted values for each attribute.

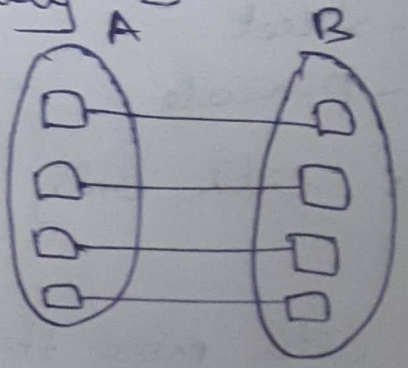
REPRESENTATION :-



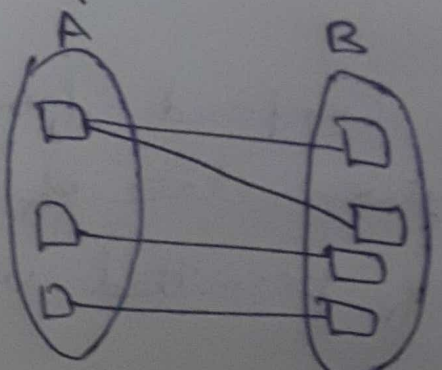
Mapping Cardinality Constraints

* Express the no. of entities to which another entity can be associated via a relationship set.
* For binary relationship set, the mapping cardinality must be of the following types :-

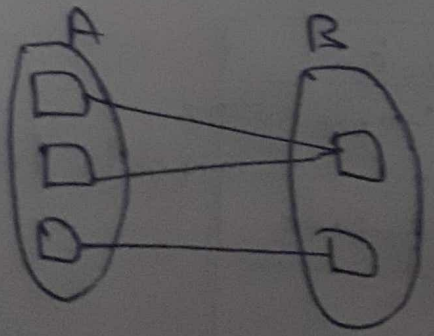
1) One - One :- Each attributes in entity A is mapped exactly to one element in entity B.



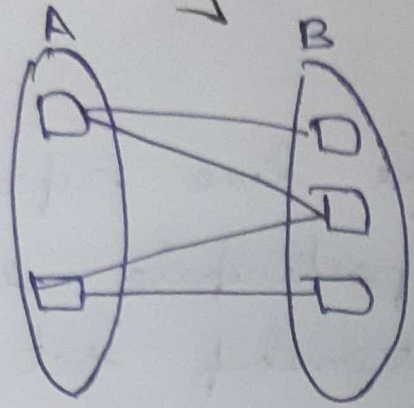
2) One to many :-



3) Many to One :-

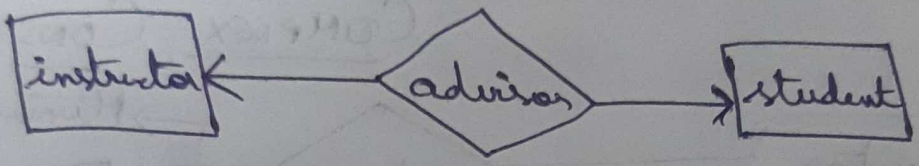


4) Many to many :-

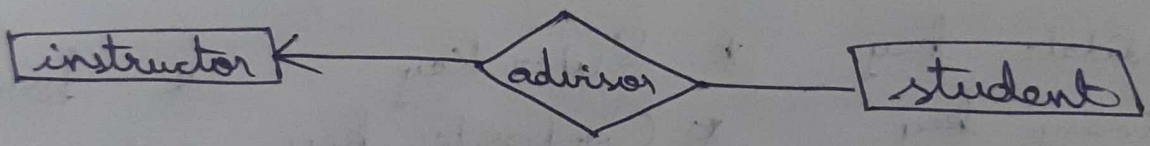


ER DIAGRAM :-

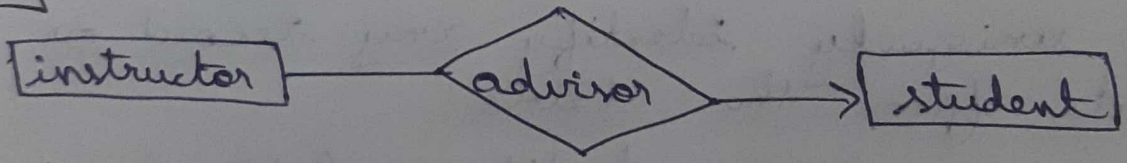
1) One - to - one :- We will have <=>.



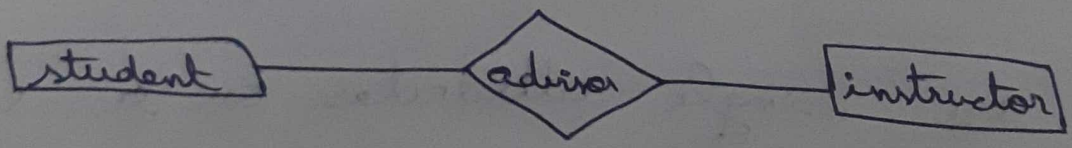
2) One - to - many :- We will have simple line on many portion.



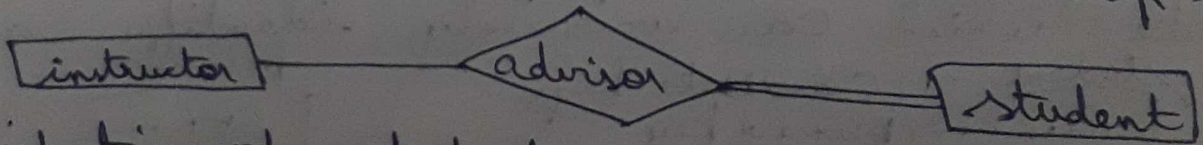
3) Many to one :-



4) Many to many :-



Total participation :- indicated by double line, every entity in the entity set participates in atleast one relationship in the relationship set.



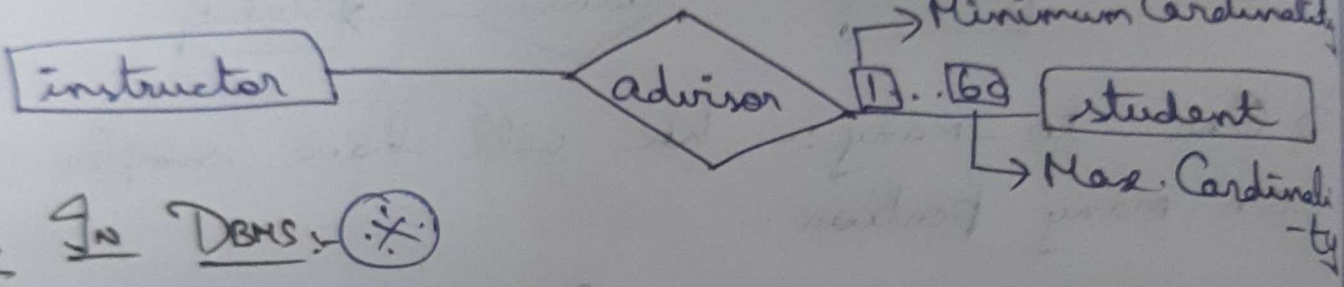
participation of student in advisor relation is total.

* every student must have an associated instructor.

Partial Participation :- Single line representation some entities may not participate in any relationship in the relationship set.

Minimum & Maximum Cardinality :-

↳ NOTATION FOR EXPRESSING MORE COMPLEX CONSTRAINTS



KEYS IN DBMS :-

* Candidate key
* Super Key
* Foreign key
* Primary key
* Composite key

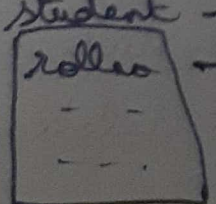
→ to uniquely identify any record or row of data from the table.

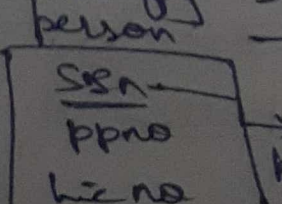
→ to establish & identify relationships between tables.

→ keys are single attributes or a set of attributes.

CANDIDATE KEY :-

is an attribute / set of an attribute which can uniquely identify a tuple



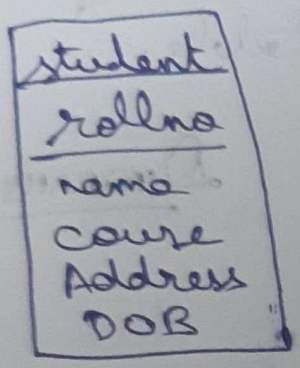


PRIMARY KEY:- Used to identify one & only one instance of an entity uniquely.

SUPER KEY:-

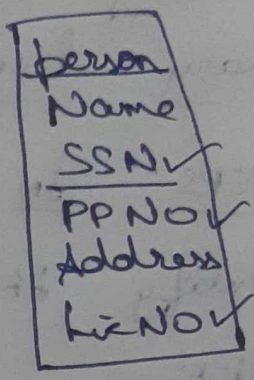
→ A set or of an attribute which can uniquely identify a tuple.

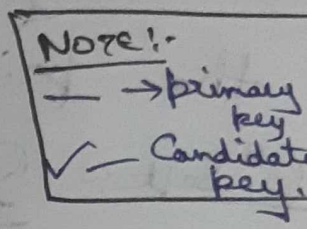
→ A superset of a candidate key.



(RollNO, Course) → SK1
(RollNO, DOB) → SK2

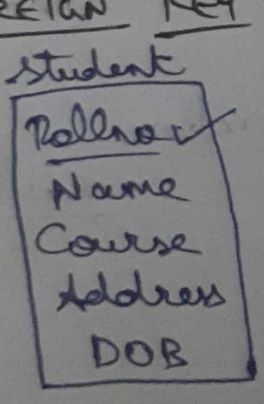
(PPNO, address) – SK1
(LicNO, Name) – SK2
(PPNO, LICNO) – SK2

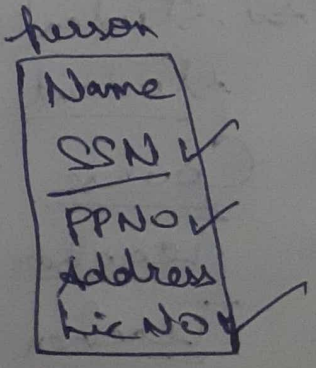




⇒ super key should have the candidate key together with some other key. Does not need to be primary key all the time.

FOREIGN KEY:-

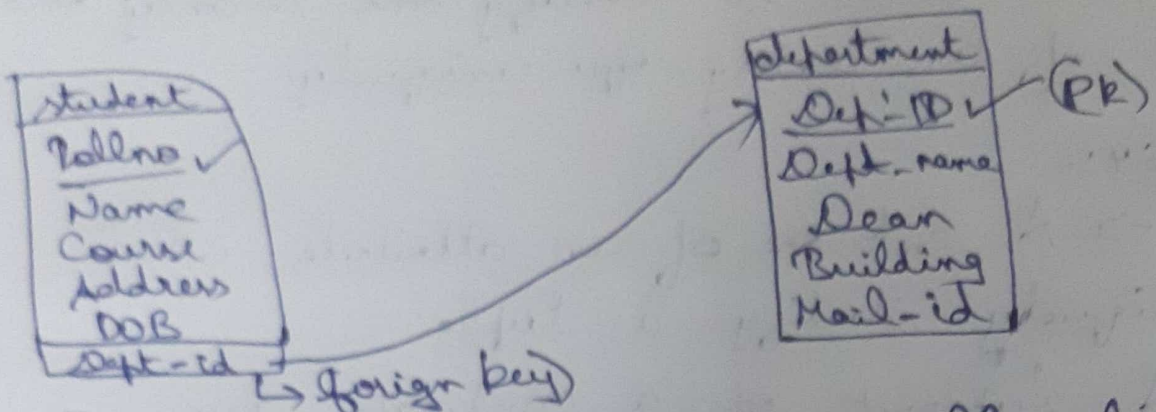




We can have connection between 2 tables in DBMS; only if we have the primary key in one table connected with the other table.

⇒ Foreign key is a column / group of columns in a table that provides a

link between data in 2 tables.



⇒ Foreign key of a table will always be the primary key of another table.

13th Sept :- STEPS IN CONSTRUCTING ER DIAGRAMS

⇒ ~~Into~~ Identify the

* Entities
* Attributes & primary key
* Relationships
* Cardinalities.

⇒ Construct a story. (Eg :- College Management system)

Eg :- ~~*~~ student attends courses

* student is taught by teachers
* Course has subjects
* Teacher teaches course
* Teacher handles subjects

entities

Entities :- student, course, teacher, subjects (4)

Attributes :- student (stu_no) PK. (stu_name) (stu_dob, stu_age, st_address, stu_phno)

Course (C_id, C_name, C_duration,)

teacher (t_id, t_name, t_qualification t_phno)

subje (s_no, s_type)

Cardinalities :-

* Student — (M) attends (1) — Courses

* Student — (M) taught by (N) — teachers

⇒ M : N relationship because student is taught by many teachers or one teacher teaches many students.

* Course — (1) has (M) — subjects

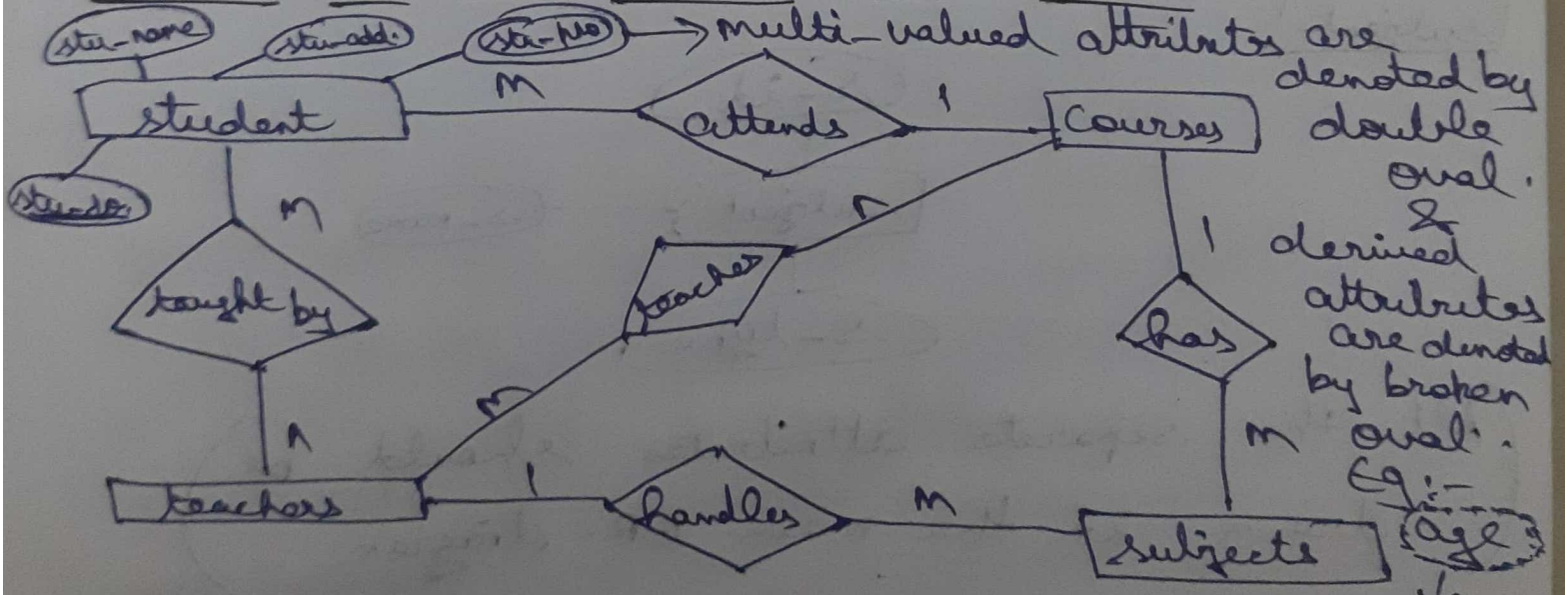
* teacher — (1) teaches (M) — Course

⇒ M : N relationship

* teacher — (1) handles (M) — subject

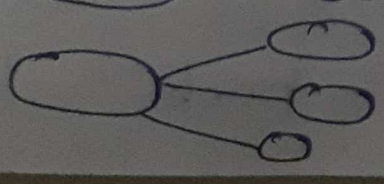
⇒ 1 : N/M relationship

REDUCTION OF ER DIAGRAMS TO TABLES :-



ATTRIBUTES :-

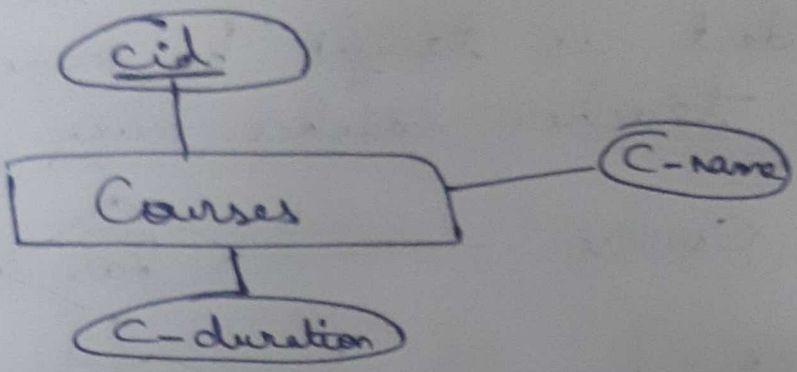
* ⬭ — oval (normal attribute)

*  — Composite attributes

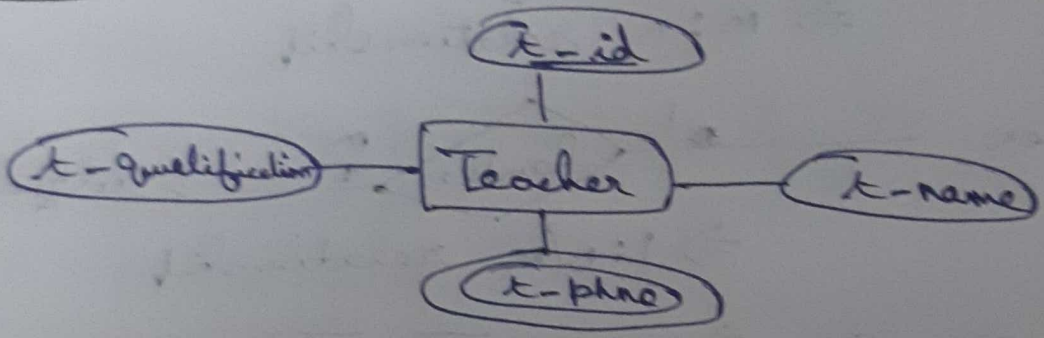
*  → multi valued attribute

*  → derived attribute

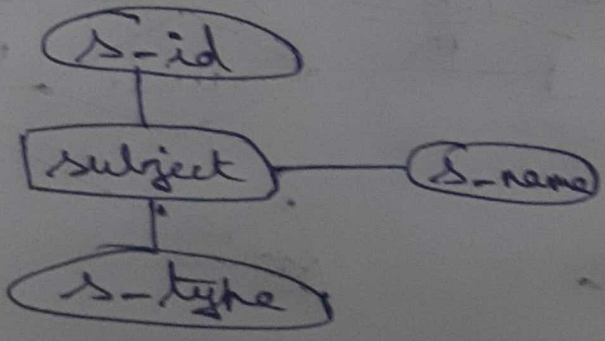
Courses



Teacher



Subject



(All the separate attributes should be drawn in the main ER-diagram)

H.W:- 1) Draw an E-R diagram for a library management system.

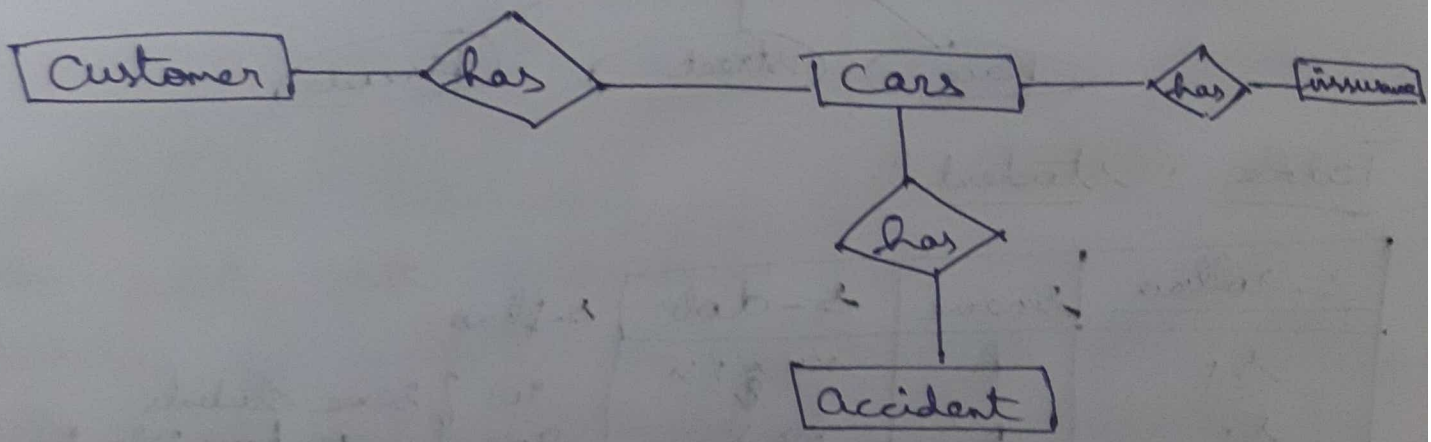
* student/reader issues a book
* librarian arranges book.

* books published by publisher (p_id, p_name, p_address, p_url)
* teacher issues a book (b_id, b_name, b_category, b_price, b_isbn)
* Books written by author

student, teacher, book, librarian, publisher, author ⇒ Entities

2) Draw an E-R diagram for a Car Insurance Company.

* Customer 1 — has — n Cars
* ~~Customer~~ Car 1 — ~~has~~ ~~has~~ 1 — insurance
* Car — m has — n accident



REDUCTION OF ER DIAGRAMS TO TABLES :-

→ Based on the ER diagram, we get
* The no. of tables
* primary key, foreign key
* attributes.

STRONG ENTITY :-

An entity where a primary key is easily identifiable is called as a strong entity.

For strong entities:-

1) ★ Entity becomes a table.
2) All single value attributes becomes a column for the table.
3) A key attribute of the entity type represented by the primary key.

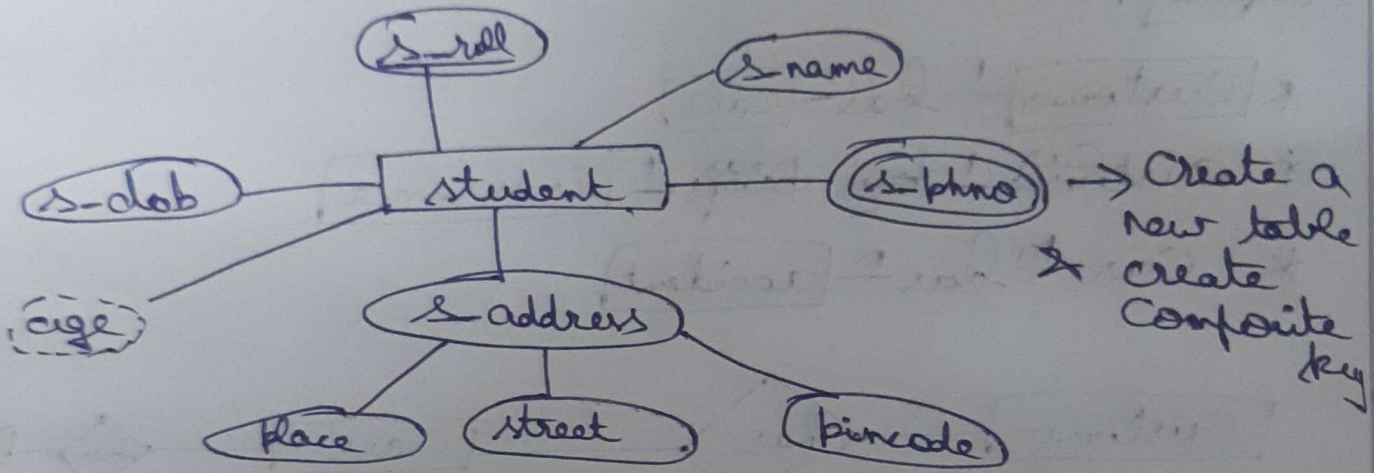


Table : student :-

s_rollno	sname	s-dob
s1	A	xxxxxx
s1	A	xxx

s_phno
901 } same students
902 } can have xle phno.

4) The multi-valued attributes is represented by a separate table - Create a Composite key.
Table: student_phno.

stu_rollno	phno.
s1	901
s1	902
s2	301

Include the primary key of the parent table student. Here, the primary key is

the combination of the primary key of the parent table along with the newly included column. That is also called as composite key.

Primary key(new table) = Primary key(old) + new field (multi-valued field)

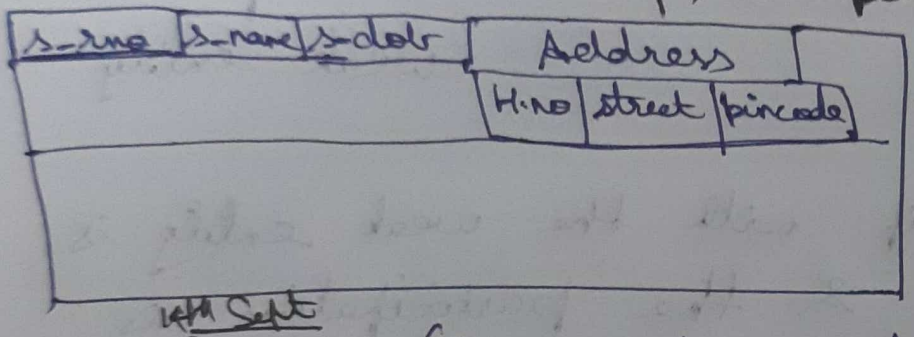
⇓

Composite key.

5) Composite attribute represented by components.

6) Derived attributes are not considered in the table. (They are not included in the table).

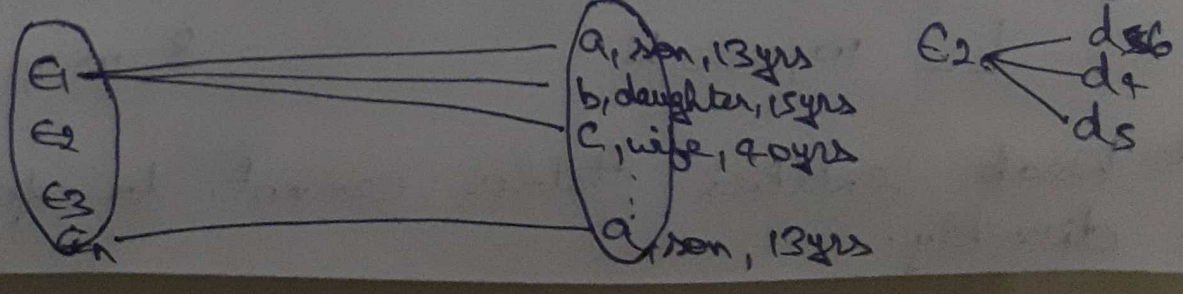
Table : student



WEAK ENTITIES :- (Represented in double rectangle entity)

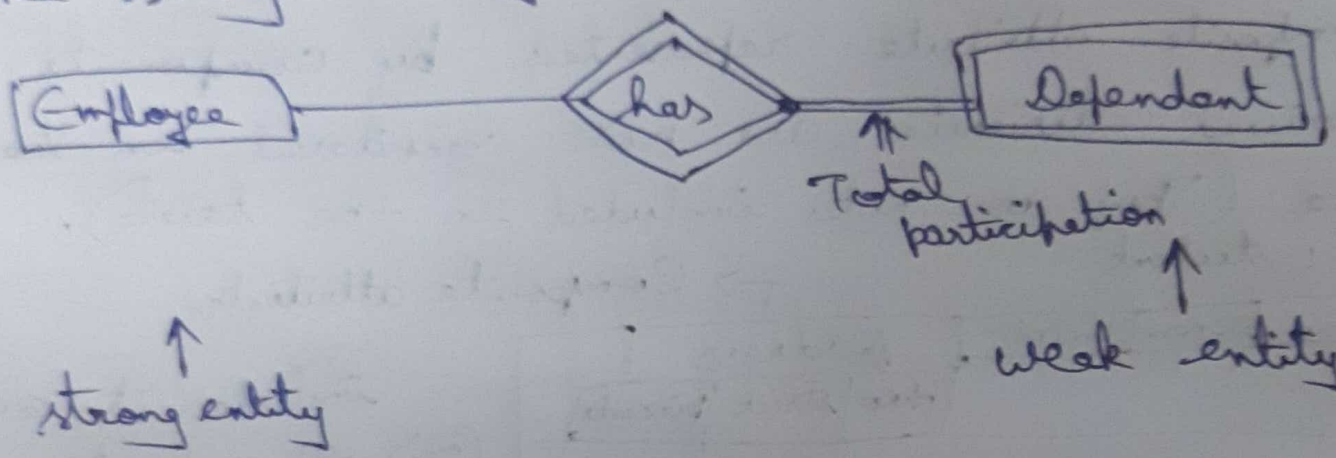
* An entity set that does not have sufficient attributes for unique identification / to form a pkey.
* In weak entity, primary key is difficult to find. Whereas, in strong entity it is easy to identify the primary key.
* There is a concept called dependent in weak entities.

Eg:- Employee has dependents.



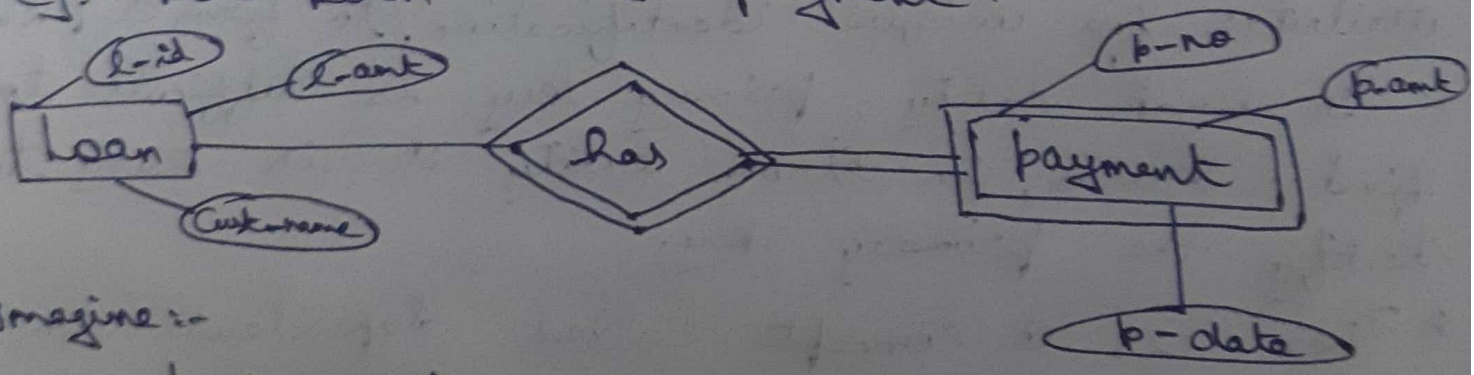
Having a dependent id, is not very sensible. In such cases, we won't you be able to give a proper primary key. We call such entity as weak entity.

* Weak entity should always be connected to the strong entity.



* The relationship with the weak entity is weak relationship & the participation is always total participation.

Eg:- ~~Loan~~ Loan has payment.



Imagine:-

L_1	30,000	A
L_2	40,000	A
L_3	20,000	A
L_4	10,000	B

1,	1000,	13/9/21
3,	2000,	13/9/21
3,	1000,	13/9/21
3,	100,	13/9/21

* Weak entities tables cannot be formed directly.

* The weak entity primary key is denoted by broken lines. ⇒ Discriminator.

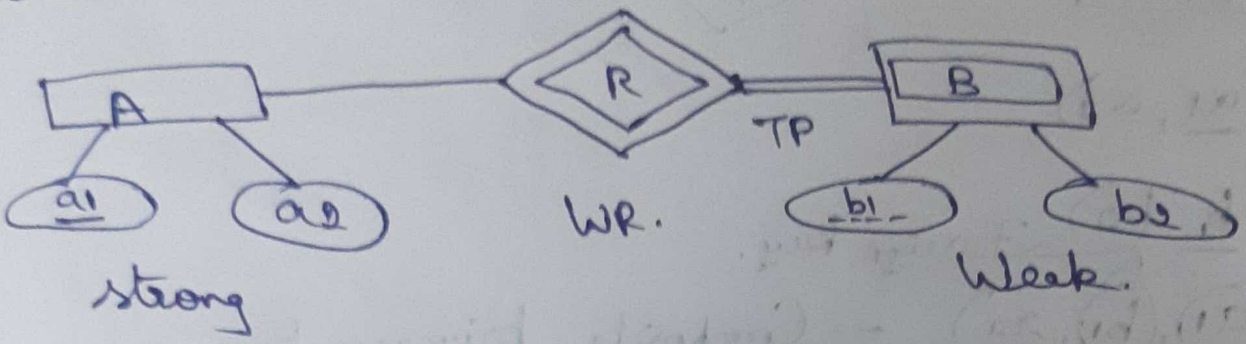


Table Construction :-

1) One table A – (a_1, a_2) (a_1 = Primary key)

2) Two table BR – (b_1, b_2) + (a_1)

BR ⇒ (a_1, b_1, b_2) (a_1 = Foreign key)

(a_1b_1 = PK (Composite key))

Eg:- Loan (l-id, l-amt, l-cust)

Payment ((l-id), p-no, p-amt, p-date)

H.W :- Find out the difference between strong & weak entity.

* A relationship set will require one table in the relational model.

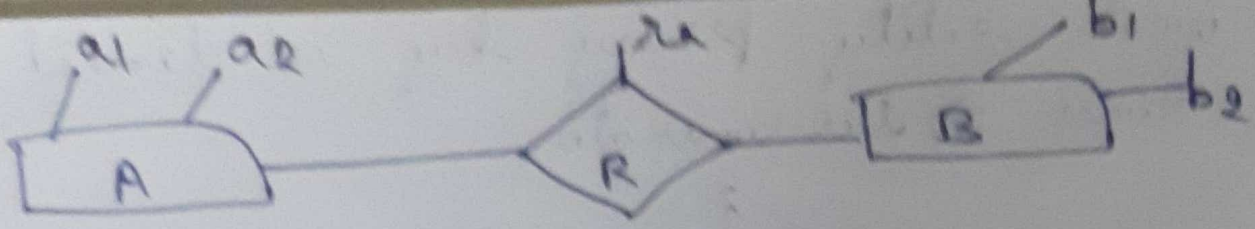
Eg:- Employee (Em-no . . .)

Dept (Dept-id, . . .)

Relation-ship ⇐ Works (Empno, Dept-id, since)

FK FK

Primary key ⇓ Composite key



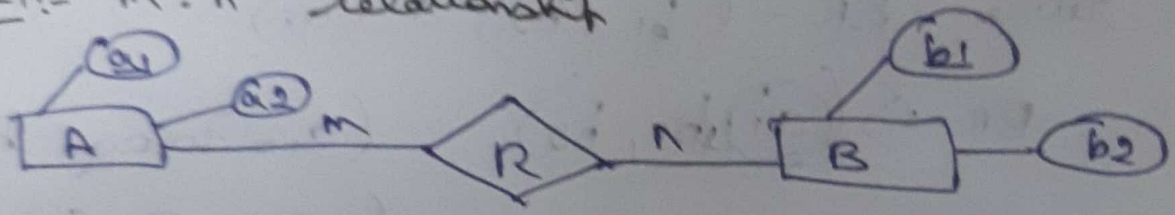
1) A (a1, a2)

2) B (b1, b2) → Foreign key.

3) R ((a1), (b1), a) — Composite primary key

BINARY RELATIONSHIPS:-

Case-1:- m : n relationship

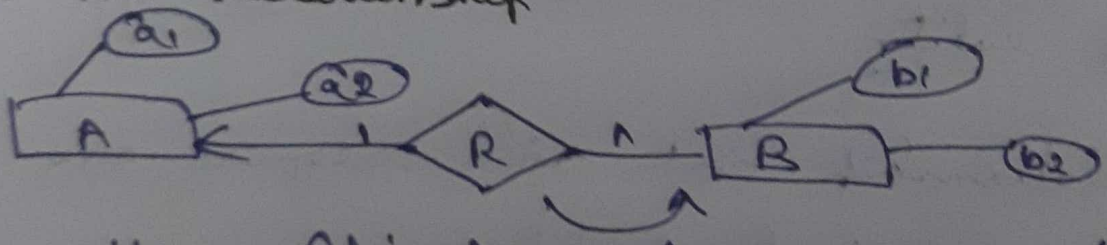


1) A (a1, a2)

2) B (b1, b2)

3) R ((a1), (b1)) — Composite primary key.

Case-2 :- 1 : n relationship

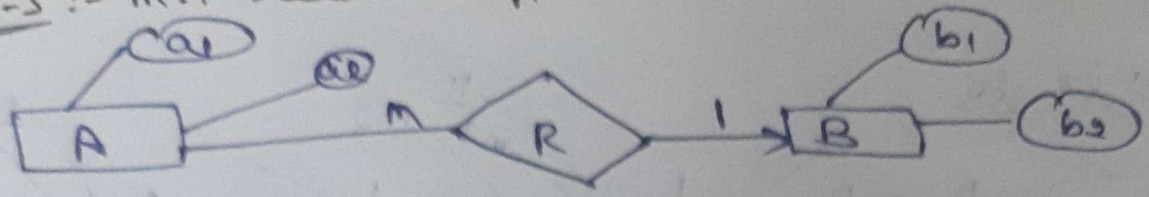


☞ Here, the relationship always move to n part.

1) A (a1, ~~a2~~)

2) BR (b1, b2, (a1)) ⇒ Since, relationship moves towards B.
FK

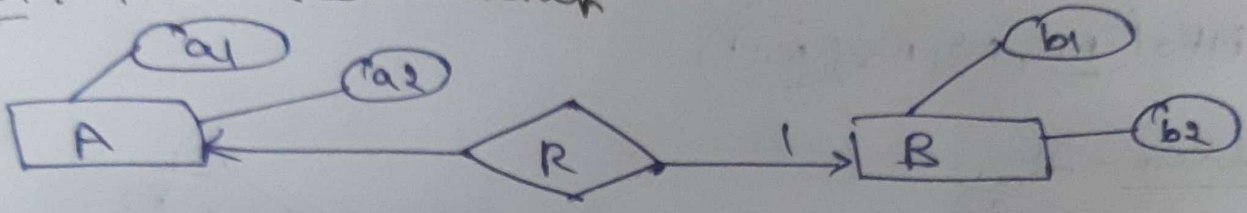
Case-3 :- m:1 relationship



1) AR($\underline{a_1}$, a_2, b_1)

2) B($\underline{b_1}$, b_2)

Case-4 :- 1:1 relationship



Way 1

1) AR($\underline{a_1}$, a_2, b_1)

2) B($\underline{b_1}$, b_2)

Way 2

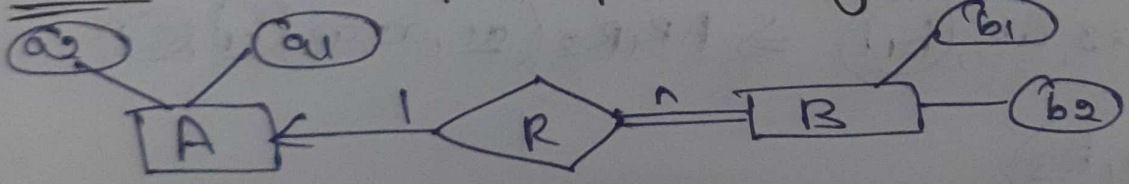
1) A($\underline{a_1}$, a_2)

2) BR($\underline{b_1}$, b_2, a_1)

TOTAL PARTICIPATION :-

Case-1 :- Total participation from one side

↳ same for 1:1, 1:n, n:1 m:n

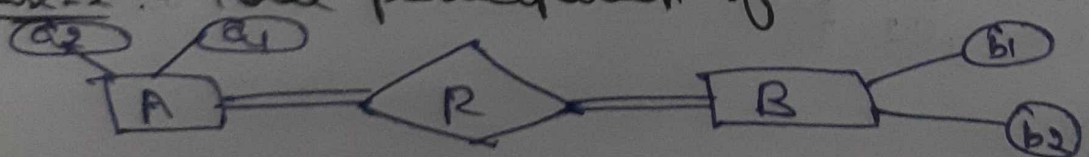


$a_1 \neq a_1 \neq$ null

1) A($\underline{a_1}$, a_2)

2) BR($\underline{b_1}$, b_2, a_1)

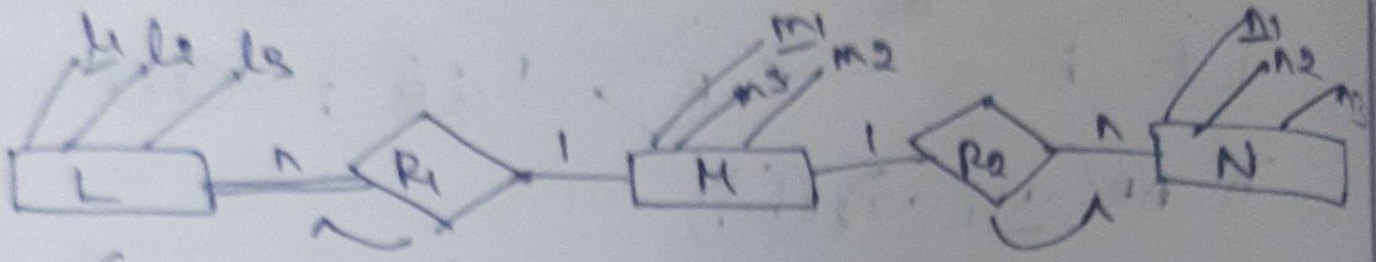
Case-2 :- Total participation from both sides.



ARB($\underline{a_1}$, a_2, $\underline{b_1}$, b_2)

↳ Composite primary key

Problem -1:-

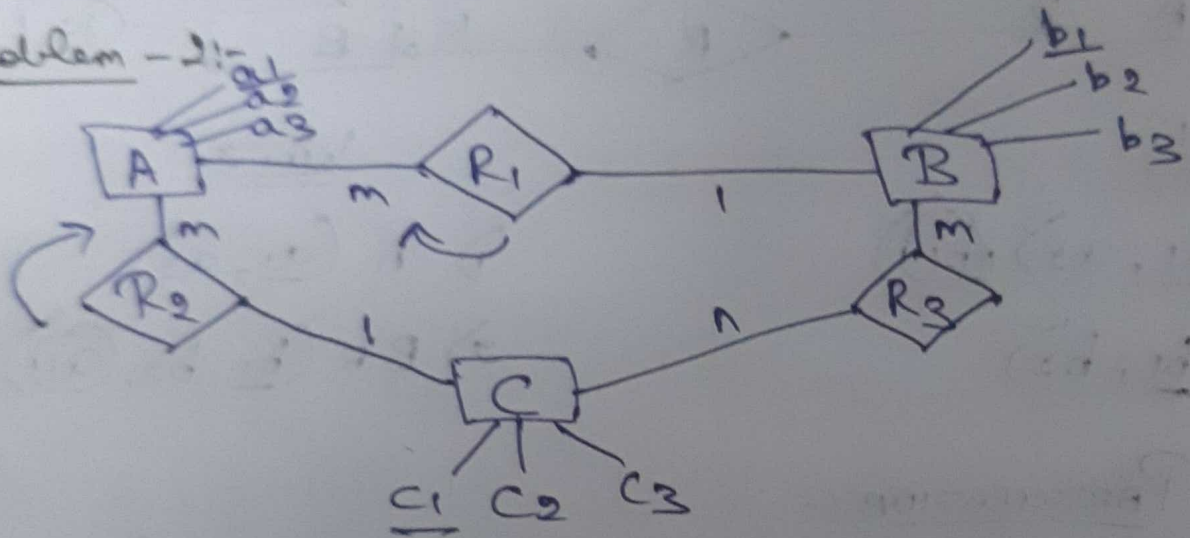


1) $LR_1(\underline{l_1}, l_2, l_3, (m_1))$

2) $M(\underline{m_1}, m_2, m_3)$

3) $NR_2(\underline{n_1}, n_2, n_3, (m_1))$

Problem -2:-



1) $AR_1(\underline{a_1}, a_2, a_3, (b_1)) \Rightarrow AR_1R_2(\underline{a_1}, a_2, a_3, (b_1), (c_1))$

2) $B(\underline{b_1}, b_2, b_3)$

~~3) $R_1(a_1, a_2, a_3)$~~

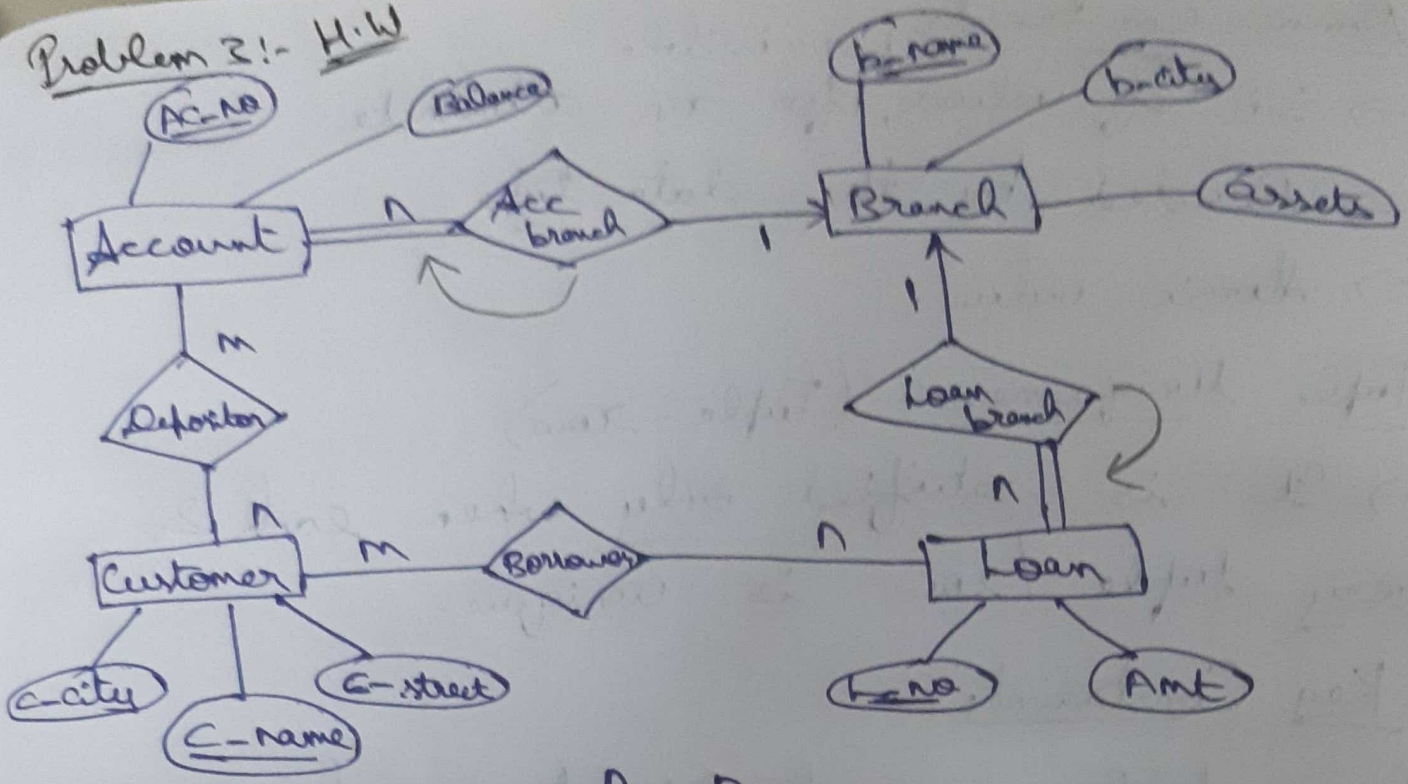
~~3)~~ $B(\underline{b_1}, b_2, b_3)$ [Can be ignored since already written]

~~4)~~ $C(\underline{c_1}, c_2, c_3)$

5) $R_3((b_1), (c_1))$

6) $AR_2(\underline{a_1}, a_2, a_3, (c_1))$ [should be added to the first table]

Problem 3:- H.W



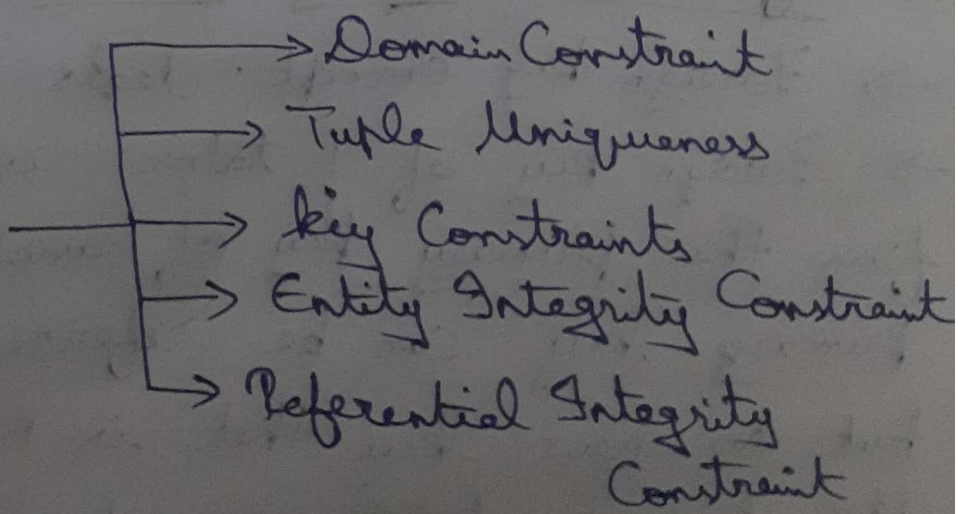
A - B

B - L

L - C

C - A

RELATIONAL CONSTRAINTS :-



1) Domain Constraints:- permissible
→ Set of values, for an attribute.
→ Denoted by the data type.
→ Atomic value.

2) Tuple Uniqueness:- (Tuple - row)
→ It is satisfied only when each & every tuple / row is unique.

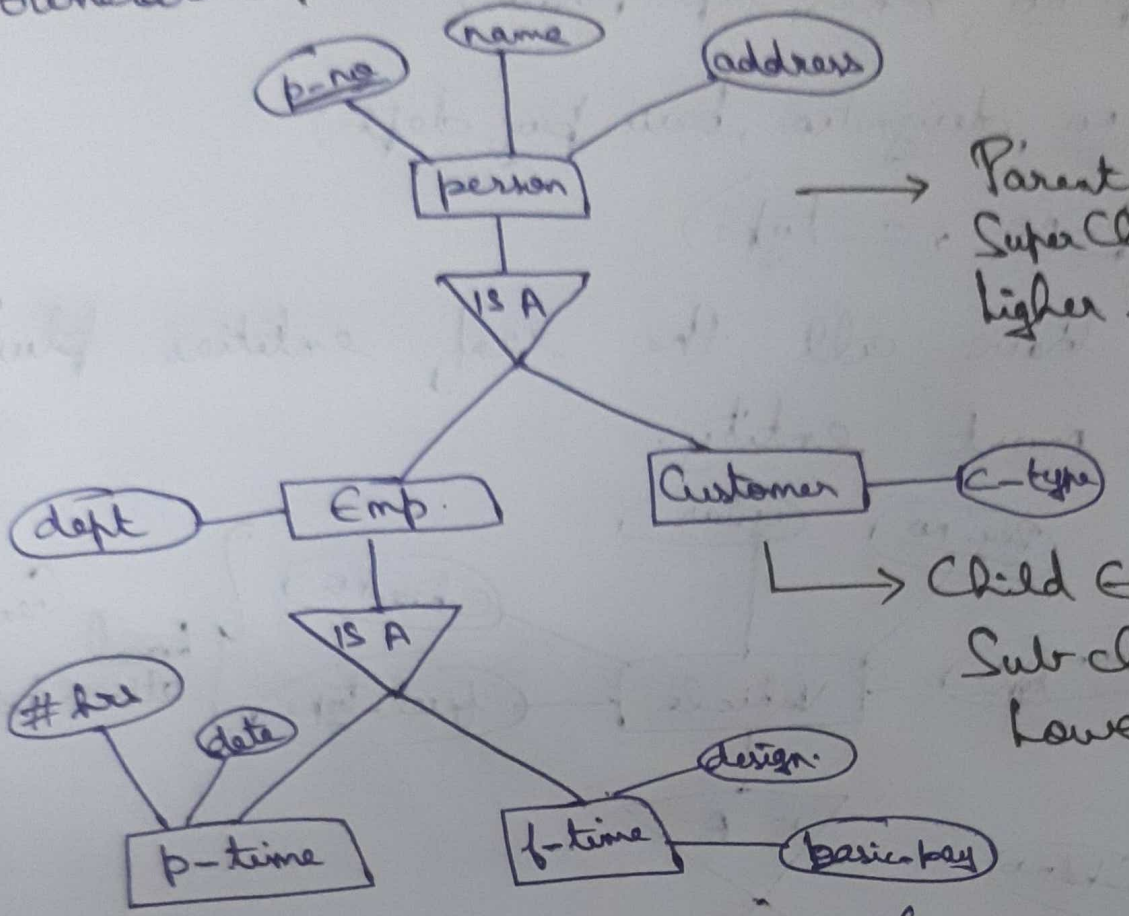
3) Key Constraints:-
→ In a table, we should always have a primary key or composite key (which is a combination of more than 1 key).
→ Primary key should always be not null. It cannot be null.

4) Entity Integrity Constraint:-
→ No attribute of the primary key can be null.

5) Referential Integrity Constraint:-
→ Foreign key concept. deals with
→ Foreign key of 1 table is to be referred from the primary key of another table. Same primary key can be referred 1le times but no new key that is not present in the parent table can be referenced.

EXTENDED E-R FEATURES :- ②

① Generalization & Specialization.



→ Parent Entity / Super Class entity / higher level Entity

└→ Child Entity / Sub-class entity / lower level entity.

Specialisation :- Top to bottom. (Top down approach)

Generalisation :- Bottom up approach.

leaf entity :- Entities which don't have any more sub entities attached to it.

Reduction to tables :-

* Tables are possible only for the leaf entities.

ptime (p-no, name, address, dept, hrs, date)

ftime (fno, name, address, dept, designation, basic pay)

customer (fno, name, address, c-type).

~~p-time~~

~~ftime~~

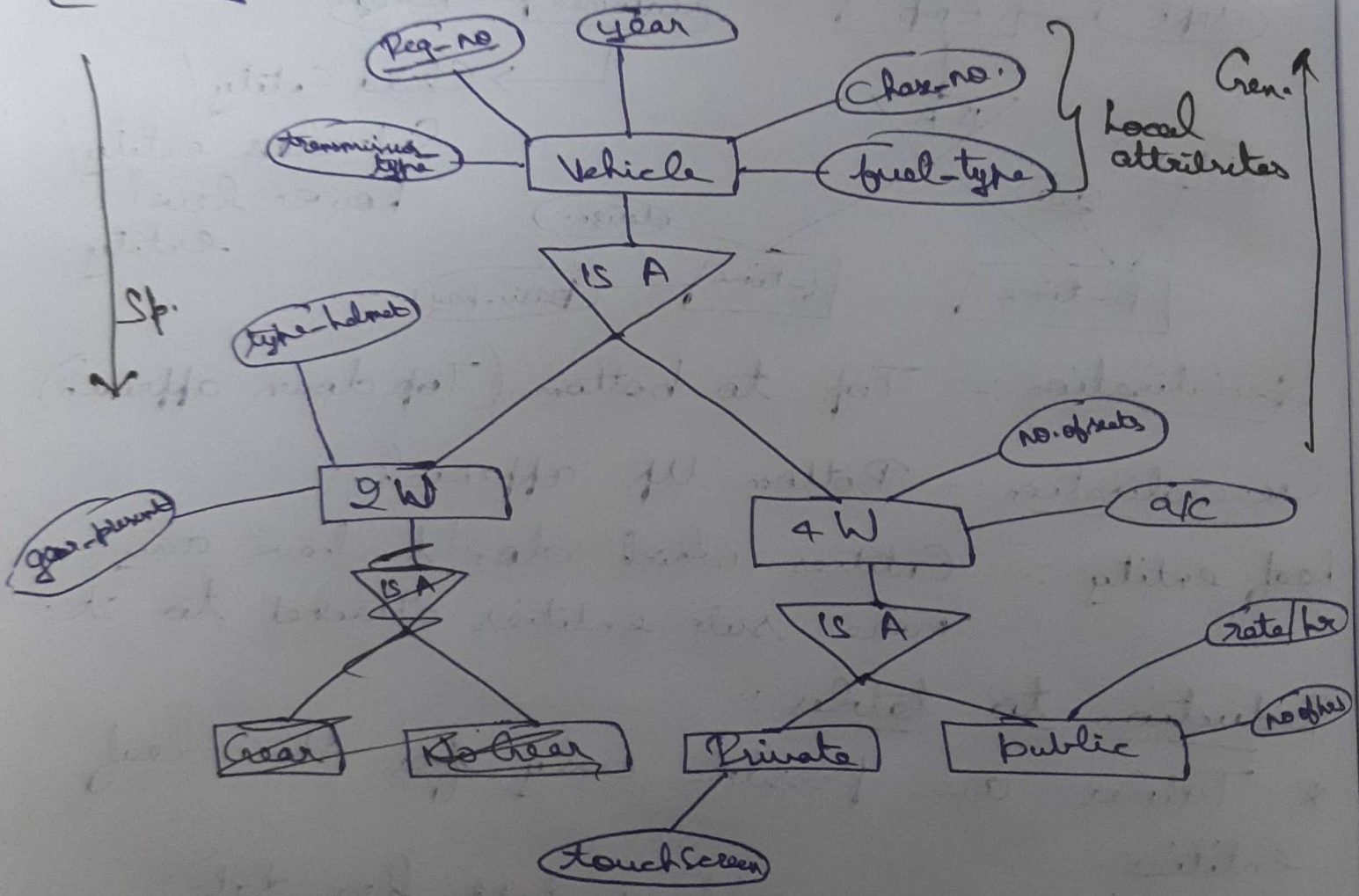
Person (p-no, name, address)

ptime (p-no, #hrs, dept, date)

ftime (p-no, designation, basic pay, dept)

Customer (p-no, c-type)

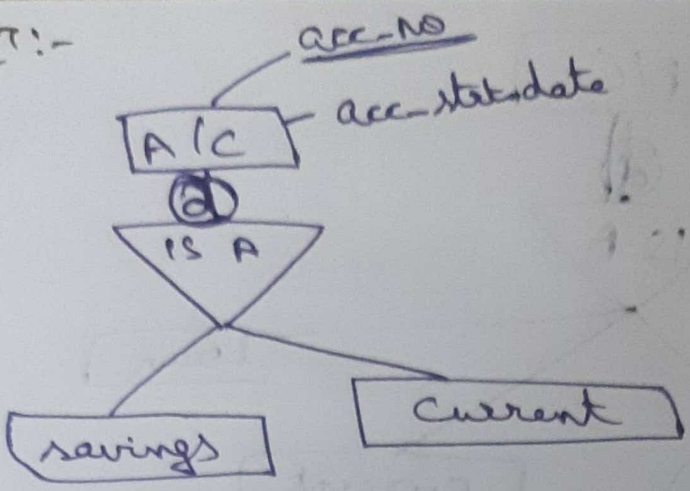
[We will have all the leaf entities plus the higher most entity.]

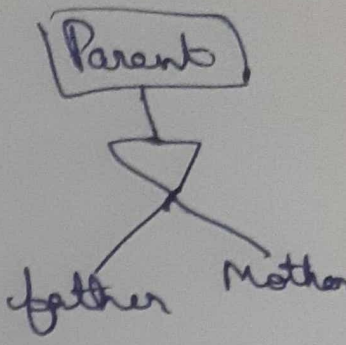


Constraints :- which occurs in specialisation & gen.

1) Disjoint
2) Overlapping
3) Total } Completeness constraints
4) Partial

1) DISJOINT:-

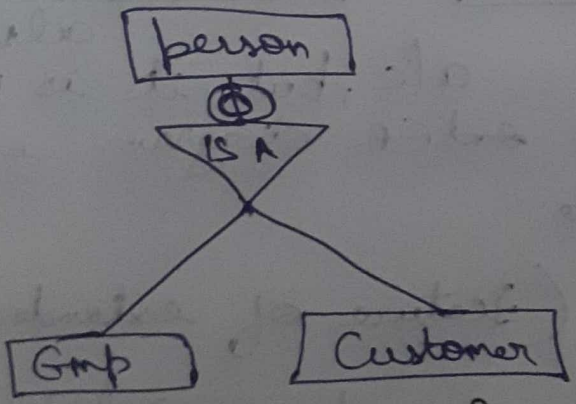




An account cannot be savings & current at the same pt. of time. all of
ie, a high level entity cannot be ~~1 or more~~ the low level entities at the same pt. of time. Represented by (d) at the connection line. Higher level entity can only be one of the low level entity

2) Overlapping:-

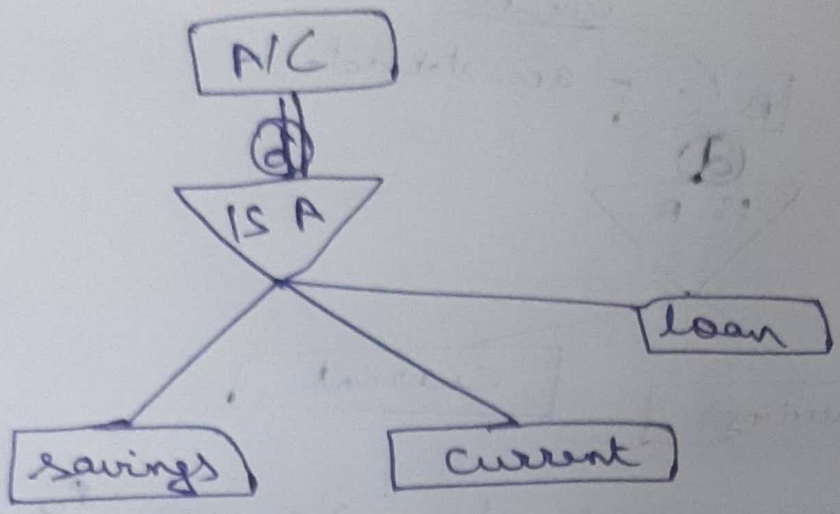
A higher level entity can be more than 1 lower level entity at the same point of time. Represented by (o) at the connection line.



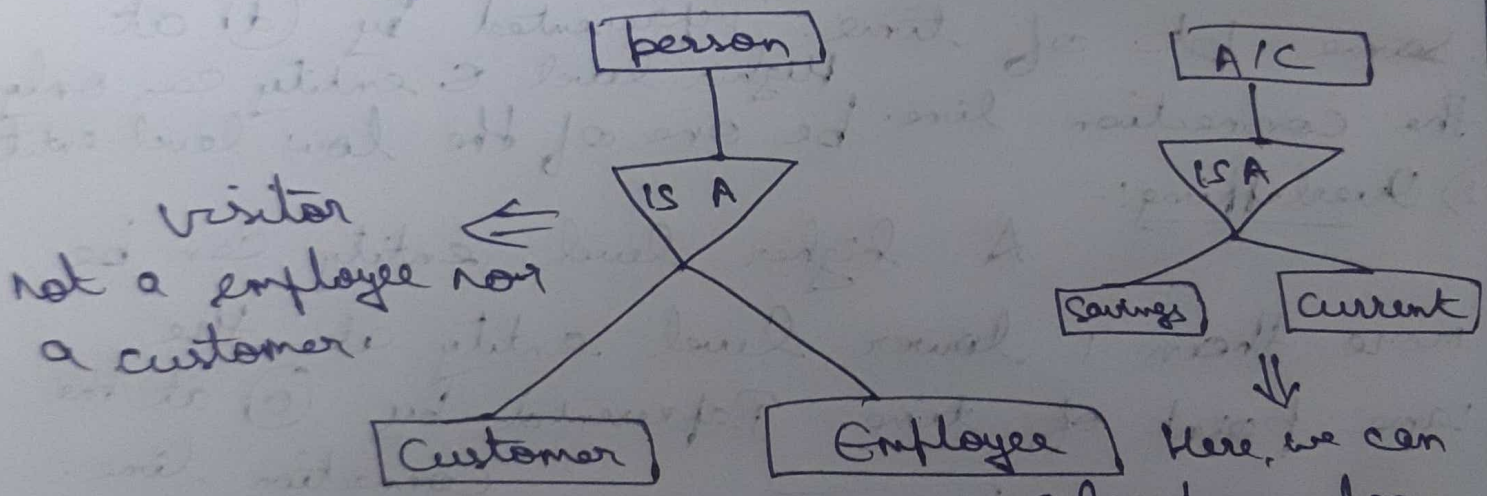
3) Completeness Constraint in Sp. & Gen:-

i) Total participation:-

The higher level entity should belong to any one lower level entity. Denoted by ~~double~~ double connection line.



ii) Partial:- If a higher level entity does not belong to any of the lower level entities, then we say partial participation.



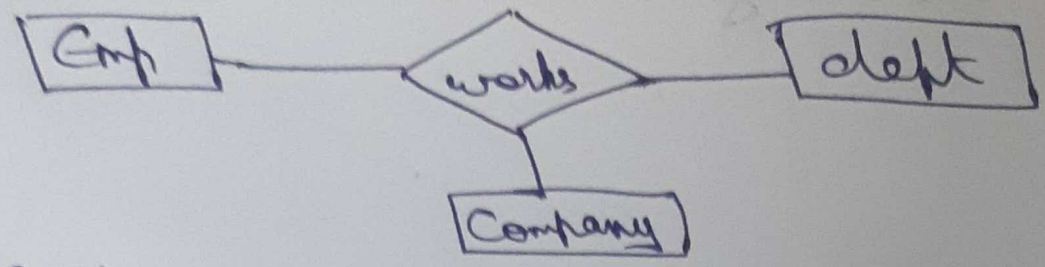
That is, the ~~ext~~ ER diagram exhibits partial participation.

3) AGGREGATION:- (Feature of extended ER diagrams)

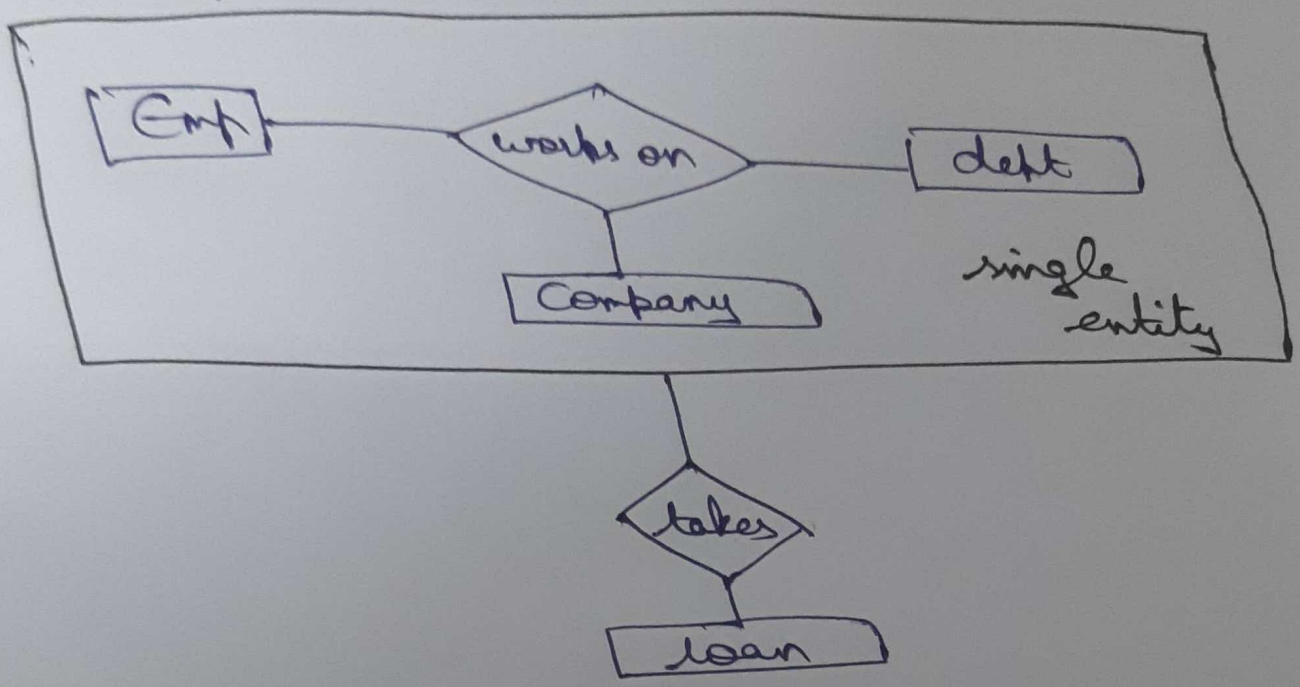
→ Rises when degree of a relationship > 2.

→ An abstraction where higher level relationships are treated as entities.

Eg:-) Employees work in a dept of a company.



ii) Employee takes loan.



Here, the grouping of the entity with relationship it to a single entity is called aggregation.

→ To also eliminate redundant relationship we make use of the concept called aggregation.

→ Grouping of relationships & entities together as a single entity is called aggregation. when the relationship is greater than 2 so as to eliminate redundant relationship is called aggregation.

What are the design issues in ER Modelling?

Unit:- DATATYPES IN SQL (LAB)

→ char → small int → find out?
→ varchar → date
→ number/integers
number(10) → to store whole number.
integer(3,2) → to store decimals;
↳ decimal pt.

DDL:- (Data Definition Lang.)
→ Deals with the structure of a table.
* Create * Alter

Queries:-

Create table <tablename>
(attr1 datatype;
attr2 datatype;
);

Create table student
(std-id varchar2(5),
stu-name varchar2(10),
stu-age number(4));

} Copy in notepad & Click right click in notepad.

Description of a table - desc.

desc student;

select / see the content → select query.

select select <attr1, attr2 ...> from
from <tablename>
wher <condition>; // optional.

Insert data into table:-

insert into <tablename> values (value1, value2...);

insert into student values ('S1', 'Anett', 21);

insert into student values ('S2', 'Akhil', 22);

insert into student values ('S3', 'Atul', 23);

select * from student; (Content of the table)

select stu-id from student;

select stu-name, age from student;

select * from student where age > 22;

sllect name from student where stu-id = 'S1';

Delete row in a table:-

delete from <tablename>
where <condition>;

delete from student where stu-name = 'Anett';

select * from student;

Delete all the rows in the table:-

delete from <tablename>;

delete from student;

Delete all rows + together with structure of the table:-

drop table <tablename>; drop table student;

select * from student ; // table does not exist

desc student ; // does not exist. (Exit → Query to exit sql)

Ques:- Create a student table,

1) Insert 3 rows
2) display the student who are of 20yrs of age.
3) delete 'Sidharth' from student
4) Show the structure of the table.
5) delete students with rollno = '1'.
6) drop the ~~tot~~ table.

4) Create table student
(std_id varchar(5),
stu_name varchar(10);
stu_age number(4));

1) insert into student values ('S1', 'Anu', 20);
insert into student values ('S2', 'Jeevan', 23);
insert into student values ('S3', 'Sidharth', 21);

2) select * from student
where stu_age = 20;

3) delete from student
where stu_name = 'Sidharth';

(or) delete from student where stu_name like 'Sidharth';

4) desc student;

5) delete from student
where stu_rollno = '51';

6) drop table student;

CONSTRAINTS IN SQL:-

1) Check:-

→ Have to specify while the creation itself.
* like, in → in case of choice.

2) Not null:-

→ Values of the attributes cannot be null.

3) Default:-

→ Default values will be taken.

Queries:-

1) Enforce check constraint:-

Eg:- t-id → start with 'T'

Create table tchr
(t-id varchar(10) check (t-id like 'T%'),
t_name varchar(10),
age ~~integer~~ number(4));

Eg:- T101, 745, Tf45

Eg:- Create table tchr
(t-id varchar(10) check (t-id like('T%')),
t_gender varchar(4) check (t-gen in('M','F','T')),
t_name varchar(10), age number(4));

insert into tchr values ('T101', 'F', 'Alphy', 23);

2) Enforcing not null:-

* Create table tchr
(t_id varchar(10) check (t_id like 'T%'),
t_gender varchar(2) check (t_gender in ('M', 'F', 'TG'))
t_name varchar(10),
age number(4) not null);

insert into tchr
values ('T101', 'M', 'Jeffy', null); // throws error because of not null constraint.

* create table tchr
(t_id varchar(10) not null check...)
t_gender varchar(2) not null check...,
t_name varchar(10) not null,
age number(4) not null);

Ques:- Create a student table with the foll. constraints-

stu_id, stu_name, gender, address

* should start with MCA.
gender ('M', 'F', 'TG')
address - optional

~~Creti~~ Query:-

```
create table student
(stu-id varchar(6) not null check (stu-id like 'MCA%'),
stu-name varchar(20) not null;
stu-gender varchar(3) not null check (stu-gender
IN('M','F','TG'),
stu-address varchar(30));

insert into student
values('MCA101','Anu','F','Plot 28 th Avenue,NY');
insert into student
values('MCA201','Jeevan','M');
```

3) Enforce default :

```
Create table stu
(s-id varchar(20) not null check (s-id like 'MCA%'),
s-name varchar(20) not null;
s-course varchar(10) default 'MCA',
age number(4));
insert into stu
values('M101','Ann','Msc',25);
insert into stu(s-id,s-name,s-course,age)
values('M101','Akhil','Msc',25);  -> use this method while using default
insert into stu
values('M103','Jeevan',25); //do not use
```

Primary key Constraint :- (2 marks) → By default, it will be a not null & unique

→ All tables should have a primary key.

```
create table stu
(s_id varchar(10), primary key check (s_id like 'H%'),
 s_name varchar(20) not null,
 s_cours varchar(10) default 'MCA',
 age number(4));
```

Foreign key :- Let student takes a book.

```
Create table book
(b_id varchar(10) primary key check (b_id like 'B%'),
 b_name varchar(10) not null,
 s_id varchar(10) foreign key references stu(s_id));
```

s_id → datatype should be same

```
s_id varchar(10),
foreign key(s_id) references stu(s_id));
```

```
insert into book
values('B101', 'DBMS', 'M101');
```

H.W

stu - 5 entries

book - 7 entries

select * from tab; // to see all the tables available in the oracle.

Rename the table:

rename stu to student;

rename <oldtablename> to <new tablename>

desc stu; // error;

desc student;

Alter - many variants:-

1) Alter column name.

alter table <tablename>
rename column age to stu-age;

desc student;

Rename should be used along with alter to change the column name

2) To add new column.

alter table <tablename>
add (s-gender varchar(3));

desc student;

3) Remove an existing column:-

alter table <tablename>
drop column stu-age;

desc student;

4) Change/Modify the column description:-

alter table <tablename>
modify (s-gender varchar(10));

desc student;

⇒ Update / Change the content of the existing rows in a table.

update <tablename>
set <field name> = <value>
where <Condition>;

```
update student
set s_gender = 'M'
where s_id = 'M101';
```

```
update student
set s_age = 23
where s_name = 'Ancy';
```

Updating all ages:-

```
update student
set s_age = s_age +10;
```

```
update student
set s_gender = 'Male';
```

Ques:-

1) Change arjun's course to MSc.

```
update student
set s_course = 'MSc'
where s_name = 'Arjun';
```

Ques:- (27th Sept)

What is the salary of Shyam.?

```
select t_salary
from teacher
where t_name = like 'Shyam';
```

(or)

```
select t_salary from teacher
where t_name = 'Shyam';
```

2) What is the dept. name of Shyam?

select dept_name, t_name
from dept, teacher

```
select dept.dept_name, teacher.tea_name
from dept, teacher
where teacher.dept_id = dept.dept_id
        and teacher.t_name = 'Shyam';
```

(or)

```
select d.dept_name, t.t_name
from dept as d, teacher as t  // from dept d, teacher t
where d.dept_id = t.dept_id and
        t.t_name = 'Shyam';
```

3) What is the budget of Geeta's dept?

```
select d.d_budget , t.t_name
from dept d , teacher t
where d.did = t.t_did and
        t.t_name = 'Geeta';
```

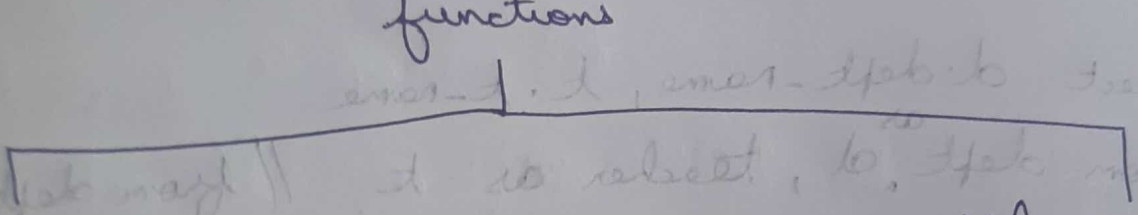
4) Display the dept_name, budget & salary of Gopi

```
select d.d_budget , d.d_name, t.t_salary
from dept d, teacher t
where d.d_id = t.t_did and
        t.t_name = 'Gopi';
```

5) Find the name of employees & their corresponding dept name for those who have salary greater than 500.

```
select d.dept_name, t.t_name
from dept d, teacher t
where d.did = t.t_did and t.t_salary > 500;
```

TYPES OF FUNCTION IN SQL :-



Single row functions
→ returns a single row

xle row function
→ ~~set~~ return set of rows
Eg :- Aggregate functions

Aggregate functions :-

* average - avg
* sum
* min
* max
* count - count the no of rows.

Aggregate functions will be returning only one result.

1) Find the avg. salary of teachers.

```
select avg(t_salary)
from teacher;
```

2) Find the min salary of a teacher.

```
select min(salary)
from teacher;
```

3) max salary - select max(salary) from teacher;

4) sum salary - select sum(salary) from teacher;

5) How many teachers are there in the teacher table?

```
select count(t_name)
from teacher;
```

6) What is the avg. budget?

```
select avg(d.d_budget)
from department d;
```

7) Find the name of teachers who have a salary between 300 & 800.

```
select t_name, t_salary
from teacher
where t_salary between 300 and 800;
```

8) Find the teachers whose name start with G.

```
select t_name
from teachers
where t_name = 'G%';
```

(or)

```
select t_name
from teachers
where t_name like 'G%';
```

↳ Matching string patterns.

* % → any no. of characters.

* _ (underscore) → exactly one character.

G% → starts with G.

%a → ends with a.

second character a → _a%;

second last character i → %i_;

third last character h → %h__;

9) a anywhere in the name.

```
select t_name
from teacher
where t_name like '%a%';
```

SET FUNCTIONS IN SQL:- (should be done using 2 select queries)

* union * union all
* intersect * minus

Union:- (duplication is avoided) & the result is sorted out

```
select * from fruits
union
select * from veggies;
```

```
select fname from fruits
union
select vname from veggies;
```

Intersect:- (common items)

```
select fname from fruits
intersect
select vname from veggies;
```

union all:-

```
select fname from fruits.
union all. // duplication is not avoided
select vname from veggies;
```

minus:- (elements present in one & not in another).

Find the id's of veggies whose vname is a null value.

```
select v_id
from veggies
where v_name is null;
```

Ordering of tuples :- (by default ascending order)

```
select v_name
from veggies
order by vname;

select vid, vname
from veggies
order by vname;
```

```
select vname
from veggies
order order by vname desc;
                ↳ descending order.

select vid, vname
from veggies
order by vid desc;
```

Display the name & salaries of teachers sorted on an ascending of their names.

```
select t_name, t_salary
from teacher
order by t_name;
```

NUMERIC FUNCTIONS IN SQL :-

* sqrt * div * degrees * ceil ceiling
* floor * cos/sin/tan * acos/asin/atan * abs
* greatest.

Dual Table :- only a single row - dummy - varchar2 - dummy table available only in oracle.

```
select sqrt(4) from dual;
```

desc dual;
select * from dual; // single field dummy ~~whd~~ with just one character ie, varchar(1)

dual is a dummy table which is used to work with all numeric & string functions in oracle. Available only in oracle.

degrees is a radians to degrees conversion

~~select 10 div 5 from dual;~~

Ceil - goes to the highest number.

select ceil(12.5) from dual; // 13 OP

floor - goes to the lowest number.

select floor(12.5) from dual; // 12 OP.

select cos(30) from dual;

select tan(30) from dual;

select sin(30) from dual;

select asin(.5) from dual;

select atan(.5) from dual;

select abs(-23.33) from dual; // Gives the absolute value.

~~sto~~ select sign(-23.33) from dual; // -1 OP

select sign(23.33) from dual; to get the sign.

select greatest(12,34,456) from dual; // gives the greatest of the nos.

~~select pi() from dual;~~

select mod(10,3) from dual; // returns the reminder in normal division

select power(2,3) from dual;

~~select radian~~ sle select ln(2) from dual; // ln - natural logarithm

select log10(2) from dual; ↳ // log to the base of 10

// (or)

select log(2,10) from dual; // base to be given in second position

select log(2,8) from dual;

select round(12.34) from dual; // to round the no. to the nearest integer.

select round(12.84) from dual;

whatever function we give in dual table, the func is displayed as the field name.

select round(12.84) "R" from dual;

↳ gives the field name as "R".

select mod(10,3) from dual;

⇒ select mod(10,3) "Remainder" from dual;

↳ gives the field name as "reminder".

select sqrt(82) from dual;

↳ sqrt of the no.

STRING FUNCTIONS IN SQL:-

1) length
2) wsize
3) lower
4) upper
5) initcap
6) substr
7) ascii
8) instr
9) translate

select length('Bottle') "LB" from dual;

select wsize('Bottle') "SIZE" from dual;

vsize → used in other databases. Here, in this oracle, both length & vsize gives the length of

select lower('Bottle') "Low"
from dual;

lower → all characters are displayed as lower characters

select upper('Bottle')
from dual;

upper → all characters are displayed as upper characters.

and initcap → works if we have more than one word. Initial character of each word is made capital in output.

select initcap('this bottle') "INIT" from dual;

O/P:- This Bottle

substring — to extract a substring from a particular string.

select substr('Participant', 2, 4) // start starting from 2nd character till 4 characters.
from dual;

O/P:- arti

select substr('Participant', 4, 2)
from dual;

O/P:- ti

ascii — get the ascii code of a character.

select ascii('a') from dual; // OP - 97

select ascii('A') from dual; // OP - 65.

instr — helps to find the first occurrence of a character / a substring from a long string.

select instr('Participant', 'a') from dual;

o/p :- 2

select instr ('This is a Toy', 'To')
from dual;

NOTE :-
clear screen - to clear the screen.

o/p :- 11

translate - similar to replace function in word. replaces the specific character to any other character in a specific string.

select translate ('Participant', 'a', '%*u')
from dual;

o/p :- ~~P%*uarticip%*unt~~ P%rticip%nt (since only one character is given to be replaced)

select translate ('Participant', 'ait', '%*u')
from dual;

o/p :- P%r%u*c*p%nu → [a replaced with %, i with *, t with u]

select translate ('Participant', 'art', '%*u')
from dual;

o/p :- P%*uicip%nu [art → %*u, t → u, r → *, a → %]

String trimming functions :-

1) LTRIM :- Left trim → helps in cutting the length of the string

select ltrim ('Meeting', 'M')
from dual; → Checks for the left most character

o/p :- eeting // If the 'M' is there cut the letter

select ltrim ('Meeting', 'e') from dual;

O/P:- Meeting.

RTRIM:- Right trim — checks the Rightmost character

select rtrim ('Meeting', 'e') from dual;

O/P:- Meeting

select rtrim ('Meeting', 'g') from dual;

O/P:- Meetin

TRIM:-

select trim (both 'x' from 'xxxxmeetingxxxx') from dual;
(both → keyword, from → keyword)

O/P:- meeting

select trim (leading 'x' from 'xxxmeetingxxx') from dual;
(leading → keyword (onto the left))

O/P:- meetingxxx

select trim (trailing 'x' from 'xxxmeetingxxx') from dual;
(trailing → keyword (onto the right))

O/P:- xxxmeeting

trimming cannot be done in middle.

PADDING FUNCTIONS:- stuffing

1) lpad - left padding

select lpad ('Screen', 10, '*') from dual;

O/P:- ****Screen

[Screen - 6 letters, extra 4 is padded with *]

2) rpad → right padding

select rpad('Screen',10,'*') from dual;

O/P:- Screen****

select rpad('Screen',10,'*&') from dual;

O/P:- Screen*&*&

select rpad('Screen',5,'*&') from dual;

O/P:- Scree // only 5 characters gets printed since the number given is 5

DATE FUNCTIONS :-

sysdate — system date.

select sysdate from dual;

O/P:- SYSDATE
04-Oct-21 (DD-MM-YY)

add_months —

select add_months(sysdate,4) from dual;

O/P:- ADD_MONTH
04-FEB-22

select add_months('01-JAN-90',4) from dual;
↳ our system date format

O/P:- 01-May-90

last_day - select last_day(sysdate) from dual;

O/P:- 31-Oct-21

select last_day('01-Jul-21') from dual;

O/P:- 31-Jul-21

Also, work with alias.
select last_day('01-Jul-21') "LD" from dual;

O/P:- LD
31-Jul-21

months_between-
select months_between('01-Jan-90',
'31-Jul-93')
from dual;

O/P:- -42.96... (-ve because 90 year is less than 93yr).

select months_between('01-Jan-93','31-Jul-90')
from dual;

O/P:- Months_between
29.0322581

next_day - takes 2 args.
select next_day(sysdate,'Monday') from dual;
(To find out the date of next monday).

O/P:- next_day
11-Oct-21

round - takes 2 args.
select round(to_date('9-Jul-95'),'MON')
from dual;

O/P:- 01-Jul-95.
<15 ⇒ round to lower month

select round (to_date ('18-Jul-95'), 'yyyy') from dual;

≥ 15 ⇒ round to higher year

O/P:-
01-Aug-95

16-Sept-2015,
01-Jan-2016

select round (to_date (22-Jul-95), 'yyyy') from dual;

O/P:- Round to
01-Jan-96

date-diff—

select datediff ('01-Jan-90', '18-Jul-95') from dual;

now curdate date
curtime datediff
→ not work in this sql. but there are date functions

extract — (year, month, date)

select extract (year from sysdate) "YR"
from dual.

→ only this work.

O/P:- YR
2021

[04-Oct-21]

select extract (day from sysdate) "MN"
from dual;

O/P:- MN
4

⇒ also same as month.

O/P:- MN
10

to-char → date to a specified date format.

select to_char (sysdate, 'dd-mm-yy') from dual;

O/P:- To_char
04-10-21

select to_char(sysdate, 'DAY') from dual;
⇒ also works

O/P:- To_char
Monday ⇒ also dd for month.

Tochar To_char
04 October

to_date (string func)

select to_date('01-Jan-2020', 'dd-mm-yy')
from dual;

O/P:- 01-Jan-20

Formatting In Date Functions :- [dd, mm, sp, th]

to_char:-

select to_char(sysdate, 'ddth') from dual;

O/P:- To_c
04th

select to_char(sysdate, "mmsp") from dual;

O/P:- To_char ⇒ also (Oct - ten) (th - spell)
ten ddspth

⇒ also work as mmspth.

O/P:- fourth.

O/P:- tenth

Subtables from an existing table :-

Ques:- The students in Kochi, store their details in a new table.

select sid, name, city from test_stu

where city = 'Kochi';

⇒ create table su_Kochi
as
select sid, name, city from test_stu
where city = 'Kochi';

Create new calculated fields in a table:
Assume, here the table consists of 3 marks. So, we want to calculate sum.
Add an extra col. "total" = m1+m2+m3

Display the marks with total.

select sid, name, m1, m2, m3 from test_stu;

Display the student details with the total marks.

select sid, name, m1, m2, m3, m1+m2+m3
as "total" from
test_stu;

Also, find avg.

select sid, name, m1, m2, m3, m1+m2+m3 as
total, (m1+m2+m3)/3 as avg
from test_stu;

⇒ Also, apply where condition after select, where city = 'Kochi'. Also we order by name; after where, order by name;

IN / NOT IN FUNCTIONS:

Display the details of students in Kochi or in Trum.

```
select * from test_stu
where city = 'Kochi' or city = 'Trum';
```

Display the same not in Kochi/Trum.

```
select * from test_stu where city
not in ('Kochi', 'Trum');
```

or in

```
select * from test_stu where city
in('Kochi', 'Trum');
```

Display the details of student whose m1 mark is not in 40.

```
select * from test_stu where
mark not in (40);
```

Ques: Find the students who have m1 marks between 50 & 70.

```
select name
from test_stu
where m1 between 50 and 70;
```

Find the name of students in sorted order.

```
select name from test_stu
where m1 between 50 & 70
order by name;
```

Find the average m1 mark.
select avg(m1) from test_stu;
Find city wise avg. marks.
select city, avg(m1)
from test_stu
group by city;
Arrange the result alphabetically.
select city, avg(m1)
from test_stu
group by city
order by city;
Arrange the result by avg. marks.
select avg(m1), city
from test_stu
group city
order by avg(m1);
HAVING:- (To be used with group by).
select avg(m1), ~~city~~ m2
from test_stu
group by ~~city~~ m2;
having to be used when we want to filter the result that we got after using group by. Having can be used only after group by.

SYNTAX:-
select
from
where
group by
having ();

order by - sorting / arranging / alphabetically / numerically
group by - grouping

Select the city wise avg. of m1 & find the details of avg. mark greater than 200. 30.

```
select city, avg(m1) "AM"
from test-stu
group by city
having avg(m1) >30&
order by city;
```

having – if we want to place a condition after grouping

Group with respective cities, the avg. marks of m1 & find those marks where the avg is between any 2 nos.

```
select city, avg(m1) "AM"
from test-stu
group by city
having avg(m1) between 60 & and 70
order by city;
```

NESTED SUBQUERIES

A subquery is a select statement embedded in a clause of another SQL statement.

```
select          ← Main Query
from
where
        (select...
         from...      ← Sub Query
         where...)
```

SYNTAX:-

```
select . .
from table
where exp-operator (select ..
                    from table
                    where ..);
```

TYPES OF SUBQUERIES:-

1) Single Row subquery:-
The inner query

* Can return only one result to the outer query..
* Operators inside <=, <, >=, >, =, <>

Eg:- select
```
. . . (select m1
       from test_stu
       where name='Ryan');
```

2) Xle row subquery:-

* Returns Xle result to the outer query.

Eg:- select . . .
```
(select *
 from test_stu
 where city='Kochi');
```

Find the students who have marks greater than Ryan's m1 marks. // single row subquery.

```
select * from test_stu
where m1 > (select m1 from test_stu
            where name='Ryan')
order by name;
```

Find the name & marks of students whose ~~live~~ in marks are greater than those living in Trivandrum.

```
select name, m1
from test_stu
where m1 > (select m1 from test_stu
            where city = 'Tvm')
order by name;
```

Single row subquery using 'having' :-

Find the cities where avg mark is greater than Ryan's mark.

```
select city, avg(m1)
from test_stu
where avg(m1) > (select m1 from test_stu
                 where name = 'Ryan')
```

select city
grouping
avg(mark) > Ryan's;

~~Group by city~~

```
select city from test_stu
group by city
having avg(m1) > (select m1 from test_stu
                  where name = 'Ryan');
```

Single row subquery using 'select' :-

Display the name & avg. marks of students

```
select name, (select avg(m1) from test_stu)
from test_stu order by name;
```

We can have the select query as condition also apart from that we can have it as a field just like the above example.

The row subquery:-

* Require use of IN, ANY, ALL or EXISTS operators.

Find the marks of students in Kochi.

```
select m1 from test_stu
where city = 'Kochi';
```

Find names & marks of students where m1 marks are lesser than those in Kochi.

```
select name, m1
from test_stu
where m1 <Any (select m1 from test_stu
               where city = 'Kochi');
```

Any & All operators (Combine with arithmetic operators)

>All — More than the highest value returned by sub query.

<All — Less than the lowest value returned by sub query.

<Any — Less than the highest value returned by sub query

>Any — More than the lowest value returned by subquery.

=Any — Equal to any value returned by the subquery (same as IN)

=All — Equal to all value returned.

```
select name, m1
from test-stu
where m1 > any
(select m1 from test-stu where city = 'Kochi');
```

Find name & marks of students whose mark is equal to those in Kochi.

```
select name, m1
from test-stu
where m1 = ~~any~~ all
(select m1 from test-stu where city = 'Kochi')
```

DISTINCT :- avoids repetitions

Find the cities of the students.

```
select distinct (city)
from test-stu;
```

Find the m3 without repetition.

```
select distinct (m3)
from test-stu;
```

```
select name, m1
from test-stu
where m1 > any
(select m1 from test-stu where city='Kochi');
```

Find name & marks of students whose mark is equal to those in Kochi.

```
select name, m1
from test-stu
where m1 = ~~any~~ all
(select m1 from test-stu where city='Kochi');
```

DISTINCT:- avoids repetitions

Find the cities of the students.

```
select distinct (city)
from test-stu;
```

Find the m3 without repetition.

```
select distinct (m3)
from test-stu;
```

EXIST / NOT EXIST :- (X le row & column subqueries)

Which are the dept. which do not have teachers?

```
select dname
from dept1
~~where~~ where not exists
(select did from teacher1
 where dept1.did = teacher1.did);
```

Which are the department which have teachers?

```
select dname
from dept1
where exists
(select did from teacher1
where dept1.did = teacher1.did); // also include order by dname; if needed in alphabetical order.
```

Xle column subqueries :-

```
select dname, did, dbudget
from dept1
where not exists
(select did from teacher1
where dept1.did = teacher1.did);
```

→ Xle columns.

same query can be used for exists.

NOTE:-
When using exist & not exist, don't use the common field name before the exist/not exist ~~keywork.~~ keyword. SQL will take care of that on its own.

IN/NOT IN — (Xle rows / columns subqueries)

↳ IN, not IN can be used in place of exists / not exists.

Which are the dept. which do not have teachers?

```
select dname, did, dbudget
from dept1
where did not in
(select did from teacher1)
order by dname;
```

→ And also no need of comparison.
→ use the common field name before the IN/NOT IN keyword.

Which are the dept. which have teachers?

```
select dname, did, dbudget
from dept1
where did IN
(select did from teacher1)
order by dname;
```

→ Also the diff between EXISTS & IN.

→ Common field name included before the IN keyword & no need of the common field comparison in sub query.

Using exist, find the departments where teachers have a salary greater than 500

```
seled did, dname
from dept1
where exists
(select did, t-sal
from teacher1
where dept1.did = teacher1.did
                and teacher1.tsal > 500);
```

Using exist & in:, find dept. where teacher names end with 'i'.

```
select dname, did, dbudget
from dept1
where did in
(select did from teacher1
 where tname like '%i');
select dname, did, dbudget
from dept1
where exists
(select did from teacher1
```

where dept1.did = teacher1.did and
teacher1.name not like '%.i');

delete / update operations — Exist, Not exists, In, Not In.

1) Delete operation :-
Delete the teacher whose name ends in 'l'.
(use exists / In).

delete from teacher1
where exists
(select did from dept1
where dept1.did = teacher1.did and
tname like '%.l');

delete from teacher1
where did in
(select did from dept1
where tname like '%.i');

2) Update operation :-
Increase all the teacher's salaries by 10. (use exist & In)

update teacher1
set tsal = tsal + 10
where exists
(select did from dept1
where dept1.did = teacher1.did);

update teacher1
set tsal = tsal + 10
where did in
(select did from dept1);

VIEWS IN SQL :-

Create a view of all students who stay in Kochi.

H.W
Diff. bet view, subtable & a table

```
create view V1
as
select * from test_stu
where city = 'Kochi';

select * from V1;
```

update a view :- (same as ~~create~~ create)

```
create ~~view~~ V1 or replace view V1 // if we already have V1, replace it; else create the view V1.
as
select * from test_stu
where city <> 'Kochi';
```

dropping a view :-

```
drop view V1;
select * from V1; // error.
```

Questions :-

1) Display all client name
2) List all clients & city name of all people who live in city starting with 'M'.
3) List all the clients who have done 'COD' payment.
4) Find out the total qty ordered from the orders table.

5) Which are the clients whose have ordered who have qty ordered is 2.

```
select C.name
from Cust_Master C, Order_Details O,
                    Sales_Order S
where C.Client_no = S.Client_no and
      S.Orderno = O.Order_no and
                  O.Qtyordered = 2;
```

6) Find the client name whose order status is either Cancelled / Fulfilled.

```
select C.name
from Cust_Master C, Sales_Order S
where C.Client no = S.Client_no and
                    S.OrderStatus In
                    ('Cancelled', 'Fulfi
                    -lled');
```

TABLE

* The table is an actual or real table that exists in physical locations.
* A table allows to perform add, update or delete operations on stored data.
* We cannot replace the table obj. directly because it is stored as a physical entry.

VIEW

* Views are the virtual / logical table that does not exist in any physical loc.
* We cannot perform add, update, or delete operations on any data from a view. If we want to make any changes, we need to update the data in the source table.
* We can easily use the replace option to recreate the view because it is a pseudo name to the SQL statement running behind on the database server.

SUBTABLE

Same col definitions
* Subtable is created mostly for grouping purposes. Grouping in this case means that when you have large gps of similar data, you can group them according to their values. Eg:- To use an aggregated func. such as Sum to sum all the info; hence providing only the calculated totals from the subtable.

JOINS IN SQL:-

* Joins → combining / join the fields from more than 2 tables.
* We can combine the tables only if they are related / if they a have a common fields. In short, if 2 tables are connected

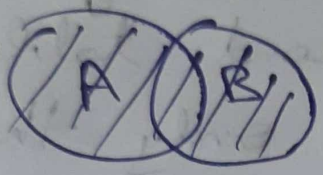
via primary key foreign key relationship.

TYPES :-

1) Natural Join — No nulls
2) Inner Join — ON keyword — commons
3) Outer Join — ON keyword :— nulls also included

1) NATURAL JOIN :- (or Join)

Assume, we have A & B (2 tables) [PK ↔ FK]



[Includes A, B & A∩B]

select * ↱ / Join (both same)
from A Natural Join B ; ↳ In our version of SQL, use the keyword natural Join.

select *
from dept1 Natural Join teacher1 ;

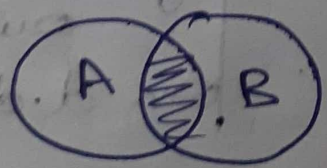
[In Natural join, the null entries are avoided or not null entries are only taken.]

→ The first field in the result, will always be the connecting field.

2) INNER JOIN :- ON keyword should be used in Inner Join

2 tables → A & B.

Inner join means only the common portion:



select *
from A inner join B
on <related field> ;

select *
from A inner join B
On A.did = B.did ;

select dname, tname
from dept1 inner join teacher1
on dept1.did = teacher1.did;

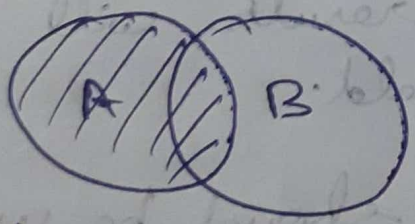
(or)

select d.dname, t.tname
from dept1 d inner join teacher1
on d.did = t.did;

Here, in inner join, all the fields from the first table & second table is displayed in result with only the common field data.

3) OUTER JOIN :- — ON keyword.
   a) Left outer Join / Left Join
   b) Right Outer Join / Right Join
   c) Full Outer Join.

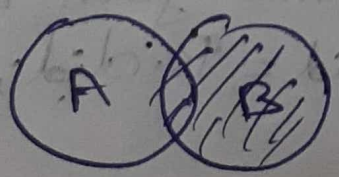
Left Join / Left Outer Join



select *
from A left outer join B
on A.did = B.did;

select d.dname,
d.did, t.name
from dept1 d left
outer join teacher1 t
on d.did = t.did;

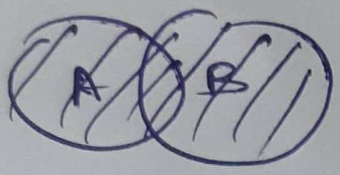
Right Outer Join



select *
from A right outer join B
on A.did = B.did;

```
select d.did, d.name, t.tname
from dept1 d right outer join teacher1 t
on d.did = t.did;
```

~~Full~~ Full Outer Join



Combination of left outer & right outer join.
[left outer join + right outer join]

```
select *
from A full outer join B
On A.did = B.did;

select d.did, d.name, t.tname
from dept1 d full outer join teacher1 t
On d.did = t.did;
```

Outer joins → can always have null values

[In natural join, it is also the combination of left & right tables; but the difference is we don't have a null value. But in outer join, we can have null values.]

and also, the common field is not repeated in case of natural join, but in case of outer join, the field name is repeated. to take care of the null values. No need of ON keyword in case of natural join, but in outer join we use ON keyword.