DDL data definition lang. / DML data manipulation lang / TCL transaction control lang / DCL data control lang / DQL data Query lang (or) DRL Data Retrieval lang

Scope of RDBMS in IT industry.

RDBMS It is a collection of data and a set of programs to access the data. It is called RDBMS because it is based on the relational model. In relational model, the data's are represented in terms of rows & colums.

File processing system

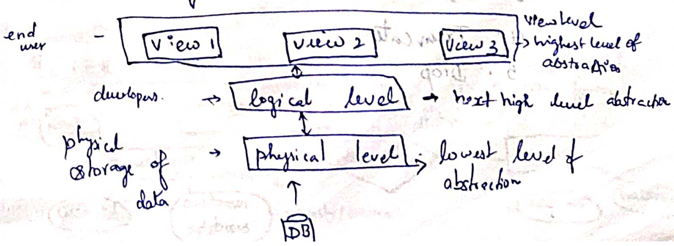
Disadvantages of File / Need for DBMS

1. Data redundancy & Inconsistency
2. Difficulty in accessing data
3. Data Isolation
4. Integrity problem
5. Atomicity problem
6. Concurrent access Anamolies
7. Security problems

View of data

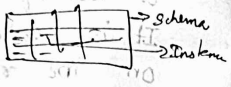
→ purpose is to provide user with abstract data

→ hiding the difficulties



Schema → design / structure of the database

Instance → amount of data / information in the table at particular moment of time



Int → schema (not change)

di → Instance (change)

Data Independence → ability to change one schema without affecting schema is next higher level

View level → sub schema

what data is stored — logical level → logical schema

how data is stored — physical level (storage area) → physical schema

DDL - Data definition lang

interact direct with db (Create structure of data)

1. Create
2. Alter
3. Rename
4. Truncate
5. Drop

DML - Data manipulation lang

1. Insert
2. Update
3. Delete

Datatype

1. Char — A2 C5 D C53
2. Varchar 2
3. Number — Integer, float
4. Date → to store Date (DD-MM-YY)

26-JUN-2023

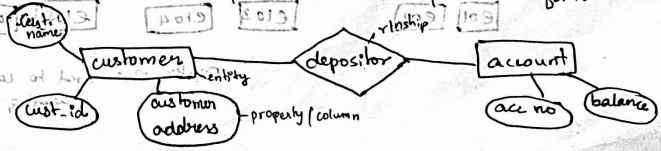
Data models

⇒ It is a conceptual tool of how data is organised stored & accessed within a database

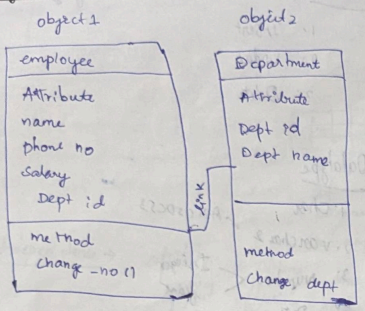
⇒ Blueprint for designing & implementing

- Data models
  - object based
    1. ER model
    2. object-oriented
  - Record based
    1. Relational
    2. network model
    3. hierarchical
  - physical Data model
    i) unifying model
    ii) Frame memory

1:1 ER models ⇒ represent the real world problem in pictorial form



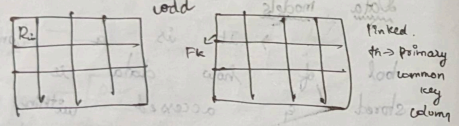
2. object - oriented



3. Relational model — widely used model.

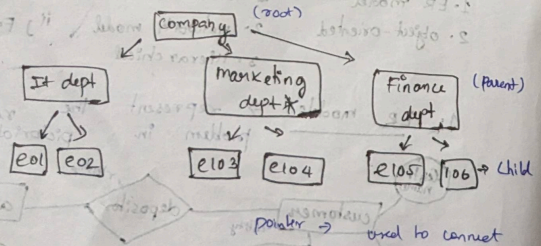
Dr. E. F. codd

codd



4. Hierarchical

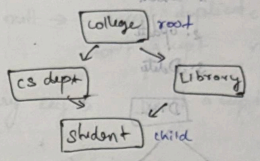
1. First - DBMS model
2. Organises in hierarchical tree structure



5. network model

→ extension of hierarchical model

→ many to many r/ship allowed



6. physical data model

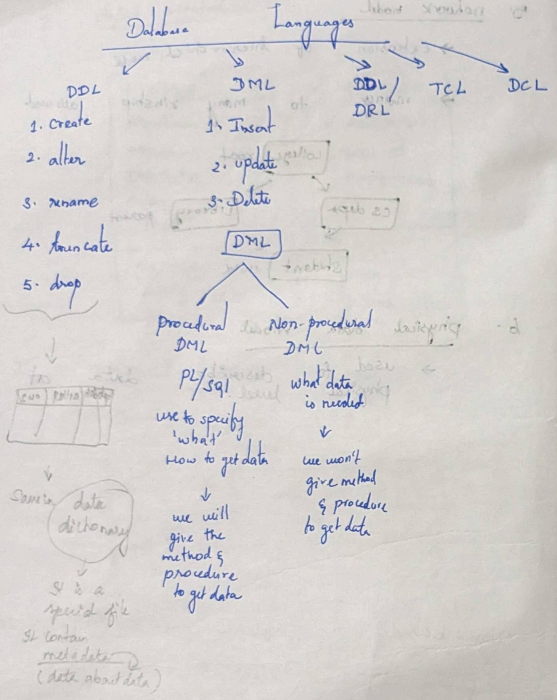
→ used to describe data at physical level.

Object based | Record based | Physical

ER | Relational | unity of frame memory

Object oriented | Network

Hierarchical



DCL

1. Grant → permission given
2. Revoke → getback permission

Integrity constraints

1. unique → accept null value
   not accept duplicate / repeated value
2. Not null → accept duplicate
   don't accept null value
3. primary key → Don't accept duplicate & null value.

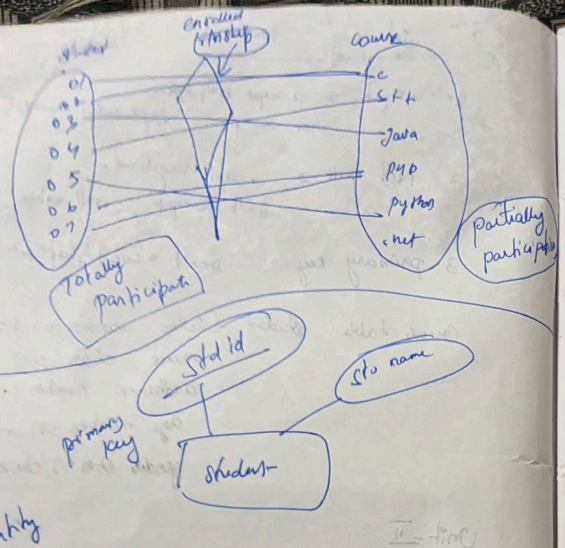
```
Create table Student (Rollno Varchar(10) primary key,
                      name char (20) not null,
                      aadharno number (12) unique,
                      age number (3) not null,
                      gender char (1) check (gender in ('M','F')));
```

Unit - II

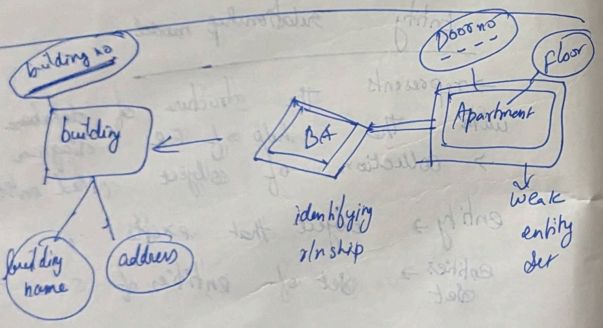
Entity relationship model

→ represents the structure of database with the help of ER diagram
→ collection of object called entities

entity → object that exists
entities set → set of entities of same type.

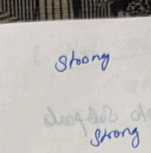


Strongly entity



—— strong entity → single rectangle

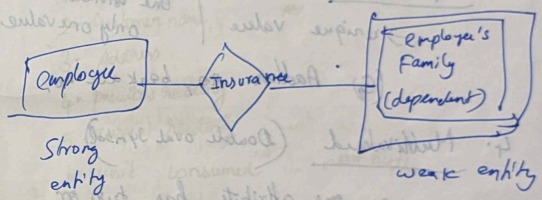
- - - - weak entity → double rectangle



appartment depend on Building

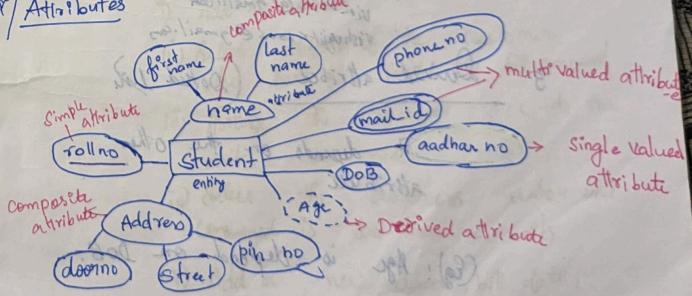
weak entity depend on strong entity.

To have weak entity, we need strong entity.



properties / Attributes

Relationship.



1. Simple
→ cannot be divided into subparts furtherly
(eg): Roll no

2. Composite
→ can be divided into subparts furtherly
(eg): firstname, lastname

3. Single valued
→ unique value | one attribute has only one value
(Eg) Aadhar no, bank acc no,

4. Multivalued (Double oval symbol)
→ one attribute has two or more value
(Eg) phone no.
Sim1, Sim2

mail
vinkboi4@gmail.com
vishaajitr2002@gmail.com

5. Derived attribed (Dotted line)
It depends on the other attribute

(Eg): Age is depend on DOB.

6. Null valued
attribute has no value,
Someone has phone but no sim card.
So, phone no. attribute is null

1. Create a table EB bill with the following columns

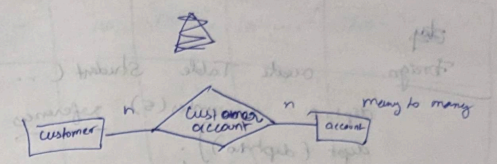
column name / Field name	constraints
1. customer id	- primary key
2. Customer name	
3. Address	
4. previous reading	
5. current reading	
6. unit consumed	- not null
7. amount	
8. bill status	

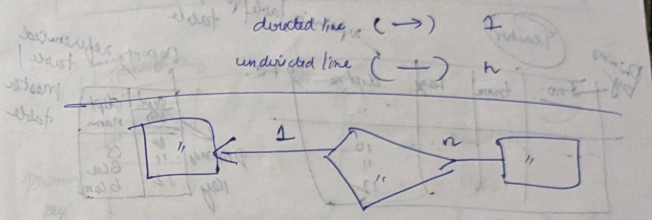
i) Add 5 records
ii) Select details of the customer who haven't paid the bill amount
iii) Arrange the records acc to the unit consumed
iv) Select the customers whose unit consumed is less than 100 units
v) Add a new column, bill date, (datatype is date)

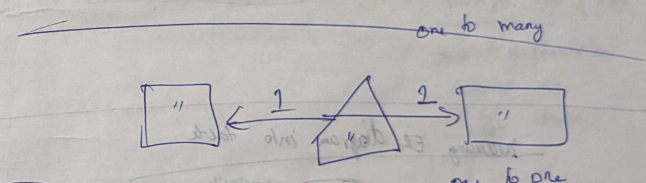
2. create a table inventory with the following fields & answer the Queries

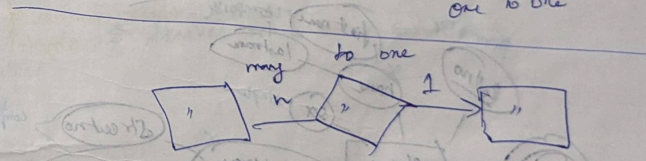
column name	constraints
i) product Id	primary key
ii) product name	
iii) Supplier name	
iv) unit price	
v) Quantity	not null
vi) Total price	

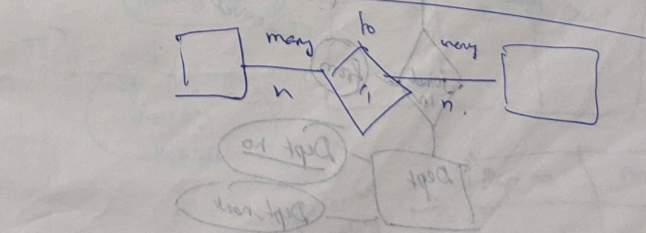
1. Add 5 records
2. Calculate the total amount
3. Select the productname, unique price from the table
4. Arrange the records based on Quantity
5. Add a new field stock number of size 10



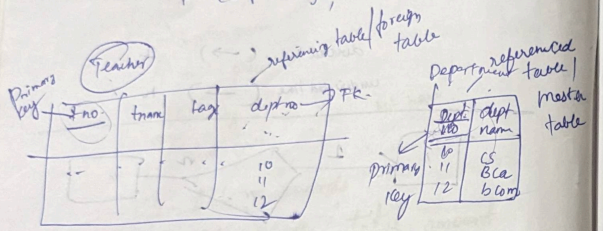


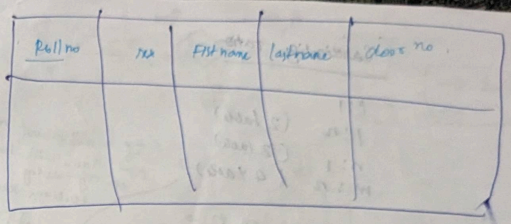


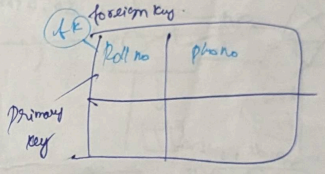




Foreign create Table student (...
deptno number(5) references
dept (deptno));



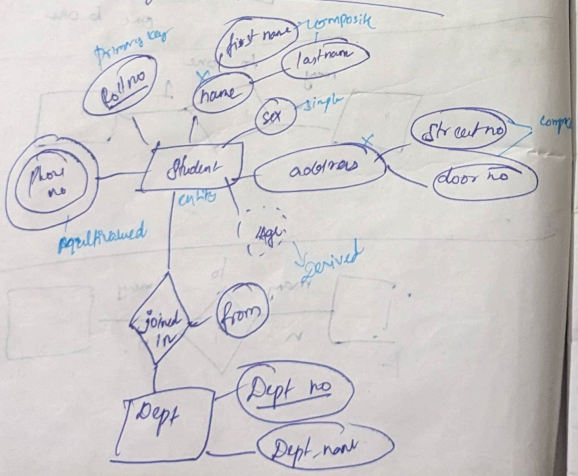


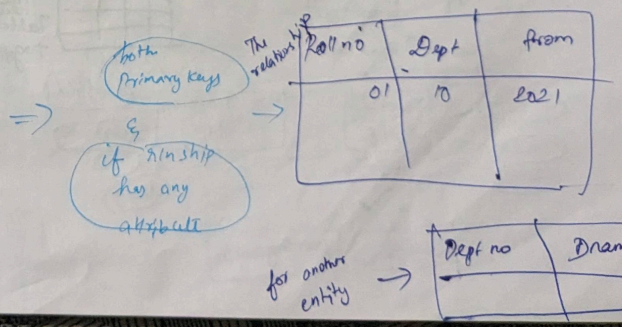


← Multivalued uses foreign key (FK) and it should be created as new table

⇒ Derived one is not entered.

Reducing ER diagram into table



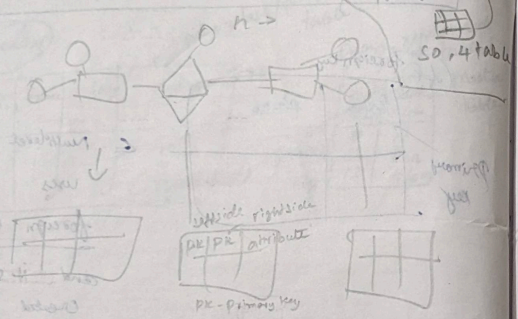


Cardinality ratio

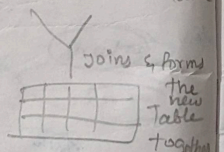
1:1
1:n (2 table)
n:1 (2 table)
m:n (4 table)?

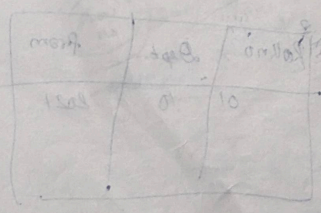
If multivalued means new table will be coming using foreign key

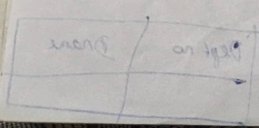
so, 4 table



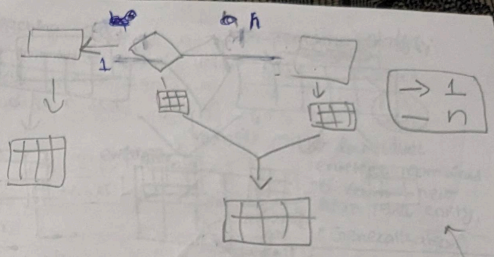
(n) → Table side → yh table + that corresponding 'n' table



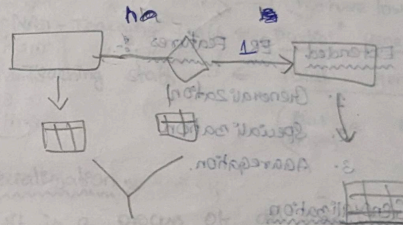


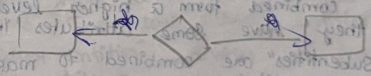


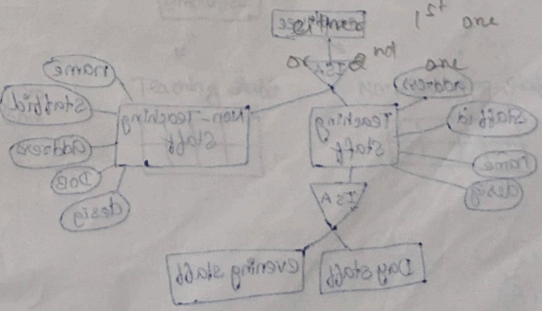
i)

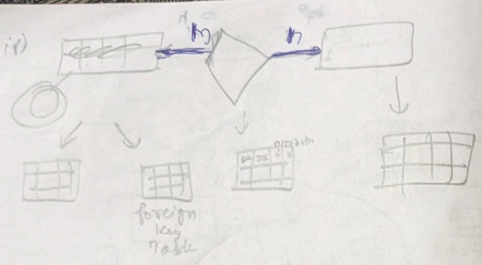


ii)







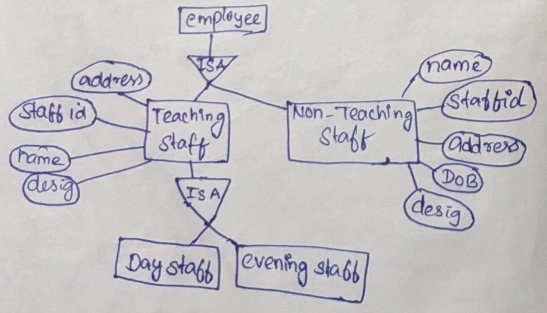


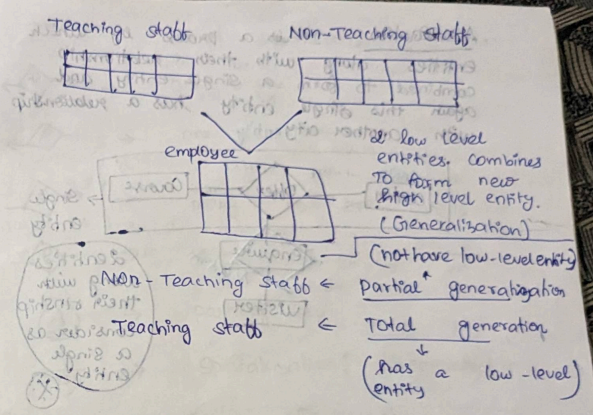
Extended ER Features :-

1. Generalization
2. Specialisation
3. Aggregation.

1. Generalization

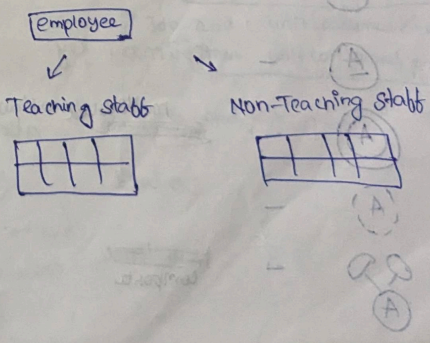
1. It is the process of extracting common properties from a set of entities and create a generalised entity from it.
2. Two or more lower level entities can be combined form a higher level entity if they have some attributes in common
3. "Subentities" are combined to make a "Super entity"
4. Generalization is a 'Bottom up approach'



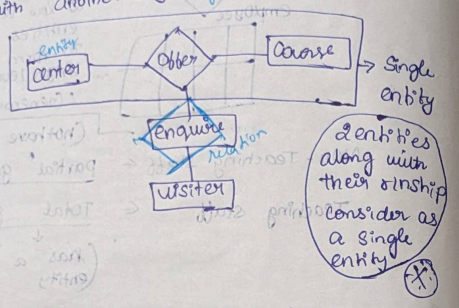


2. Specialisation :

1. It is a process of devideng (specialising) one high level entity into two low level entity.
2. Specialisation is "top down approach"
3. one high level entity divides and forms Two low level entities
4. "Super entities" divides to make a "Sub entities"



Aggregation — It is a process by which entities along with their relationship combined to form a single entity and again this single entity has a relationship with another entity



E – Strong entity

E – Weak entity

R – rlnship

R – Identifying / weak entity rlnship

A – entity attribute

A – Primary key

A – multivalued / composite attribute

A – derived

A – composite

— → Partial participation

R → one to one 1-1

R → one to many 1-M

R → many to one M-1

R → many to many M-N

A → Partially key

ISA → Specialisation / Generalisation

= → Totally Participation

```
update EBBill set amount = case
when unitconsumed <= 100 then unitconsumed *2
when unitconsumed >100 and unitconsumed <=200
        then unitconsumed *3
     >200 and unitconsumed <=300
        then unitconsumed *4
     > 300 and unitconsumed <=400
        then unitconsumed *5
else unitconsumed *6 end;
```

Ex NO:4 Views and Snapshots

The syntax to create a sql view is

create view view_name
As
SELECT column_list
FROM table_name [WHERE condition];

⇒ view name is the name of the view

⇒ The SELECT statement is used to define the columns and rows that you want to display. Views.

Table Creation

SQL> Select *

DCL Operations

User 1

username : A21CSDC01

Pass :

student table

Grant previlage on tablename to username;

Grant select /all on student to A21CSDC02 / public

Grant Succeeded

User 2

username : A21CSDC02

pass :

Select * from A21CSDC02 student

Revoke purvibge on tablename from username

Ex NO:5 Multiple tables and nested Queries

(Query within a Query)

SQL> Select * from emp;

EMPNO	EMPName	Salary	Departmentno	ManagerID
101	Kumar	12000	10	(null)
102	John	15000	50	101
103	Suresh	9000	10	101
104	Jai	4000	20	103
105	Sam	11000	30	103
106	Raman	7000	20	104
107	Surya	15000	30	105
108	Kamnan	5000	30	102
109	Joseph	9000	20	106
110	Jegan	16000	10	107

10 Rows Selected

1) SQL> Select empname from emp where Salary = (Select Salary from emp where empname = 'Joseph');

EMPNAME
Suresh
Joseph

2) SQL> Select empname from emp where Salary > (Select Salary from emp where empname = 'Joseph');

EMP_NAME	
Kumar	Surya
John	Jegan
Sam	

3.

EMPNAME
Kumar
John
Suresh
Sam

Surya
Joseph
Jegan

7 rows selected

4.

EMP_NAME
Suresh
Jai
Raman
Kannan
Joseph

5.

EMP NAME
Jai
Raman
Kannan

(not equal to)

6.

EMP NAME	
Kumar	Raman
John	Surya
Jai	Kannan
Sam	Jegan

8 rows selected

7. SQL> Select empname from emp where salary > (Select salary from emp where empname = 'joseph') and departmentno = (Select departmentno from emp where empno = 103);

EMP NAME
Kumar
Jegan

8. SQL> Select empno, empname, salary, departmentno from emp where salary = (Select max (salary) from emp);

EMP	EMPNAME	Salary	Departmentno
110	Jegan	16000	10

9. SQL> select departmentno, avg (salary) from emp group by departmentno having avg (Salary) = (Select min (avg (Salary)) from emp group by departmentno);

Avg (salary)	
20	10
6666.66667	20
	30
	50

12000, 9000, 16000, 4000, 7500, 9000, 11000, 13000, 5000, 15000

10. SQL> Select empname from emp where empno in (Select managerid from emp);

Emp Name
Kumar
John
Suresh
Jai
Sam
Raman

7 rows selected

Surya

11. SQL> Select empname from emp where empno not in (select managerid from emp where managerid is not null);

Emp_Name
Kanan
Joseph
jegan

12. SQL> Select empno, empname, salary, departmentno from emp where salary > any (select salary from emp where departmentno = 10)

EMPNO	EMPName	Department no
110	1600	10
102	15000	50
107	15000	30
101	12000	10
105	11000	30

13. SQL> Select empno, empname, salary, departmentno from emp where salary > all (select salary from emp where departmentno = 20);

Keys :

Key is
→ used to identify any record uniquely from the table
→ It used to establish & identify relationships between the tables

There are five keys namely,

i) Super key
ii) Candidate key
iii) Primary key
iv) Foreign key
v) Unique key

Super key :

* It consists of any no. of the attributes or tuples
* The every each attributes are uniques

Can identify each tuple uniquely

(eg):

* Student (name, rollno, gender, class, m1, m2)
* Person (name, height, weight, colour, age, IDno)

Super key used to identify each record uniquely

Candidate Key :-

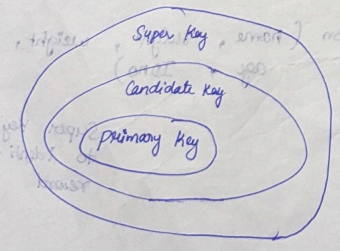
* The minimal number of attributes of data in a table is candidate key
* It has uniques data

* Student (roll no , gender)
* person (Height , weight)

* A table can have any no. of Candidate key

primary key :-

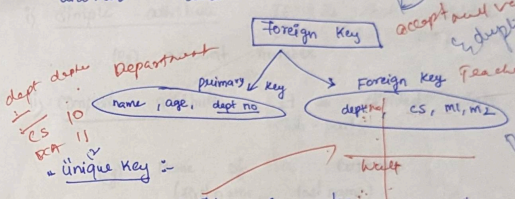
* The primary key is never be null
* The primary key is always unique.
* The primary key never be changed (no updation is possible)
* A table has only one primary key



* A table can only have one primary key
* Don't accept null & duplicate value

Foreign Key :-

* A foreign key is a column or set of column in a table refers to the primary key of another table



* It can be a null value
* Once it is assigned, it won't be change
* It is unique comparing all other Tuples | data's in the table

(Eg) : student (aadhar number)

Teacher referencing department master table.

T.no	name	deptno FK
		10
		Null

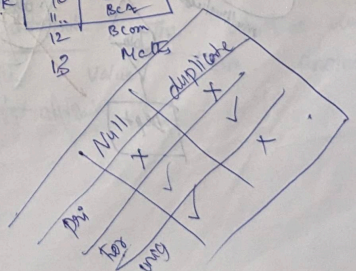
PK

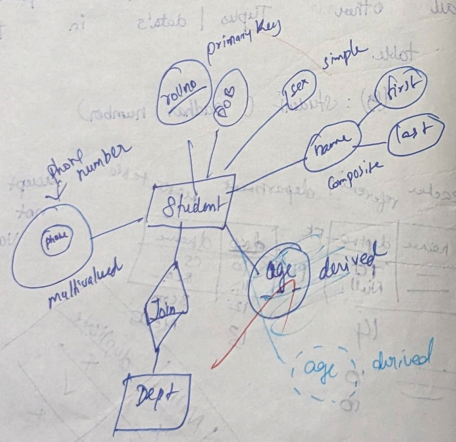
dno	dname
10	CS
11	BCA
12	BCom
13	Maths

14

10
10

accept null value not duplicate value





Attribute :-

⇒ In E-R model, there are attributes are characterized by the following types :-

i) Simple attribute — It is not divided into sub parts
(Eg) account number

ii) Composite attribute — It is divided into sub-parts.
(Eg) Name of a customer (firstname, lastname)

iii) Single valued attribute — The attribute holds only single value
(Eg) ID of a person

iv) Multi-valued attribute — The attribute holds set of values for a specific entity
(Eg) Employee details (name, age, salary)

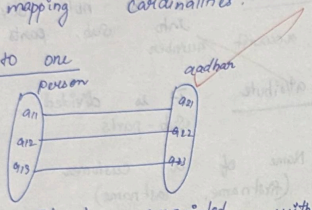
v) Null attribute — An attribute doesn't have any value. It might be not applicable, not known or missing

vi) Derived attribute — The attribute derived from the value from another ~~entity set attribute~~

Mapping Cardinalities

The number of entities associated with the number of entities of another entity set via relationship is mapping cardinalities.

one to one

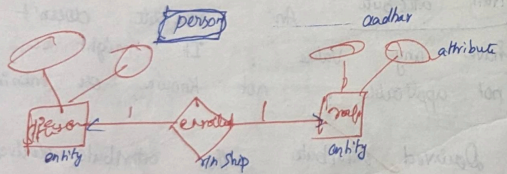


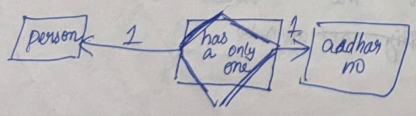
⇒ The entity A associated with atmost one entity in B.

⇒ The entity B associated with atmost one entity in A

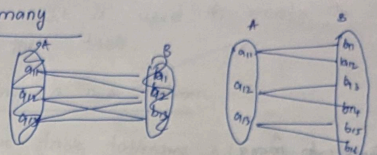
⇒ A person has only one aadhar no

⇒ An aadhar can be associated to only one person



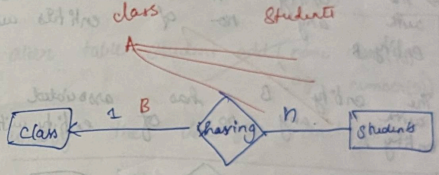


one to many

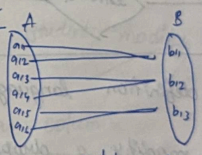


⇒ The entity A is associated with any no. of entities in B

⇒ The entity B has atmost one entity in entity A.

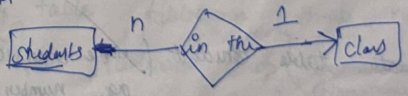


many to one

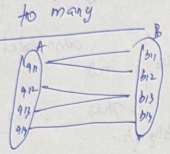


⇒ The entity A is associated with atmost one entity set in entity B

⇒ The entity B is associated with any no. of entity in entity A

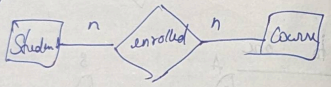


many to many



⇒ The entity A has associated with any no. of entities with entity B

⇒ The entity B has associated with any no. of entity with entity A



DDL (Data definition language)

It is to create, modify & dropping the structure of the database objects

1. Create It is used to create a new table.

Query: create table tablename (columnname Datatype (Value 1) : value n);

Eg: create table Student (name char(10), age number(2));

2. Alter It is used to modify the structure of the table.

i) adding a new column

Query: alter table tablename add (columnname datatype (Size));

Eg: alter table student add (name char(10));

ii) adding more than one column

Query: alter table tablename add (columnname 1 datatype (Size), : 2 : n);

Eg: alter table Student add (name char(10) age number(5));

iii) Modify the datatype and size

Query: alter table tablename modify (columnname datatype (Size));

Eg: alter table student modify (name char(15));

iv) delete a column

Query: alter table tablename drop column columnname;

Eg: alter table Student drop column name;

v) delete multiple columns

Query: alter table tablename drop column name, age;

(Eg)

Syntax: alter table tablename drop column (columnname 1 , columnname 2 : n);

3) rename :- It is used to rename table & column name

i) renaming tablename :-

Syn: rename oldname to newname;

Query: rename student to emp;

(Eg)

ii) renaming column name :-

Syn: alter table tablename rename column oldname to newname;

Query: alter table student rename column student1 to student5;

(Eg)

4) Truncate It is used to remove all the values from the table

Query: Truncate table tablename;

(Eg): Truncate table student;

5) drop remove the table permanently from database

Query: drop table tablename;

(Eg): drop table student;

DML - Storing / retrieving / modifying & deleting data in a table.

DCL - providing security to db objects

GRANT REVOKE

DML storing, retrieving, modifying, deleting data in a table

1) i) Insert

Insert into tablename values (value);

ii) specific insertion

Insert into tablename (columnname, ...) values (values, ...)

iii) multiple insertion

Insert into tablename values (&columnname);

2. i) update

update tablename set columnname = 'value';

ii) update a particular record

update tablename set columnname = ' ' where columnname = 'value';

3. Delete

i) delete from tablename (optional) (or) delete tablename;

ii) Delete a particular record

delete from tablename where columnname = value;

delete from tablename

DCL

Grant privilage on tablename

revoke

Unit - III

Relational algebra

→ Theoretical procedural language given (They'll tell, what data needed & how to get it. need to write procedure)
by E.F codd in 1970

→ It is the basics for design & development of query language like SQL

→ It is essential for understanding of how databases work
  * for designing effecient queries
  * data manipulation.

(Don't need to tell the procedure & methods)

To retrieve data from relation (Table)

- Theoretical
- practical SQL

i) Relational algebra (procedural language)

ii) Relational Calculus (non-procedural language)

Relational algebra

- Unary operations (operations performed on single data)
  1. Selection (σ)
  2. projection (π)
  3. Rename (ρ)
- Set theory operations
  1. union ($\cup$)
  2. Intersection ($\cap$)
  3. Set difference/minus (-)
  4. Cartesian product ($\times$)
- Binary operations
  1. Joins ($\bowtie$)
     - Inner Join
     - Theta join
     - Natural Join
  2. Division ($/$)
- Extended/Derived operations
  1. Outer join
     - Left
     - Right
     - full
  2. Aggregate function

Relational algebra } → takes (table) relation as input & the output will also be a relation. (table)

unary operations → operations performed on a single relation

Binary operation → two relation.

1. unary

i) selection operation (σ) Sigma

→ It works on rows

→ used to retrieve rows that satisfy a given condition

Syntax: $\sigma_{(predicate\ /\ condition)}(Relation\ name)$ (Relation name means Table name)

Student

Rno	name	marks
01	A	70
02	B	75
03	C	80
04	D	90

(Student)

$\sigma_{(mark > 75)}$

in SQL (select * from tablename where (condn);)

Selects the records/tuples that satisfy the given condition / predicate

---

σ (Student) ← To select the whole entire table

(select * from tablename ;) In SQL

∧ - and
∨ - or
¬ - not

$\sigma_{(mark > 75 \vee mark < 90)}$ (Student)

$\sigma_{(mark > 75 \wedge mark < 90)}$ (Student)

ii) projection operation (π) pi

→ It works on columns.
→ It avoids the duplication

Syntax

$\pi_{R.no, mark}$ (Student)

Rno → column name
Student → Table name / Relation name

InSQL: Select Rno, mark from student;

Rno	mark
01	100
02	99
03	98

Eg no: 2

R.no	name	mark
01	A	60
02	B	65
03	C	70
04	A	80

i) Select name from student

output

name
A
B
C

(avoids duplication)

π & σ together

Select name, Rno from student whose mark is 70

$\pi_{name, Rno}\left(\sigma_{mark=70}(Student)\right)$

iii) Rename (ρ) (rho)

→ Rename the table name
→ Rename the column name
→ Rename both table & column name
→ Rename the resulting Relation

Syntax

$\rho_{(new\ name)}$ (Relation name)

(eg) ρ (Student)

Student details

Rno	name	mark
1	A	60
2	B	65
3	C	70

only change attribute / column

$\rho_{(Rollno, Sname, Result)}$ (Student)

change both table name & attribute/column

ρ candidate (Rollno, Sname, Result) (Student)

Rename the resulting relation

$\rho_{result}\left(\pi_{mark}(student)\right)$

Result

60
65
70

SQL	PL/SQL
Structural query language	procedural language with SQL
→ In SQL only one query is executed at a time	→ (collection of programming statements + SQL queries)
→ There is no programming constructs like variable, loops conditional & Branching statements	→ Extension of SQL
→ Queries are not saved in database	→ multiple queries can be executed
→ It is primarily used for performing SQL operations like Select, Insert, update, delete	→ It allows developers to write code blocks that include loops, variable, conditional & Branching statements, ex: Declare, begin, for, if, Declaration
→ It is the collection of pre-defined objects like table, views, sequences, clusters, Indexes, synonyms	→ Queries can be saved in form of PL/SQL object
	→ It is used to perform complex operations within the database like automate business activities
	→ It is collection of user defined objects like procedures, Functions, Triggers, Packages

Structure of PL/SQL

```
Declare        //optional
<Statement>;
Begin          //mandatory
<Assignment>;
<SQL Queriess>;
<O/P statements>;
Execution
<Error handling>;   //optional
End;           //mandatory
```

(Eg): 1 Addition of two no's

```
declare
n1 number(2);        -- datatype, size
n2 number(3);        -- variable name
c  number(3);
begin
n1 := 50;            -- assignment statements
n2 := 100;
c  := n1+n2;
dbms_output.put_line('The addition of two nos:' || c);   -- print statement
end;
```

To get the value after compilation (runtime value getting)

```
n1 := &n1
n2 := &n2
```

pipe symbol || for concordination

Procedure

Syntax (optional)

create or replace procedure procedurename [list of args] (optional)

is/as

variable declaration;
begin // mandatory
  Execution code;
Exception //optional
  Exception code;
end; // mandatory

(body)

Function → (It returns the value)

Syntax (optional)

create or replace function functionname [args] return datatype is

declaration statement;
begin //must
  Execution code
return (value); // must
end; //must

PL/SQL program to swap two numbers without using 3rd variable

is/as

```
declare
a number(3);
b number(3);
begin
a:=&a;
b:=&b;
dbms_output.put_line('before swapping a=' ||a|| ' and b=' ||b);
a:=a+b;
b:=a-b;
a:=a-b;
dbms_output.put_line('after swapping a=' ||a|| ' and b=' ||b);
end;
```

output

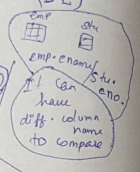
```
enter value for a: 5
old 5: a:=&a
new 5: a= 5
Enter value for b: b
old b: b=&b
new b: b= 6
swapping before a=6 b=9
after a=9 and b=6
swapping
```

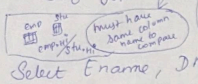
Program to count no. of digits

```
declare
a number;
num number:=0;
d number;
begin
a:=&a
while a>0
      loop
d:=(mod(a,10));
num:=num+1
a:=trunc(a/10,0);
      end loop
dbms_output.put_line('no of digits are' ||to_char(num));
end;
```

Joins (⋈) ← cartesian Product + Selection

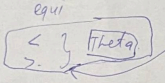
- Inner Join
  - → Theta Join / conditional Join ($\bowtie_\theta$)
  - → Equi Join ($\bowtie_=$)
  - → Natural join ($*$)
  - → Self Join
- outer join
  - Left Outer Joint ($\bowtie$)
  - Right " " ($\bowtie$)
  - full " " ($\bowtie$)





Checks the condn ($>, <, >=, <=, =$) & gives the output

Select Ename, Dname from EMP, Dept
where Emp.Eno (=) Dept.Ename
equi



Emp

E.NO	E.name	address
1	Ram	Chennai
2	Ravi	velloore
3	Rahul	Chennai

Dept

Deptno	Dname	E.name
D1	HR	1
D2	IT	2

Cartesian product

E.NO	Ename	address	deptno	Dname	E.name
1	Ram	Chennai	D1	HR	1
2	Ravi	velloore	D2	IT	2
3	Rahul	Chennai	D1	HR	1
			D2	IT	2
			D1	HR	1
			D2	IT	2

Self Join joining a same table two time by creating / using alice name

dept d , dept e.

deptno	dname	E.NO
D1	HR	1
D2	IT	2
D3	MRKT	4

d.deptno = e.Dname

d.deptno	d.Dname	d.Eno	e.deptno	e.Dname	e.Eno
D2	HR	1	D1	HR	1
D2	HR	1	D2	IT	2
D2	HR	1	D3	MRKT	4
D2	IT	2	D1	HR	1
D2	IT	2	D2	IT	2
D2	IT	2	D3	MRKT	4
D3	MRKT	4	D1	HR	1
D3	MRKT	4	D2	IT	2
D3	MRKT	4	D3	MRKT	4

3 rows will show.

Left Outer Join

Eno	Ename	address
1	Ram	delhi
2	vasun	chennai
3	Ravi	Bangalore
4	Rahul	chennai

emp

deptno	Dname	Eno
D1	HR	1
D2	IT	2
D3	MRKT	4

dept

Full / Left / Same answer

emp.ENO	Ename	address	deptno	Dname	dept.ENO
1	Ram	delhi	D1	HR	1
2	vasun	chennai	D2	IT	2
3	Ravi	Bangalore	null	null	null
4	Rahul	chennai	D3	MRKT	4

Right

deptno	Dname	Eno	emp.eno	e name	address
D1	HR	1	1	Ram	delhi
D2	IT	2	2	vasun	chennai
D3	MRKT	4	4	Rahul	chennai

full

emp eno	Ename	address	deptno	Dname	dept. eno
1	Ram	delhi	D1	HR1	1
2					

A

Id	name
10	Jai
20	Veer
30	John

B

name	mark
Rahul	50
Veer	36
John	60
Sam	70

A ⋈ B

Id	A.name	B.name	mark
10	Jai	null	null
20	Veer	veer	36
30	John	John	60

A ⟕ B

B.name	mark	Id	A.name
Rahul	50	null	null
Veer	36	20	Veer
John	60	30	John
Sam	70	null	null

A ⋈ B.

Id	A.name	B.name	mark
10	Jai	null	null
20	Veer	Veer	36
30	John	John	60
nal	null	Rahul	50
null	null	Sam	70

26/09/23

÷ / Divison operator → Derived operator

It is used if the Query contains the keywords "for all"

Ex: Find the name of the student who is registered for all the courses

Find the name of the employee who is working in in every project.

If R_1 / R_2 (i) ⇒ R_2 must be proper subset (R_2's column should be in R_1)

$R_1 \supset R_2$

(ii) ⇒ $R_1 \neq R_2$

R_1

Name	Course
Ramesh	DBMS
Raju	DBMS
Mahesh	DBMS
Ramesh	Java
Mahesh	Java
Raju	C++
Ramesh	C++

R_2

Course
DBMS
Java
C++

$= \frac{R_1}{R_2}$

$= (name, course) / course$

$= name$

Output :-

Name
Ramesh

Ramesh only come, because he only associated with all the courses (DBMS, Java, C++) mentioning in the R_2

Aggregate function

1. Sum()
2. min()
3. max()
4. avg()
5. count()
6. distinct()

$G_1 G_2 .. G_n$ G_1 $F_1(A_1)$, $F_2(A_2)$ (E)

group by Name — Aggregate — Rel. to name

27/09/23

Relational Calculus

Tuple Relational calculus (TRC) (rows)

Domain Rc (DRC) (columns)

TRC: $\{ t \mid P(t) \}$

t → tuple variable; | → pipe or slash; P(t) → predicate / condition

1. Non-procedural
2. Syntax
3. Quantifiers.
4. free variable / bound variable
5. atoms - formula
6. Safety of Expression

Quantifiers

1. Universal quantifiers ($\forall$) → for all
2. Existential quantifiers ($\exists$) → there exist

$\wedge$ (and), $\vee$ (or), $\neg$ (not)

$\forall t \in r\,(Q(t))$

for all tuple (t) belongs to relation (r) of $Q(t)$ (condn).

$\exists t \in r\,(Q(t))$

there exists tuple (t) belongs to relation (r) of $Q(t)$ (condn)

with Quantifiers → bound variable
$\forall t \in r(Q(t))$

without Quantifiers → free variable
$t \in r(Q(t))$
✗ (no)

relation — Table name
predicate — condn.

$t \in loan \wedge \exists s \in Customer\,[t[branchname] = s[branchname]]$

t — free variable
s — bound variable

Normalization :-

→ process of effectively organizing data in the database.

Delete anamolies (problems)
Insert anamolies
update anamolies
Inconsistency
Redundancy

Advantages

Sid	Sname	marks	Deptname	Building	Room no
1	Bala		CS	RD	72
2			CS	RD	72
3			Bcom	RD	72
4			B.com	VB	33
5			maths	TB	48
6			english	TB	7

To overcome this, we going to normalization

Student

Sid	Sname	marks	Dname

Dept

Dname	Building	Room no

First normal form (1 NF)

A table is said to be in 1 NF, if it satisfies the following condition

i) Each column should contain atomic values (single)

ii) Each column name should be unique

Sid	Sname	courses
1	Bala	C, C++
2	Ram	Java, .net
3	Prem	RDBMS

atomic (X)
unique (✓)
↓
so, not 1NF

I type

Sid (PK)	Sname	course
1	Bala	C
1	Bala	C++
2	Ram	Java
2	Ram	.net
3	Prem	RDBMS

II type

PK

Sid	Sname
1	Bala
2	Ram
3	Prem

FR

Sid	Course
1	C
1	C++
2	Java
2	.net
3	RDBMS

2 NF

i) It should be in 1 NF

ii) There should not be partially dependency

X : → Y

here X : → Y (Y dependent)

X determinant

X determines Y

Y is functionally dependent on X

dependency

(eg). Rollno → name

Roll no determines name

Key (primary key) non-key

Sid	course	course fee
1	C++	2000
1	C++	2000
2	Java	2500
3	.net	3000

partially dependency

A non-key column is partially dependent on the part of a key

(eg). course fee column depend on only course column not sid column

Sid	course
1	C
1	C++
2	Java
3	.net

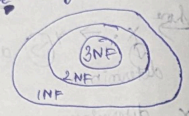
course	course fee
C	1000
C++	2000
Java	2500
.net	300

if a table is partially dependency we should make it into 2 tables

3NF

i) It should be in 2NF

ii) There should not be any Transitive dependency



↓

A non-key column should not depend on another non-key column

A → B → C (A determines B, B determines C)

A → C through B

Studentno (PK)	course (non-key)	coursefee (non-key)
101	Java	2000
102	C	1000
103	C++	1500
104	C	1000
105	Java	2000

A → B → C

Studentno → course → coursefee

(PK) (Non-key) (Non-key)

coursefee depend on course (or) course determines coursefee (B → C)

So it is Transitive dependency.

So we have to split into two tables

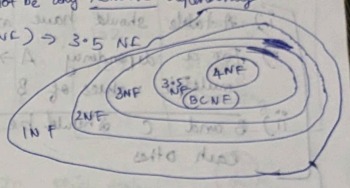
Studentno	course
101	Java
102	C
103	C++
104	C
105	Java

course	coursefee
Java	2000
C	1000
C++	1500

Boyce-codd normal form (BCNF) | (3.5 NF)

1. A table should be in 3NF
2. There should not be any reverse dependency

(Extension of 3NF) → 3.5 NF



Reverse dependency

↓

A key column should not depend on any other non-key column

Studentno (PK)	Course	Professor (non key)	
101	Java	Joseph	P_1
101	C	Bala	P_2
102	C++	Santhosh	P_3
102	Java	Ramesh	P_4
103	C	Suresh	P_5

Joseph takes Java course

non key colum determines key column

So, it is reverse dependency

Studno (PK candidate)	course	professor id (non key)
101	Java	P_1
101	C	P_2
102	C++	P_3
102	Java	P_4
103	C	P_5

Professor id (PK)	professor
P_1	Joseph
P_2	Bala
P_3	Santhosh
P_4	Ramesh
P_5	Suresh

(Introducing / creating a new column to overcome reverse dependency (eg) → professor id)

4NF

1. A table should in BCNF
2. There should not be any multivalued dependency

i) A table should have atleast 3 columns

ii) For a dependency $A \to B$ (for a single value of A, multiple values of B and C exists)

iii) B and C should be independant of (not determines) each other

If these 3 condn are satisfies a table, then it is multivalued dependency. And we have to avoid it

PK (composite key)

Studentno	course	hobby ← non-key
101	Java	Cricket
101	C	dancing
102	C++	drawing
102	Java	singing
103	C	Drawing

101 < Java, C
101 < Cricket, dancing

For a single value of A column(101) we have multiple value of B column (Java, C)

For a single value of A column (101) we have multiple value of 'C' column (Cricket, dancing)

So, it is multivalued dependency and we have to overcome this by splitting the table

Candidate keys

Sno	course
101	Java
101	C
102	C++
102	Java
103	C

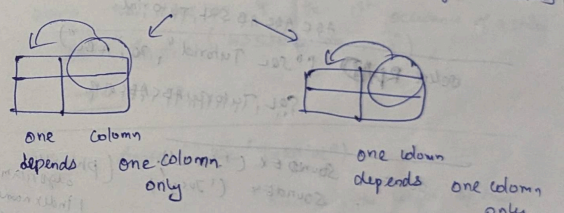
PK

Sno	hobby
101	Cricket
101	dancing
102	Drawing
102	Singing
103	Drawing

Multivalued dependency

In first table, 2 columns depends on one column.

To overcome we spliting into 2 columns



Database object → whatever stores in DB, that is DB objects.

(eg) Procedures, functions, Sequence, Tables, sequence, synonym, Package index, cluster

Select CHR(65) (as number code to character) → A

Select lower ('HI'); → hi

Select upper ('Select Hello') → SELECT HELLO

Select LPAD ("SQL Tutorial", 20, "ABC");
sticks length lpad_string.

ABCABCABSQL Tutorial

Select RPAD ("SQL Tutorial", 20, "ABC")

SQL TutorialABCABCABC

Select SOUNDEX ('Jurgil'), SOUNDEX ('Jurgil'); (phonetic algorithm) (index name pronunciation in english)

J200

Compare the Soundex codes given 2 String expressions & return an Integer.

Select LTRIM (' SQL Tutorial') as LeftTrimmedString;

output → SQL Tutorial

DO RTRIM(Hi kl hi)

Select Replace ('SQL Tutorial', 'T', 'M');

SQL MuMorial

Replace all occurances of a substring within a string, with a new string

Select Substring ('Geeks for Geeks', 1, 5);
start length

Output Geeks.

Extracts a substring starting from a position in an input string with given length

Select TRANSLATE ('Monday', 'Monday', 'Sunday')

Output Sunday

return the string from 1st arg after the change specified in 2nd arg translated into the change specified in the 3rd arg

return the position of the 1st occurance of a string

Select Instr ("W3Schools.com", "3")

Output 2

Select length ("SQL Tutorial");

Output 12

returns the length of the string.

Select concat ('W3Schools', '.com');

Output W3Schools.com

adds 2 or more strings together.

Lpad, Rpad, LTrim, RTrim, lower, upper

concat, SoundEx, Substring, length.

abs, Sin, cos, tan, Ceil, Floor, Sqrt, Cbrt, Power, round, log

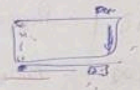
Select LTRIM (" Hi") from dual

Set linesize 200;

Select * from tablename;

Commit; ← to save the changes of the table.

Desc tablename ←



Value = ' ' ;
Single quotes

update Student set dept = Null; } drop all the data(record) from a column.

exec proc1 (01, 30, 40)

exec DBMS_Output.put_line (fun1 (___))

Select

Sql plus username Scott

password Tiger

Host string ojc

① Connect System ojc ②

connected

Version 11

username : System

password : ojc

Sql>

③ Sql > Create user A21CSDC53 (username) identified by vishwajit; (password)

user created

④ Sql> Grant Resource, connect to A21CSDC53;

Grant Succeeded

⑤ Sql> show user;

User is A21CSDC53

Connect

Enter username : A21CSDC53

password : Vishwajit

DDL

1. Create

Syntax

create table tablename (columnname1 datatype (15),
columnname2 datatype (20),
.....);

(Eg): create table employee (empid varchar2 (10),
ename char (20),
designation char (30),
DOB date,
Salary number (4));

Sql > Desc employee;

— shows the description of employee

2. Alter used to modify the structure of the table

i) adding a new column

alter table tablename add columnname datatype (size);

ii) adding more than one column

alter table tablename add (columnname1 datatype size,
columnname2 datatype size);

iii) To modify the column datatype & size

alter table tablename modify (columnname datatype (size));

iv) To delete a column

(no datatype & size in syntax)

alter table tablename drop column columnname;

v) To delete multiple column

alter table tablename drop (column1, column2, column3 ....);

vi) rename

It is used to rename the tablename and the column name

i) To rename table name

Rename oldname to newname;

ii) To rename column name

alter table tablename rename column oldname to newname;

iii) Truncate It is used to remove the values stored in the table.

Truncate table tablename;

iv) drop It removes the table permanently from the table

drop table tablename;

all datatype → ' '
numbers → no quote

DML

1) Insert

i) Single insertion — It is used to insert values in all the columns

Insert into tablename values(value 1, value 2, ... value n);

ii) specific insertion — To insert values in the particular columns

Insert into tablname (column 1, column 2 ... column n) values(value 1, value 2 ... value n);

iii) Multiple insertion — to insert more than one record (rows)

Insert into tablename values ('&column 1', '&column 2', ... '&column n');

2) i) update — It is used to update the values in the table

update tablename set columnname = value;
update student set dept = 'bsc';

ii) To update a particular record (specific update)

update Tablename set column name = value where columnname = value;

update student set name = 'Ram' where rollno = '40';

3) Delete — It used to delete all the record or any particular record

i) to delete all record

delete from tablename ; (optional)

(structure will be there) (only records in the table is deleted)

delete student;

ii) to delete particular record

Syn → delete from tablename where columnname = value;

(Eg): delete from emp where eid = 'e101';

1. Simple Queries using DDL & DML commands

i) Inventory

Select * from tablname;

DRL / DQL → Data retrieval language
Data Query language

1. Select → to fetch the needed data from the table

* To select all the data from the table

1. Select (*) (all data) from table name;

Ex: Select * from student;

2. Selecting particular record with all columns

select (*) (all data) from tablename where columnname = 'value';

Ex: select * from student where rollno = 'A21CSD033';

3. Selecting particular record with particular columns

Ex: select name, result from student where rollno = '53';

A: Select the particular column rollno = '53';
Select name, result from student;
(tablename)

Null Value

⇒ Missing value while inserting is called Null value

1- Insert into student value (Null, 'Arjun', 'cs', Null);

Distinct clause

⇒ To select unique values from a column or group of columns.

To select distinct value from single column

Ex: select distinct dept from college;

To select distinct value from group of columns

Ex: select distinct designation, branch from employee;

Order by clause

⇒ To display the table data or a column data in sorting orders (Ascending | Descending)

By default ⇒ Ascending

Select * from tablename orderby column name;
space

Ex:
- select * from emp orderby salary;
- " " student order by name;
- " " emp orderby gender desc;

desc → descending order

operators

1. Arithmetic operations [+, -, *, /]

⇒ To perform arithmetic operations.

Ex: select salary, salary*2 from emp;
- update student set total = (sub1+sub2+sub3);
- update student set average = (total/3);

2. Relational operators [=, <, >, <=, >=]

⇒ To compare one value against another value

- select * from employee where job = 'clerk';
- Select * from employee where salary > 2000;
- select Rollno, name from student where result = 'fail';
- select Rollno from student where total <=250

3. Special operators [between, in, is null, like]

⇒ Between - [can support numeric range, Data range, char range]

- Select * from emp where salary between 2000 And 5000;

⊙ select * from emp where join_data between '01-jan-2020' and '01-jan-2023';

⇒ IN → [To check in list of values]

⊙ select * from emp where designation in ('manager', 'clerk');

⊙ Select * from student where dept in ('cs', 'maths', 'B.com');

⇒ IS Null [to fetch the null value]

⊙ select * from emp where empid is NULL;

⊙ select * from student Where name is NULL;

Like → It is used to select the field values starting or ending with some alphabets

To select name ends with y

Select * from student where name like '%y';

To Select name starts with h

select * from student where name like 'h%';

To select name with character (a) in between (middle)

select * from student where name Came like '%a%';

Navigation operators [! = not in, is not null, not between]

select * from student where mark 1 is not full

logical operators [and, or]

select * from emp where designation = 'manager' and salary > 50000;

Select * from student where sex = 'M' or result = 'pass'; (see 22th Query on next page)

update student set result = 'pass' where ((sub1 >= 40) and (sub2 >= 40) and (sub3 >= 40));

update student set result = 'fail' where ((sub1 < 40) or (sub2 < 40) or (sub3 < 40));

%a% → Englishman (name with a will display)
a is here
it will display

Select * from Tablename
where columnname Trim(columnname) like 'a%';

1. Create a table with the following columns :-

Rollno name dob age gender Mark1 Mark2 Mark3

2. Insert 10 records (4 records for male, 4 " " female
all subject failed 1 female record
all " " 1 male record

3. Calculate Add column total, average, result.
update student set total = (m1 + m2 + m3)

4. Calculate the total for all students
average = (total/9);

5. " " average "

6. Select all the male students
select * from student where gender = 'male';

7. " female 'female'

8. " all the failed male 'male'

9. " all the failed female 'female' 'result=' fail';
select * from student where gender = 'male'

10. " all the passed male gender = 'female'
select * from student where and
all the passed female result = 'pass';

11. " "

12. Select the male student who failed in all subject
" " " = 'male'

13. " female " " " " "
select * from student where gender = 'female'
and m1 <= 40 and m2 <= 40 and m3 <= 40

14. Select the students name whose age is greater than 20
select * from student where age > 20;

15. name starts whose
select * from student with A where name like 'a%';

16. Selects the students where name ends with a
not working in sql select * from student where name like '%a';

17. " select * from student where dob between '01-jan-200' and '12-dec-2005';
from 1-Jan-2002 to 1-may-2002 dob are

18. Selects the students whose department is in cs, maths. Select * from student where department in ('cs', 'maths');

19. " " " " " rollno is in 6, 10 select * from student where m1 in ('100', '75', '85');
select * from student where rollno in ('06', '10', '45');

20. Arrange the records by name
select * from student order by name;

21. " " " rollno in descending order select * from student order by rollno desc;

22. Select the students who failed in only one subject Select * from student where (m2 <= 10 or m3 <= 10);

O/P

1. Create table student (Rollno varchar2 (3),
name char (10),
dob Date,
age Number (2),
gender char (6),
m1 number (3),
m2 number (3),
m3 number (3));

2. Insert into student values ('&rollno',
'&name',
'&dob',
&age,
'&gender',
&m1,
&m2,
&m3);

Enter the rollno :
Enter the name :
1 row created
Sql> / <

Type slash & hit enter to enter next row

Field name	Datatype	Size	Constraint
Emp_id			
name			
Gen			check
Department			
Basic pay			not null
DA			
HRA			
PF			
Gross pay			
net pay			

i) Add 5 record

ii) Select the employee whose name ending with 'a'

iii) Calculate Grosspay & netpay

iv) Select employe from maths dept.

v) Calculate tot. amount paid by organisation to employe

vi) Select employe whose grosspay above ₹50,000

vii) Select employe whose got lowest salary

Create a table Telephone with the following fields & Answer Queries

Field_name	Datatype	Size	constraint
customer_id			primary key
customer_name			
street			
city			
phone no			
bill_date			
Due_date			
bill_amount			
bill_status			

i) Add 10 records

ii) Select customer who paid the amount after payment date

iii) Select customer from respective city (eg. Thambaram)

iv) Select the details of the customer with phone no (23645 2)

v) Select the customer whose bill amount is less than Rs 500

Create a table Library with the following fields & answer the Questions

Field name	Datatype	Size	Constraint
Id_number			primary key not null
Title			
Author_name			
publisher_name			
year_of_publication			

i) Add 5 records
ii) Select books which published in year 2000
iii) Delete records with id_number (12345)
iv) Modify field length to 20 for the fieldname title
v) Add a new field no_of_pages numbr (6) at the end of all field.

Ex:2 SQL Aggregate functions

SQL count(): This function returns no of rows in the table that satisfies the conditions specified in the WHERE condition. If the world WHERE conditions is not specified, then the Query returns the total no. of rows in the table.

SQL Max(): This function is used to get maximum value from column.

SQL MIN(): " Minimum "

SQL Avg(): " Average "

SQL Sum(): This function is used to get sum of a numeric column

SQL distinct(): This function is used to Select the distinct rows

id	name	dept	age	Salary	location
100	Ramesh	Electrical	24	25000	bangalore
101	Hrithik	Electronics	28	35000	bangalore
102	Harsha	Aeronautics	28	35000	mysore
103	Soumya	Electronics	22	20000	bangalore
104	priya	Infotech	25	30000	bangalore

SQL > Select count (*), FROM empl where dept = 'Electronics';

count (*)
2

SQL > Select MAX (Salary) FROM empl;

MAX (Salary)
35000

SQL > Select MIN (Salary) FROM empl;

MIN (Salary
20000

SQL > Select AVG (Salary) FROM emp);

AVG (Salary)
29000

SQL > Select SUM (Salary) FROM empl;

SUM (Salary)
145000

SQL > Select DISTINCT dept FROM empl;

DEPT
Aeronautical
electronics
Infotech
Electrical

Ex: 3 Set Operations

1. Union all - combines the result of two Select statements into one result set
2. Union - combines the result of two Select statements into one result set and then eliminates any duplicate rows from the result set
3. Minus - takes the result set of one Select statement and remove those rows that are also returned by a second select statement.
4. Intersect - returns only those rows that are returned by each of two Select statements.

SQL> create table sales2009 (person varchar2(10), Amount number(10));

Table created.

SQL> Select * from sales2009;

PERSON	AMOUNT
Rahul	7000
Babu	2000
Joe	5000
Chandru	3000

SQL> create table sales2008 (person varchar2(10), Amount number(10));

Table created.

SQL> Select * from sales2008;

PERSON	AMOUNT
Joe	1000
Surendar	6000
Vijay	6500
Rahul	7000

UNION

```
SQL> Select * from Sales_2008
     union
     Select * from sales_2009;
```

2 same records name with same data ↓ consider as one data

2 same records name with different data ↓ consider as different records There will be 2 records

PERSON	AMOUNT
Rahul	7000
Babu	2000
Joe	5000
Chandru	3000
Joe	1000
Surendar	6000
Vijay	6500

UNION ALL

(return the repeated record too with same data)

```
SQL> Select * from Sales_2008 unionall
     Select * from Sales2009;
```

PERSON	AMOUNT
Rahul	7000
Babu	2000
Joe	5000
Chandru	3000
Joe	1000
Surendar	6000
Vijay	6500
Rahul	7000

create table Sales_2008 as select * from Sales 2009 where 1 = 2 ;

with structure only

with data & structure

MINUS

```
SQL> Select * from Sales2008 minus
     Select * from Sales 2009
```

PERSON	AMOUNT
Joe	1000
Surendar	6000
Vijay	6500

INTERSECT

(name & data should be same) record

```
SQL> Select * from Sales2008 Intersect
     Select * from 2009
```

o/p

PERSON	AMOUNT
Rahul	7000

2008

Ex:4 Views and Snapshots

The syntax to create a SQL view is

```
Create view view-name
AS
SELECT column-list
FROM table-name [WHERE condition]
```

⇒ view name is the name of the view.

⇒ The SELECT statement is used to define the columns and rows that you want to display in the view

Table creation

SQL> select * from child;

EID	ITEMNO	ITEMNAME	AMOUNT	DATEODT
100	5	Pen	26	20-DEC-00
200	6	Pencil	13	21-Dec-01
201	4	Eraser	5	12-Jan-00
202	5	Gum	23	12-Dec-00
203	6	paper	23	17-Dec-00

Creation of view :

Viewname

SQL> create view childview as select * from child where & amount>20;

view created

view table:

SQL> select * from childview;

EID	ITEMNO	ITEMNAME	AMOUNT	DATEODT
100	5	pen	26	20-Dec-00
202	5	Gum	23	12-Dec-00
203	6	paper	23	17-Dec-00

Updating the view :-

SQL> update childview set itemno='7' where eid='202';

1 row created

updated view Table :-

SQL> select * from childview;

EID	ITEMNO	ITEMNAME	AMOUNT	DATEODT
100	5	pen	26	20-Dec-00
202	7	gum	23	12-Dec-00
203	6	paper	23	17-Dec-00

(1 5)

Updated Master Table :-

SQL> select * from child;

EID	ITEMNO	ITEMNAME	AMOUNT	DATEODT
100	5	Pen	26	20-Dec-00
200	6	Pencil	13	21-Dec-01
201	4	Eraser	5	12-Jan-00
202	7	gum	23	12-Dec-00
203	6	paper	23	17-Dec-00

Result :-

The view and snapshot are created successfully.

exit

connect system

sys

Grant create view to username;

Grant succeeded

connect

pass vishvajit

done.

if you're inserting or deleting a record into the child table (2nd table) it will reflect on the master table (1st table)

Group by clause

Rollno	name	age	sex	dept	total
01	-	21	M	CS	460
02	-	22	F	CS	450
03	-	20	M	maths	440
04	-	21	F	maths	350
05	-	22	M	B.com	320
06	-	20	F	CS	300

1. Select no of male & Female student

Select sex, count (*) from student

group by sex

sex	count
M	3
F	3

Syntax :- Select columnname from tablename where <conditions>

[Groupby clause]
[having clause]
[Orderby clause]

Square bracket [optional functions] everything comes this order only

2. Select no. of male & Female student in each dept.

Select dept, sex, count (*) from student

groupby dept, sex

(Space)

dept	sex	count
CS	M	1
CS	F	2
Maths	M	1
Maths	F	1
Bcom	M	1
B-com	F	0

Result :-

The view and snapshot are created successfully.

exit

connect system

sys

Grant create view to <username>;

Grant succeeded

connect <username>/<password>;

pass vishvajit

done.

if you're inserting or deleting a record into the child table (2nd table) it will reflect on the master table (1st table)

Groupby clause

Rollno	name	age	sex	dept	total
01	-	21	M	CS	460
02	-	22	F	CS	450
03	-	20	M	maths	440
04	-	21	F	maths	350
05	-	22	M	B.com	320
06	-	20	F	B.com	300

1. Select no of male & Female student

Select sex, count (*) from student

group by sex

sex	count
M	3
F	3

Syntax :- Select columnname from tablename where <condition>

[Groupby clause]
[having clause]
[Orderby clause]

Square brackets [optional functions] — everything comes this order only

2. Select no. of male & Female student in each dept.

Select dept, sex, count (*) from student

groupby dept, sex

(Space)

dept	sex	count
CS	M	1
CS	F	2
Maths	M	1
Maths	F	1
Bcom	M	1
B.com	F	0

having clause :-

```
Select , dept, sex , count(*) from
student having count(*) >= 1
```

dept	Sex	count
CS	M	1
maths	M	1
maths	F	1
B.com	M	1
B.com	F	1

Orders by :-

```
Select , dept, sex , count(*) from
student having count(*)>=1 order by
dept;
```

dept	Sex	count
B.com	M	1
B.com	F	1
CS	M	1
maths	M	1
maths	F	1

Ex: 7 Function & procedures

Table creation

```
create table stud (regno number(3),
                   m1 number(2),
                   m2 number(2),
                   m3 number(2),
                   result varchar2(6));
```

Function creation (return value)

```
create or replace function fun1( m1 number,
                                 m2 number,
                                 m3 number )
                                 return varchar2 as
Begin
if((m1>40) and (m2>40) and (m3>40)) then
return 'pass';
else
return 'Fail';
end if;
end;
/ slash
```

procedure creation (not return value)

```
create or replace procedure pro1 (regno number,
                                  m1 number,
                                  m2 number,
                                  m3 number)
res stud.result%type;
begin
res := fun1(m1, m2, m3);
insert into stud values(regno, m1, m2, m3, res);
end;
/ slash
```

as

get the datatype of result from the stud table and it assign the datatype to res.

Main program

Take / consider the datatype of the Stud's Table's datatype

```
declare
regno   stud.regno %type;
m1      stud.m1 %type;
m2      stud.m2 %type;
m3      stud.m3 %type;
begin
regno := &regno;   <- get the value from the user
m1 := &m1;
m2 := &m2;
m3 := &m3;
pro1(regno, m1, m2, m3);
end;
```

Output

```
Enter the value of regno : 53
old 4: regno := &regno
new 4: regno := 53
Enter the value for m1 : 66
old 8: m1 := &m1;
new 8: m1 := 66;
Enter the value for m2 : 44
old 9: m2 := &m2;
new 9: m2 := 44;
Enter the value for m3 : 88
old 10: m3 := &m3;
new 10: m3 := 88;
```

Pl/SQL procedure successfully completed

Select * from stud;

regno	M1	M2	M3	Result
53	66	44	88	pass

Ex:8 Sub programs & Packages

```
SQL> create table stud (regno number(2)
                        m1 number(2)
                        m2 number(2)
                        m3 number(2)
                        result varchar2(6));
```

package

```
create or replace store as
function fun1 (m1 number,
               m2 number,
               m3 number) return varchar2;
procedure single (regno number);
procedure double (regno number, m1 number);
procedure final (regno number, m1 number,
                 m2 number, m3 number,
                 result varchar2);
end store;
```

package body

```
create or replace package body store as
function fun1 (m1 number,
               m2 number
               m3 number ) return varchar2
                                        as
begin
if ((m1>40) and (m2>40) and (m3>40)) then
return 'pass';
else
return 'fail';
end if;
end;
```

```
procedure single (regno number) is
begin
insert into stud (regno) values (regno);
end;
Procedure double (regno, m1) values (regno, m2)
is
begin
insert into stud (regno, m1) values (regno,
                                     m2);
end;
Procedure final (regno number,
                 m1 number,
                 m2 number,
                 m3 number,
                 result varchar2 ) is
begin
insert into stud values (regno, m2, m2,
                         m3, result);
end;
end store;
```

Main program:

```
declare
regno stud.regno%type;
m1 stud.m1%type;
m2 stud.m2%type;
m3 stud.m3%type;
result stud.result%type;
begin
regno := &regno;
m1 := &m1
m2 := &m2
m3 := &m3
store.single (regno);
store.double (regno, m2);
result := store.fun1 (m1, m2, m3);
store.final (regno, m1, m2, m3, result);
end
```

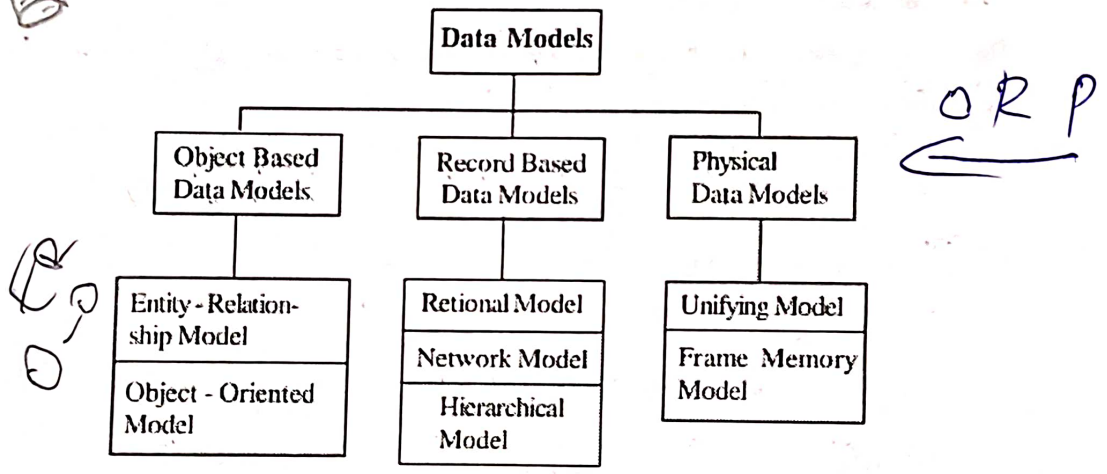
Output:

```
Enter value for regno : 888
old 9: regno := &regno
new 9: regno := 888;
Enter value for m1 : 89
old 10: m1 := &m1,
new 10: m1 := 89
Enter value for m2: 78
old 10: m2 := &m2
new 10: m2 := 78
Enter value for m3 : 95
old 12: m3 := &m3
new 12: m3 := 95
```

o/p

Data Models

- Data Model is the conceptual representation of how data is organized, stored and accessed within a database. In other words, Planning the structure of database is called Data models.
- Data model provides a blueprint that helps developers and designers understand how data should be organized and related to each other.
- It helps to highlight any drawbacks of the plan and correct it at the initial stage itself.
- It gives the idea about how the final system will look like after implementation.



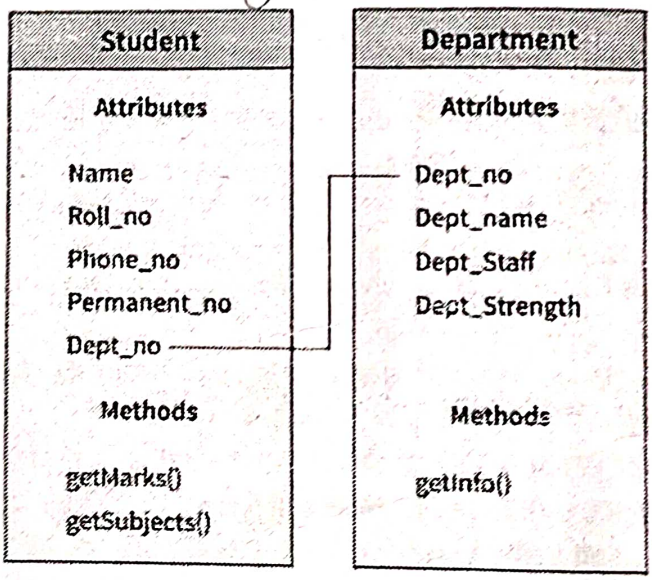
There are three different categories in data models

1. Object-based models
2. Record-based models
3. Physical data models.

mapping cardinality. (One-One, One-Many, ... Many-Many)

ii) The Object-Oriented Model

- The Object-Oriented Model in DBMS is the data model where data is stored in the form of objects.
- This model is used to represent real-world entities.
- The data and data relationship are stored together in a single entity known as an object.
- The components of the Object-Oriented Data Model are classes, objects, attributes, methods, and inheritance.



Here Student and Department are two different objects. Each one of them has its attributes and methods.

They are linked by a common attribute Dept_no which establishes a relationship between objects.

2) Record-Based Models

Record-based logical models are used in describing data at the logical and view levels.

The three most widely accepted data models are

i) The Relational Model

ii) The Hierarchical Model

iii) The Network Model

i) The Relational Model

- This model is introduced by Dr.E.F.Codd in 1969.
- The relational model represents data and relationships among data by a collection of tables, each of which has a number of columns and rows.
- In the relational model, relationships between tables are created by using keys.
- The use of tables to store the data provided a straightforward, efficient, and flexible way to store and access structured information.
- Because of this simplicity, this data model is most widely used.

Customer database

customer-name	*social-security*	*customer-street*	*Customer-city*	*account-number*
Johnson	192-83-7465	Alma	Palo Alto	A-101
Smith	019-28-3746	North	Rye	A-215
Hayes	677-89-9011	Main	Harrison	A-102
Turner	182-73-6091	Putnam	Stamford	A-305

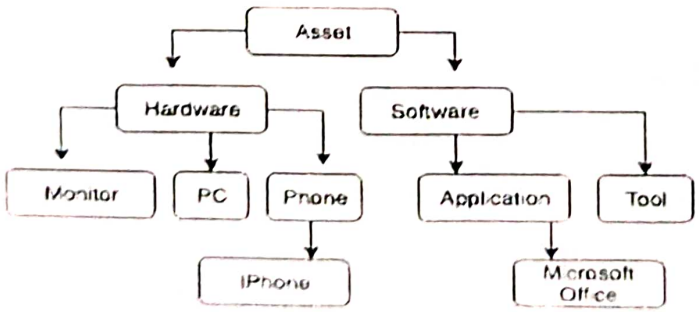
Account database

account-number	*Balance*
A-101	500
A-215	700
A-102	400
A-305	350
A-201	900
A-217	750
A-222	700

A sample Relational database

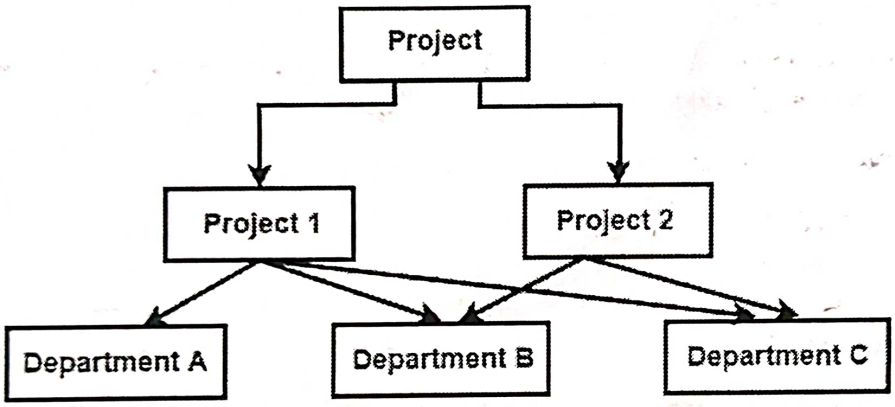
ii) Hierarchical Model

- This was the first DBMS model.
- It organizes the data in the Hierarchical Tree structure. The Hierarchy starts from the Root node and expands in the form of Tree, adding Child node to the Parent node.
- Pointers are used to connect the Parent and Child node. Each Child node can have only one Parent node, but a Parent node can have more than one child node.
- Any modification to the Parent is automatically reflected in Child node ensuring Data Integrity but when a Parent node is removed the Child node will also be removed.



iii) Network Model

- Network model is an extension of hierarchical model. Data in the network model are represented in the form of arbitrary graphs.
- It supports many to many relationships. A parent node can have many children and a child node can have more than one parent node.
- Network models represent complex data relationships better than the hierarchical model.
- Data access is more flexible than hierarchical models.
- Any change like updating, deletion, insertion is very complex.

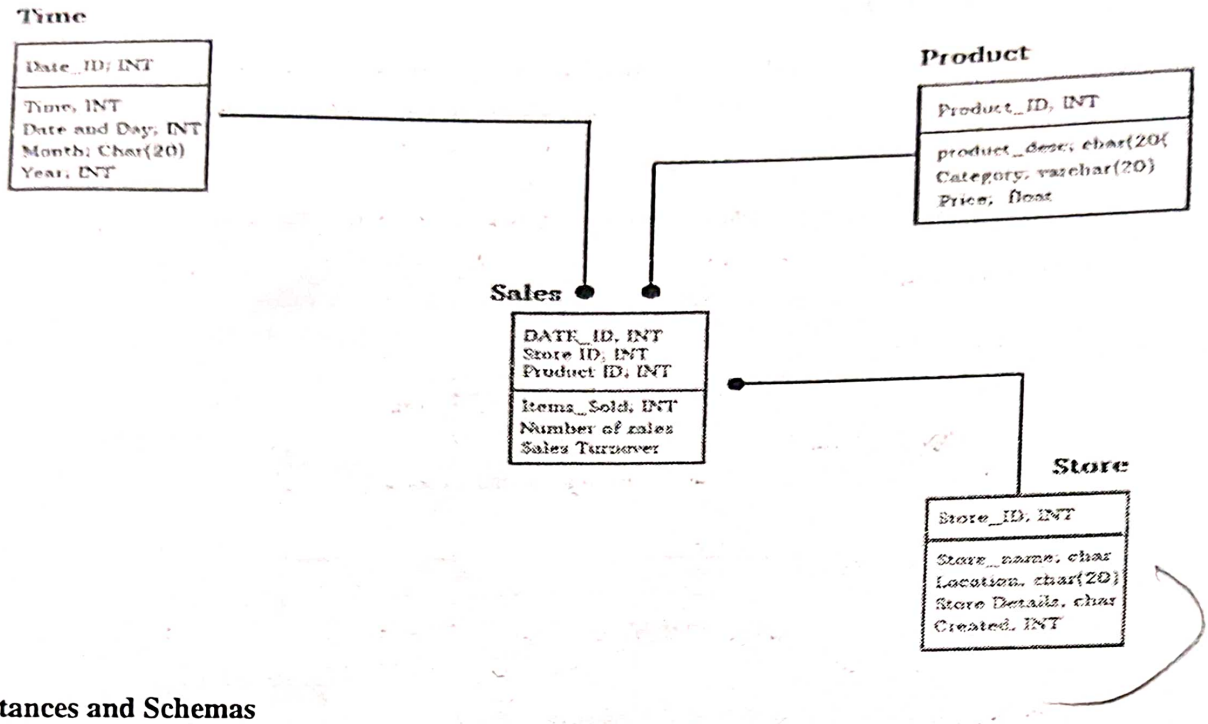


3) Physical Data Model

- Physical data models are used to describe data at the lowest level i.e., in the Physical Level.
- It describes how data is stored in computer memory, how data is ordered in the memory and how they could be retrieved from memory.
- Physical data model will be different for different RDBMS like MY SQL, SQL Server.
- Two of the widely known ones are:

i) Unifying model

ii) Frame memory



Instances and Schemas

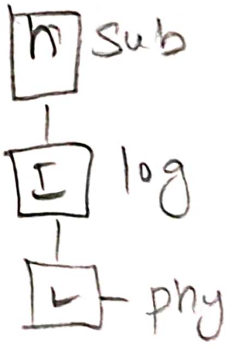
• The collection of information stored in the database at a particular moment in time is called an **instance** of the database.

• The overall design of the database is called the database **schema.**

• Schemas are changed infrequently.

The concept of a database **schema** corresponds to the programming language notion of **type definition**. A variable of a given type has a particular value at a given instant in time.

Thus, the concept of the value of a **variable** in programming languages corresponds to the concept of an **instance** of a database schema.



Database systems have several schemas, partitioned according to the levels of abstraction.

The lowest level is the physical schema;

The intermediate level is the logical schema;

The highest level is a subschema.

The database systems support one physical schema, one logical schema, and several subschemas.

Var → Particular value
(Type) Schema
At a moment / instant

Buffer manager: which is responsible for fetching data from disk storage into main memory,and deciding what data to cache in memory.

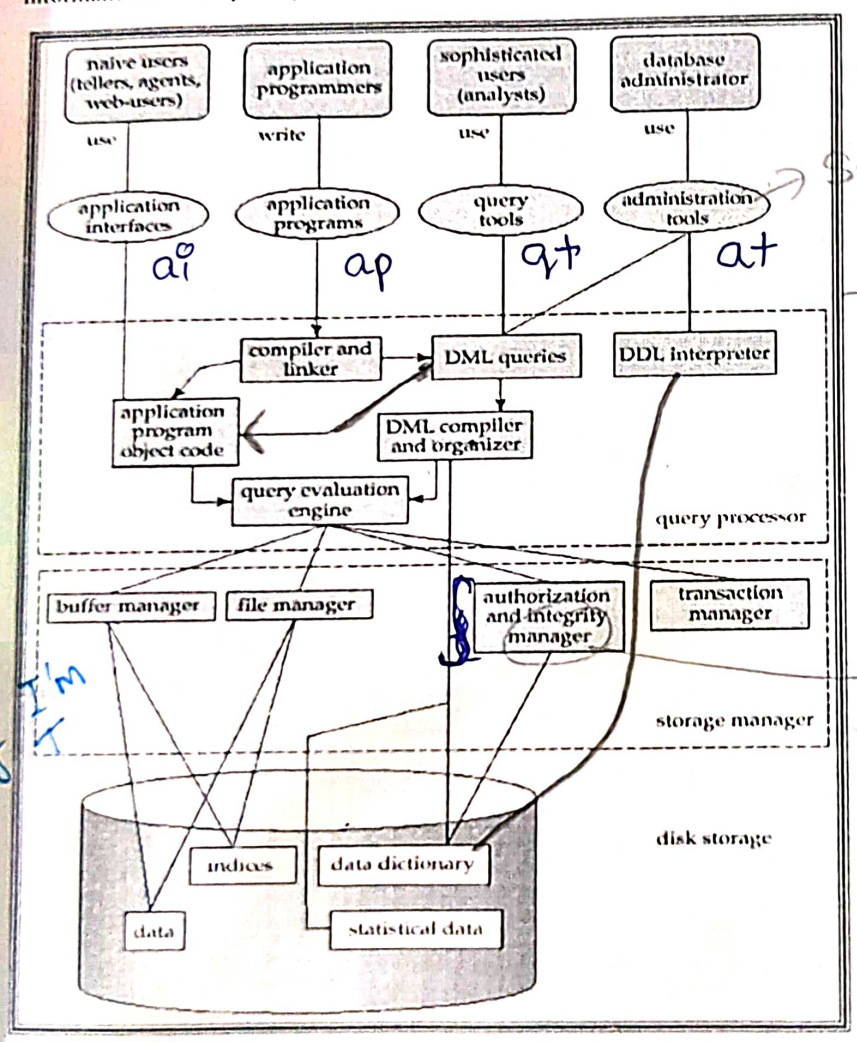
Several data structures are required as part of the physical system implementation:

Data files: which store the database itself.

Data dictionary: which, stores metadata about the structure of the database. The data dictionary is used heavily.

Indices: which provide for fast access to data items holding particular values.

Statistical data: which stores statistical information about the data in the database. This information is used by the query processor to select efficient ways to execute a query



Entity-Relationship

The entity-re
entities, and *relation*

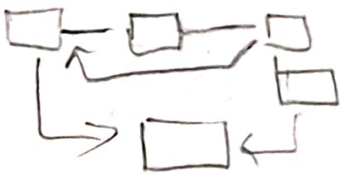
Entities and Entity

An ***entity*** is
An ***entity s***

Types of E

An entity

- Strong ent
- Weak enti



Stron

1. Stron

- A strong identif
- In other
- Primary

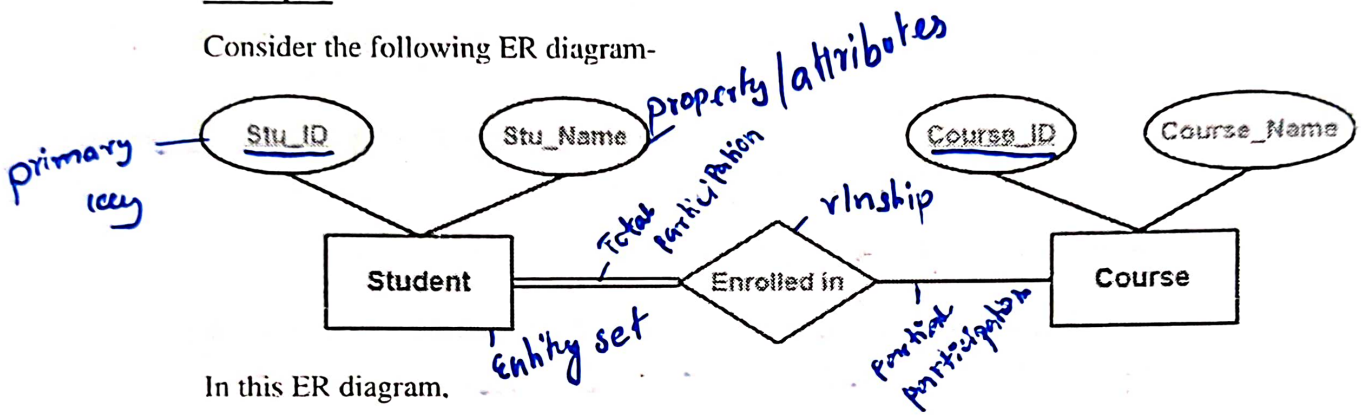
Symbo

A single rectan

- A diamo strong
- A single relati
- A doub relati
- Total p

Example-

Consider the following ER diagram-



In this ER diagram,

- Two strong entity sets "**Student**" and "**Course**" are related to each other.
- Student ID and Student name are the attributes of entity set "Student".
- Student ID is the primary key using which any student can be identified uniquely.
- Course ID and Course name are the attributes of entity set "Course".
- Course ID is the primary key using which any course can be identified uniquely.
- Double line between Student and relationship set signifies total participation.
- It suggests that each student must be enrolled in at least one course.
- Single line between Course and relationship set signifies partial participation.
- It suggests that there might exist some courses for which no enrollments are made.

2. Weak Entity Set-

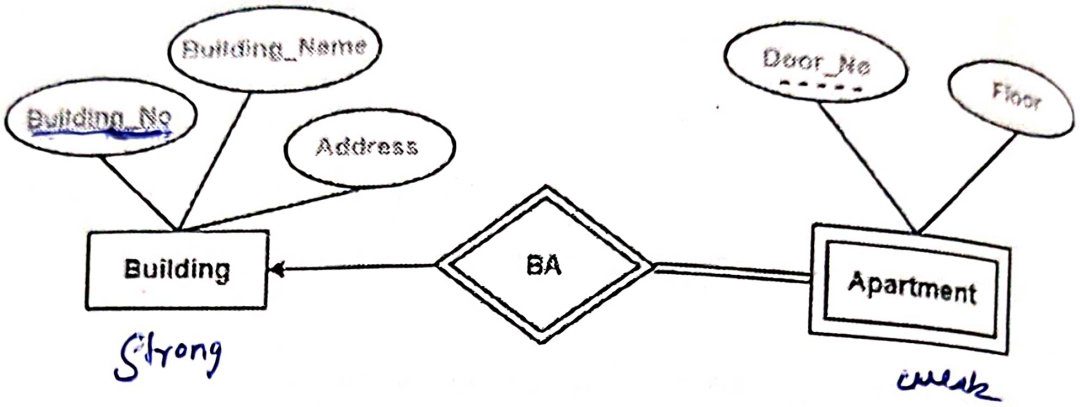
- A weak entity set is an entity set that does not contain sufficient attributes to uniquely identify its entities.

- In other words, a primary key does not exist for a weak entity set.
- However, it contains a partial key called as a **discriminator.**
- Discriminator can identify a group of entities from the entity set.
- Discriminator is represented by underlining with a dashed line.

Symbols Used-

- A double rectangle is used for representing a weak entity set.
- A double diamond symbol is used for representing the relationship that exists between the strong and weak entity sets and this relationship is known as **identifying relationship**.
- A double line is used for representing the connection of the weak entity set with the relationship set.
- Total participation always exists in the identifying relationship.

Example-



In this ER diagram,

- One strong entity set "**Building**" and one weak entity set "**Apartment**" are relate to each other.
- Strong entity set "Building" has building number as its primary key.
- Door number is the discriminator of the weak entity set "Apartment".
- This is because door number alone can not identify an apartment uniquely as the may be several other buildings having the same door number.
- Double line between Apartment and relationship set signifies total participation.
- It suggests that each apartment must be present in at least one building.
- Single line between Building and relationship set signifies partial participation.
- It suggests that there might exist some buildings which has no apartment.

Oliver	654-32-1098	Main	Austin	259	1000
Harris	890-12-3456	North	Georgetown	630	2000
Marsh	456-78-9012	Main	Austin	401	1500
Pepper	369-12-1518	North	Georgetown	700	1500
				199	500
				467	900
				115	1200
				183	1300

Relationships and Relationship Sets

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

Example-

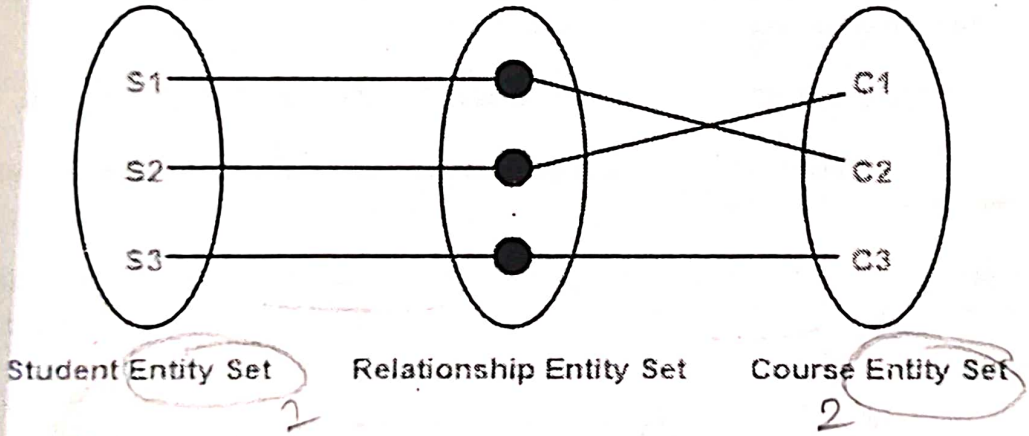
'Enrolled in' is a relationship that exists between entities **Student** and **Course**.

Relationship Set-

A relationship set is a set of relationships of same type.

Example-

Set representation of above ER diagram is-



Degree of a Relationship Set-

The number of entity sets that participate in a relationship set is termed as the degree of that relationship set.

Types of Relationship Sets-

On the basis of degree of a relationship set, a relationship set can be classified into the following types-

1. Unary relationship set
2. Binary relationship set
3. Ternary relationship set
4. N-ary relationship set

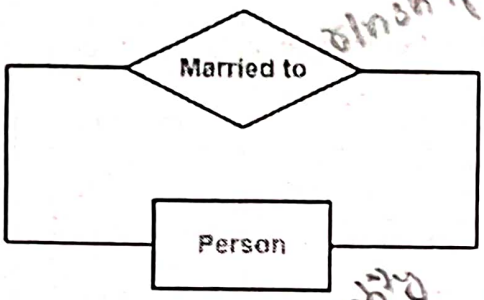
Participate

1. Unary Relationship Set-

Unary relationship set is a relationship set where only one entity set participates in a relationship set.

Example-

One person is married to only one person



relationship

Degree = 1

entity set

2. Binary Relationship Set-

Binary relationship set is a relationship set where two entity sets participate in a relationship set.

Example-

Student is enrolled in a Course

relationship

entity set

entity set

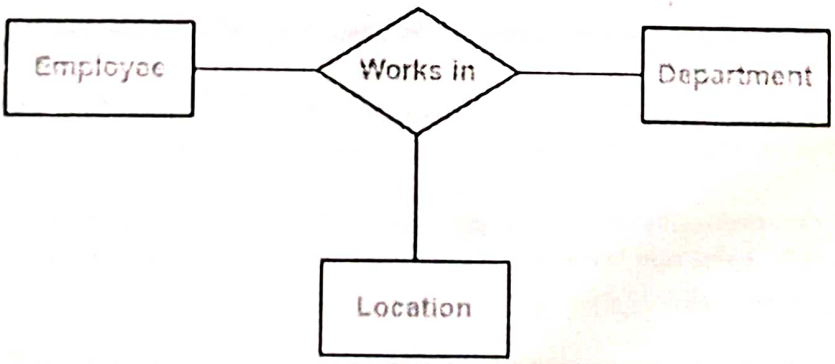
Degree = 2

3. Ternary Relationship Set-

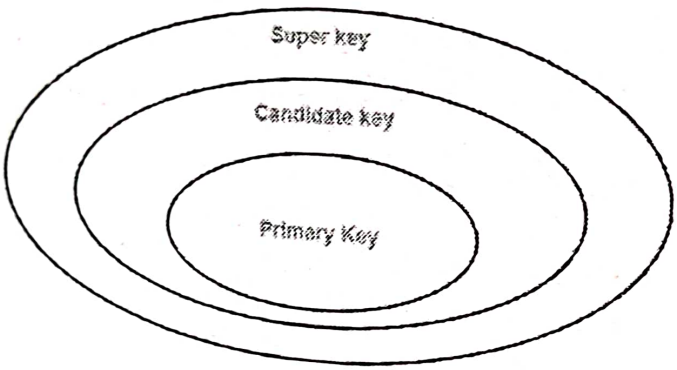
Ternary relationship set is a relationship set where three entity sets participate in a relationship set.

Example-

Degree = no of entity set



Degree = 3

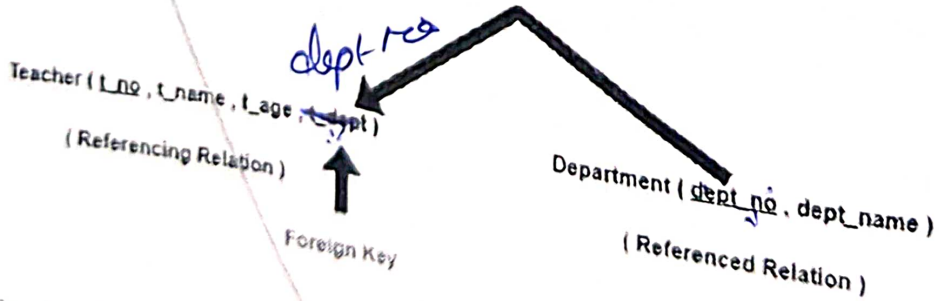


Foreign Key-

A foreign key is a column or set of columns in one table that refers to the primary key of another table.

Example-

Consider the following two schemas-



Foreign key can take only those values which are present in the primary key of the referenced relation.

- Foreign key can take the NULL value.
- There is no restriction on a foreign key to be uni
- Referenced relation may also be
- Referencing relation

UNIT – III

Introduction:

The inputs and outputs of a query are relations. A query used to evaluate using instances of each input relation and it produces an instance of the output relation.

- Query language can be categorized as procedural or nonprocedural.
- In nonprocedural language, the user describes the information desired without giving a specific procedure for obtaining that information (ex: relational calculus)
- In procedural language, the user instructs the system to perform a sequence of operations on the database to compute the desired result (ex: relational algebra).

Relational-Algebra Operations:

The relational algebra is a procedural query language. It consists of a set of operations that take one or two relations as input and produce a new relation as their result.

The Select operation:

The Select operation selects tuples that satisfy a given predicate.

We use the lowercase Greek letter sigma (σ) to denote selection.

The predicate appears as a subscript to . The argument relation is in parenthesis after the

Thus, to select those tuple of the loan relation where the branch is "perryridge", we write,

$$\sigma_{branch\_name = \text{“perryridge”}}(loan)$$

Loan relation

Loan_number	Branch_name	Amount
L-11	Round hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000

The result of preceding query is,

Loan_number	Branch_name	Amount
L-15	Perryridge	1500
L-16	Perryridge	1300

We can find all tuples in which the amount is more than 1200,

$$\sigma_{amount > 1200}(loan)$$

domain (datatype)
no. of attributes should be same for doing union on both tables

$$\Pi_{customer\_name\ (borrower)} \cup \Pi_{customer\_name\ (depositor)}$$

The depositor relation

Customer_name	Account_number
Smith	A-102
John	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Brooks	A-305

Borrower relation

Customer_name	Loan_number
Jones	L-16
Smith	L-93
Williams	L-15
Johnson	L-14
Smith	L-17
John	L-11

The result relation for Union query is,

Customer_name
Smith
John
Johnson
Jones
Lindsay
Brooks
Williams

only valid combination

The Set-Difference Operation:

The Set-difference operation, denoted by _ , allows us to find tuples that are in one relation but are not in another.

The expression r-s produces a relation containing those tuples in r but not in s.

We can find all customers of the bank who have an account but not a loan by writing.

$$\Pi_{customer\_name\ (depositor)} - \Pi_{customer\_name\ (borrower)}$$

The result relation for this query is,

depositor relation

Customer_name	Account_number
Smith	A-102
John	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Brooks	A-305

Borrower relation

Customer_name	Loan_number
Jones	L-16
Smith	L-93
Williams	L-15
Johnson	L-14
Smith	L-17
John	L-11

Customer_name
Lindsay
Brooks

relation

e	Loan_number
	L-16
	L-93
	L-15
	L-14
	L-17
	L-11

les that are in one

but not in s.

ut not a loan by

Customer_name
Lindsay
Brooks

gives all possible combination (have valid & invalid combination).

The Cartesian-Product Operation: / cross product / cross Joint

The Cartesian-Product operation, denoted by a cross(X), allows us to combine information from any two relations.

We write the Cartesian product of relations r_1 and r_2 as $r_1 X r_2$.

The same attribute name may appear in both r_1 and r_2, we need to devise a naming schema to distinguish between these attributes.

For example, the relation schema for r = borrower X loan is (borrower.customer_name, borrower.loan_number, loan.loan_number, loan.branch_name, loan.amount).

Borrower relation

Customer_name	Loan_number
Adams	L-16
Jones	L-17
Smith	L-23

Loan relation

Loan_number	Branch_name	Amount
L-23	Round hill	900
L-17	Downtown	1000
L-16	Perryridge	1300

The result of borrower X loan is

(if the two column name is same; then it'll add the Table name with it)

Customer_name	borrower.loan_number	loan.loan_number	Branch_name	Amount
Adams	L-16	L-23	Round hill	900
Adams	L-16	L-17	Downtown	1000
Adams	L-16	L-16	Perryridge	1300
Jones	L-17	L-23	Round hill	900
Jones	L-17	L-17	Downtown	1000
Jones	L-17	L-16	Perryridge	1300
Smith	L-23	L-23	Round hill	900
Smith	L-23	L-17	Downtown	1000
Smith	L-23	L-16	Perryridge	1300

only valid combination

If we want to find the names of all customer who have a loan at the "Perryridge" branch. If we write,

$$\sigma_{branch\_name = \text{"Perryridge"}}(borrower\ X\ loan)$$

Customer_name	borrower.loan_number	loan.loan_number	Branch_name	Amount
Adams	L-16	L-16	Perryridge	1300
Jones	L-17	L-16	Perryridge	1300
Smith	L-23	L-16	Perryridge	1300

The Rename Operation:

The results of relational algebra expressions do not have a name that we can use to refers to them. It is useful to be able to give them names.

The rename operator, denoted by the lowercase Greek letter rho (ρ).

A relation credit_info, which lists the credit limit and credit_balance on the account.

If we want to find how much more each person can spend, we can write the following expression,

$\Pi_{customer\_name, limit - credit\_balance}(credit\_info)$.

Credit_info relation

Customer_name	Limit	Credit_balance
Curry	2000	1750
Hayes	1500	1500
Jones	6000	700
Smith	2000	400

The attribute resulting from the expression limit - credit_balance does not have a name.

We can apply the rename operation to the result of generalized projection in order to give it a name.

As a notational convenience, remaining of attributes can be combined with generalized projection,

$\Pi_{customer\_name, (limit - credit\_balance)\ as\ credit\_available}(credit\_info)$.

The result of $\Pi_{customer\_name, (limit - credit\_balance)\ as\ credit\_available}(credit\_info)$

Customer_name	Credit_available
Curry	250
Jones	5300
Smith	1600
Hayes	0

Aggregate Functions:

Aggregate functions take a collection of values and return a single value as a result. For example, The aggregate function **sum** takes a collection of values and returns the sum of the values. Thus, the function sum applied on the collection, {1, 1, 3, 4, 4, 11} returns the value 24.

The aggregate function **avg** returns the average of the values, it returns the value 4.

The aggregate function **count** returns the number of the elements in the collection and returns 6.

Other common aggregate functions include **min** and **max**, which return the minimum and maximum values in a collection, they return 1 an 11 respectively.

The collections on which aggregate functions operate can have multiple occurrences of a value; the order in which the values appear in not relevant. Such collections Are called **multisets**.

Sets are a special case of multisets where there is only one copy of each element.

Pt_works relation

Employee_name	Branch_name	Salary
Adams	Perryridge	1500
Brown	Perryridge	1300
Gopal	Perryridge	5300
Johnson	Downtown	1500
Loreena	Downtown	1300
Peterson	Downtown	2500
Rao	Austin	1500

If we want to find out the total sum of salaries of all the part-time employees in the bank. The relational-algebra expression for this query,

$$G_{sum(salary)}(pt\_works)$$

The symbol is the letter G in calligraphic font.

The result of the expression above is a relation with a single attribute, containing a single row with a numerical value corresponding to the sum of all the salaries of all employees.

$$G_{count\text{-}distinct(branch\_name)}(pt\_works)$$

We want to find tee total salary sum of all part_time employees at each branch of the bank separately rather than the sum for the entire bank.

We need to partition the relation pt_works in to groups based on the branch and to apply the aggregate function on each group.

The pt_works relation after grouping

Employee_name	Brance_name	Salary
Rao	Austin	1500
Sat		
Johnson	Downtown	1500
Loreena	Downtown	1300
Peterson	Downtown	2500
Adams	Perryridge	1500
Brown	Perryridge	1300
Gopal	Perryridge	5300

$$branch\_name\ G_{sum(salary)}(pt\_works)$$

In this expression, the attribute branc_name indicates the input relation pt_works must be divided in to groups based on the value of branch name.

then

$1200:

0:

hat may generate

lues in the result

bout the form of

UNIT – IV

Normalization

What is Normalization?

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from the database table.
- It is also used to eliminate Insertion, Update, and Deletion Anomalies.
- Normalization divides the larger table into smaller and links them using relationships.

Insertion Anomaly: Insertion Anomaly refers to the situation, when one cannot insert a new record into a table due to lack of data.

Deletion Anomaly: The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some other important data.

Updation Anomaly: The update anomaly is when an update of a single data value requires multiple rows of data to be updated.

Advantages of Normalization

- Normalization helps to minimize data redundancy.
- Improved Data consistency within the database.
- Enforces the concept of relational integrity.
- Efficient Database Design.
- Improved database performance.
- Much more flexible database design.
- Less Storage Space

First Normal Form (1NF) write example (in exam) Refer note

First Normal Form (1NF) is a fundamental concept in database design that ensures the elimination of duplicate data and enforces atomicity (each attribute contains only indivisible, single-value data). To satisfy 1NF, a table must meet the following criteria:

- **Atomic Values:** Each attribute (column) in a table should contain indivisible values. If an attribute contains multiple values, it should be split into separate attributes.

data.
Functions can modify individual data items.
Functions can very easily manipulate output for group of rows.
Functions can alter date formats for display.

Types of function:
Character functions
Arithmetic functions
Other functions
Date functions
Aggregate /group functions
Conversion functions

5.3 STRING FUNCTION

These function all take arguments in the character family and return character values. The majority of the functions return a varchar2values, except where noted. The return type of character function is subject to the same restrictions as the base database type, namely that varchar2 values are limited to 2000 characters and char values are limited to 255 characters. When in procedural statements, they can be assigned to either varchar2 or char pl/sql variables.

Types:
CHR(X)– returns the charac that has ASCII code value specified by the arg
CONCAT(STRING1,STRING2)
LOWER(STRING)
UPPER(STRING)
LPAD(CHAR1,N[CHAR2])– left pads a string with another string to certain length.
RPAD(CHAR1,N[CHAR2]) – right
SOUNDEX(STRING)– converts string to a 4 charac code based on how it sounds
LTRIM(STRING,'CHAR/S')– removes all ~~pads~~ found on left hand side of a string
RTRIM(STRING,'CHAR/S') right
REPLACE(STRING,SEARCH_STR[,REPLACE_STR])
SUBSTR(STRING,M[,N])
TRANSLATE(STRING,FROM_STR,TO_STR) ASCII(STRING)
INSTR(STRING,CHAR)
LENGTH(STRING)

SQL> create table emp(eno number,ename varchar2(15),address varchar2(10));

select *from emp;

ENO	ENAME	ADDRESS
1	ram	delhi
2	varun	chennai
3	ravi	banglore
4	amir	delhi

SQL> create table dept(dno varchar2(3),dname varchar2(15),eno number(5));

select *from dept;

DNO	DNAME	ENO
d1	hr	1
d2	it	2
d3	mrkt	4

CROSS JOIN

SQL> select* from emp,dept;

or

SQL> select* from emp cross join dept;

Output:

emp. dept.

ENO	ENAME	ADDRESS	DNO	DNAME	ENO
1	ram	delhi	d1	hr	1
2	varun	chennai	d1	hr	1

3	ravi	banglore	d1	hr	1
4	amir	delhi	d1	hr	1
1	ram	delhi	d2	it	2
2	varun	chennai	d2	it	2
3	ravi	banglore	d2	it	2
4	amir	delhi	d2	it	2
1	ram	delhi	d3	mrkt	4
2	varun	chennai	d3	mrkt	4
3	ravi	banglore	d3	mrkt	4
4	amir	delhi	d3	mrkt	4

12 rows selected.

INNER JOIN

1. **THETA JOIN** (From cartesian) we comparing

SQL> select* from emp,dept where emp1.eno>dept.eno;

or

SQL> select* from emp e,dept d where e.eno>d.eno;

ENO	ENAME	ADDRESS	DNO	DNAME	ENO
2	varun	chennai	d1	hr	1
2	varun	chennai	d1	hr	1
3	ravi	banglore	d1	hr	1
4	amir	delhi	d1	hr	1
3	ravi	banglore	d2	it	2
4	amir	delhi	d2	it	2

2. EQUI JOIN (From cartesian) we comparing

SQL> select* from emp,dept where emp.eno=dept.eno;

SQL> select* from emp e,dept d where e.eno=d.eno;

ENO	ENAME	ADDRESS	DNO	DNAME	ENO
1	ram	delhi	d1	hr	1
2	varun	chennai	d2	it	2
4	amir	delhi	d3	mkrt	4

3. NATURAL JOIN (no. cartesian)

select* from emp natural join dept ;

ENO	ENAME	ADDRESS	DNO	DNAME
1	ram	delhi	d1	hr
2	varun	chennai	d2	it
4	amir	delhi	d3	mkrt

4. SELF JOIN (from cartesian like comparing alice names)

SQL> select * from emp e,emp f where e.ename= f.ename;

ENO	ENAME	ADDRESS	ENO	ENAME	ADDRESS
1	ram	delhi	1	ram	delhi
2	varun	chennai	2	varun	chennai
3	ravi	banglore	3	ravi	banglore
4	amir	delhi	4	amir	delhi

Here also cartesian
emp.e x emp.f product
↓
16 rows

when comparing
emp.e = emp.f
4 values comparing

SQL> select * from emp e,emp f where e.eno< f.eno

ENO	ENAME	ADDRESS	ENO	ENAME	ADDRESS
1	ram	delhi	2	varun	chennai
1	ram	delhi	3	ravi	banglore
1	ram	delhi	4	amir	delhi
2	varun	chennai	3	ravi	banglore
2	varun	chennai	4	amir	delhi
3	ravi	banglore	4	amir	delhi

6 rows selected.

OUTER JOIN

1. LEFT OUTER JOIN

```
SQL> select * from emp1 left outer join dept on emp1.eno=dept.eno;
                    Or
SQL>select * from emp1 , dept where emp1.eno=dept.eno(+);
                    Or
SQL> select * from emp1 e, dept d where e.eno=d.eno(+);
```

ENO	ENAME	ADDRESS	DNO	DNAME	ENO
1	ram	delhi	d1	hr	1
2	varun	chennai	d2	it	2
4	amir	delhi	d3	mrkt	4
3	ravi	banglore			

2. RIGHT OUTER JOIN

```
SQL> select * from emp1 right outer join dept on emp1.eno=dept.eno;
                    Or
SQL> select * from emp1 , dept where emp1.eno(+)=dept.eno;
                    Or
SQL> select * from emp1 e, dept d where e.eno(+)=d.eno;
```

ENO	ENAME	ADDRESS	DNO	DNAME	ENO
1	ram	delhi	d1	hr	1
2	varun	chennai	d2	it	2
4	amir	delhi	d3	mrkt	4

3. FULL OUTER JOIN

```
SQL> select * from emp1 full outer join dept on emp1.eno=dept.eno
```

ENO	ENAME	ADDRESS	DNO	DNAME	ENO
1	ram	delhi	d1	hr	1
2	varun	chennai	d2	it	2
3	ravi	banglore			
4	amir	delhi	d3	mrkt	4

It differs from the network model in that the records are organized as collections of trees rather than arbitrary graphs.

1.3.2.4 Differences between the Models

The relational model differs from the network and hierarchical models in that it does not use pointers or links. Instead, the relational model relates records by the values they contain.

1.3.3 Physical Data Models

Physical data models are used to describe data at the lowest level i.e., in the Physical level.

Two of the widely known ones are:

i) Unifying model
ii) Frame memory

1.4 Database Languages

A database system provides two different types of languages: one to specify the database schema, and the other to express database queries and updates.

1.4.1 Data Definition Language

A database schema is specified by a set of definitions which are expressed by a special language called a *data definition language* (DDL).

The result of compilation of DDL statements is a set of tables which are stored in a special file called *data dictionary* (or *directory).*

A data directory is a file that contains *metadata;* that is, "data about data."

The storage structure and access methods used by the database system are specified by a set of definitions in a special type of DDL called a *data storage and definition* language.

1.4.2 Data Manipulation Language

A *data manipulation language* (DML) is a language that enables users to access or manipulate data.

Data manipulation in terms of

- The retrieval of information stored in the database
- The insertion of new information into the database
- The deletion of information from the database
- The modification of data stored in the database.

There are basically two types:

i) Procedural DMLs
ii) Nonprocedural DMLs

- **Procedural** DMLs require a user to specify *what* data is needed and *how* to get it.
- **Nonprocedural** DMLs require a user to specify *what* data is needed *without* specifying how to get it. Nonprocedural DMLs are easier to learn which is not efficient.

Query & Query Language

- A *query* is a statement requesting the retrieval of information.
- The portion of a DML that involves information retrieval is called a *query language.*

1.5 Database Manager

A *database manager* is a program module which provides the interface between the low-level data stored in the database and the application programs and queries.

The database manager is responsible for the following tasks: **Interaction with the file manager:** The database manager translates the various DML statements into low-level file system commands. Thus, the database manager is responsible for the actual storing, retrieving, and updating of data in the database.

Integrity enforcement: The data values stored in the database must satisfy certain types of consistency constraints. The database manager determines whether updates to the database result in the violation of the constraint; if so, appropriate action must be taken.
For example, the number of hours an employee may work in one week may not exceed some specific limit (say, 80 hours). Such a constraint must be specified explicitly by the database administrator.

Security enforcement: It is the job of the database manager to enforce the security requirements. That is every database user need not have access to the entire content of the database.

Backup and recovery: A computer system, like any other mechanical or electrical device, is subject to failure. Causes of failure include disk crash, power failure, and software errors. In each of these cases, information concerning the database is lost. It is the responsibility of the database manager to detect such failures and restore the database to a state that existed prior to the occurrence of the failure. This is usually accomplished through the initiation of various backup and recovery procedures.

Concurrency control: When several users update the database concurrently, the consistency of data may no longer be preserved. Controlling the interaction among the concurrent users is another responsibility of the database manager.

1.6 Database Administrator

The person who is having central control (i.e., overall control) of both data and programs accessing that data is called the *database administrator* (DBA).
The various functions of the database administrator:

Schema definition: The original database schema is created by writing a set of definitions which are translated by the DDL compiler to a set of tables that are permanently stored in the *data dictionary*.

Storage structure and access method definition: Appropriate storage structures and access methods are created by writing a set of definitions which are translated by the data storage and definition language compiler.

Schema and physical organization modification: Modifications to either the database schema or the description of the physical storage organization, are accomplished by writing a set of definitions which are used by either the DDL compiler or the data storage and definition language compiler to generate modifications to the appropriate internal system tables (for example, the data dictionary).

Granting of authorization for data access. The database administrator regulates the access of the database, by granting different types of authorization to the various users.

Integrity constraint specification: Integrity constraints are kept in a special system structure that is consulted by the database manager whenever an update takes place in the system.

1.7 Database Users

A primary goal of a database system is to provide an environment for retrieving information from and storing new information into the database.
There are four different types of database system users;

Application programmers: Application programmers are computer professionals who interact with the system through DML calls, which are included in a program written in a *host* language. These programs are commonly referred to as *application programs*.
The DML syntax is usually quite different from the host language syntax. A special preprocessor, called the DML *precompiler*, converts the DML statements to normal procedure calls in the host language. The resulting program is then run through the host language compiler, which generates appropriate object code.
There are special types of programming languages which combine control structures of Pascal-like languages with control structures for the manipulation of a database objects. These languages, sometimes called *fourth-generation languages*, often include special