Relational Model

Key Constraints : It refers to the set of attributes (or) single attribute is used in order to access the tuples (or) rows from relation (or) table uniquely

single attribute ⟶ <rollno>
primary key

Multiple attribute ⟶ <loanno, Payment No>

there are five different types of keys

(i) Primary key
(ii) Canditate key
(iii) super key
(iv) composite key
(v) foreign key

(i) **Primary Key** : It refers to the attribute name has unique and not null properties which makes the accessing of data easier from the table

Eg: Rollno

(ii) **Candidate Key** : While excluding the primary key from the table the remaining attributes in the table can form a candidate key. by having the property of uniqueness in order to retrive the data from table

Eg: <phoneno, Address>
<Email, name>

(iii) **Super Key** : It is the combination of primary key and non primary key

Eg: <rollno, name>
<rollno, address>
<rollno, marks>

(iv) **Composite Key** : It is the collection of primary keys from the relations

Eg: rollno, Empid, dept-id, project id.

(v) Foreign Key : It is used to create a link in between two different tables

Consider Employee and department tables where the departmend id in dept table is primary key and it is acting as foreign key for the employee table

Primary key — Empid, Ename; Foreign key — Deptid; Primary key — Deptid, Dname

Empid	Ename	Deptid
1	Rajesh	10
2.	Suresh	20
3.	Mahesh	30
4.	Vijju	10

Deptid	Dname
10	Sales
20	Amounts
30	Admin

Anamolies → insert, delete, update

```
create table Emp(rollno varchar2(5), primarykey,
    Ename varchar2(10), Deptid number(3),
    foreign key(deptid) references Dept(Deptid));
```

Anamolies :-
Anamolies over the operations

Insertion :- To enter the data in the department table it will not affect the employee details
In case of employee table check either the departmend id is existed in the department table (or) not

Deletion :- To delete the data from the department table check whether any employee is working under department or not. If it is first remove the employee

Updation :- To update the data in the department table check wheither the department id is also updated in the employee table

To resolve the anamolies create a cascading operation for the foreign key table such that during deletion (or) updation operations the results could be effected in both the tables

Create table Emp (Empid number(5) primary key, Ename varchar2(
Deptid number(3) foreign key(Deptid) references Dept(Deptid)
on delete cascade on update cascade);

Relational Algebra & Relational Calculus

these two are formal query languages which are used to develop SQL (structured Query Language)

Relational Algebra :-

selection :- It is used to retrive the rows from the table denoted by σ (sigma)

syntax :- $\sigma_{\text{condition}}$ (tablename)

Display the details of the Employee whose dept id is 10.

$\sigma_{deptid = 10}$

Output :-

Empid	Ename	Deptid
1.	Rajesh	10
2.	Vijju	10

Projection :- It is used to retrive the ~~rows~~ column from the table denoted by π (pi)

syntax :- $\pi_{\text{condition}}$ (tablename)

Display Employee id & Employee name who department id is 10

$\pi_{Empid, Ename}$ (Emp)

Output :-

Empid	Ename
1	Rajesh
2.	Vijju

Display employee id & employee name whose department id = 10

$\Pi_{Empid, Ename}(\sigma_{deptid=10}(Emp))$

Output :-

	Empid	Ename
1.		Rajesh
2.		Vijju

Set Operations :- To perform selection and projection it requires single table

(i) Union
(ii) Intersect
(iii) Set difference
(iv) Cross Product (or) Cartesian Product (x)

To perform set operation it requires two tables.

(i) To perform union, intersect, set difference it requires two tables of same type that is no. of attributes and type of domain

Emp1

Empid	Ename	Deptid
1.	Rajesh	10
2.	Vijju	10

Emp2

Empid	Ename	Deptid
2.	Vijju	10
3.	Suresh	20

Emp1 ∪ Emp2

Emp1 e1 ⟶ e1 ∪ e2
Emp2 e2 ⟶

Empid	Ename	Deptid
1.	Rajesh	10
2.	Vijju	10
3.	Suresh	20

Union :- It is used to combine the rows from 2 relation instances by excluding duplications

Intersect :- It is used to display the common rows from two deletion instances.

Empid	Ename	Deptid
1.	Rajesh	10
2.	Vijju	10

Empid	Ename	Deptid
2.	Vijju	10
3.	Suresh	20

Emp1 ∩ Emp2

Emp1 e1 ↘
Emp2 e2 → e1 ∩ e2

Empid	Ename	Deptid
2.	Vijju	10

Set Difference :- It produces the rows of e_1 which are not in e_2

Employee

Emp1

Empid	Ename	Deptid
1.	Rajesh	10
2	Vijju	10

Emp2

Empid	Ename	Deptid
2.	Vijju	10
3	Suresh	20

Emp1 − Emp2

Emp1 e1 ↘
Emp2 e2 → e1 − e2

Empid	Ename	Deptid
1.	Rajesh	10

Cross Product :- To perform this operation it requires two different tables

Employee

Empid	Ename	Deptid
1.	Rajesh	10
2.	Suresh	20
3.	Vijju	10

Dept

Deptid	Dname
10	sales
20	Accounts

Empid	Ename	Deptid	Dept id	Dname
1.	Rajesh	10	10	sales
1.	Rajesh	10	20	Accounts
2.	Suresh	20	10	Sales
2	Suresh	20	20	Accounts
3	Vijju	10	10	Sales
3	Vijju	10	20	Accounts

Consider two different tables to perform cartesian product that is every row of first table is combine with the everyother row of second table

Rename Operator :- It is used to modify the relation nam (or) attributes name. It is denoted by ρ (Rho)

To Modify the table name **syntax**:

ρ newtablename oldtablename

Eg: ρ employee Emp

To Modify the attributes of tablename use **syntax**:

ρ new tablename (New attributes) oldtablename

Eg: ρ employee (Eid, name, Did) Emp

Joins :- There is a limitation in cartesian product i.e, if the two tables having 1000 no. of rows then it is double in the rows after performing cartesian product

→ To perform operations over these largest resultant is very difficult, this is resolved by using joins operation

→ It is used to combine 2 (or) more tables having same column name

→ It performs cartesian product followed by selection process

(i) **InnerJoin** :- It is used to produce the tuples with matched attributes and the remaining are discarded
It includes all the tuples from all the relations

(ii) ~~OuterJoin~~ :-
* Conditional Join ($\bowtie_C$)
* Equi Join ($\bowtie_E$)
* Natural Join ($\bowtie$)

ConditionalJoin($\bowtie_C$) :- Combine Tuples from different relations having condition that is to be satisfied

Emp.deptid < dept.deptid

Emp $\bowtie_C$ dept

Empid	Ename	Deptid	Deptid	Dname
1.	Rajesh	10	20	Accounts
2.	Vijju	10	20	Accounts.

Equi Join ($\bowtie_=$) :- It is used to combine the tuples of different relations by equalizing the values of common attributes

Emp.deptid = dept.deptid

Emp $\bowtie_=$ dept

Empid	Ename	Deptid	Deptid	Dname
1.	Rajesh	10	10	Sales
2.	Suresh	20	20	Accounts

Natural join :- It is used to combine the tuples of different relations having common attribute name.

Empld	Ename	Deptld	Dname
1.	Rajesh	10	sales
2.	Suresh	20	Accounts
3.	Vijju	10	sales.

Outer Join :- It is used to display all the tuples from two relations along with matched rows from the relation

There are three types

(i) **Left Outer Join** :- It is denoted by ⟕

It is used to produce the matched tuples from right side table and all the rows from the leftside table

$C \overset{O}{\bowtie}_{c.custid = o.custid} O$

(ii) Customer

Custid	Cname	Amount
1.	x	100
2.	y	200
3.	z	300
4.	a	400
5.	b	500

Order

Orderid	Amount	Custid
10	100	3
20	150	10
30	300	5
40	400	16

Customer ⟕ orders

Customer C;
order O;
C ⟕ O;

3=3
5=3

Custid	Cname	Amount	Orderid	Amount	Custid
1.	x	100	–	–	–
2.	y	200	–	–	–
3.	z	300	10	100	3
4	a	400	–	–	–
5	b	500	30	300	5

→ To display all the columns from both tables

↔ To display only few columns from the resultant of left outer Join

$\Pi_{C.custid, O.orderid, O.amount}$

$(C \overset{I}{\bowtie} C.custid = O.custid\ O)$

custId	Orderid	Amount
1	–	–
2	–	–
3	10	150
4	–	–
5	30	300

(ii) Right Outer Join :- It is used to produce the result from two diff relations by writing the right side as it is along with matched tuples from left side tables
It is denoted by ⟖

$C \overset{I}{\bowtie} C.custid = O.custId\ O$

Custid	Cname	Amount	Orderid	Amount	Custid
3	z	300	10	100	3
–	–	–	20	150	10
5	b	500	30	300	5
–	–	–	40	400	16

→ To display all the column names from the two relations
→ To display only few columns from two relations use

$\Pi_{C.custid, O.orderid, O.amount}\ (C \overset{I}{\bowtie} C.custid = O.custid\ O)$

custId	orderid	Amount
3	10	100
–	20	250
5	30	300
–	40	400

(iii) Full outer join :- It is used to produce the result from left outer join and right outer join.

It is denoted by ⟗

$$C \text{ ⟗ } C.custid = O.custid \; O$$

custid	Cname	Amount	Orderid	Amount	custid
1	x	100	–	–	–
2	y	200	–	–	–
3	z	300	10	150	3
4	a	400	–	–	–
5	b	500	30	200	5
3	z	300	10	150	3
–	–	–	20	250	10
5	b	500	30	200	5
–	–	–	40	350	16

→ To display only few columns from two relations

$$\Pi C.custido.orderid, O.Amount$$

$$C \text{ ⟗ } C.custido = O.custid$$

custid	Orderid	Amount
1	–	–
2	–	–
3	10	100
4	–	–
5	30	300
3	10	100
–	20	150
5	30	300
–	40	400

Join in SQL

Syntax : select columnames from table1 inner/outer join table2 on condition ;

$$Emp \bowtie_c Dept$$

Inner Join

Conditional Join

(i) All columns

$$Emp \bowtie_c Dept$$

$$E \bowtie_{e.deptid < deptid}$$

```
select *.e, *.d from
    emp e inner Join
    dept d on e.deptid<d.deptid
```

(ii) Only few Column

$$\Pi_{e.empid, ename, d.deptid} (e \bowtie_{e.deptid < d.deptid} d)$$

→

```
select e.empid,ename ,d.deptid
    from emp e innerJoin
        dept d on
        e.deptid < d.deptid.
```

EquiJoin

$$E \bowtie_{e.deptid = d.deptid}$$

```
select *.e, *.d from
    emp e inner join
    dept d on e.deptid=d.deptid
```

$$(e \bowtie_{e.deptid = d.deptid} d)$$

```
select e.empid, e.name, d.deptid
    from emp e inner join
        dept d on
        e.deptid = d.deptid.
```

NaturalJoin

$$E \bowtie D$$

$$Emp \bowtie_c Dept$$

```
select *.e,*.d from
    emp e inner join
    dept d
```

$$\Pi_{e.empid, e.ename, d.deptid} (e \bowtie d)$$

```
select e.empid, e.ename, d.deptid
    from emp e inner join
    dept d.
```

Outer Joins

Left Outer Join

All columns

customer $\bowtie_c$ order

$C \bowtie O$

```
Select *.C, *.O from
  customer c left outerjoin
    order o on
  c.custid = o.custid;
```

Only few columns

$\Pi_{C.custid, O.orderid, O.Amount}$

$(C \bowtie_{C.custid = O.custid} O)$

```
Select c.custid, o.orderid, o.Amo
from customer c left outerjoi
  order o;
```

Right Outer Join

customer $\bowtie_c$ order

$C \bowtie O$

```
Select *.C, *.O from
customer c right outerjoin
  order o on
c.custid = o.custid;
```

$\Pi_{C.custId, O.orderid, O.Amount}$

$(C \bowtie_{C.custid = O.custid} O)$

```
select c.custid, o.ordereid,
  o.Amount
from customer c right outerjoi
```

Full Outer Join

customer $\bowtie_c$ order

$C \bowtie O$

```
Select *.C, *.O from
customer c full outerjoin
order o on
  c.custid = o.custid;
```

$\Pi_{C.custid, O.orderid, O.amount}$

$(C \bowtie_{C.custid = O.custid} O)$

```
Select c.custid, o.ordereid,
  O.Amount
from customer c full outer
  join
```

Division Operator

Consider two different relations R_1 and R_2 where it is denoted by symbol (÷) which is represented by $R_1 \div R_2$. R_2 is an subset of R_1 then only ~~distant~~ division possible i.e all the rows from R_2 must be decided in R_1

$$R_1 \div R_2$$

Subjects

Name	C Name
DBMS	B.Tech
P&S	B.Tech
DBMS	M.Tech
DMS	M.Tech

Course

Cname
B.Tech
M.Tech

→ Display the name of subject that is taught in all courses.

Relational Calculus : It is an another formal query language which is used to develop SQL. It is an non-procedural (or) declarative language where its specifies what is the need of data and doesn't specifies how to get the data

→ In the case of relational Algebra it is procedural language where it specifies need of data and how to get the data.

→ there are two categories (i) Tuple Relational Calculus (TRC)
(ii) Domain Relational Calculus (DRC)

(i) **TRC** :- It uses tuple variable t to each and every row of table and uses the condition written in predicate logic or first order logic P(t) to the rows and display the result once the condition is satisfied.

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

Write the TRC query to display all details of employees how are working in department id = 10

$\{t \mid t \in Emp \wedge t.deptid = 10\}$

To display employeeId of all employees

$\{t \mid S \in Emp \wedge (S.empid = t.empid)\}$

t

Empid	

S

Empid	

(ii) DRC : It uses Domain variable to refer the columns and all apply condition to the columns if it satisfied with condition when it produces the column values The condition is written in predicate logic

syntax : $\{ <x_1, x_2, x_3, \ldots> \mid p(<x_1, x_2, x_3 \ldots>)\}$

- x_1, x_2, x_3 : Domain variables of column names
- $p(<x_1, x_2, x_3 \ldots>)$: condition on column values in predicate logic

To display the details of all employees whose department NO =

$\{ <x_1, x_2, x_3> \mid x_1, x_2 \in Emp \wedge x_3 = 10 \}$

To display employee id of all employees

$\{<x_1> \mid x_2, x_3 \in Emp < x_1, x_2, x_3 > \}$

SQL :- It is a structure Query Language which is an commerical query language developed by IBM

features:-

1.) It has DDL and DML statements

2.) It supports embedded & dynamic SQL

3.) SQL can be called from (or) attached to the programming languages

4.) It allows the users to construct a query at runtime

Triggers & Integrity constraints

Triggers: It is used to activate the database and IC

- IC is used to apply conditions over the domain (or)
- key constraints
- SQL allows a client program execution at remote
- database server
- transaction management
- Provides security by using views & tables

syntax for writting SQL queries

Select column names from tablename
where condition.

find total no. of emlp's working in org

Select count (empid) from emp;

o/p:-

count(empid)
3

Aggregate Operators :-

(1) Count (columnname) :- It is used to count the no. of row of a col

(2) Avg (Columnname) :- To find out avg value of a column

find the avg of salaries of all empl's

Select Avg (salary) from emp;

o/p :-

Avg(salary)
2667.

(3) Max (Column Name) :-

To find the max value from the column

find the max salary from employees

select Max (salary) from emp;

o/p :-

Max(salary)
5000

find the smallest salary a/emp

select Min(salary) from emp;

o/p :-

Min(salary)
1000

(4) Min (ColumnName) :- It is used to find the smallest value from the column

(5) sum (Column name) :- It is used to find out the total of values from the column name

find the sum of salaries of employees

Select sum (salary) from emp;

o/p :-

sum(salary)
8000

'Group by' & 'Having' clauses :-

Groupby clause :- It is associated with ~~having a~~ aggregate functions

It is used after select command

Having clause :- It is used for checking function similar to where clause

Syntax: select columnname, functionname (column)
from tablename group by columnname
having condition;

Ex :- find the total no. of employees working in an depts and display deptid.

Empid	Ename	Deptid	sal
1	Ravi	10	1000
2	Ram	20	5000
3	Rajesh	10	2000

Deptid	Count(empid)
10	2
20	1

```
select deptid, count(empid) from emp
    groupby deptid
```

Display highest salary of employee's department wise and display employee id

```
select empid, Max(sal) from emp
    groupby deptid;
```

o/p :-

empid	Max(salary)
1	2000
2	5000

Display all the details of employees whose salary is greaterthan thousand in departmentwise

```
select * from emp
    groupby deptid having
        sal>1000;
```

O/P:-

Empid	Ename	Deptid	sal
3	Rajesh	10	2000
2	Ram	20	5000

Triggers:-

It is an program code which is automatically called whenever modifications are performed in Database.

It is consists of (1) Event (2) Condition (3) Action.

(1) To perform insert, deletion or update operation to change the data base

(2) A query is executed whenever trigger is activated

(3) If the condition is satisfied then what is the alert msg is activated.

Syntax:-

```
create [or replace] trigger <triggername>
{Before | After}
{Insert | delete | update} on tablename
[for each row | for each statement]
```

create trigger to perform action before delete or update operation,

```
create trigger t1 before delete or update on dept foreach row begin
if (old.deptid=10 ∧ old.deptid=20) then
    raise_application_error (-201, "can't delete/update unless
            data in emp is deleted");
  end if;
end;
```

NULL values : Null value is used to represent the empty value

To compare whether the value is null value or not

Is null is used to verify the value is empty or not

→ find whether the salary of employee is empty or not

Select * from emp where sal is NULL;

Constraints over single table

At the time of table creation use the constraints like unique, normal, default, primary key in order to avoid inconsistence data over

Domain Constraints over single table

Over the columnname apply the conditions using constraint keys

Create an salary with the range of 1000 to 10,000

syntax:-

Create domain domainname datatype constraint,

↳ Create domain salary number(5)

check (value >=1000 and value <10000);

Constraints over Multiple table with Assertion

Integrety constraints are applied over several tables it is called assertion

Syntax:-

Create Assertion Assertionname

constraint (Query1 from table1 + Query2 from table2);

Check whether the total no of employees (emp) should be less than (dep) then total no. of departments should be less than 3

↳ Create Assertion A1

check ((select count (Empid) from emp where count<5)+

(select count (deptId) from dept where (count<3))