Relational Algebra

Definition of Relation :

The mathematical concept underlying the relational model is the set-theoretic relation, which is a subset of the Cartesian Product (CP) of a list of domains. A domain is simply a set of values.

The CP of domains D_1, D_2,……..,D_k written as D_1 X D_2 X ….. X D_k is the set of all k tuples (v1,v2,……,vk) such that v1 is in D_1, v2 is in D_2, and so on.

Example : k = 2, D_1 = {0,1}, D_2 = {a, b, c}, then D_1 X D_2 = { (0,a) , (0,b), (0,c), (1,a), (1,b), (1,c)}

A relation is any subset of the CP of one or more domains. In the context of DBMS, it is pointless to discuss ''infinite relation" rather the assumption is “all relations are finite”.

Terminology :

Tuple : The members of a relation are called tuples.

Arity : Each relation that is a subset of D_1 X D_2 X, ….XD_k is said to have arity k; another term is degree.

A tuple (v_1, v_2, ….,v_k) has k components i.e. the ith component is v_i.

We can view a relation as a table, where each row is a tuple and each column corresponds to one component. The columns are often given names, called attributes. The set of attribute names for a relation is called the relation scheme.

Example : A relation whose attributes are CITY, STATE and POP. The arity of the relation is 3. Here (Calcutta, W.B., 100160).

The relation scheme for this relation may be CITYINFO (CITY, STATE, POP).

Relational Algebra

Relational algebra(RA) is required to the design of languages for expressing queries about relations. The aspects of data manipulation language are query answering, insertion, deletion and modification of tuples.

Query languages for the relational model broadly classified into two classes.

i) Algebraic languages where queries are expressed by applying specialised operators to relations.
ii) Predicate calculus languages where queries describe a desired set of tuples by specifying a predicate the tuples must satisfy.

The constraint of finiteness introduces some difficulties into the definition of RA. We cannot allow algebraic operation of complementation, since -R generally denotes an infinite relation, the set of all tuples not in R. There is no way to list the relation -R, even if the query language permitted such an expression

The operands of RA are either constant relations or variables denoting relations of a fixed arity. There are five basic operators that serve to define RA. To define the operators, a few relations are given below :

A	B	C
a	b	c
d	a	f
c	b	d

R

D	E	F
b	g	a
d	a	f

S

---

a	b	c
d	a	f
c	b	d
b	g	a

$R \cup S$

1. UNION

The union of relations R and S, denoted by $R \cup S$, is the set of tuples those are in R or S or both. We only apply the union operator to relations of the same arity, so all tuples in the result have the same number of components.

2. DIFFERENCE

The difference of relations R and S denoted by R - S, is the set of tuples in R but not in S. We again require that R and S have the same arity.

a	b	c
c	b	d

R - S

3. CARTESIAN PRODUCT

Let R and S be relations of arity k1 and k2 respectively. The R X S, cartesian product of R and S, is the set of (k1+k2)-tuples whose first k1 components come from a tuple in R and whose last k2 components come from tuple in S.

A	B	C	D	E	F
a	b	c	b	g	a
a	b	c	d	a	f
d	a	f	b	g	a
d	a	f	d	a	f
c	b	d	b	g	a
c	b	d	d	a	f

R X S

4. PROJECTION

If R is a relation of arity k, we let $\prod_{i1,i2,\ldots\ldots im}(R)$ where the ij's are distinct integers in the range 1 to k, denote the projection of R onto components i1,i2,……,im.

A	C

a	c
d	f
c	d

$\Pi_{A,C}$ (R)

5. SELECTION

This operation is used to select a subset of the tuples in a relation that satisfy a selection condition.

A	B	C
a	b	c
c	b	d

$\sigma_{B=b}$ (R)

R X S

ADDITIONAL ALGEBRAIC OPERATIONS :

There are a number of useful operations that can be expressed in terms of the five previously mentioned operations which are known as primitive operations.

1. INTERSECTION

Tuples present in R and S both termed as R ∩ S . This may be derived as R - (R - S)

---

d	a	f

R ∩ S

2. QUOTIENT

Let R and S be relations of arity r and s where r > s and S ≠ ∅. Then R ÷ S = Π_Y (R) - Π_Y ((Π_Y (R) X S) - R) where y = 1,2,……,r-s

a	b	c	d
a	b	e	f
b	c	e	f
e	d	c	d
e	d	e	f
a	b	d	e

R

c	d
e	f

S

a	b
e	d

R ÷ S

3. JOIN

The θ- join of R and S on columns I and j written R X S where θ is an arithmetic comparison operator (=, <, and so on). The θ join of R and S is those tuples in the CP of R and S such that the ith component of R stands in relation θ to the jth component of S. If θ is =, the operation is often called an equijoin.

4. NATURAL JOIN

The natural join written as R ⋈ S is applicable only when both R and S have columns that are named by attributes.

i) Compute R X S

ii) For each attribute A that names both a column in R and a column in S, select those tuples from R X S whose values agree in the columns for R.A and S.A.

iii) For each attribute A above, project out the column S.A.

SOME MORE ADDITIONAL OPERATIONS :

Some common database requests cannot be performed with the standard RA operations described above. Most commercial query languages for RDBMS include capabilities to perform these requests. These operations enhance the expressive power of the RA.

1. AGGREGATE FUNCTIONS

The first type of request that can't be expressed in RA is to specify mathematical aggregate functions on collections of values from the database.

They are SUM, MAXIMUM, MINIMUM, AVERAGE, COUNT.

Let us consider a table named as EMPLOYEE as follows :

NAME	DNO	SSN	SEX	SALARY
J. Saha	5	123456789	M	30000
F. Wadekar	5	333445555	M	40000
A. Zardari	4	999887777	F	45000
J. Wong	4	987654321	F	43000
R. Nath	5	666884444	M	38000
J. Epal	5	453453453	F	50000
A. Josh	4	987987987	M	25000
J. Bose	1	888665555	M	50000

EMPLOYEE

Example :

The following are three different solution to retrieve each department number, the number of employees in the department and their average salary.

R(DNO, NO_OF_EMP, AV_SAL) ← DNO ○ COUNT SSN, AVERAGE SALARY (EMPLOYEE)

DNO	NO_OF_EMP	AV_SAL
5	4	39500
4	3	37667
1	1	50000

DNO ○ COUNT SSN, AVERAGE SALARY(EMPLOYEE)

DNO	COUNT_SSN	AVERAGE_SALARY
5	4	39500
4	3	37667
1	1	50000

o COUNT SSN, AVERAGE SALARY(EMPLOYEE)

COUNT_SSN	AVERAGE_SALARY
8	40125

Example :

There is a STS database as follows :

TEACHER (tname, emp-no, dept)

SUBJECT (sub-no, sub-title, credit)

STUDENT(sname, roll-no, hostel)

TAUGHTBY(emp-no, sub-no)

TAKENBY(sub-no, roll-no, status, marks)

Q1 :

Find the department to which Prof. “XYZ” belongs.

$\Pi_{dept}\ (\sigma_{tname = \text{‘XYZ’}}\ (\text{TEACHER}))$

$\bowtie$

Q2 :

Find the name of the teacher who teaches the subject DBMS.

$\Pi_{tname}\ (\text{TEACHER} \bowtie \text{TAUGHTBY} \bowtie \sigma_{sub\text{-}title = \text{‘DBMS’}}\ (\text{SUBJECT}))$

Q3 :

Find the name of the student who study the subject DBMS as elective.

Π_{sname} (STUDENT $\bowtie$ ($\sigma_{status = \text{'Elec'}}$ (TAKENBY) $\bowtie$

$\sigma_{sub\text{-}title = \text{'DBMS'}}$ (SUBJECT)))

Q4 :

Find the name of students who study 'DBMS ' but not 'OPERATING SYSTEM'.

Π_{sname} ($\sigma_{sub\text{-}title = \text{'DBMS'}}$ (SUBJECT $\bowtie$ TAKENBY $\bowtie$ STUDENT) - $\sigma_{sub\text{-}title = \text{'OS'}}$ (SUBJECT $\bowtie$ TAKENBY $\bowtie$ STUDENT))

Q5 :

Find the name of students who study DBMS and SOFTWARE ENGG.

Π_{sname} ($\sigma_{sub\text{-}title = \text{'DBMS'}}$ (SUBJECT X TAKENBY X STUDENT) $\cap$ $\sigma_{sub\text{-}title = \text{'SW ENGG'}}$ (SUBJECT X TAKENBY X STUDENT))

The following is a banking enterprise database:

DEPOSIT (branch-name, ac-no, cust-name, bal)

BRANCH (branch-name, assets, branch-city)

CUSTOMER (cust-name, street, cust-city)

BORROW (branch-name, loan-no, cust-name, amt)

Q1 :

Find all customers who have a loan for an amount greater than Rs. 10,000/-.

$\Pi_{cust\text{-}name}$ ($\sigma_{amt > 10000}$ (BORROW))

Q2 :

Find all the customers having a loan or an account or both at ‘BEC’ branch.

$\Pi_{\text{cust-name}}$ ($\sigma_{\text{branch-name = ‘BEC’}}$ (DEPOSIT) $\cup$ $\sigma_{\text{branch-name = ‘BEC’}}$ (BORROW))

Q3 :

Find all the customers who have an account at the ‘BEC’ branch but do not have a loan from that branch.

$\Pi_{\text{cust-name}}$ ($\sigma_{\text{branch-name = ‘BEC’}}$ (DEPOSIT)) - $\Pi_{\text{cust-name}}$ ($\sigma_{\text{branch-name = ‘BEC’}}$ (BORROW))

Q4 :

Find all the customers who have an account at all branches located at Howrah.

$\Pi_{\text{cust-name, branch-name}}$ (DEPOSIT) $\div$ $\Pi_{\text{branch-name}}$ ($s_{\text{branch-city = ‘ Howrah’}}$ (BRANCH)

SQL

Relational Algebra is rarely used as the query language in real world. SQL is most commonly used commercial query language. It has been standarised many times. It is a declarative, query language.

Simple SQL Query Syntax

```
SELECT COLUMNS
FROM  TABLES
WHERE PREDICATE
        |
SELECT A1, A2, ….., An
```

FROM r1, r2, … , rm
WHERE P

The above is nearly equivalent to

$\Pi_{A1, A2, \ldots, An} (\sigma_{P} (r1 \times r2 \times \ldots \times rm))$

Here duplicate will be eliminated but in the above SQL query, duplicate will not be eliminated.

SQL Components

It has three parts

a) Data Definition language (DDL)
   i) Specifies database schema
   ii) Creates, modifies and destroys tables
b) Data Manipulation Language (DML)
   i) Manipulate data using INSERT , MODIFY, DELETE statements
   ii) Query the data in the database
c) Data control language (DCL)
   i) Grant and revoke authorisation for DB access
   ii) Audit database use
   iii) Transaction Mgt.

Basic DDL statements

i) create ii) drop iii) alter

i) create

Q1:

Create a table having three attributes

```
create table course (
        cno     integer ,
        cname   varchar(20),
        prereq  integer );
```

Q2 :

Create a table having attributes with constraints.

```
create table course (
        cno  integer primary key,
```

```
cname      varchar(20) not null,
prereq     integer);
```

Q3:

Create a domain.

```
create domain name_of_course varchar(30);
```

Now create a table where an attribute takes value from the defined domain.

```
create table course (
    cno     integer default 0 not null,
    cname   name_of_course,
    prereq          integer);
```

ii) drop & iii) alter

Q1: Destroy a table.

```
drop table student ;
```

i) Removes all data from student

ii) Removes the table structures

Q2: Modify a table removing a column

```
alter table student
    drop column hno;
```

Q3: Modify a table adding a column

```
alter table student
    add columns age integer;
```

There is a database as follows :

FACULTY (fno, fname, dno, sal, age)

DEPT (dno, dname, budget, floor)

COURSE(cno, cname, prereq)

STUDENT (sno, sname, hno, age, year, grade)

REG-FOR (sno, cno)

TAUGHT-BY (sno, fno)

Q1 :

Print names of faculty in ‘CST’ department

```
select FACULTY.fname
 from FACULTY, DEPT
```

```
where FACULTY.dno = DEPT.dno
and DEPT.name = 'CST'
```

Q2 :

List the course names with their prerequisite course names

```
select C.cname, P.cname
from COURSE C, COURSE P
where C.prereq = P.cno ;
```

SET Functions

i) COUNT
ii) MAX
iii) MIN
iv) AVG
v) SUM

Q3:

Display number of 'CST' department faculty.

```
select count (*)
from FACULTY, DEPT
where DEPT.dno = FACULTY.dno
and DEPT.dname = 'CST' ;
```

Q4:

Give the number of hostels

```
select count (distinct hno)
from STUDENT;
```

Q5:

Give oldest faculty member's age

```
select max (age)
from FACULTY;
```

Q6:

Display age of youngest and oldest faculty

```
select max (age), min (age)
    from FACULTY;
```

Q7:

Display sum of the salaries of the faculty

```
select sum (sal)
 from FACULTY;
```

Q8:

Find out average age of 'IT' dept faculty

```
select avg (age)
from FACULTY, DEPT
where DEPT. dno = FACULTY.dno
and DEPT.dname = 'IT';
```

Q9:

Print name and age of oldest faculty

```
select name, max (age)
 from FACULTY;
```

The above query is syntactically incorrect

The correct syntax is as follows :

```
select name, age
from FACULTY
where age = ( select max (age)
                from FACULTY);
```

P.S. i) Null values are ignored by set functions.

ii) Result is null for all set functions (except count) if number of qualifying tuples is zero.

GROUP BY and HAVING CLAUSE

```
SELECT < select list >
FROM < table list>
WHERE < search condition >
```

GROUP BY < specification list >

HAVING < search condition>

Steps to perform Group By & Having

Step 1 : Perform the cross product and the selection corresponding to the from clause and where clause respectively.

Step 2 : Divide the resulting table of Step 1 into minimum groups such that within a group all rows have same value for column(s) on which grouping is specified.

Step 3 : Eliminate groups of Step 2 that do not satisfy the having clause.

Step 4 : The result table has one tuple per group and the attributes correspond to select list.

Q1 :

Find out year and maximum grade obtained by a student of that year.

```
select year, max (grade)
from STUDENT
group by year;
```

Q2 :

Give average salary of faculty within a department provided the dept has at least 5 faculty.

```
select dno, avg ( sal)
from FACULTY
group by dno
having count > = 5;
```

Security mechanism from within SQL

The data stored in the database needs to be protected from unauthorised access, malicious destruction or alteration and accidental introduction of inconsistency. It is easier to protect against accidental loss of data consistency than to protect against malicious access to the database. Absolute protection of the database from malicious abuse is not possible.

Security is needed against i) unauthorised access to data

ii) multiuser databases expose one's data to other

Integrity constraints provide a means of ensuring that changes made to the database by authorised users do not result in a loss of data consistency. Thus integrity constraints guard against accidental damage to the database. In general, an integrity constraint can be an arbitrary checking pertaining to the database. However, arbitrary constraints may be costly to test. Thus we usually limit ourselves to the integrity constraints that can be tested with minimal overhead.

The concept of 'views' provides a means for a user to design a 'personalised' model of the database. A view can hide data that a user does not need to see. Security is provided if there is a mechanism to restrict the user to his/her personal view (s). A combination of table level security and view level security can be used to limit a user's access precisely to the data that user needs.

A user may have several forms of authorisation on parts of the database. Authorisation is a means by which the database system can be protected against malicious or unauthorised access. A user who has some form of authority may be allowed to pass this authority on to other users. However, we need to be careful about how authorisation may be passed among users in order to ensure that we can revoke authorisation at some future time.

SQL provides capability to protect or control access to i) table ii) view iii) columns of a table.

Integrity constraints

An integrity constraint is a rule that restricts the values for one or more columns in table/view. There are several types of constraints which fulfil our real life requirements.

The following are two examples of integrity constraints :

i) rate < 50 ii) quantity > 1000

Integrity constraints may be enforced either by application program or by DBMS. Here we shall restrict ourselves to the second one. The integrity constraint can appear in either 'create table' and 'alter table'.

column_constraint

not null

alter table vendor

```
modify ( vendor_add  varchar(20)
                constraint nn_add not null);
```

unique

```
create table item (
        item_id varchar(6),
        item_descrip varchar(15) unique,
                .
                .
                . );
```

primary key

```
create table customer (
    customer_code varchar(8) primary key,
    customer_add   varchar(20),
                .
                .
                . );
```

A primary key constraint designates a column or combination of columns as table's primary key. A table can have only one primary key. To satisfy a primary key constraint, both of the following conditions must be true :

i) No primary key value can appear in more than one row in the table.

ii) No column that is part of the primary key can contain a null.

iii) No column or combination of columns can be designated both as a primary key and a unique key.

Combination of columns as primary key

Consider a table STUDENT having attributes roll_no , name, year, dept and address. This table is maintained to keep information of student of different years and disciplines of an Institution. Now to identify a particular student not only roll_no is sufficient rather dept, year and roll_no may make a primary key to identify a student uniquely.

```
create table student (
        roll_no  number(2),
        name varchar(15),
         year number(4),
```

```
        dept number(2),
    address  varchar(20),
    primary key (dept, year, roll_no));
```

Referential Integrity Constraint

```
create table enquiries_1 (
                .
                .
        vendor_code varchar(8)
        references vendor (vendor_code)
        on delete cascade);
```

The same constraint may be written as follows :

```
create table enquiries_1 (
                .
                .
    vendor_code varchar(8),
    enquiry_date    date,
    foreign key (vendor_code)
    references vendor(vendor_code)




    on delete cascade);
```

Check Constraint

The following are two examples of check constraint using two different syntaxes.

```
create table purchase_order_3 (
                .
                .
        delv_date       date,
        qty     number(4) check
    ( delv_date < '01-JAN-99' and
```

```
                                        qty >= 5 )   disable );
```

P.S. In some implementation, the ‘ delv_date < ‘01-JAN-99’ ‘ should be replaced by

delve_date < to_date(’01-JAN-1999’, ‘dd-mon-yyyy’))

```
create table shop (
        shop_code   varchar(10),
        item_stock      number(2),
                .
                .
        check (shop_code in
        (‘Foundry’, ‘Fitting’, ‘Lathe’));
```

View Mechanism

View is virtual table which does not physically exist. It restricts access to data and hence can be used as a security mechanism.

Create View

```
create view vendor_list
        select vendor_name, vendor_add
        from vendor;
```

Vendor code	Vendor name	Vendor add	Vendor tel
CP123	Computer Point	Camac street	294954
DM459	Data Maintenance	Park street	602364
BARC102	Bharat Cmp	Alipore	312357
TELI65	Telephone Cmp	S.N.Ban. Rd	471234

Vendor_list

The objective of the view vendor_list may be to hide vendor_code and vendor_tel from some users. Instead of the full table the view is available to the users.

View creation taking data from two tables

Let us consider a table named as vendor_item (vendor_code, item_id, present_price, old_price). The objective of this is to maintain information regarding which vendor may supply which item. The other table vendor having schema vendor(vendor_code, vendor_name, vendor_add, vendor_tel).

```
create view roller_vendor
        select vendor_name, vendor_add, vendor_tel
        from vendor_item, vendor
        where item_id = 'roller'
         and vendor_item.vendor_code = vendor.vendor_code;
```

This view gives the information of vendors who supply roller.

Query and Updation on view

Views can be used like base tables in queries. A view can be used anywhere we use a table in any of the SQL statements like DELETE, INSERT, UPDATE, SELECT etc.

```
select count(*)
        from roller_vendor;
```

Updation of views are tricky, complicated and ambiguous. It is still in active area of research. Any updation on a view need to be translated into an update on base tables. A few of the problems with updation of views are described below :

Let us consider a table Quotation_2 in the purchase order system. The attributes are quotation_no, item_id, price, qty. Now let us create a view rod_prices which will give a list of quotations for the item 'iron rod' and correponding prices.

```
create view rod_prices
        select quotation_no, price
        from quotation_2
         where item_id = 'iron rod';
```

Now the problem is, an insert into rod_prices gets converted insert on quotation_2 without value for item_id. Now the problem is ‘ should this be rejected or accepted with item_id set to null ?’

Another problem is as follows :

```
create view V (minprices)
    select min(prices)
     from quotation_2
     where item_id = ‘iron rod’;
```

The problem is – If the value of the minprice is attempted to change by applying the command set minprice = 500 , what to do here ?

Let us consider another problem. There are two table schema FACULTY (eno, dno, ename, age, sal) and DEPT (dno, dname, budget, floor). A view may be created in the following way :

```
create view fac_dept_name (facname, depname)
    select faculty.name, dept.name
        from faculty, dept
    where faculty.dno = dept.dno;
```

Now let us do the following :

```
update fac_dept_name
    set depname = ‘computer science’
    where facname = ‘susan’;
```

The problem with this is described below :

There are two options – 1. Set name of susan’s current department to computer science.

2. Set dno in FACULTY for susan’s tuple to be the dno corresponding to computer science dno.

The second one was what was probably intended but there is no way to know.

Authorisation

A user may have several forms of authorisation on parts of the database. The ultimate form of authority is that given to the database administrator. The database administrator may authorise new users, restructures the database etc. This form of authorisation is analogous to that provided to a “superuser” for an operating system.

A user who has been granted some form of authority may be allowed to pass this authority onto other users. However, we need to be careful about how authorisation may be passed among users in order to ensure that we can revoke authorisation at some future time.

GRANT (object privileges)

The purpose of privileges is to grant different types of authorisation for a particular object to users. There is another types of privileges called system privileges. This is needed to manage the system. In this context only grant on 'object privileges' is discussed.

There are various types of object privileges. A few of them are listed below :

1. ALTER
2. DELETE
3. INSERT
4. SELECT
5. UPDATE
6. REFERENCE

If you want to give someone to use all the privileges available on a particular object, it is possible to give it by the keyword ALL PRIVILAGES provided you have been granted all privileges on that particular object. The user who owns the schema containing an object automatically has all the privileges on the object.

The object may be of various types say TABLE, VIEW etc. Privileges may be granted to either selective user(s) by giving their user_id or to all users by giving the keyword PUBLIC.

Among the privileges listed earlier ALTER and REFERENCE privileges must not be on object VIEW.

Examples

Let us grant to see the table ITEM to the user with user_id storekeeper.

```
grant select
    on item
     to storekeeper;
```

Let us take another example. Here a user, the managing director, with user_id md is considered.

```
grant delete, select
    on purchase_order
    to md
    with grant option;
```

It allows managing director to cancel all purchase order. A WITH GRANT OPTION is used, managing director can in turn pass these privileges onto others, say , deputy director.

The following is an example of granting UPDATE privileges.

```
grant update (vendor_name, vendor_add)
        on vendor
        to data_operator;
```

It allows the data operator, a valid user, to update two columns of the table VENDOR. If you do not mention the column names, on all of the columns the privilege will be granted.

The following example shows how to grant privileges to all the users.

```
grant select
  on customer
  to public;
```

It permits everybody to look at the CUSTOMER table.

The last example shows how to grant REFERENCE privilege.

```
grant references (vendor_code)
  on vendor
  to analyst;
```

Here the analyst may define referential integrity constraints that refer to the vendor_code column. However, since the GRANT statement lists only on the specified column, analyst cannot perform operations on any other columns of the VENDOR table.

Points to remember on GRANT statement

Public is a pseudo-usrid that identifies every usrid that is known or will be known to the DBMS.

Grantor should have grant privileges on all specified privileges.

If ‘WITH GRANT OPTION’ is not specified, privileges cannot be passed onto others by the user who is getting the privilege.

Cancellation of Authorisation

If REVOKE is used on designated object it means the withdrawl of the privileges from the designated user. Question of revoking does arise only when the user either directly or indirectly has been granted the privilege.

If you revoke a privilege from a user, the system removes the privilege from the user’s privilege domain. The user cannot exercise the privilege as soon as it is removed.

If you revoke a privilege from public, the system removes the privilege from

Revoking Privileges

Examples

```
revoke select
    on item
      from storekeeper;
```

```
revoke all
    on purchase_order
      from md;
```