Relational Model

Relational Model Concepts :- This model represents the database as a collection of relations. When a relation is thought of as a table of values, each row in the table represents a collection of related data values. The column names specify how to interpret the data values in each row, based on the column each value is in. All values in a column are of the same data type.

In relational model, a row is called a tuple and a column header is called an attribute.

Domains, Attributes, Tuples and Relations :-

A domain D is a set of atomic values, which means that each value in the domain is indivisible as far as the formal relational model is concerned

Eg :- Names: The set of character strings that represents names of persons.

GPA : possible values of computed grade point-averages each must be a real no between 0 and 4.

A data type or format is also specified for each domain

A domain is given a name, data type and format.

A relational schema R denoted by $R(A_1, A_2 \ldots, A_n)$, is made up of a relation name R and a list of attributes, $A_1, A_2 \ldots A_n$.

Each attribute is the name of a role played by some domain D in the relation schema R. D is called the domain of A_i and is denoted by $dom(A_i)$. The degree of a relation is the number of attributes n of its relation schema.

Eg :- STUDENT (Name, Ssn, Home_phone, Address, Off_phone, Age, GPA)

Relation name R - STUDENT, degree - 7.

dom (Name) = Names ; dom (Ssn) = Social_security_numbers

dom (Home_phone) = USA_phone_numbers etc.

A relation (or relation state) r of the relation schema $R(A_1, A_2, \ldots, A_n)$, also denoted by $r(R)$, is a set of n-tuples $r = \{t_1, t_2, \ldots, t_m\}$.

Each n-tuple t is an ordered list of n values $t = \langle v_1, v_2, \ldots, v_n \rangle$, where each value v_i, $1 \le i \le n$, is an element of $dom(A_i)$ or is a special NULL value.

The i^{th} value in tuple t, which corresponds to the attribute A_i, is refered as $t[A_i]$ or $t.A_i$.

Eg:- STUDENT - relation name. Attributes

Name	SSN	Home-Phone	Address	Office-Phone	Age	GPA
Bayer	305-61-2345	817 113 1616	Grandhi Nagar	NULL	19	3.21
Kim	381-62-1245	817 375 4409	125, T Nagar	NULL	18	2.99
Davidson	422-11-2320	NULL	3452, Main Road	817 749 1253	25	3.53
Rohan	489-22-1100	817 3769821	265, Lane.	817 749 1492	28	3.93
Newton	533-69-1238	817 839 1461	7384, T Nagar	NULL	19	3.23

Tuples

Each tuple in the relation represents a particular student entity.

The current relation state reflects only the valid tuples that represent a particular state of the real world.

Characteristics of Relations:-

Ordering of Tuples in a Relation:- A relation is defined as a set of tuples. Tuples in a relation do not have any particular order, but they are displayed in a certain order.

Many tuple orders can be specified on the same relation.

For the above eg:, tuples can be ordered by values of Name, SSN, Age or some other attribute.

Ordering of values within a Tuple and an alternative definition of a relation:- An n-tuple is an ordered list of n values, so the ordering of values in a tuple is important.

represent facts about relationships.

An alternative definition of a relation can be given, making the ordering of values in a tuple unnecessary.

In this definition, a relation schema $R = \{A_1, A_2, \ldots A_n\}$ is a set of attributes, and a relation state $r(R)$ is a finite set of mappings $r = \{t_1, t_2 \ldots t_m\}$, where each tuple t_i is the mapping from R to D and D is the union of the attribute domains.

$t[A_i]$ must be in $dom(A_i)$ for $1 \le i \le n$ for each mapping t in r. Eg:- $t = <(Name, DAkison), (SSn, 422\text{-}11\text{-}2320) \ldots>$
$t = <(Address, 3452\ MGRoad), (Name, Davison) \ldots>$

According to this definition of tuple as a mapping, a tuple can be considered as a set of (<attribute>, <value>) pairs, where each pair gives the value of the mapping from an attribute A_i to a value V_i from $dom(A_i)$

Values and NULLs in the Tuples:- Each value in a tuple is an atomic value. Composite and multivalued attributes are not allowed. This model is sometimes called the flat relational model.
Multivalued attributes must be represented by seperate relations

NULL values are used to represent the values of attributes that may be unknown or many not apply to a tuple. A special value, called NULL, is used in these case.
Several meanings for NULL values are Value Unknown, Value exists but is not available., or attribute does not apply to this tuple.
The exact meaning of a NULL value governs how it fares during arithmetic aggregations or comparisons with other values.

Interpretation of a Relation:- The relation schema can be interpreted as a declaration or a type of assertion.
Each tuple in the relation can be interpreted as a fact or a particular instance of the assertion.
Some relations may represent facts about entities and some may represent facts about relationships.

Hence the relational model represents facts about entities and relationships uniformly as relations. The ER model represents entities and relationships.

An alternative interpretation of a relation schema is as a predicate. The values in each tuple are interpreted as values that satisfy the predicate.

For eg, the predicate STUDENT (Name, Ssn ...) is true for the five tuples in relation STUDENT. These tuples represent five different propositions or facts in the real world.

An assumption called the closed world assumption states that the only true facts in the universe are those present within the extension of the relation.

Relational Model Constraints and Relational Database Schemas:-

There are many restrictions or constraints on the actual values in a database state. These constraints are derived from the rules in the miniworld.

Constraints are divided into three main categories:

1. Constraints that are inherent in the data model are known as inherent model-based or implicit constraints -
2. Constraints that can be directly expressed in schemas of the data model are known as schema-based or explicit constraints.
3. Constraints that must be expressed and enforced by the application programs are known as application-based or semantic or business rules.

The characteristics of relations that we discussed above are the implicit constraints of the relational model.

For eg, the constraint that a relation cannot have duplicate tuples is an implicit constraint.

The schema based or explicit constraints include domain constraints, key constraints, constraints on NULLs, entity integrity constraints and referential integrity constraints.

Domain Constraints :— These constraints specify that within each tuple, the value of each attribute A must be an atomic value from the domain dom(A). The data types associated with domains are numeric, characters, booleans, fixed-length strings, variable length strings, date, time and other special data types.

Key Constraints and Constraints on NULL values :—

In the formal relational model, a relation is defined as a set of tuples. All elements of a set are distinct; hence all tuples in a relation must also be distinct. This means that no two tuples can have the same combination of values for all their attributes.

Then for any two distinct tuples t_1 and t_2 in a relation state r of R, $t_1[SK] \neq t_2[SK]$

Any such set of attributes SK is called a Superkey of the relation schema R. A Super key SK specifies a uniqueness constraint that no two distinct tuples in any state r of R can have the same value for SK.

Every relation has at least one default Super key.

A Key K of a relation schema R is a super key of R which satisfies the following two properties :

1. Two distinct tuples in any state of the relation cannot have identical values for the attributes in the key.
2. It is a minimal Super Key — that is, a super key from which we cannot ~~have identical~~ remove any attributes and still have the uniqueness constraint in condition 1 hold.

The first property applies to both keys and super keys, the second property is required only for keys. Hence a key is also a super key but not vice versa.

Eg:- {SSN} - Key

{SSN, Name, Age} - Super key but not a key.

In general, any super key formed from a single attribute is also a key. (time invariant)

A key with multiple attributes must require all its attributes together to have the uniqueness property. The value of a key attribute can be used to identify uniquely each tuple corresponding in the relation.

A relation schema may have more than one key. Each of the key is called a candidate key. One of the candidate keys can be designated as a primary key of the relation. The other candidate keys are designated as unique keys.

To permit or not to permit NULL values, constraint can be specified on that particular attributes to be NULL or NOT NULL.

Relational Databases and Relational Database schemas:-

A relational database schema S is a set of relational schemas $S = \{R_1, R_2 \cdots R_m\}$ and a set of integrity constraints IC.

A relational database state DB of S is a set of relation states $DB = \{r_1, r_2 \cdots r_m\}$ such that each r_i is a state of R_i and such that the r_i relation states satisfy the integrity constraints specified in IC.

A database state that does not obey all the integrity constraints is called an invalid state, and the one that satisfies is called a valid state.

Each relational DBMS must have a DDL for defining a relational database schema.

Integrity, Referential integrity and foreign keys :-

The entity integrity constraint states that no primary key value can be NULL

Key constraints and entity integrity constraints are specified on individual relations.

The referential integrity constraint is specified between two relations and is used to maintain the consistency among tuples in the two relations.

The conditions for a foreign key specify a referential integrity constraint between the two relation schemas R_1 and R_2.

A set of attributes FK in relation schema R_1 is a foreign key of R_1 that references relation R_2 if it satisfies the following rules :

1. The attributes in FK have the same domain as the primary key attributes PK of R_2; the attributes FK are said to reference or refer to the relation R_2.
2. A value of FK in a tuple t_1 of the current state $r_1(R_1)$ either occurs as a value of PK for some tuple t_2 in the current state $r_2(R_2)$ or is NULL.

R_1 is called the referencing relation and R_2 is the referenced relation. If these two conditions hold, a referential integrity constraint from R_1 to R_2 is said to hold.

A foreign key can refer to its own relation.

The transaction concept :- A transaction is an executing program that includes some database operations, such as reading from the database, or applying insertions, deletions or updates to the database. At the end of the transaction, it must leave the database in a valid or consistent state that satisfies all the constraints specified on the database schema.

A large number of commercial applications running against relational databases in OLTP systems are executing transactions at rates that reach several hundreds per second.

SQL Data definition and Data types :- An SQL schema is identified by a schema name, and includes an authorization identifier to indicate the user or account who owns the schema, as well as descriptors for each element in the schema. Schema elements are tables, constraints, views, domains and other constructs.

DDL Commands - create, alter, drop
DML " - insert, update, delete

Eg :- Company database

Employee — Fname, Lname, Ssn, Bdate, address, sex, salary
Super-ssn, Dno.

Department — Dname, Dnumber, Mgr_ssn, Mgr_start_date.

Dept_locations - Dnumber, Dlocation.

Project — Pname, Pnumber, plocation, Dnum.

works_on — Essn, Pno, Hours

Dependent - Essn, Dependent_name, sex, Bdate, Relationship.

Complex SQL Queries :-

NULL values :- Each individual NULL value is considered to be different from every other NULL value in the various database records. When a NULL is involved in a comparison operation, the result is considered to be unknown.

SQL allows that check whether an attribute value is NULL.

Eg :- Retrieve the names of all employees who do not have supervisors.

Select Fname, Lname from employee where super_ssn is NULL.

Nested Queries :- Nested queries are complete select-from-where blocks within the where clause of another query. The other query is called the outer query.

Eg:- 1. Select the project numbers of projects that have an employee with last name 'smith' as manager.

Select distinct pnumber from project where pnumber in
(select pnumber from project, Department, employee
where dnum = dnumber and
mgr_ssn = ssn and Lname = 'smith')

In general, the nested query will return a table. which is a set or multiset of tuples.

2. List the Essns of all employees who work the same project_hours on some project that 'John smith' works on.

Eg:- Select distinct Essn. from works_on where.
(Pno, hours) in (select Pno, hours from
works_on where
Essn = '123456789')

3. In addition to the IN operator, a number of comparison operators [~~Comparison operators~~] can be used to compare a single value V (an attribute name) to a set or multiset V (a nested query).

The =ANY operator returns true if the value V is equal to some value in the set V and is hence equivalent to IN.

Other operators that can be combined with ANY are >, >=, <, <= and <>. The key word ALL can also be used with these operators. For eg (>ALL) returns true if the value V is greater than all the values in the set.

Eg:- 1. List the names of employees whose salary is greater than the salary of all the employees in department 5.

Select Lname, Fname. from employee where
Salary > ALL (select salary from
employee where Dno = 5);

2. Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.

```
Select E.Fname, E.Lname from Employee as E
    where E.Ssn IN (Select Essn from Dependent as D
        where E.Fname = D.Dependent_name
        and E.Sex = D.Sex).
```

Correlated Nested Queries :- Whenever a condition in the WHERE clause of a nested query references some attribute of a table declared in the outer query, the two queries are said to be correlated. A correlated subquery is evaluated once for each row processed by the parent statement, which can be any of SELECT, DELETE or UPDATE

The above query can also be rewritten as.

```
Select E.Fname, E.Lname from employee as E, dependent D
    where E.Ssn = D.Essn and E.Sex = D.Sex
        and E.Fname = D.Dependent_name;
```

The EXISTS function in SQL is used to check whether the result of a correlated nested query is empty or not. The result of EXISTS is a Boolean value TRUE if the nested query result contains atleast one tuple, or FALSE if the nested query result contains no tuples.

The above query can also be rewritten as

1.
```
Select E.Fname, E.Lname from Employee as E
    where EXISTS (select * from dependent as D
        where E.Ssn = D.Essn and
        E.Sex = D.Sex and
        E.Fname = D.Dependent_name)
```

NOT EXISTS returns TRUE if there are no tuples in the result of nested query, and it returns FALSE otherwise.

2. Retrieve the names of employees who have no dependents.

```
Select Fname, Lname from Employee.
    where NOT EXISTS (select * from dependent
        where Ssn = Essn)
```

3. List the names of managers who have atleast one dependent

```
Select Fname, Lname from employee where
    exists ( select * from dependent
             where ssn = Essn ) and
    exists ( select * from department
             where ssn = Mgr_ssn )
```

4. Retrieve the name of each employee who works on all the projects controlled by department no 5.

```
( select Fname, Lname from employee where
    not exists ( select * from works_on B
                 where ( B.Pno in ( select Pnumber
                                    from project
                                    where Dnum = 5 )
                 and not exists
                     ( select * from works_on C
                       where C.Essn = Ssn
                       and C.Pno = B.Pno ))) ;
```

Select each employee such that there does not exist a project controlled by dept. 5 that the employee does not work on.

(or)

```
select Fname, Lname from employee where
    not exists ( ( select Pnumber from project where Dnum = 5 )
                 except ( select Pno from works_on
                          where Ssn = Essn ) )
```

5. Retrieve the name of all employees who have two or more dependents

```
select Lname, Fname from employee where ( select count(*) from dependent
                                          where Ssn = Essn ) >= 2
```

Joined Tables in SQL and outer joins :- This concept was introduced in SQL to permit users to specify a table resulting from a join operation in the FROM clause of a query.

Ex:- Retrieve the name and address of every employee who works for the 'Research' department.

```
Select Fname, Lname, Address
from ( employee join department on Dno = Dnumber )
where Dname = 'Research'
```

Different types of joins – Equijoin, Natural join, Outer join.

In a Natural join on two relations R and S, no join condition is specified; an implicit equijoin condition for each pair of attributes with the same name from R and S is created. Each such pair of attributes is included only once in the resulting relation.

Eg:- Select Fname, Lname, Address
from (Employee Natural join (Department as dept (Dname, Dno, Mssn, Msdate)))
where Dname = 'Research'

In the above examples the names of the join attributes are renamed ~~to~~ inorder to match them in both the relations.

The default type of join in a joined table is called an inner join, where a tuple is included in the result only if a matching tuple exists in the other relation.

If the user requires that all tuples be included, an OUTER JOIN must be used explicitly.

i. Retrieve names of employees who have supervisors.

Eg:- Select E.Lname as Emp_name,
S.Lname as Sup_name
FROM (EMPLOYEE as E left outer join EMPLOYEE as S
ON E.Super_ssn = S.Ssn)

Left outer join – Every tuple in the left table must appear in the result. If it does not have a matching tuple, it is padded with NULL values for the attributes of the right table.

Right outer join – Every tuple in the right ~~tuple~~ table must appear in the result. If it does not have a matching tuple, it is padded with NULL values for the attributes of the ~~right~~ left table.

CROSS JOIN is used to specify the Cartesian product operation which generates all possible tuple combinations.

The GROUP BY and HAVING clauses :-

The Group by clause specifies the grouping attributes so that the value resulting from applying each aggregate function to a group of tuples appears along with the value of the grouping attribute(s).

Eg :- 1. For each department, retrieve the dept no, the number of employees in the department, and their average salary.

```
select  Dno, count(*), avg(salary)
  from  employee  group by dno;
```

2. For each project, retrieve the project no, the project name and the number of employees who work on that project.

```
Select  Pnumber, Pname, Count(*)
 from   project, works_on
 where  Pnumber = Pno
 group by  Pnumber, Pname.
```

3. For each project on which more than the employee work, retrieve the project no, the project name and the number of employees who work on the project.

```
select  pnumber, Pname, Count(*)
from    project, works_on
where   Pnumber = Pno
group by  Pnumber, Pname
having  count(*) > 2
```

4. For each project, retrieve the project number, the project name and the number of employees from dept 5 who work on the project.

```
select Pnumber, Pname, Count(*)
from project, works_on, employee
where Pnumber=Pno and ssn=Essn and Dno=5 group by Pnumber, Pname
```

5. Count the total number of employees whose salaries exceed 40,000 in each dept, but only for departments where more than five employees work.

```
Select Dname, Count(*)
from Department, Employee
where Dnumber = Dno and Salary > 40000
group by Dname
having Count(*) > 5
```

6. For each department that has more than five employees, retrieve the department no and the number of its employees who are making more than 40,000

```
Select Dnumber, Count(*)
from Department, Employee
where Dnumber = Dno and Salary > 40000 and
      ( Select Dno from Employee
        group by Dno
        having Count(*) > 5 )
```

Formal Relational Languages

Relational Algebra: The basic set of operations for the relational model is the relational algebra. These operations enable a user to specify basic retrieval requests as relational algebra expressions.

The relational algebra provides a formal foundation for relational model operations. It is also used as a basis for implementing and optimizing queries in the query processing and optimization modules that are integral parts of RDMSs.

Operations of Relational Algebra :-

Select :- Selects all tuples that satisfy the selection condition from a relation R.

Notation :- $\sigma_{\langle \text{selection condition} \rangle}(R)$

Eg :- Select the tuples for all employees who either work in dept 4 and make over 25,000 per year, or work in dept 5 and make over 35000.

$\sigma_{(Dno = 4 \text{ and } Salary > 25000) \text{ or } (Dno = 5 \text{ and } Salary > 35000)}(emp)$

Project :- produces ~~all tuples that satisfy~~ a new relation with only some of the attributes of R, and removes duplicate tuples.

Notation :- $\pi_{\langle \text{attribute list} \rangle}(R)$

Eg :- List each employee's first and last name and salary.

$\pi_{Fname, Lname, Salary}(emp)$.

Rename :- Relation names and attribute names can be renamed using ρ operator.

Notation :- $\rho_S(R)$ or $\rho_{S(B_1, B_2, \ldots, B_n)}(R)$ or $\rho_{(B_1, B_2, \ldots, B_n)}(R)$

eg :- $\rho_{emp}(employee)$

Relational algebra operations from Set Theory :-

UNION, INTERSECT, MINUS (SET DIFFERENCE)

UNION :- Produces a relation that includes all the tuples in ~~both~~ R_1 ~~and~~ or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible - Same no. of attributes and each pair of attributes from same domain.

Notation :- $R_1 \cup R_2$

eg :- Retrieve the SSNs of all employees who either work in Dept 5 or directly supervise an employee who works in dept 5.

Dept5_emps $\leftarrow \sigma_{dno=5}(emp)$

Result1 $\leftarrow \pi_{ssn}$ (Dept5_emps)

Result2(ssn) $\leftarrow \pi_{super\_ssn}$ (Dept5_emps)

Result $\leftarrow$ Result1 $\cup$ Result2

Intersection :- produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.

Notation :- $R_1 \cap R_2$

eg :- Student $\cap$ Instructor

Difference :- produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.

Notation :- $R_1 - R_2$

eg :- Student - Instructor

Cartesian product :- Produces a relation that has the attributes of R_1 and R_2 and includes all tuples all combinations of tuples from R_1 and R_2

Notation : $R_1 \times R_2$

Binary relational operations JOIN and Division :—

The JOIN operation denoted by ⋈, is used to combine related tuples from two relations into single "longer" tuples.

Notation :- $R \bowtie_{\text{<join-condition>}} S$

The result of the join is a relation Q with n+m attributes $Q(A_1, A_2 \ldots A_n, B_1, B_2 \ldots B_m)$ in that order; Q has one tuple for each combination of tuples - one from R and one from S - whenever the combination satisfies the join condition.

Eg :- Retrieve the name of the manager of each department

$$\text{Dept\_mgr} \leftarrow \text{Dept} \bowtie_{\text{mgr-ssn} = \text{ssn}} \text{Emp}$$

$$\text{Result} \leftarrow \pi_{\text{Dname, Lname, Fname}} (\text{Dept\_mgr})$$

The main difference between Cartesian product and join is:

In join, only combinations of tuples satisfying the join condition appear in the result, whereas in the Cartesian product all combinations of tuples are included in the result.

A join operation with a general join condition is called a Theta join (θ) ($\theta = \{=, <, \leq, >, \geq, \neq\}$)

A join, where the only comparison operator used is =, is called an equi join.

A natural join denoted by * was created to get rid of the second attribute in an equijoin condition.

Eg :- proj_dept ← project * Dept

For natural join the two join attributes have to be same name in both relations. If not, a renaming operation is applied first

eg :- Proj_dept ← project * ρ(Dname, Dnum, mgr_ssn) (Department)

The attribute Dnum is the join attribute here.

A Complete set of Relational Algebra operations :-

The set of relational algebra operations $\{\sigma, \pi, \cup, \rho, -, \times\}$ is a complete set; that is, any of the other relational algebra operations can be expressed as a sequence of operations from this set.

Eg:- $R \cap S = (R \cup S) - ((R - S) \cup (S - R))$

$R \bowtie_{<condition>} S = \sigma_{condition}(R \times S)$

The Division operation :- The division operation is applied to two relations $R(Z) \div S(X)$, where the attributes of R are a subset of the attributes of S; that is, $X \subseteq Z$.

Eg:- Retrieve the names of employees who work on all the projects that 'John Smith' works on

$SMITH \leftarrow \sigma_{Fname='John' \text{ and } Lname='Smith'}(EMP)$

$SMITH\_PNOS \leftarrow \pi_{Pno}(WORKS\_ON \bowtie_{Essn=Ssn} SMITH)$

$SSN\_PNOS \leftarrow \pi_{Essn, Pno}(WORKS\_ON)$

$SSNS \leftarrow SSN\_PNOS \div SMITH\_PNOS$

$RESULT \leftarrow \pi_{Fname, Lname}(SSNS * EMP)$

SSN_PNOS

ESSN	PNO
123456789	1
123456789	2
666884444	3
453453453	1
453453453	2
333445555	2
333445555	3
333445555	10
999887777	30
999887777	10
987987987	30
987654321	20
888665555	20

SMITH_PNOS

PNO
1
2

SSNS

SSN
123456789
453453453

The division operation can be expressed as a sequence of π, $\times$ and $-$ operations as follows:

$T_1 \leftarrow \pi_Y(R)$, $T_2 \leftarrow \pi_Y((S \times T_1) - R)$

$T \leftarrow T_1 - T_2$

Let Y be the set of attributes of R that are not attributes of S; i.e $Y = Z - X$

The result of division is a relation $T(Y)$ that includes a tuple t if tuples t_R appear in R with $t_R[Y] = t$ and with $t_R[X] = t_S$ for every tuple t_S in S.

Eg:- $X = \{A\}$ $Y = \{B\}$ and $Z = \{A, B\}$

R

A	B
a_1	b_1
a_2	b_1
a_3	b_1
a_4	b_1
a_1	b_2
a_3	b_2
a_2	b_3
a_3	b_3
a_4	b_3

S

A
a_1
a_2
a_3

$R \div S \rightarrow$

T

B
b_1
b_4

For a tuple t to appear in the result T of the division, the values in t must appear in R in combination with every tuple in S.