Module - II → Relational Model, Relational Algebra

Module-II	Relational Model, Relational Algebra	9Hrs

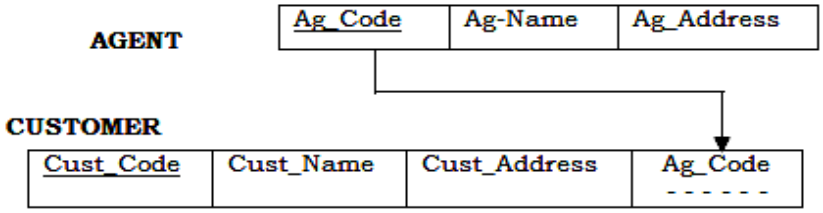
Relational Model: Introduction to the Relational Model – Integrity Constraints over Relations, Enforcing Integrity constraints, querying relational data, Logical data base Design, Views.

Relational Algebra: Introduction to Relational algebra, selection and projection, set operations, renaming, joins, division.

The Relational Model

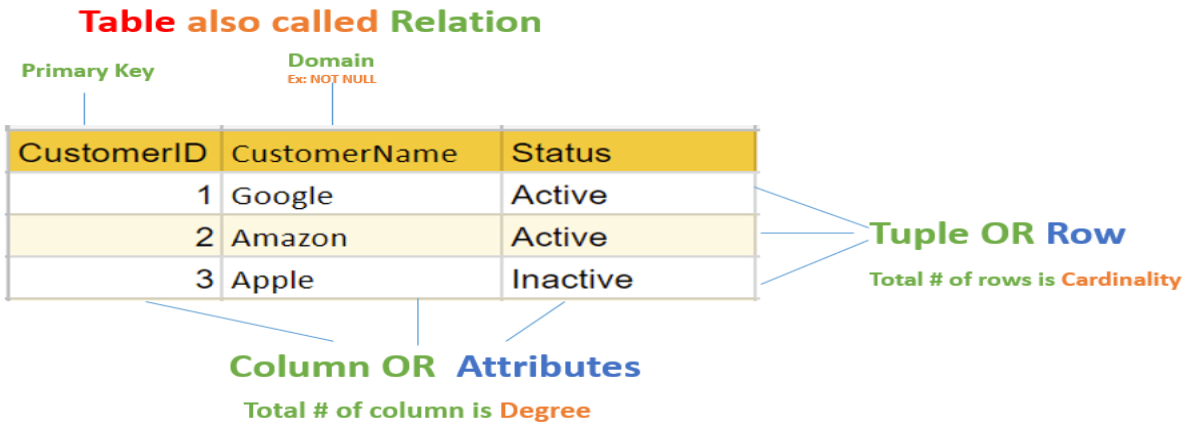
Relational Model:-

- The Relational Model was developed by Dr.E.F.Codd in 1970.
- It is the most common model to represent the data in the database applications.
- It supports different types of relationships
- It uses referential integrity constraints to establish relationship between tables.
- In this, tables are called ***relations***, columns are called ***fields***, and rows are called ***tuples***
- In this model the most widely used language is ***SQL***



Relational Model Concepts in DBMS

1. **Attribute:** Each column in a Table. Attributes are the properties of an entity which define a relation. e.g., Student_Rollno, NAME,etc.
2. **Tables** – In the Relational model the relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.
3. **Tuple** – It is nothing but a single row of a table, which contains a single record.
4. **Relation Schema:** A relation schema represents the name of the relation with its attributes.
5. **Degree:** The total number of attributes which in the relation is called the degree of the relation.
6. **Cardinality:** Total number of rows present in the Table.
7. **Column:** The column represents the set of values for a specific attribute.
8. **Relation instance** – Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.
9. **Relation key** – Every row has one, two or multiple attributes, which is called relation key.
10. **Attribute domain** – Every attribute has some pre-defined value and scope which is known as attribute domain



Integrity Rules (Or) Integrity Constraints

An integrity constraint is a rule that cannot be violated by the user. The constraints are used to prevent invalid data entry into the table. It enforces rules for the columns in a table. In SQL the integrity constraints are classified into 3 types. They are

1) Domain integrity constraints
2) Entity Integrity constraints
3) Referential integrity constraints

1. Domain Integrity Constraints:

A domain is a set of values that may assign to a column. Domain Integrity Constraints are 2 types. They are

i) Not Null
ii) Check

i) Not Null:

The value which is absent is called NULL. The NOT NULL constraint is used to avoid NULL values into a column of a table. The NOT NULL constraint allows duplicate values.

Eg: SQL> Create table student(Sno number(10) NOT NULL,sname varchar2(20));

In the above example, Sno column should not allow null values.

ii) Check:

The Check constraint is used to specify a particular condition to a column. It checks the condition satisfied or not.

Eg: SQL> Create table student(Sno number(10), Sname varchar2(20),
Sfee number(8,2) CHECK (Sfee>10000));

In the above example, the column 'Sfee' is allows only the values, which are greater than 10000.

2. Entity Integrity Constraints:

The entity integrity constraints are used to avoid invalid data into column of a table. There are 2 types

i) Unique
ii) Primary Key

i) Unique:

It is used to prevent duplicate values to a particular column. i.e., It does not allow duplicate values, but allows NULL values.

Eg: SQL> Create table emp(E_no number(3) UNIQUE,
Ename varchar2(30), Deptno number(5));

ii) Primary Key:

A filed (or) combination of fields used to identify a single record will be called as Primary key. It does not allow duplicate values and null values.

Eg: SQL> Create table emp(E_no number(3) PRIMARY KEY,
Ename varchar2(30), Deptno number(5));

3. Referential Integrity Constraints:

The referential integrity constraint is used to validation of a rule between two tables. A filed which references the primary key of another table will be called as *Foreign key*.

To implement this type of constraint the Master (parent) table must contain primary key and the same column in the Detailed (child) table as a reference of primary key known as foreign key.

Master Table:

Eg: SQL> Create table Dept(Deptno number(5) PRIMARY KEY,
Dname varchar2(20), Locaction varchar2(10));

Detailed Table:

Eg: SQL> Create table Emp(E_no number(3) PRIMARY KEY, **Ename varchar2(20),**
Deptno number(5) references Dept(Deptno));

The records from the Master table will not be deleted until the corresponding records are deleted from the detail table. Similarly new records can not be inserted into the detail table, until the corresponding values are inserted in the master table.

The Relational Data Model Keys (Or) Keys Used in Relational Data Model

The keys are used for identifying a single row inside a relation. A key consists of one or more attributes, that determine other attributes. The key's role is based on a concept known as "determination". Consider the statement "A determines B". It can be represented as A➔B. In other words "The attribute B is functionally dependent on the attribute A".

In relational environment the keys are played vital role. Such keys are

i) Primary Key ii) Composite Key iii) Foreign Key iv) Candidate Key

v) Secondary Key vi) Super Key

i) Primary Key:- A field which is used for identifying a single unique record will be called as "Primary key" **OR** The key will not allow duplicate and Null values in the table is called "Primary Key".

Ex:-

Empno	**Ename**	**Sal**

EMPLOYEE

Empno➔ Ename, Sal

Here "Empno" is a primary key field.

ii) Composite Key:- The combination of fields are used for identifying a single unique record will be called as "Composite Key".

Ex:- STUDENT

Sno	Sname	College _ Code

Sno,College_code➔ Sname

Here "Sno, College_code" is called Composite Key

iii) Foreign Key:- A field, which references the primary key of another table will be called as "Foreign Key". It is used for maintaining the relationship between the tables.

Ex:-

Deptno	Dname	Loc

DEPARTMENT

EMPLOYEE

Empno	Ename	Job	Sal	Deptno.

In Employee table, "Deptno" is called foreign Key.

iv) Candidate Key :- A field, which is used to identify each row of table uniquely will be called as "Candidate Key". If the table has more than one candidate key, one of them will be called primary key of the table, and the rest are called "Alternate Keys"

Ex:- STUDENT

Sno	Sname	Course	Date_Completed

Sno, course➔ Date_Completed

Sno ➔Sname

Here "Sno, Course" is called candidate key

v) Secondary Key:- A key which is used strictly for data retrieval purpose. It is used alternative of primary key.

Ex:- CUSTOMER

Cno	Cname	Phno

Here "Cno" is a Primary key, combination of "Cname, Phno" is called secondary key.

vi) Super Key :- Different set of attributes which are able to identify any row in the database is known as "Super key". A Candidate key is a subset of "Super Key"

Virtual tables (or) Views (or) Stored queries

A View is a logical window of the physical database i.e, it is a mask of the table. A view takes the output of query. So it is also called as "stored query". A view is one of the database object. A view on which it is created is called a "base table" and that view is called "virtual table".

Generally View is created based on select query. The query can contain columns computed columns, aliases and aggregate functions from one or more tables. These views are called "complex (or) Non-Updatable" views

Advantages of Views:-

- Views provide the security in the Data base. Because the view can restrict users to only specified columns and rows in a table
- Views hides the base table name
- Views requires less storage space i.e It does not contain any data.
- View simplifies queries i.e, it avoid the reconstruction of complex queries.

Creating a View:- In SQL "Create View" command is used to create a view on one or more base tables.

Let us consider the following EMP table we can create a view

EMP_ID	E_NAME	E_SALARY
1	Smith	6500
2	Allen	8500
3	Martin	4500

Syn:- Create view<view name>as select query;

Ex:- 1) Create view v1 as select *from emp;

Now, we can display view v1 in a similar way as we display an actual table.

Eg: Select * from v1;

EMP_ID	E_NAME	E_SALARY
1	Smith	6500
2	Allen	8500
3	Martin	4500

2) Create view V2 as select EMP_ID, E_NAME, from emp;

Now, we can display view v2 in a similar way as we display an actual table.

Eg: Select * from v2;

EMP_ID	E_NAME
1	Smith
2	Allen
3	Martin

3) Create view v3 (eno,empname,empsalary) as select emp_id, e_name, e_salary from emp where e_name= 'Martin';

Now, we can display view v3 in a similar way as we display an actual table.

Eg: Select * from v3;

ENO	EMPNAME	EMPSALARY
3	Martin	4500

Insert record into View:- In SQL views allow to "Insert" records into existing virtual tables, insert data into Views is similar to insert records into table.

Ex:- insert into v1 values('&emp_id','&e_name','&e_salalry');

SQL> Select *from v1;

EMP_ID	E_NAME	E_SALARY
1	Smith	6500
2	Allen	8500
3	Martin	4500
4	Peter	6800

Updating a View:-

It is a view that can be used to update columns in the base table. i.e., A view that allows DML operators, those views are called "updatable views"

After creating the view we can used the DML command to update the view.

Ex:-Update v1 set e_salary=5000 where emp_id=3;

Dropping Views:- In SQL, "drop view" command is used to remove the views from the database. When a view is dropped, it does not affect to the base table data. **Syn:-** Drop view<view name>;

Ex:- Drop view v2;

When ever the changes are made in a table, then automatically the changes are made in a view

Some Restrictions/Disadvantages of views:-

1) Aggregate functions can not be used or Group by cannot be used
2) Set operators cannot be used i.e union, intersect and minus etc.

3) The views must be created only a “single table”

4) The “Primary Key” and “Not Null” columns must be included in views

5) “Sub queries” must not be included

6) The “distinct” and “having” clauses can not be included

Querying Relational Data in DBMS

A relational database query is a question about the data, and the answer consists of a new relation containing the result. For example, we might want to find all students AGE less than 18 or all students enrolled in perticular course.

The ***SELECT*** statement is used to fetch the data from a database table which returns this data in the form of a result table. These result tables are called result-sets.

Syntax in Mysql

```
SELECT column1, column2, ...  FROM table_name;
```

If you want to select all the fields available in the table, use the following syntax:

Syntax in Mysql

```
SELECT * FROM table_name;
```

The symbol ´*´ means that we retain all fields of selected tuples in the result.

We can retrieve rows corresponding to students who are younger than 18 withthe following SQL query:

```
Ex:- SELECT * FROM Students WHERE age < 18;
```

The condition **age < 18** in the WHERE clause specifies that we want to select only tuples in which the age field has a value less than 18.

In addition to selecting a subset of tuples, a query can extract a subset of the fields of each selected tuple. we can compute the student_id and First_name of students who are younger than 18 with the following query:

```
Ex:- SELECT ID,FirstName FROM Students WHERE age < 18;
```

SQL Aliases

Aliases are the temporary names given to tables or columns. An alias is created with the **AS** keyword.

Alias Column Syntax in Mysql

```
SELECT column_name AS alias_name  FROM table_name;
```

Alias Table Syntax in Mysql

```
SELECT column_name(s) FROM table_name AS alias_name;
```

```
Ex:-    SELECT studentID AS ID, FROM students AS S;
```

Aliases can be useful when:

- There are more than one table involved in a query
- Functions are used in the query
- Column names are big or not very readable
- Two or more columns are combined together

SELECT data from Multiple Tables

We can also combine information from multiple tables.

Syntax in Mysql

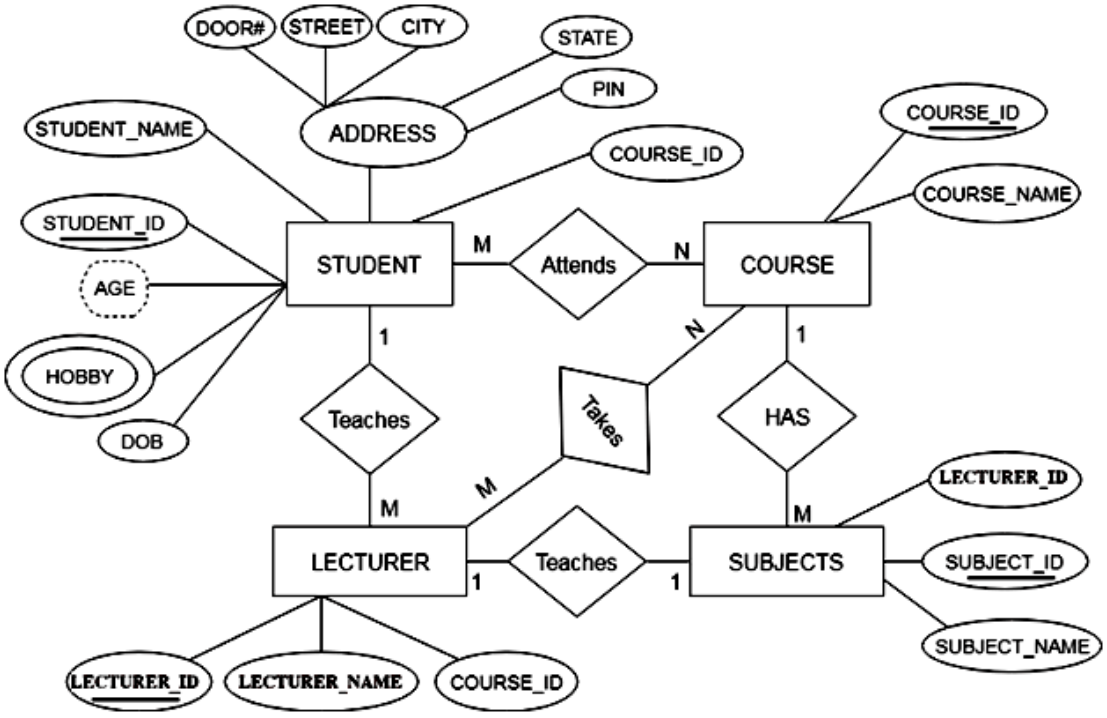
```
SELECT table1.column1, table2.column2

FROM table1, table2

WHERE table1.column1 = table2.column1;
```

```
Ex:- SELECT S.name, E.cid
FROM Students AS S, Enrolled AS E
WHERE S.sid = E.sid;
```

Logical Database Design / ER Diagram to Table Conversion/ Reducing of ER diagram to Table

- Logically the Conversion of ER_Model into Relational datamodel (tables).
- The goal of logical database design is to create well structured tables that properly reflect the company's business environment.
- The tables will be able to store data about the company's entities in a non-redundant manner and foreign keys will be placed in the tables, so that all the relationships among the entities will be supported.
- The database can be represented using the notations, and these notations can be reduced into tables.
- In the database, every entity set or relationship set can be represented in tabular form.

The ER diagram is given below:



Converting the ER diagram to the table:

- **Entity type becomes a table.**
  In the given ER diagram, LECTURE, STUDENT, SUBJECT and COURSE is individual tables.
- **All single-valued attribute becomes a column for the table.**
  In the STUDENT entity, STUDENT_NAME and STUDENT_ID form the column of STUDENT table. Similarly, COURSE_NAME and COURSE_ID form the column of COURSE table and so on.
- **A key attribute of the entity type represented by the primary key.**
  In the given ER diagram, COURSE_ID, STUDENT_ID, SUBJECT_ID, and LECTURE_ID are the key attribute of the entity.
- **The multi valued attribute is represented by a separate table.**
  In the student table, a hobby is a multi valued attribute. So it is not possible to represent multiple values in a single column of STUDENT table. Hence we create a table STUD_HOBBY with column name STUDENT_ID and HOBBY. Using both the column, we create a composite key.
- **Composite attribute represented by components.**

In the given ER diagram, student address is a composite attribute. It contains CITY, PIN, DOOR#, STREET, and STATE. In the STUDENT table, these attributes can merge as an individual column.

- **Derived attributes are not considered in the table.**

  In the STUDENT table, Age is the derived attribute. It can be calculated at any point of time by calculating the difference between current date and Date of Birth.

Ex:-

Strong Entity set with Simple attributes Attributes

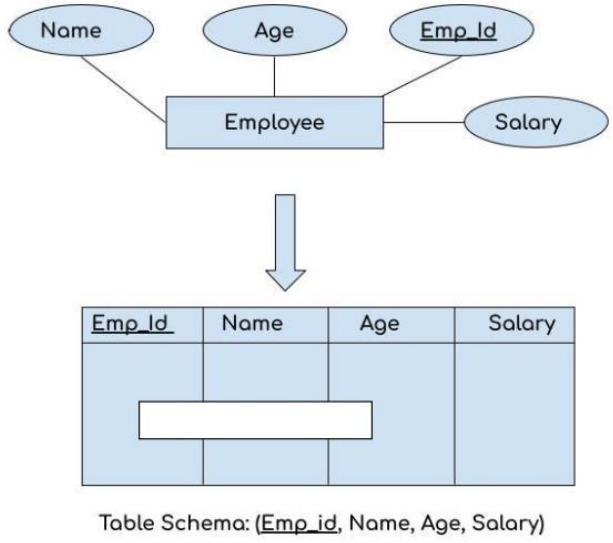


Table Schema: (Emp_id, Name, Age, Salary)

Strong Entity Set With Composite

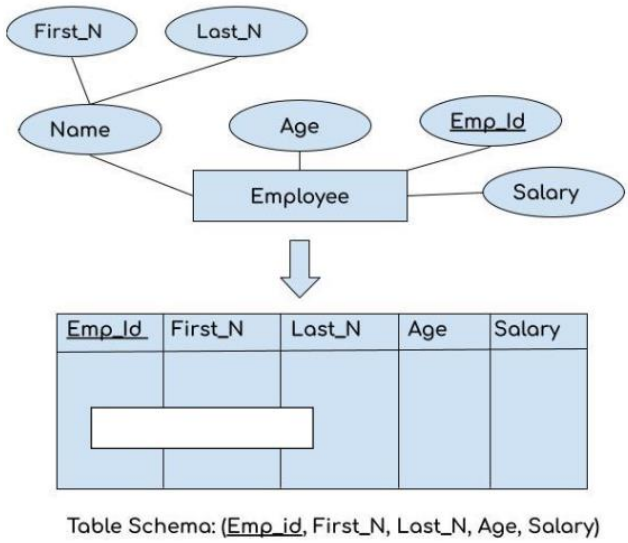
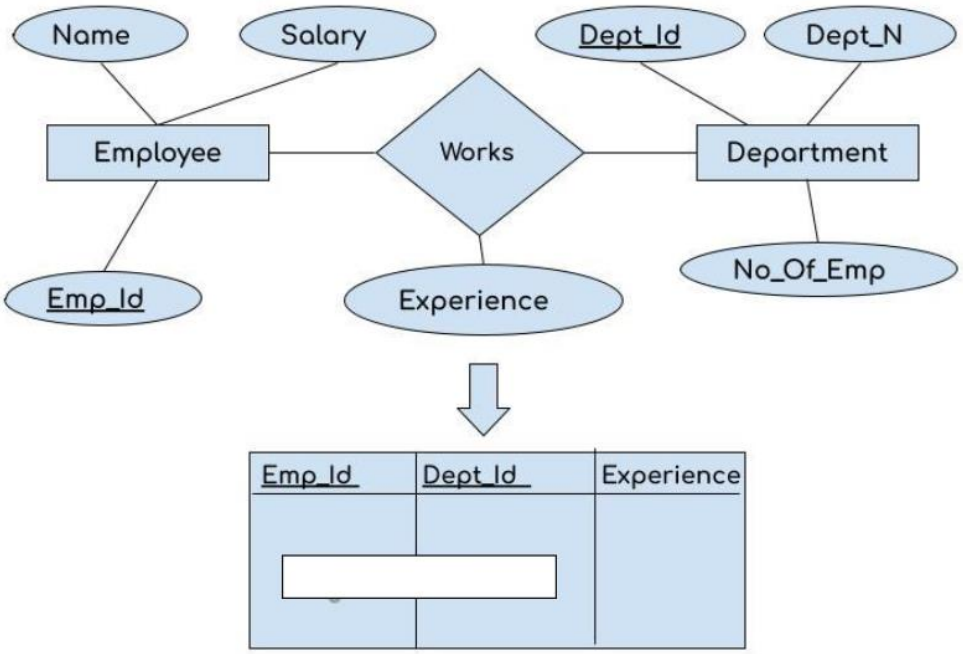


Table Schema: (Emp_id, First_N, Last_N, Age, Salary)

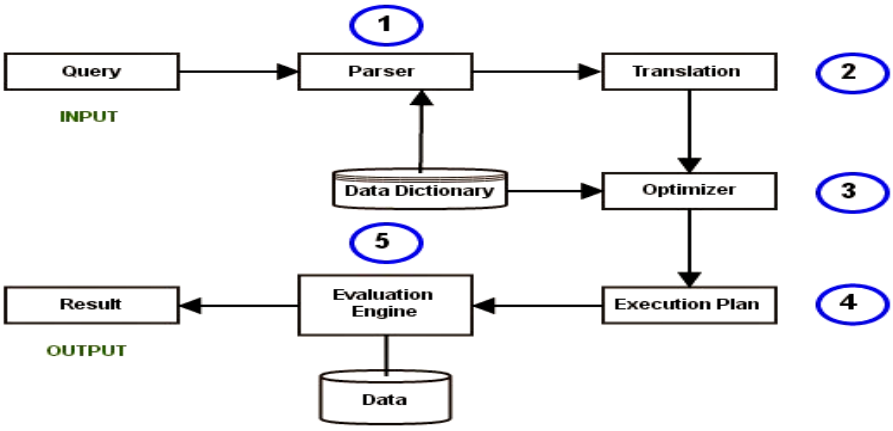
Relationship Set to Table conversion



Query Processing In DBMS

It is the step by step process of breaking and translating the high level language queries into low level language.

The following diagram shows Query processing in the DBMS



There are 4 Phases in query processing

1. Parsing and Translation
2. Query Optimization
3. Execution & Evaluation or query code generation

1. Parsing and Translation:-

In this phase the parser checks the syntax and verify the relations, attributes. Which are used in the query.

The translator, is used to translates the SQL Query into a Relational algebraic expressions.

Ex:- SELECT balance from account WHERE balance<2500;

The above query is translating high level language into relational algebraic expression as shown below

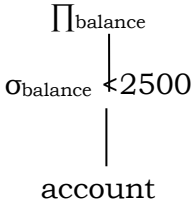
$\sigma_{balance} <2500(\prod_{balance}(account))$ (or) $\prod_{balance}(\sigma_{balance} <2500(account))$

2. **<u>Query Optimization:-</u>**

Query optimization is the process of selecting an efficient execution plan for evaluating the query.

- Query optimization provides faster query processing
- It gives less stress to the database
- It requires less cost per query
- It provides high performance of the system
- It consumes less memory

The query optimizer generates different execution plans to evaluate, and select the plan with least estimated cost. In this phase the evaluation using several statistical algorithms. The following diagram show query execution plan



3. <u>Execution & Evaluation or Query code generation:-</u>

In this phase the execution plan access the data from the database to give the final result.

The evaluation is the last phase of query processing. In this phase the evaluation engine takes a query evaluation plan, executes that plan and return the answer to the query.

In this phase the query is executed and gets the data from memory. All these process is knows as query processing.

<u>Relational Algebra</u>

The Relational Algebra is a Procedural Query Language. It consists of a set of operations that takes one or two relations as input and produce a new relation as their result. The functional operations in the relational algebra are

1. Selection(σ)
2. Projection(π)
3. Rename(ρ)
4. Cartesian Product(X)
5. Join Operations

1. SELECTION Operation(σ):

The SELECTION operation works on a single relation **R**. this operation selects tuples (rows) that will satisfy a given condition (predicate). This operation performs the row wise filtering. It is denoted by a Greek letter **Sigma** (σ).

Notation: $\sigma_{Predicate}(R)$

Here **R** is the relation, from which we want to select the rows.

Predicate is the condition, based on which we want to filter the rows.

In this operation, we use operators like Λ(and), V(or), ¬(not), =, ≠,<, ≤,>,≥.

Example:

Relation: STUDENT

Sid	Sname	GPA
100	Venu	8.1
101	Kumar	6.5
102	Naveen	8.2
103	Satvika	8
104	Sanvika	7.5

Query: List Sid, Sname and GPA of students, who are having GPA above 8.0

Expression: $\sigma_{GPA>8.0}(STUDENT)$

2. PROJECTION Operation(Π):

The PROJECTION operation works on a single relation **R**. This operation selects specific attributes (columns) of a relation. The projection operation performs a column wise filtering. It is denoted by a Greek letter pi (**Π**).

Notation: $\Pi\, a_1,a_2,a_3,\ldots a_n(R)$

Here $a_1,a_2,a_3,\ldots a_n$ are the attributes of a relation **R**.

Example: Relation: EMPLOYEE

Empno	Ename	Sal
E001	Venu	11000

E002	Kumar	12000
E003	Satvika	9000
E004	Sanvika	8500
E005	Saradha	10000

Query: List the "Ename" and "Sal" of the Employees.

Expression: $\Pi_{Ename, sal}(EMPLOYEE)$

3. RENAME operation(ρ):

The results of relational algebra are also a relation, but without any name. The Rename operation allows us to rename the output of a relation. The relation operation is denoted with Greek letter rho (ρ).

Notation: $\rho_x(E)$ Where the result of expression **E** is saved with name of **x**.

Example: We can use the rename operator to rename STUDENT relation to STUDENT1.

ρ(STUDENT1, STUDENT)

4. SET operations:

There are the following operations can be performed on relations using set theory. The set operations are performed on two relations **R** and **S**, and produce a new relation.

For set operations to be valid, the following conditions must hold-

- ✓ R and S must have the same number of attributes.
- ✓ Attributes domains must be compatible.

. Let us consider the two relations –

R

SNO	SNAME	MARKS
1	Jack	92
3	Tomy	78

S

SNO	SNAME	MARKS
2	Anil	56
1	Jack	92

a) UNION(U):

The UNION operation defines a relation, which contains all the tuples of **R and S by eliminating duplicate tuples**. The UNION operation between two relations R and S is denoted by:

Notation: RUS

Expression: $\Pi_{SNAME}(R) \cup \Pi_{SNAME}(S)$

SNO	SNAME	MARKS
1	Jack	92
2	Anil	56
3	Tomy	78

Output:

b) INTERSECTION(∩):

The Intersection operation defines a relation that **contains the common records of R and S**.The INTERSECTION operation between two relations R and S is denoted by:

Notation: R∩S

Expression: $\Pi_{SNAME}(R) \cap \Pi_{SNAME}(S)$

Output:

SNO	SNAME	MARKS
1	Jack	92

c) DIFFERENCE(—):

The Difference operation defines a relation that contains the records in relation R but not in S. Difference operation between two relations R and S is denoted by:

Notation: R—S

Expression: $\Pi_{SNAME}(R) - \Pi_{SNAME}(S)$

SNO	SNAME	MARKS
3	Tomy	78

Output:

d) Cartesian-Product:

The Cartesian-Product operation is denoted by a cross(**X**). It allows us to combine information from any two relations. The Cartesian-Product operation between two relations R and S is denoted by:

Notation: R X S

Expression: **Π FNAME (** STUDENT) x **Π AGE** (DETAIL)

STUDENT

SNO	FNAME	LNAME
1	Albert	Singh
2	Nora	Fatehi

DETAIL

ROLLNO	AGE
5	18
9	21

Output:-

FNAME	AGE
Albert	18
Albert	21
Nora	18
Nora	21

5. Join Operations:

The Join operation combines two relations to form a new relation. The tables should be joined based on a common column.

1) Natural Join(⋈):

Natural Join performs an equi join of the two relations R and S over all common attributes. Natural join eliminates duplicate attributes in the result set.

Syntax: R ⋈ S

Here R and S are two relations whose tuples are going to be joined based on a common column.

Example:

EMPLOYEE

EID	ENAME	DEPT
345	Satvika	Finance
346	Sanvika	Sales
347	Vijay	Finance
348	Sravan	Production

DEPARTMENT

DEPT	MANAGER
Finance	Sarvika
Sales	Sanvika
Production	Suman

Expression: Π EID, ENAME, DEPARTMENT.DEPT **(EMPLOYEE ⋈ DEPARTMENT)**

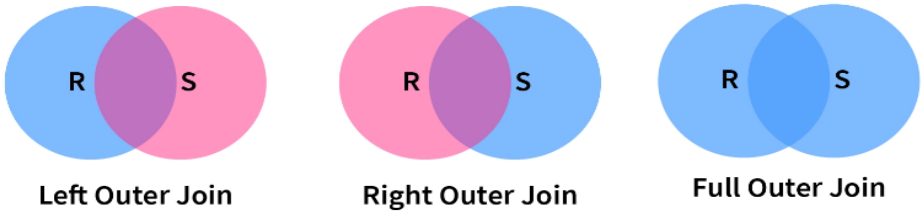
EMPLOYEE ⋈DEPARTMENT

EID	ENAME	DEPT	MANAGER

345	Satvika	Finance	Sarvika
346	Sanvika	Sales	Sanvika
347	Vijay	Finance	Sarvika
348	Sravan	Production	Suman

2) Outer Join:

By using Outer Join operation, we can read the rows that are having match in other table. In addition to that, unmatched rows from one table or both tables can also be read.



a) Left Outer Join:

This operation returns matched rows from both the tables and unmatched rows from left relation.

Syntax: $R \supset\!\bowtie S$

Ex:- EMPLOYEE ⋈EMPLOYEE.E_NO = DEPARTMENT.E_NO DEPARTMENT

b) Right Outer Join:

This operation returns matched rows from both the tables and unmatched rows from right relation.

Syntax: $R \bowtie\!\subset S$

Ex:- EMPLOYEE ⋈EMPLOYEE.E_NO = DEPARTMENT.E_NO DEPARTMENT

c) Full Outer Join:

This operation returns matched and unmatched rows from both the tables.

Syntax: $R \bowtie S$

Ex:- EMPLOYEE ⋈EMPLOYEE.E_NO = DEPARTMENT.E_NO DEPARTMENT

Division (÷)

The Division Operation is represented by "division"(÷ or /) operator and is used in queries that involve keywords "**every**", "**all**", etc.

Notation : R(X,Y)/S(Y)

Here,

➔ R is the first relation from which data is retrieved.

➔ S is the second relation that will help to retrieve the data.

➔ X and Y are the attributes/columns present in relation. We can have multiple attributes in relation, but keep in mind that attributes of S must be a proper subset of attributes of R.

➔ For each corresponding value of Y, the above notation will return us the value of X from tuple<X,Y> which exists **everywhere**.

Let's have two relations, **ENROLLED and COURSE.** ENROLLED consist of two attributes **STUDENT_ID and COURSE_ID.** It denotes the map of students who are enrolled in given courses.

COURSE contains the list of courses available.

See, here attributes/columns of COURSE relation are a proper subset of attributes /columns of ENROLLED relation. Hence Division operation can be used here.

ENROLLED

STUDENT_ID	COURSE_ID
Student_1	DBMS
Student_2	DBMS
Student_1	OS
Student_3	OS

COURSE

COURSE_ID
DBMS
OS

Now the query is to return the STUDENT_ID of students who are **Enrolled** in **every Course** .

Ex:- ENROLLED(STUDENT_ID, COURSE_ID)/COURSE(COURSE_ID)

This will return the following relation as output.

STUDENT_ID
Student_1

Prepared by

DVH. Venu Kumar

M.Sc.,M.Ed.,M.Tech(CSE)

Asst.Professor., Dept.of CSE., Geethanjali Institute of Science and Technology : Gangavaram : Nellore