UNIT – II

Relational Model: Codd's rule, Logical database design, Structure of relational databases, Relational Algebra, Fundamental relational algebra operations, Additional relational algebra operations, Extended relational algebra operations, Null values, Relational calculus, Tuple relational calculus, Domain relational calculus

COURSE OBJECTIVES:

- To get familiar with fundamental concepts of database management such as database design, database languages, and database-system implementation

COURSE OUTCOMES:

- ✓ Develop the knowledge of fundamental concepts of database management systems.

Relational Model

CODD'S RULE:

Rule 1: The Information Rule

All information, whether it is user information or metadata, that is stored in a database must be entered as a value in a cell of a table. It is said that everything within the database is organized in a table layout.

Rule 2: The Guaranteed Access Rule

Each data element is guaranteed to be accessible logically with a combination of the table name, primary key (row value), and attribute name (column value).

Rule 3: Systematic Treatment of NULL Values

Every Null value in a database must be given a systematic and uniform treatment.

Rule 4: Active Online Catalog Rule

The database catalog, which contains metadata about the database, must be stored and accessed using the same relational database management system.

Rule 5: The Comprehensive Data Sublanguage Rule

A crucial component of any efficient database system is its ability to offer an easily understandable data manipulation language (DML) that facilitates defining, querying, and modifying information within the database.

Rule 6: The View Updating Rule

All views that are theoretically updatable must also be updatable by the system.

Rule 7: High-level Insert, Update, and Delete
A successful database system must possess the feature of facilitating high-level insertions, updates, and deletions that can grant users the ability to conduct these operations with ease through a single query.

Rule 8: Physical Data Independence
Application programs and activities should remain unaffected when changes are made to the physical storage structures or methods.

Rule 9: Logical Data Independence
Application programs and activities should remain unaffected when changes are made to the logical structure of the data, such as adding or modifying tables.

Rule 10: Integrity Independence
Integrity constraints should be specified separately from application programs and stored in the catalog. They should be automatically enforced by the database system.

Rule 11: Distribution Independence
The distribution of data across multiple locations should be invisible to users, and the database system should handle the distribution transparently.

Rule 12: Non-Subversion Rule
If the interface of the system is providing access to low-level records, then the interface must not be able to damage the system and bypass security and integrity constraints.

LOGICAL DATABASE DESIGN:

A Logical Database is a special type of ABAP (Advance Business Application and Programming) that is used to retrieve data from various tables and the data is interrelated to each other. Also, a logical database provides a read-only view of Data.

Structure Of Logical Database:
A Logical database uses only a hierarchical structure of tables i.e. Data is organized in a Tree-like Structure and the data is stored as records that are connected to each other through edges (Links). Logical Database contains Open SQL statements which are used to read data from the database. The logical database reads the program, stores them in the program if required, and passes them line by line to the application program.

Features of Logical Database:
We can select only that type of Data that we need.
Data Authentication is done in order to maintain security.
Logical Database uses hierarchical Structure due to this data integrity is maintained.

Goal Of Logical Database:

The goal of Logical Database is to create well-structured tables that reflect the need of the user. The tables of the Logical database store data in a non-redundant manner and foreign keys will be used in tables so that relationships among tables and entities will be supported.

Tasks Of Logical Database:

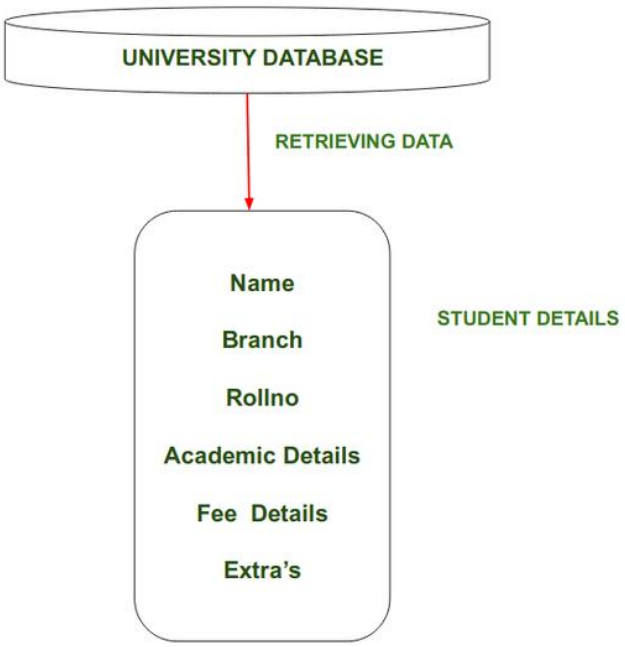
With the help of the Logical database, we will read the same data from multiple programs.
A logical database defines the same user interface for multiple programs.
Logical Database ensures the Authorization checks for the centralized sensitive database.
With the help of a Logical Database, Performance is improved. Like in Logical Database we will use joins instead of multiple SELECT statements, which will improve response time and this will increase the Performance of Logical Database.

Data View Of Logical Database:

Logical Database provides a particular view of Logical Database tables. A logical database is appropriately used when the structure of the Database is Large. It is convenient to use flow i.e

SELECT
READ
PROCESS
DISPLAY

Example:
Suppose in a University or College, a HOD wants to get information about a specific student. So for that, he firstly retrieves the data about its batch and Branch from a large amount of Data, and he will easily get information about the required Student but didn't alter the information about it.



Advantages Of Logical Database:

In a Logical database, we can select meaningful data from a large amount of data.
Logical Database consists of Central Authorization which checks for Database Accesses is Authenticated or not.
In this Coding, the part is less required to retrieve data from the database as compared to Other Databases.
Access performance of reading data from the hierarchical structure of the Database is good.
Easy to understand user interfaces.
Logical Database firstly check functions which further check that user input is complete, correct, and plausible.

Disadvantages Of Logical Database:

Logical Database takes more time when the required data is at the last because if that table which is required at the lowest level then firstly all upper-level tables should be read which takes more time and this slows down the performance.
In Logical Database ENDGET command doesn't exist due to this the code block associated with an event ends with the next event statement.

STRUCTURE OF RELATIONAL DATABASES:

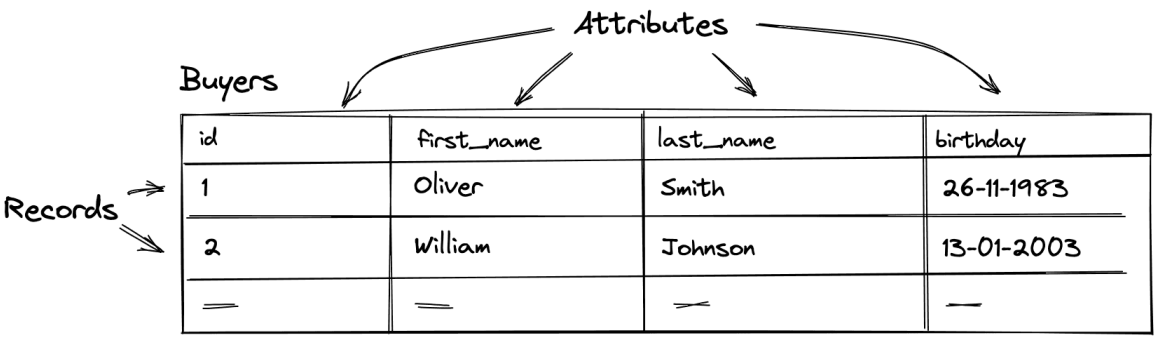
Data in relational structures is organized as a set of tables, called relationships, consisting of columns and rows. Each row of the table is a set of related values related to a single object or entity. Each row in a table can be labeled with a unique identifier called a primary key, and rows from multiple tables can be linked using foreign keys.

Features of relational databases

The data model in relational databases is defined in advance and is strictly typed
Data is stored in tables consisting of columns and rows
Only one value is allowed at the intersection of each column and row
Each column is named and has a specific type, followed by values from all rows in this column
The columns are arranged in a certain order, which is determined when creating the table
There may not be a single row in the table, but there must be at least one column
Queries to the database return the result in the form of tables.

Table Structure
In relational databases, information is stored in tables linked to each other. The tables themselves consist of:

- rows, which are called "records"
- columns, which are called "fields" or "attributes"



In each table, each column has a predetermined data type. For example, these types can be:

- VARCHAR (string data type)
- INTEGER (numeric data type)
- DATETIME (date and time data type)
- and others

Relation: A relation is usually represented as a table, organized into rows and columns. A relationship consists of multiple records. For example: student relation which contains tuples and attributes.

Tuple: The rows of a relation that contain the values corresponding to the attributes are called tuples. For example: in the Student relation there are 5 tuples.

The value of tuples contains (10112, Rama, 9874567891,islam ganj, F) etc.

Data Item: The smallest unit of data in the relation is the individual data item. It is stored at the intersection of rows and columns are also known as cells. For Example: 10112, "Rama" etc are data items in Student relation.

Domain: It contains a set of atomic values that an attribute can take. It could be accomplish explicitly by listing all possible values or specifying conditions that all values in that domain must be confirmed. For example: the domain of gender attributes is a set of data values "M" for male and "F" for female. No database software fully supports domains typically allowing the users to define very simple data types such as numbers, dates, characters etc.

Attribute: The smallest unit of data in relational model is an attribute. It contains the name of a column in a particular table. Each attribute Ai must have a domain, dom(Ai). For example: Stu_No, S_Name, PHONE_NO, ADDRESS, Gender are the attributes of a student relation. In relational databases a column

entry in any row is a single value that contains exactly one item only.

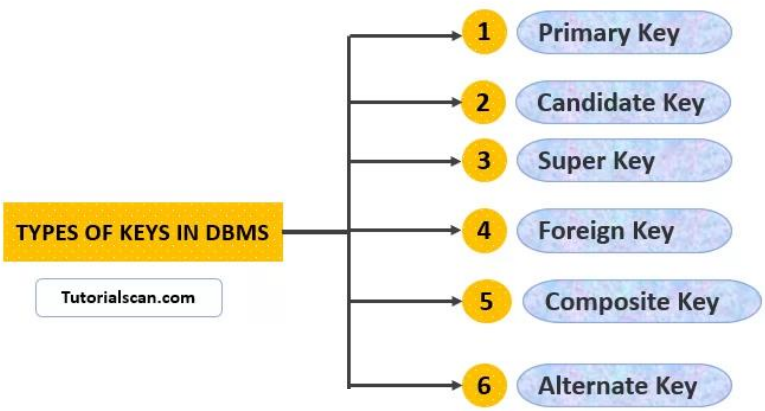
Cardinality: The total number of rows at a time in a relation is called the cardinality of that relation. For example: In a student relation, the total number of tuples in this relation is3 so the cardinality of a relation is 3. The cardinality of a relation changes with time as more and more tuples get added or deleted.

Degree: The degree of association is called the total number of attributes in a relationship. The relation with one attribute is called unary relation, with two attributes is known a binary relation and with three attributes is known as ternary relation. For example: in the Student relation, the total number of attributes is 5, so the degree of the relations is 5. The degree of a relation does not change with time as tuples get added or deleted.

Relational instance: In the relational database system, the relational instance is represented by a finite set of tuples. Relation instances do not have duplicate tuples.

Relational schema: A relational schema contains the name of the relation and name of all columns or attributes.

Relational key: In the relational key, each row has one or more attributes. It can identify the row in the relation uniquely.



There are mainly seven types of keys in DBMS such as:

1. Primary Key
2. Candidate Key
3. Super Key
4. Foreign Key
5. Composite Key
6. Alternate Key
7. Unique Key

A primary key is a column of a table or a set of columns that helps to recognised/identify every record present in that table uniquely. Furthermore, there can be only 1(one) primary Key in a table. Also, the primary Key can’t have identical values repeating for any row. Each value of the primary key has to be

different with no repetitions/duplicates.

Candidate keys in DBMS are those attributes that uniquely identify/recognize rows of a table. The Primary Key(PK) of a table is selected from one of the candidate keys. thus, candidate keys have identical properties to the primary keys explained above. as a result, there can be more than one candidate key in a table.

Super Key is the set of all the keys that facilitate identifying rows in a table uniquely. this implies that all those columns of a table that are capable of identifying the other columns of that table uniquely will all be considered super keys.
Thus, a super Key is the superset of a candidate key . The Primary Key(PK) of a table is picked from the super key set to be made the table's identity attribute.

Foreign Key is used to establish relationships between two tables. Furthermore, a foreign key will require each value in a column or set of columns to match the Primary Key(PK) of the referential table. Thus, foreign keys help to maintain data & referential integrity.

Composite Key in DBMS is a set of two or more attributes that facilitate or help identify each tuple in a table uniquely. Furthermore, the attributes in the set may not be unique or distinctive when considered separately. However, when taken all together, they will ensure/confirm uniqueness.

Alternate key:As explicit on top of, a table can have multiple choices for a primary key(PK). However, it can choose or select only one. So, all the keys that didn't become the primary Key are referred to as alternate keys.

A unique key is a column or set of columns that uniquely identify or determine every record in a table. Therefore, all values will have to be unique in this Key. thus, a unique Key differs from a primary key(PK) because it can have only 1(one) null value, whereas a primary Key can't have any null values.

RELATIONAL ALGEBRA:

The relational algebra is a *procedural* query language. It consists of a set of operations that take one or two relations as input and produce a new relation as their result. The fundamental operations in the relational algebra are *select*, *project*, *union*, *set difference*, *Cartesian product*, and *rename*. In addition to the fundamental operations, there are several other operations—namely, *set intersection*, *natural join*, and *assignment*. We shall define these operations in terms of the fundamental operations.

Fundamental Operations

The select, project, and rename operations are called *unary* operations, because they operate on one relation. The other three operations operate on pairs of relations and are, therefore, called *binary* operations.
Consider the Two relations STUDENT, QUARTERLY. The STUDENT relation is used to describe the complete personal information about student, his roll no, name, date of birth, 2nd language. Another relation QUARTERLY used to describe the students marks in 3 subjects with roll no's.

STUDENT

Roll-No	Name	Date-of-Birth	Second-Language
1	Sunny	01-07-70	Hindi
2	Rashni	15-08-72	Sanskrit
3	Anthra	29-01-71	Hindi
4	Nasreen	31-12-70	Telugu

QUARTERLY

Roll-No	Maths	Physics	Computers
1	72	85	90
2	65	74	68
3	97	94	96
4	87	93	72

The following are queries based on relational algebra to obtain required informationfromstored relationaldatabase.

1) **The SELECT Operation:** The Select operation selects tuples that satisfy a given predicate. Alower case Greek letter sigma (σ) is used to denote Select operation. Predicate appears as subscript tor.

The argument relation is given in parentheses. The General form of selection operation is:

$\sigma_{predicate}$ (relation)

All comparisons =, #, <, >, <=, >= were allowed in the select operation predicate. Furthermore, several predicates may be combined into a large predicate using the connectives *and* (∧) and *or* (V).

Ex: 1) List out the complete information about all students whose 2nd language is Hindi

$\sigma_{2nd\text{-}language = Hindi}$

(STUDENT)Result of the

above Query is:

Roll-No	Name	Date-of-Birth	Second-Language
1	Sunny	01-07-70	Hindi
3	Anthra	29-01-71	Hindi

Ex: 2) Display all students with Roll no with their marks who secured more than 90 in allthe three subjects

σ ((Maths > 90) ∧(Physics > 90) ∧(Computers > 90))

(QUARTERLY) Result of the above Query is:

Roll-No	Maths	Physics	Computers
3	97	94	96

2) **The PROJECT Operation:** The projection of a relation is defined as a projection of all its tuples over some set of attributes. i.e., it yields a "vertical subset" of the relation. The projection operation is used to either reduce the number of attributes in the resultant relation or to reorder attributes. Projection is denoted by Greek letter pi

(Π). We listthese attributes that we wish to appear in the result as a subscript to. The argument

relation follows inparenthesis.

General form of projection operation is

$$\Pi_{\text{List-of-attributes(Predicate)}}(\text{relation})$$

Ex: 1) List out all Roll Nos. and their ComputerMarks

$$\Pi_{\text{Roll No, Computers}}(\text{QUARTERLY})$$

Result of the above Query is:

Roll-No	Computers
1	90
2	68
3	96
4	72

2) Display all the student names with their date of birth whose 2nd language isHindi.

$$\Pi_{\text{Name, Date-of-birth}}(\sigma_{\text{2nd-language = Hindi}})(\text{STUDENT})$$ Result of the above query is:

Name	Date-of-Birth
Sunny	01-07-70
Anthra	29-01-71

3) What is the Date of Birth ofRashni?

$$\Pi_{\text{Date-of-Birth}}(\sigma_{\text{Name = Rashni}})$$

(STUDENT)Result of the above query is:

Date-of-Birth
15-08-72

4) Find all Roll Nos who secured more than 90 marks inComputers

$$\Pi_{\text{Roll-No}}(\sigma_{\text{computers > 90}})$$

(QUARTERLY)Result of the above query is:

Roll-No
1
3

3) The RENAME Operation: Unlike relations in the DB, the results of relational-algebra expressions do not have a name that we can use to refer to them. It is useful to be able to give them names; the **rename** operator, denoted by the lowercase Greek letter rho (ρ), lets us do this. Given a relational- algebra expression *E*, the following expression returnsthe result of expression *E* under the name*x*.

$$\rho_x(E)$$

Ex: $\rho_{\text{teacher}}(instructor)$

A relation *r* by itself is considered a (trivial) relational-algebra expression. Thus, we can also apply the rename operation to a relation *r* to get the same relation under a new name.

A Second form of the rename operation is as follows: Assume that a relational algebra expression *E* has arity *n*. Then, the following expression returns the result of expression*E* under the name *x*, and with the attributes renamed to *A*1, *A*2, . . . , *An*.

$$\rho_{x(A1,A2,\ldots,An)}(E)$$

Ex: $\rho_{\text{teacher (id, name, sal)}}(instructor)$

4) **CARTESIAN PRODUCT Operation:** This operation allows us to combine information from several relations. Thus operation is denoted by a cross(X). Thus operation is a binary. Suppose r1 and r2 are two relations, Cartesian product of these tworelations can be written are r1 Xr2.

In other words, Cartesian product of two relations is the concatenation of tuples belonging to the two relations. A new resultant relation scheme is created consisting of all possible combinations of tuples.

If there are m tuples in relation r1, and n tuples in relation r2, then there is m x n ways of choosing a pair of tuples. One tuple from each relation is chosen, so there are n1 x n2 tuplesin r.

Ex: (i) Find student names and their Computer marks.

To list out the Name, Computer Marks, we have to refer both the relations, STUDENT & QUARTERLY. Student name is an attribute from STUDENT relation and Computers is an attribute from QUARTERLY relation. Referring 2 relations is denoted by"X"

STUDENT X QUARTERLY

Roll-No	Name	Date-of-Birth	Second-Language	Roll-No	Maths	Physics	Computers
1	Sunny	01-07-70	Hindi	1	72	85	90
1	Sunny	01-07-70	Hindi	2	65	74	68
1	Sunny	01-07-70	Hindi	3	97	94	96
1	Sunny	01-07-70	Hindi	4	87	93	72
2	Rashni	15-08-72	Sanskrit	1	72	85	90
2	Rashni	15-08-72	Sanskrit	2	65	74	68
2	Rashni	15-08-72	Sanskrit	3	97	94	96
2	Rashni	15-08-72	Sanskrit	4	87	93	72
3	Anthra	29-01-71	Hindi	1	72	85	90
3	Anthra	29-01-71	Hindi	2	65	74	68
3	Anthra	29-01-71	Hindi	3	97	94	96
3	Anthra	29-01-71	Hindi	4	87	93	72
4	Nasreen	31-12-70	Telugu	1	72	85	90

4	Nasreen	31-12-70	Telugu	2	65	74	68
4	Nasreen	31-12-70	Telugu	3	97	94	96
4	Nasreen	31-12-70	Telugu	4	87	93	72

Information is retrieved from the above relation STUDENT X QUARTERLY. By selecting the common attribute in the same relation i.e., in given two relations, Roll No is the common attribute.

STUDENT.ROLLNO = QUARTERLY.ROLLNO

Π Name, Computers (σ STUDENT.ROLL-NO = QUARTERLY.ROLL-NO) (STUDENT X QUARTERLY)

Name	Computers
Sunny	90
Rashni	68
Anthra	96
Nasreen	72

(ii) Find the Student Roll No, Date of Birth, 2nd Language, Maths, Physics and Computer Marks.

σ STUDENT.ROLL NO = QUARTERLY.ROLL NO. (STUDENT X QUARTERLY)

Result of the query is:

Roll-No	Name	Date-of-Birth	Second-Language	Roll-No	Maths	Physics	Computers
1	Sunny	01-07-70	Hindi	1	72	85	90
2	Rashni	15-08-72	Sanskrit	2	65	74	68
3	Anthra	29-01-71	Hindi	3	97	94	96
4	Nasreen	31-12-70	Telugu	4	87	93	72

5) **UNIONOperation:**Theunionoftworelationsrandsisdenotedbyr∪s.Theoutputrelation Z=r ∪s has tuples drawn from r and s. The result relation Z contains tuples that are in eitherr or s or in both of them. The duplicate tuples are eliminated.

For a union operation $r \cup s$ to be valid, we require that two conditions hold:

- The relations *r* and *s* must be of the same arity. i.e., they must have the same numberofattributes.
- The domains of the *i*th attribute of *r* and the *i*th attribute of *s* must be the same, for all*i*.

Note that *r* and *s* can be either database relations or temporary relations that are the result ofrelational algebra expressions.

As an example, consider the relations CULTURAL (name, class) and SPORTS (name, class). These two relations represent information about all cultural competition winners &

sports winnersseparately.

CULTURAL

Name	Class
Kavya	MPC III
Lahari	MSC II
Bhanu	MCA II
Zeba	MPC III
Nisha	MBA II
Hima	MPC III

SPORTS

Name	Class
Deepta	MCA III
Lahari	MSC II
Hima	MPC III
Archana	MBA II
Sheela	MPC III

Ex: Find all the student Names of MPC III who won cultural competition or sports competition or both competitions.

$\Pi_{\text{Name}}(\sigma_{\text{Class = 'MPC III'}})$ (CULTURAL) $\cup \Pi_{\text{Name}}(\sigma_{\text{Class = MPC III}})$ (SPORTS)

NAME
Kavya
Zeba
Hima

NAME
Hima
Sheela

NAME
Kavya
Zeba
Hima
Sheela

6) **SET-DIFFERENCE Operation:** The difference between two relations r and s is r - s.The result relation contains the set of tuples belonging to r and not ins.

Ex: Find Student Names of MPC III who won cultural prizes but not sports.

$\Pi_{\text{Name}}(\sigma_{\text{Class = 'MPC III'}})$ (CULTURAL) - $\Pi_{\text{Name}}(\sigma_{\text{Class = MPC III}})$ (SPORTS)

NAME
Kavya
Zeba
Hima

NAME
Hima
Sheela

NAME
Kavya
Zeba

Formal Definition of the Relational Algebra: The fundamental operations of relational algebra allow us to give a complete definition of an expression in the relational algebra. A basic expression in the relational algebra consists of either one of the following:

- A relation in thedatabase
- A constantrelation

A constant relation is written by listing its tuples within { }, for example

{(22222, Einstein, Physics, 95000), (76543, Singh, Finance, 80000)}.

A general expression in the relational algebra is constructed out of smaller sub expressions. Let *E*1 and

*E*2 be relational-algebra expressions. Then, the following are all relational-algebra expressions:

- ✓ *E*1 U*E*2

- ✓ $E1 - E2$
- ✓ $E1 \times E2$
- ✓ $\sigma_P(E1)$, where P is a predicate on attributes in $E1$
- ✓ $\Pi_S(E1)$, where S is a list consisting of some of the attributes in $E1$
- ✓ $\rho_x(E1)$, where x is the new name for the result of $E1$

Additional Relational-Algebra Operations

We define additional operations that do not add any power to the algebra, but simplify commonqueries.

1) **SET-INTERSECTION Operation:** The intersection of two relations r and s is denoted by r ∩s. The output relation contains the set of all tuples belonging to both r ands.

Ex: List out all the names belonging to MPC III who won both the cultural & sports competition.

$$\Pi_{\text{Name}}(\sigma_{\text{Class = 'MPC III'}})(\text{CULTURAL}) \cap \Pi_{\text{Name}}(\sigma_{\text{Class = MPC III}})(\text{SPORTS})$$

NAME
Kavya
Zeba
Hima

NAME
Hima
Sheela

NAME
Hima

2) **NATURAL JOIN Operation:** Natural join is a binary operation that allows us to combine certain selections and a Cartesian product into one operation. It is denoted by the "join" symbol. Natural join operation forms a Cartesian product of its two arguments, performs a selection forcing equalities on those attributes that appear in both relation schemas, and finally removes duplicatecolumns.

Ex:STUDENT ⋈ QUARTERLY

This operation performs a Cartesian product (X) of two relations, performs selection equality on those attributes that appear in both the relation schemas and finally removes duplicate columns. In STUDENT X QUARTERLY relations ROLLNO is common in both relations.

i.e.STUDENT ⋈ QUARTERLY becomes:

Roll-No	Name	Date-of-Birth	Second-Language	Roll-No	Maths	Physics	Computers
1	Sunny	01-07-70	Hindi	1	72	85	90
2	Rashni	15-08-72	Sanskrit	2	65	74	68
3	Anthra	29-01-71	Hindi	3	97	94	96

4	Nasreen	31-12-70	Telugu	4	87	93	72

Query can be written as

$\Pi_{\text{Roll No, Name, Date of Birth, 2nd language, Maths, Physics, Computers}}$ (STUDENT ⋈ QUARTERLY) Now, to find student names and Computer marks, thequery willbe:

$\Pi_{\text{Name,Computers}}$(STUDENT ⋈ QUARTERLY)

Name	Computers
Sunny	90
Rashni	68
Anthra	96
Nasreen	72

3) **The ASSIGNMENT Operation:** It is convenient at times to write a relational-algebra expression by assigning parts of it to temporary relation variables. The **assignment** operation, denoted by ←, works like assignment in a programming language. To illustrate this operation, consider the definition of the natural-join operation. We could write *r* ∪ *s*as:

$$result \leftarrow r \cup s$$

The evaluation of an assignment does not result in any relation being displayed to the user. Rather, the result of the expression to the right of the ← is assigned to the relation variable on the left of the ←.

This relation variable may be used in subsequent expressions.

For relational algebra queries, assignment must always be made to a temporary relation variable. Note that the assignment operation does not provide any additional power to the algebra. It is, however, a convenient way to express complex queries.

Extended Relational-Algebra Operations

The relational algebra operations that provide the ability to write queries that cannot be expressed using the basic relational-algebra operations are called **extended relational-algebra** operations.

1) **GENERALIZED PROJECTION:** The first operation is the **generalized-projection** operation, which extends the projection operation by allowing operations such as arithmetic and string functions to be used in the projection list. The generalized-projection operation has theform:

$$\Pi_{F1,F2,\ldots,Fn}(E)$$

where E is any relational-algebra expression, and each of $F1, F2, \ldots, Fn$ is an arithmetic expression involving constants and attributes in the schema of E. As a base case, the expression may be simply an attribute or a constant. In general, an expression can use arithmetic operations such as +, −, ∗, and ÷ on

numeric valued attributes, numeric constants, and on expressions that generate a numeric result. Generalized projection also permits operations on other data types, such as concatenation of strings.

For example, the expression:

$\Pi_{ID,\ name,\ deptname,\ salary \div 12}(instructor)$ gives the *ID*, *name*, *deptname*, and the monthly salary of each instructor.

2) **AGGREGATION:** The second extended relational-algebra operation is the aggregate operation *G*, which permits the use of aggregate functions such as min or average, on sets ofvalues.

Aggregate Functions: Aggregate Functions take a collection of values and return a singlevalue as a result. Aggregate operation in relational algebra is expressed as:

$G_1, G_2, \ldots, Gn\ G_{F1(A1),\ F2(A2),\ldots,\ Fn(An)}(E)$

Where, E is any relational algebra expression Each $F_i$is an aggregate function

Each $A_i$is an attribute name

$G_1, G_2 \ldots, G_n$ is a list of attributes on which to group (can beempty) The Aggregate Functions include:

1. **sum:** It is used to find the sum of values of an attribute in a relation.

   Ex: $G_{\mathbf{sum}(salary)}(instructor)$

2. **avg**: It is used to find the average value of an attribute in a relation.

   Ex: $G_{\mathbf{avg}(salary)}(instructor)$

3. **count**: It is used to find the number of values in an attribute in a relation.

   Ex: $G_{\mathbf{count}(salary)}(instructor)$

There are cases where we must eliminate multiple occurrences of a value before computingan aggregate function. If we do want to eliminate duplicates, we use the same function namesas before, with the addition of the key word "**distinct**" appended to the end of the function name (for ex. **count-distinct**). Now, the above example can also be written as:

$G_{\mathbf{count\text{-}distnict}(salary)}(instructor)$

4. **min:** It is used to find the minimum value in an attributein arelation.

   Ex: $G_{\mathbf{min}(salary)}(instructor)$

5. **max:** It is used to find the maximum value in an attributein arelation.

   Ex: $G_{\mathbf{max}(salary)}(instructor)$

❖ Result of aggregation does not have aname.

- Can use rename operation to give it a name.
- For convenience, we permit renaming as part of aggregate operation

NULL VALUES: It is possible for tuples to have a null value, denoted by *null*, for some oftheir attributes. *null* signifies an unknown value or that a value does not exist.

✓ The result of any arithmetic expression involving *null* is*null*.

- ✓ Aggregate functions simply ignore null values
  - Is an arbitrary decision. Could have returned null as result instead.
  - We follow the semantics of SQL in its handling of null values.
- ✓ For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumedto be the same. As an alternative, assume each null is different from each other like inSQL.

Arithmetic Operations and Comparisons with null values return the special truth value *unknown or null.*

For logical operators with an input as null:

OR: (*unknown* **or** *true*) = *true*,
(*unknown* **or** *false*) = *unknown*,
(*unknown* **or** *unknown) = unknown*

AND: *(true* **and** *unknown) = unknown*,
(false **and** *unknown) = false*,
(unknown **and** *unknown) = unknown*

NOT: *(***not** *unknown) = unknown*

Relaional calculus:

Relational calculus is a non-procedural query language, and instead of algebra, it uses mathematical predicate calculus. The relational calculus is not the same as that of differential and integral calculus in mathematics but takes its name from a branch of symbolic logic termed as predicate calculus. When applied to databases, it is found in two forms. These are

1. Tuple relational calculus which was originally proposed by Codd in the year 1972
2. Domain relational calculus which was proposed by Lacroix and Pirotte in the year 1977

In first-order logic or predicate calculus, a predicate is a truth-valued function with arguments. When we replace with values for the arguments, the function yields an expression, called a proposition, which will be either true or false.

Tuple Relational Calculus (TRC) is a non-procedural query language used in relational database management systems (RDBMS) to retrieve data from tables. TRC is based on the concept of tuples, which are ordered sets of attribute values that represent a single row or record in a database table.
TRC is a declarative language, meaning that it specifies what data is required from the database, rather than how to retrieve it. TRC queries are expressed as logical formulas that describe the desired tuples.
Syntax: The basic syntax of TRC is as follows:
{ t | P(t) }
where t is a **tuple variable** and P(t) is a **logical formula** that describes the conditions that the tuples in the

result must satisfy. The **curly braces {}** are used to indicate that the expression is a set of tuples.
For example, let's say we have a table called "Employees" with the following attributes:

Employee ID
Name
Salary
Department ID

To retrieve the names of all employees who earn more than $50,000 per year, we can use the following TRC query:
{ t | Employees(t) ∧ t.Salary > 50000 }
In this query, the "Employees(t)" expression specifies that the tuple variable t represents a row in the "Employees" table. The "∧" symbol is the logical AND operator, which is used to combine the condition "t.Salary > 50000" with the table selection.
The result of this query will be a set of tuples, where each tuple contains the Name attribute of an employee who earns more than $50,000 per year.
TRC can also be used to perform more complex queries, such as joins and nested queries, by using additional logical operators and expressions.
While TRC is a powerful query language, it can be more difficult to write and understand than other SQL-based query languages, such as Structured Query Language (SQL). However, it is useful in certain applications, such as in the formal verification of database schemas and in academic research.
Tuple Relational Calculus is a **non-procedural query language,** unlike relational algebra. Tuple Calculus provides only the description of the query but it does not provide the methods to solve it.

Domain Relational Calculus

Domain Relational Calculus is a non-procedural query language equivalent in power to Tuple Relational Calculus. Domain Relational Calculus provides only the description of the query but it does not provide the methods to solve it. In Domain Relational Calculus, a query is expressed as,
{ < x1, x2, x3, ..., xn > | P (x1, x2, x3, ..., xn) }
where, < x1, x2, x3, …, xn> represents resulting domains variables and P (x1, x2, x3, …, xn) represents the condition or formula equivalent to the Predicate calculus.
Predicate Calculus Formula:

1. Set of all comparison operators
2. Set of connectives like and, or, not
3. Set of quantifiers

Example:
Table-1: Customer

Customer name	Street	City
Debomit	Kadamtala	Alipurduar
Sayantan	Udaypur	Balurghat
Soumya	Nutanchati	Bankura
Ritu	Juhu	Mumbai

Table-2: Loan

Loan number	Branch name	Amount
L01	Main	200
L03	Main	150
L10	Sub	90
L08	Main	60

Table-3: Borrower

Customer name	Loan number
Ritu	L01
Debomit	L08
Soumya	L03

Query-1: Find the loan number, branch, amount of loans of greater than or equal to 100 amount.
$\{\prec l, b, a\succ \mid \prec l, b, a\succ \in loan \wedge (a \geq 100)\}$
Resulting relation:

Loan number	Branch name	Amount
L01	Main	200
L03	Main	150

Query-2: Find the loan number for each loan of an amount greater or equal to 150.
$\{\prec l\succ \mid \exists\ b, a\ (\prec l, b, a\succ \in loan \wedge (a \geq 150))\}$

Resulting relation:

Loan number
L01
L03

Query-3: Find the names of all customers having a loan at the "Main" branch and find the loan amount .
{≺c, a≻ | ∃ l (≺c, l≻ ∈ borrower ∧ ∃ b (≺l, b, a≻ ∈ loan ∧ (b = "Main")))}
Resulting relation:

Customer Name	Amount
Ritu	200
Debomit	60
Soumya	150