UNIT - II: THE RELATIONAL MODEL

Introduction to The Relational Model –Integrity Constraints over Relations – Querying Relational Data–Logical Database Design: ER to Relational. Relational Algebra And Calculus – Preliminaries – Relational Algebra – Relational Calculus – Expressive power of Algebra and Calculus.

Introduction to the Relational Model

The relational model, proposed by Edgar F. Codd in 1970, is a foundational approach to managing data that organizes it into one or more tables, also known as relations. This model simplifies data storage, retrieval, and management by representing data and the relationships between data in a logical, tabular format.

The main components of the relational model are:

- **Tables (or Relations):** The central building block that organizes data into rows and columns. Each table contains a set of related data and has a unique name.
- **Rows (or Tuples):** Each row in a table represents a single, real-world entity or a record. For example, in a STUDENT table, each row would represent a single student.
- **Columns (or Attributes):** These represent the properties or characteristics of the entity described by the table. Each column has a name and a data type, such as Name (string) or Age (integer).
- **Keys:** Special attributes that establish links between tables and ensure data integrity.
    - **Primary Key:** An attribute or set of attributes that uniquely identifies each row in a table. It cannot contain NULL values and must be unique.
    - **Foreign Key:** An attribute in one table that refers to the primary key of another table. It is used to create and enforce a relationship between the two tables.
- **Relational Database Schema:** A collection of relation schemas for a database, including the table names, attributes, and integrity constraints.

Examples of relational models

Consider a simplified database for a library to see how the relational model works in practice.

Example 1: One-to-many relationship

In a one-to-many relationship, one row in a table can be linked to many rows in another table. In a library, one author can have written many books.

Table 1: Authors

This table has a primary key AuthorID.

AuthorID (PK)	FirstName	LastName
A101	Stephen	King
A102	J.K.	Rowling
A103	George	Orwell

Table 2: Books

This table has its own primary key, BookID, and a foreign key, AuthorID. The AuthorID in this table connects a book back to its author in the Authors table.

BookID (PK)	Title	AuthorID (FK)
B001	*Java programming*	A101
B002	*IT*	A101
B003	*Python programming*	A102
B004	*Machine Learning*	A103

Example 2: Many-to-many relationship

A many-to-many relationship exists when multiple records in one table are associated with multiple records in another table. In a library, many students can enroll in many courses. This requires a third, "junction" table to resolve the relationship.

Table 1: Students
This table lists each student with a unique primary key StudentID.

StudentID (PK)	StudentName
101	Rajesh
102	Gowtham
103	Anil

Table 2: Courses
This table lists each course with a unique primary key CourseID.

CourseID (PK)	CourseName
CS101	Introduction to Programming
MA101	Calculus I

Table 3: Enrollment (Junction Table)

This table uses the primary keys from both the Students and Courses tables as foreign keys to track which students are in which courses.

StudentID (FK)	CourseID (FK)
101	CS101
101	MA101
102	CS101
103	MA101

Integrity Constraints over Relations

Integrity constraints are a set of rules used in a database management system (DBMS) to maintain the quality, consistency, and accuracy of data within relational tables. These rules prevent invalid data from being entered, ensuring that the database remains in a correct and valid state.

The main types of integrity constraints include:

- Domain constraints
- Entity integrity constraints
- Referential integrity constraints
- Key constraints

Domain constraints

Domain constraints define the set of valid values for each attribute (column) in a relation (table). These constraints enforce data types, formats, ranges, and other restrictions to prevent invalid data entry.

Example:

Consider a table named Students:

- age: A domain constraint can specify that the age attribute must be an integer and fall within a valid range, such as 15 to 80. Entering a value outside this range (e.g., 10 or 150) or a non-integer value would violate the constraint.
- semester: A CHECK constraint can be used to limit the semester attribute to specific values like 5th, 6th, 7th, and 8th. An entry of 10th would be rejected.
- phone_number: Can be restricted to a specific format, such as a 10-digit number.

Entity integrity constraints

The entity integrity constraint states that the primary key of a relation cannot have a null value. A primary key is used to uniquely identify each row (tuple), and a null value would make it impossible to uniquely identify that row.

Example:
In a table Employees, EmpID is designated as the primary key.

- **Valid entry:** (EmpID: 101, Name: 'Jackson')
- **Invalid entry:** (EmpID: NULL, Name: 'James') would be rejected because the EmpID, as a primary key, cannot be null.

Referential integrity constraints

Referential integrity is a rule involving a foreign key that ensures the consistency of relationships between two tables. A foreign key in one table must either match the primary key of a related table or be null. This prevents "orphan" records that reference non-existent data.

Example:
Consider two tables, Departments and Employees:

- **Departments table (Parent table):** DeptID is the primary key.

DeptID (PK)	DeptName
1	Sales
2	Marketing

- **Employees table (Child table):** DeptID is a foreign key referencing Departments(DeptID).

EmpID (PK)	Name	DeptID (FK)
101	John David	1
102	Arun	2

Valid and invalid operations:

- **Valid insertion:** Inserting an employee with DeptID = 1 is valid because a matching DeptID exists in the Departments table.
- **Invalid insertion:** Attempting to insert an employee with DeptID = 4 would be rejected because no department with DeptID = 4 exists.
- **Invalid deletion:** Deleting the Sales department (with DeptID = 1) would be blocked if employees still exist in that department, as it would violate referential integrity and leave orphan records.

Key constraints

Key constraints ensure that certain attributes or sets of attributes can uniquely identify a tuple within a relation. Different types of key constraints serve specific purposes:

- **Primary Key:** A primary key is a candidate key selected by the database designer to uniquely identify each tuple. It must be both unique and not null.
  - **Example:** In a Students table, StudentID is a primary key. No two students can have the same ID, and StudentID cannot be null.
- **Unique Key:** A unique key also ensures that all values in a column or set of columns are unique. Unlike a primary key, a table can have multiple unique keys, and they can contain a single null value.
  - **Example:** In a Users table, the email column can be set as a unique key. This prevents two users from registering with the same email address.
- **Foreign Key:** As discussed in referential integrity, a foreign key is an attribute that references the primary key of another table, enforcing relationships between them.

Querying Relational data

Querying relational data is the process of retrieving information from a relational database using a query language, most commonly Structured Query Language (SQL). The relational model organizes data into tables, with each row representing a record and each column representing an attribute. A key concept is that relationships can be established between these tables using a shared attribute.

Key concepts

- **Table (Relation):** A collection of rows and columns that represents a specific entity, such as Employees or Products.
- **Column (Attribute):** A property or characteristic of an entity, such as EmployeeName or ProductID.

- **Row (Tuple):** A single record in a table that represents a unique instance of an entity, containing a value for each column.
- **Primary Key (PK):** A column or set of columns with values that uniquely identify each row in a table. It cannot contain NULL values.
- **Foreign Key (FK):** A column in one table that refers to the primary key of another table. It establishes a link between the two tables, defining their relationship.

Example database

For this example, consider a simple database for a company with two tables: Employees and Departments.

Employees table

EmployeeID (PK)	FirstName	LastName	DepartmentID (FK)
101	Alice	Smith	10
102	Bob	Johnson	20
103	Charlie	Davis	10
104	Diana	Evans	30

Departments table

DepartmentID (PK)	DepartmentName	Location
10	Sales	New York
20	Marketing	London
30	IT	New York

Simple querying examples

1. Selecting all employees

To retrieve every record and all columns from the Employees table, you use SELECT *.
Concept: The SELECT statement tells the DBMS to retrieve data, and the * is a wildcard for "all columns".
Query:

```
SELECT * FROM Employees;
```

Result: The entire Employees table is returned.

2. Selecting specific columns

To retrieve only the first and last names of employees, you specify the column names.

Concept: By listing columns explicitly, you limit the data retrieved, which is more efficient than selecting all columns.

Query:

```
Sql
        SELECT FirstName, LastName FROM Employees;
```

Result:

FirstName	LastName
Alice	Smith
Bob	Johnson
Charlie	Davis
Diana	Evans

3. Filtering data

To find all employees in a specific department, you use the WHERE clause.
Concept: The WHERE clause applies a condition to filter the rows retrieved from the table.
Query:

```
Sql SELECT FirstName, LastName FROM Employees WHERE DepartmentID = 10;
```

Result:

FirstName	LastName
Alice	Smith
Charlie	Davis

4. Joining tables

To see the department names alongside the employees, you join
the Employees and Departments tables on their common column, DepartmentID.

Concept: The JOIN operation combines rows from two or more tables based on a related column, using the foreign key (DepartmentID in Employees) to match with the primary key (DepartmentID in Departments).

Query:

```
Sql
    SELECT  E.FirstName,  E.LastName,  D.DepartmentName FROM Employees AS E
JOIN Departments AS D ON E.DepartmentID = D.DepartmentID;
```

Result:

FirstName	LastName	DepartmentName
Alice	Smith	Sales
Bob	Johnson	Marketing
Charlie	Davis	Sales
Diana	Evans	IT

Logical Database Design: ER to Relational

Converting an Entity-Relationship (ER) model to a relational model involves transforming the conceptual design of a database into a logical structure of tables. This is a crucial step in database development, as the relational model can be directly implemented in a Relational Database Management System (RDBMS) like MySQL or Oracle.

Mapping rules with examples

1. Regular entity set

A regular entity is converted into a table. The attributes of the entity become the columns of the table, and the key attribute becomes the primary key.

ER component: Student entity with attributes *S_ID* (key) and *S_Name*.
Relational schema: Student table.

- S_ID (Primary Key)
- S_Name

2. Weak entity set

A weak entity is converted into a table that includes all its own attributes. It also takes the primary key of its identifying (strong) entity as a foreign key. The primary key of the weak entity's table is a combination of this foreign key and its own partial key.

ER component: Dependent (weak entity) with partial key *Dep_Name* and Employee (strong entity) with key *Emp_ID*.

Relational schema: Dependent table.

- Emp_ID (Foreign Key)
- Dep_Name
- **Primary Key:** (Emp_ID, Dep_Name)

3. 1:1 relationship

For a one-to-one relationship between entities A and B, you add the primary key of one entity's table as a foreign key in the other's table. If one side has total participation, the foreign key is placed in that entity's table.

ER component: A 1:1 relationship where Employee manages a Department, and Department has total participation (every department has a manager).

Relational schema: Department table.

- Dept_ID (PK)
- Dept_Name
- Mgr_ID (FK to Employee.Emp_ID)

4. 1:N relationship

For a one-to-many relationship, the primary key of the "one" side's table is added as a foreign key to the "many" side's table.

ER component: A 1:N relationship where a Department has many Employees.
Relational schema: Employee table.

- Emp_ID (PK)
- Emp_Name
- Dept_ID (FK to Department.Dept_ID)

5. M:N relationship

A many-to-many relationship is converted into a new, separate table. This new table's primary key is a combination of the foreign keys of both participating entities. Any attributes of the relationship itself are also included.

ER component: An M:N relationship where an Employee can work on multiple Projects, and a Project can have multiple Employees. The relationship has the attribute *Hours*.

Relational schema: Works_On table.

- Emp_ID (FK to Employee.Emp_ID)
- Proj_ID (FK to Project.Proj_ID)
- Hours
- **Primary Key:** (Emp_ID, Proj_ID)

6. Multivalued attributes

A multivalued attribute is converted into a new table. The new table contains the primary key of the original entity as a foreign key and the multivalued attribute itself. The primary key of the new table is the combination of both columns.

ER component: A Professor entity with a multivalued attribute Phone_*Number*.

Relational schemas:

Professor table.

- P_ID (PK)
- P_Name

Professor_Phone table.

- P_ID (FK)
- Phone_Number
- **Primary Key:** (P_ID, Phone_Number)

Relational Algebra and Calculus

Relational algebra is a procedural language that defines how to perform a query by specifying a sequence of operations, while relational calculus is a non-procedural language that specifies what data to retrieve without detailing the method. Both are theoretical foundations for SQL, but relational algebra uses operators like selection (σ), projection (π), and join (\Join), and calculus uses mathematical expressions with variables and predicates.

The basic operations included in relational algebra are:

- 1. Select (σ) : Filters rows (tuples) from a relation based on a specified condition.

  2. Project (Π): Selects specific columns (attributes) from a relation.

  3. Union (U) : Combines all tuples from two relations. The relations must have the same number of

attributes and compatible domains.

4. Set Difference (-) : Returns the tuples that are in the first relation but not in the second. The relations must be union-compatible.

5. Cartesian product (X) : Combines every tuple of the first relation with every tuple of the second relation.

6. Rename (ρ) : Renames the output relation or its attributes.

Advantages of Relational Algebra

- Simplicity and Precision: On the same note, Relational Algebra offers a simple and efficient methodology for how questions can be framed especially for a layman who has a crystal-clear understanding of the entire method.
- Optimization Potential: It makes optimization possible due to the procedural aspect, and shows how one can find the best way to execute the statement.
- Foundational for SQL: Most of the SQL commands can be traced to the Relational Algebra operations.

Disadvantages of Relational Algebra

- **Complex for Complex Queries:** Relational algebra is more or less effective when the query is highly complex because with increasing complexity, Relational Algebra analysis in full becomes a cumbersome exercise.
- **Less Abstract:** It is also procedural and thus it forces users to detail out the procedure of data access, which may not be easily understood by end users who are not very conversant with databases.

Relational Calculus

Relational Calculus is the formal query language. It is also known as **Declarative language**. It is the order is not specified in which the operation has to be performed. Relational Calculus means what result we have to obtain.

Relational Calculus has two variations:

1. Tuple Relational Calculus (TRC)
2. Domain Relational Calculus (DRC)

Relational Calculus is denoted as:

```
{ t | P(t) }

Where,
t: the set of tuples
p: is the condition which is true for the given set of tuples.
```

Advantages of Relational Calculus

- Higher Abstraction: Relational Calculus enables a user to tell what information is needed but not how it needs to be retrieved which makes it easier to express yourself.
- User-Friendly for Non-Experts: This form is easier for the users who may not know the steps that have to be taken when formulating queries.

Disadvantages of Relational Calculus

- Less Optimizable: It can be said that this is a disadvantage of the ALG, as being non-procedural, it does not produce definite recommendations to be followed in optimization of a query, which is likely to result in inferior query processing.
- Potentially Ambiguous: The abstract nature can at time make it difficult to get the sense of interpretation on the query side and therefore predict results

Preliminaries

- Preliminaries in a Database Management System (DBMS) are the fundamental concepts needed to understand how databases are structured, managed, and interacted with. These core concepts lay the groundwork for everything from simple data retrieval to complex database design.

Core concepts

- Data and Information: Data consists of raw, unprocessed facts and figures. Information is organized, processed data that is given context and meaning.
- Database: A structured, logically related collection of data stored electronically, designed for efficient storage, access, and management.
- DBMS: The software system that allows users to create, define, update, and manage a database. It acts as an interface between the user and the database.
    - Database Schema: The overall structure or blueprint of the database, which defines the tables, attributes, relationships, and constraints.
    - Relational Model: A data model that organizes data into tables, where each table represents a real-world entity.

Relational model terminology

- Relation (Table): A collection of related data organized in rows and columns.
- Tuple (Row/Record): A single entry or record in a table.
- Attribute (Column/Field): A named characteristic or property that defines the data in a table.
- Domain: The set of permissible values for an attribute.
- Cardinality: The number of rows in a relation.

- Degree: The number of columns in a relation.

Keys

Keys are special attributes or sets of attributes used to uniquely identify records and establish relationships between tables.

- Primary Key: An attribute that uniquely identifies each record within a table. It cannot be null and must be unique.
- Foreign Key: An attribute in one table that references the primary key of another table, creating a link between them.
- Candidate Key: Any minimal set of attributes that can uniquely identify a tuple. All candidate keys are potential primary keys.

Data integrity constraints

These are rules enforced by the DBMS to ensure data quality and consistency.

- Domain Constraint: Ensures that all values in an attribute come from a valid set of values.
- Entity Integrity: Enforces that the primary key of a relation cannot be null, guaranteeing every record has a unique identifier.
- Referential Integrity: Ensures that a foreign key value must either be null or correspond to an existing primary key value in the referenced table.

SQL command categories

Structured Query Language (SQL) is the standard language for interacting with relational databases. Its commands are grouped into categories.

- Data Definition Language (DDL): Commands that define and modify the structure of database objects, including CREATE, ALTER, DROP, and TRUNCATE.
- Data Manipulation Language (DML): Commands that manipulate the data within the database, including INSERT, UPDATE, DELETE, and SELECT.
- Data Control Language (DCL): Commands that control access to data and manage permissions, such as GRANT and REVOKE.
- Transaction Control Language (TCL): Commands that manage database transactions, including COMMIT and ROLLBACK.

Relational Algebra

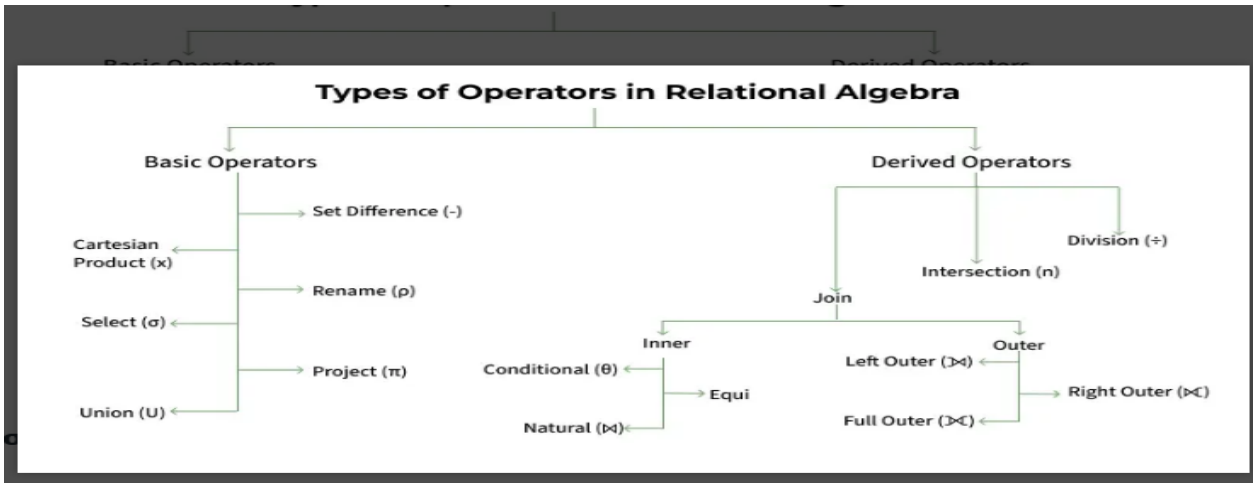
Relational Algebra is a formal language used to query and manipulate relational databases, consisting of a set of operations like selection, projection, union, and join. It provides a

mathematical framework for querying databases, ensuring efficient data retrieval and manipulation. Relational algebra serves as the mathematical foundation for query SQL

Relational algebra simplifies the process of querying databases and makes it easier to understand and optimize query execution for better performance. It is essential for learning SQL because SQL queries are based on relational algebra operations, enabling users to retrieve data effectively.

Basic Operators in Relational Algebra

Relational algebra consists of various basic operators that help us to fetch and manipulate data from relational tables in the database to perform certain operations on relational data. Basic operators are fundamental operations that include selection (σ), projection (π), union (U), set difference (−), Cartesian product (×), and rename (ρ).



Operators in Relational Algebra

1. Selection(σ)

The Selection Operation is basically used to filter out rows from a given table based on certain given condition. It basically allows us to retrieve only those rows that match the condition as per condition passed during SQL Query.

Example: If we have a relation R with attributes A, B, and C, and we want to select tuples where C > 3, we write:

A	B	C
1	2	4
2	2	3
3	2	3

4	3	4

σ (c>3) (R) will select the tuples which have c more than 3.

2. Projection(π)

While Selection operation works on **rows**, similarly projection operation of relational algebra works on columns. It basically allows us to pick specific columns from a given relational table based on the given condition and ignoring all the other remaining columns.

Example: Suppose we want columns B and C from Relation R.

π(B,C)(R)

Output:

B	C
2	4
2	3
3	4

Explanation: By Default, projection operation removes duplicate values.

3. Union(U)

The **Union** Operator is basically **used to combine the results of two queries into a single result**. The only condition is that both queries must return same number of columns with same data types. Union operation in relational algebra is the same as union operation in set theory.

Example: Consider the following table of Students having different optional subjects in their course.
Table1: FRENCH

Student_Name	Roll_Number
Ram	01
Mohan	02
Vivek	13
Giri	17

Table2 : GERMAN

Student_Name	Roll_Number
Vivek	13
Giri	17
Shyam	21
Rohan	25

If FRENCH and GERMAN relations represent student names in two subjects, we can combine their student names as follows:

π(Student_Name) (FRENCH) U π(Student_Name)(GERMAN)

Output:

Student_Name
Ram
Mohan
Vivek
Geeta
Shyam
Rohan

Explanation: The only constraint in the union of two relations is that both relations must have the same set of Attributes.

4. Set Difference(-)

Set difference basically provides the rows that are present in one table, but not in another tables. Set Difference in relational algebra is the same set difference operation as in set theory.

Example: To find students enrolled only in FRENCH but not in GERMAN, we write:

π(Student_Name)(FRENCH) - π(Student_Name)(GERMAN)

Student_Name
Ram

Student_Name
Mohan

Explanation: The only constraint in the Set Difference between two relations is that both relations must have the same set of Attributes.

5. Rename(ρ)

Rename operator basically allows you to give a temporary name to a specific relational table or to its columns. It is very useful when we want to avoid ambiguity, especially in complex Queries. Rename is a unary operation used for renaming attributes of a relation.

Example: We can rename an attribute **B** in relation **R** to **D**

A	B	C
1	2	4
2	2	3
3	2	3
4	3	4

ρ(D/B)R

will rename the attribute 'B' of the relation by 'D".

Output Table:

A	D	C
1	2	4
2	2	3
3	2	3
4	3	4

6. Cartesian Product(X)

The **Cartesian product** combines every row of one table with every row of another table, producing all the possible combination. It's mostly used as a precursor to more complex operation like joins. Let's say A and B, so the cross product between A X B will result in all the attributes of A followed by each attribute of B. Each record of A will pair with every record of B.

Relation A:

Name	Age	Sex
Ram	14	M
Sona	15	F
Kim	20	M

Relation B:

ID	Course
1	DS
2	DBMS

Output: If relation **A** has 3 rows and relation **B** has 2 rows, the Cartesian product **A × B** will result in 6 rows.

Name	Age	Sex	ID	Course
Ram	14	M	1	DS
Ram	14	M	2	DBMS

Name	Age	Sex	ID	Course
Sona	15	F	1	DS
Sona	15	F	2	DBMS
Kim	20	M	1	DS
Kim	20	M	2	DBMS

Explanation: If A has 'n' tuples and B has 'm' tuples then A X B will have 'n*m' tuples.

Derived operators are built using basic operators and include operations like join, intersection, and division. These operators help perform more complex queries by combining basic operations to meet specific data retrieval needs.

1. Join Operators

Join operations in relational algebra combine data from two or more relations based on a related attribute, allowing for more complex queries and data retrieval. Different types of joins include:

Inner Join

An inner join combines rows from two relations based on a matching condition and only returns rows where there is a match in both relations. If a record in one relation doesn't have a corresponding match in the other, it is excluded from the result. This is the most common type of join.

- **Conditional Join**: A conditional join is an inner join where the matching condition can involve any comparison operator like equals (=), greater than (>),
  etc. **Example:** Joining Employees and Departments on DepartmentID where Salary > 50000 will return employees in departments with a salary greater than 50,000

- **Equi Join**: An equi join is a type of conditional join where the condition is specifically equality (=) between columns from both
  relations. **Example:** Joining Customers and Orders on CustomerID where both relations have this column, returning only matching records.

- **Natural Join**: A natural join automatically combines relations based on columns with the same name and type, removing duplicate columns in the result. It's a more efficient way of joining. **Example:** Joining Students and Enrollments where StudentID is common in both, and the result contains only unique columns.

Outer Join

An outer join returns all rows from one relation, and the matching rows from the other relation. If there is no match, the result will still include all rows from the outer relation with NULL values in the columns from the unmatched relation.

- **Left Outer Join**: A left outer join returns all rows from the left relation and the matching rows from the right relation. If there is no match, the result will include NULL values for the right relation's attributes. **Example:** Joining Employees with Departments using a left outer join ensures all employees are listed, even those who aren't assigned to any department, with NULL values for the department columns.

- **Right Outer Join**: A right outer join returns all rows from the right relation and the matching rows from the left relation. If no match exists, the left relation's columns will contain NULL values. **Example:** Joining Departments with Employees using a right outer join includes all departments, even those with no employees assigned, filling unmatched employee columns with NULL.

- **Full Outer Join**: A full outer join returns all rows when there is a match in either the left or right relation. If a row from one relation does not have a match in the other, NULL values are included for the missing side. **Example:** Joining Customers and Orders using a full outer join will return all customers and orders, even if there's no corresponding order for a customer or no customer for an order.

2. Set Intersection(∩)

Set Intersection basically allows to fetches only those rows of data that are common between two sets of relational tables. Set Intersection in relational algebra is the same set intersection operation in set theory.

Example: Consider the following table of Students having different optional subjects in their course.

Relation FRENCH

Student_Name	Roll_Number
Ram	01

Student_Name	Roll_Number
Mohan	02
Vivek	13
Giri	17

Relation GERMAN

Student_Name	Roll_Number
Vivek	13
Giri	17
Shyam	21
Rohan	25

From the above table of FRENCH and GERMAN, the Set Intersection is used as follows:

π(Student_Name)(FRENCH ∩ π(Student_Name)(GERMAN) **Output:**

Student_Name
Vivek
Giri

Explanation: The only constraint in the Set Difference between two relations is that both relations must have the same set of Attributes.

3. Division (÷)

The Division Operator is used to find tuples in one relation that are related to all tuples in another relation. It's typically used for "**for all**" queries.

Student_Course (Dividend Table):

Student_ID	Course_ID
101	C1
101	C2
102	C1
103	C1
103	C2

Course (Divisor Table):

Course_ID
C1
C2

Example: Query is to find students who are enrolled in all courses listed in the Course table. In this case, students must be enrolled in both C1 and C2.

Student_Course(Student_ID, Course_ID)÷ Course(Course_ID)

Output:

Student_ID
101
103

Relational Calculus

Relational Calculus in a Database Management System (DBMS) is a non-procedural query language used to define what data to retrieve, rather than how to retrieve it. It is based on predicate calculus and allows expressing queries using logical formulas and quantifiers. There are two main types:

1. Tuple Relational Calculus (TRC)

TRC focuses on tuples (rows) and uses tuple variables to represent individual rows in a relation.

Syntax:

Code :
{t | P(t)}

where t is a tuple variable and P(t) is a logical formula (predicate) that describes the conditions the tuples in the result must satisfy.

Example:

Consider a STUDENT table with attributes StudentID, Name, and Age. To retrieve the names of students older than 20:

Code :
{t.Name | STUDENT(t) AND t.Age > 20}

Explanation: This query selects the Name attribute of all tuples t such that t belongs to the STUDENT relation and the Age attribute of t is greater than 20.

2. Domain Relational Calculus (DRC)

DRC focuses on attribute values (domains) and uses domain variables to represent individual attribute values.

Syntax:

Code:
{<x1, x2, ..., xn> | P(x1, x2, ..., xn)}

where x1, x2, ..., xn are domain variables representing attribute values, and P(...) is a logical formula describing conditions on these values.

Example:

Using the same STUDENT table, to retrieve the names of students older than 20:

Code:
{<N> | ∃ S, A (<S, N, A> ∈ STUDENT AND A > 20)}

Explanation: This query selects the domain variable N (representing Name) such that there exist domain variables S (StudentID) and A (Age) forming a tuple <S, N, A> within the STUDENT relation, and the value of A is greater than 20

Expressive power of algebra and Calculus

calculus builds upon and surpasses that of algebra by enabling the description and analysis of continuous change, motion, and infinite processes. While algebra provides the foundational tools to work with static relationships and unknown values, calculus extends this ability to handle dynamic situations.

Expressive power of algebra

Algebra is a generalization of arithmetic that uses symbols, or variables, to represent numbers and express relationships. Its expressive power lies in its ability to model and solve problems involving fixed, discrete quantities.

- **Static relationships**: Algebra is used to describe and solve equations that involve a set relationship between variables, such as y=2x+1y equals 2 x plus 1$y=2x+1$
- **Discrete problems**: It can find specific, unknown values based on given conditions, such as determining how many apples can be afforded with a certain amount of cash.
- **Average rate of change**: With algebra, one can calculate the slope of a straight line, which represents a constant or average rate of change between two points.
- **Geometric shapes with constant dimensions**: It can be used to find the areas and volumes of regularly shaped objects, such as a circle or a rectangular prism

Expressive power of calculus

Calculus introduces the concept of the infinite, allowing it to move beyond static relationships and describe the nuances of continuous change. Its expressive power allows for the analysis of rates, accumulations, and motion at an instantaneous, rather than just average, level.

- **Dynamic relationships and instantaneous rates**: Calculus uses derivatives to find the instantaneous rate of change of a function at any given point. This allows for the precise analysis of motion and growth in dynamic systems. For example, a derivative can find the exact velocity of a car at a particular moment, even if its acceleration is changing.

- **Infinite sums and accumulations**: Integral calculus uses infinite sums to calculate the total accumulation of a quantity, such as the area under a curve. This is impossible with algebra alone.
- **Optimal values**: Calculus can be used for optimization problems—finding the maximum or minimum value of a function—such as determining the shape of an enclosure that maximizes area for a given perimeter.
- **Geometric shapes with variable boundaries**: It can find the area and volume of irregularly shaped objects by summing an infinite number of infinitesimally small parts.