BTCOC501

Database Systems

Teaching Notes

Lecture Number	Topic to be covered
1	**Unit II - Relational Data Model, Relational Algebra and Calculus (07 Hrs)** ➢ Structure of Relational Databases ➢ Database Schema
2	➢ Keys Relational algebra: Fundamental Operations,
3	➢ Additional Relational Algebra Operations,
4	➢ Extended Relational Algebra Operations
5	➢ Calculus: Tuple relational calculus,
6	➢ Domain relational Calculus
7	➢ Calculus vs algebra

Submitted by:

Prof. S. H. Sable

Nutan College Of Engineering & Research,
Talegaon Dabhade, Pune- 410507

DEPARTMENT OF THIRD YEAR ENGINEERING

Database systems

Unit 2 – Relational Data Model, Relational Algebra and Calculus

1. Structure of Relational Databases

A relational database consists of a collection of tables, each of which is assigned a unique name.A row in a table represents a relationship among a set of values. Since a table is a collection of such relationships,there is a close correspondence between the concept of table and the mathematical concept of relation,from which the relational data model takes its name. The table name and column names are helpful to interpret the meaning of values in each row. The data are represented as a set of relations. However, the physical storage of the data is independent of the way the data are logically organized.

Basic Structure

consider the customer table of Figure 1, which stores information about customers in bank The table has four column headers: Account_no,ename,street ,city.

Account _no	ename	street	ecity
101	Johnson	Pender	Vanconver
215	Smith	Nirth	Burnaby
102	Hayes	Cirtis	Burnaby
304	Adams	Ng road	Richmond
505	James	Oak	Vancover

bname	Account _no	ename	Balance
Downtown	101	Johnson	300
Longheed mall	215	Smith	700
SPU	102	Hayes	400
SPU	304	Adam	1300
Downtown	505	James	500

Figure 1: The *deposit* and *customer* relations(tables).

- **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.
- **Attribute:** Attributes are the properties that define a relation and is the name of column in the table
  The customer table has four attributes. Account_no,ename,street ,city.
- **Relation Schema:** A relation schema represents name of the relation with its attributes. e.g.; Customer (Account_no,ename,street ,city) is relation schema for Customer Table.
- **Tuple:** Each row in the relation is known as tuple. The above relation contains 4 tuples, one of which is shown as:
  101 Johnson pender Vanconver
- **Relation Instance:** The set of tuples of a relation at a particular instance of time is called as relation instance. Table 1 shows the relation instance of CUTOMER at a particular time. It can change whenever there is insertion, deletion or updation in the database.
- **Degree:** The number of attributes in the relation is known as degree of the relation. The **Customer** relation defined above has degree 4.
- **Cardinality:** The number of tuples in a relation is known as cardinality. The **customer** relation defined above has cardinality 5
- **NULL Values:** The value which is not known or unavailable is called NULL value. It is represented by blank space
- **Relation key** - Every row has one, two or multiple attributes, which is called relation key.
- **Attribute domain** :For each attribute there is a permitted set of values, called the **domain** of that attribute.
- E.g. the domain of *bname* is the set of all branch names.Let D_1 denote the domain of *bname*, and D_2 denote the domain of *ACCOUNT_NO*, D_3 denote the domain of *ename* and D_4 denote the domain of *balance* .Then, any row of *deposit* consists of a four-tuple(v1,v2,v3,v4) where

$$v_1 \in D_1, v_2 \in D_2, v_3 \in D_3, v_4 \in D_4$$

In general, *deposit* contains a subset of the set of all possible rows.That is, *deposit* is a subset of

$$D_1 \times D_2 \times D_3 \times D_4, \quad or, \ abbreviated \ to, \quad \times_{i=1}^{4} D_i$$

In general, a table of n columns must be a subset of

$$\times_{i=1}^{n} D_i \quad \text{(all possible rows)}$$

Mathematicians define a relation to be a subset of a Cartesian product of a list of domains. You can see the correspondence with our tables.

Use the terms **relation** and **tuple** in place of **table** and **row** from now on.

1. Some more formalities:
   - let the tuple variable t refer to a tuple of the relation r.
   - We say $t \in r$ to denote that the tuple t is in relation r.
   - Then t[*bname*] = t[1] = the value of t on the *bname* attribute.
   - So t[*bname*] = t[1] = ``Downtown",

- and $\mathbf{t}$[*cname*] = $\mathbf{t}$[3] = ``Johnson".

2. Database Scheme

1.We distinguish between a **database scheme** (logical design) and a **database instance** (data in the database at a point in time).
2.A **relation scheme** is a list of attributes and their corresponding domains.
3. The text uses the following conventions:
   - italics for all names
   - lowercase names for relations and attributes
   - names beginning with an uppercase for relation schemes

For example, the relation scheme for the *deposit* relation:

- ***Deposit-scheme = (bname, account_no, cname, balance)***

We may state that *deposit* is a relation on scheme *Deposit-scheme* by writing *deposit(Deposit-scheme)*.

If we wish to specify domains, we can write:

- ***(bname: string, account_no: integer, cname: string, balance: integer).***

Note that customers are identified by name. In the real world, this would not be allowed, as two or more customers might share the same name.

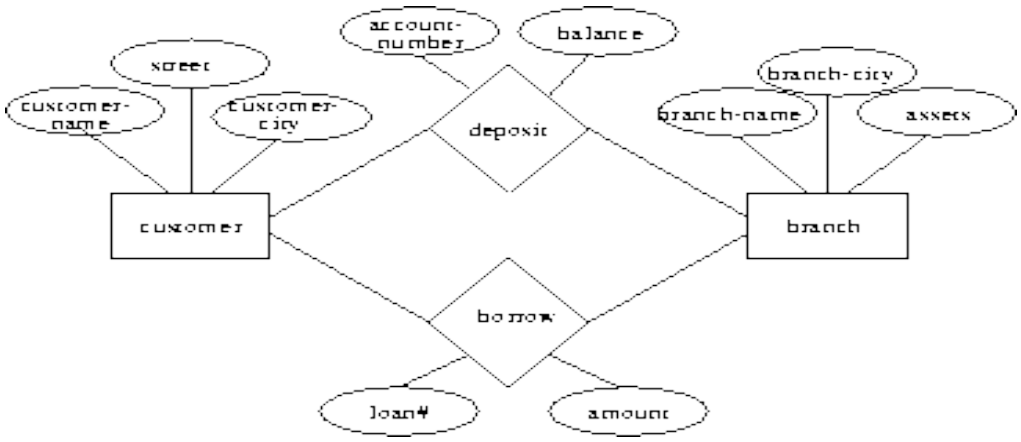


Figure : E-R diagram for the banking enterprise

4. The relation schemes for the banking example used throughout the text are:
   - *Branch-scheme = (bname, assets, bcity)*
   - *Customer-scheme = (cname, street, ccity)*
   - *Deposit-scheme = (bname, account#, cname, balance)*
   - *Borrow-scheme = (bname, loan, cname, amount)*

some attributes appear in several relation schemes (e.g. *bname, cname*). This is legal, and provides a way of **relating** tuples of distinct relations.

5. **Why not put all attributes in one relation?**

Suppose we use one large relation instead of *customer* and *deposit*:
Account-scheme = (bname, account#, cname, balance, street, ccity)

If a customer has several accounts, we must duplicate her or his address for each account.

- If a customer has an account but no current address, we cannot build a tuple, as we have no values for the address.
- We would have to use **null values** for these fields.
- Null values cause difficulties in the database.
- By using two separate relations, we can do this without using null values

3. Key Constraints in Relational Model

While designing Relational Model, we define some conditions which must hold for data present in database are called Constraints. These constraints are checked before performing any operation (insertion, deletion and updation) in database. If there is a violation in any of constrains, operation will fail.

Key Integrity: Every relation in the database should have atleast one set of attributes which defines a tuple uniquely. Those set of attributes is called key. e.g.; ROLL_NO in STUDENT is a key. No two students can have same roll number. So a key has two properties:

- It should be unique for all tuples.
- It can't have NULL values.

Referential Integrity: When one attribute of a relation can only take values from other attribute of same relation or any other relation, it is called referential integrity. Let us suppose we have 2 relations

STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE	BRANCH_CODE
1	RAM	DELHI	9455123451	18	CS
2	RAMESH	GURGAON	9652431543	18	CS
3	SUJIT	ROHTAK	9156253131	20	ECE
4	SURESH	DELHI		18	IT

BRANCH

BRANCH_CODE	BRANCH_NAME
CS	COMPUTER SCIENCE
IT	INFORMATION TECHNOLOGY
ECE	ELECTRONICS AND COMMUNICATION ENGINEERING
CV	CIVIL ENGINEERING

BRANCH_CODE of STUDENT can only take the values which are present in BRANCH_CODE of BRANCH which is called referential integrity constraint. The relation which is referencing to other relation is called REFERENCING RELATION (STUDENT in this case) and the relation to which other relations refer is called REFERENCED RELATION (BRANCH in this case).

Student Table

Domain Constraints:

ROLL_NO	NAME	ADDRESS	PHONE	AGE	BRANCH_CODE
1	RAM	DELHI	9455123451	18	CS
2	RAMESH	GURGAON	9652431543	18	CS
3	SUJIT	ROHTAK	9156253131	20	ECE
4	SURESH	DELHI		18	IT

These are attribute level constraints. An attribute can only take values which lie inside the domain range. This is specified as data types which include standard data types integers, real numbers, characters, Booleans, variable length strings, etc e.g,; In student table if a roll_no has integer datatype and we try to insert character value constraints value of AGE it will result in failure.

Query language

A query language is a language in which a user requests information from the database. These languages are usually on a level higher than that of a standard programming language.Query languages can be categorized as either procedural or non procedural. In a procedural language, the user instructs the system to perform a sequence of operations on the database to compute the desired result User has to specify "what to do" and also "how to do" (step by step procedure). In a non procedural language, the user describes the desired information without giving a specific procedure for obtaining that information.The user has to specify only "what to do" and not "how to do"

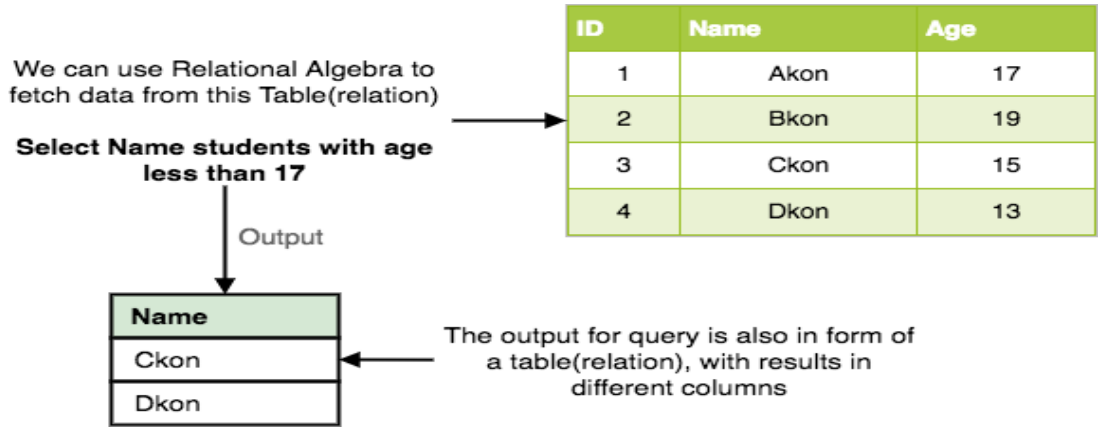
4. Relational algebra

Relational Algebra is procedural query language, which takes Relation as input andgenerates relation as output. Relational algebra mainly provides theoretical foundation for relational databases and SQL.

Relational algebra is a procedural query language, it means that it tells what data to be retrieved and how to be retrieved.

Relational Algebra works on the whole table at once, so we do not have to use loops etc to iterate over all the rows (tuples) of data one by one.

All we have to do is specify the table name from which we need the data, and in a single line of command, relational algebra will traverse the entire given table to fetch data for you.



We will use STUDENT_SPORTS, EMPLOYEE, and STUDENT relations as given in Table 1, Table 2, and Table 3 respectively to understand the various operators.

Table 1: STUDENT_SPORTS

ROLL_NO	SPORTS
1	Badminton
2	Cricket
2	Badminton
4	Badminton

Table 2: EMPLOYEE

EMP_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
5	NARESH	HISAR	9782918192	22
6	SWETA	RANCHI	9852617621	21
4	SURESH	DELHI	9156768971	18

Table 3: STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	9156768971	18

Basic/Fundamental Operations of relational algebra:

1. Select (σ)
2. Project (∏)
3. Union (U)
4. Set Difference (-)
5. Cartesian product (X)
6. Rename (ρ)

1. Select Operation (σ) :

This is used to fetch rows (tuples) from table(relation) whichsatisfies a given condition. Select is denoted by a lowercase Greek sigma (σ)with the predicate appearing as a subscript. The argument relation is given in parentheses following the σ.
We allow comparisons using =, $\neq$, <, $\leq$, > and $\geq$ in the selection predicate.
We also allow the logical connectives $\vee$ (or) and $\wedge$ (and).

Syntax: $\sigma_{(Cond)(Relation\ Name)}$ **or** $\sigma p(r)$

- σ is the predicate
- r stands for relation which is the name of the table/relation name is name of table

- p is prepositional logic
- Cond: Condition

Example1:
Find the students who have age is greater than 18 from STUDENT relation given in Table 3
σage > 18 (Student)

RESULT:

ROLL_NO	NAME	ADDRESS	PHONE	AGE
3	SUJIT	ROHTAK	9156253131	20

This will fetch the tuples(rows) from table **Student**, for which **age** will be greater than **18**.

Example2: Find the students who have age is greater than 18 and also the address of the student should be Delhi from STUDENT relation given in table3

σage > 18 and address = 'Delhi' (Student)

This will return tuples(rows) from table **Student** with information of students from delhi of age more than 18.

RESULT:

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
4	SURESH	DELHI	9156768971	18

Note: SELECT operator does not show any result, the projection operator must be called before the selection operator to generate or project the result. So, the correct syntax to generate the result is**:**
∏(σ (AGE>18)(STUDENT))

2. **Project Operation (∏):**
Project operation is used to project only a certain set of attributes of a relation. Projection is denoted by the Greek capital letter pi (**Π**). The attributes to be copied appear as subscripts. Duplicate rows are eliminated.

Syntax :

∏(Column 1,Column 2….Column n)(Relation Name)

Example1: Extract ROLL_NO and NAME from STUDENT relation/table

Query: **∏(ROLL_NO,NAME)(STUDENT)**

RESULT:

ROLL_NO	NAME
1	RAM
2	RAMESH
3	SUJIT
4	SURESH

Note: If the resultant relation after projection has duplicate rows, it will be removed. For Example $\prod_{(ADDRESS)}$(STUDENT) will remove one duplicate row with the value DELHI and return three rows.

3. Cross Product(X):

Cross product is used to join two relations. The cross **product** of two relations is denoted by a cross (**×**).This is used to combine data from two different relations(tables) into one and fetch data fromthe combined relation.

For every row of Relation1, each row of Relation2 is concatenated. If Relation1 has m tuples and and Relation2 has n tuples, cross product of Relation1 and Relation2 will have m X n tuples.

Syntax:
Relation1 X Relation2

Example 1:

Table 1: STUDENT_SPORTS

ROLL_NO	SPORTS
1	Badminton
2	Cricket
2	Badminton
4	Badminton

Table 3: STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	9156768971	18

To apply Cross Product on STUDENT relation given in Table 1 and STUDENT_SPORTS relation given in Table 2

STUDENT X STUDENT_SPORTS

RESULT:

ROLL_NO	NAME	ADDRESS	PHONE	AGE	ROLL_NO	SPORTS
1	RAM	DELHI	9455123451	18	1	Badminton
1	RAM	DELHI	9455123451	18	2	Cricket
1	RAM	DELHI	9455123451	18	2	Badminton
1	RAM	DELHI	9455123451	18	4	Badminton
2	RAMESH	GURGAON	9652431543	18	1	Badminton
2	RAMESH	GURGAON	9652431543	18	2	Cricket
2	RAMESH	GURGAON	9652431543	18	2	Badminton
2	RAMESH	GURGAON	9652431543	18	4	Badminton
3	SUJIT	ROHTAK	9156253131	20	1	Badminton
3	SUJIT	ROHTAK	9156253131	20	2	Cricket
3	SUJIT	ROHTAK	9156253131	20	2	Badminton
3	SUJIT	ROHTAK	9156253131	20	4	Badminton
4	SURESH	DELHI	9156768971	18	1	Badminton
4	SURESH	DELHI	9156768971	18	2	Cricket

4	SURESH	DELHI	9156768971	18	2	Badminton
4	SURESH	DELHI	9156768971	18	4	Badminton

4. **Union (U):**
The union operation is denoted ∪ as in set theory.Union on two relations R1 and R2 can only be computed if R1 and R2 are **union compatible** (**These two relations should have the same number of attributes and corresponding attributes in two relations have the same domain**). This operation is used to fetch data from two relations(tables) or temporary relation (result of another operation).Union operator when applied on two relations R1 and R2 will give a relation with tuples that are either in R1 or in R2. The tuples which are in both R1 and R2 will appear only once in the result relation.

Syntax:
Relation1 U Relation2

Table 2: EMPLOYEE

EMP_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
5	NARESH	HISAR	9782918192	22
6	SWETA	RANCHI	9852617621	21
4	SURESH	DELHI	9156768971	18

Table 3: STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	9156768971	18

Example1:Find the name,address of person who is either student or employees, we can use Union operators like:

STUDENT U EMPLOYEE

$$\prod_{NAME,ADDRESS} (STUDENT) \cup \prod_{NAME,ADDRESS} (EMPLOYEE)$$

RESULT:

NAME	ADDRESS
RAM	DELHI
RAMESH	GURGAON
SUJIT	ROHTAK
SURESH	DELHI
NARESH	HISAR
SWETA	RANCHI

5. **Set Difference /Minus(-):**
Set difference is denoted by the minus sign (–). It finds tuples that are in one relation, but not in another.**Set Difference** Minus on two relations R1 and R2 can only be computed if R1 and R2 are **union compatible**(**These two relations should have the same number of attributes and**

corresponding attributes in two relations have the same domain). Minus operator when applied on two relations as R1-R2 will give a relation with tuples that are in R1 but not in R2.

Syntax:

Relation1 - Relation2

Table 2: EMPLOYEE

EMP_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
5	NARESH	HISAR	9782918192	22
6	SWETA	RANCHI	9852617621	21
4	SURESH	DELHI	9156768971	18

Table 3: STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	9156768971	18

Example1:

Find the,name and address of person who is a student but not an employee, we can use minus operator like:

STUDENT – EMPLOYEE

$$\prod \text{NAME,ADDRESS (STUDENT)} - \prod \text{NAME,ADDRESS (EMPLOYEE)}$$

RESULT:

NAME	ADDRESS
RAMESH	GURGAON
SUJIT	ROHTAK

6. Rename(ρ):

The rename operation is used to rename the output relation. It is denoted by **rho** (ρ). Rename operator is used to giving another name to a relation. The **rename** operation solves the problems that occurs with naming when performing the cartesian product of a relation with itself.

Syntax:

ρ(Relation2, Relation1)

To rename STUDENT relation to STUDENT1, we can use rename operator like:

ρ(STUDENT1, STUDENT)

Result:

Table3 STUDENT1

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	9156768971	18

Example2:

If you want to create a relation STUDENT_NAMES with ROLL_NO and NAME from STUDENT, it can be done using rename operator as:

ρ(STUDENT_NAMES), $\prod_{(ROLL\_NO, NAME)(STUDENT))}$

Table3 STUDENT_NAMES

Result:

ROLL_NO	NAME
1	RAM
2	RAMESH
3	SUJIT
4	SURESH

Example 3:

Find the query to the students who have age 18 from student table relation and rename the relation student as student_age and the attributes of students roll_no ,name as sno,sname

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	9156768971	18

$\rho STUDENT\_AGE_{(sno,sname)} \prod_{(ROLL\_NO, NAME)}(\sigma_{(AGE>18)}(STUDENT))$

Result:

Table3 STUDENT_AGE

SNO	SNAME
1	RAM
2	RAMESH
4	SURESH

5. Additional Relational Algebra Operations:

Additional operations are defined in terms of the fundamental operations. They do not add power to the algebra, but are useful to simplify common queries.

- **Join**
- **Intersection**
- **Divide**

a) **Intersection ($\cap$):** It is denoted by intersection $\cap$. Intersection on two relations R1 and R2 can only be computed if R1 and R2 are **union compatible** (These two relation

should have same number of attributes and corresponding attributes in two relations have same domain). Intersection operator when applied on two relations as R1∩R2 will give a relation with tuples which are in R1 as well as R2. The set intersection operation contains all tuples that are in both R1 & R2.

Syntax:

Relation1 ∩ Relation2

Table 2: EMPLOYEE

EMP_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
5	NARESH	HISAR	9782918192	22
6	SWETA	RANCHI	9852617621	21
4	SURESH	DELHI	9156768971	18

Table 3: STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	9156768971	18

STUDENT ∩ EMPLOYEE

Example1:

Find a name and address of person who is student as well as employee-

$\prod_{(NAME,address)}$**(STUDENT)** ∩ $\prod_{(NAME,address)}$**(EMPLOYEE)**

RESULT:

NAME	ADDRESS
RAM	DELHI
SURESH	DELHI

b) The Division Operation:

Division, denoted ÷, is suited to queries that include the phrase ``for all".

Division operator A÷B can be applied if and only if:

- Attributes of B is proper subset of Attributes of A.
- The relation returned by division operator will have attributes = (All attributes of A – All Attributes of B)
- The relation returned by division operator will return those tuples from relation A which are associated to every B's tuple

To apply division operator as

STUDENT_SPORTS÷ ALL_SPORTS

- The operation is valid as attributes in ALL_SPORTS is a proper subset of attributes in STUDENT_SPORTS.
- The attributes in resulting relation will have attributes {ROLL_NO,SPORTS}-{SPORTS}=ROLL_NO

.

Consider the relation STUDENT_SPORTS and ALL_SPORTS given in Table 2 and Table 3 above.

STUDENT_SPORTS

ROLL_NO	SPORTS
1	Badminton
2	Cricket
2	Badminton
4	Badminton

ALL_SPORTS

SPORTS
Badminton
Cricket

Example 1:
Find the roll no of student who play all sports

STUDENT_SPORTS÷ ALL_SPORTS

$\prod_{(Roll\_no)}$(STUDENT_SPORTS) – ($\prod_{ROLL\_NO}$ (($\prod_{(ROLL\_NO)}$(STUDENT_SPORTS) X$\prod_{(SPORTS)}$(ALL_SPORTS)- ∏(STUDENT_SPORTS))))

The tuples in resulting relation will have those ROLL_NO which are associated with all B's tuple {Badminton, Cricket}. ROLL_NO 1 and 4 are associated to Badminton only. ROLL_NO 2 is associated to all tuples of B. So the resulting relation will be:
Result:

ROLL_NO
2

c) The Assignment Operation

Sometimes it is useful to be able to write a relational algebra expression in parts using a temporary relation variable The assignment operation, denoted ← works like assignment in a programming language.

We could rewrite our division definition as

STUDENT_SPORTS÷ ALL_SPORTS

$\prod_{(Roll\_no)}$(STUDENT_SPORTS) – ($\prod_{ROLL\_NO}$((($\prod_{(ROLL\_NO)}$(STUDENT_SPORTS) X$\prod_{(SPORTS)}$(ALL_SPORTS)- ∏(STUDENT_SPORTS))))

temp1←**$\prod_{(Roll\_no)}$(STUDENT_SPORTS)**
temp2←**(∏ Roll_no((temp1)X$\prod_{(SPORTS)}$(ALL_SPORTS)- ∏(STUDENT_SPORTS))))**
result←temp1-temp2

No extra relation is added to the database, but the relation variable created can be used in subsequent expressions.

d) Join in DBMS:

- A JOIN clause is used to combine rows from two or more tables, based on a relatedcolumn between them.
- **Join in DBMS** is a binary operation which allows you to combine join product and selection in one single statement.

- The goal of creating a join condition is that it helps you to combine the data from two or more DBMS tables.
- The tables in DBMS are associated using the primary key and foreign keys.

Types of SQL JOIN

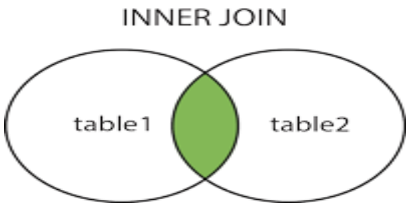
There are mainly two types of joins in DBMS:

1. Inner Joins
2. Outer Join

Inner Join

Inner Join is used to return rows from both tables which satisfy the given condition. In SQL, INNER JOIN selects records that have matching values in both tables as long as the condition is satisfied.

Inner Join further divided into three subtypes:



1. Theta join
2. Natural join
3. EQUI join

1) Theta /Conditional Join

Theta Join allows you to merge two tables based on the condition represented by theta. Theta joins work for all comparison operators(<,>,>=.<=,=,!=).

It is denoted by symbol **θ or C**

Conditional Join is used when you want to join two or more relation based on some conditions

Example1: Select students whose ROLL_NO is greater than EMP_NO of employees

. Table 2: EMPLOYEE

EMP_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
5	NARESH	HISAR	9782918192	22
6	SWETA	RANCHI	9852617621	21
4	SURESH	DELHI	9156768971	18

Table 3: STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	9156768971	18

Consider the tables student and employee for conditional join on roll_no from student table greater than and emp_no from employee table

STUDENT $\bowtie_{c\ \text{STUDENT.ROLL\_NO>EMPLOYEE.EMP\_NO}}$ **EMPLOYEE**

In terms of basic operators (cross product and selection) :

$\sigma_{(STUDENT.ROLL\_NO>EMPLOYEE.EMP\_NO)}(STUDENT \times EMPLOYEE)$

Result:

ROLL_NO	NAME	ADDRESS	PHONE	AGE	EMP_NO	NAME	ADDRESS	PHONE	AGE
2	RAMESH	GURGAON	9652431543	18	1	RAM	DELHI	9455123451	18
3	SUJIT	ROHTAK	9156253131	20	1	RAM	DELHI	9455123451	18
4	SURESH	DELHI	9156768971	18	1	RAM	DELHI	9455123451	18

2) **Equijoin*(⋈):***

Equijoin is a **special case of conditional join** where only equality condition holds between a pair of attributes. As values of two attributes will be equal in result of equijoin, only one attribute will be appeared in result.
Example:

Select students whose ROLL_NO is equal to EMP_NO of employees

Table 2: EMPLOYEE

EMP_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
5	NARESH	HISAR	9782918192	22
6	SWETA	RANCHI	9852617621	21
4	SURESH	DELHI	9156768971	18

Table 3: STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	9156768971	18

Example 1:

Consider the tables student and employee for equi join on roll_no and emp_no

$STUDENT \bowtie_{STUDENT.ROLL\_NO=EMPLOYEE.EMP\_NO} EMPLOYEE$

In terms of basic operators (cross product, selection and projection)

$\prod_{(STUDENT.ROLL\_NO,\ STUDENT.NAME,\ STUDENT.ADDRESS,\ STUDENT.PHONE,\ STUDENT.AGE,\ EMPLOYEE.NAME,\ EMPLOYEE.ADDRESS,\ EMPLOYEE.PHONE,\ EMPLOYEE.AGE)} (\sigma_{(STUDENT.ROLL\_NO=EMPLOYEE.EMP\_NO)} (STUDENT \times EMPLOYEE))$

Result:

ROLL_NO	NAME	ADDRESS	PHONE	AGE	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18	RAM	DELHI	9455123451	18

4	SURESH	DELHI	9156768971	18	RAM	DELHI	9455123451	18

3) Natural Join(⋈):

It is a special case of equijoin in which equality condition hold on all attributes which have same name in relations R and S (relations on which join operation is applied). In this type of join, the attributes should have the same name and domain. In Natural Join, there should be at least one common attribute between two relations.While applying natural join on two relations, there is no need to write equality condition explicitly. Natural Join performs selection forming equality on those attributes which appear in both relations and eliminates the duplicate attributes.

Example1: Select students whose ROLL_NO is equal to ROLL_NO of STUDENT_SPORTS as: **STUDENT⋈STUDENT_SPORTS**

In terms of basic operators (cross product, selection and projection) :

Consider the tables student and student_sports join on Roll_no from both tables

STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	9156768971	18

STUDENT_SPORTS

ROLL_NO	SPORTS
1	Badminton
2	Cricket
2	Badminton
4	Badminton

∏(STUDENT.ROLL_NO, STUDENT.NAME, STUDENT.ADDRESS, STUDENT.PHONE, STUDENT.AGE STUDENT_SPORTS.SPORTS) (σ (STUDENT.ROLL_NO=STUDENT_SPORTS.ROLL_NO) (STUDENT×STUDENT_SPORTS))

RESULT:

ROLL_NO	NAME	ADDRESS	PHONE	AGE	SPORTS
1	RAM	DELHI	9455123451	18	Badminton
2	RAMESH	GURGAON	9652431543	18	Cricket
2	RAMESH	GURGAON	9652431543	18	Badminton
4	SURESH	DELHI	9156768971	18	Badminton

Natural Join is by default inner join because the tuples which does not satisfy the conditions of join does not appear in result set. e.g.; The tuple having ROLL_NO 3 in STUDENT does not match with any tuple in STUDENT_SPORTS, so it has not been a part of result set.

Outer Join

An **Outer Join** doesn't require each record in the two join tables to have a matching record. In this type of join, the table retains each record even if no other matching record exists.

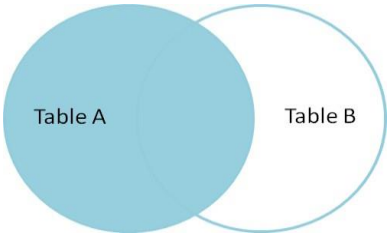
Three types of Outer Joins are:

- Left Outer Join
- Right Outer Join
- Full Outer Join

1) Left Join(⟕):

The SQL left join returns all the values from left table and the matching values from the right table. When applying join on two relations R and S, some tuples of R or S does not appear in result set which does not satisfy the join conditions. But Left Outer Joins gives all tuples of R in the result set. The tuples of R which do not satisfy join condition will have values as NULL for attributes of S.It is represented by ⟕.

Syntax:Relation1 ⟕Relation 2



Example1:

Find square and cube of number where square table number matches with cube table number and details of other number also

Tsquare

Num	Square
2	4
3	9
4	16

TCube

Num	Cube
2	8
3	27
5	125

TSquare⟕(Tsquare.num=Tcube.num)TCube

TSqaure⟕ TCube

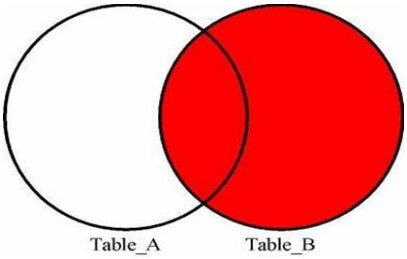
Num	Square	Cube
2	4	8
3	9	27
4	16	–

2) Right Join(⟖):

RIGHT JOIN returns all the values from the values from the rows of right table andthe matched values from the left table. When applying join on two relations R and S, some tuples of R or S does not appear in result set which does not satisfy the join conditions. But Right Outer Joins gives all tuples of S in the result set. The tuples of S which do not satisfy join condition will have values as NULL for attributes of R.

It is represented by ⟖

Syntax:Relation1 ⟖Relation 2



Example1:

Find square and cube of number where square table number matches with cube table number and details of other number also

Tsquare

Num	Square
2	4
3	9
4	16

TCube

Num	Cube
2	8
3	27
5	125

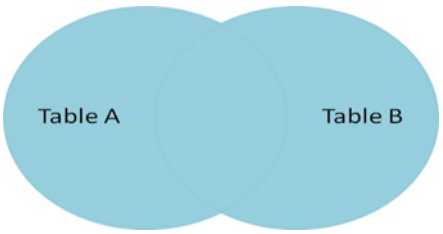
TSquare⋈(Tcube.num=Tsquare.num) ETCube

TSqaure⋈TCube

Num	Cube	Square
2	8	4
3	27	9
5	125	–

3) *Full Join(⋈):*

FULL JOIN is the result of a combination of both left and right outer join. Join tables have all the records from both tables.
When applying join on two relations R and S, some tuples of R or S does not appear in result set which does not satisfy the join conditions. But Full Outer Joins gives all tuples of S and all tuples of R in the result set. The tuples of S which do not satisfy join condition will have values as NULL for attributes of R and vice versa.



Tsquare

Num	Square
2	4
3	9
4	16

TCube

Num	Cube
2	8
3	27
5	125

TSquare⋈(Tcube.num=Tsquare.num)TCube

Example 1:
Find square and cube of number where square table number matches with cube table number and details of other number also

TSquare⟗ETCube

Num	Square	Cube
2	4	8
3	9	27
4	16	–
5	–	125

6. Extended Relational Algebra Operations.

These operations enhance the expressive power of the original relational algebra.

1.Generalized Projection

The generalized projection operation extends the projection operation by allowing functions of attributes to be included in the projection list. The generalized form can be expressed as:

$$\pi_{F1, F2, \ldots, Fn}(R)$$

where F_1, F_2, ..., F_n are functions over the attributes in relation R and may involve arithmetic operations and constant values. This operation is helpful when develop-ing reports where computed values have to be produced in the columns of a query result.

Example 1: consider the relation

EMPLOYEE (Ssn, Salary, Deduction, Years_service)

Net Salary = Salary – Deduction,

Bonus = 2000 * Years_service, and

Tax = 0.25 * Salary.

Then a generalized projection combined with renaming may be used as follows:

REPORT $\leftarrow \rho_{(Ssn, Net\_salary, Bonus, Tax)}(\pi_{(Ssn, Salary - Deduction, 2000 * Years\_service, Tax)=0.25 * Salary)}$ **(EMPLOYEE).**

Example 2:

consider the relation

Credit_info (Cust_name, limit, credit_balance)

cust_name	limit	credit_balance
A	5000	2000
B	6000	4000
C	10000	6000

Example 1:

Find how much each customer can spend.

Π(cust_name,limit-credit_balance)

cust_name	Limit- credit_balance
A	3000
B	2000
C	4000

2. Aggregation/Aggregate Function

Aggregate functions on collections of values from the data-base. Examples of such functions include retrieving the average or total salary of all employees or the total number of employee tuples. These functions are used in simple statistical queries that summarize information from the database tuples. Common functions applied to collections of numeric values include SUM, AVERAGE, MAXIMUM, and MINIMUM.
The COUNT function is used for counting tuples or values.

We can define an AGGREGATE FUNCTION operation, using the symbol g
<grouping attributes> ^{g}<function list> (relation)where <grouping attributes> is a list of attributes of the relation specified in *R by which we want to group*, and <function list> is a list of (<function> <attribute>) pairs.
If no grouping attributes are specified, the functions are applied to *all the tuples* in the relation, so the resulting relation has a *single tuple only*. Duplicates are *not eliminated* when an aggregate function is applied .

Example 1:

consider a relation account with attributes

branch_name	Acc_no	balance
Mumbai	501	2000
Mumbai	604	4000
Pune	103	6000
Pune	203	4000

1.Find the sum of the balance of all customers

$g_{sum(balance)}$ (account)

Result:

Sum(balance)
16000

2.Find the sum of the balance of all customers and rename the relation by sum_balance

$g_{sum(balance)\ as\ sum\_balance}$(account)

Sum(balance)
16000

'as' is used for renaming the result

3.Find the sum of the credit balance of all customers and rename the relation by sum_balance and group by branch name

Branch_name	Sum_balance
Mumbai	6000
Pune	10000

branch_name $\mathcal{G}$ sum(balance) as sum_balance(account)

Relational Calculus:

Relational calculus is a non-procedural query language that tells the system what data to be retrieved but doesn't tell how to retrieve it. Relational Calculus exists in two forms:

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

7. Tuple Relational Calculus (TRC)

It is a non-procedural query language which is based on finding a number of tuple variables also known as range variable for which predicate holds true. It describes the desired information without giving a specific procedure for obtaining that information. The tuple relational calculus is specified to select the tuples in a relation. In TRC, filtering variable uses the tuples(row) of a relation. The result of the relation can have one or more tuples.

A query in the tuple relational calculus is expressed as

$$\{t \mid P(t)\}$$

Where **T** is the resulting tuples and **P(T)** is the condition used to fetch T,i.e. the set of tuples t for which predicate P is true.

We also use the notation

- $t[a]$ to indicate the value of tuple t on attribute a.
- $t \in rt \in r$ to show that tuple t is in relation r.
- $s[x]\ \theta\ u[y]$ where s and u are tuple variables, and x and y are attributes, and θ is a comparison operator ($<, \leq, =, \neq, >, \geq$).
- $s[x]\ \theta\ c$, where c is a constant in the domain of attribute x.
- we can use Existential ( ) and Universal Quantifiers ($\forall$).
- The logical connectives $\wedge$ (AND) and $\vee$ (OR) are allowed, as well as $\neg$ (negation).

A tuple variable is said to be a **free variable** unless it is quantified by a ∃ or a ∀. Then it is said to be a **bound variable**

Example Queries

Consider the tables/relations for bank management
Branch(bname,bcity,assets)
Customer(cname,cstreet,ccity)
Loan(loan_number,bname,amount)
Borrow(cname, loan_number)
Account(anumner,bname,balance)
Deposit(cname,anumber)

Example 1:
To find the branch-name, loan number, and amount for loans over $1200:
{t|t ∈ loan ∧ t[amount]>1200}

This gives us all attributes, but suppose we only want the loan number. (We would use **project** in the algebra.)

Example 2:
To find the loan number for each loan of amount for loans over $1200:
{t| ∃ l ∈ loan t[loan_number]=l[loan_number] ∧ l[amount]>1200}

In English, we may read this equation as ``the set of all tuples **t** such that there exists a tuple l in the relation *loan* for which the values of **t** and l for the *loan_number* attribute are equal, and the value of for the *amount* attribute is greater than 1200.".

Example 3:
Find names of all customers having a loan, an account, or both at the bank
{t| ∃ b ∈ borrow(t[cname]=b[cname] ∨ ∃ d ∈ deposit(t[cname]=d[cname])}

Example 4:
Find names of all customers who have **both** a loan and an account at the bank
Solution: simply change the ∨ connective in 1 to a ∧.
{t| ∃ b ∈ borrow(t[cname]=b[cname] ∧ ∃ d ∈ deposit(t[cname]=d[cname])}

Example 5:
Find the name of customers who have a loan at the SFU branch.
{t| ∃ b ∈ borrow(t[cname]=b[cname] ∧ ∃ l ∈ loan(l[loan_number]=b[loan_number]) ∧ l[bname="SFU"]}

8. Domain Relational Calculus (DRC)

The second form of relation is known as Domain relational calculus. In domain relational calculus, filtering variable uses the domain of attributes. Domain relational calculus uses the same operators as tuple calculus. It uses logical connectives ∧ (and), ∨ (or) and ¬ (not). It uses Existential (∃) and Universal Quantifiers (∀) to bind the variable. The QBE or Query by example is a query language related to domain relational calculus.

A query in the domain relational calculus is expressed as

{ <a1, a2, a3, ..., an> | P <a1, a2, a3, ... ,an>}

a1, a2 are attributes
P stands for formula built by inner attributes
An atom in the domain relational calculus is of the following forms

- <a1, a2, a3, ..., an> ∈ r where **r** is a relation on **n** attributes, and $a_{i,,}1<=i<=n$ are domain variables or constants.
- $x \ \theta \ y$, where **x** and **y** are domain variables, and **θ** is a comparison operator.
- $x \ \theta \ c$, where c is a constant.

Example Queries

Consider the tables/relations for bank management
Branch(bname,bcity,assets)
Customer(cname,cstreet,ccity)
Loan(loan_number,bname,amount)
Borrow(cname, loan_number)
Account(anumner,bname,balance)
Deposit(cname,anumber)

Example1:
Find branch name, loan number and amount for loans of over $1200.

{<l,b,a>|<l,b,a>∈loan ∧a>1200}

Example2:
Find all loan numbers who have a loan for an amount > than $1200.

{<l>|∃b,a∈|<l,b,a>∈loan ∧ a >1200}

Example3:
Find all customers having a loan, an account or both at the SFU branch.
{<c>|∃l<c,l>∈borrow|<l,b,a>∧∃b,a<l,b,a>∈loan ∧ b=”SFU”) ∨ ∃a(<c,a>∈account ∧b=”SFU”}

9. Difference between relational algebra and relational Calculus

Sr. No.	Key	Relational Algebra	Relational Calculus
1	Language Type	Relational Algebra is procedural query language.	Relational Calculus is a non-procedural or declarative query language.
2	Objective	Relational Algebra targets how to obtain the result.	Relational Calculus targets what result to obtain.
3	Order	Relational Algebra specifies the order in which operations are to be performed.	Relational Calculus specifies no such order of executions for its operations.

Sr. No.	Key	Relational Algebra	Relational Calculus
4	Dependency	Relational Algebra is domain independent.	Relational Calculus can be domain dependent.
5	Programming Language	Relational Algebra is close to programming language concepts.	Relational Calculus is not related to programming language concepts.