UNIT-2 DBMS (RCS-501)

Relational Model concepts :

1. Attribute : Each column in a table. Attributes are the properties which define a relation.

2. Tables : In the relational Model, the relations are saved in the table format. A table has two properties rows and columns. Rows represents records and columns represents attributes.

3. Tuple : It is single row of a table, which contains a single record.

4. Relation Schema : A relation schema represents the name of the relation with its attributes.

5. Degree : Total number of attributes which in the relation is called the degree of the relation.

6. Cardinality : Total number of rows present in the table is called the cardinality of the relation.

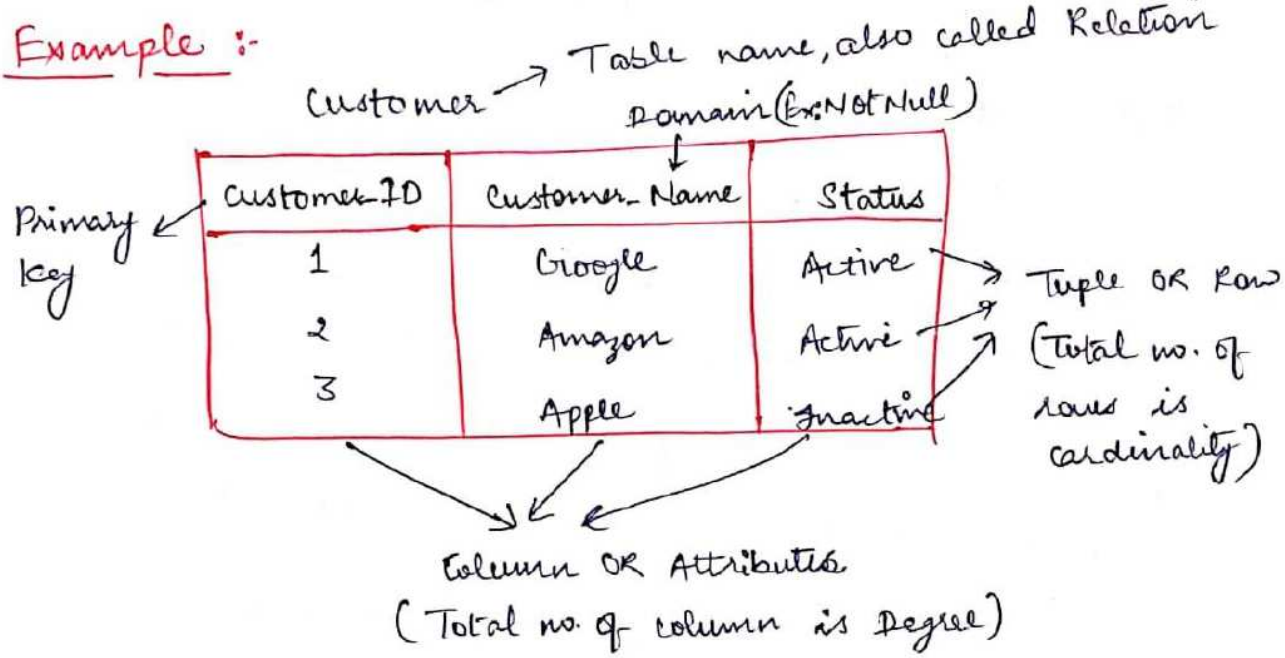
7. Column : The column represents the set of values for a specific attribute.

8. Relation Instance : Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.

9. Relation Key :- In the relation key, each row has one or more attribute. It can identify the row in the relation uniquely.

10. Attribute domain : Every attribute has some pre-defined value and scope which is known as attribute domain.

Example :-



Characteristics of Relations :

i) Name of the relation is distinct from all other relations.

ii) Each relation cell contains exactly one single value.

iii) Each attribute contains a distinct name.

iv) Attribute domain has no significance.

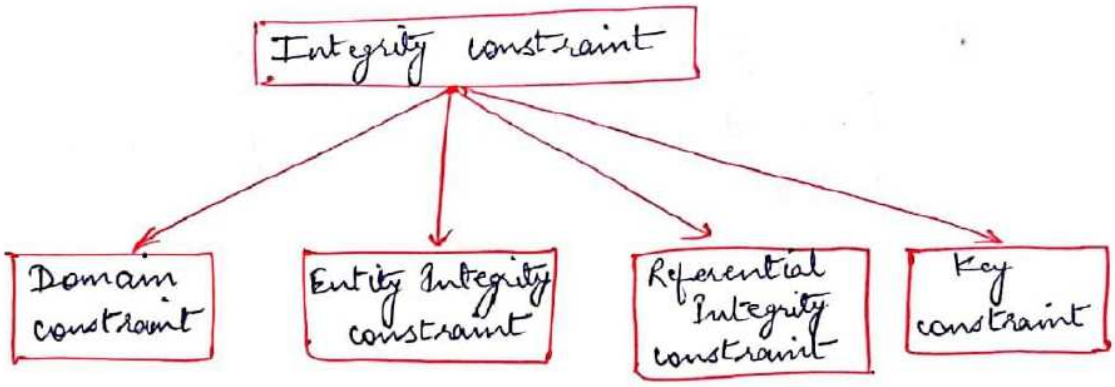
v) Tuple has no duplicate value.

vi) Order of tuple can have a different sequence.

Integrity constraints :

- Integrity constraints are a set of rules. It is used to maintain the quality of information.
- It ensure that the data insertion, updating and other processes have to be informed in such a way that data integrity is not affected. It is used to guard against accidental damage to the database.

Types of Integrity constraint :



1> Domain constraints :

Domain constraints can be defined as the definition of a valid set of values for an attribute. The value of the attribute must be available in the corresponding domain.

Example: Student

Stud_ID	Stud_Name	Age
1	X	24
2	Y	21
3	Z	A

→ It is not allowed because Age is an integer attribute.

ii> Entity integrity constraints:

Entity integrity constraint states that primary key value can't be null. If the primary key has a null value, then we can't identify those rows. A table can contain a null value other than the primary key field.

Example: Student

Stud_Id	Stud_Name	Age
1	x	24
2	y	21
	z	23

↓

It is not allowed as primary key can't contain a null value.

iii> Referential Integrity constraints:

A referential integrity constraint is specified between two tables. In Referential Integrity constraints, if a foreign key in first table refers to the primary key of second table, then every value of the foreign key in first table must be null or be available in second table.

Example:

Student :

Stud ID	Stud_Name	Course_Id → Foreign key
1	X	E11
4	Y	E02
3	Z	E12 → It is not allowed as it is not defined as a primary key of course table and in student table, course_Id is a foreign key defined.
2	W	E09

(Relationship)

Course :

Primary Key	Course_Id	Course_Name
	E11	commerce
	E02	Science
	E09	Arts

iv) Key constraints :-

Keys are the entity set that is used to identify an entity within its entity set uniquely. An entity set can have multiple keys, but out of which one key will be the primary key.

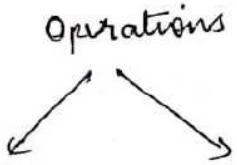
Example: Student

Stud_ID	Stud_Name	Age
1	X	21
2	Y	22
3	W	24
2 ←	Z	23

It is not allowed as all row must be unique.

Relational Algebra :

Relational Algebra is a collection of operations on relations. Each operation takes one or more relation as its operands and produces another relation as its result. It is a procedural query language. It gives a step by step process to obtain the result of the query.



* Unary operations : (operate on one relation)
ex. Select, Project etc.

* Binary operations : (operate on pair of relation)
ex. Union, intersect etc.

Types of Relational Operations are as follows :-

1> Select Operation :- It select tuples (rows) from a relation (table).

Denotion : It is denoted by Sigma (σ).

Syntax : $\sigma_{(\text{select condition})}(R)$

Example :

Student

Sid	S_Name
1	A
2	B
3	C

Query :

$\sigma_{s\_id=1}(\text{student})$

output :-

S-id	S_Name
1	A

* In Select Operation, all relational operators may be used.
$(=, \neq, <, \leq, \geq, >)$

Query: $\sigma_{S\_id > 1}(Student)$

Output:

S_id	S_Name
2	B
3	C

* condition may be combined using i) and ($\wedge$)
↓
both condition must be true

ii) or($\vee$) → either one condition true.

Example: $\sigma_{S\_id = 1 \text{ or } S\_id = 3}(Student)$

$\sigma_{S\_id = 1 \text{ and } S\_id = 2}(Student)$

Solved Question: Table is given as follows:

Student:

Stud_Id	Stud_Name	Faculty	Marks
1	A	F_1	90
2	B	B	80
3	C	F_1	85
4	D	B	75

i) Find the details of student whose name is same as of their faculty.

Ans: $\sigma_{Stud\_Name = Faculty}(Student)$

output:

Stud_Id	Stud_Name	Faculty	Marks
2	B	B	80

ii) Find the details of student whose marks are greater than or equals to 85.

Ans:

$$\sigma_{Marks \geq 85}(Student)$$

output:

Stud_Id	Stud_Name	Faculty	Marks
1	A	F1	90
3	C	F1	85

ii) Project Operation: It yields column (attribute) of a relation. Using project Operation, duplicate values are automatically removed.

Denotion: It is denoted by symbol (π).

Syntax: $\pi_{A_1, A_2, \ldots, A_n}(R)$

$A_1, A_2, \ldots, A_n$ → Attributes of a relation(R)

Example: Student:

Stud_Id	Stud_Name	class
1	A	11
2	B	12
3	C	11
4	B	12

query: $\pi_{Stud\_Name}(Student)$

output:

Stud_Name
A
B
C

Solved Question:

Query: $\pi_{Stud\_Name}\left[\sigma_{class=12}(Student)\right]$

Output:

Stud_Name
B

iii) Cartesian Product Operation :- It yields new relation which has a degree (no. of attributes) equal to the sum of the degrees of two relations operated upon. The no. of tuples is product of no. of tuples of two relations.

Denotion: It is denoted by symbol 'X'.

Syntax: $(R_1 \times R_2)$

Note:

	R_1	R_2	$(R_1 \times R_2)$
Attributes	x	y	(x+y) → no. of attributes
Tuples	n_1	n_2	$(n_1 \times n_2)$ → no. of tuples

Example:

R

A	B
α	11
β	21

S

C	D
P	103
Q	101

R X S →

4

A	B	C	D	
α	11	P	103	4
α	11	Q	101	
β	21	P	103	
β	21	Q	101	

iv) **Set Union Operation :** It produces a relation that contains tuples from both operand relations.
↳ must be union compatible.

Condition of Union - compatibility :

i) Relations 'R_1' and 'R_2' must be of same degree (no. of columns).

ii) Domain of i^{th} attribute of 'R_1' and 'R_2' must be same for all 'i'.

Denotion : It is denoted by symbol '$\cup$'.

Syntax : $[R_1 \cup R_2]$
tuples from either R_1 or R_2 or both.
(Duplicate tuples are removed)

Example :

Student 1

Stud_Id	Stud_Name
1	A
2	B

Student 2

Stud_Id	Stud_Name
3	C
1	A

Query : $\Pi_{Stud\_Id,\ Stud\_Name}(Student) \cup \Pi_{Stud\_Id,\ Stud\_Name}(Student)$

Output :

Stud_Id	Stud_Name
1	A
2	B
3	C

v> **Set Difference Operation :** It is used to find tuples that are in one relation but not in another relation.

Denotion: It is denoted by symbol '−'.

Syntax : $[R_1 - R_2]$

↳ tuples present in R_1 but not in R_2.

Example :- Find the name of students who are passed.

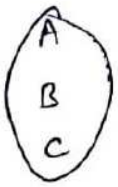
Student

Id	Name
1	A
2	B
3	C

Fail_Student

Id	Name
1	A
2	B

Query: $\pi_{name}(Student) - \pi_{name}(Fail\_Student)$



output :

Name
C

vi) **Set Intersection Operation :** It is used to find tuples that are common to the two operand relations.

Denotion : It is denoted by symbol '$\cap$'.

Syntax : $[R_1 \cap R_2]$

↳ tuples are present in both 'R_1' and 'R_2'.

Example : Find the names of students that are present in both R and S.

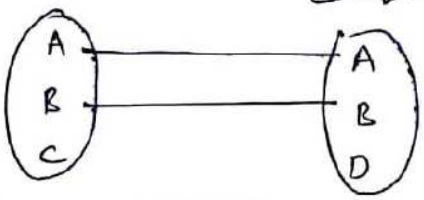
R

Sid	SName
1	A
2	B
3	C

S

Sid	Sname
4	A
5	B
6	D

Query: $\pi_{SName}(R) \cap \pi_{SName}(S)$



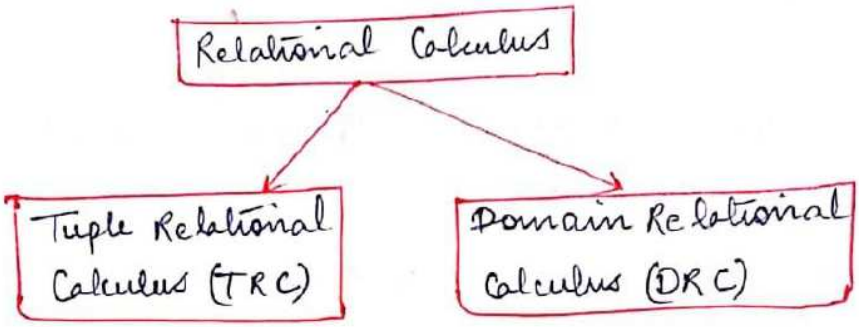
Output :

S_Name
A
B

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.

Types of Relational Calculus



i) Tuple Relational Calculus :-

The tuple relational calculus is specified to select the tuples in a relation. The result of the relation can have one or more tuples.

Notation :- $\{T \mid P(T)\}$ or $\{T \mid condition(T)\}$

where, T is the resulting tuples and
P(T) is the condition used to fetch T.

For Example : Table: Student

First_Name	Last_Name	Age
A	Q	30
B	Q	31
C	R	27
D	S	28

i> Query to display all the details of students where Last name is 'Q'

⇒ { T | Student(T) AND T. Last_Name = 'Q' }

Output :-

First_Name	Last_Name	Age
A	Q	30
B	Q	31

ii> Query to display the last name of those students where age is greater than 30

⇒ { T. Last_Name | Student(T) AND T.age > 30 }

Output :-

Last_Name
Q

ii> Domain Relational Calculus :-

In domain relational calculus, the records are filtered based on the domains.

Notation : $\{a_1, a_2, a_3, \ldots, a_n \mid P(a_1, a_2, a_3, \ldots, a_n)\}$

where $a_1, a_2, a_3, \ldots, a_n$ are attributes (columns) and P defines the formula including the condition for fetching the data.

For example : Table : Student

First_Name	Last_Name	Age
A	Q	30
B	Q	31
C	R	27
D	S	28

i) Query to find the first name and age of students where student age is greater than 27.

⇒ $\{ < \text{First\_Name}, \text{Age} > | \in \text{Student} \wedge . \text{Age} > 27 \}$

Output :-

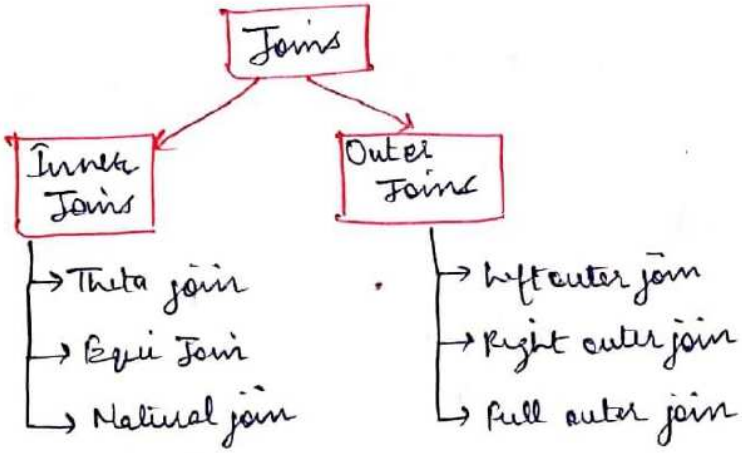
First_Name	Age
A	30
B	31
D	28

Note :- The symbols used for logical Operators are : $\wedge$ for AND, $\vee$ for OR and $\neg$ for NOT.

Joins in DBMS :

Join is a combination of a cartesian product followed by a selection process. A join operation pairs two tuples from different relations, if and only if a given join condition is satisfied.

Types of Joins :-



1) **Inner Joins :** These joins are the one that has the tuples that satisfy some conditions and rest are discarded.

2) **Theta Join :** Theta join combines tuples from different relations provided they satisfy the theta condition. The join condition is denoted by the symbol 'θ'.

Notation : $[R_1 \bowtie_\theta R_2]$

Note :- Theta join can use all kinds of comparison operators ($=, \neq, >, <, \geq, \leq$).

Example 1:

Student:

ID	Name	Std
1	A	11
2	B	12

Subjects:

class	subject
11	w
11	x
12	y
12	z

Student_Detail – Student $\bowtie_{Student.std = Subjct.class}$ Subject

Student_detail:

ID	Name	Std	class	subject
1	A	11	11	w
1	A	11	11	x
2	B	12	12	y
2	B	12	12	z

Note: R_1 and R_2 are relations having attributes $(A_1, A_2 \ldots A_n)$ and $(B_1, B_2, \ldots, B_n)$ such that the attributes don't have anything in common, i.e. $(R_1 \cap R_2 = \phi)$.

b) Equi join: When theta join uses only equality comparison operator, it is said to be equijoin. The above example corresponds to equijoin.

c) Natural join :-

- Natural join doesnot use any comparison operator.
- Natural join can be done if there is at least one common attribute that exists between two relations.
- The attributes must have the same name and domain.
- Natural join acts on those matching attributes where the values of attributes in both the relations are same.

Notation : $[R_1 \bowtie R_2]$

Example :

Student

ID	Name	class
1	A	10
4	D	12
2	B	11
3	C	12

Subjects

class	subject
10	X
12	Y
11	Z

output : (Student ⋈ subjects)

ID	Name	class	subject
1	A	10	X
4	D	12	Y
2	B	11	Z
3	C	12	Y

ii) Outer joins : These have all the tuples from either or both the relations.

a) left outer join :- All the tuples from the left relation 'R' are included in the resulting relation.

- If there are tuples in R without any matching tuple in the Right relation S, then the S-attributes of the resulting relation are made NULL.

Notation : $(R \bowtie S)$

Example :

R

Stud_Id	Stud_Name
1	A
4	B
2	C
3	D

S

Stud_Id	class
1	10
2	11
3	12
5	11

output : $(R \bowtie S)$.

Stud_Id	Stud_Name	class
1	A	10
4	B	NULL
2	C	11
3	D	12

b) Right outer join :-

- All the tuples from the right relation 'S', are included in the resulting relation.
- If there are tuples in S without any matching tuple in R, then the R-attributes of resulting relation are made NULL

Notation : $(R \bowtie S)$

Example : From the given relations 'R' and 'S',

output : $(R \bowtie S)$

Stud_Id	Stud_Name	class
1	A	10
2	C	11
3	D	12
5	NULL	11

c) Full outer join :-

- All the tuples from both participating relations are included in the resulting relation.
- If there are no matching tuples for both relations, their respective unmatched attributes are made NULL.

Notation : $(R \bowtie S)$

Example:

output: From the given relations 'R' and 'S'.

(R ⋈ S)

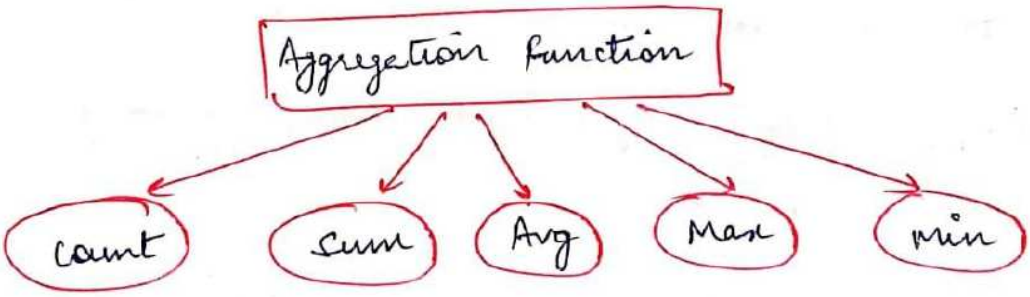
Stud_Id	Stud_Name	Class
1	A	10
4	B	NULL
2	C	11
3	D	12
5	NULL	11

Aggregate functions :

- Aggregate functions in DBMS take multiple rows from the table and return a value according to the query.
- It is also used to summarize the data.
- All the aggregate functions are used in select statement.

Syntax :- Select < Function Name > (< Parameter >) from < Table Name >

Types of Aggregate functions :



i> COUNT Function : The count function returns the number of rows in the result. It does not count the null values.

Example : Write a query to return number of rows where salary > 20000.

⇒ Select COUNT(*) from Employee where salary > 20000 ;

Types:

1. COUNT(*): counts all the number of rows of the table including null.

2. COUNT (Column_Name): counts number of non-null values in column.

3. COUNT (DISTINCT Column_Name): count number of distinct values in a column.

ii) SUM function: This function sums up the values in the column supplied as a parameter. It works on numeric fields only.

Example: write a query to get the total salary of employees.

⇒ Select SUM (salary) from Employee;

iii) AVG function: This function returns the average value of the numeric column that is supplied as a parameter.

Example: write a query to select average salary from employee table.

⇒ Select AVG (salary) from Employee;

iv) MAX Function : The MAX function is used to find maximum value in the column that is supplied as a parameter.

Example: write a query to find the maximum salary in employe table.

⇒ Select MAX (Salary) from Employe;

v) MIN function : The MIN function is used to find minimum value in the column that is supplied as a parameter.

Example : write a query to find the minimum salary in employe table.

⇒ Select MIN (Salary) from Employee;

* Difference between SQL and PL/SQL :-

Basis	SQL	PL/SQL
i) Basic	i) In SQL, we can execute a single query or a command at a time	i) In PL/SQL, we can execute a block of code at a time.
ii) Full Form	ii) Structured Query Language	ii) Procedural Language, extension of SQL.
iii) Purpose	iii) It is like a source of data that is to be displayed, means data oriented language.	iii) It is language that creates an application that display's the data acquired by SQL, means application oriented language.
iv) Writes	iv) In SQL, we can write queries and command using DDL, DML statements.	iv) In PL/SQL, we can write block of code that has procedures, functions, Packages, or variables.
v) Use	v) Using SQL, we can retrieve, modify, add, delete, or manipulate the data in the database.	v) Using PL/SQL, we can create applications or server pages that displays the information obtained from SQL in a proper format.