QUERYING RELATIONAL DATA

Relational data in the form of rows and columns.Querying relational data means a relational database query is question about the data and the answer consists of a new relation as a result. The data may be contained in one or two or more tables in a relational database. The relational query must specify the tables required and what the condition is that links them.

SQL stands for Structured Query Language, as it is the special purpose domain-specific language for querying data in Relational Database Management System (RDBMS). It is used to extract the data from the relations. e.g.; SELECT. A generic query to retrieve data from a relational database is:

SELECT [**DISTINCT**] Attribute_List **FROM** R1,R2….RM
[**WHERE** condition]
[**GROUP BY** (Attributes)[**HAVING** condition]]
[**ORDER BY**(Attributes)[**DESC**]];

Part of the query that is line number1 is compulsory if you want to retrieve from a relational database. The statements written inside [] are optional.

Ex:

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

Case 1: If we want to retrieve attributes **ROLL_NO** and **NAME** of all students, the query will be:

SELECT ROLL_NO, NAME **FROM** STUDENT;

ROLL_NO	NAME
1	RAM
2	RAMESH
3	SUJIT
4	SURESH

Case 2: If we want to retrieve **ROLL_NO** and **NAME** of the students whose **ROLL_NO** is greater than 2, the query will be:

SELECT ROLL_NO, NAME **FROM** STUDENT
WHERE ROLL_NO>2;

ROLL_NO	NAME
3	SUJIT
4	SURESH

CASE 3: If we want to retrieve all attributes of students, we can write * in place of writing all attributes as:

SELECT * FROM STUDENT
WHERE ROLL_NO>2;

ROLL_NO	NAME	ADDRESS	PHONE	AGE
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	9156768971	18

CASE 4: If we want to represent the relation in ascending order by **AGE**, we can use ORDER BY clause as:

SELECT * FROM STUDENT **ORDER BY** AGE;

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

CASE 5: If we want to retrieve distinct values of an attribute or group of attribute, DISTINCT is used as in:

SELECT DISTINCT ADDRESS **FROM** STUDENT;

ADDRESS
DELHI
GURGAON
ROHTAK

If DISTINCT is not used, DELHI will be repeated twice in result set.

RELATIONAL ALGEBRA

Relational algebra introduced in the year 1971 by E.F.CODD. Relational algebra is a procedural query language. Generally query language is used to retrieve data from the database or perform various operations such as insert,update,delete on the data. Procedural means step by step process to obtain the result of the query.Relational algebra is a procedural query language, it means that it tells what data to be retrieved and how to be retrieved. Relational Algebra is procedural query language, which takes Relation as input and generate relation as output. Relational algebra mainly provides theoretical foundation for relational databases and SQL. It uses operators to perform queries.

Operators in Relational Algebra:-

An operator can be categorized into 3 types

1)Unary operators

2)Set operators

3)Binary operators

Unary Relational Operations

- SELECT (symbol: σ)
- PROJECT (symbol: π)
- RENAME (symbol: ρ)

Relational Algebra Operations From Set Theory

- UNION (υ)
- INTERSECTION (∩)
- SET DIFFERENCE (-)
- CARTESIAN PRODUCT (x)

Binary Relational Operations

- JOIN
- DIVISION

1)Select Operator (σ):

It is unary operator. Select Operator is denoted by sigma (σ) and it is used to find or retrieve the tuples (or rows) in a relation (or table) which satisfy the given condition.

Notation: $\sigma_p(r)$

Where, σ represents the Select Predicate,

r is the name of relation(table name in which you want to look for data),

p is the prepositional logic, where we specify the conditions that must be satisfied by the data. In prepositional logic, logical connectives and relational operators are used.

Ex: Student

Rollnum	Name	Marks	Age	Gender
1	John	50	25	M
2	Deepika	75	30	F
3	Ramya	60	29	F
4	Kiran	50	25	M
5	Surya	80	40	M
6	Rian	90	35	M

Q)Find out the students whose age is above 35.

$\sigma_{age > 35}$ (Student)

Output:

Rollnum	Name	Marks	Age	Gender
5	Surya	80	40	M

Q)Find out the Male students whose marks is above 70.

$\sigma_{gender = 'M' \text{ and } Marks>70}$ (Student)

Output:

Rollnum	Name	Marks	Age	Gender
5	Surya	80	40	M
6	Rian	90	35	M

Projection or Project operator(π):

It is also an unary operator. Project operator is denoted by $\prod$ symbol and it is used to select desired columns (or attributes) from a table (or relation) and rest of attributes are dicarded.This helps to eliminate Duplicate tuples from the resulting relation are eliminated automatically.

Notation: $\prod_{A1, A2...}(r)$

where A1, A2 etc are attribute names(column names).

r is a relation

Ex:To fetch the names of students

$\prod_{Name}(Student)$

Output:

Name
John
Deepika
Ramya
Kiran
Surya
Rian

Ex: what are the types of genders in student table?

$\prod_{Gender}(Student)$

Output:

Gender
M
F

EX: Display the details of student name and age.

$\prod_{Name,Age}(Student)$

Output:

Name	Age
John	25
Deepika	30
Ramya	29
Kiran	25
Surya	40
Rian	35

Ex: Display the name of the student whose age is more than 30.

$\prod_{Name}(\sigma_{age > 30}(Student))$

Output:

Name
Surya
Rian

Rename Operation (ρ):

Rename is an unary operator.It is used to rename the output relation or an attribute of a relation. It is denoted by **rho** (ρ).

For Relation

Notation − ρ_x (E)

Where E is the expression with different relational algebra operations and x is, the name given to their result.

For attribute

Notation: ρ (a/b)R

It will rename the attribute 'b' of relation by 'a'.

Ex: To change the attribute rollnum to sid of student table

$\rho_{sid/Rollnum}$ (Student)

Ex:

SNO	Sname	Marks

SNO	Grade

Display the student names who are having A grade

$\prod_{Sname}$ (ρ **Studentgrade**($\sigma_{Grade='A'}$ (Student X Grade))

Relational Algebra Operations From Set Theory or Set Operators:

The set operations in the algebra are

Union operation (U):

Union operator is denoted by U symbol and it is used to select all the rows (tuples) from two tables (relations).It eliminates duplicates.

Syntax: A ∪ B

Where A,B are relations

For a union operation to be valid, the following conditions must hold

- A and B must be the same number of attributes.
- Attribute domains need to be compatible and also the attributes of A and B must occur in the same order.

Example:

STUDENT

Rollnum	Name
1	A
2	B
3	C
4	D

EMPLOYEE

Empno	Name
2	B
8	G
9	H
3	C

STUDENT ∪ EMPLOYEE

Output:

Rollnum	Name
1	A
2	B
3	C
4	D
8	G
9	H

Intersection Operator (∩):

Intersection operator is denoted by ∩ symbol and it is used to select common rows (tuples) from two tables (relations).

Syntax: A ∩ B

Where A,B are relations

Ex: STUDENT ∩ EMPLOYEE

Output:

Rollnum	Name
2	B
3	C

Ex: $\prod_{Name}(STUDENT) \cap \prod_{Name}(EMPLOYEE)$

Output:

Name
B
C

Set Difference (-):

Set Difference operator is denoted by '–'.This operation is used to find data present in one relation and not present in the second relation.

Syntax: A - B

Where A,B are relations

Ex: STUDENT – EMPLOYEE

Rollnum	Name
1	A
4	D

Cartesian product (X):

Cartesian Product is denoted by X symbol. It is also called Cross Product. The Cartesian product would combine each row in one table with all the rows in the other table. Generally, a cartesian product is never a meaningful operation when it performs alone. However, it becomes meaningful when it is followed by other operations.Generally it is followed by a select operation.

Syntax: A X B

Where A,B are relations

EX: Example

Consider R1 table −

RegNo	Branch	Section
1	CSE	A
2	ECE	B
3	CIVIL	A
4	IT	B

Table R2

Name	RollNo
Bhanu	2
Priya	4

R1 X R2

RegNo	Branch	Section	Name	RollNo
1	CSE	A	Bhanu	2
1	CSE	A	Priya	4
2	ECE	B	Bhanu	2

RegNo	Branch	Section	Name	RollNo
2	ECE	B	Priya	4
3	CIVIL	A	Bhanu	2
3	CIVIL	A	Priya	4
4	IT	B	Bhanu	2
4	IT	B	Priya	4

Characteristics:

1)Consider two relations R1 and R2 have a & b attributes respectively then the resulting relation will have a+b attributes from both the input relations.

2) Consider two relations R1 and R2 have n1 & n2 tuples respectively then the resulting relation will have n1*n2 tuples.

	R1	R2	R1 X R2
Attributes	a	b	(a+b)
Tuples	n1	n2	(n1*n2)

Division Operator(÷,/):

Division operator is a derived operator,not supported as a primitive operator. Derived operators are those operators which can be derived from basic operators.Division can be expressed in terms of cross product,set difference,projection.It is best suited to queries that include the keyword 'ALL' or 'EVERY' like for all,at all,in all,at every,for every.Division operator returns tuples of one relation which are associated with all the tuples of another relation.That means, R1 ÷ R2 = tuples of R1 associated with all tuples of R2.

Ex: Find the person that has account in all the banks of a particular city.

Find employees who works on all projects of company.

Find students who have registered for every course.

Notation: A÷ B **(or)** A/B

Division operator A÷B can be applied if and only if:
1)Attributes of B is proper subset of Attributes of A.
2)The relation returned by division operator will have attributes = (All attributes of A – All Attributes of B)
3)The relation returned by division operator will return those tuples from relation A which are associated to every B's tuple.

EX: Student

Name	Sub_failed
A	DBMS
A	JAVA
B	DBMS
C	JAVA

Subject

SUBJECT
DBMS
JAVA

Q)Find the names of students who failed in all the subjects.

$\prod_{Name}$(Student ÷ Subject)

Output:

Name
A

Ex:

STUDENT_SPORTS

ROLL_NO	SPORTS

ALL_SPORTS

1	Badminton
2	Cricket
2	Badminton
4	Badminton

SPORTS
Badminton
Cricket

To apply division operator as

STUDENT_SPORTS÷ ALL_SPORTS

Output:

ROLL_NO
2

- ✓ 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
- ✓ 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.

JOINS:

Join is an derived/additional operator.Join is a combination of a cartesian product followed by a selection process.

Join=Cartesian product+Selection

A Join operation combines related tuples from different relations, if and only if a given join condition is satisfied. It is denoted by ⋈.

Syntax: A ⋈ B= σ (AXB)

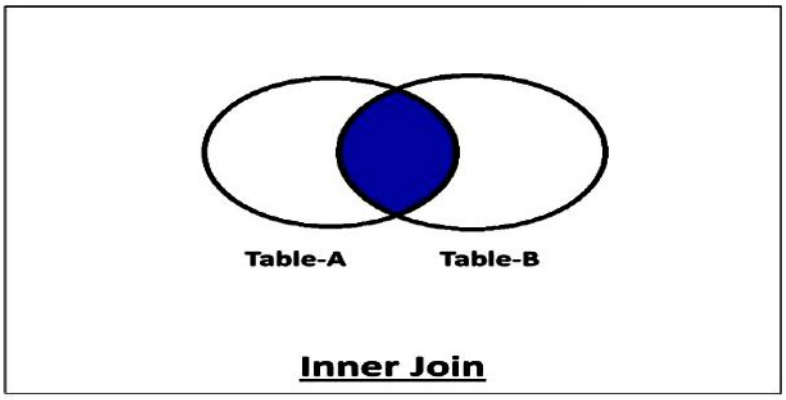
Types of JOIN:

A Join can be broadly divided into two types:

1. Inner Join
2. Outer Join

Inner Join

Inner Join is a join that can be used to return all the tuples that have matching criteria satisfied in both the tables,while the rest are excluded. Inner Join can be depicted using the below diagram.



The inner join can be further divided into the following types:

1. Theta/Conditional Join
2. Equi Join
3. Natural Join.

Theta/Conditional Join:

Theta Join is used to join two or more tables based on some conditions. The condition can be on any attributes of the tables performing Theta join. Any comparison operator can be used in the condition. The join condition is denoted by the symbol **θ**.

Notation: $A \bowtie_\theta B$
Where A,B are relations

θ is the condition for join.

Ex:Consider two relations Sailors and Reserves

Sailors

Sid	Name	Rating	Age
22	Rian	7	45
31	John	8	55
58	Kiran	10	35

Reserves

Sid	Bid	Day
22	101	10/10/96
58	103	11/12/96

Ex: Sailors $\bowtie_{\text{sailors.sid<reserves.sid}}$ Reserves =

$\sigma_{\text{sailors.sid<reserves.sid}}$(SailorsXReserves)

Output:

Sailors.sid	Name	Rating	Age	Reserves.Sid	Bid	Day
22	Rian	7	45	58	103	11/12/96
31	John	8	55	58	103	11/12/96

2)Equi Join: Equi Join is an inner join that uses the equivalence condition for fetching the values of two tables. That means, When a theta join uses only equality **comparison operator**, it becomes a equi join.

Notation: $A \bowtie_{A.a1 = B.b1 \ldots\ldots A.an = B.bn} B$
Where A,B are relations

Ex: Sailors $\bowtie_{sailors.sid=reserves.sid}$ Reserves

Output:

Sailors.sid	Name	Rating	Age	Reserves.sid	Bid	Day
22	Rian	7	45	22	101	10/10/96
58	Kiran	10	35	58	103	11/12/96

Natural Join: Natural join can only be performed if there is a common attribute (column) between the relations. The name and type of the attribute must be same. Natural join does not use any comparison operator for join condition. In the result of the Natural Join the common attribute only appears once. It also removes the duplicate attribute from the results. It is a special case of equijoin in which equality condition have to write explicitly, While applying natural join on two relations, there is no need to write equality condition explicitly.

Notation: $A \bowtie B$
Where A,B are relations

The natural join of two relations can be obtained by applying a projection operation to equi join of two relations.

Natural Join=Cartesian product+Selection+Projection

Ex: Sailors $\bowtie$ Reserves

sid	Name	Rating	Age	Bid	Day
22	Rian	7	45	101	10/10/96

58	Kiran	10	35	103	11/12/96

Outer Join:

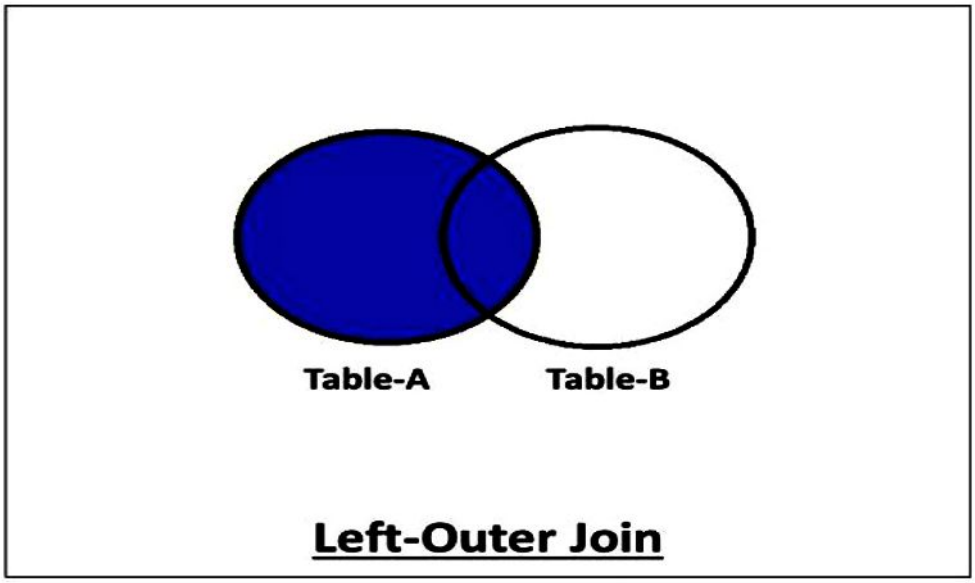
The outer join operation is an extension of the join operation. In an outer join, along with tuples that satisfy the matching criteria, we also include some or all tuples that do not match the criteria.

Outer join = Natural join + Extra information(from left table or right table or both the tables)

The outer join can be further divided into three types:

1. **Left-Outer Join**
2. **Right-Outer Join**
3. **Full-Outer Join**

1. Left-Outer Join: The Left-Outer Join is an outer join that returns all the tuples of the left table, and the values of the right table that has matching values in the left table. If there is no matching result in the right table, it will return null values in that fields. The Left-Outer Join can be depicted using the below diagram.



Symbol: ⟕

Notation: $A \text{ ⟕ } B$

Where A,B are relations

Ex: Courses

CID	Course
100	Database
101	Mechanics
102	Electronics

Hod

CID	Name
100	Balaji
102	Kusuma
104	Prakash

Ex: Courses ⋈ Hod

CID	Course	Name
100	Database	Balaji
101	Mechanics	NULL
102	Electronics	Kusuma

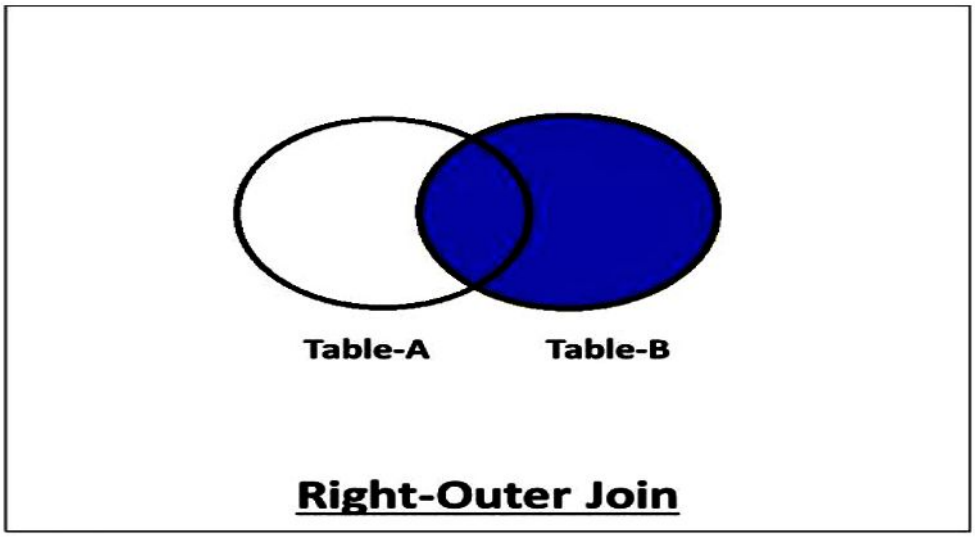
2.Right Outer Join:

In the right outer join, operation allows keeping all tuple in the right relation. However, if there is no matching tuple is found in the left relation, then the attributes of the left relation in the join result are filled with null values.

Symbol: ⟖

Notation: A ⟖ B

Where A,B are relations



Ex: Courses ⋈ Hod

CID	Course	Name
100	Database	Balaji
102	Electronics	Kusuma
104	NULL	Prakash

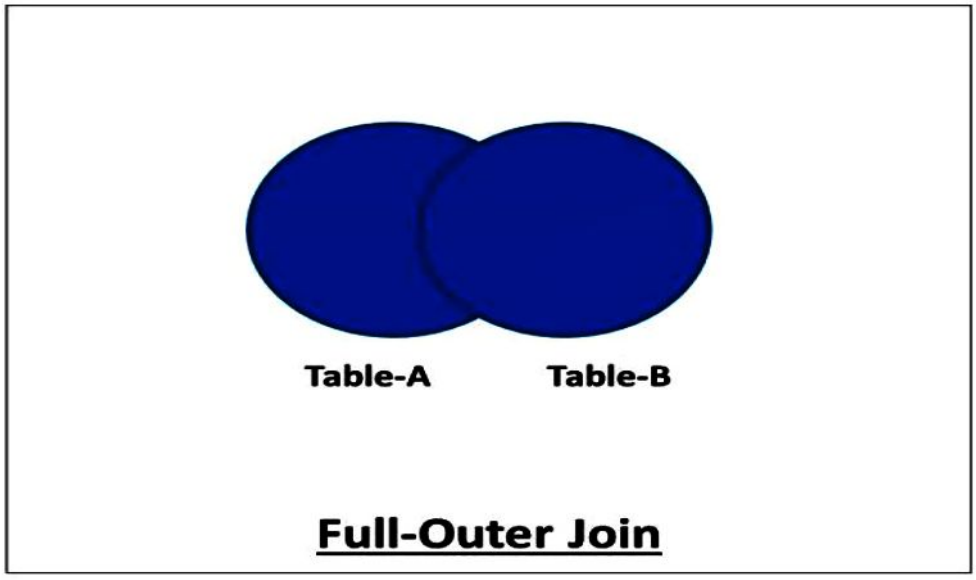
3. Full Outer Join:

In a full outer join, all tuples from both relations are included in the result, irrespective of the matching condition.That means, the tuples of one relation which do not satisfy join condition will have values as NULL for attributes of another relation and vice versa.

Symbol: ⋈

Notation: A⋈B

Where A,B are relations



Ex: Courses ⋈ Hod

CID	Course	Name
100	Database	Balaji
101	Mechanics	NULL
102	Electronics	Kusuma
104	NULL	Prakash

INTEGRITY CONSTRAINTS OVER RELATIONS

Constraints means rules.Constraints may apply to each attribute or they may apply to relationships between tables.Database integrity means completeness,Correctness and consistency of data.Integrity constraints(IC's) are set of rules which is used to maintain the quality of information.An integrity constraint is a condition specified on a database schema and restricts the data that can be stored in an instance of the database.If a database instance satisfies all the IC's specified on the database,it is called as legal instance.

DBA defines the database schema and specifies integrity constraint that must hold on any instance of relations.

Types of integrity constraints:

There are different types of integrity constraints.

1)Key Constraints (or) Uniqueness Constraints

2)Foreign key Constraints (or) Referential integrity constraint

3)General constraints.

4)Domain Constraints

5)Entity integrity Constraints

1)Key Constraints:

A Key Constraint is a statement that a certain minimal subset of the fields of a relation is a unique identifier for tuple.A set of fields that uniquely identifies a tuple according to Key Constraint is called a candidate key for the relation. A relation can have multiple keys or candidate keys(minimal superkey), out of which we choose one of the keys as primary key, we don't have any restriction on choosing the primary key out of candidate keys, but it is suggested to go with the candidate key with less number of attributes. Null values are not allowed in the primary key, hence Not Null constraint is also

a part of key constraint.So Primary key constraint is a combination of NOT NULL and UNIQUE.A Primary key column will not accept null and duplicate values.A relation can have only one primary key.

Example: Student Relation

ID	NAME	SEMESTER	AGE
100	Naren	4	27
101	Lalit	6	28
102	Shivanshu	3	22
103	Navdeep	5	29
102	Karthik	7	25

All row ID must be unique hence 102 is not allowed.

2)Foreign key Constraints (or) Referential integrity constraint:

Referential integrity constraint in DBMS are based on the concept of Foreign keys. A referential integrity constraint is specified between two tables. Referential integrity constraint is enforced when a foreign key references the primary key of a table. In the Referential integrity constraints, if a foreign key in Table 2 refers to the Primary Key of Table 1, then either every value of the Foreign Key in Table 2 must be be available in primary key value of Table 1 or it must be NULL.That means, It states that if a foreign key exists in a relation then either the foreign key value must match a primary key value of some tuple in its home relation or the foreign key value must be null.The foreign key,primary key datatypes must be compatible but having different column names.

The rules are:

1. You can't delete a record from a primary table if matching records exist in a related table.

corresponding domain or it is not of the appropriate data type.That means It specifies that the value taken by the attribute must be the atomic value from its domain.

Example: Consider a Student's table having STU_ID, NAME,AGE of students.

STU_ID	Name	Age
S001	Akshay	20
S002	Abhishek	21
S003	Shashank	20
S004	Rahul	A

Here, value **'A'** is not allowed since only integer values can be taken by the age attribute.

5)Entity integrity Constraints:

The entity integrity constraint states that primary key value can't be null.This is because the primary key value is used to identify individual rows in relation and 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.

Ex: CREATE TABLE STUDENT(

SNO NUMBER(3),

NAME VARCHAR2(30),

MARKS NUMBER(3) CHECK(MARKS BETWEEN 0 and 100),

AGE NUMBER(3) CHECK(AGE>16));

Then insert the rows into the table

SNO	NAME	MARKS	AGE	
1	KIRAN	85	24	
2	RAVI	-50	27	Not allowed
3	RAJESH	75	14	Not allowed

The row with sno=2 is not allowed because marks less than 0 and row with sno=3 is not allowed because age is less than 16.

Assertions:

When a constraint involves 2 (or) more tables, the table constraint mechanism is sometimes hard and results may not come as expected. To cover such situation SQL supports the creation of assertions that are constraints not associated with only one table. Assertions involve several tables and are checked whenever any one of these tables is modified.

Syntax –

CREATE ASSERTION [assertion_name]

CHECK ([condition]);

To drop assertion

DROP ASSERTION <assertion_name>;

2. You can't change a primary key value in the primary table if that record has related records.
3. You can't enter a value in the foreign key field of the related table that doesn't exist in the primary key of the primary table.
4. However, you can enter a Null value in the foreign key, specifying that the records are unrelated.

Example:

Consider an Employee and a Department table where Dept_ID acts as a foreign key between the two tables

Employees Table

EID	Name	Salary	Dept_ID
1101	Jackson	40000	3
1102	Harry	60000	2
1103	Steve	80000	4
1104	Ash	1800000	3
1105	James	36000	1

Department Table

Dept_ID	Dept_Name
1	Sales
2	HR
3	Technical

In the above example, Dept_ID acts as a foreign key in the Employees table and a primary key in the Department table. Row having *DeptID=4* violates the referential integrity constraint since DeptID 4 is not defined as a primary key column in the Departments table.

4)Domain Constraints:

Domain integrity means the definition of a valid set of values for an attribute. You define data type, length or size, is null value allowed , is the value unique or not for an attribute ,the default value, the range (values in between) and/or specific values for the attribute. Domain constraints can be violated if an attribute value is not appearing in the

Example:

EMPLOYEE

EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can't contain a NULL value

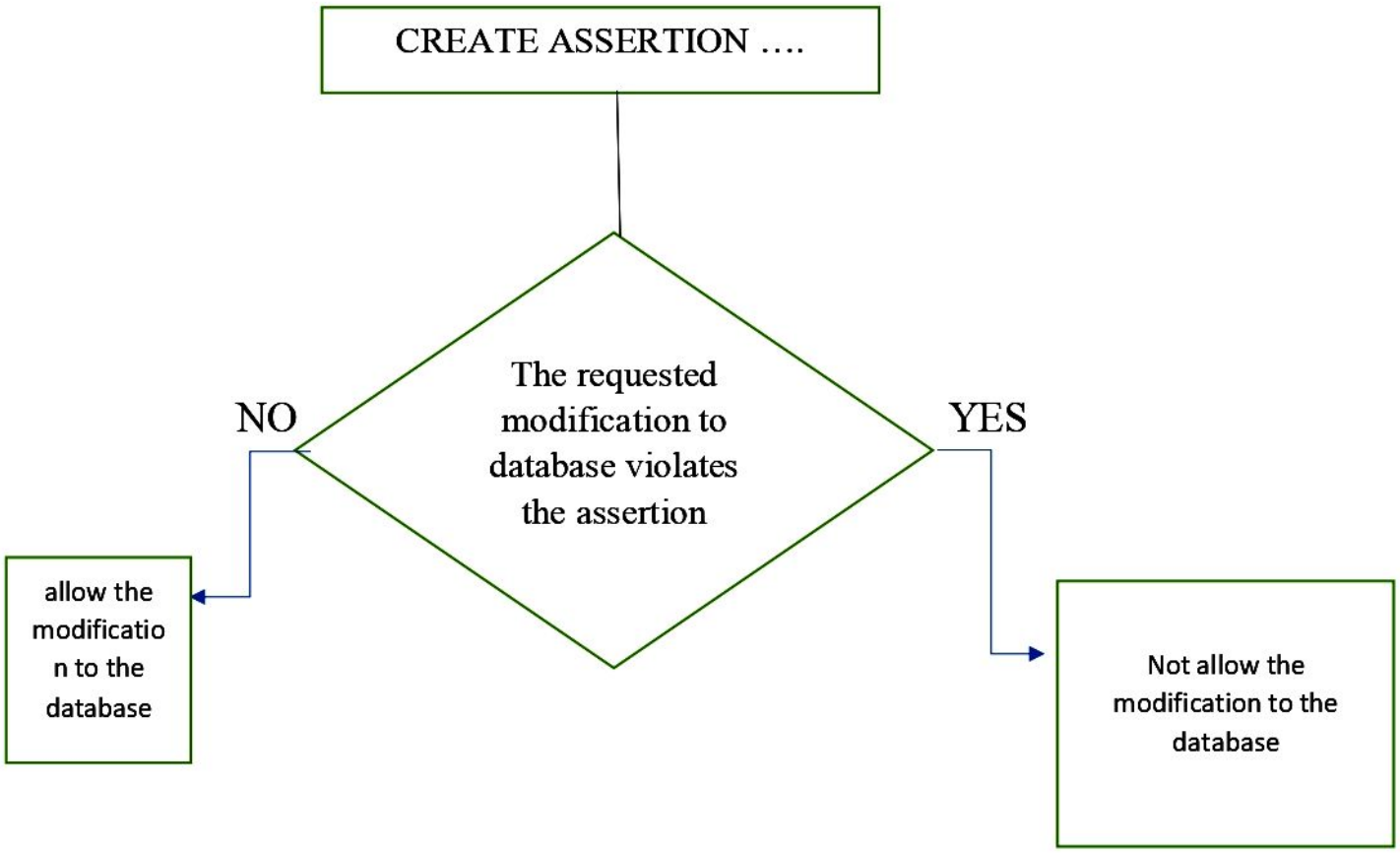
3)General constraints:

Primary key,foreign key constraints are considered to be a fundamental part of the relational data model.Current relational database systems support such general constraints in the form of Table Constraints and Assertions.

Table Constraints: Table Constraints are associated with a single table and are checked whenever table is modified.Table Constraints which have the form CHECK conditional_expression.

Syntax:

```
CREATE TABLE table_name(
                    Column1 datatype,
                    ----------
                    ColumnN datatype,
          CONSTRAINT constraint_name CHECK(Expression) );
```



Ex: CREATE ASSERTION smallclub

CHECK((select count(s.sid) from sailors s) +

(select count(b.bid)from boats b)<100));

DESTROYING/ALTERING VIEWS

Alter or Updating View:

There are certain conditions needed to be satisfied to update a view. If any one of these conditions is **not** met, then we will not be allowed to update the view.

1. The SELECT statement which is used to create the view should not include GROUP BY clause or ORDER BY clause.
2. The SELECT statement should not have the DISTINCT keyword.
3. The View should have all NOT NULL values.
4. The view should not be created using nested queries or complex queries.
5. If the view is created using multiple tables then we will not be allowed to update the view.

Suppose we want to add more columns to the created view so we will have to update the view.

For updating the views we can use CREATE OR REPLACE VIEW statement, new columns will added or remove columns from the view.

Syntax:

CREATE OR REPLACE VIEW ViewName AS

SELECT column1,coulmn2,..

FROM TableName

WHERE condition;

For Updating a view CREATE OR REPLACE statement is used. Here, viewName is the Name of the view we want to update,

Syntax:

DELETE FROM ViewName

WHERE condition;

Here, **viewName** is the view from which data has been deleted,

condition is the Condition by which we select rows to be deleted.

Destroying a View :

The DROP statement completely deletes the structure of the view.
Syntax:
DROP VIEW Viewname;

Here ViewName is the name of the view to be deleted.

Example:

Let's delete one of our created views, say EmpView1.

DROP VIEW EmpView1;

Let's use the SELECT statement on Deleted View.

SELECT * from EmpView1;

Output: Now SQL Editor will give us an error saying the table does not exist as the table is now been deleted.

ORA-00942: table or view does not exist

tableName is the Name of the table and condition is the Condition by which we select rows.

Example 1:

let's take the above created simple view EmpView1 and we want to one more column of salary to our EmpView1 from Employee Table.

CREATE OR REPLACE VIEW EmpView1 AS

SELECT Empno, Ename,sal

FROM Emp;

Now to see the data in the EmpView1 view

SELECT * from EmpView1;

Output:

Inserting a row in a view

Inserting a row in the view takes the same syntax as we use to insert a row in a simple table.

Syntax:

INSERT INTO ViewName(column1, column2,..)

VALUES(value1, value2,..);

Here, **viewName** is the view in which we have to insert data and we add values according to the columns in the view table.

Ex:

Deleting a row in a view

Deleting a row in the view takes the same syntax as we use to delete a row in a simple table.

ENFORCING INTEGRITY CONSTRAINTS

Data integrity means correctness , completeness and consistency of data within a database.To enforce data integrity, you can **constrain or restrict the data values that users can insert, delete, or update in the database**. Both database designers and database developers are responsible for implementing data integrity of database.Integrity constraints are specified when relation is created and are enforced when relation is modified.

Types of Data Integrity

There are four types of data integrity:

1. Row integrity
2. Column integrity
3. Referential integrity
4. User-defined integrity

Row integrity

Row integrity refers to the requirement that all rows in a table must have a unique identifier that can be used to tell apart each record. This unique identifier is normally known as Primary Key of the table. A Primary Key can be formed by a single column or a combination of multiple columns.

Column integrity

Column integrity refers to the requirement that data stored in a column must adhere to the same format and definition. This includes data type, data length, default value of data, range of possible values, whether duplicate values are allowed, or whether null values are allowed.

Referential integrity

Referential integrity is defined at the database design time and enforced by creating table relationships between tables. After the referential relationship is set up, database engine will follow the two rules stated to guarantee data integrity. It will raise errors if the rules are violated.

User-defined integrity

Some applications have complex business logic that can't be enforced by defining criteria in the three data integrity types we have discussed so far (row integrity, column integrity, and referential integrity). In this circumstance, we need to implement our own code logic to make sure data is saved accurately and consistently across all business domains. The code logic can be implemented by using database triggers, stored procedures or functions, or by using tools external to the database engine such as embedding non SQL languages (like VBScript or C# in SQL Server) in the database, or by using scripting or programming languages in the middle-tier or front-tier of the application.

Data integrity is enforced by database constraints

Database Constraints are declarative integrity rules of defining table structures. They include the following 7 constraint types:

1.Data type constraint:

Data type constraint defines the type of value a column should contain, including numeric, date and time, character, etc.

2.Nullability constraint:

Nullability constraint defines whether a column is allowed to store NULL values.

The NOT NULL constraint prevents the column from having a NULL value as it specifies that a NULL value is not allowed.

NULL constraint represents unknown data which is not the same as no data or empty data.

3.Default constraint:

Default constraint specifies a default value for a column if no value is supplied when adding a record to a table.

4.Primary key constraint:

A primary key is a column (or multiple columns) in a table that can be used to uniquely identify each row in the table. The primary key column cannot contain NULL values. A UNIQUE constraint is automatically generated for a primary key column.

One table can only have one primary key, which may consist of single or multiple columns. If the primary key contains multiple columns, it's called a composite primary key, and the combination of the multiple columns must contain distinct values.

The primary key can be either a sequentially incremented integer number or a natural selection of data that represents what is happening in the real world (e.g., Social Security Number).

5.Foreign key constraint:

Foreign key constraint defines how referential integrity is enforced between two tables. It uses a foreign key column in a table to link to a primary key column in another table.

The table with the foreign key can be called child table, referencing table, or foreign key table. The table with the primary key can be called parent table, referenced table, or primary key table.

Database engines use FOREIGN KEY constraint to enforce data in sync between the parent table and the child table. You can't delete a row in the parent table when the value of the primary key still exists in the foreign key column in the child table.

6.Unique constraint:

Sometimes the data in a column must be unique even though the column does not act as PRIMARY KEY of the table. For example, CategoryName column is unique in categories table, but CategoryName is not a primary key of the table. In this case, we create UNIQUE constraint which defines that the values in a column or columns must be unique, and no duplicates can be

stored at any time. You can have many UNIQUE constraints per table.

7.Check constraint:

Check constraint enforces that values in a column must satisfy a specific condition. In the background, the database engine uses a validation rule to check data quality when they are entered into the column. This rule is defined by the user when designing the column in a table. Not every database engine supports check constraints.

Data integrity type	Enforced by database constraint
Row integrity	• Primary key constraint • Unique constraint
Column integrity	• Foreign key constraint • Check constraint • Default constraint • Data type constraint • Nullability constraint
Referential integrity	• Foreign key constraint
User-defined integrity	• Check constraint

Use database constraints whenever possible

There are two main reasons why using database constraints is a preferred way of enforcing data integrity.

First, constraints are inherent to the database engine and so use less system resources to perform their dedicated tasks. We resort to external user-defined integrity enforcement only if constraints are not sufficient to do the job properly.

Second, database constraints are always checked by the database engine before insert, update, or delete operation. Invalid operation is

cancelled before the operation is undertaken. So they are more reliable and robust for enforcing data integrity.

INTRODUCTION TO VIEWS

A **view** is a virtual or logical table that allows to view or manipulate parts of the tables.To reduce REDUNDANT DATA to the minimum possible, Oracle allows the creation of an object called a VIEW. A view also has rows and columns as they are in a real table in the database. We can create a view by selecting fields from one or more tables present in the database. A View can either have all the rows of a table or specific rows based on certain condition.

Some Views are used only for looking at table data. Other Views can be used to Insert, Update and Delete table data as well as View data. If a View is used to only look at table data and nothing else the View is called a Read-Only View. A View that is used to look at table data as well as Insert, Update and Delete table data is called an Updateable View.

The reasons why views are created are:

- When Data security is required .
- When Data redundancy is to be kept to the minimum while maintaining data security.

Types of views :

1)Simple view

2)Complex View

3)Read only view

4)With check option view

5)Materialized view

6)Forced view

Creating a View :

We can create View using **CREATE VIEW** statement. A View can be created from a single table or multiple tables.

Syntax:

CREATE VIEW view_name AS

SELECT column1, column2.....

FROM table_name(s)

WHERE condition;

Where **view_name**: Name for the View
table_name: Name of the table
condition: Condition to select rows

Creating a Simple View

Simple view is the view that is made from a single table, It takes only one table and just the conditions. While creating a simple view, we are not creating an actual table, we are just projecting the data from the original table to create a view table.

Example 1: In this example, we are creating a view table from Employee Table for getting the EmpID and EmpName. So the query will be:

CREATE VIEW EmpView1 AS
SELECT Empno, Ename
FROM Emp;

Now to see the data in the EmpView1 view created by us, We have to simply use the SELECT statement.

SELECT * from EmpView1;

Output:

Example 2: In this example, we are creating a view from Emp Table for getting the EmpID and job for employees with department number 10.

```
CREATE VIEW EmpView2 AS

SELECT Empno, job

FROM Emp

WHERE deptno=10;
```

Now to see the data in the EmpView2 view

```
SELECT * from EmpView2;
```

Output:

Creating a Complex View

The complex view is the view that is made from multiple tables, It takes multiple tables in which data is accessed using joins or SELECT clause contains inbuilt SQL aggregate functions like AVG(), MIN(), MAX() etc, or GROUP BY clause.

Example:

```
CREATE VIEW EmpView3 AS

SELECT E.empno,E.ename,E.sal,D.dname,D.loc

FROM emp E,dept D

WHERE E.deptno=D.deptno;
```

Now to see the data in the EmpView3 view

```
SELECT * from EmpView3;
```

Output: