

RELATIONAL MODEL

INTRODUCTION TO THE RELATIONAL MODEL: The relational data model was first introduced in 1970 by E.F.Codd at IBM research laboratory. The second at the university of California in 1980 and named as RDBMS.

Definitions: The relational data model represents data in the form of tables. This model consists of three components.

- 1) **Data structure :** In RDBMS data are organized in the form of tables with rows and columns.
- 2) **Data manipulation :** In data manipulation the powerful operations are used to manipulate data stored in the relations.
- 3) **Data integrity:** In RDBMS, facilities are included to specify business rules, that maintain the integrity of data when they are manipulated.

The E.F. Codd proposed the relational data model in 1970. At that time, the hierarchical and network models were using for database. To reduce the draw backs of hierarchical and network models, introduced relational database system. The relational database model is the product of DBMS. The relational model were introduced by IBM's DB2 family (Oracle, Sybase) and other are Microsoft's Access, SQL server, Foxbase, Foxpro etc.

What is relational model? How to create a relational table? Explain with suitable example.

The E.F. Codd proposed the relational data model in 1970 To reduce the draw backs of hierarchical and network models.

The relational model is very simple and elegant. A database is a collection of one or more relations. This relation is table with rows and columns. In this, the users can easy to understand and easy to permit the high level languages to query the data.

The relational model uses Data Definition Language (DDL) and Data Manipulation Language (Query language) for creating, accessing, manipulating, and querying data in a relational DBMS

The relational model represents both data and relationship among data in the form of tables. Each table has multiple columns and each column has a unique name. This is created by create command. Consider the following relational model.

SQL> Create table customer(cid number(6), customer_city varchar(20), customer_street varchar(10), customer_city varchar(10));

Customer Relation			
cid	customer_name	Customer_street	customer_city
5555	Chaitanya	Gandhi nagar	Ongole
6666	Bhagya	sujatha nagar	Ongole
7777	Sumathi	ivothi bazzar	Ongole

An important of a relational model is that it specifies some conditions that must satisfied while accepting the values from the user which prevents the entry of incorrect information. Such conditions are called Integrity constraints.

What is Relation? Define the terms relation schema, relation instance, unary, degree of relation and domain constraints and its types.

Representing the data in the relational model is called relation. A relation contains set of records in the form of two-dimension table which contain rows and columns of data. A relation consists of two things, such as a relation schema and a relation instance.

Relation schema: A relation schema contains the basic information of a table or relation. This information includes the name of the entire table, the names of the column and the data types associated with each column.

For example: A relation schema for the relation called students could be expressed using the following representation,

Students (sid : string, name :string, course: string, age: integer, fees : real);

In this table, contain five column attributes with respective data types such as string, integers, real are associated with it.

Relation instance: A relation instance is a set of rows (also known as records) that when combined together forms the schema of the relation. A relation instance can be a table which each tuple (record) is a row and all rows has same number of fields (columns).

For example,

cid	customer_name	Customer_street	customer_city
5555	Chaitanya	Sri nagar	Ongole
6666	Anusha	Madhuri nagar	Ongole
7777	Nithisha	Swathi bazzar	Ongole

Relational Database Schema: A relational database schema is a collection of relation schemas, describing one or more relations.

Domain: Domain is synonymous with data type. Attributes is a column in a table. Therefore an attribute domain refers to the database associated with a column.

Relational Cardinality: The relation cardinality is the number of tuples (rows or records) in the relation.

Relation Degree: The relation degree is the number of fields (column or attribute) in the relation.

Tuples/Records: The rows of the table are also known as fields or attributes.

Unary: A relation with only one attribute and have only one degree is called a unary relation.

How to create the relation and modify the relation using SQL:

1) Creating relation in SQL (Structured Query Language) : Relations are created in SQL using create command.

SQL> Create table customer(cid number(6), cname varchar(20), accno number(6), amount number(8,2));

2) Inserting records in a table : Records are inserted in a table using an insert command as,

SQL> insert into customer(cid, cname, accno, amount) values (5, 'badri', 5555, 15000.00);

or

SQL> insert into customer values (5, 'badri', 5555, 15000.00);

or

SQL> insert into customer values(&cid, '&cname', &accno, &amount);

3) Deleting tuples from a table : Tuples in the table can be deleted using delete command in SQL as,

SQL> delete from customers where accno = 5555;

4) Modifying the column values: Values in a particular row can be changed using an update command as,

SQL> update customer set cname = 'Chaithanya' amount = 15000.00;

SQL> update customer set amount = amount + 1000 where accno <= 5555;

Importance of NULL values: The SQL NULL is the term used to represent a missing value. A NULL value in a table is a value in a field that appears to be blank.

A field with a NULL value is a field with no value. It is very important to understand that a NULL value is different than a zero value or a field that contains spaces.

Principles of NULL values:

- Setting a null value is appropriate when the actual value is unknown or when a value would not be meaningful.
- A null value is not equivalent to a value of zero.
- A null value will evaluate to null in any expression: null multiplied by 10 is null.
- When a column name is defined as not null, then that column becomes a mandatory column. It implies that the user is forced to enter data into that column.

INTEGRITY CONSTRAINTS OVER RELATION:

What is an integrity constrain? Explain "Integrity constraints over relations".

Integrity constraint is a condition (rule) that ensures the correct insertion of the data and prevents unauthorized data access thereby preserving the consistency of the data.

For example, the roll number of a student cannot be a decimal value. The database enforces the constraint that the instance of roll number can have only integer values.

The Integrity constraints can be applied in the following cases.

1. At the time of defining the database, the constraints are must be specified.
2. Validity of the data must be checked while executing the application.

The major integrity constraints are 1) **Domain Constraints**. 2) **Key constraints**

3) **Referential integrity constraints** 4) **General Constraints**.

- 1) **DOMAIN CONSTRAINTS** : The domain constraints are used to prevent from null values and allows specified range of values. A domain definition consists domain name, meaning, data type and size (or) length. The domain integrity constraints are classified into two types.

NULL/NOT NULL: The word 'NULL' means zero and it is a default value in a column_name. The word 'NOT NULL' means not zero, it is used with one or more column_names that do not accept zero's.

Note: 1) zero and null are not equivalent. 2) One null is not equivalent to another null

Eg: **CREATE table bvsr_table(stu_id number(6) not null);**

CHECK constraint : The CHECK constraint can be specified to column_name to allow only a particular range of values. The CHECK constraint contain a condition that must satisfy in each row. These conditions governed by logical expressions (or) Boolean expression. Note: Check constraint cannot be allowed in sub-queries.

Eg: 1) **CREATE TABLE student(sno number(6) CHECK (sno <=100), sname varchar2(20) NOT NULL, course varchar2(10) CHECK(course IN('btech','mtech','degree')));**

2) KEY CONSTRAINTS : Key constraints are used in relations to uniquely identify the records of the relation. A Key constraint can be defined as a statement that consists of minimal subset of attribute that uniquely determine a record in a table. The different types of key constraints are,

i) Entity Integrity Constraint : An entity represents a table and each row of table represents an instance of the entity. Each row in a table can be uniquely identified by using the entity constraint. This constraint can be classified into two types. They are a) Unique constraint b) Primary key constraint.

a) **UNIQUE constraint:** It is defined with column to handle unique values. This can be used to prevent from duplication values within the rows of specified column (or) set of columns. It means, it cannot allow duplicate values in the table. But it allows NULL values. It can be defined in more than one column. (i.e up to 16 columns).

Eg: 1) **CREATE TABLE emp_table(emp_id number(6) unique, emp_name char(20));**

b) **Primary Key :** A PRIMARY KEY is a constraint that uniquely identify each row in a table. The primary key cannot accept NULL value and same value (i.e duplicates). A table can allow only one primary key. For example, the primary key for the student is defined to stu_id (or) Employee is emp_id.

Eg: **CREATE TABLE student(sno number(6) PRIMARY KEY, sname char(20), fname char(20), fee number(7,2), date_of_join date);**
OR

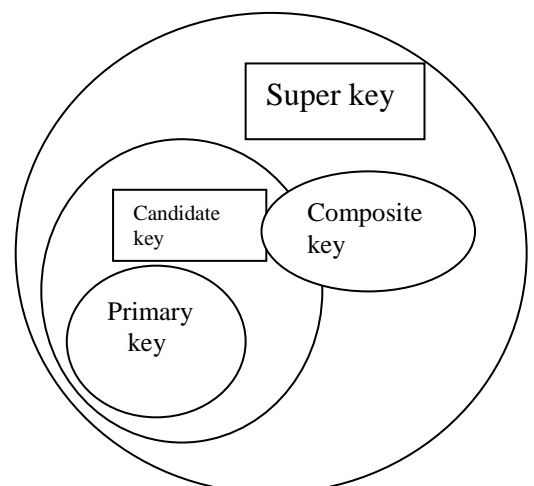
Eg: **CREATE TABLE student(sno number(6), sname char(20), fname char(20), fee number(7,2), date_of_join date, PRIMARY KEY(sno));**

ii) Candidate Key: A candidate key is a collection of fields / columns / attributes that uniquely identifies a tuple (record). For example, in customer relation (table) the attribute "cid" is a key that uniquely defines a tuple in a relation. No two rows in a "customer" relation of "cid" can have same value.

For example, **customer (cid, cname)** in this, either cid or cname can be taken as key but not both. Each of them uniquely identifies a particular row. The alternate keys are the candidate keys that are not taken as keys.

iii) Composite Key: Composite key consist of more than one attribute that uniquely identifies a tuple in a relation. All the attributes that form a set of keys and all of them taken together determines a unique row in a table. For example, the set (cid, accno) is a composite key which maintains the uniqueness of each row. Both cid, accno are taken as keys.

iv) Super Key: A super key is a combination of both candidate key and composite key. That is a super key is a set of attributes or a single attribute that uniquely identifies a tuple in a relation. For example, consider the super key, {cid, accno, cname}



Here all the three attributes taken together can identify a particular record or a combination of any two attributes can identify a particular record or any one of the attribute can identify a particular record.

3) Referential Integrity Constraint : It establish the relation between two tables when having a common column in both tables. A referential Integrity constrain is a rule that states the relation between primary key value and foreign key value. It means, it establishes the relation between two tables. In this case, the primary key value will be referenced in another table by a foreign key and that cannot be deleted. The referential integrity constraints are a) Referenced key b) Foreign key

a) Referenced key: It is a unique (or) primary key which is defined on a column belongs to the parent table. The referential integrity constraint does not use foreign key to identify the column that make up the foreign key. The foreign key is automatically enforced on the columns.

Eg: 1. CREATE TABLE student(sno number(5) PRIMARY KEY, sname char(20) NOT NULL, fname char(20) NOT NULL, fee number(7,2) NOT NULL, course char2(10));

Eg: 2. CREATE TABLE library (sno number(5) NOT NULL, sname char(20) NOT NULL, issued_book_name char(20) NOT NULL, issued_book_id number(7,2) NOT NULL, course varchar2(10), CONSTRAINT sno REFERENCES student(sno));

⇒ In the library table, the value of sno is referenced in the student table (sno).

b) Foreign Key : Foreign key provides links between two tables. Foreign keys are used to maintain data consistency. A foreign key is a set of attributes or a single attribute in one relation that forms a point of candidate key in other relation. A column (or) combination of columns included in the definition of referential integrity which would refer to a referenced key. The foreign key operation is only in one table. For example,

Eg: 1 CREATE TABLE dept(deptno number(4) primary key, dname varchar2(20), loc varchar2(20);
CREATE TABLE emp(emp_id number(5) primary key, ename char(20), sal number(7,2), hiredate date, deptno number(4) constraint FK FOREIGN KEY(deptno) references dept(deptno));

4) **General Constraints:** Domain, primary key, foreign key constraints are considered to be a fundamental part of the relational data model and are given special attention in most systems. Sometimes, it is necessary to specify more general constraints.

Default constraint: The DEFAULT constraint is used to provide a default value for a column. The default value will be added to all new records IF no other value is specified.

Ex: create table sailors(sid number(4), sname varchar2(10), address varcahr2(15) default 'ONG');

In general constraints, the domain, primary key and foreign key can also specify the maximum limit. **For example,** we require a student whose age is greater than 16 must have a gpa less than 3.5.

There are two types of general constraints. They are,

1) **Table constraints:** These are applied on a particular table and are checked every time whenever that specific table is updated.

2) **Assertions:** These assertions are applied on collection of tables and are checked every time whenever these tables are updated.

DATA TYPES IN SQL : The data types are used to define the letters or words as a variable name. The variable name is in a group, is called field name or column name. The first letter in a column name must be an alphabet and next letters may be an alphabet or numbers. No special character is allowed except underscore.

The column name can hold the data, carry the data and can change the data. The data types are

Number data type : This data type is used to define a field as a numeric variable to hold the numeric data. The numeric data may be fixed (whole) or floating point values.

Syntax : column name number(size);

- The column name is name of field in a table to store the data.
- The word 'number' is reserved word, used to define the column name as numeric type to store numeric values.
- The word 'size' specifies 'p' and 's'. The 'p' means precision that determine maximum length and 's' means scale that determine number of decimal places in the maximum length.

Eg: 1) sno number(5); This defines the word 'sno' as integer variable with size 5 to store 5 digit integer data.

2) rate number(7,2); This defines the word 'rate' as floating point variable with size 7,2 to store floating point data. [5 integer places and two decimal places. Eg: 00045.75

Character data types : These data types are used to define the field names as character variable to store alphanumeric data. The character data types are

Char data type: This data type is used to define a field name as character variable to store alphanumeric data. (Maximum size of data 2^7 bits=255). CHAR on the other hand is store fixed length character data.

Syntax : column name char(size);

- The column name is name of field in a table to store the data.
- Fixed-length character data of length *size* bytes.
- The word 'char' is reserved word that define a column name as alphanumeric variable to store alphanumeric data.
- The word 'size' specifies maximum number of characters can stored in column name.

Eg: sname char(20); It defines the word 'sname' as character variable to store alphanumeric data.

Nchar data type: Fixed for every row in the table (with trailing blanks). Column *size* is the number of characters for a fixed-width national character set or the number of bytes for a varying-width national character set. Maximum *size* is determined by the number of bytes required to store one character, with an upper limit of 2000 bytes per row. Default is 1 character or 1 byte, depending on the character set. Nchar is used to store fixed length Unicode data. It is often used to store data in different languages

- Fixed-length character data of length *size* characters or bytes, depending on the national character set.

Varchar data type: This data type is used to define a field name as character variable to store alphanumeric data up to maximum 2000. VARCHAR is reserved by Oracle to support distinction between NULL and empty string in future, as ANSI standard prescribes

Syntax : column name varchar(size);

- The column name is name of field in a table to store the data.
- The word 'varchar' is reserved word that define a column name as alphanumeric variable to store alphanumeric data.
- The word 'size' specifies maximum number of characters can stored in column name.

Eg: sname varchar(20); It defines the word 'sname' as character variable to store alphanumeric data.

Varchar2 data type: This data type is used to define a field name as character variable to store alphanumeric data up to maximum 2000. VARCHAR2 does not distinguish between a NULL and empty string, and never will.

Syntax : column name varchar2(size);

- The column name is name of field in a table to store the data.
- The word 'varchar2' is reserved word that define a column name as alphanumeric variable to store alphanumeric data.
- The word 'size' specifies maximum number of characters can stored in column name.

Eg: sname varchar2(20); It defines the word 'sname' as character variable to store alphanumeric data.

Date data type : This data type used to define the column name as date type to date in the format DD-MM-YY.

Syntax : column name date;

- The column name is name of field in a table to store the date.
- The word 'date' is reserved word that define a column name as date variable to store date.

Eg: dob date; It defines the word date of birth as date variable to the date in the format.

For example : 05-aug-2005.

Note: Month must be 3 character data.

LONG data type : This data type is used to define a column name as character type variable to store alphanumeric data up to 2GB.

RAW data type : This data type is used to define a column name as binary variable to store binary data upto 255 bytes. The binary data may be digitized picture or image.

LONG RAW data type : This data type is used to define a column name as binary variable to store binary data upto 2GB. The binary data may be digitized picture or image.

RELATIONAL ALGEBRA:

The relational is a procedural query language. It consists of a set of operations that take one or two relations (tables) as input and produce a new relation, on the request of the user to retrieve the specific information, as the output (result).

The relational algebra contains the following operations:

- 1) Selection 2) projection 3) union 4) Rename 5) Difference 6) Cartesian product
- 7) Intersection 8) join 9) division .

The selection, projection and rename operations are called unary operations because they operate only on one relation.

The other operations operate on pairs of relations and are therefore called binary operations.

The Selection (σ) Operation: The selection is a relational algebra operation that uses a condition to select rows from a relation. A new relation (as a result) is created from another existing relation by selecting only rows requested by the user that satisfy a specified condition. This is denoted by greek letter sigma (σ) to denote selection.

Syntax: selection condition (relation_name)

→ condition allows relational and logical operators.

Eg: Find the customer details who are living in Hyderabad city from customer relation.

$\sigma_{city = 'Hyderabad'}(customer)$ it is equal to

select * from customer where city = 'Hyderabad';

Eg: Find the customer details who are living in Hyderabad city and whose customer_id is greater than 1000 in customer relation.

$\sigma_{city = 'Hyderabad' \wedge customer_id > 1000}(customer)$ it is equal to

select * from customer where city='Hyderabad' and customer_id > 1000;

The Projection (π) Operation : The projection is a relational algebra operation that creates a new relation by deleting columns from an existing relation i.e., a new relation (as a result) is created from another existing relation by selecting only those columns requested by the user from projection and is denoted by letter pi (π) .

The selection operation eliminates unwanted rows whereas the projection operation eliminates unwanted columns. In simple words, the projection operation extracts specified columns from a table.

Example : Find the customer names (not all customer details) who are living in Hyderabad city from customer relation.

$\pi_{customer_name}(\sigma_{city = 'Hyderabad'}(customer))$

In the above example, the selection operation is performed first. Next, the projection of the resulting relation on the customer_name column is carried out. It means, it will display only customer names who is living in Hyderabad city. So, it is equal to

Select customer_name from customer where city = 'Hyderabad';

Example: Find all customer names of the customer relation.

$\pi_{\text{customer_name}}(\text{customer}) \rightarrow$ This is equal to "Select customer_name from customer"

SET OPERATIONS:

The Union (\cup) Operation: The union denoted by \cup . It is a relational algebra operation that creates a union or combination of two relations. The result of this operation is denoted by "table1 \cup table2". It means, all tuples (records) that are either in table1 or in table2 or in both table1 and table2 will be displayed. But duplicate tuples are eliminated.

Example: Find the customer_id of all customers in the bank who have either an account (depositor table) or a loan (borrower) or both.

$\pi_{\text{customer_id}}(\text{depositor}) \cup \pi_{\text{customer_id}}(\text{borrower})$ This is equal to

(select customer_id from depositor) union (select customer_id from borrower);

The Intersection (\cap) Operation : The intersection operation is denoted by \cap . This is relational algebra operation that finds tuples that are in both relations. The result of this operation is denoted by table1 \cap table2. In this all tuples, (records) which are common in both depositor relation (table) and borrower relation (table).

Example: Find the customer_id of all customers in the bank table and who have an account in borrower table and a loan.

$\pi_{\text{customer_id}}(\text{depositor}) \cap \pi_{\text{customer_id}}(\text{borrower})$

(select customer_id from depositor) intersect (select customer_id from borrower);

The Difference (-) Operation : The set difference operation is denoted by symbol $-$. This is a relational algebra operation that finds tuples that are in one relation but not in another. The result of this relation is denoted by table1 $-$ table2. It means, it will display all tuples of depositor relation but not in borrower relation.

Example: $\pi_{\text{customer_id}}(\text{depositor}) - \pi_{\text{customer_id}}(\text{borrower})$

(select customer_id from depositor) minus(select customer_id from borrower);

The result of this query is to display the customer ids of all customers who have an account but not a loan. It means, it display the customer ids of all customers of depositor table but not having an account in borrower table.

The Cross-Product (or) Cartesian Product (\times) Operation: The Cartesian-product operation is denoted by a cross symbol (\times). This is a relation algebra operation which allows us to combine information from two relations into one relation. For example, table1 \times table2 is considered as to display all records of both table i.e., table1 \times table2. Suppose if a table1 contain 10 records and table2 contain 5 records then outputted records are $10 \times 5 = 50$ records. But draw back is same records and same attributes are repeated.

Example: Find customer ids of all customers and borrower.

$\pi_{\text{customer_id}}(\text{customer} \times \text{borrower})$

It is equal to select customer_id from customer, borrower;

Find customer ids of all customers who have loan > 10000.

$\pi_{\text{customer_id}} (\sigma_{\text{customer.loan_no} = \text{borrower.loan_no} (\sigma_{\text{amount} > 10000}(\text{customer} \times \text{borrower})))$ This is equal to
 select customer_id from cusotmer, borrower where customer.loan_no = borrower.loan_no and customer.amount > 10000;

Renaming (ρ): The rename operation is denoted by rho (ρ), is a relational algebra operation. This is used to give the new names to the relation algebra expressions. For example, the relation “customer” can be changed as new name as x.

$\rho_x(\text{customer}) \rightarrow$ It means, customer name is renamed as x.

Ex: select ename, job*12 as "annualsal" from emp;

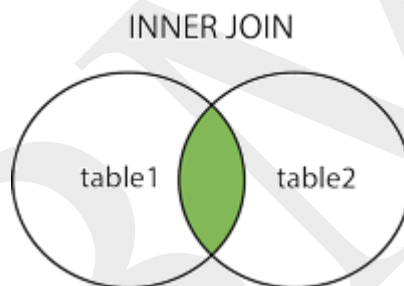
SQL JOINS:

Join is a query in which data is retrieved from two or more tables. A join matches from two or more tables, based on the value of one or more columns in each table.

The join is denoted by join symbol \bowtie It is a relational algebra operation which is used to combine (join) two relations like Cartesian-product but finally removes duplicate attributes and makes the selection, projection operation. The join operations are 3 types.

- 1) Inner join
- 1) Natural join
- 2) Outer join

Inner join: inner join returns the matching rows from the table that are being joined



Syntax: select * from table1 inner join table2 on table1.column=table2.column;

Ex: select * from emp e inner join dept d on e.deptno=d.deptno;

Natural join or Equi-join: Natural Join is a type of Inner join which is based on column having same name and same datatype present in both the tables to be joined. The Natural join is a binary operation that allows us to combine two different relations into one relation and makes the same column in two different relations into only one-column in the resulting relation. Suppose we have relations with following schemas that explained for natural join and for equi-join.

Customer table

Cust_id	Cust_name	Address	City	state	postal_code
101	Ramesh	Market center	Ongole	A.P	523001
102	Avinash	Ram nagar	Ongole	A.P	523002
103	Sunil	Gandh nagar	Ongole	A.P	523003
104	Vasanthi	Anjaiah road	Ongole	A.P	523003
105	Krishna	Mm road	Ongole	A.P	523003

Order table

Order_id	Order_date	Product_name	Qty	Cust_id
1001	'21-Apr-08'	Xxx	5	101
1002	'22-Apr-08'	Yyy	2	103
1003	'25-May-08'	Zzz	8	101
1004	'27-May-08'	Aaa	7	106
1005	'27-May-08'	Bbb	3	104

A join in which the joining condition is based on equality between values in common columns. Common columns appear in the result table. For example, 'Customer' and 'Order' tables are using to apply equi-join operation for retrieving the related rows.

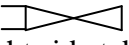
```
SQL> SELECT CUST_ID, CUST_NAME, ADDRESS, ORDER_ID, ORDER_NAME, ORDER_DATE,
Product_NAME, qty FROM CUSTOMER, ORDER WHERE CUSTOMER.CUST_ID =
ORDER.CUST_ID;
```

```
SQL> SELECT * FROM EMP NATURAL JOIN DEPT ;
```

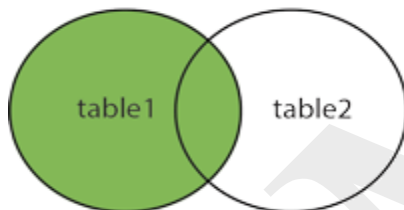
OUTER-JOIN: When tables are joined using inner join, rows which contains matching values in the join predicate are returned. Sometimes you may want both matching and non matching rows returned for the table are being joined. This kind of operation is known as an outer join.

An outer join is an extended form of the inner join. In this the rows in one table having no matching rows in the other table will also appear in the result table with nulls.

Types of outer join: 1) Left Outer-join. 2) Right Outer-join 3) Full Outer-join

LEFT OUTER JOIN :  It displays all rows from left side table (i.e CUSTOMER table) and non-matching values of right side table are not added but displays with null values. (i.e ORDER table).


LEFT JOIN



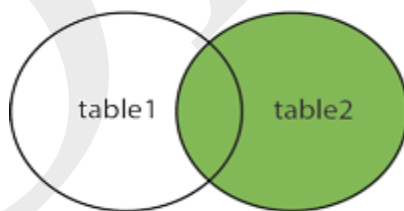
Query : SELECT * FROM EMP E,DEPT D WHERE E.DEPTNO=D.DEPTNO(+);

OR

SELECT * FROM EMP E LEFT OUTER JOIN DEPT D ON E.DEPTNO=D.DEPTNO;

RIGHT OUTER JOIN :  It displays all rows from right side table (i.e ORDER table) and matching values from left side table (i.e CUSTOMER table) and non-matching values of left side table are not added but displays with null values.

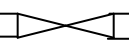
RIGHT JOIN



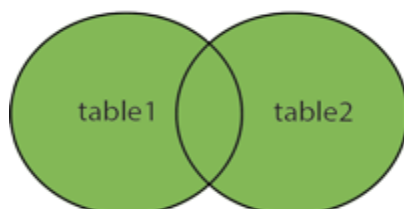
SELECT * FROM EMP E,DEPT D WHERE E.DEPTNO(+)=D.DEPTNO;

OR

SELECT * FROM EMP E RIGHT OUTER JOIN DEPT D ON E.DEPTNO=D.DEPTNO;

FULL OUTER JOIN :  It displays all rows when there is a match in either left or right table.

FULL OUTER JOIN



Query: SELECT C.CUST_ID, C.CUST_NAME, C.ADDRESS, O.ORDER_ID, O.ORDER_NAME, O.ORDER_DATE, O.P_NAME, O.QTY FROM CUSTOMER C FULL OUTER JOIN ORDER O WHERE C.CUST_ID = O.CUST_ID;

The Division (\div) Operation: The division operation is denoted by \div , is a relational algebra operation that creates a new relation by selecting the rows in one relation that does not match rows in another relation.

Let relation A is $(x_1, x_2, x_3, \dots, x_m, y_1, y_2, \dots, y_m)$ and relation B is (y_1, y_2, \dots, y_m)

Where y_1, y_2, \dots, y_m tuples are common to the both relation A and B with same domain compulsory. Then $A \div B$ = new relation with $(x_1, x_2, x_3, \dots, x_m)$ tuples.

Relation A and B represent the dividend and divisor respectively.

A tuple 't' is in $a \div b$ if and only if two conditions are satisfied,

1. t is in $\pi(A - B)(r)$.

2. For every tuple t_b in B, there is a tuple t_a in A satisfying the following two things.

i) $t_a[B] = t_b[B]$ ii) $t_a[A - B] = t$

Ex:A

SNO	PNO
S1	P1
S1	P2
S1	P3
S1	P4
S2	P1
S2	P2
S3	P2
S4	P2
S4	P4

B1

PNO
P1
P2
P4

A/B1

SNO
S1

B2

PNO
P2

A/B2

SNO
S1
S2
S3
S4

RELATIONAL CALCULUS : Relational calculus is an alternative to relational algebra. In contrast to the algebra, which is procedural, the relational calculus is non-procedural or declarative because in that it allows us to describe the set of answers without showing procedure about how they should be computed.

Relational calculus has a big influence on the design of commercial query languages such SQL and especially, Query-by Example(QBE).

Relational calculus is of two types,

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

Tuple Relational Calculus (TRC): The tuple relational calculus is a non-procedural query language because it gives the desired information without showing procedure about how they should be computed.

A query in tuple relational calculus is expressed as $\{T | p(T)\}$

where, $T \rightarrow$ tuple variable, $p(T) \rightarrow$ 'p' is a condition or formula that is true for 't'

In addition to that we use,

$T[A] \rightarrow$ to denote the value of tuple t on attribute A and

$t \in r \rightarrow$ to denote that tuple t is in relation r.

Syntax of TRC Queries:

Let 'rel' be a relation name, 'R' and 'S' be tuple variables, 'a' is an attribute of R. 'b' is an attribute of S.

Let 'op' denote an operator in the set. $\{<, >, =, >=, <=, \neq\}$

Then, an atomic formula is one of the following.

- 1) $R \in \text{Rel.}$
- 2) $R.a \text{ op } S.b.$

$R.a \text{ op constant, or constant op } R.a.$

A formula is recursively defined to be one of the following, where 'p' and 'q' are themselves formulas and 'p(R)' denotes a formula in which the variable R appears,

- 1) any atomic formula,
- 2) $\neg p, p \wedge q, p \vee q$ or $p \implies q$
- 3) $\exists R (p(R))$, where R is a tuple variable,
- 4) $\forall R (p(R))$, where R is a tuple variable.

In the last two clauses above, the quantifiers \exists and \forall are said to be the variable R.

Semantics of TRC Queries : The TRC query $\{ T | p(T) \}$ is the set of all tuples 't' for which the formula $p(T)$ evaluates to true with variable T assigned the tuple value t. To complete this definition, we must state which assignments of tuple values to the free variables in a formula make the formula evaluates to true.

A query is evaluated on a given instance of the database. Let each free variable in a formula F be bound to a tuple value. For the given assignment of tuples to variables, with respect to the given database instance, F evaluates to true if one of the following condition is true.

- 1) F is an atomic formula $R \in \text{Rel}$ and R is assigned a tuple in the instance of relation Rel.
- 2) F is a comparison $R.a \text{ op } S.b$, $R.a \text{ op constant}$, or $\text{constant op } R.a$ and the tuples assigned to R and S have filed values R.a and S.b that make the comparison true.
- 3) F is of the form $\neg p$ and p is not true, or of the form $p \wedge q$ and both p and q are true, or of the form $p \vee q$ and one of them is true, or of the form $p \implies q$ and q is true whenever p is true.
- 4) F is of the form $\exists R (p(R))$ and there is some assignment of tuples to the free variables in $p(R)$, including the variable R, that makes the formula $p(R)$ true.
- 5) F is of the form $\forall R (p(R))$ and there is some assignment of tuples to the free variables in $p(R)$ that makes the formula $p(R)$ true no matter what tuple is assigned R.

Domain Relational Calculus (DRC): A domain variable that comes in the range of the values of domain (data types) of some columns (attributes).

Syntax of DRC Queries: Let 'op' denote an operator in the set $\{ <, >, =, >=, <=, \neq \}$ and let x and y be domain variables.

An atomic formula in DRC is one of the following,

- 1) $\langle x_1, x_2, x_3, \dots, x_n \rangle \in \text{Rel}$, where Rel is a relation with n attributes, each $x_i, 1 \leq i \leq n$ is either a variable or a constant.
- 2) $x \text{ op } y$
- 3) $x \text{ op constant}$, or $\text{constant op } x$:

A formula is recursively defined to be one of the following, where p and q are themselves formulas and each $p(x)$ denotes a formula in which the variable x appears.

- 1) any atomic formula,
- 2)
- 2) $\neg p, p \wedge q, p \vee q$ or $p \implies q$
- 3) $\exists R (p(x))$, where x is domain variable,
- 4) $\forall R (p(x))$, where x is a domain variable.

The reader can remember these formulas very easily by comparing them with TRC formulas and see how closely these two formulas correspond.

SCHEMA DEFINITIONS IN SQL:

The schema is collection of related objects which are created by user with create command such as tables, views, domains, constraints, character sets, triggers, roles etc., In simple way, schema is nothing but plan how to store the data on secondary storage device and how to extract the from secondary storage device.

SQL Commands are used to define the definition to Schema by using DDL commands,

Data Definition Language Commands (DDL) : The DDL Commands are used to define database table in schema. It includes create, alter, drop the tables and establishing constraints. These commands are restricted to production version database.

CREATING THE TABLES: The tables are created in SQL by CREATE command. The CREATE command is used to create a table with column_names. The column_name are called variables or field names. To create a table, must follow the below steps.

1. Identify the appropriate data type, including length, precision, and scale for each attribute if required.
2. Identify the columns that should accept null values.
3. Identify the columns that need to be unique.
4. Identify the primary key and foreign key for attributes.
5. Determine the values to be inserted in any column for which a default value is specified.
6. Identify any column for which domain specifications may be stated. Eg: CHECK

7. Create index using existed table.

Syntax: CREATE TABLE table_name(column_name1 data type1 [<constraint_type attribute_name>], column_name2 data type2 [<constraint_type attribute_name>]

 [CONSTRAINT constraint_name]);

- The table_name is name of table to store the data in secondary memory.
- The column_name is name of filed name, used to store the data.
- The data type is used to define column_name to store related data. The data types are number(), char(), varchar2() and date.
- The constraint_types are used with column_name to specify the constraint type. They are i) PRIMARY KEY, ii) FOREIGN KEY, iii) UNIQUE KEY iv) REFERENCE v) NOT NULL vi) CHECK.

Eg: CREATE TABLE student(sno number(5),sname char(20), fname char(20),fee
 number(7,2),date_of_join date);

Performing the operations on table definitions: Once the table is created, the user can perform operations on table. They are adding the coulumn_names, change the size of column_name, and can remove the column_name and table. These operations are done by using ALTER command, DESC command, DROP command etc.

ALTER command: The ALTER command is used to change the definition of a table.

Syntax: ALTER TABLE table name

ADD
MODIFY
DROP
RENAME

 (column_name data type(size));

ADD clause : The ADD is a keyword, used to add a column_name and/or constraints to an existing table.

Syntax : ALTER TABLE table_name ADD(new column_name data type);

Eg: ALTER TABLE student ADD(date_of_join date);

MODIFY clause : The MODIFY is a keyword, used to modify the definition of an existing column_name. The MODIFY keyword in the ALTER command cannot make following three changes. They are

- i) can not change the specification of column_name from NULL to NOT NULL when column_name containing nulls.
- ii) can not decrease the size of a column_name (or) data type when column_name contain a data.
- iii) can not define constraints on a column_name except NULL/NOT NULL.

Ex: alter table student modify sno number(4);

DROP clause : It is used with ALTER command to remove a constrained from a table.

Syntax : ALTER TABLE table_name DROP constraint constraint_name;

Eg: ALTER TABLE emp DROP constraint PK;

RENAME clause: it is used to rename the column name

Syntax: alter table table_name rename column old column name to new column name;

Ex: alter table student rename column date_of_join to doj;

DESC command : This command is used to display the definition of an existed table. I.e., it will display the structure of specified table.

Syntax : DESC table_name; Eg: DESC student;

DROP command: This command is used to delete the table name from the secondary memory.

Syntax : DROP TABLE table_name; Eg: DROP TABLE student;

RENAME command: This command is used to change the name of the table definition.

Syntax : RENAME old name TO new name;

Eg : RENAME student TO student_table;

DBMS