UNIT-II

RELATIONAL MODEL

Relational Model in DBMS

After designing the conceptual model of the Database using ER diagram, we need to convert the conceptual model into a relational model which can be implemented using any RDBMS language like Oracle SQL, MySQL, etc.

What is the Relational Model?

The relational model represents how data is stored in Relational Databases. A relational database stores data in the form of relations (tables). Consider a relation STUDENT with attributes ROLL_NO, NAME, ADDRESS, PHONE, and AGE shown in Table 1.

Student

ROLL_NO	**NAME**	**ADDRESS**	**PHONE**	**AGE**
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	9455133456	18

Attribute: Attributes are the properties that define a relation. e.g.; ROLL_NO, NAME etc

Domain: All attributes have some pre-defined scope and value which is called attribute domain.

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

Ex: **STUDENT (ROLL_NO, NAME, ADDRESS, PHONE, and AGE)** is the relation schema for STUDENT.
If a schema has more than 1 relation, it is called Relational Schema.

Tuple: Each row in the relation is known as a tuple.The above relation contains 4 tuples, one of which is shown as:

1 RAM DELHI 9455123451 18

Relation Instance: The set of tuples of a relation at a particular instance of time is called a relation instance. Table 1 shows the relation instance of STUDENT at a particular time. It can change whenever there is an insertion, deletion, or update in the database.

Degree: The number of attributes in the relation is known as the degree of the relation.
The **STUDENT** relation defined above has degree 5.

Cardinality: The number of tuples in a relation is known as cardinality.
The **STUDENT** relation defined above has cardinality 4.

Column: The column represents the set of values for a particular attribute. The column ROLL_NO is extracted from the relation STUDENT. Eg: ROLL_NO.

ROLL_NO
1
2
3
4

NULL Values: The value which is not known or unavailable is called a NULL value. It is represented by blank space**.** Eg: PHONE of STUDENT having ROLL_NO 4 is NULL.

IMPORTANCE OF NULL VALUES

It is important to understand that a NULL value is different from a zero value.

A NULL value is used to represent a missing value, but that it usually has one of three different interpretations:

- The value unknown (value exists but is not known)
- Value not available (exists but is purposely with held)
- Attribute not applicable (undefined for this tuple)

It is often not possible to determine which of the meanings is intended. Hence, SQL does not distinguish between the different meanings of NULL**.**

Principles of NULL values

- Setting a NULL value is appropriate when the actual value is unknown, or when a value is not meaningful.
- A NULL value is not equivalent to a value of ZERO if the data type is a number and is not equivalent to spaces if the data type is a character.

- A NULL value can be inserted into columns of any data type.
- A NULL value will evaluate NULL in any expression.
- Suppose if any column has a NULL value, then UNIQUE, FOREIGN key, and CHECK constraints will ignore by SQL.

Ex:

SNO	SNAME	PHONE	AGE
101	HARI	9236445222	17
102	RAMU		16
103	SAI	8738596092	17

It is a null value i.e the value which is unknown.

IMPORTANCE OF CONSTRAINTS:

- A constraint can be defined as the property assigned to a column or the set of columns in the database table which prevents inconsistent data values to be stored into the certain columns.
- Constraints are used to ensure the data integrity. This enforce the reliability and accuracy of the data stored in the database.
- In Relational Database Model, There are some Integrity Constraints which are used for accuracy and consistency of data in relational database.
- Data integrity defines some data quality requirements that the data in the database needs to be met.
- The Relational Database Management System will not allow, If a user tries to insert data that doesn't meet these requirements.

CONSTRAINTS IN RELATIONAL MODEL

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

Domain Constraints: These are attribute-level constraints. An attribute can only take values that lie inside the domain range. e.g: If a constraint AGE>0 is applied to STUDENT relation, inserting a negative value of AGE will result in failure**.**

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

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

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

STUDENT

ROLL_NO	**NAME**	**ADDRESS**	**PHONE**	**AGE**	**BRANCH_CODE**
1	RAM	DELHI	9455123451	18	CS
2	RAMESH	GURGAON	9652431543	18	CS
3	SUJIT	ROHTAK	9156253131	20	ECE
4	SURESH	DELHI	9652437548	18	IT

BRANCH

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

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

INTEGRITY CONSTRAINTS OVER RELATIONS

Integrity constraint is a mechanism used by oracle to prevent invalid data entry into the table. The following are the various types of Integrity constraints.

NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK and DEFAULT.

1. **NOT NULL Constraint:** We know that by default all columns in a table allow null values. When not null constraint is enforced, either on a column or a set of columns in a table, it will not allow null values.

Ex: SQL>create table student1(sno number(3), sname varchar2(10) not null);
```
Table created.
SQL> insert into student1(sno,sname) values(101,'ramu');
1 row created.
SQL> insert into student1(sno) values(102);

ERROR at line 1:
ORA-01400: cannot insert NULL into ("SYSTEM"."STUDENT1"."SNAME")
SQL> insert into student1(sno,sname) values(101,'ramu');
    1   row created.
```

2. **UNIQUE Constraint:** The unique constraint is used to prevent the duplication of values within the rows of specified column or a set of columns in a table. Columns defined with this constraint can also allow null values.

Ex: create table student2 (sno number(3) unique, sname varchar2(10));
```
SQL> insert into student2(sno,sname) values(101,'ramu');
1 row created.
SQL> insert into student2(sno,sname) values(101,'ramu');

ERROR at line 1:
ORA-00001: unique constraint (SYSTEM.SYS_C004037) violated
SQL> insert into student2(sname) values('hari');
1 row created.
```

3. **PRIMARY KEY:** The primary key constraint avoids both duplication of rows and null values, when enforced on a column or set of columns.

Ex: SQL> create table student3(sno number(3) primary key, sname varchar2(10));
```
SQL> insert into student3(sno,sname) values(01,'ramu');
1 row created.
SQL> insert into student3(sno,sname) values(01,'ramu');

ERROR at line 1:
ORA-00001: unique constraint (SYSTEM.SYS_C004038) violated
SQL> insert into student3(sname) values('hari');
```

```
ORA-01400: cannot insert NULL into ("SYSTEM"."STUDENT3"."SNO")
```

4. **FOREIGN KEY:** To establish a ‘parent_ child’ relationship between two tables having a common column, we make use of referential integrity constraints. To implement this, we should define the column in the parent table as a primary key and the same column in the child table as a foreign key referring to the corresponding parent entry.

A foreign key in one relation **references** primary key in another relation, the foreign key value must match the primary key value.

```
Ex : SQL> create table dept (did number(2) primary key, dname varchar2(10));
Table created.
SQL> create table emp(eno number(3) primary key, ename varchar2(10), deptno number(2)
references dept(did));
Table created.
SQL> insert into emp(eno,ename,deptno) values(101,'ramu',22);

ERROR at line 1:
ORA-02291: integrity constraint (SYSTEM.SYS_C004044) violated - parent key not found
SQL> insert into dept(did,dname) values(22,'CSE');
1 row created.
SQL> insert into emp(eno,ename,deptno) values(101,'ramu',22);
1 row created.
SQL> delete from dept;

ERROR at line 1:
ORA-02292: integrity constraint (SYSTEM.SYS_C004044) violated - child record Found
SQL> delete from emp;
1 row deleted.
SQL> delete from dept;
1 row deleted.
```

5. **CHECK Constraint:**

The check constraint can be defined to allow only a specified range of values. Check Constraint specified conditions that each row must satisfy. These rules governed by logical expressions or Boolean expressions.

```
SQL> create table student4(sno number(3),
                           sname varchar2(10),
                            branch varchar2(5) check(branch in('cse','ece'))
                           );
Table created.
SQL> insert into student4(sno,sname,branch) values(101,'ramu','cse');
1 row created.
SQL> insert into student4(sno,sname,branch) values(102,'hari','ece');
1 row created.
SQL> insert into student4(sno,sname,branch) values(103,'sita','mech');
```

```
ERROR at line 1:
ORA-02290: check constraint (SYSTEM.SYS_C004045) violated
```

6. **DEFAULT Constraint:** The DEFAULT Constraint is used to set a default value for a column.

The default value will be added to all new records, if no other value is specified.

```
Ex: SQL> create table student5(sno number(3),
                                          sname varchar2(10),
                                        branch varchar2(5) default 'cse'
                                     );
Table created.
SQL> insert into student5(sno,sname,branch)values(101,'hari','ece');
1 row created.
SQL> insert into student5(sno,sname)values(102,'mahesh');
1 row created.
SQL> select * from student5;
       SNO SNAME      BRANC
---------- ---------- -----
       101 hari       ece
       102 mahesh     cse
```

RELATIONAL ALGEBRA

Relational Algebra:

Relational algebra is a procedural query processing language that provides the base of relational databases and SQL. Relational algebra has various operators like select, project, rename, cartesian product, union, intersection, set difference, joins, etc. These operators are used to form queries in relational algebra.

Unary Operators:

The operations that operate on only one relation are called unary operations in relational algebra. The three unary operations in relational algebra are:

- Selection
- Projection
- Rename

Selection Operation:

The selection operation is a unary operation that is performed on one relation. The selection operation is used to retrieve tuples from the relation that satisfies the given condition. It is denoted by σ. Selection operation in relational algebra is written as:

Syntax: $\sigma_{condition}$ (Relationname)
Ex: Consider the table student,

REGNO	BRANCH	SECTION
1	CSE	A
2	ECE	B
3	CIVIL	B

To display all the records of CSE branch in student table, **$\sigma_{branch=cse}$(student)**

Output:

REGNO	BRANCH	SECTION
1	CSE	A

We can also add multiple conditions if required using the operators ∧ (AND), ∨and (OR). These operators are used to combine multiple conditions as required in the problem. Below is the selection operator representation with multiple conditions.

$\sigma_{condition1\ operator\ condition2\ \ldots\ condition\ n}$ (Relationname)

To select all the tuples of a relation we write the selection operation without any condition.

σ (Relationname)

Projection Operation:

The projection operation is a unary operation that is performed on a relation. A projection operation is used to retrieve all the values of one or more attributes. It is denoted by π. Projection operation in relational algebra is written as:

Syntax: $\pi_{columnname}$(Relationname)

Ex: To display regno column of above student table, $\prod_{regno}$(student)

Output:

REGNO
1
2
3

We can add multiple column names in projection operation using the comma operator if required. The comma operator is used when we have to retrieve multiple column values. Below is the projection operation representation to output multiple columns.

$\pi_{\text{columnname1, columnname2, ...,columnnamen}}$ **(Relationname)**

Rename Operation:

The rename operation is operation applied on one relation. Rename operation as the name suggests is used to rename the relation, attributes or both. It is denoted by ρ.

Rename Operation for Renaming Relation:

Rename operation for renaming relation is written as:

Syntax: $\rho_{\text{New\_relation\_name}}$ **(Old_relation_name)**

Ex: The above student name is changed to student1 by using ρ_{student1}**(student)**

Cartesian product operation:

CROSS PRODUCT is a binary set operation means, at a time we can apply the operation on two relations. But the two relations on which we are performing the operations do not have the same type of tuples, which means Union compatibility (or Type compatibility) of the two relations is not necessary.

Notation: R1 X R2

Where R1 and R2 are the relations,
the symbol '✕' is used to denote the CROSS PRODUCT operator.

It combines R1 and R2 without any condition.

Degree of R1 XR2 = degree of R1 + degree of R2

{degree = total no of columns}

Example:

Consider table R1

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

Table R2

Name	RegNo
Bhanu	2
Priya	4

R1 X R2

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

<u>Set Operators:</u>

Following operators are called as set operators.

1. Union Operator (∪)
2. Intersection Operator (∩)
3. Difference Operator (-)

<u>Condition For Using Set Operators</u>

To use set theory operators on two relations, the two relations must be union compatible.

Union compatible property means-

- Both the relations must have same number of attributes.
- The attribute domains (types of values accepted by attributes) of both the relations must be compatible.

1. Union Operator (∪) : The union operation retrieves all different rows from both tables and removes duplicate rows.

Let R and S be two relations.Then-

- R ∪ S is the set of all tuples belonging to either R or S or both.
- In R ∪ S, duplicates are automatically removed.
- Union operation is both commutative and associative.

Example-

Consider the following two relations R and S-

Relation R

ID	Name	Subject
101	Ramya	English
102	Ramu	Maths
103	Sai	Science

Relation S

ID	Name	Subject
101	Ramya	English
104	Hari	French

Relation R ∪ S

ID	Name	Subject
101	Ramya	English
102	Ramu	Maths
103	Sai	Science
104	Hari	French

2. Intersection Operator (∩) : The intersection operation retrieves only the common rows between two tables.

Let R and S be two relations.Then-

- R ∩ S is the set of all tuples belonging to both R and S.
- In R ∩ S, duplicates are automatically removed.
- Intersection operation is both commutative and associative.

Example-

Considering the above two relations R and S.Then, R ∩ S is-

Relation R ∩ S

ID	**Name**	**Subject**
101	Ramya	English

3. Difference Operator (-) : It retrieves the rows from first table which are not present in the second table.

Let R and S be two relations.Then-

- R – S is the set of all tuples belonging to R and not to S.
- In R – S, duplicates are automatically removed.
- Difference operation is associative but not commutative.

Example:

Considering the above two relations R and S
Then, R – S is-

Relation R – S

ID	**Name**	**Subject**
102	Ramu	Maths
103	Sai	Science

JOINS

JOIN:

A join is an operation that combines the rows of two or more tables based on related columns. This operation is used for retrieving the data from multiple tables simultaneously using common columns of tables. In this article, we are going to discuss every point about joins.

Join is an operation in DBMS(Database Management System) that combines the row of two or more tables based on related columns between them. The main purpose of Join is to retrieve the data from multiple tables in other words Join is used to perform multi-table queries. It is denoted by ⋈.

Types of Joins:

There are many types of Joins in SQL. Depending on the use case, you can use different types of SQL JOIN clauses. Here are the frequently used SQL JOIN types.

1. Inner Join:

Inner Join is a join operation in DBMS that combines two or more tables based on related columns and returns only rows that have matching values among tables.

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

1. Conditional Join:

Conditional join or Theta join is a type of inner join in which tables are combined based on the specified condition.
In conditional join, the join condition can include <, >, <=, >=, ≠ operators in addition to the = operator.
Example: Consider student and dept tables

student

SNO	SNAME	DEPTNO
101	SHIVA	1
102	SAI	2

dept

DID	DNAME
1	AIML
5	CSE

student $\bowtie_{deptno<did}$ dept

SNO	SNAME	DEPTNO	DID	DNAME
101	SHIVA	1	5	CSE
102	SAI	2	5	CSE

SQL Query:

SELECT s.sno,s.sname,s.deptno,d.did,d.dname FROM student s JOIN dept d on s.deptno<d.did.
Explanation: This query joins the tables student and dept and projects attributes sno , sname ,deptno ,did ,dname where condition s.deptno<d.did is satisfied.

2. Equi Join:

Equi Join is a type of Inner join in which we use equivalence('=') condition in the join condition.
Example: Consider student and dept tables

student

SNO	SNAME	DEPTNO
101	SHIVA	1
102	SAI	2

dept

DID	DNAME
1	AIML
5	CSE

student $\bowtie_{deptno=did}$ dept

SNO	SNAME	DEPTNO	DID	DNAME
101	SHIVA	1	1	AIML

SQL Query:

SELECT s.sno,s.sname,s.deptno,d.did,d.dname FROM student s JOIN dept d on s.deptno=d.did;

Explanation: This query joins the tables student and dept and the rows are retrieved based on the equality condition.

3. Natural Join

Natural join is a type of inner join in which we do not need any comparison operators. In natural join, columns should have the same name and domain. There should be at least one common attribute between the two tables.
Example: Consider student and dept tables

student

SNO	SNAME	DEPTNO
101	SHIVA	1
102	SAI	2

dept

DEPTNO	DNAME
1	AIML
2	CSE

student ⋈ dept

SNO	SNAME	DEPTNO	DNAME
101	SHIVA	1	CSE
102	SAI	2	CSE

SQL Query:

SELECT * FROM student NATURAL JOIN dept;

Explanation – Column deptno is available in both tables. Hence we write the deptno column once after combining both tables.

2. Outer Join:

Outer join is a type of join that retrieves matching as well as non-matching records from related tables. These three types of outer join

1. **Left outer join**
2. **Right outer join**
3. **Full outer join**

1. Left Outer Join:
It is also called left join. This type of outer join retrieves all records from the left table and retrieves matching records from the right table.
Example: consider two tables student and dept

student

SNO	SNAME	DEPTNO
101	SHIVA	1
102	SAI	2

dept

DID	DNAME
1	AIML
5	CSE

student ⟕ dept

SNO	SNAME	DEPTNO	DID	DNAME
101	SHIVA	1	1	AIML
102	SAI	2		

SQL Query:

SELECT * FROM student LEFT OUTER JOIN dept ON student.deptno=dept.did.

Explanation: Since we know in the left outer join we take all the columns from the left table (student).In the table dept we can see that there is no match value for deptno 2. so we mark this as NULL.

2. Right Outer Join:

It is also called a right join. This type of outer join retrieves all records from the right table and retrieves matching records from the left table. And for the record which doesn't lies in Left table will be marked as NULL in result Set.

Example: consider the above tables student and dept

student ⟖ dept

SNO	SNAME	DEPTNO	DID	DNAME
101	SHIVA	1	1	AIML
			5	CSE

SQL Query:

SELECT * FROM student RIGHT OUTER JOIN dept ON student.deptno=dept.did.

Explanation: Since we know in the right outer join we take all the columns from the right table (dept) In table student we can see that there is no match value for deptno 5. So we mark this as NULL.

3.Full Outer Join:

FULL JOIN creates the result set by combining the results of both LEFT JOIN and RIGHT JOIN. The result set will contain all the rows from both tables. For the rows for which there is no matching, the result set will contain NULL values.
Example: consider the above two tables student and dept

student ⟗ dept

SNO	**SNAME**	**DEPTNO**	**DID**	**DNAME**
101	SHIVA	1	1	AIML
102	SAI	2		
			5	CSE

SQL Query

SELECT * FROM student FULL OUTER JOIN dept ON student.deptno=dept.did.

Explanation: Since we know in full outer join we take all the columns from both tables. In the table student and dept we can see that there is no match for deptno 2 and no match for did 5 so we mark this as NULL.

RELATIONAL CALCULUS

Relational Calculus:

There is an alternate way of formulating queries known as Relational Calculus. Relational calculus is a non-procedural query language. In the non-procedural query language, the user is concerned with the details of how to obtain the end results. The relational calculus tells what to do but never explains how to do.

It is based on Predicate calculus, a name derived from branch of symbolic language. A predicate is a truth-valued function with arguments. On substituting values for the arguments, the function result in an expression called a proposition. It can be either true or false. It is a tailored version of a subset of the Predicate Calculus to communicate with the relational database.

Many of the calculus expressions involves the use of Quantifiers. There are two types of quantifiers:

- **Universal Quantifiers:** The universal quantifier denoted by ∀ is read as for all which means that in a given set of tuples exactly all tuples satisfy a given condition.
- **Existential Quantifiers:** The existential quantifier denoted by ∃ is read as for all which means that in a given set of tuples there is at least one occurrences whose value satisfy a given condition.

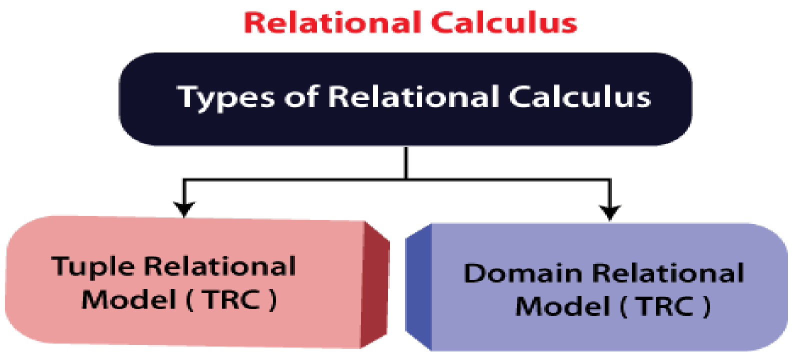
Before using the concept of quantifiers in formulas, we need to know the concept of Free and Bound Variables.

Bounded variables: A tuple variable t is bound if the condition contain quantifiers.

Free variables: A tuple variable t is free if the condition doesn't contain quantifiers.

Free and bound variables may be compared with global and local variable of programming languages.

Types of Relational calculus:


Relational Calculus
Types of Relational Calculus
Tuple Relational Model (TRC)
Domain Relational Model (TRC)

1. Tuple Relational Calculus (TRC):

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

A Query in the tuple relational calculus is expressed as following notation

Notation: {T | P (T)}   or {T | Condition (T)}

Where T is the resulting tuples

P(T) is the condition used to fetch T.

For example: { T.name | Author(T) AND T.article = 'database' }

Output: This query selects the tuples from the AUTHOR relation. It returns a tuple with 'name' from Author who has written an article on 'database'.

TRC (tuple relation calculus) can be quantified. In TRC, we can use Existential (∃) and Universal Quantifiers (∀).

For example: { R| ∃T ∈ Authors(T.article='database' AND R.name=T.name)}

Output: This query will yield the same result as the previous one.

2. Domain Relational Calculus (DRC):

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

Notation: { a1, a2, a3, ..., an | P (a1, a2, a3, ... ,an)}

Where a1, a2 are attributes
P stands for formula built by inner attributes

For example: {< article, page, subject > | ∈ javapoint ∧ subject = 'database'}

Output: This query will yield the article, page, and subject from the relational javapoint, where the subject is a database.

SQL

Database Schema:

The overall design of the database is called schema.

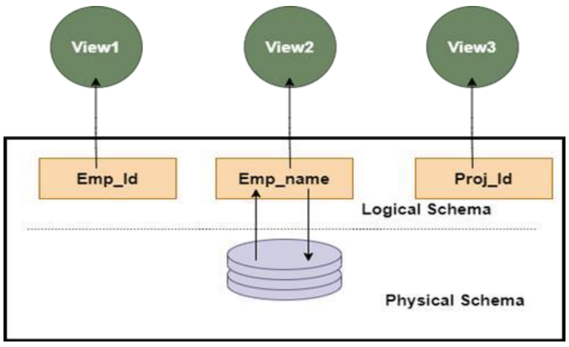
A database schema is a structure that represents the logical storage of the data in a database. It represents the organization of data and provides information about the relationships between the tables in a given database

A database schema contains schema objects that may include tables, fields, packages, views, relationships, primary key, foreign key,

The database schema is divided into three types, which are:

- Logical Schema

- Physical Schema
- View Schema



SQL Data Types:
Data types are used to represent the nature of the data that can be stored in the databasetable. Data types mainly classified into three categories for every database.

1. String Data types
2. Numeric Data types
3. Date and time Data types

1.SQL String Data Types:

CHAR(size)	It is used to store character data within the predefined length. It can be stored up to 2000 bytes.
VARCHAR(SIZE)	It is used to specify a variable length string that can contain numbers, letters, and special characters. Its size can be from 0 to 65535 characters.
VARCHAR2(size)	It is used to store variable string data within the predefined length. It can be stored up to 4000 byte.

2.SQL Numeric Data Types:

NT(size)	It is used for the integer value. Its signed range varies from 2147483648 to 2147483647 and unsigned range varies from 0 to 4294967295. The size parameter specifies the max display width that is 255.
INTEGER(size)	It is equal to INT(size).
FLOAT(size, d)	It is used to specify a floating point number. Its size parameter specifies the total number of digits. The number of digits after the decimal point is specified by **d** parameter.

FLOAT(p)	It is used to specify a floating point number. MySQL used p parameter to determine whether to use FLOAT or DOUBLE. If p is between 0 to24, the data type becomes FLOAT (). If p is from 25 to 53, the data type becomes DOUBLE().
NUMBER(p, s)	It contains precision p and scale s. The precision p can range from 1 to 38, and the scale s can range from -84 to 127.

3.SQL Date and Time Data Types:

DATE	It is used to specify date format YYYY-MM-DD. Its supported range is from '1000-01-01' to '9999-12-31'.
DATETIME(fsp)	It is used to specify date and time combination. Its format is YYYY-MM-DD hh:mm:ss. Its supported range is from '1000-01-01 00:00:00' to 9999-12-31 23:59:59'.
TIMESTAMP(fsp)	It is used to specify the timestamp. Its value is stored as the number of seconds since the Unix epoch('1970-01-01 00:00:00' UTC). Its format is YYYY-MM-DD hh:mm:ss. Its supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC.
TIME(fsp)	It is used to specify the time format. Its format is hh:mm:ss. Its supported range is from '-838:59:59' to '838:59:59'
YEAR	It is used to specify a year in four-digit format. Values allowed in four digit format from 1901 to 2155, and 0000.

TABLE DEFINITION:

1. DATA DEFINITION LANGUAGE (DDL):

The Data Definition Language is used to define schemas, relations, and other database structures and also used to update these structures.
The DDL commands are
1. CREATE
2. ALTER
3. DROP

1. CREATE COMMAND:
The CREATE command is used to implement the schemas of individual relations.

Syntax:

```
CREATE TABLE table_name( column_name1 datatype1(size) [constraint],
                         column_name2 datatype2(size) [constraint],
                          ............................
                         column_nameN datatypeN [constraint]
                         );
```

Ex:
```
SQL>create table student (sno number(3),
                         sname char(10),
                         dbms number(3),
                         ads number(3),
                         coa number(3)
                         );
Table Created.
```

2. ALTER COMMAND:

i) ALTER-ADD: We can add a column to the table by using ALTER ADD command.

Syntax:

```
ALTER TABLE table_name ADD(column_name1 datatype(size),
                           column_name2 datatype(size),….
                           );
```

Ex:
```
SQL> alter table student add(total number(4),
                             avg number(5,2),
                             GPA number(5,2)
                             );
Table altered.
```

ii) ALTER-MODIFY: We can modify the width of the data type of the column by using ALTER MODIFY command.

Syntax:

```
ALTER TABLE table_name MODIFY(column_name1 datatype(size),
                              column_name2 datatype(size),.........
                             );
```

Ex:
```
SQL> alter table student modify (sname varchar2(12));

Table altered.
```

iii) ALTER-DROP: We can delete the column of the table by using the ALTER DROP command

Syntax:
ALTER TABLE table_name DROP COLUMN column_name;
Ex: alter table student drop (GPA);

3. DROP COMMAND:

The definition of the table as well as the contents of the table is deleted by using DROP command.

Syntax: DROP TABLE table_name;
Ex: SQL> drop table student.

table dropped.

2. DATA MANIPULATION LANGUAGE:

The data manipulation language is used to add, update, and delete data in the database. The SQL command INSERT is used to add data into the database, UPDATE is used to modify the data in the database, SELECT is used to retrieve data from the database and DELETE is used to delete data in the database.

The DML Commands are
1. INSERT
2. UPDATE
3. SELECT
4. DELETE

1. INSERT COMMAND: INSERT command is used to insert records into the table.

Syntax: INSERT INTO table_name (column_names) VALUES (a list of values);

Ex: SQL> insert into student (sno,sname,dbms,ads,coa) values(101,’kiran’,66,77,88);

1 row created.

2. UPDATE COMMAND:
The UPDATE command is used to update records in the table. A single column may be updated or more than one column could be updated.

Syntax: UPDATE table_name SET column_name=new_value WHERE condition;

Ex: 1. update student set physics = 89 where sname ='swamy';
2. update student set total = maths+physics+cs;

3. SELECT COMMAND:
SELECT command is used to retrieve records from the table.

Syntax: SELECT * FROM table_name;
Ex: select * from student;

sno	**sname**	**dbms**	**ads**
101	hari	70	69
102	ramu	75	72

Here the asterisk symbol (*) indicates the selection of all the columns of the table.
Selection operation can be considered as row wise filtering. We can select specific rows using condition.

Syntax: SELECT * FROM table_name WHERE condition;

Example: SQL> Select * from student where dbms=75;

sno	**sname**	**dbms**	**ads**
102	ramu	75	72

This operation filters the rows of the relation is called SELECTION.

4. DELETE COMMAND:
The DELETE command is used to delete the records from the table.

Syntax: DELETE FROM table_name WHERE condition;
Ex: delete from student where sno=1;