DATABASE MANAGEMENT SYSTEMS

UNIT II

2.1 Introduction to relational model

Relational Model was proposed by E.F Codd to model data in the form of relations or tables. After designing the conceptual model of database using ER diagram, we need to convert the conceptual model in the relational model which can be implemented using any RDBMS (Relational Data Base Management System) like SQL, MY SQL etc.

The relational model is very simple and elegant; a database is a collection of one or more relations, where each relation is a table with rows and columns.

This simple tabular representation enables even new users to understand the contents of a database, and it permits the use of simple, high-level languages to query the data.

2.2 Relational Model

Relational Model represents how date 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 as shown in table.

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	Nishma	Hyderabad	9455123451	28
2	Sai	Guntur	9652431843	27
3	Swetha	Nellore	9156253131	26
4	**Raji**	**Ongole**	**9215635311**	**25**

Attribute: Attributes are the properties that define a relation. Ex:
ROLL_NO, NAME

Tuple: Each row in a relation is known as tuple.
Ex:

1	Nishma	Hyderabad	9455123451	28

Degree: The number of attributes in the relation is known as degree.

Ex: The degree of the given STUDENT table is 5.

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

Ex:

ROLL_NO
1

Null values: The value which is not known or unavailable is called NULL VALUE. It is represented by blank space.

Cardinality: The number of tuples are present in the relation is called as its cardinality.

Ex: The Cardinality of the STUDENT table is 4.

2.3 Concept Of Domain

The domain of a database is the set of all allowable values (or) attributes of the database.

Ex: Gender (Male, Female, Others).

Relation

- A relation is defined as a set of tuples and attributes.
- A relation consists of Relation schema and relation instance.
- Relation schema: A relation schema represents the name of the relation with its attributes. Ex: STUDENT (ROLL_NO, NAME, ADDRESS, PHONE and AGE) is Relation schema for STUDENT.
- Relation instance: The set of tuples of a relation at a particular instance of a time is called Relation Instance.

An instance of „Employee „relation

Emp_code	**Emp_Name**	**Dept_Name**
01234	John	HR
12567	Smith	Sales
21678	Sai	Production
12456	Jay	Design

2.4 Importance of Null values:

- SQL supports a special value known as NULL which is used to represent the values of attributes that may be unknown or not apply to a tuple.
- For example, the apartment_number attribute of an address applies only to the address that is in apartment buildings and not to other types of residences.
- It is important to understand that a NULL value is different from Zero value.
- A Null value is used to represent a missing value, but that is usually has one of the following interpretations:
  - Value unknown (Value exists but it is unknown)
  - Value not available (exists but it is purposely withheld)
  - Attribute not applicable (undefined for this tuple)
- It is often not possible to determine which of the meanings is intended.

2.5 Constraints

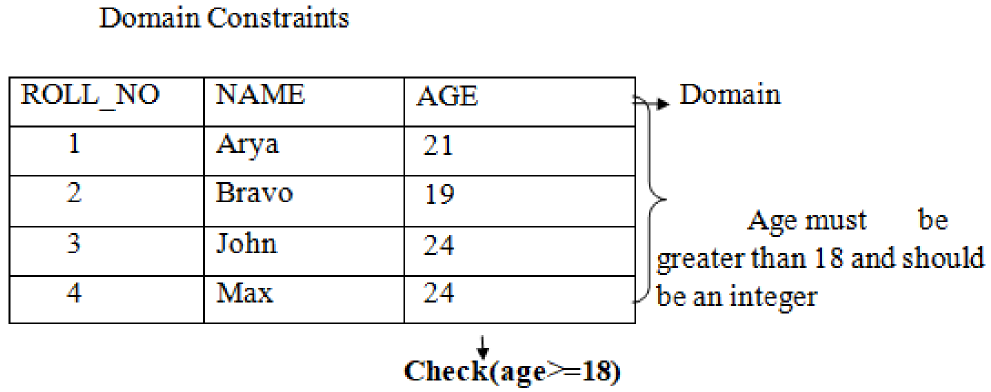
- On modeling the design of the relational data base, we can put some rules(conditions) like what values are allowed to be inserted in the relation
- Constraints are the rules enforced on the data columns of a table. These are used to limit the type of data that can go in to a table
- This Ensure the accuracy and reliability of the data in the database. Constraints could be either on a column level on a table level.

2.6 Domain Constraints In DBMS

- In DBMS table is viewed as a combination of rows and columns
- For **example,** if you are having a column called month and you want only (jan, feb, march……) as values allowed to be entered for that particular column which is referred to as domain for that particular column

Definition: Domain constraint ensures two things it makes sure that the data value entered for that particular column matches with the data type defined by that column

It shows that the constraints (NOT NULL/UNIQUE/PRIMARY KEY/FOREIGN KEY/CHECK/DEFAULT)



Domain Constraints

ROLL_NO	NAME	AGE
1	Arya	21
2	Bravo	19
3	John	24
4	Max	24

Domain
Age must be greater than 18 and should be an integer
Check(age>=18)

Domain constraint= data type check for the column +constraints.

Example:, we want to create a table “STUDENT” with “stu_id” field having a value greater than 100, can create a domain and table like this.

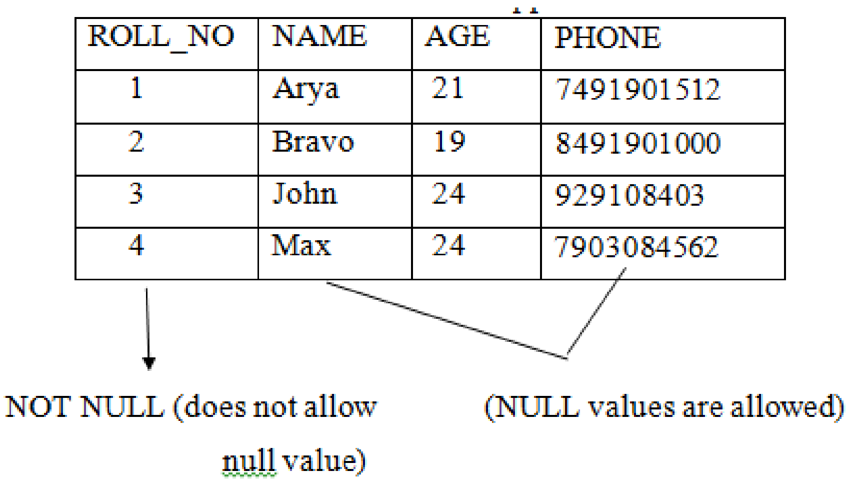
- Create domain id_value int constraint id_test check (value>=100);
- CREATE table STUDENT (stu_id id value primary key, stu_name varchar (30), stu_age int);

2.7 Key constraints in DBMS:

- Constraints are nothing but the rules that are to be followed while entering data into columns of the database table.
- Constraints ensure that the data entered by the user into columns must be within the criteria specified by the condition.
- We have 6 types of key constraints in DBMS
    1. Not Null
    2. Unique
    3. Default
    4. Check
    5. Primary key
    6. Foreign key

1. Not Null:

- Null represents a record where data may be missing data or data for that record may be optional.
- Once not null is applied to a particular column, you cannot enter null values to that column.
- A not null constraint cannot be applied at table level.



ROLL_NO	NAME	AGE	PHONE
1	Arya	21	7491901512
2	Bravo	19	8491901000
3	John	24	929108403
4	Max	24	7903084562

NOT NULL (does not allow null value)
(NULL values are allowed)

Example:

Create table EMPLOYEE (id int Not null, name varchar Not null, Age int not null, address char (25), salary decimal (18,2), primary key(id));

- In the above example we have applied not null on three columns id, name and age which means whenever a record is entered using insert statement all three columns should contain a value other than null.
- We have two other columns address and salary, where not null is not applied which means that you can leave the row as empty.

2. Unique:

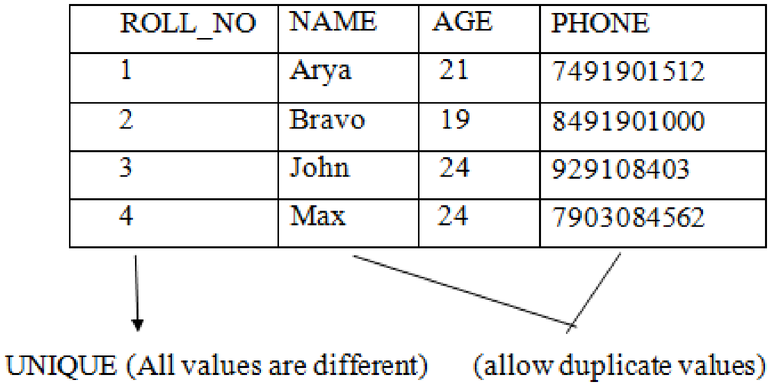
Some times we need to maintain only. Unique data in the column of a database table, this is possible by using a Unique constraint.

- Unique constraint ensures that all values in a column are Unique.

Example:

Create table PERSONS (id int unique, last_name varchar (25) not null, First name varchar (25), age int);

- In the above example, as we have used unique constraint on ID column we are not supposed to enter the data that is already present, simply no two ID values are same.

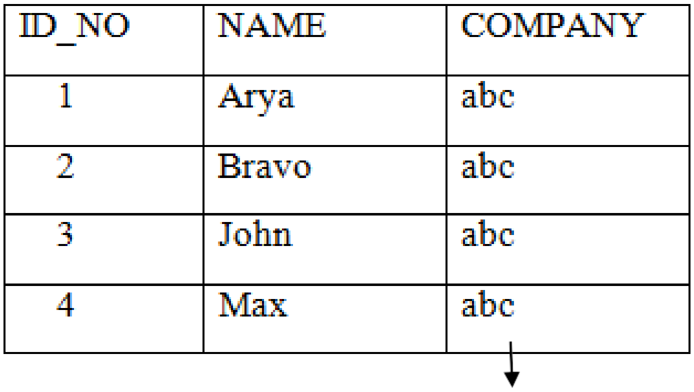


ROLL_NO	NAME	AGE	PHONE
1	Arya	21	7491901512
2	Bravo	19	8491901000
3	John	24	929108403
4	Max	24	7903084562

3. Default:

Default in SQL is used to add default data to the columns.

- When a column is specified as default with same value then all the rows will use the same value i.e., each and every time while entering the data we need not enter that value.
- But default column value can be customised i.e., it can be over ridden when inserting a data for that row based on the requirement.



ID_NO	NAME	COMPANY
1	Arya	abc
2	Bravo	abc
3	John	abc
4	Max	abc

(Row with default values “abc”)

Example:

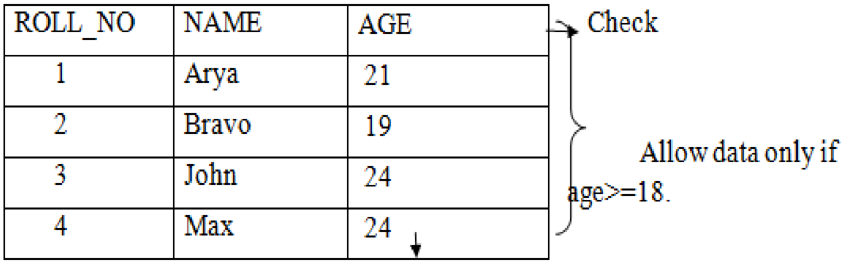
Create table EMPLOYEE (id int Not null, last_name varchar (25) Not null, first_name varchar (25), Age int, city varchar (25) Default Hyderabad);

- As a result, whenever you insert a new row each time you need not enter a value for this default column that is entering a column value for a default column is optional.

4. Check:

- Check constraint ensures that the data entered by the user for that column is within the range of values or possible values specified.

Example: ***Create table STUDENT (id int, name varchar (25), age int, check(age>=18));***



ROLL_NO	NAME	AGE
1	Arya	21
2	Bravo	19
3	John	24
4	Max	24

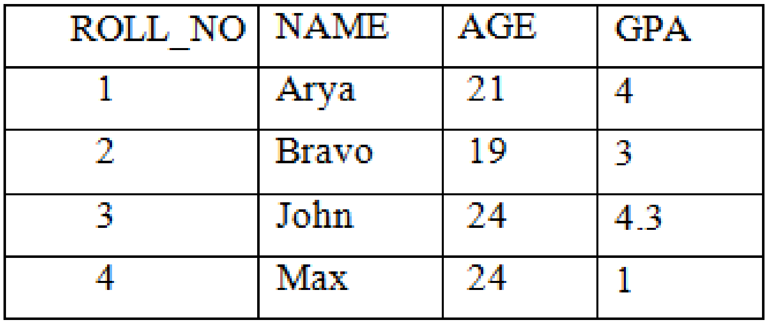
Check
Allow data only if age>=18.

Check(age>=18)

- As we have used a check constraint as (age>=18) which means value entered by user for this age column while inserting the data must be less than or equal to 18.

5. Primary Key:

- A primary key is a constraint in a table which uniquely identifies each row record in a database table by enabling one or more column in the table as primary key.



ROLL_NO	NAME	AGE	GPA
1	Arya	21	4
2	Bravo	19	3
3	John	24	4.3
4	Max	24	1

Creating a primary key:

- A particular column is made as a primary key column by using the primary key keyword followed by the column name.

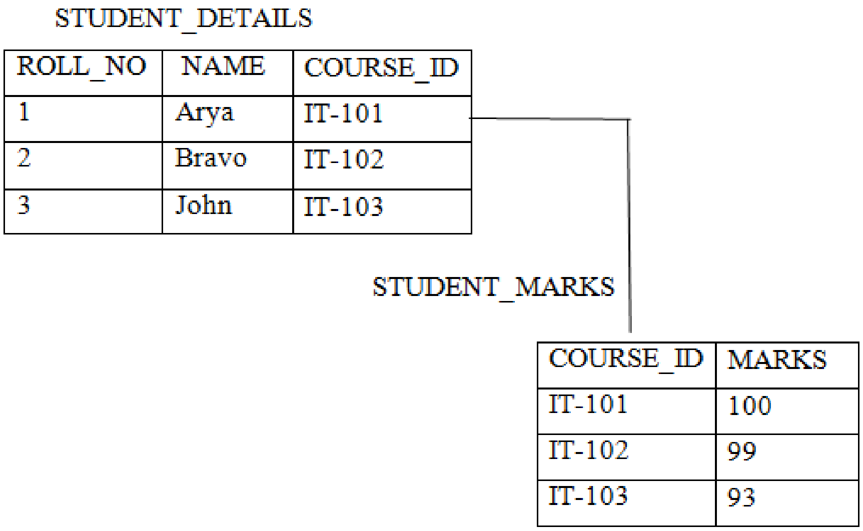
Example:

Create table EMP (ID int, name varchar (20), age int, course varchar (10), Primary key (ID));

- Here we have used the primary key on ID column then ID column must contain unique values i.e., one ID cannot be used for another student.

6. Foreign Key:

- The foreign key constraint is a column or list of columns which points to the primary key column of another table.
- The main purpose of the foreign key is only those values are allowed in the present table that will match to the primary key column of another table.



STUDENT_DETAILS

ROLL_NO	NAME	COURSE_ID
1	Arya	IT-101
2	Bravo	IT-102
3	John	IT-103

STUDENT_MARKS

COURSE_ID	MARKS
IT-101	100
IT-102	99
IT-103	93

From the above two tables, COURSE_ID is a primary key of the table STUDENT_MARKS and also behaves as a foreign key as it is same in STUDENT_DETAILS and STUDENT_MARKS.

Example:

(Reference Table)

Create table CUSTOMER1 (id int, name varchar (25), course varchar (10), primary key (ID));

(Child table)

Create table CUSTOMER2 (id int, marks int, references customer1(ID));

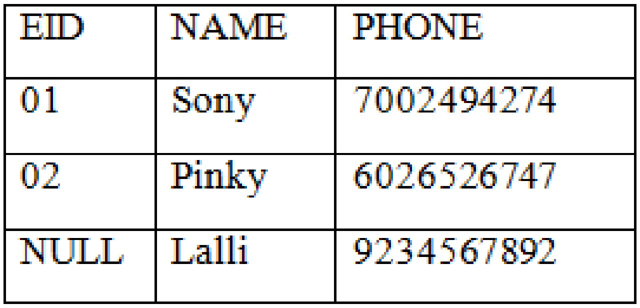
2.8 Integrity Constraints in DBMS:

- There are two types of integrity constraints
  1. Entity Integrity Constraints
  2. Referential Integrity Constraints

Entity Integrity constraints:

- These constraints are used to ensure the uniqueness of each record or row in the data table.
- Entity Integrity constraints says that no primary key can take NULL VALUE, since using primary key we identify each tuple uniquely in a relation.

Example:



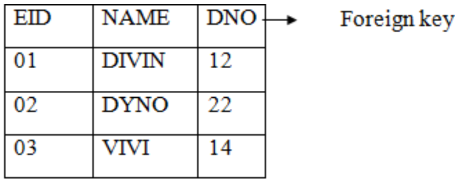
EID	NAME	PHONE
01	Sony	7002494274
02	Pinky	6026526747
NULL	Lalli	9234567892

Explanation:

- In the above relation, EID is made primary key, and the primary key can‟t take NULL values but in the 3rd tuple, the primary key is NULL, so it is violating Entity integrity constraints.

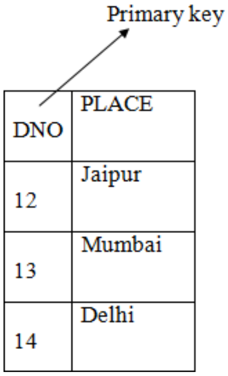
Referential Integrity constraints:

- The referential integrity constraint is specified between two relations or tables and used to maintain the consistency among the tuples in two relations.
- This constraint is enforced through foreign key, when an attribute in the foreign key of relation R1 have the same domain as primary key of relation R2, then the foreign key of R1 is said to reference or refer to the primary key of relation R2.
- The values of the foreign key in a tuple of relation R1 can either take the values of the primary key for some tuple in Relation R2, or can take NULL values, but can‟t be empty.



EID	NAME	DNO
01	DIVIN	12
02	DYNO	22
03	VIVI	14

Foreign key



Primary key

DNO	PLACE
12	Jaipur
13	Mumbai
14	Delhi

Explanation:

- In the above, DNO of the first relation is the foreign key and DNO in the second relation is the primary key
- DNO=22 in the foreign key of the first relation is not available in the second relation so, since DNO=22 is not defined in the primary key of the second relation therefore Referential integrity constraints is violated here.

2.9 Basic SQL (introduction)

- SQL stands for Structure Query Language it is used for storing and managing data in relational database management system.
- It is standard language for relational database system. It enables a user to create, read, update and delete relational databases and tables.
- All the RDBMS like MYSQL, Oracle, MA access and SQL Server use SQL as their standard database language.
- SQL allows users to Query the database in a number of ways using statements like common English.

Rules: SQL follows following rules

- SQL is not a case sensitive. Generally, keywords are represented in UPPERCASE.
- Using the SQL statements, you can perform most of the actions in a database.
- Statements of SQL are dependent on text lines. We can use a single SQL statement on one or multiple text line.

2.10 SQL Process:

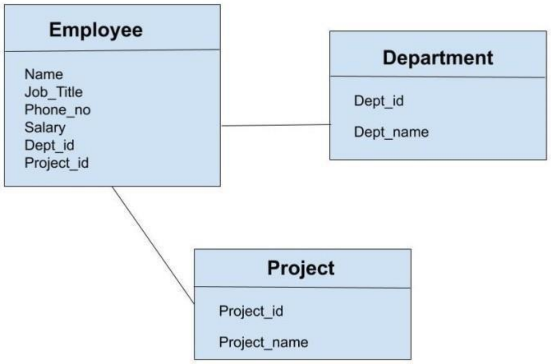
- When an SQL command is executing for any RDBMS, then the system figure out the best way to carry out the request and the sql engine determines that how to interrupt the task.
- In the process, various components are included. These components can be optimization engine, query engine, query dispatcher etc.,
- All the non-sql queries are handled by the classic query engine, but sql query engine won"t handle logical files.

2.11 Characteristics of SQL:

- SQL is easy to learn.
- SQL is used to access data from relational database management system.
- SQL is used to describe the data.
- SQL is used to create and drop the database and table.
- SQL allows users to set permissions on tables, procedures and views.

2.12 Simple database Schema:

- A database schema is a structure that represents the logical storage of the data in the database.
- It represents the organization of data and provides information about the relationships between the tables in a given database.
- A database schema is the logical representation of a database, which shows how the data is stored logically in the entire database.
- It contains list of attributes and instruction that informs the database engine that how the data is organized and how the elements are related to each other.
- A database schema contains schema objects that may include tables, fields, packages, views, relationship, primary key, foreign key.
- In actual, the data is physically stored in files that may be in unstructured form, but to retrieve it and use it, we need to keep them in a structured manner. To do this a database schema is used. It provides knowledge about how the data is organized in a database and how it is associated with other data.
- A database schema object includes the following:
  - Consistent formatting for all data entries.
  - Database objects and unique keys for all data entries.
  - Tables with multiple columns, and each column contains its names and datatypes.
  - The given diagram is an example of a database schema it contains three tables, their data types. This also represents the relationships between the tables and primary keys as well as foreign keys.

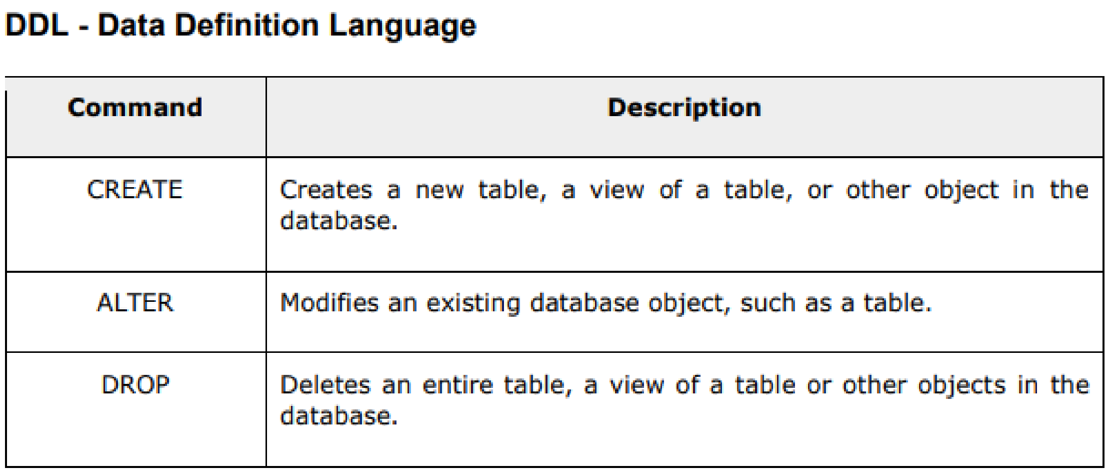


Schema

2.13 SQL Commands:

SQL commands are categorized into three types.

1. **Data Definition Language (DDL):** used to create (define) a table.
2. **Data Manipulation Language (DML):** used to update, store and retrieve data from tables.
3. **Data Control Language (DCL):** used to control the access of database created using DDL and DML.



DDL - Data Definition Language

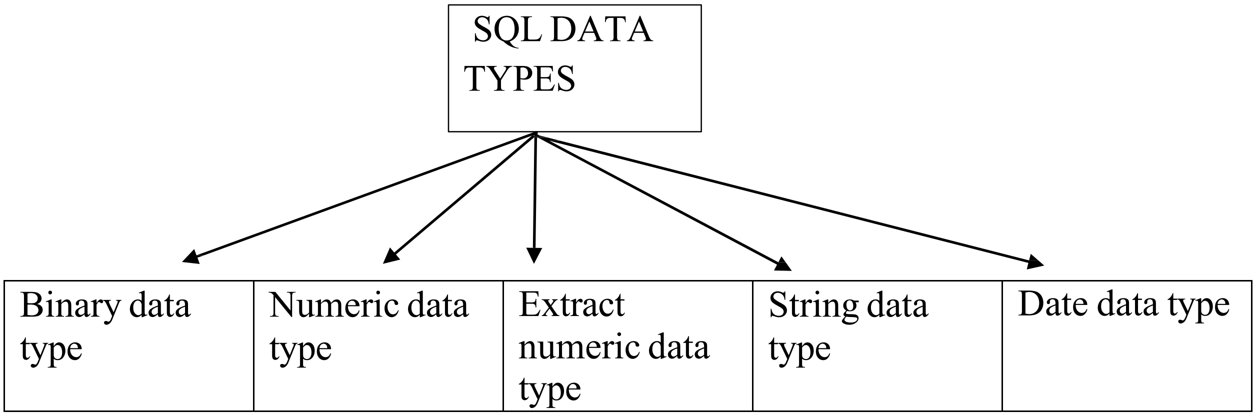
Command	Description
CREATE	Creates a new table, a view of a table, or other object in the database.
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other objects in the database.

2.14 SQL DATATYPES :

SQL data type is used to define the values that a column can contain

Every column is required to have a name and data type in the database table. DATA

TYPES OF SQL :



1. BINARY DATATYPES:

There are three types of binary data types which are given below

DATA TYPE	DESCRIPTION
Binary	It has a maximum length of 800 bytes. It contains a fixed- length binary data
Var binary	It has a maximum length of 800 bytes. It contains a variable - length binary data
Image	It has a maximum length of 2,147,483,647 bytes. It contains a variable - length binary data

2. NUMERIC DATATYPE:

DATA TYPE	FROM	TO	DESCRIPTION
Float	-1.79 E +308	1.79 E +308	It is used to specify a floating-point value. Ex: 6.2, 2.9 etc
Real	-3.40 E +38	3.40 E +38	It specifies a single precision floating point number.

3. EXACT NUMERIC DATA TYPE:

DATA TYPE	DESCCRIPTION
Int	It is used to specify an integer value
Small int	It is used to specify small integer value
Bit	It has the number of bits to store
Decimal	It specifies a numeric value that can have a decimal number
Numeric	It is used to specify a numeric value

4. DATE AND TIME DATATYPES:

DATA TYPE	DESCRIPTION
Date	It is used to store the year, month, and days value
Time	It is used to store the hour, minute, and seconds value
Time stamp	It stores the year, month, hour, minute, and the second value

5. STRING DATATYPE:

DATA TYPE	DESCRIPTION
Char	It has a maximum length of 8000 characters. It contains fixed-length non-Unicode characters.
Varchar	It has a maximum length of 8000 characters. It contains variable-length non-Unicode characters.
Text	It has a maximum length of 2,147,483,647 characters. It contains variable-length non-Unicode characters.

2.15 TABLE DEFINITIONS: (CREATE, ALTER)

SQL TABLE: SQL table is a collection of data which is organized in terms of rows and columns.

- In DBMS, the table is known as relation and row as a tuple
- Let"s see an example of the "EMPLOYEE "table

EMP_ID	EMP_NAME	CITY	PHONE_ID
1	Kristen	Washington	7289201223
2	Anna	Franklin	9378282882
3	Jackson	California	9264783838
4	Daniel	Hawaii	9638482678

- In the above table, "EMPLOYEE" is the table name, "EMP_ID, "EMP_NAME", "CITY"," PHONE-NO" are the column names.
- The combination of data of multiple columns forms a row

EG: 1, "Kristen", "Washington" and "7289201223 "are the data of one row

2.16 OPERATIONS ON TABLE:

1. Create table
2. Alter table
3. Drop table

1. **Create table:** SQL create table is used to create a table in the database. To define the table, you should define the name of the table and also define its column and column"s data type.

SYNTAX:

```
Create table table_name ("column1" "datatype",
                         "column2" "datatype", "column3"
                         "datatype",
                                   ….
                            "column N" "datatype");
```

EXAMPLE:

SQL > **create table employee (emp_id int, emp_name varchar (25), phone_no int, address char (30));**

- If you create the table successfully, you can verify the table by looking at the message by the sql server. else you can use DESC command as follows

SQL > DESC employee;

FIELD	TYPE	NULL	DEFAULT	EXTRA
Emp_id	Int (11)	No	NULL	
Emp_name	Varchar (25)	No	NULL	
Phone_no	No	Int (11)	NULL	
address	yes		NULL	Char(30)

2. ALTER TABLE:

- The alter table command adds, delete or modifies columns in a table
- The alter table command also adds and deletes various constraints in a table
- The following SQL adds an "EMAIL" column to the "EMPLOYEE "table

SYNTAX:

ALTER table table_name add column1 datatype;

EXAMPLE:

ALTER table employee add email varchar (255);

SQL > DESC employee;

FIELD	TYPE	NULL	DEFAULT	EXTRA
Emp_id	Int (11)	No	NULL	
Emp_name	Varchar (25)	No	NULL	
Phone_no	No	Int (11)	NULL	
Address	Yes		NULL	Char (30)
Email	Varchar (255)		NULL	

3. DROP TABLE:

- The drop table command deletes a table in the data base
- The following example SQL deletes the table "EMPLOYEE"

SYNTAX :

DROP table table_name;

EXAMPLE:

DROP table employee;

- Dropping a table results in loss of all information stored in the table.

2.17 Different DML Operations (insert, delete, update):

- DML-Data Manipulation Language.
- Data Manipulation Commands are used to manipulate data to the database.
- Some of the data manipulation commands are
  1. Insert
  2. Update
  3. Delete

1. Insert:

SQL insert statement is a sql query. It is used to insert a single multiple records in a table.

Syntax:

Insert into table name values (value 1, value 2, value 3);

Let‟s take an example of table which has 3 records within it.

- insert into student values(„alekhya‟,501,‟hyderabad‟);
- insert into student values(„deepti‟,502,‟guntur‟);
- insert into student values(„ramya‟,503,‟nellore‟);

The following table will be as follows:

NAME	ID	CITY
Alekhya	501	Hyderabad
Deepti	502	Guntur
Ramya	503	Nellore

2. Update:

- The SQL Commands update are used to modify the data that is already in the database.
- SQL Update statement is used to change the data of records held by tables which rows is to be update, it is decided by condition to specify condition, we use “WHERE” clause.
    - The update statement can be written in following form:

Syntax:

Update table_name set column_name=expression where condition;

Example:

Let‟s take an example: here we are going to update an entry in the table.

Update students set name=‟rasi‟ where id=503;

After update the table is as follows:

NAME	ID	CITY
Alekhya	501	Hyderabad
Deepti	502	Guntur
Rasi	503	Nellore

3. Delete:

- The SQL delete statement is used to delete rows from a table.
- Generally, delete statement removes one or more records from a table.

Syntax:

delete from table_name [where condition];

Example:

Let us take a table named “student” table

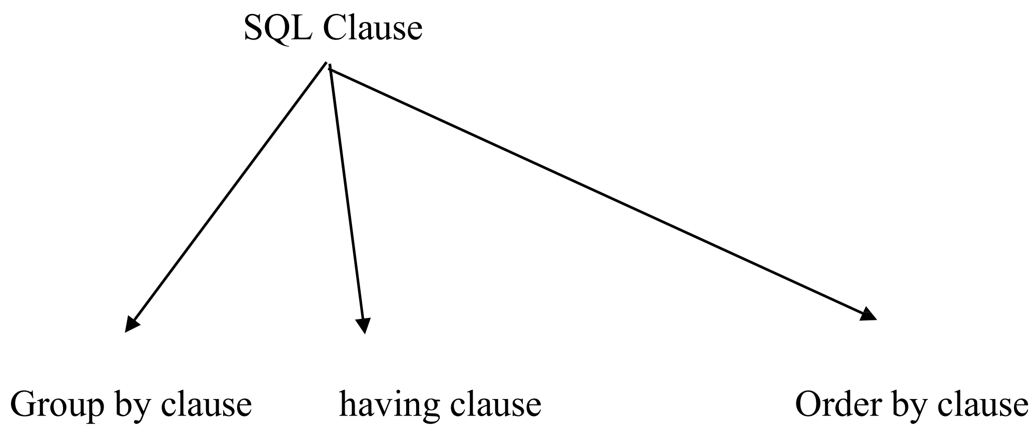
Delete from students where id=501;

Resulting table after the query:

NAME	ID	CITY
Deepti	502	Guntur
Rasi	503	Nellore

2.18 Basic SQL querying (select and project) using where clause:

- The following are the various SQL clauses:



1. Group by:

- SQL group by statement is used to arrange identical data into groups.
- The group by statement is used with the SQL select statement.
- The group by statement follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

Syntax:

Select column from table_name where column group by column, order by column;

Sample table: product

PRODUCT	COMPANY	QTY	RATE	COST
Item 1	Com 1	2	10	20
Item 2	Com 2	3	25	75
Item 3	Com 1	2	30	60
Item 4	Com 3	5	10	50
Item 5	Com 2	2	20	40

Example:

- Select company count (*) from product group by company;

Output:

Com 1	2
Com 2	3
Com 3	5

2. Having clause:

- Having clause is used to specify a search condition for a group or an aggregate.

Having clause is used in a group by clause, if you are not using group by clause then you can use having function like a where clause.

Syntax:

Select column1, column2 from table_name

Where conditions

Group by column1, column2 Having

conditions

Order by column1, column2;

Example:

- **select company count (*) from product**

Group by company Having

count (*) > 2;

Output:

Com 3	5
Com 2	2

3. Order by clause:

The order by clause sorts the result _set in ascending or descending order.

Syntax:

Select column1, column2, from table_name

Where condition

Order by column1, column2…asc;

Sample table:

Take a student table

Example:

Select * from student order by name;

Output:

NAME	ID	CITY
Alekhya	501	Hyderabad
Deepti	502	Guntur
Rasi	503	Nellore

2.19 SQL Where clause:

- A where clause in SQL is a data manipulation language statement.
- Where clauses are not mandatory clauses of SQL DML statements but it can be used to limit the number of rows affected by a SQL DML statement or returned by query.
- Actually, it follows the records.it returns only those queries which the specific conditions.

Syntax:

Select column1, column2, …………column from table_name where[condition];

- Where clause uses same conditional selection.

=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

2.20 Arithmetic and logical operations:

SQL operators:

- SQL statements generally contain some reserved words or characters that are used to perform operations such as arithmetic and logical operations etc. Their reserved words are known as operators.

SQL arithmetic operator:

- We can use various arithmetic operators on the data stored in tables.
- Arithmetic operators are:

+	Addition
-	Subtraction
/	Division
*	Multiplication
%	modulus

1. **Addition (+):**
   It is used to perform addition operation on data items.

Sample table:

EMP_ID	EMP_NAME	SALARY
1	Alex	25000
2	John	55000
3	Daniel	52000
4	Sam	12312

- select emp id, emp_name, salary, salary+100 as “salary +100” from addition;

Output:

EMP_ID	EMP_NAME	SALARY	SALARY+100
1	Alex	25000	25100
2	John	55000	55100
3	Daniel	52000	52100
4	Sam	12312	12412

- Here we have done addition of 100 to each emp‟s salary.

2. **Subtraction (-):**

- It is used to perform subtraction on the data items.

Example:

Select emp_id, emp_name, salary, salary-100 as “salary-100” from subtraction;

EMP_ID	EMP_NAME	SALARY	SALARY-100
1	Alex	25000	24900
2	John	55000	54900
3	Daniel	52000	51900
4	Sam	90000	89900

Here we have done subtraction of 100 for each emp‟s salary.

3. **Division (/):**

- The division function is used to integer division (x is divided by y).an integer value is returned.

Example:

- **Select emp_id, emp_name, salary, salary/100 as “salary/100” from division;**

EMP_ID	EMP_NAME	SALARY	Salary/100
1	Alex	25000	250
2	John	55000	550
3	Daniel	52000	520
4	Sam	90000	900

4. Multiplication (*):

- It is used to perform multiplication of data items.
- **Select emp_id, emp_name, salary, salary*100 as "salary*100" from multiplication;**

EMP_ID	EMP_NAME	SALARY	SALARY*100
1	Alex	25000	2,500,000
2	John	55000	5,500,000
3	Daniel	52000	5,200,000
4	Sam	90000	9,000,000

- Here we have done multiplication of 100 to each emp"s salary.

5. Modulus (%):

- It is used to get remainder when one data is divided by another.
- **Select emp_id, emp_name, salary, salary%25000 as "salary%25000" from modulus;**

Output:

EMP_ID	EMP_NAME	SALARY	SALARY%25000
1	Alex	25000	0
2	John	55000	5000
3	Daniel	52000	2000
4	Sam	90000	15000

- Here we have done modulus operation to each emp"s salary.

2.21 Logical operations:

- Logical operations allow you to test for the truth of a condition.
- The following table illustrates the SQL logical operator.

OPERATOR	MEANING
ALL	Returns true if all comparisons are true
AND	Returns true if both expressions are true
ANY	Returns true if any one of the comparisons is true
BETWEEN	Return true if the operand is within a range
IN	Return true if the operand is equal to one of the values in a list
EXISTS	Return true if the sub query contains any rows

1. AND:

The AND operator allows you to construct multiple condition in the WHERE clause of an SQL statement such as select.

- The following example finds all employees where salaries are greater than the 5000 and less than 7000.
- **Select first_name, last_name, salary from employees where salary>5000 AND salary<7000 order by salary;**

Output:

FIRST_NAME	LAST_NAME	SALARY
John	Wesley	6000
Eden	Daniel	6000
Luis	Popp	6900
Shanta	Suji	6500

2. ALL:

The ALL operator compares a value to all values in another value set.

- The following example finds all employees whose salaries are greater than all salaries of employees.

EX:

```
select first_name, last_name, salary from employees where salary>=ALL (select salary from employees
       where department_id =8) order by salary DESC;
```

Output:

FIRST_NAME	LAST_NAME	SALARY
Steven	King	24000
John	Russel	17000
Neena	Kochhar	14000

3. ANY:

The ANY operator compares a value to any value in a set ascending to condition.

The following example statement finds all employees whose salaries are greater than the average salary of every department.

EX:

```
select first_name, last_name, salary from employees where salary >ANY (select avg (salary) from
employees“ group by department_id) order by first_name, last_name;
```

Output:

FIRST_NAME	LAST_NAME	SALARY
Alexander	Hunold	9000.00
Charles	Johnson	6200.00
David	Austin	4800.00
Eden	Flip	9000.00

4. Between:

- The between operator searches for values that are within a set of values.
- For example, the following statement finds all employees where salaries are between 9000 and 12000.

EX:

```
select first_name, last_name, salary from employees where salary between 9000 AND 12000
order by salary;
```

Output:

FIRST_NAME	LAST_NAME	SALARY
Alexander	Hunold	9000.00
Den	Richards	10000.00
Nancy	Prince	12000.00

5. IN:

- The IN operator compares a value to list of specified values. The IN operator return true if compared value matches at least one value in the list.
- The following statement finds all employees who work in department _id 8 or 9. EX:

select first_name, last_name, department_id from employees where department_id IN (8,9) order by department_id;

Output:

FIRST_NAME	LAST_NAME	DEPARTMENT_ID
John	Russel	8
Jack	Livingstone	8
Steven	King	9
Neena	Kochhar	9

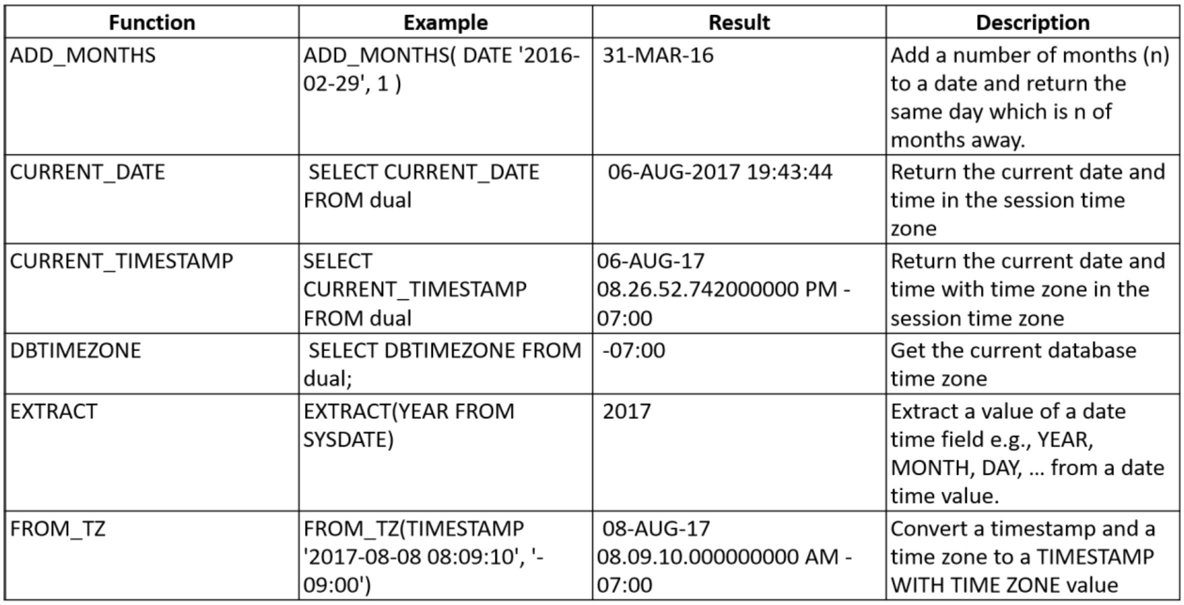
6. Exists:

- The EXISTS operator tests if a sub query contains any rows.
- For example, the following statement finds all employees who have dependents.
- select first_name, last_name from employees where EXISTS (select 1 from dependent d where d.employee_id=e.employee_id);

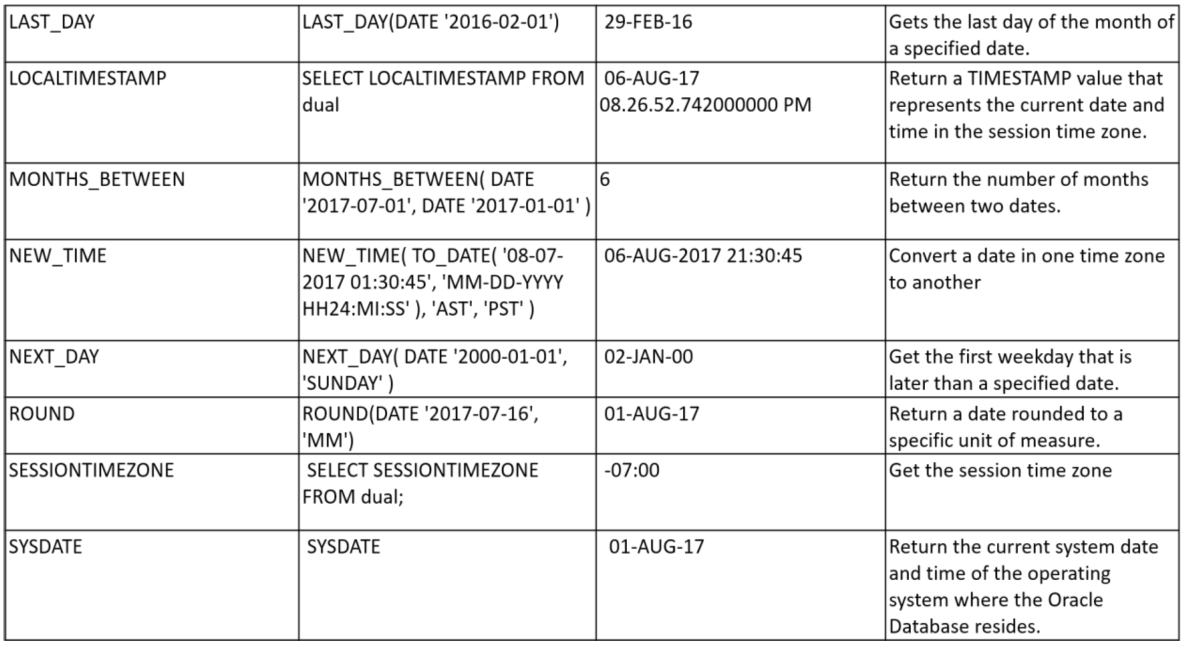
FIRST_NAME	LAST_NAME
Steven	King
Neena	Kochhar
Alexander	Hunold

2.22 SQL FUNCTIONS (Date & Time, Numeric, Aggregate, String conversions):

DATE & TIME FUNCTIONS:



Function	Example	Result	Description
ADD_MONTHS	ADD_MONTHS(DATE '2016-02-29', 1)	31-MAR-16	Add a number of months (n) to a date and return the same day which is n of months away.
CURRENT_DATE	SELECT CURRENT_DATE FROM dual	06-AUG-2017 19:43:44	Return the current date and time in the session time zone
CURRENT_TIMESTAMP	SELECT CURRENT_TIMESTAMP FROM dual	06-AUG-17 08.26.52.742000000 PM -07:00	Return the current date and time with time zone in the session time zone
DBTIMEZONE	SELECT DBTIMEZONE FROM dual;	-07:00	Get the current database time zone
EXTRACT	EXTRACT(YEAR FROM SYSDATE)	2017	Extract a value of a date time field e.g., YEAR, MONTH, DAY, … from a date time value.
FROM_TZ	FROM_TZ(TIMESTAMP '2017-08-08 08:09:10', '-09:00')	08-AUG-17 08.09.10.000000000 AM -07:00	Convert a timestamp and a time zone to a TIMESTAMP WITH TIME ZONE value



LAST_DAY	LAST_DAY(DATE '2016-02-01')	29-FEB-16	Gets the last day of the month of a specified date.
LOCALTIMESTAMP	SELECT LOCALTIMESTAMP FROM dual	06-AUG-17 08.26.52.742000000 PM	Return a TIMESTAMP value that represents the current date and time in the session time zone.
MONTHS_BETWEEN	MONTHS_BETWEEN(DATE '2017-07-01', DATE '2017-01-01')	6	Return the number of months between two dates.
NEW_TIME	NEW_TIME(TO_DATE('08-07-2017 01:30:45', 'MM-DD-YYYY HH24:MI:SS'), 'AST', 'PST')	06-AUG-2017 21:30:45	Convert a date in one time zone to another
NEXT_DAY	NEXT_DAY(DATE '2000-01-01', 'SUNDAY')	02-JAN-00	Get the first weekday that is later than a specified date.
ROUND	ROUND(DATE '2017-07-16', 'MM')	01-AUG-17	Return a date rounded to a specific unit of measure.
SESSIONTIMEZONE	SELECT SESSIONTIMEZONE FROM dual;	-07:00	Get the session time zone
SYSDATE	SYSDATE	01-AUG-17	Return the current system date and time of the operating system where the Oracle Database resides.

SYSTIMESTAMP	SELECT SYSTIMESTAMP FROM dual;	01-AUG-17 01.33.57.929000000 PM -07:00	Return the system date and time that includes fractional seconds and time zone.
TO_CHAR	TO_CHAR(DATE'2017-01-01', 'DL')	Sunday, January 01, 2017	Convert a DATE or an INTERVAL value to a character string in a specified format.
TO_DATE	TO_DATE('01 Jan 2017', 'DD MON YYYY')	01-JAN-17	Convert a date which is in the character string to a DATE value.
TRUNC	TRUNC(DATE '2017-07-16', 'MM')	01-JUL-17	Return a date truncated to a specific unit of measure.
TZ_OFFSET	TZ_OFFSET('Europe/London')	+01:00	Get time zone offset of a time zone name from UTC

Some important date and time functions are below:

Sysdate: It generates the system date**. Ex:**

Select sysdate from dual;

Output: 05-DEC-2021.

ADD_MONTHS: This function returns a date after adding data with specified no of months. EX:

Select ADD_MONTHS („2017-02-29‟,1) from dual;

Output: 31-MAR-17.

Select add_months(sysdate,3) from dual;

Output: 05-MAR-22.

CURRENT_DATE: This function displays the current date.

Ex: Select CURRENT_DATE from dual;

Output: 05-DEC-2021.

NEXT_DAY: This function represents both day and date and returns the day of the next given day.

EX: **Select NEXT_DAY(SYSDATE,‟MONDAY‟) from dual;**

Output: 07-DEC-21.

LAST_DAY: This function returns a day corresponding last day of months.

EX: **Select LAST_DAY (sysdate) from dual;**

Output: 31-DEC-21.

MONTHS_BETWEEN: It is used to find no of months between two given dates.

EX: **Select MONTHS_BETWEEN(‟16-APRIL-2021‟,‟16-AUGUST-2021) from dual;**

Output: -4.

ROUND: It gives the nearest value or round off value for the argument pass. (or) It returns a date rounded to a specific unit of measure.

EX: **Select ROUND(‟26-NOV-21‟,‟YYYY‟) from dual;**

Output: 01-JAN-22.

TRUNC: This function returns the date with the time(co-efficient) portion of the date truncated to the unit specified.

EX**: Select TRUNC (sysdate, ‟MM‟) from dual;**

Output: 01-DEC-21.

TO_DATE: This function converts date which is in the character string to a date value.

EX: **Select TO_DATE (‟01 jan 2017‟,‟DD MM YYYY‟) from dual;**

Output: 01-JAN-17.

TO_CHAR: This function converts DATE or an INTERVAL value to a character string in a specified format.

EX: **Select TO_CHAR (sysdate,‟DD MM YYYY‟) from dual;**

Output: 05 12 2021.

LEAST: This function displays the oldest date present in the argument list.

EX: **Select LEAST(‟01-march-2021‟,‟16-feb-2021‟,‟28-dec-2021‟) from dual;**

Output: 01-MAR-21.

GREATEST: This function displays the latest date present in the argument list.

EX: **Select GREATEST (‟01-march-2021‟,‟16-feb-2021‟,‟28-dec-2021‟) from dual;**

Output: 28-DEC-21.

2.23 Aggregate Functions:

Aggregate Functions take a collection of values as input and returns a single value.

1. Count ()
2. Sum ()
3. Avg ()

4. Max ()

5. Min ()

1. **Count ():** This function returns number of rows returned by a query.

Syntax: Select count(column_name)

From table_name Where

condition);

Example: Select count (distinct manager_id) from employees;

2. **Sum ():** It will add/ sum all the column values in the query.

Syntax: Select sum (column_name)

From table_name

Where condition);

Example: Select sum(salaries) from employees;

3. **Avg ():** Avg function used to calculate average values of the set of rows.

Syntax: Select avg (column_name)

From table_name Where

condition);

Example: Select avg(salary) from employees;

4. **Max ():** This function is used to find maximum value from the set of values.

Syntax: Select max (column_name)

From table_name

Where condition);

Example: Select max (salary) from employees;

5. **Min ():** This function is used to find minimum value from the set of values.

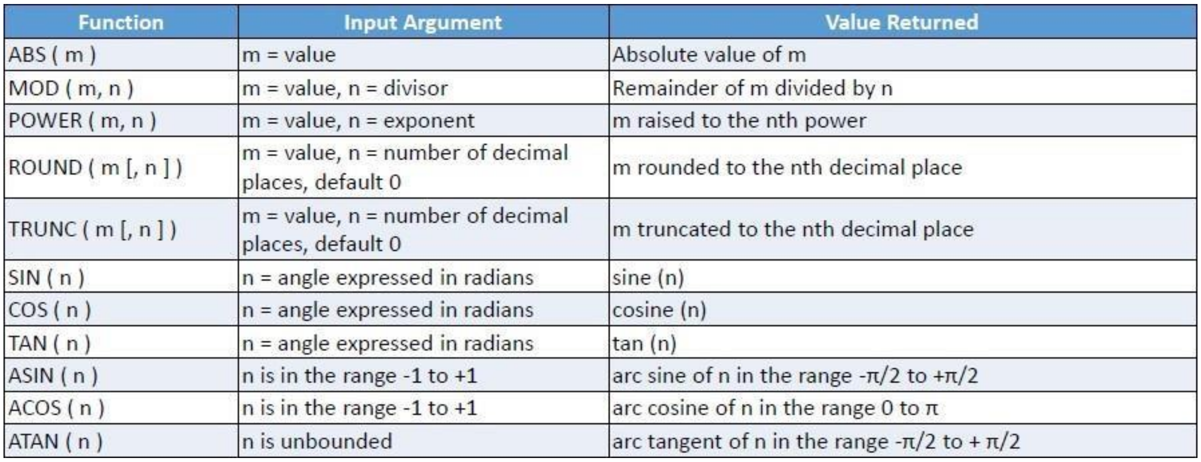
Syntax: Select min (column_name)

From table_name

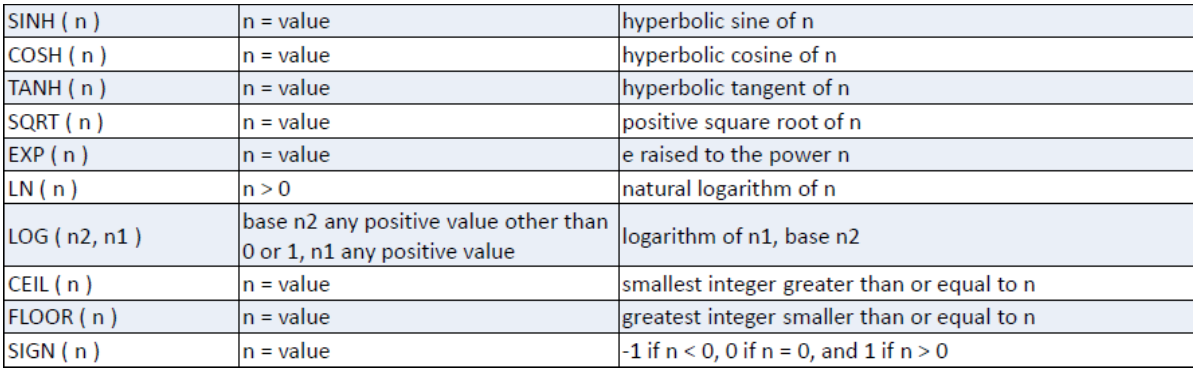
Where condition);

Example: Select min (salary) from employees;

2.24 SQL NUMERIC FUNCTIONS:



Function	Input Argument	Value Returned
ABS (m)	m = value	Absolute value of m
MOD (m, n)	m = value, n = divisor	Remainder of m divided by n
POWER (m, n)	m = value, n = exponent	m raised to the nth power
ROUND (m [, n])	m = value, n = number of decimal places, default 0	m rounded to the nth decimal place
TRUNC (m [, n])	m = value, n = number of decimal places, default 0	m truncated to the nth decimal place
SIN (n)	n = angle expressed in radians	sine (n)
COS (n)	n = angle expressed in radians	cosine (n)
TAN (n)	n = angle expressed in radians	tan (n)
ASIN (n)	n is in the range -1 to +1	arc sine of n in the range -π/2 to +π/2
ACOS (n)	n is in the range -1 to +1	arc cosine of n in the range 0 to π
ATAN (n)	n is unbounded	arc tangent of n in the range -π/2 to + π/2



SINH (n)	n = value	hyperbolic sine of n
COSH (n)	n = value	hyperbolic cosine of n
TANH (n)	n = value	hyperbolic tangent of n
SQRT (n)	n = value	positive square root of n
EXP (n)	n = value	e raised to the power n
LN (n)	n > 0	natural logarithm of n
LOG (n2, n1)	base n2 any positive value other than 0 or 1, n1 any positive value	logarithm of n1, base n2
CEIL (n)	n = value	smallest integer greater than or equal to n
FLOOR (n)	n = value	greatest integer smaller than or equal to n
SIGN (n)	n = value	-1 if n < 0, 0 if n = 0, and 1 if n > 0

Numeric functions are used to perform operations on numbers and return numbers. Following are some of the Numeric functions

1. **ABS ():** It returns the absolute value of a number.

   EX: **select ABS (-243.5) from dual;** OUTPUT:

   243.5

2. **ACOS ():** It returns the cosine of a number.
   EX: **select ACOS (0.25) from dual;**

   OUTPUT: 1.318116071652818

3. **ASIN ():** It returns the arc sine of a number.
   EX: **select ASIN (0.25) from dual**;

   OUTPUT: 0.253680255142

4. **CEIL ():** It returns the smallest integer value that is a greater than or equal to a number. EX:
   select CEIL (25.77) from dual;

   OUTPUT: 26

5. **FLOOR ():** It returns the largest integer value that is a less than or equal to a number. EX:
   select FLOOR (25.75) from dual;

   OUTPUT: 25

6. **TRUNCATE ()**: This does not work for SQL server. It returns the truncated to 2 places right of the decimal point.
   EX: **select TRUNCATE (7.53635, 2) from dual;**

   OUTPUT: 7.53

7. **MOD ():** It returns the remainder when two numbers are divided. EX:
   select MOD (55,2) from dual;

   OUTPUT: 1.

8. **ROUND ():** This function rounds the given value to given number of digits of precision.
   EX: **select ROUND (14.5262,2) from dual;**

   OUTPUT: 14.53.

9. **POWER ():** This function gives the value of m raised to the power of n.
   EX: **select POWER (4,9) from dual;**

   OUTPUT: 262144.

10. **SQRT ():** This function gives the square root of the given value n.
    EX: **Select SQRT (576) from dual;**

    OUTPUT: 24.

11. **LEAST ():** This function returns least integer from given set of integers. EX:

    select LEAST (1,9,2,4,6,8,22) from dual;

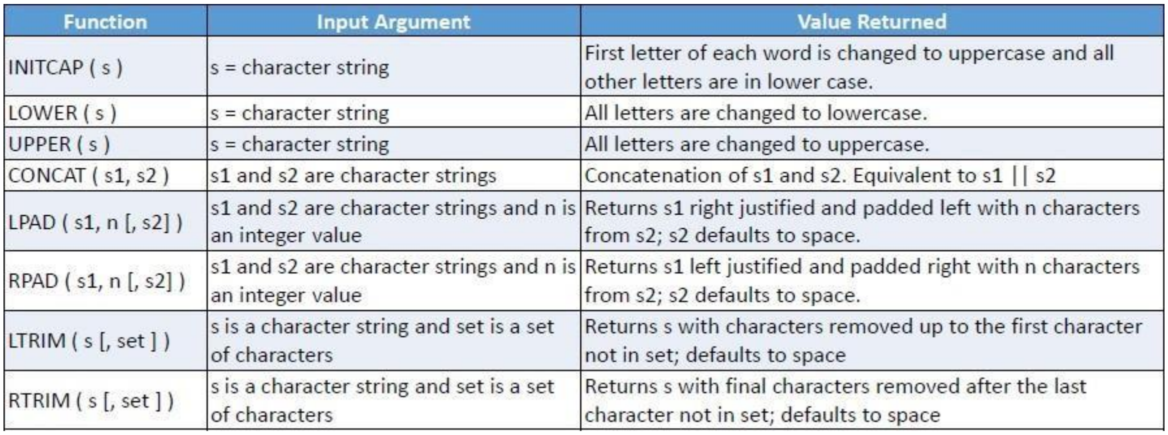
    OUTPUT: 1.

12. **GREATEST ():** This function returns greatest integer from given set of integers. EX:

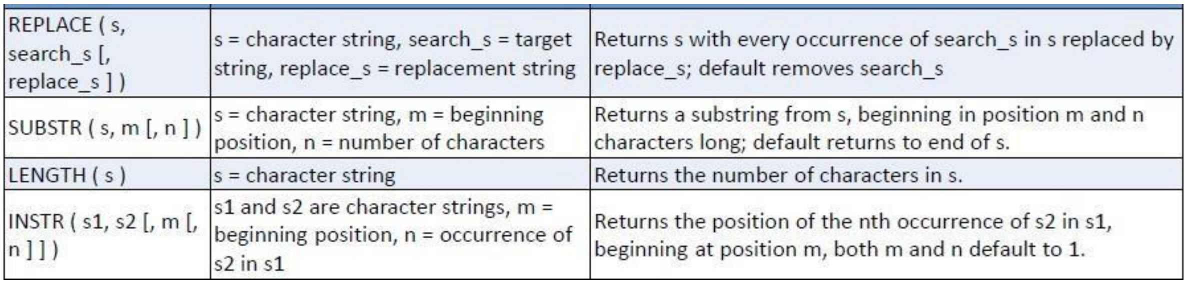
    select GREATEST (1,9,2,4,6,8,22) from dual;

    OUTPUT: 22

2.25 STRING CONVERSION FUNCTIONS OF SQL:



Function	Input Argument	Value Returned
INITCAP (s)	s = character string	First letter of each word is changed to uppercase and all other letters are in lower case.
LOWER (s)	s = character string	All letters are changed to lowercase.
UPPER (s)	s = character string	All letters are changed to uppercase.
CONCAT (s1, s2)	s1 and s2 are character strings	Concatenation of s1 and s2. Equivalent to s1 || s2
LPAD (s1, n [, s2])	s1 and s2 are character strings and n is an integer value	Returns s1 right justified and padded left with n characters from s2; s2 defaults to space.
RPAD (s1, n [, s2])	s1 and s2 are character strings and n is an integer value	Returns s1 left justified and padded right with n characters from s2; s2 defaults to space.
LTRIM (s [, set])	s is a character string and set is a set of characters	Returns s with characters removed up to the first character not in set; defaults to space
RTRIM (s [, set])	s is a character string and set is a set of characters	Returns s with final characters removed after the last character not in set; defaults to space



REPLACE (s, search_s [, replace_s])	s = character string, search_s = target string, replace_s = replacement string	Returns s with every occurrence of search_s in s replaced by replace_s; default removes search_s
SUBSTR (s, m [, n])	s = character string, m = beginning position, n = number of characters	Returns a substring from s, beginning in position m and n characters long; default returns to end of s.
LENGTH (s)	s = character string	Returns the number of characters in s.
INSTR (s1, s2 [, m [, n]])	s1 and s2 are character strings, m = beginning position, n = occurrence of s2 in s1	Returns the position of the nth occurrence of s2 in s1, beginning at position m, both m and n default to 1.

String Functions are used to perform an operation on input string and return the output string. Following are the string functions

1. **CONCAT ():** This function is used to add two words (or) strings.

EX: **select „database‟ ||‟ „|| „management system‟ From dual;**

OUTPUT: „database management system‟

2. **INSTR ():** This function is used to find the occurrence of an alphabet.

EX: **instr („database system‟,‟ a‟) from dual;**

OUTPUT: 2 (the first occurrence of „a‟)

3. **LOWER ():** This function is used to convert the given string into lowercase.

EX: **select lower („DATABASE‟) from dual;**

OUTPUT: database

4. **UPPER ():** This function is used to convert the lowercase string into uppercase.

EX: **select upper („database „) from dual;**

OUTPUT: DATABASE

5. **LPAD ():** This function is used to make the given string of the given size by adding the given symbol.

EX: > **lpad („system „, 8, „0‟) from dual;**

OUTPUT: 00system

6. **RPAD ():** This function is used to make the given string as long as the given size by adding the given symbol on the right.

EX: **rpad („system „,8, „0„) from dual;**

OUTPUT: system00

7. **LTRIM ():** This function is used to cut the given substring from the original string.

EX: **ltrim („database „, „data „) from dual;**

OUTPUT: base

8. **RTRIM ():** This function is used to cut the given substring from the original string.

EX: **rtrim („database „, „base „) from dual;**

OUTPUT: data.

9. **INITCAP ():** This function returns the string with first letter of each word starts with uppercase.

EX: **Select INITCAP („data base management system‟) from dual;**

OUTPUT: Data Base Management System.

10. **LENGTH ():** Tis function returns the length of the given string.

EX: **select LENGTH („SQ LANGUAGE‟) from dual;**

OUTPUT: 11.

11. **SUBSTR ():** This function returns a portion of a string beginning at the character position.

EX: **select SUBSTR („MY WORLD IS AMAZING‟,12,3) from dual;**

OUTPUT: AM.

12. **TRANSLATE ():** This function returns a string after replacing some set of characters into another set.

EX: **select TRANSLATE („Delhi is the capital of India‟,‟i‟,‟a‟) from dual;**

OUTPUT: Delha as the capatal andaa.

Predicates:

A predicate is an expression that evaluates to a Boolean value (TRUE, FALSE, o r UNKNOWN). Predicates are used to filter rows based on specific conditions.

They are commonly found in:

- **WHERE clauses:** Used to filter rows from a single table or the result of a join before they are returned.

SELECT * FROM Employees WHERE Salary > 50000;

- **HAVING clauses:** Used to filter groups of rows after aggregation.

SELECT Department, COUNT(*) FROM Employees GROUP BY Department HAVING COUNT(*) > 10;

- **ON clauses (within joins):** Used to specify the conditions for joining tables.

SELECT * FROM Orders o JOIN Customers c ON o.CustomerID = c.CustomerID;

Joins:

SQL JOINs are fundamental operations in relational databases used to combine rows from two or more tables based on a related column between them. They allow for the retrieval of data that is distributed across various tables, creating a unified result set. A join is an SQL operation that combines rows from two or more tables based on related columns between them.

Different types of joins exist:

- **INNER JOIN:** Returns only the rows where there is a match in both tables based on the join condition.

Code

```
SELECT e.Name, d.DepartmentName
FROM Employees e INNER JOIN Departments d ON e.DepartmentID = d.DepartmentID;
```

- **LEFT (OUTER) JOIN:** Returns all rows from the left table and matching rows from the right table. If no match is found in the right table, NULL values are returned for the right table's columns.

Code

```
SELECT c.CustomerName, o.OrderDate
FROM Customers c LEFT JOIN Orders o ON c.CustomerID = o.CustomerID;
```

- **RIGHT (OUTER) JOIN:** Returns all rows from the right table and matching rows from the left table. If no match is found in the left table, NULL values are returned for the left table's columns.

Code

```
SELECT p.ProductName, s.SaleAmount
FROM Products p RIGHT JOIN Sales s ON p.ProductID = s.ProductID;
```

- **FULL (OUTER) JOIN:** Returns all rows when there is a match in either the left or the right table. If no match is found in one table, NULL values are returned for that table's columns.

Code

```
SELECT e.Name, d.DepartmentName
FROM Employees e FULL JOIN Departments d ON e.DepartmentID = d.DepartmentID;
```

- **CROSS JOIN:** Returns the Cartesian product of the two tables, meaning every row from the first table is combined with every row from the second table.

Code

```
SELECT * FROM TableA CROSS JOIN TableB;
```

SET MEMBERSHIP

In a Database Management System (DBMS), "set membership" refers to the concept of determining whether a specific data value or a set of data values is present within another set of values. This concept is fundamental to querying and manipulating data, particularly within the context of relational databases and SQL.

The primary mechanism for checking set membership in SQL is the IN operator.

Using the IN operator:

The IN operator allows for checking if a value exists within a specified list of values or within the result set of a subquery. Checking against a list of values.

Code

```
SELECT *
FROM employees
WHERE department_id IN (101, 103, 105);
```

This query retrieves all employees whose department_id is either 101, 103, or 105.

- **Checking against the result of a subquery:**

Code

```
SELECT employee_name
FROM employees
WHERE employee_id IN (SELECT manager_id FROM departments WHERE location = 'New York');
```

This query retrieves the names of employees who are also managers in departments located in 'New York'. The inner subquery generates a set of manager_id values, and the outer query checks if an employee_id is a member of that set.

The NOT IN operator:

Conversely, the NOT IN operator is used to check if a value is not present within a specified set.

Code

```
SELECT product_name
FROM products
WHERE category_id NOT IN (SELECT category_id FROM categories WHERE category_name = 'Electronics');
```

This query retrieves product names that do not belong to the 'Electronics' category.

Other Set Operators:

While IN and NOT IN directly address set membership, other set operators like UNION, INTERSECT, and EXCEPT (or MINUS in some systems) also leverage set theory principles to combine or compare result sets, effectively performing operations on collections of data. For example, INTERSECT finds common members between two sets, which is a form of set membership applied across two query results.

COMPARISON OPERATOR

- SOME (or ANY) Operator: Used with comparison operators (=, !=, >, <, >=, <=) to compare a value against at least one value in a subquery's result set.

Code

```
SELECT column_name
FROM table_name
WHERE column_name > SOME (SELECT another_column FROM another_table WHERE condition);
```

Ordering of Tuples (Rows) in SQL

The ordering of tuples, or rows, in SQL is achieved using the ORDER BY clause. This clause sorts the result set of a query based on one or more specified columns, either in ascending (ASC) or descending (DESC) order. Ascending order is the default if no order is specified.

Example:

Code

```
SELECT column1, column2
FROM table_name
ORDER BY column1 ASC, column2 DESC;
```

Aggregate Functions in SQL

Aggregate functions perform calculations on a set of rows and return a single summary value. They are often used in conjunction with the GROUP BY clause to perform calculations on grouped data.

Common Aggregate Functions:

- **COUNT()**: Returns the number of rows in a set. COUNT(*) counts all rows, while COUNT(column_name) counts non-NULL values in a column. COUNT(DISTINCT column_name) counts unique non-NULL values.
- **SUM()**: Calculates the sum of values in a numeric column.
- **AVG()**: Calculates the average of values in a numeric column.
- **MIN()**: Returns the minimum value in a column.
- **MAX()**: Returns the maximum value in a column.
-

Using Aggregate Functions with GROUP BY and ORDER BY:

When using aggregate functions to summarize data for different groups, the GROUP BY clause is used to define these groups. The ORDER BY clause can then be used to sort the results based on the aggregated values or other columns.

Example:

Code

```
SELECT department_id, AVG(salary) AS avg_salary
FROM employees
GROUP BY department_id
ORDER BY avg_salary DESC;
```

This query calculates the average salary for each department and then orders the results by the average salary in descending order.

nested queries in dbms

A nested query, also known as a subquery or inner query, is an SQL query embedded within another SQL query. The inner query executes first, and its result is then used by the outer query to perform its operations. This structure allows for more complex data retrieval and manipulation.

Key characteristics of nested queries:

- **Structure:**

A subquery is enclosed in parentheses () and typically appears within the WHERE, FROM, or HAVING clauses of the outer query. It can also be used with SELECT, INSERT, UPDATE, and DELETE statements.

- **Execution Order:**

The inner query executes completely before the outer query begins processing, and its results are passed to the outer query as input.

- **Purpose:**

Nested queries are used to break down complex problems into smaller, more manageable parts, enabling the retrieval of specific data based on conditions derived from other data.

- **Operators:**

They often use operators like =, <, >, IN, NOT IN, EXISTS, NOT EXISTS, ANY, and ALL to compare values from the outer query with the results of the subquery.

Example:

To find the names of employees who work in departments located in 'New York', a nested query could be used:

Code

```
SELECT EmployeeName
FROM Employees
WHERE DepartmentID IN (SELECT DepartmentID FROM Departments WHERE Location = 'New York');
```

In this example, the inner query (SELECT DepartmentID FROM Departments WHERE Location = 'New York') first retrieves the DepartmentID values for departments in 'New York'. The outer query then uses these DepartmentID values to select the EmployeeName from the Employees table.

PL- SQL

1. PL/SQL, which stands for Procedural Language/Structured Query Language, is Oracle Corporation's procedural extension to SQL.
2. It integrates the data manipulation capabilities of SQL with the procedural programming constructs found in other languages, such as loops, conditional statements (IF-THEN-ELSE), variables, and exception handling. This allows for the creation of more complex and robust database applications.

Stored Procedures and Functions are named PL/SQL blocks that are stored and compiled within the database, offering several benefits:

- **Stored Procedures:**
  - A named PL/SQL block designed to perform a specific set of actions or tasks.
  - They may or may not return a value. If a value needs to be returned, it is typically done through OUT or INOUT parameters.
  - Often used for data manipulation (INSERT, UPDATE, DELETE), implementing business logic, or automating routine database operations.

- **Stored Functions:**
  - A named PL/SQL block designed to compute a value and return a single result.
  - They must return a single value using a RETURN statement, and the return data type is specified during function creation.
  - Can be called within SQL statements (e.g., in a SELECT clause or WHERE clause) or within other PL/SQL blocks.
  - Primarily used for calculations or retrieving a specific piece of information.

Key Differences Summarized:

Feature	**Stored Procedure**	**Stored Function**
Return Value	May or may not return a value (uses OUT/INOUT parameters)	Must return a single value (uses RETURN keyword)
Usage in SQL	Cannot be directly called within SQL statements	Can be called within SQL statements (e.g., SELECT clause)
Primary Purpose	Performing actions, data manipulation, business logic	Calculating and returning a single value

In Database Management Systems (DBMS), a cursor is a control structure that enables traversal over the rows of a result set returned by a query. It acts as a pointer to the current row within this result set, allowing you to process data one row at a time.

Here are the key aspects of cursors in DBMS:

- **Purpose:**

Cursors are used to retrieve and process individual rows from a result set, especially when a query returns multiple rows. This allows for row-by-row manipulation, such as updating or deleting specific records based on certain conditions.

- **Types:**
  - **Implicit Cursors:** These are automatically created and managed by the DBMS for DML (Data Manipulation Language) statements like INSERT, UPDATE, and DELETE, and also for SELECT statements that return a single row.
  - **Explicit Cursors:** These are explicitly declared and controlled by the programmer for SELECT statements that are expected to return multiple rows. They offer more fine-grained control over data processing.

- **Explicit Cursor Lifecycle (in PL/SQL, for example):**
  - **Declaration:** Defining the cursor with a SELECT statement.
  - **Opening:** Executing the SELECT statement and populating the cursor's result set (active set).
  - **Fetching:** Retrieving individual rows from the active set into program variables.
  - **Closing:** Releasing the resources associated with the cursor.

TRIGGERS

SQL triggers are special types of stored procedures that automatically execute (or "fire") in response to specific events occurring in a database. They are primarily used to enforce data integrity, automate tasks, and maintain audit trails.

Key characteristics of SQL triggers:

- **Event-driven:**

Triggers are activated by specific database events, most commonly Data Manipulation Language (DML) events like INSERT, UPDATE, or DELETE operations on a table. Some database systems also support triggers on Data Definition Language (DDL) events (e.g., CREATE, ALTER, DROP) or logon events.

- **Automatic execution:**

Once defined, triggers execute automatically when their associated event occurs, without requiring explicit calls from applications or users.

- **Associated with tables:**

Triggers are typically tied to a specific table or view, and they are automatically dropped if the associated table or view is deleted.

- **Before or After:**

Triggers can be set to fire before the triggering event (e.g., BEFORE INSERT) or after the event (e.g., AFTER UPDATE). This allows for different types of actions, such as validating data before it's stored or recording changes after they occur.

Temporary Tables (Inserted and Deleted):

During INSERT, UPDATE, and DELETE operations, SQL Server (and similar systems) create special temporary tables in memory called Inserted and Deleted. Triggers can access these tables to inspect the data being inserted, updated, or deleted, enabling complex logic.

Common uses of SQL triggers:

- **Maintaining data integrity:**

Enforcing complex business rules that cannot be handled by simple constraints (e.g., ensuring a total quantity across multiple tables).

- **Auditing and logging:**

Recording changes to data, such as who made the change, when, and what the old and new values were.

- **Automating tasks:**

Automatically updating related tables or performing other actions in response to data modifications (e.g., updating a total_orders column when a new order is placed).

- **Replicating data:**

Synchronizing data between different tables or even different databases.

ASSERTION

In SQL, an assertion is a database object used to enforce a specific condition or constraint across multiple tables or the entire database. It ensures that a given condition is always true within the database system. If the condition specified in the assertion is violated by any data modification operation (e.g., INSERT, UPDATE, DELETE), the operation is rejected, and an error is typically raised.

Assertions are defined using the CREATE ASSERTION statement, followed by a name for the assertion and a CHECK clause containing the condition to be enforced.

Key characteristics of SQL assertions:

- **Global Scope:**

Unlike CHECK constraints, which are tied to a single table or column, assertions can enforce conditions that span multiple tables or involve aggregate functions across the database.

- **Integrity Enforcement:**

They are a mechanism to maintain data integrity and consistency by ensuring that complex business rules are always upheld.

- **Automatic Checking:**

The database system automatically checks assertions whenever data modifications occur that could potentially violate the specified condition.

- **Standard Feature (Limited Support):**

While CREATE ASSERTION is part of the SQL standard, its implementation and support vary significantly across different database management systems (DBMS). Many modern DBMS prefer the use of triggers or more advanced constraint mechanisms to achieve similar functionality.

Example of an assertion (conceptual, as support varies):

Code

```
CREATE ASSERTION NoEmployeeSalaryGreaterThanManager
CHECK (NOT EXISTS (
   SELECT E1.salary, E2.salary
   FROM Employees E1 JOIN Employees E2 ON E1.manager_id = E2.employee_id
   WHERE E1.salary > E2.salary
));
```

This assertion would conceptually ensure that no employee's salary is greater than their manager's salary. If an attempt to insert or update an employee record would violate this condition, the operation would be prevented.

Difference Between Assertions and Triggers

Assertions	**Triggers**
We can use Assertions when we know that the given particular condition is always true.	We can use Triggers even particular condition may or may not be true.
When the SQL condition is not met then there are chances to an entire table or even Database to get locked up.	Triggers can catch errors if the condition of the query is not true.
Assertions are not linked to specific table or event. It performs task specified or defined by the user.	It helps in maintaining the integrity constraints in the database tables, especially when the primary key and foreign key constraint are not defined.
Assertions do not maintain any track of changes made in table.	Triggers maintain track of all changes occurred in table.
Assertions have small syntax compared to Triggers.	They have large Syntax to indicate each and every specific of the created trigger.
Modern databases do not use Assertions.	Triggers are very well used in modern databases.
Purpose of assertions is to Enforces business rules and constraints.	Purpose of triggers is to Executes actions in response to data changes.
Activation is checked after a transaction completes	Activation is activated by data changes during a transaction
Granularity applies to the entire database	Granularity applies to a specific table or view
Syntax Uses SQL statements	Syntax Uses procedural code (e.g. PL/SQL, T-SQL)
Error handling Causes transaction to be rolled back.	Error handling can ignore errors or handle them explicitly

Assertions	Triggers
Assertions may slow down performance of queries.	Triggers Can impact performance of data changes.
Assertions are Easy to debug with SQL statements.	Triggers are more difficult to debug procedural code
Examples- CHECK constraints, FOREIGN KEY constraints	Examples - AFTER INSERT triggers, INSTEAD OF triggers

Roles and Privileges.

In a Database Management System (DBMS), roles and privileges are fundamental concepts for managing database security and access control.

Privileges

Privileges represent the specific permissions granted to a user or a role, allowing them to perform certain actions within the database. These actions can be broadly categorized into:

- **System Privileges:**

These relate to operations on the database system itself, such as creating users, creating tablespaces, or performing database backups. Examples include CREATE TABLE, CREATE USER, DROP ANY TABLE.

- **Object Privileges:**

These relate to operations on specific database objects like tables, views, sequences, or stored procedures. Examples include SELECT on a table, INSERT into a table, EXECUTE on a stored procedure, or UPDATE on a specific column.

Roles

Roles are named collections of related privileges that can be granted to users or other roles. They are designed to simplify privilege management, especially in databases with many users and complex access requirements. Instead of granting individual privileges to each user, administrators can:

- **Define a role**: with a set of necessary privileges.
- **Grant that role**: to multiple users.

Review Questions:

1. Discuss GROUPBY and HAVING clauses with an example. And also give the constraints related to their usage.

2. Consider the SAILOR DATABASE

Sailors (Sid: string, sname: string, rating: integer, age: real)

Boats (bid: integer, bname: string, colour: string)

Reserves (Sid: integer, bid: integer, day: date)

Based on the above schema, write the corresponding SQL queries for the following? Find the colours of boats reserved by Lubber.

Find the names of sailors who have reserved at least one boat. Find the names of sailors who have reserved a red or green boat. Find the names of the sailors who have reserved both a red boat and a green boa Find names of sailors who have reserved all boats.

3. Write about the usability of „group by‟ and „having‟ clauses In SQL.

4. How would you use the operators IN, EXISTS, UNIQUE, ANY and ALL in writing nested queries? Why are they useful? Explain with an example.

5. Explain commands with respect to SQL: (i) Rename (ii) Alter (iii) View

6. Explain two aggregate functions of SQL.

7. Explain the following SQL constructs with examples:
(1) order by (2) group by and having (3) as select (4) schema

8. Write SQL Queries for following set of tables:

EMPLOYEE (EmpNo, Name, DoB, Address, Gender, Salary, DNumber)

DEPARTMENT (DNumber, Dname, ManagerEmpNo, ManagerStartDate).

Display the Age of „male‟ employees.

Display all employees in Department named „Marketing‟. Display the name of highest salary paid „female‟ employee. Which employee is oldest manger in company?

Display the name of department of the employee „SMITH.

9. Describe creating and modifying relations using SQL. Give examples for each.

10. Explain about integrity constraints over relations.

11. Consider the following insurance database, where the primary keys are underlined.

Construct the following SQL queries for this relational database. person (driver-id#, name, address)

car (license, model, year)

accident (report-number, date, location) owns

(driver-id#, license)

participated (driver-id, car, report-number, damage-amount)

a. Find the total number of people who owned cars that were involved in accidents in 1989.

b. Find the number of accidents in which the cars belonging to “John Smith” were involved.

c. Add a new accident to the database; assume any values for required attributes.

d. Delete the Mazda belonging to "John Smith".

e. Update the damage amount for the car with license number "AABB2000" in the accident with report number "AR2197" to $3000.

12. List the SQL functions for string conversions.

13. Write SQL statements for following:

Student (Enrno, name, courseId, emailId, cellno)

Course (courseId, course_nm, duration)

i. Add a column city in student table.

ii. Find out list of students who have enrolled in "computer" course.

iii. List name of all courses with their duration.

iv. List name of all students start with „a".

v. List email Id and cell no of all mechanical engineering students.

14. How to define a domain constraint? Give an example.

15. What is an integrity constraint? Explain its enforcement by DBMS with illustrative

16. List the data types supported by SQL.

17. Demonstrate the use of DISTINCT keyword in SQL select statement.

18. Consider the following database schema to write nested queries in SQL

Supplier (id, name, city)

Parts (pno, pname, pdescription)

Supply (id, pno, cost)

i. Find the names of the parts supplied by "RamRaj"

ii. Find the names of the suppliers who supply "Nuts"

iii. Find the cost of bolts being supplied by Nagpur suppliers.

References:

- Raghurama Krishnan, Johannes Gehrke, *Database Management Systems*, 3rd Edition, Tata McGraw Hill.
- C.J. Date, *Introduction to Database Systems*, Pearson Education.
- Elmasri Navrate, *Fundamentals of Database Systems*, Pearson Education.