SQL

**Databases**

The place we can store the related data and later retrieve the data is known as Database

Storing the related Data like Employee ID, Employee Name, Employee Salary, Department etc. can be stored

Retrieving the data based on some conditions

**Database Management Systems**

Shortly called as DBMS

Software that stores data in databases in an organized way to make it easier to create, retrieve, update etc.

**Data Models**

data is connected to each other and stored the data inside the database.

Types of Data Models:

Hierarchical Model

Network Model

Entity-Relationship Model

Relational Model

Data is stored in the form of tables

Tables organize the data in the form of rows and columns

It is a popular and widely used by most of the DBMS Software

**RDBMS**

DBMS using Relational Data Models are known as RDBMS

In RDBMS the data is stored in the form of database objects called tables.

.Tables organize the data in the form of rows and columns

**SQL**

Stands for Structured Query Language

In SQL we can access and manipulate databases

Things can be performed on Database using SQL:

Inserting Data

Retrieving Data

Updating Data

Deleting Data

And many more

SQL can set permissions on tables, procedures, and views

**SELECT**

1. is to retrieve the data from the tables
2. Retrieving all the data from Customers table

Syntax: Select \* from TableName;

Select \* from Customers;

**distinct keyword**

To retrieved the unique values, we have to use distinct keyword with Select command

1. Select distinct Country from Customers;

**Where Clause**

1. Purpose is to filter records based on some condition

Select \* from Customers where City='London';

**Using Logical Operators (AND, OR, NOT)**

Purpose: To filter the records based on Multiple Conditions

AND

OR

NOT

AND

1. Select \* from Customers where Country='Mexico' AND CustomerID>3;

OR

1. Select \* from Customers where Country='Germany' OR City='London';

NOT

* + 1. Select \* from Customers where NOT Country='Germany' AND City='London';

**Between Operator**

Purpose is to filter records based on some Range

Select \* from Products where Price Between 10 And 20;

**Order By Clause**

Purpose is to order the retrieved records in ascending or descending order

Practical Demonstration:

S0ELECT \* FROM Customers Order By Country ASC;

SELECT \* FROM Customers Order By Country DESC;

**In Operator**

Simplifies providing multiple values in Where Clause, when all the values are from the same column

Select \* from Products where Price=18 OR Price=30 OR Price=10;

Select \* from Products where Price in (18,30,10);

Select \* from Customers where Country='USA' OR Country='Canada' OR Country='UK';

Select \* from Customers where Country in ('USA','Canada','UK');

**Like Operator and Wildcard Characters**

We can use Like Operator and Wildcard Characters for Pattern matching needs

Select \* from Customers Where Country Like '%a';

Select \* from Customers Where Country Like 'F%';

**Aliases for Table Column Names (As Keyword)**

While the records are retrieved these alias names provided for Column names will be displayed temporary in place of Original Column names

Select CategoryID as ID,CategoryName as Name from Categories;

**Limit Keyword**

1. When a Table has 1000's of records, the Application will slow down when trying to display them.
2. Using Limit keyword we can decide how many records needs to be displayed
3. Select \* from Customers where Country='USA' Limit 5;
4. Select \* from Customers Limit 2,6;

**upper() MySQL String Function**

Select upper(Country) AS Country from Customers;

**length() MySQL String Function**

1. Select length(Country) AS SIZE from Customers;

**instr() MySQL String Function**

Finds the position of the given text in the data of the specified Column

Select instr('Arun Motoori','n');

Select instr('Arun Motoori','n') AS Position;

**substr() MySQL String Function**

1. Retrieves a portion of text from the data of the specified column
   1. Select substr('Arun',2,3);
   2. Select substr('Arun',2,3) AS Portion;
   3. Select substr('Arun',-3,2);

**concat() MySQL String Function**

Adds Two or More Table Column data together

Select concat('Arun',' ','Motoori') AS FullName;

Select Concat(FirstName,' ',LastName) from Employees;

**trim() MySQL String Function**

Removes the leading and trailing spaces of the Column data

Select trim(ColumnName) from TableName;

**truncate() MySQL Numeric Function**

truncate() - Returns the numerical values with the allowed number of digits after decimal point

Select truncate(123.4567,2);

123.45

**Current\_date(), curdate(), current\_time(), curtime(), now() and sysdate() MySQL Date Time Functions**

* 1. Select current\_date();
  2. Select curdate();
  3. Select current\_time();
  4. Select curtime();
  5. select now();
  6. select sysdate()

**year(),month(),day(),monthname(),dayname() MySQL Date Time Functions**

Select year('2020-04-25');

Select monthname('2020-04-25');

Select dayname('2020-04-25');

SELECT \* FROM Orders where month(OrderDate)=4;

SELECT \* FROM Orders where monthname(OrderDate)='May';

**Delete Statement (For Deleting the Records from Table)**

* 1. delete from employees where id=5;
  2. delete from employees where name='Tharun';
  3. delete from employees;

**Update Statement and Set Keyword (For Updating the Table Records)**

* 1. update employees set name='Kiran' where id=3;
  2. update employees set name='Tharun',id=4 where experience=5;

**Rename Statement and To keyword (For Renaming Table Name)**

* 1. show tables;
  2. rename table employees to emp;
  3. rename table emp to employees;

**Alter Statement, Add, Modify Column, Rename Column and Drop Column Keywords**

* 1. show tables;
  2. describe employees;
  3. alter table employees add location varchar(15);
  4. alter table employees modify column location varchar(20);
  5. alter table employees rename column location to loc;
  6. alter table employees drop column loc;

**Set Autocommit**

1. set autocommit=0;
2. insert into employees values(11,'Isha',6);
3. Select \* from employees;
4. Restart the workbench client and observe that the inserted data is not stored
5. set autocommit=1;
6. insert into employees values(11,'Isha',6);
7. Select \* from employees;
8. Restart the workbench client and observe that the inserted data is stored

**Commit Statement**

* 1. set autocommit=0;
  2. Select \* from employees;
  3. insert into employees values(9,'Dinesh',3);
  4. Restart the workbench client and observe that the inserted data is not permanently stored
     1. Select \* from employees;
  5. insert into employees values(9,'Dinesh',3);
  6. commit;
  7. Select \* from employees;
  8. Restart the workbench client and observe that the inserted data is stored permanently
     1. use qafox;
     2. Select \* from employees;
  9. set autocommit=1;

**Unique (Integrity Constraint)**

Unique is an Integrity Constraint

Unique when specified to a column will not allow the column to allow duplicate values

create table emp(id int unique,name varchar(15),experience int);

Unique values can be used at Table and Column levels

**Primary Key (Integrity Constraint)**

Primary Key is an Integrity Constraint

Primary Key = Not Null + Unique;

create table emp(id int not null unique,name varchar(15),experience int);

Primary Key at Column Level and Table Level

**Foreign Key (Integrity Constraint)**

1. Foreign Key is one of the Integrity Constraints
2. We need to create two tables having a common column
   1. Parent Table (Reference Table)
      1. Should have a Primary Key (Candidate Key) specified for Common Column
   2. Child Table
      1. Should have the Foreign Key Specified for Common Column
3. Child Table Depends on the Parent Table
   1. Inserting the records in Child table should match with the Parent Table common column
   2. Deleting the records in Parent Table is not possible if there are dependent records in the Child Table
4. On Delete Cascade
5. Practical Demonstration
   1. create table employee(id int primary key,name varchar(15),experience int);
   2. create table salary(id int,sal int,foreign key(id) references employee(id));

**Views**

1. The main purpose of views is to create different varieties of the same table or tables (Virtual Tables)
2. We can create Views by customizing the same table columns or by selecting the columns from multiple tables
3. Hide the Database implementation of the actual tables and show the desired virtual views to the Users
4. Practical Demonstration
   1. create table empone(id int,name varchar(15),country varchar(15));
   2. insert into empone values (1,'Arun','India');
   3. insert into empone values (2,'Varun','UK');
   4. insert into empone values (3,'Tharun','Spain');
   5. insert into empone values (4,'Dinesh','New Zealand);
   6. select \* from empone;
   7. create table emptwo(id int,experience int);
   8. insert into emptwo values(1,12);
   9. insert into emptwo values(2,7);
   10. select \* from emptwo;
   11. create view emponeviewa as Select \* from empone;
   12. create view emponeviewb as Select id,name from empone;
   13. create view emponeviewc as Select id,country from empone;
   14. create view emponeviewd as Select country,name,id from empone;
   15. create view empviewb as Select empone.id,empone.name,emptwo.experience from empone,emptwo where empone.id=emptwo.id;
5. Performing operators on the Views will effect the original tables and vice versa

**Indexes**

1. when indexes are implemented for a table, will increase the performance of the Application by retrieving the records at faster speed
2. Indexes are like Table of Contents
3. it provides the faster access of the required topics

Insertions and Updates will slow down if you create indexes

* 1. create table empone(id int,name varchar(15),experience int,country varchar(15));
  2. insert into empone values(1,'Arun',12,'India');
  3. insert into empone values(2,'Varun',7,'USA');
  4. insert into empone values(3,'Tharun',9,'UK');
  5. Select \* from empone;
  6. Select \* from empone where country='USA';
  7. show indexes from empone;
  8. create index empcountry
  9. on empone(country);
  10. show indexes from empone;
  11. drop index empcountry on empone;
  12. create table emptwo(id int primary key,name varchar(15));
      1. primary index will be automatically created
      2. show indexes from emptwo;
  13. create table empthree(id int unique,name varchar(15));
      1. id index will be automatically created
      2. show indexes from empthree;

WHERE HAVING

Ø Where clause is used to Filter the records Ø Having clause is used to Filter the groups . Ø Where clause executes row By row . Ø Having clause executes Group by group

Ø In Where Clause we cannot Use MRF( ) Ø Can use MRF( ).

Ø Where clause executes before Group by clause . Ø Having clause executes After group by clause .

2

02+0+2