SQL

By

Praveen Oruganti

Blog: https://praveenoruganti.blogspot.com
Facebook Group: https://www.facebook.com/groups/2340886582907696/
Github repo: https://github.com/praveenoruganti

What are the different types of SQL commands?

SQL commands are segregated into the following types:

DDL – Data Definition Language
DML – Data Manipulation Language
DQL – Data Query Language
DCL – Data Control Language
TCL – Transaction Control Language

What are the different DDL commands in SQL?

DDL commands are used to define or alter the structure of the database.

CREATE: To create databases and database objects
ALTER: To alter existing database objects
DROP: To drop databases and databases objects
TRUNCATE: To remove all records from a table but not its database structure
RENAME: To rename database objects

What are the different DML commands in SQL?

DML commands are used for managing data present in the database.

SELECT: To select specific data from a database
INSERT: To insert new records into a table
UPDATE: To update existing records
DELETE: To delete existing records from a table

What are the different DCL commands in SQL?

DCL commands are used to create roles, grant permission and control access to the database objects.

GRANT: To provide user access
DENY: To deny permissions to users
REVOKE: To remove user access

What are the different TCL commands in SQL?

TCL commands are used to manage the changes made by DML statements.

COMMIT: To write and store the changes to the database
ROLLBACK: To restore the database since the last commit

What is the need of MERGE statement?

This statement allows conditional update or insertion of data into a table. It performs an UPDATE if a row exists, or an INSERT if the row does not exist.

Let's see some basic Queries

Creating a table and inserting data

SQL: CREATE table with multiple data types

```
CREATE TABLE friends (
   id INTEGER PRIMARY KEY,
   name TEXT,
   age INTEGER,
   weight REAL);
INSERT INTO friends VALUES (1, "Praveen", 34, 80);
```

SQL: CREATE table with a primary key

```
CREATE TABLE customers (
   id INTEGER PRIMARY KEY,
   phone TEXT);
INSERT INTO customers VALUES (1, "555-222-3333");
```

SQL: CREATE table with a primary key and autoincrement

```
CREATE TABLE customers_orders (
   id INTEGER PRIMARY KEY AUTOINCREMENT,
   customer_id INTEGER,
   item TEXT);
/* It will automatically pick an id that's different from other ones. */
INSERT INTO customers_orders (customer_id, item) VALUES (1, "Hot Air Balloon");
```

SQL: Specifying DEFAULT values in CREATE TABLE

```
CREATE TABLE courses
   (id INTEGER PRIMARY KEY,
   name TEXT,
   prereqs TEXT DEFAULT "None",
```

```
    num_hours INTEGER DEFAULT 15);
INSERT INTO courses (name, num_hours) VALUES
    ("Intro to JS", 25);
```

SQL: CREATE TABLE with foreign key reference

```
CREATE TABLE artists (
  id    INTEGER PRIMARY KEY,
  name  TEXT
);
CREATE TABLE tracks (
  traid     INTEGER,
  title   TEXT,
  artist INTEGER,
  FOREIGN KEY(artist) REFERENCES artists(id)
);
INSERT INTO artists VALUES (1, "Tom Chapin");
INSERT INTO tracks VALUES (1, "Great Big Words", 1);
```

SQL: INSERTing values in tables

```
CREATE TABLE us_states (
    id INTEGER PRIMARY KEY,
    name TEXT,
    flower TEXT);
INSERT INTO us_states VALUES (1, "California", "California Poppy");
/* Or you can only specify some columns */
INSERT INTO us_states (name, flower) VALUES ("Alaska", "Forget-me-not");
```

SQL: SELECTing with WHERE conditions

```
CREATE TABLE fruits (
    id INTEGER PRIMARY KEY,
```

```
    name TEXT,
    price INTEGER,
    season TEXT);
/* What are the cheap fruits? */
SELECT * FROM fruits WHERE price < 4;
/* What are the expensive fruits? */
SELECT * FROM fruits WHERE price > 5;
/* What are the cheap fruits in the summer? */
SELECT * FROM fruits WHERE price < 4 AND season = "summer";
/* What fruits can I get in fall or winter? */
SELECT * FROM fruits WHERE season = "summer" OR season = "winter";
```

SQL: SELECTing with WHERE conditions

```
CREATE TABLE fruits (
    id INTEGER PRIMARY KEY,
    name TEXT,
    price INTEGER,
    season TEXT);
/* What are the cheap fruits? */
SELECT * FROM fruits WHERE price < 4;
/* What are the expensive fruits? */
SELECT * FROM fruits WHERE price > 5;
/* What are the cheap fruits in the summer? */
SELECT * FROM fruits WHERE price < 4 AND season = "summer";
/* What fruits can I get in fall or winter? */
SELECT * FROM fruits WHERE season = "summer" OR season = "winter";
```

SQL: Filter with IN

```
CREATE TABLE shoes (
```

```
    id INTEGER PRIMARY KEY,
    name TEXT,
    price REAL,
    type TEXT);
/* Which shoes will make me taller? */
SELECT * FROM shoes WHERE type IN ("heel", "wedge");
/* This is equivalent to: */
SELECT * FROM shoes WHERE type = "heel" OR type = "wedge";
/* Which shoes *won't* make me taller? */
SELECT * FROM shoes WHERE type NOT IN ("heel", "wedge");
```

SQL: SELECTing rows

```
CREATE TABLE veggies (
    id INTEGER PRIMARY KEY,
    name TEXT,
    season TEXT);
/* Which of your vegetables should you eat? */
SELECT * FROM veggies;
/* Just show the names and seasons, we don't need to see ids */
SELECT name, season FROM veggies;
```

SQL: SELECT with ORDER BY

```
CREATE TABLE skyscrapers (id INTEGER PRIMARY KEY,
    name TEXT,
    country TEXT,
    height_meters INTEGER,
    floors INTEGER
);
/* Sort by their height in meters */
```

```
SELECT * FROM skyscrapers ORDER BY height_meters;
/* Reverse that sort (highest to lowest) */
SELECT * FROM skyscrapers ORDER BY height_meters DESC;
/* Sort by their floors, lowest to highest */
SELECT * FROM skyscrapers ORDER BY floors ASC;
```

SQL: Transform SELECT results with CASE

```
CREATE table species (
   id INTEGER PRIMARY KEY,
   name TEXT,
   appeared INTEGER /* mya, millions of years ago */,
   phylum TEXT
   );
SELECT name, CASE
   WHEN appeared < 540 AND appeared >= 245 THEN "Paleozoic"
   WHEN appeared < 245 AND appeared >= 65 THEN "Mesozoic"
   WHEN appeared < 65 THEN "Cenozoic"
   END "era"
   FROM species
   ORDER BY era;
```

SQL: Filtering with LIKE

```
CREATE TABLE dresses (
   id INTEGER PRIMARY KEY,
   name TEXT,
   description TEXT,
   price REAL);
/* Which ones have floral patterns? */
SELECT  * FROM dresses WHERE description LIKE "%floral%";
```

```
/* Which ones are blue? */
SELECT  * FROM dresses WHERE description LIKE "%blue%";
```

SQL: Using SELECT with LIMIT

```
CREATE TABLE top_programs(id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT,
    creator TEXT,
    votes INTEGER);
/* But let's just show the top 3! */
SELECT * FROM top_programs ORDER BY votes DESC LIMIT 3;
```

SQL: ROUND and other core functions

```
CREATE TABLE organs (id INTEGER PRIMARY KEY,
    name TEXT,
    weight INTEGER,
    important_functions TEXT);
SELECT LOWER(name) AS name,
    ROUND(weight) AS rounded_weight,
    LOWER(TRIM(important_functions, ".")) AS desc
    FROM organs WHERE weight > 1;
```

SQL: SELECT with aggregate functions

```
CREATE TABLE paintings
    (id INTEGER PRIMARY KEY,
    name TEXT,
    artist TEXT,
    year INTEGER,
    price REAL);
/* What's the most expensive price for a painting? */
SELECT MAX(price) as most_dollars_paid
```

```
    FROM paintings;
/* What's the average year these were painted? */
SELECT AVG(year)
    FROM paintings;
/* How much money was paid for these paintings total? */
SELECT SUM(price) AS total_dollars_paid
    FROM paintings;
/* How many cost more than 2 million? */
SELECT COUNT(*) AS greater_than_2mil
    FROM paintings WHERE price > 200000000;
/* How many unique years are there? */
SELECT COUNT(DISTINCT year) AS years_represented
    FROM paintings;
```

SQL: Grouping SELECT results with GROUP BY

```
CREATE TABLE bridges (
    id INTEGER PRIMARY KEY,
    name TEXT,
    length INTEGER, /* meters */
    country TEXT);
SELECT country, COUNT(*)
    FROM bridges GROUP BY country;
SELECT country, SUM(length)
    FROM bridges GROUP BY country;
```

SQL: Using HAVING with GROUPED BY

```
CREATE TABLE nobel_prizes (
    id INTEGER PRIMARY KEY,
    name TEXT,
```

```
    field TEXT,
    country TEXT,
    year INTEGER
    );
/* Which countries had more than 1 person win? */
SELECT country, COUNT(*) as num_won FROM nobel_prizes
    GROUP BY country
    HAVING num_won > 1;
```

SQL: JOIN on tables

```
CREATE TABLE students (
    id INTEGER PRIMARY KEY,
    name TEXT,
    email TEXT,
    teacher_id INTEGER);
CREATE TABLE teachers (
    id INTEGER PRIMARY KEY,
    name TEXT,
    email TEXT);
/* Show students next to their teachers */
SELECT students.name, teachers.name AS teacher_name
    FROM students
    JOIN teachers
    ON students.teacher_id = teachers.id;
/* Show students next to their teachers,
   even if they don't have a teacher assigned
*/
SELECT students.name, teachers.name AS teacher_name
```

```
    FROM students
    LEFT OUTER JOIN teachers
    ON students.teacher_id = teachers.id;
```

SQL: UPDATE and DELETE

```
CREATE TABLE inventory (
    id INTEGER PRIMARY KEY,
    item TEXT,
    price REAL);
/* It's sale time for Halloween merchandise! */
UPDATE inventory SET price = 20.00 WHERE id = 4;
SELECT * FROM inventory;
/* Woo, it got sold! */
DELETE FROM inventory WHERE id = 4;
SELECT * FROM inventory;
```

More complex queries with AND/OR

```
CREATE TABLE exercise_logs
    (id INTEGER PRIMARY KEY AUTOINCREMENT,
    type TEXT,
    minutes INTEGER,
    calories INTEGER,
    heart_rate INTEGER);
SELECT * FROM exercise_logs WHERE calories > 50 ORDER BY calories;
/* AND */
SELECT * FROM exercise_logs WHERE calories > 50 AND minutes < 30;
/* OR */
SELECT * FROM exercise_logs WHERE calories > 50 OR heart_rate > 100;
```

Querying IN subqueries

```
CREATE TABLE exercise_logs
    (id INTEGER PRIMARY KEY AUTOINCREMENT,
    type TEXT,
    minutes INTEGER,
    calories INTEGER,
    heart_rate INTEGER);
SELECT * FROM exercise_logs WHERE type = "biking" OR type = "hiking" OR type =
"tree climbing" OR type = "rowing";
/* IN */
SELECT * FROM exercise_logs WHERE type IN ("biking", "hiking", "tree climbing",
"rowing");
CREATE TABLE drs_favorites
    (id INTEGER PRIMARY KEY,
    type TEXT,
    reason TEXT);
SELECT type FROM drs_favorites;
SELECT * FROM exercise_logs WHERE type IN (
    SELECT type FROM drs_favorites);
SELECT * FROM exercise_logs WHERE type IN (
    SELECT type FROM drs_favorites WHERE reason = "Increases cardiovascular
health");
/* LIKE */
SELECT * FROM exercise_logs WHERE type IN (
    SELECT type FROM drs_favorites WHERE reason LIKE "%cardiovascular%");
```

Restricting grouped results with HAVING

```
CREATE TABLE exercise_logs
    (id INTEGER PRIMARY KEY AUTOINCREMENT,
    type TEXT,
```

```
    minutes INTEGER,
    calories INTEGER,
    heart_rate INTEGER);
SELECT * FROM exercise_logs;
SELECT type, SUM(calories) AS total_calories FROM exercise_logs GROUP BY type;
SELECT type, SUM(calories) AS total_calories FROM exercise_logs
    GROUP BY type
    HAVING total_calories > 150;
SELECT type, AVG(calories) AS avg_calories FROM exercise_logs
    GROUP BY type
    HAVING avg_calories > 70;
SELECT type FROM exercise_logs GROUP BY type HAVING COUNT(*) >= 2;
```

Calculating results with CASE

```
CREATE TABLE exercise_logs
    (id INTEGER PRIMARY KEY AUTOINCREMENT,
    type TEXT,
    minutes INTEGER,
    calories INTEGER,
    heart_rate INTEGER);
SELECT * FROM exercise_logs;
SELECT COUNT(*) FROM exercise_logs WHERE heart_rate > 220 - 30;
/* 50-90% of max*/
SELECT COUNT(*) FROM exercise_logs WHERE
    heart_rate >= ROUND(0.50 * (220-30))
    AND  heart_rate <= ROUND(0.90 * (220-30));
/* CASE */
SELECT type, heart_rate,
```

```
    CASE
        WHEN heart_rate > 220-30 THEN "above max"
        WHEN heart_rate > ROUND(0.90 * (220-30)) THEN "above target"
        WHEN heart_rate > ROUND(0.50 * (220-30)) THEN "within target"
        ELSE "below target"
    END as "hr_zone"
FROM exercise_logs;
SELECT COUNT(*),
    CASE
        WHEN heart_rate > 220-30 THEN "above max"
        WHEN heart_rate > ROUND(0.90 * (220-30)) THEN "above target"
        WHEN heart_rate > ROUND(0.50 * (220-30)) THEN "within target"
        ELSE "below target"
    END as "hr_zone"
FROM exercise_logs
GROUP BY hr_zone;
```

Difference between WHERE and HAVING clause:

1. WHERE clause can be used with - Select, Insert, and Update statements, where as HAVING clause can only be used with the Select statement.

2. WHERE filters rows before aggregation (GROUPING), where as, HAVING filters groups, after the aggregations are performed.

3. Aggregate functions cannot be used in the WHERE clause, unless it is in a sub query contained in a HAVING clause, whereas, aggregate functions can be used in Having clause.

Mathematical functions in MySQL

1. COUNT()

```
SELECT COUNT(salary)
FROM employees
WHERE salary >= 40000;
```

2. AVG()

```
SELECT AVG(salary)
FROM employees;
```
3. SUM()

```
SELECT SUM(Price)
FROM Products;
```
4. MIN()

```
SELECT MIN(column_name)
FROM table_name;
```
5. MAX()

```
SELECT MAX(column_name)
FROM table_name;
```

What is a Composite PRIMARY KEY?

Composite PRIMARY KEY is a primary key created on more than one column (combination of multiple fields) in a table.

What is the difference between Having and Where clause?

Where clause is used to fetch data from a database that specifies particular criteria whereas a Having clause is used along with 'GROUP BY' to fetch data that meets particular criteria specified by the Aggregate functions. Where clause cannot be used with Aggregate functions, but the Having clause can.

What is the order of SQL SELECT?

Order of SQL SELECT statement is as follows

SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY.

What do you mean by foreign key?

A Foreign key is a field which can uniquely identify each row in another table. And this constraint is used to specify a field as Foreign key. That is, this field points to primary key of another table. This usually creates a kind of link between the two tables.

```
CREATE TABLE Orders
(
O_ID int NOT NULL,
ORDER_NO int NOT NULL,
C_ID int,
PRIMARY KEY (O_ID),
FOREIGN KEY (C_ID) REFERENCES Customers(C_ID)
)
```

What is a primary key, a foreign key and unique key?

Primary key is a field in the table which uniquely identifies a row. It cannot be NULL

Foreign key is a field in one table which is a primary key in another table. A relationship is created between the two tables by referencing the foreign key of one table with the primary key of another table.

Unique Key uniquely identifies a record in a table. There can be many unique key constraints defined on a table.

What is the difference between BETWEEN and IN condition operators?

The BETWEEN operator is used to display rows based on a range of values. The values can be numbers, text, and dates as well. BETWEEN operator gives us the count of all the values occurs between a particular range.

The IN condition operator is used to check for values contained in a specific set of values. IN operator is used when we have more than one value to choose.

Which are the different character-manipulation functions in SQL?

CONCAT: join two or more values together.
SUBSTR: used to extract the string of specific length.

LENGTH: return the length of the string in numerical value.

INSTR: find the exact numeric position of a specified character.

LPAD: padding of the left-side character value for right-justified value.

RPAD: padding of right-side character value for left-justified value.

TRIM: remove all the defined character from the beginning, end or both beginning and end.

REPLACE: replace a specific sequence of character with other sequences of character.

What is the difference between TRUNCATE and DELETE commands?

TRUNCATE is a DDL command whereas DELETE is a DML command. Hence DELETE operation can be rolled back, but TRUNCATE operation cannot be rolled back. WHERE clause can be used with DELETE and not with TRUNCATE.

What is a join in SQL? What are the types of joins?

An SQL Join statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are:

INNER JOIN: The INNER JOIN keyword selects all rows from both the tables as long as the condition satisfies. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be same.

Notice that, Pam employee record which does not have a matching DepartmentId in departments table is eliminated from the result-set.select tblperson.name,tblperson.email,tblperson.city,tblgender.gender from tblperson Inner Join tblgender on tblperson.genderid=tblgender.id

LEFT JOIN:This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of join. The rows for which there is no matching row on right side, the result-set will contain null. LEFT JOIN is also known as LEFT OUTER JOIN

select tblperson.name,tblperson.email,tblperson.city,tblgender.gender from tblperson left Join tblgender on tblperson.genderid=tblgender.id

RIGHT JOIN:RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of join. The rows for which there is no matching row on left side, the result-set will contain null. RIGHT JOIN is also known as RIGHT OUTER JOIN.

select tblperson.name,tblperson.email,tblperson.city,tblgender.gender from tblperson right Join tblgender on tblperson.genderid=tblgender.id

FULL JOIN: FULL JOIN creates the result-set by combining result of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both the tables. The rows for which there is no matching, the result-set will contain NULL values.

select tblperson.name,tblperson.email,tblperson.city,tblgender.gender from tblperson full Join tblgender on tblperson.genderid=tblgender.id

What is Cross-Join?

Cross join produces a result set which is the number of rows in the first table multiplied by a number of rows in the second table if no WHERE clause is used along with Cross join. This kind of result is known as Cartesian Product. If suppose, Where clause is used in cross join then the query will work like an Inner join.

select tblperson.name,tblperson.email,tblperson.city,tblgender.gender from tblperson cross Join tblgender

What is the difference between INNER JOIN and RIGHT JOIN

INNER JOIN returns only the matching rows between the tables involved in the JOIN, where as RIGHT JOIN returns all the rows from the right table including the NON-MATCHING rows.

What is the difference between INNER JOIN and FULL JOIN

FULL JOIN returns all the rows from both the left and right tables including the NON-MATCHING rows.

What is the Difference between INNER JOIN and JOIN

There is no difference they are exactly the same. Similarly there is also no difference between

LEFT JOIN and LEFT OUTER JOIN

RIGHT JOIN and RIGHT OUTER JOIN

FULL JOIN and FULL OUTER JOIN

Retrieve only the non matching rows from the left table

select tblperson.name,tblperson.email,tblperson.city,tblgender.gender from tblperson left Join tblgender on tblperson.genderid=tblgender.id where tblgender.id is null

Retrieve only the non matching rows from the right table

select tblperson.name,tblperson.email,tblperson.city,tblgender.gender from tblperson right Join tblgender on tblperson.genderid=tblgender.id where tblperson.genderid is null

Retrieve only the non matching rows from both the left and right table

select tblperson.name,tblperson.email,tblperson.city,tblgender.gender from tblperson full outer Join tblgender on tblperson.genderid=tblgender.id where tblgender.id is null or tblperson.genderid is null

Self Join Query

A MANAGER is also an EMPLOYEE. Both the, EMPLOYEE and MANAGER rows, are present in the same table. Here we are joining tblEmployee with itself using different alias names, E for Employee and M for Manager. We are using LEFT JOIN, to get the rows with ManagerId NULL. You can see in the output TODD's record is also retrieved, but the MANAGER is NULL. If you replace LEFT JOIN with INNER JOIN, you will not get TODD's record.

Select E.Name as Employee, M.Name as Manager

from Employee E

Left Join Employee M

On E.id = M.mid

In short, joining a table with itself is called as SELF JOIN. SELF JOIN is not a different type of JOIN. It can be classified under any type of JOIN - INNER, OUTER or CROSS Joins. The above query is, LEFT OUTER SELF Join.

Inner Self Join Employee table:

```
Select E.Name as Employee, M.Name as Manager
from Employee E
Inner Join Employee M
On E.id = M.mid
```

Cross Self Join Employee table:

```
Select E.Name as Employee, M.Name as Manager
from Employee E
Cross Join Employee M
```

Replacing NULL value using CASE Statement

```
SELECT E.Name as Employee, CASE WHEN M.Name IS NULL THEN 'No Manager'
  ELSE M.Name END as Manager
FROM  Employee E
LEFT JOIN Employee M
ON   E.mid = M.id
```

Replacing NULL value using COALESCE() function: COALESCE() function, returns the first NON NULL value.

```
SELECT E.Name as Employee, COALESCE(M.Name, 'No Manager') as Manager
FROM Employee E
LEFT JOIN Employee M
ON E.mid = M.id
```

What does UNION do? What is the difference between UNION and UNION ALL?

UNION merges the contents of two structurally-compatible tables into a single combined table. The difference between UNION and UNION ALL is that UNION will omit duplicate records whereas UNION ALL will include duplicate records.

It is important to note that the performance of UNION ALL will typically be better than UNION, since UNION requires the server to do the additional work of removing any duplicates. So, in cases where is is certain that there will not be any duplicates, or where having duplicates is not a problem, use of UNION ALL would be recommended for performance reasons.

Select Id, Name, Email from tblIndiaCustomers

UNION ALL

Select Id, Name, Email from tblUKCustomers

For this reason, UNION ALL is much faster than UNION.

Note: For UNION and UNION ALL to work, the Number, Data types, and the order of the columns in the select statements should be same.

Difference between JOIN and UNION

JOINS and UNIONS are different things. However, this question is being asked very frequently now.

UNION combines the result-set of two or more select queries into a single result-set which includes all the rows from all the queries in the union,

where as JOINS, retrieve data from two or more tables based on logical relationships between the tables.

In short, UNION combines rows from 2 or more tables, where JOINS combine columns from 2 or more table.

What is Stored Procedures in SQL ?

Stored Procedures are created to perform one or more DML operations on Database. It is nothing but the group of SQL statements that accepts some input in the form of parameters and performs some task and may or may not returns a value.

Example:
Imagine a table named with emp_table stored in Database. We are Writing a Procedure to update a Salary of Employee with 1000.

```
CREATE or REPLACE PROCEDURE INC_SAL(eno IN NUMBER, up_sal OUT
NUMBER)
IS
BEGIN
UPDATE emp_table SET salary = salary+1000 WHERE emp_no = eno;
COMMIT;
SELECT sal INTO up_sal FROM emp_table WHERE emp_no = eno;
END;
```

Difference between user defined functions(UDF) and stored procedures(SO) in PL/SQL?

1.SP may or may not return a value but UDF must return a value.

2.SP can have input/output parameter but UDF only has input parameter.

3.We can call UDF from SP but cannot call SP from a function.

4.We cannot use SP in SQL statement like SELECT, INSERT, UPDATE, DELETE, MERGE etc. but we can use them with UDF.

5.We can use try-catch exception handling in SP but we cannot do that in UDF.

6.We can use transaction in SP but it is not possible in UDF.

Explain different types of index?

- ✓ **Unique Index**
  This index does not allow the field to have duplicate values if the column is unique indexed. If primary key is defined, a unique index can be applied automatically.
- ✓ **Clustered Index**
  This index reorders the physical order of the table and searches based on the basis of key values. Each table can only have one clustered index.
- ✓ **Non-Clustered Index**
  It doesn't alter the physical order of the table and maintain a logical order of the data. Each table can have many non-clustered indexes.

What is the difference between clustered and non-clustered indexes?

One table can have only one clustered index but multiple nonclustered indexes.

Clustered indexes can be read rapidly rather than non-clustered indexes.

Clustered indexes store data physically in the table or view and non-clustered indexes do not store data in the table as it has separate structure from the data row.

What is a View?

A view is like a subset of a table which is stored logically in a database. A view is a virtual table. It contains rows and columns similar to a real table. The fields in the view are fields from one or more real tables. Views do not contain data of their own. They are used to restrict access to the database or to hide data complexity.

Now let's create a view, using the JOINS query, we have just written.

```
Create View vWEmployeesByDepartment

as

Select Id, Name, Salary, Gender, DeptName

from tblEmployee

join tblDepartment

on tblEmployee.DepartmentId = tblDepartment.DeptId
```

To select data from the view, SELECT statement can be used the way, we use it with a table.

```
SELECT * from vWEmployeesByDepartment
```

When this query is executed, the database engine actually retrieves the data from the underlying base tables, tblEmployees and tblDepartments. The View itself, doesnot store any data by default. However, we can change this default behaviour, which we will talk about in a later session. So, this is the reason, a view is considered, as just, a stored query or a virtual table.

What are the advantages of Views?

Some of the advantages of Views are

1.Views occupy no space
2.Views are used to simply retrieve the results of complicated queries that need to be executed often.
3.Views are used to restrict access to the database or to hide data complexity.

What is a Trigger?

A Trigger is a SQL procedure that initiates an action in response to an event (Insert, Delete or Update) occurs. When a new Employee is added to an Employee_Details table, new records will be created in the relevant tables such as Employee_Payroll, Employee_Time_Sheet etc.

What is a CURSOR?

CURSOR is a Handle OR Pointer To The CONTEXT AREA

What is The CURSOR Usage?

Using a CURSOR, The PL/SQL program can control the CONTEXT AREA, As the SQL Statement is being processed.

What are the CURSOR Features?

CURSOR Allows to FETCH and process Rows returned by a SELECT statement, One Row at a time.
A CURSOR is named, such that it can be referenced by the PL/SQL programmer dynamically at run time.

What is normalization and what are the normal forms?

Normalization is a process in database design to minimize data redundancy and dependency. The database is divided into two or more tables and relationships are defined between them.

First Normal Form: Every record is unique in a table and is identified by a primary or a composite key

Second Normal Form: The table must be in First Normal Form and it should have a single column as its primary key

Third Normal Form: The table must be in Second Normal Form and must have no transitive functional dependencies. I.e a non-key column must not be dependent on another non-key column within the same table

What is de-normalization and when do you go for it?

De-normalization is a technique sometimes used to improve performance so the table design allows redundant data to avoid complex joins. If the application involves heavy read operations then de-normalization is used at an expense of the write operations performance.

Difference between Stored Procedure and Trigger?

Triggers get invoked automatically when the event specified in it gets fulfilled, it can't return any value.So user have no control over it's invocation.
Whereas stored procedure can be called as per application requirement and can get output parameters. stored procedure is a block of code contains sql statements. stored procedure is a precompiled block of code,which need not be compiled for every time calling.

Important Queries

How to print duplicate rows in a table?

Let us consider below table.
Name Section
abc CS1
bcd CS2
abc CS1
In the above table, we can find duplicate row using below query.

```
SELECT name, section FROM tbl
GROUP BY name, section
HAVING COUNT(*) > 1
```

Fetch only odd rows from table

SELECT E.EMPID,E.PROJECT,E.SALARY FROM(SELECT *,ROW_NUMBER() OVER(ORDER BY EMPID) AS ROWNUMBER FROM EMPLOYEESALARY E WHERE E.ROWNUMBER%2=1

Create an empty table with same structure as so other table

SELECT * INTO NEWTABLE FROM EMPLOYEEDETAILS WHERE 1=0;

Write a SQL query to find current daya-time

MYSQL

SELECT NOW();

SQLSERVER

SELECT getdate();

ORACLE

SELECT SYSDATE FROM DUAL;

How to Get the names of the table in SQL?

SELECT * FROM INFORMATION_SCHEMA.TABLES

Return employee record with max salary

Select * from employee where sal=(select max(sal) from employee);

Select highest salary in employee table

SELECT MAX(SAL) FROM EMPLOYEE;

Select 2nd highest salary in employee table

SELECT MAX(SAL) FROM EMPLOYEE WHERE SAL NOT IN (SELECT MAX(SAL) FROM EMPLOYEE);

How to find Nth highest salary from a table?

SQLSERVER:

SELECT TOP 1 SALARY

FROM(SELECT DISTINCT TOP N SALARY

FROM EMPLOYEE ORDER BY SALARY DESC)

ORDER BY SALARY ASC;

MYSQL:

SELECT SALARY FROM EMPLOYEE ORDER BY SALARY DESC LIMIT N-1,1;

WITHOUT USING LIMIT and TOP

SELECT e1.SALARY FROM EMPLOYEE e1

WHERE N = (SELECT COUNT(DISTINCT(e2.SALARY))

FROM EMPLOYEE e2 WHERE e2.SALARY > e1.SALARY);

Select range of employee based on id

Select * from employee where employee_id between 2003 and 2008

Return employee name,highest salary and department name

select e.employee_name,e.salary,d.department_name from Employee e Inner Join Department d ON (e.department_id=d.department_id) where salary in(select max(salary) from employee);

Display the names of employees who are not working as managers.

select * from emp minus (select * from emp where empno in (select mgr from emp));

(or)

select * from emp where empno not in (select mgr from emp where mgr is not null);

(or)

select * from emp e where empno not in (select mgr from emp where e.empno=mgr)

Display those employees who are working as manager?

select * from emp where empno in(select mgr from emp);

Display the manager who is having maximum number of employees working under him?

Select mgr from emp group by mgr having count(*)=(select max(count(mgr)) from emp group by mgr);

Return highest salary,employee name, department name for each department

Select e.employee)name,e.salary,d.department_name from Employee e Inner Join Department d ON (e.department_id=d.department_id) where salary in (select max(salary) from Employee group by department_id);

Delete duplicate data from table only first data remains constant.

DELETE M1
From managers M1, managers M2
Where M2.Name = M1.Name AND M1.Id>M2.Id;

Drop primary key on employee table

ALTER TABLE EMPLOYEE drop CONSTRAINT EMPLOYEE_PK;

Get department wise minimum salary from employee table order by salary ascending

SELECT DEPARTMENT,min(SALARY) MINSALARY FROM EMPLOYEE GROUP BY DEPARTMENT ORDER BY MINSALARY ASC.

How to select unique records from a table?

Select DISTINCT EmployeeId, EmployeeName from Employee

Display the list of employees who have joined the company before 30th June 90 or after 31st dec 90.

select * from emp where hiredate between '30-jun-1990' and '31-dec-1990';

Write an sql query to find number of employees according to gender whose DOB is between 01/01/1960 to 31/12/1975.

SELECT COUNT(*),sex from Employees where DOB BETWEEN '01/01/1960' AND '31/12/1975' GROUP BY sex;

Display the names of employees whose name starts with alphabet S.

select ename from emp where ename like 'S%';

Display employee names for employees whose name ends with alphabet S.

select ename from emp where ename like '%S';

Display the names of employees whose names have second alphabet S in their names.

select ename from emp where ename like '_S%';

Write a SQL query to fetch common records between two tables.

SELECT * FROM EmployeeSalary
INTERSECT
SELECT * FROM ManagerSalary

Write a SQL query to fetch records that are present in one table but not in another table.

SELECT * FROM EmployeeSalary
MINUS
SELECT * FROM ManagerSalary

Write a SQL query to fetch top n records?

In mySQL using LIMIT,
SELECT * FROM EmployeeSalary ORDER BY Salary DESC LIMIT N
In SQL server using TOP command,
SELECT TOP N * FROM EmployeeSalary ORDER BY Salary DESC
In Oracle using ROWNUM,
SELECT * FROM (SELECT * FROM EmployeeSalary ORDER BY Salary DESC)
WHERE ROWNUM <= N;

Find the factorial of a number in pl/sql

declare
-- it gives the final answer after computation
fac number :=1;

-- given number n

```
-- taking input from user
n number := &1;

-- start block
begin

-- start while loop
while n > 0 loop

-- multiple with n and decrease n's value
fac:=n*fac;
n:=n-1;
end loop;
-- end loop

-- print result of fac
dbms_output.put_line(fac);

-- end the begin block
end;
```

Count odd and even digits in a number in PL/SQL

```
--Odd and Even digits in a number
--in PL/SQL
DECLARE
--num variable declared
--num assign with a number
num NUMBER := 123456;
--len varible char declared
len VARCHAR2(20);
--cntvarible declared
cnt1 NUMBER(5) := 0;
cnt2 NUMBER(5) := 0;
BEGIN
--for loop go from 1 to length of the number
FOR i IN 1..Length(num)
LOOP
len := Substr(num, i, 1);
IF mod(len, 2) != 0 THEN
cnt1 := cnt1 + 1;
```

```
ELSE
cnt2:=cnt2+1;
END IF;
END LOOP;
--end loop
dbms_output.Put_line('Odd Digits: '
|| cnt1);
dbms_output.Put_line('Even Digits: '
|| cnt2);
--display result
END;
--end program
```

Check whether a string is palindrome or not in PL/SQL

```
DECLARE
-- Declared variables are s, l, t .
-- These variables are of same data type VARCHAR.
s VARCHAR2(10) := 'abccba';
l VARCHAR2(20);
t VARCHAR2(10);
BEGIN
FOR i IN REVERSE 1..Length(s) LOOP
l := Substr(s, i, 1);
-- here || are used for concatenation of string.
t := t
||''
||l;
END LOOP;
IF t = s THEN
dbms_output.Put_line(t
||''
||' is palindrome');
ELSE
dbms_output.Put_line(t
||''
||' is not palindrome');
END IF;
END;
-- Program End
```

Count no. of characters and words in a string in PL/SQL

```
DECLARE
-- Declare required variables
str VARCHAR2(40) := 'Praveen Oruganti'';
noofchars NUMBER(4) := 0;
noofwords NUMBER(4) := 1;
s CHAR;
BEGIN
FOR i IN 1..Length(str) LOOP
s := Substr(str, i, 1);
-- Count no. of characters
noofchars := noofchars + 1;
-- Count no. of words
IF s = ' ' THEN
noofwords := noofwords + 1;
END IF;
END LOOP;
dbms_output.Put_line('No. of characters:'
||noofchars);
dbms_output.Put_line('No. of words: '
||noofwords);
END;
-- Program End
```

Swap two numbers in PL/SQL without using temp

```
DECLARE
-- declare variable num1, num2
-- of datatype number
num1 NUMBER;
num2 NUMBER;
BEGIN
num1 := 1000;
num2 := 2000;
-- print result before swapping
dbms_output.Put_line('Before');
dbms_output.Put_line('num1 = '
|| num1
||' num2 = '
|| num2);
-- swapping of numbers num1 and num2
num1 := num1 + num2;
```

```
num2 := num1 - num2;
num1 := num1 - num2;
-- print result after swapping
dbms_output.Put_line('After');
dbms_output.Put_line('num1 = '
|| num1
||' num2 = '
|| num2);
END;
-- Program End
```

No. of vowels and consonants in a given string in PL/SQL

```
DECLARE
-- Here variable V is varchar datatype
-- and flag variable is number datatype
-- variable c is char datatype .
v VARCHAR2(400) := 'Praveen Oruganti'';
noofvowels NUMBER := 0;
noofconsonants NUMBER := 0;
C CHAR;
BEGIN
FOR i IN 1..Length(v) LOOP
c := Substr(v, i, 1);
-- Check if the current character is vowel
IF c IN ( 'A', 'E', 'I', 'O', 'U' )
OR c IN ( 'a', 'e', 'i', 'o', 'u' ) THEN
noofvowels := noofvowels + 1;
-- Else current character is a consonant except space
ELSE
IF c NOT IN ( ' ' ) THEN
noofconsonants := noofconsonants + 1;
END IF;
END IF;
END LOOP;
dbms_output.Put_line('No. of Vowels: '
|| noofvowels);
dbms_output.Put_line('No. of Consonants: '
|| noofconsonants);
END;
-- Program End
```

Sum of the first and last digit of a number in PL/SQL

```
DECLARE
-- declare variables are A, B, C and S
-- these are same datatype integer
a INTEGER := 14598;
b INTEGER := 0;
C INTEGER := 0;
s INTEGER;
BEGIN
IF a > 9 THEN
c := Substr(a, 1, 1);
b := Substr(a, Length(a), 1);
s := b + c;
ELSE
s := a;
END IF;
dbms_output.Put_line('Sum of the first and last digit is '
||s);
END;
-- Program End
```