**SQL Assignment 4**

1. Explain different types of views. Demonstrate with suitable examples.

## What is a View?

Views are a special version of tables in SQL. They provide a virtual table environment for various complex operations. You can select data from multiple tables, or you can select specific data based on certain criteria in views. It does not hold the actual data; it holds only the definition of the view in the data dictionary.

The view is a query stored in the data dictionary, on which the user can query just like they do on tables. It does not use the physical memory, only the query is stored in the data dictionary. It is computed dynamically, whenever the user performs any query on it. Changes made at any point in view are reflected in the actual base table.

The view has primarily two purposes:

* Simplify the complex SQL queries.
* Provide restriction to users from accessing sensitive data.

## Types of Views

* **Simple View:** A view based on only a single table, which doesn't contain GROUP BY clause and any functions.
* **Complex View:** A view based on multiple tables, which contain GROUP BY clause and functions.
* **Inline View:** A view based on a subquery in FROM Clause, that subquery creates a temporary table and simplifies the complex query.
* **Materialized View:** A view that stores the definition as well as data. It creates replicas of data by storing it physically.

## **Inline View**

**An inline view is a SELECT statement in the FROM-clause of another SELECT statement to create a temporary table that could be referenced by the SELECT statement. Inline views are utilized for writing complex SQL queries without join and subqueries operations. This is called a temporary table because a duplicate copy of the data returned by the stored subquery wasn't stored in the database. In Oracle and in the Postgres community, this temporary table is called an inline view. It is referred to as a subselect.**

## **Materialized View**

**Materialized view replicates the retrieved data physically. This replicated data can be reused without executing the view again. This type of view is also known as "SNAPSHOTS". Materialized view reduce the processing time to regenerate the whole data. It helps remote users to replicate data locally and improve query performance. The challenging part here is to synchronize the changes in materialized views underlying tables.**

1. What is the difference between function and stored procedure? Write syntax for creating functions and stored procedures.

**1. SP** *may or may not*return a value but **UDF** **must** return a value. The return statement of the function returns control to the calling program and returns the result of the function.   
**Example:**

SP ->

create or replace procedure GEEKS(x int)

is

y int;

begin

.....

UDF->

FUNCTION GEEKS(x int)

/\*return statement so we must return value in function \*/

RETURN int

IS

y int;

BEGIN

.....

**2. SP** can have *input/output* parameters but **UDF** **only** has input parameters.   
**Example:**

SP ->

CREATE OR REPLACE PROCEDURE Factorial(x IN NUMBER, result OUT NUMBER)

is

begin

....

UDF ->

FUNCTION Factorial(x IN NUMBER) /\* only input parameter \*/

return NUMBER

is

result NUMBER;

begin

.....

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

Calling UDF cal() inside SP square() but reverse is not possible.

set serveroutput on;

declare

a int;

c int;

function cal(temp int)

return int

as

ans int;

begin

ans:=temp\* temp;

return ans;

end;

procedure square(x in int, ans out int)

is

begin

dbms\_output.put\_line('calling function in procedure');

ans:= cal(x);

end;

begin

a:=6;

square(a, c);

dbms\_output.put\_line('the answer is '|| c);

end;

**OUTPUT:**

calling function in the procedure

the answer is 36

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

**5.** **SP** is not allowed to be used in SELECT statements, but **UDF** can be used.

**6.** **SP** cannot be used anywhere in the WHERE/HAVING/SELECT part of SQL queries, but **UDF** can be used.

**7.** The execution plan can be re-used in **SP** but in **UDF** it will be compiled every time.

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

**9.** We can use transactions in **SP** but it is not possible in **UDF**.

**10.** We can consider **UDF**as an expression but it is not possible in **SP.**

**11. SP**cannot be used in the join clause but it is possible in**UDF**as a resultset.

**12. SP** can have both table variables and temporary tables but **UDF**can have only table variables as it does not permit the use of temporary tables.

**13.** Advantages of using **UDF**–

* Enables faster execution
* Modular Programming
* Can reduce network traffic (data)

      Advantages of using **SP**–

* Improved security measures
* Client/server traffic is reduced.
* Programming abstraction and efficient code reuse
* Pre-compiled Execution

1. What is an index in SQL? What are the different types of indexes in SQL?

## **What are Indexes in SQL?**

Indexes in SQL are specialized lookup tables that are used by the database search engine to accelerate data retrieval.

In simple terms, an index in SQL is a tool used to quickly identify rows with specific column values. If there were no indexes, the SQL server would have to start with the first row and then go through the entire table until it discovers the relevant rows. This method is known as a full-table scan and can be highly inefficient for large tables.

Type of Index

## **Primary Key Index**

A primary key is a field or a combination of fields in a database table that uniquely identifies each record (row) in that table. A primary key index is an automatically generated index associated with the primary key column(s) to enhance data retrieval and enforce data uniqueness.

## **Unique Key Index**

A unique index in a relational database is a data structure that enforces the uniqueness constraint on one or more columns within a table. Its primary purpose is to ensure that values stored in the indexed column(s) are unique across all records in the table.

## **Clustered Index**

A clustered index sorts and stores the rows of a table based on the values in one or more specified columns. Each table can have only one clustered index, and the choice of the clustering column(s) significantly impacts how data is stored and retrieved.

## **Non-Clustered Index**

A non-clustered index is a type of index used in relational databases to improve the efficiency of data retrieval operations. Unlike clustered indexes, which affect the physical order of data rows within a table, non-clustered indexes create separate data structures to allow fast access to specific data subsets. This means that non-clustered indexes do not rearrange the physical organization of data, but rather create a separate structure to facilitate quicker access to the data.

1. Showcase an example of exception handling in SQL stored procedure.

# Exception Handling in PL/SQL

* An exception is an error which disrupts the normal flow of program instructions. PL/SQL provides us the exception block which raises the exception thus helping the programmer to find out the fault and resolve it.

There are two types of exceptions defined in PL/SQL

1. User defined exception.
2. System defined exceptions.

Syntax to write an exception

**WHEN** exception **THEN**

statement;

*DECLARE  
declarations section;*

*BEGIN  
executable command(s);*

*EXCEPTION  
WHEN exception1 THEN  
statement1;  
WHEN exception2 THEN  
statement2;  
[WHEN others THEN]  
/\* default exception handling code \*/*

*END;*

**Note:**  
**When other** keyword should be used only at the end of the exception handling block as no exception handling part present later will get executed as the control will exit from the block after executing the WHEN OTHERS.

1. **System defined exceptions:**  
   These exceptions are predefined in PL/SQL which get raised WHEN certain **database rule is violated.**  
   System-defined exceptions are further divided into two categories:
   1. Named system exceptions.
   2. Unnamed system exceptions.
   3. **Named system exceptions:** They have a predefined name by the system like ACCESS\_INTO\_NULL, DUP\_VAL\_ON\_INDEX, LOGIN\_DENIED etc. the list is quite big.

So we will discuss some of the most commonly used exceptions:

Lets create a table geeks.

create table geeks(g\_id int , g\_name varchar(20), marks int);

insert into geeks values(1, 'Suraj',100);

insert into geeks values(2, 'Praveen',97);

insert into geeks values(3, 'Jessie', 99);

1. **NO\_DATA\_FOUND**: It is raised WHEN a SELECT INTO statement returns *no* rows. For eg:

|  |
| --- |
| DECLARE     temp varchar(20);    BEGIN     SELECT g\_id into temp from geeks where g\_name='GeeksforGeeks';  exception     WHEN no\_data\_found THEN        dbms\_output.put\_line('ERROR');        dbms\_output.put\_line('there is no name as');        dbms\_output.put\_line('GeeksforGeeks in geeks table');  end; |

Output:

ERROR

there is no name as GeeksforGeeks in geeks table

1. **TOO\_MANY\_ROWS**:It is raised WHEN a SELECT INTO statement returns *more* than one row.

|  |
| --- |
| DECLARE     temp varchar(20);  BEGIN  -- raises an exception as SELECT  -- into trying to return too many rows     SELECT g\_name into temp from geeks;     dbms\_output.put\_line(temp);  EXCEPTION     WHEN too\_many\_rows THEN        dbms\_output.put\_line('error trying to SELECT too many rows');  end; |

Output:

error trying to SELECT too many rows

1. **VALUE\_ERROR**:This error is raised WHEN a statement is executed that resulted in an arithmetic, numeric, string, conversion, or constraint error. This error mainly results from programmer error or invalid data input.

|  |
| --- |
| DECLARE     temp number;  BEGIN     SELECT g\_name  into temp from geeks where g\_name='Suraj';     dbms\_output.put\_line('the g\_name is '||temp);  EXCEPTION     WHEN value\_error THEN     dbms\_output.put\_line('Error');     dbms\_output.put\_line('Change data type of temp to varchar(20)');  END; |

Output:

Error

Change data type of temp to varchar(20)

1. **ZERO\_DIVIDE** = raises exception WHEN dividing with zero.

|  |
| --- |
| DECLARE     a int:=10;     b int:=0;     answer int;    BEGIN     answer:=a/b;     dbms\_output.put\_line('the result after division is'||answer);  exception     WHEN zero\_divide THEN        dbms\_output.put\_line('dividing by zero please check the values again');        dbms\_output.put\_line('the value of a is '||a);        dbms\_output.put\_line('the value of b is '||b);  END; |

Output:

dividing by zero please check the values again

the value of a is 10

the value of b is 0

**Unnamed system exceptions:**Oracle doesn’t provide name for some system exceptions called unnamed system exceptions.These exceptions *don’t* occur frequently.These exceptions have two parts *code and an associated message*.  
The way to handle to these exceptions is to *assign name* to them using **Pragma**

**EXCEPTION\_INIT**  
Syntax:

PRAGMA EXCEPTION\_INIT(exception\_name, -error\_number);

error\_number are pre-defined and have negative integer range from -20000 to -20999.

**Example:**

|  |
| --- |
| DECLARE     exp exception;     pragma exception\_init (exp, -20015);     n int:=10;  BEGIN     FOR i IN 1..n LOOP        dbms\_output.put\_line(i\*i);           IF i\*i=36 THEN              RAISE exp;           END IF;     END LOOP;  EXCEPTION     WHEN exp THEN        dbms\_output.put\_line('Welcome to GeeksforGeeks');  END; |

Output:

1

4

9

16

25

36

Welcome to GeeksforGeeks

**User defined exceptions:**  
This type of users can create their own exceptions according to the need and to raise these exceptions explicitly ***raise*** command is used.

*Example:*

* Divide non-negative integer x by y such that the result is greater than or equal to 1.

From the given question we can conclude that there exist two exceptions

* + Division be zero.
  + If result is greater than or equal to 1 means y is less than or equal to x.

|  |
| --- |
| DECLARE     x int:=&x; /\*taking value at run time\*/     y int:=&y;     div\_r float;     exp1 EXCEPTION;     exp2 EXCEPTION;  BEGIN     IF y=0 then         raise exp1;     ELSEIF y > x then        raise exp2;     ELSE        div\_r:= x / y;        dbms\_output.put\_line('the result is '||div\_r);    END IF;  EXCEPTION     WHEN exp1 THEN        dbms\_output.put\_line('Error');        dbms\_output.put\_line('division by zero not allowed');     WHEN exp2 THEN        dbms\_output.put\_line('Error');        dbms\_output.put\_line('y is greater than x please check the input');  END; |

*Input 1:* x = 20

y = 10

*Output:* the result is 2

*Input 2:* x = 20

y = 0

*Output:*

Error

division by zero not allowed

*Input 3:* x=20

y = 30

*Output:<.em>*

*Error*

*y is greater than x please check the input*

1. Create a SQL function to split strings into rows on a given character?

Input String: Stephen;peter;berry;Olivier;caroline;

|  |
| --- |
| Stephen |
| Peter |
| Berry |
| Oliver |
| Caroline |

The SQL INSTR function allows you to find the starting location of a substring within a string. It is used by Oracle SQL and MySQL; many other SQL implementations have functions which are the exact or near equivalent.

It’s worth you while to get acquainted with basic SQL functions such as INSTR, since [SQL programming is a skill that’s in high demand](https://www.udemy.com/course/sql-fundamentals/?tc=blog.sqlinstr&couponCode=half-off-for-blog). To find out more about SQL, you might want to check out [some of the online classes that help you master Master MySQL](https://www.udemy.com/course/learning-mysql5/?tc=blog.sqlinstr&couponCode=half-off-for-blog).

## SQL INSTR: The Basics

INSTR searches for a substring within a string and returns its starting location in the string, using the syntax INSTR(string,substring). This means that if you tell INSTR to look for “berry” in “strawberry” it will return 6, because “berry” starts at position 6 in “strawberry”:

INSTR('strawberry','berry')

In an actual SQL query, you might use it like this:

SELECT INSTR (Fruit, 'berry')

FROM Food

WHERE Fruit = 'strawberry';

Note that if INSTR does not find the substring, it returns the number 0 (zero).

## Oracle SQL vs. MySQL

The basic two-argument version of INSTR is common to both Oracle SQL and MySQL, but in Oracle SQL, you can optionally use two other arguments — the starting position, and which occurrence of the string you’re looking for, like this:

INSTR('she sells seashells','s',2,3)

This tells Oracle SQL to look for the third instance of “s” starting from position 2. It would skip the “s” in “she” (because it’s in position 1), and return 11, the position of the first “s” in “seashells”.

Note that regardless of the starting position (or, as described below, the direction of the search), the position returned by INSTR and the other string-index search functions described here is always based on the substring’s position in the string itself, starting at the beginning of the string, and not its position in relation to the start or direction of the search.

## Oracle: Refining the Search

What happens if you switch the last two arguments?

INSTR('she sells seashells','s',3,2)

Now it starts at position 3 (still skipping the first “s”) and looks for the second instance of “s”, returning 9, the position of the second “s” in “shells”.

The Oracle SQL version of INSTR also allows you to count backward, by entering the position argument as a negative number:

INSTR('she sells seashells','s',-2,5)

This will tell Oracle SQL to start at the second character from the end of the string (the last “l” in “seashells”) and search forward until it finds the 5th instance of “s” — in this case, the first letter of “she”, which will return 1.

1. What is a temporary and a variable table? Write suitable syntax to create temporary tables and variable tables.

## **Introduction**

Before reading this article, I will recommend you read these articles:

* [Temporary tables in SQL Server](https://www.c-sharpcorner.com/UploadFile/rohatash/temporary-tables-in-sql-server-2012/)
* [Table Variables in SQL Server](https://www.c-sharpcorner.com/UploadFile/75a48f/table-variable-in-sql-server/)

Temporary Tables and Table variables in SQL Server, both have their own pros and cons. We need to decide which one to use and when.

### **Differences between Temporary Table and Table variable in SQL Server**

1. The table variable (@table) is created in the memory. Whereas, a Temporary table (#temp) is created in the tempdb database. However, if there is memory pressure the pages belonging to a table variable may be pushed to tempdb.
2. Table variables cannot be involved in transactions, logging, or locking. This makes @table faster than #temp. So table variable is faster than the temporary table.
3. Temporary tables are allowed CREATE INDEXes whereas, Table variables aren’t allowed CREATE INDEX instead they can have an index by using Primary Key or Unique Constraint.
4. A table variable can be passed as a parameter to functions and stored procedures while the same cannot be done with Temporary tables.
5. Temporary tables are visible in the created routine and also in the child routines. Whereas, Table variables are only visible in the created routine.
6. A temporary table allows Schema modifications, unlike Table variables.

### **Table Variable in SQL Server – Example**

Table variable is a very useful programming construct, like that of any other variable.

DECLARE @TStudent TABLE

(

RollNo INT IDENTITY(1,1),

StudentID INT,

Name INT

)

--Insert data to Table variable @TStudent

INSERT INTO @TStudent(StudentID,Name)

SELECT DISTINCT StudentID, Name FROM StudentMaster ORDER BY StudentID ASC

--Select data from Table variable @TStudent

SELECT \* FROM @TStudent

--Next batch

GO

SELECT \* FROM @TStudent --gives error

DECLARE @TStudent TABLE

(

RollNo INT IDENTITY(1,1),

StudentID INT,

Name INT

)

--Insert data to Table variable @TStudent

INSERT INTO @TStudent(StudentID,Name)

SELECT DISTINCT StudentID, Name FROM StudentMaster ORDER BY StudentID ASC

--Select data from Table variable @TStudent

SELECT \* FROM @TStudent

--Next batch

GO

SELECT \* FROM @TStudent --gives error

SQL

Copy

### **Temporary Tables in SQL Server – Example**

In SQL Server, based on the scope and behavior, temporary tables are of two types,

Local Temporary Tables (#temp)

Global Temporary Tables (##temp)

CREATE TABLE #StudentTemp

(

StudentID int,

Name varchar(50),

Address varchar(150)

)

GO

INSERT INTO #StudentTemp VALUES ( 1, 'Dipendra','Pune');

GO

SELECT \* FROM #StudentTemp

CREATE TABLE #StudentTemp

(

StudentID int,

Name varchar(50),

Address varchar(150)

)

GO

INSERT INTO #StudentTemp VALUES ( 1, 'Dipendra','Pune');

GO

SELECT \* FROM #StudentTemp