**SQL Assignment 4**

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

**System Defined Views**

The System Defined Views are predefined views that already exist in the SQL Server database, such as Tempdb, Master, and temp. Each of the databases has its own properties and functions. System Defined Views will be automatically attached to all User Defined databases. And these provide information about the database, tables, and all the properties of the database and tables. There are three types of System defined views, Information Schema, Catalog View, and Dynamic Management View.

*Catalog View*

Catalog views provide an efficient way to obtain, present, and transform custom forms of information and are used to return information used by the SQL server. They do not include any information about backup, replication, or maintenance plans, etc. These views are used to access metadata of databases, and the names and column names are descriptive, helping a user to query what is expected.

SELECT object\_id, principal\_id, type

FROM sys.objects

WHERE name = 'employee';

*Dynamic Management View*

The administer can get information about the server state to diagnose problems, monitor the health of the server instance, and tune performance through these views. The Server-scoped Dynamic Management View is only stored in the Master database, whereas the Database-scoped Dynamic Management View is stored in each database.

CREATE PROC TwoSELECTS

**AS**

SELECT emp\_no, job from works\_on where emp\_no BETWEEN 1000 **and** 9999;

SELECT emp\_no, emp\_lname FROM employee where emp\_fname LIKE 'S%';

GO

SELECT is\_hidden hidden ,column\_ordinal ord,

name, is\_nullable nul, system\_type\_id id

FROM

sys.dm\_exec\_describe\_first\_result\_set('TwoSELECTS', **NULL**, 0) ;

*Information Schema*

There are twenty different schema views in the SQL server. They are used to display the physical information of the database, such as tables, constraints, columns, and views. A constraint is used on a particular column in a table to ensure that certain data rules are followed for the column.

**User Defined Views**

*Complex View*

These views can contain more than one base table or can be constructed on more than one base table, and they contain a group by clause, join conditions, an order by clause. Group functions can be used here, and it contains groups of data. Complex views cannot always be used to perform DML operations.But unlike Simple Views, Complex Views can contain group by, pseudocolumn like rownum, distinct, columns defined by expressions, also Insert, delete, and update cannot be applied directly on complex views.

*Create VIEW dept\_income AS*

*SELECT d.Name as DepartmentName, sum(e.salary) as TotalSalary*

*FROM Employees e*

*JOIN Departments d on e.DepartmentId = d.id*

*GROUP BY d.Name;*

*Simple View*

These views can only contain a single base table or can be created only from one table. Group functions such as MAX(), COUNT(), etc., cannot be used here, and it does not contain groups of data.Simple View does not contain group by, pseudocolumn like rownum, distinct, columns defined by expressions. Simple view also does not include NOT NULL columns from the base tables.

*CREATE VIEW new\_employees\_details AS*

*SELECT E.id, Fname, Salary, Hire\_date*

*FROM Employees E*

*WHERE hire\_date > date '2015-01-01';*

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

A-

| Functions | Procedures |
| --- | --- |
| A function has a return type and returns a value. | A procedure does not have a return type. But it returns values using the OUT parameters. |
| You cannot use a function with Data Manipulation queries. Only Select queries are allowed in functions. | You can use DML queries such as insert, update, select etc… with procedures. |
| A function does not allow output parameters | A procedure allows both input and output parameters. |
| You cannot manage transactions inside a function. | You can manage transactions inside a procedure. |
| You cannot call stored procedures from a function | You can call a function from a stored procedure. |
| You can call a function using a select statement. | You cannot call a procedure using select statements. |
| Stored Procedures are pre-compiled objects which are compiled for the first time and its compiled format is saved, which executes (compiled code) whenever it is called. | A function is compiled and executed every time whenever it is called. A function must return a value and cannot modify the data received as parameters. |
| Function allows only SELECT statement in it. | The procedure allows SELECT as well as DML(INSERT/UPDATE/DELETE) statement. |
| Function can be embedded in a SELECT statement. | Procedures cannot be utilized in a SELECT statement |
| Functions can be used in the [SQL](https://en.wikipedia.org/wiki/SQL) statements anywhere in the WHERE/HAVING/SELECT section | Stored Procedures cannot be used in the [SQL](https://en.wikipedia.org/wiki/SQL) statements anywhere in the WHERE/HAVING/SELECT section |

Syntax:

*-Stored Procedures-*

*delimiter &&*

*create procedure top\_players()*

*begin*

*select player\_name,country,goals from players where goals>100;*

*end &&*

*delimiter ;*

*-Functions-*

*delimiter //*

*create function no\_of\_years(date1 date) returns int deterministic*

*begin*

*declare date2 date;*

*select current\_date() into date2;*

*return year(date2)-year(date1);*

*end //*

*delimiter ;*

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

A-Indexes are used to speed-up query process in SQL Server, resulting in high performance.

There are six various types of indexes in SQL server:

Clustered Index-Clustered Index store and sort rows of data in a view or table depending on their central values. A clustered index defines the order in which data is physically stored in a table. Table data can be sorted in only way, therefore, there can be only one clustered index per table. In SQL Server, the primary key constraint automatically creates a clustered index on that particular column.

Non-Clustered Index-A non-clustered index doesn’t sort the physical data inside the table. In fact, a non-clustered index is stored at one place and table data is stored in another place. When a query is issued against a column on which the index is created, the database will first go to the index and look for the address of the corresponding row in the table. It will then go to that row address and fetch other column values. It is due to this additional step that non-clustered indexes are slower than clustered indexes.

Column Store Index- A columnstore index is a technology for storing, retrieving, and managing data by using a columnar data format, called a columnstore. A columnstore is data that's logically organized as a table with rows and columns, and physically stored in a column-wise data format.

Filtered Index- A filtered index is an optimized nonclustered index especially suited to cover queries that select from a well-defined subset of data. It uses a filter predicate to index a portion of rows in the table.

Hash Index- It works by converting input records into an array of buckets. Each bucket has the same number of records as all other buckets in the table.

Unique Index- Unique indexes are indexes that help maintain data integrity by ensuring that no rows of data in a table have identical key values.

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

A-The TRY CATCH construct allows you to gracefully handle exceptions in SQL Server.

Following is the example:

*CREATE PROC sample(*

*@a decimal,*

*@b decimal,*

*@c decimal output*

*) AS*

*BEGIN*

*BEGIN TRY*

*SET @c = @a / @b;*

*END TRY*

*BEGIN CATCH*

*SELECT*

*ERROR\_NUMBER() AS ErrorNumber*

*,ERROR\_SEVERITY() AS ErrorSeverity*

*,ERROR\_STATE() AS ErrorState*

*,ERROR\_PROCEDURE() AS ErrorProcedure*

*,ERROR\_LINE() AS ErrorLine*

*,ERROR\_MESSAGE() AS ErrorMessage;*

*END CATCH*

*END;*

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 |

A-

*select substring\_index(substring\_index("Stephen;peter;berry;Olivier;caroline;", ';', 1), ';', -1)*

*union*

*select substring\_index(substring\_index("Stephen;peter;berry;Olivier;caroline;", ';', 2), ';', -1)*

*union*

*select substring\_index(substring\_index("Stephen;peter;berry;Olivier;caroline;", ';', 3), ';', -1)*

*union*

*select substring\_index(substring\_index("Stephen;peter;berry;Olivier;caroline;", ';', 4), ';', -1)*

*union*

*select substring\_index(substring\_index("Stephen;peter;berry;Olivier;caroline;", ';', 5), ';', -1)*

*order by -1;*

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

A-

*Table variable (@table) is created in the memory. Whereas, a Temporary table (#temp) is created in the tempdb database. However, if there is a memory pressure the pages belonging to a table variable may be pushed to tempdb.*

*⇒ Table variables cannot be involved in transactions, logging or locking. This makes @table faster then #temp. So table variable is faster then temporary table.*

*⇒ Temporary tables are allowed CREATE INDEXes whereas, Table variables aren’t allowed CREATE INDEX instead they can have index by using Primary Key or Unique Constraint.*

*⇒ Table variable can be passed as a parameter to functions and stored procedures while the same cannot be done with Temporary tables.*

*⇒ Temporary tables are visible in the created routine and also in the child routines. Whereas, Table variables are only visible in the created routine.*

*⇒ Temporary table allows Schema modifications unlike Table variables.*

***Temporary Table:***

*create temporary table employee*

*(*

*id int auto\_increment Primary key,*

*emp\_name varchar(500),*

*emp\_address varchar(500),*

*emp\_dept\_id int*

*);*

*insert into employee(emp\_name, emp\_address,emp\_dept\_id)*

*values*

*('Nisarg Upadhyay','Mehsana',10),*

*('Nirali U','Mehsana',20),*

*('Milan J','Mehsana',5);*

*Select \* from employee;*

***Table 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*