**SQL Assignment 4**

1. Explain different types of views. Demonstrate with suitable examples.
2. Simple Views: A simple view is based on a single base table, presenting a subset of its columns and rows. It acts as a filter on the original table, allowing users to access only the data they need.

CREATE VIEW SalaryView AS

SELECT EmployeeID, FirstName, Salary

FROM Employees;

1. Complex Views: Complex views are based on multiple base tables, combining data from various sources to present a more comprehensive and unified view of the data. They can include joins and aggregations.

VIEW CustomerOrdersView AS

SELECT c.CustomerID, c.CustomerName, o.OrderID, o.OrderDate, o.TotalAmount

FROM Customers c

JOIN Orders o ON c.CustomerID = o.CustomerID;

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

* Function:
* A function is a database object that takes input parameters, performs some computation or manipulation on the data, and returns a single value.
* Functions are designed to be used in SQL queries or expressions, and their return value can be directly utilized in the query.
* Functions are deterministic, meaning they always produce the same output for the same input parameters.
* They cannot perform data manipulation (e.g., INSERT, UPDATE, DELETE) on tables.
* Functions are often used for calculations, data formatting, or to encapsulate reusable logic.
* Stored Procedure:
* A stored procedure is a database object that can have input and output parameters, and it contains a series of SQL statements and procedural code to perform a specific task.
* Procedures are mainly used to encapsulate complex business logic or to execute multiple SQL statements as a single unit.
* Stored procedures can perform data manipulation (e.g., INSERT, UPDATE, DELETE) on tables.
* They are often used for tasks like data validation, data transformation, or implementing complex business workflows.
* Syntax for creating a stored procedure in SQL (PostgreSQL):

CREATE PROCEDURE procedure\_name(parameter1 data\_type, parameter2 data\_type, ...)

LANGUAGE plpgsql

AS $$

BEGIN

-- Procedure logic here

END;

$$;

* Syntax for creating a function in SQL (PostgreSQL):

REATE FUNCTION function\_name(parameter1 data\_type, parameter2 data\_type, ...)

RETURNS return\_data\_type AS

$$

BEGIN

-- Function logic here

RETURN result\_value;

END;

$$

LANGUAGE plpgsql;

Sql

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

* In SQL, an index is a database object used to improve the performance of queries by allowing faster data retrieval. An index is a data structure associated with a table that organizes the data in a way that facilitates efficient searching and sorting operations. Instead of scanning the entire table, the database engine can use the index to locate the desired data more quickly, reducing the query's execution time
* Different types of indexes in SQL:
* B-Tree Index:
* B-Tree (Balanced Tree) index is the most common and default type of index used in SQL databases.
* It organizes the data in a balanced tree-like structure, making it efficient for range searches and equality searches.
* Suitable for columns that have a wide range of distinct values, such as primary keys and other frequently queried columns.
* Bitmap Index:
* Bitmap index uses a bitmap for each distinct value in the indexed column.
* It is efficient for columns with a low cardinality, i.e., columns that have a small number of distinct values.
* Bitmap indexes work well for columns with binary (0/1) attributes or columns with a small set of possible values.
* Clustered Index:
* In SQL Server, a clustered index determines the physical order of data in a table.
* The data rows in the table are sorted based on the values of the clustered index key.
* A table can have only one clustered index, and it affects how the data is stored on disk.
* Non-Clustered Index:
* A non-clustered index is a separate structure from the table, and it contains a copy of the indexed columns along with a pointer to the actual data rows.
* A table can have multiple non-clustered indexes.
* Non-clustered indexes are generally smaller in size compared to clustered indexes and are suitable for columns with frequently used search criteria.
* Unique Index:
* A unique index ensures that the indexed column contains unique values, and it disallows duplicate entries in that column.
* It is used to enforce the uniqueness constraint on a column or a combination of columns.
* Covering Index:
* A covering index includes all the columns required to fulfill a query, so the database engine doesn't need to access the actual table for data retrieval.
* Covering indexes can significantly improve query performance, especially for queries with SELECT statements that match the index columns.

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

* Suppose we have a table named "Employees" with columns: "EmployeeID," "FirstName," "LastName," and "Salary." The goal is to create a stored procedure that inserts a new employee record into the table. However, if the insertion fails due to a unique constraint violation (e.g., duplicate EmployeeID), we want to catch the exception and perform specific actions.

-- Assume we have a table named "Employees"

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Salary NUMERIC

);

-- Create the stored procedure with exception handling

CREATE OR REPLACE PROCEDURE InsertEmployee(

IN empID INT,

IN firstName VARCHAR(50),

IN lastName VARCHAR(50),

IN salary NUMERIC

)

LANGUAGE plpgsql

AS

$$

BEGIN

-- Attempt to insert the new employee record

INSERT INTO Employees (EmployeeID, FirstName, LastName, Salary)

VALUES (empID, firstName, lastName, salary);

-- If the insertion succeeds, we are done

-- and can exit the procedure

RAISE NOTICE 'Employee inserted successfully';

EXCEPTION

-- If a unique constraint violation occurs (duplicate EmployeeID)

WHEN unique\_violation THEN

-- Catch the exception and log an error message

RAISE WARNING 'Failed to insert employee. EmployeeID already exists.';

-- Additional actions could be performed here, such as rolling back a transaction

-- ROLLBACK; -- Uncomment this line if you want to roll back the transaction

END;

$$;

* In this example, the stored procedure "InsertEmployee" takes four input parameters (empID, firstName, lastName, salary) and tries to insert a new employee record into the "Employees" table. If the insertion is successful, it raises a notice stating "Employee inserted successfully." However, if a unique constraint violation occurs (due to a duplicate EmployeeID), the exception block is triggered.
* In the exception block, we catch the "unique\_violation" exception and raise a warning with a custom error message, indicating that the insertion failed due to a duplicate EmployeeID. You can add more actions to the exception block, such as logging the error in a separate error log table, rolling back transactions, or sending notifications to administrators.

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 |

* To split a string into rows on a given character in SQL, you can create a user-defined table-valued function (TVF). Unfortunately, SQL doesn't have a built-in function to split strings directly, so we'll need to create a custom function to achieve this.
* Below is an example of how to create a SQL function to split a string into rows on the ';' character:

-- Create the table-valued function

CREATE OR REPLACE FUNCTION SplitStringToRows(input\_string TEXT, delimiter TEXT)

RETURNS TABLE (split\_value TEXT)

AS

$$

BEGIN

RETURN QUERY

SELECT trim(split\_part(input\_string, delimiter, i))

FROM generate\_series(1, length(input\_string) - length(replace(input\_string, delimiter, '')) + 1) AS i;

END;

$$

LANGUAGE plpgsql;

* Now, let's use the function to split the given input string:

-- Call the function with the input string and the delimiter ';'

SELECT split\_value

FROM SplitStringToRows('Stephen;peter;berry;Olivier;caroline;', ';');

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

* Temporary Tables: Temporary tables are created for a specific session or transaction and exist only for a temporary period. They are particularly useful when you need to store and manipulate intermediate data within a session. Temporary tables are automatically dropped at the end of the session or when the transaction is committed or rolled back, depending on the database system.
* Syntax to create a temporary table in SQL:

-- Create a temporary table (PostgreSQL and SQL Server syntax)

CREATE TEMPORARY TABLE temp\_table\_name (

column1 datatype,

column2 datatype,

...

);

-- Create a temporary table (MySQL syntax)

CREATE TEMPORARY TABLE temp\_table\_name (

column1 datatype,

column2 datatype,

...

);

* Table Variables: Table variables are variables that can hold a result set, acting as a temporary storage for data during the execution of a batch, stored procedure, or function. They are often used in Transact-SQL (T-SQL) in SQL Server. Table variables are automatically deallocated at the end of the batch, stored procedure, or function in which they are declared.
* Syntax to declare and use a table variable in SQL Server:

-- Declare a table variable

DECLARE @table\_variable TABLE (

column1 datatype,

column2 datatype,

...

);