**SQL Assignment 5**

1. Explain how SQL Query keyword statements are executed in order.

* In SQL, the execution of a query involves several steps that are typically performed in a specific order. The process can be broadly summarized as follows:
* Parsing:
* The first step in query execution is parsing the SQL query. During parsing, the database system checks the query's syntax to ensure it follows the rules and grammar of the SQL language. If there are any syntax errors, the query will be rejected and an error message will be generated.
* Lexical Analysis:
* After parsing, the query goes through lexical analysis, where it is broken down into smaller units called tokens. Tokens are individual components of the query, such as keywords, identifiers, operators, and literals. The lexical analysis helps the database system understand the query's structure and identify its elements.
* Semantic Analysis:
* In this step, the database system performs semantic analysis to check the query's semantics and validity. It verifies that the referenced tables, columns, and other database objects exist and that the query can be executed without violating any constraints or rules.
* Query Optimization:
* Once the query has passed the previous steps, the database system looks for the most efficient way to execute the query. It considers various execution plans and strategies to access and manipulate data from the underlying tables. The goal is to find the best execution plan that minimizes the query's processing time and resource consumption.
* Execution Plan Generation:
* Based on the query optimization, the database system generates an execution plan. The execution plan is a set of instructions that outline the steps the database engine needs to perform to retrieve the requested data and process it according to the query.
* Data Retrieval and Manipulation:
* With the execution plan in hand, the database engine starts retrieving data from the relevant tables based on the plan's instructions. This involves reading and filtering data from the underlying tables, joining tables if necessary, and applying any other specified operations (e.g., sorting, aggregation).
* Result Set Formation:
* As the data is retrieved and processed, the result set is formed. The result set contains the rows and columns that satisfy the query's conditions and operations.
* Presentation:
* The final step is presenting the result set to the user or application that executed the query. The presentation format could be in the form of a table, list, or any other suitable structure, depending on the client application or interface.

1. Explain the advantages of stored procedures and their syntax in relation to recompiling stored procedures.

* Advantages of Stored Procedures: Stored procedures offer several advantages that make them a valuable tool in database management and application development:

1. Performance Optimization: Stored procedures are precompiled and stored in the database, which reduces the need for repetitive parsing and optimization of SQL statements. This can lead to improved performance, especially for complex queries that are executed frequently.
2. Modularity and Reusability: Stored procedures promote modular programming by encapsulating SQL logic into a single unit. This enhances code reusability, as the same stored procedure can be called from multiple parts of the application, reducing code duplication.
3. Enhanced Security: By using stored procedures, you can control access to underlying tables and data. Application users may have permission to execute a stored procedure without directly accessing the underlying tables, providing an additional layer of security.
4. Ease of Maintenance: As stored procedures are centralized in the database, changes and updates to the SQL logic can be made in one place. This simplifies maintenance and reduces the risk of inconsistencies in the application code.
5. Reduced Network Traffic: When using stored procedures, only the procedure call and its parameters are sent over the network, rather than the entire SQL query. This can significantly reduce network traffic, especially for complex queries with large result sets.
6. Transaction Management: Stored procedures allow you to group multiple SQL statements into a single unit. This facilitates better transaction management, as you can ensure that all related operations are performed together in an atomic manner.

* Syntax of Stored Procedures (using PL/pgSQL in PostgreSQL):

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

LANGUAGE plpgsql

AS

$$

BEGIN

-- Procedure logic here

END;

$$;

* Recompiling Stored Procedure Syntax (using PL/pgSQL in PostgreSQL):

-- Recompile a specific stored procedure

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

-- Recompile all stored procedures in the schema

DO $$

BEGIN

FOR r IN (SELECT proname FROM pg\_proc WHERE pronamespace = 'schema\_name'::regnamespace)

LOOP

EXECUTE 'ALTER PROCEDURE ' || r.proname || '()';

END LOOP;

END $$;

1. Give an example of the derived table.

SELECT

e.EmployeeID,

e.FirstName,

e.LastName,

d.DepartmentName

FROM

Employees e

JOIN

(SELECT DepartmentID, DepartmentName FROM Departments) AS d

ON

e.DepartmentID = d.DepartmentID;

* In this example, the derived table is the subquery (SELECT DepartmentID, DepartmentName FROM Departments) AS d. It creates a virtual table 'd' containing the "DepartmentID" and "DepartmentName" columns from the "Departments" table. This subquery is used within the main query to retrieve the department names for each employee based on their "DepartmentID."
* The main query performs a JOIN between the "Employees" table (aliased as 'e') and the derived table (aliased as 'd') on the common "DepartmentID" column. The result is a list of employees along with their corresponding department names.
* Derived tables are powerful tools for organizing and simplifying complex queries, and they help improve the readability and maintainability of SQL code. They are especially useful when dealing with large and complex queries involving multiple tables and conditions.

1. What is the database's trigger? Explain the different forms of triggers that can be found in the database.

* Different Forms of Triggers in a Database:
* Data Manipulation Language (DML) Triggers:
* DML triggers are fired in response to data manipulation operations, such as INSERT, UPDATE, and DELETE, performed on a table.
* They allow you to enforce business rules, perform data validation, or log changes made to the data.
* DML triggers can be "AFTER" triggers, which execute after the DML operation, or "INSTEAD OF" triggers, which execute instead of the DML operation.
* Data Definition Language (DDL) Triggers:
* DDL triggers are fired in response to data definition language statements that modify the database schema, such as CREATE, ALTER, or DROP statements.
* They are less common than DML triggers but can be useful for auditing schema changes or enforcing specific rules when altering the database structure.
* Instead-of Triggers:
* Instead-of triggers are specifically used with views.
* When a DML operation is performed on a view, an Instead-of trigger allows you to provide custom logic to handle the operation on the underlying base tables.
* This is particularly useful when you want to make a view updatable, even though it references multiple base tables.
* System Triggers (Database-Level Triggers):
* System triggers are associated with database-level events, such as startup, shutdown, login, or logout.
* They allow you to execute specific actions when these events occur, such as recording login attempts or initializing certain settings upon database startup.
* Instead-of Triggers on User-Defined Types (UDTs):
* In some databases that support user-defined types (UDTs), you can create Instead-of triggers on UDTs to define custom behavior when using these types.

CREATE OR REPLACE FUNCTION update\_last\_order\_date()

RETURNS TRIGGER AS

$$

BEGIN

UPDATE Customers

SET LastOrderDate = NEW.OrderDate

WHERE CustomerID = NEW.CustomerID;

RETURN NEW;

END;

$$

LANGUAGE plpgsql;

CREATE TRIGGER after\_insert\_order

AFTER INSERT ON Orders

FOR EACH ROW

EXECUTE FUNCTION update\_last\_order\_date();

In this example, the trigger "after\_insert\_order" is fired after an INSERT operation on the "Orders" table. It executes the "update\_last\_order\_date" function, which updates the "LastOrderDate" column in the "Customers" table for the corresponding customer whenever a new order is inserted.

1. What are the benefits and drawbacks of triggers?

* Benefits of Triggers:
* Data Integrity: Triggers help enforce data integrity by allowing the enforcement of complex business rules and constraints automatically. They can prevent invalid data from being inserted, updated, or deleted.
* Business Logic Encapsulation: Triggers encapsulate business logic within the database, ensuring consistent application of rules across different applications and avoiding code duplication.
* Automation: Triggers automate tasks and actions that need to be performed in response to specific events, reducing the need for manual intervention and improving system efficiency.
* Auditing and Logging: Triggers can be used to log changes made to the data, providing an audit trail for tracking data modifications and user actions.
* Complex Data Transformations: Triggers allow for complex data transformations that may not be easily achieved through simple SQL statements.
* Consistency: Triggers help maintain data consistency by automatically updating related data in other tables when changes are made, ensuring that related information stays in sync.
* Drawbacks of Triggers:
* Hidden Logic: Triggers introduce logic that is not immediately visible in the application code, which can make it harder to understand and maintain the overall system.
* Performance Impact: Poorly designed or overly complex triggers can have a negative impact on database performance, especially if they are fired frequently.
* Debugging Complexity: Triggers can complicate the debugging process, as errors or issues related to trigger logic may not be obvious.
* Order of Execution: When multiple triggers exist on a table, the order of their execution can be important and may lead to unexpected results if not handled properly.
* Dependency: Applications become dependent on triggers, which can make it challenging to modify or remove them later without affecting the application code.
* Locking and Deadlocks: Triggers can cause locking issues, leading to potential deadlocks if not managed carefully.
* Testing and Maintenance: Triggers add complexity to testing and maintenance efforts, requiring thorough testing to ensure they work as intended.

1. Create a stored procedure to call other stored procedures.

* Let's assume we have two existing stored procedures: "sp\_UpdateSalary" and "sp\_InsertLog." The first procedure updates an employee's salary, and the second procedure inserts a log entry for the salary update.

-- Create the stored procedure that calls other stored procedures

CREATE OR REPLACE PROCEDURE sp\_CallOtherProcedures(

IN empID INT,

IN newSalary NUMERIC

)

LANGUAGE plpgsql

AS

$$

BEGIN

-- Call the first stored procedure to update the salary

PERFORM sp\_UpdateSalary(empID, newSalary);

-- Call the second stored procedure to insert a log entry

PERFORM sp\_InsertLog(empID, 'Salary updated');

-- You can perform additional logic here if needed

-- ...

END;

$$;

* In this example, we create a new stored procedure called "sp\_CallOtherProcedures" that takes two input parameters: "empID" (employee ID) and "newSalary" (the new salary value to be updated).
* Within the procedure, we use the "PERFORM" statement to call the other two existing stored procedures: "sp\_UpdateSalary" and "sp\_InsertLog." The "PERFORM" statement is used to execute the specified stored procedures without returning any result set.