**DAY 4 Assignment Questions**

**Stored Procedure Practice Questions**

* **Basic Stored Procedure**: Write a simple stored procedure named **GetEmployeeCount** that returns the total number of employees in the "Employees" table.
* **Input Parameters**: Create a stored procedure named **GetEmployeesByDepartment** that takes a department name as input and returns all the employees who belong to that department.
* **Output Parameters**: Write a stored procedure named **GetEmployeeInfo** that takes an employee ID as input and returns the employee's name, job title, and hire date using output parameters.
* **Error Handling**: Create a stored procedure named **InsertEmployee** that inserts a new employee into the "Employees" table. However, if the insertion fails due to duplicate employee ID or any other reason, the procedure should return an error message.
* **Transactions**: Design a stored procedure named **TransferFunds** that transfers a specified amount from one bank account to another. The procedure should handle transactions to ensure data consistency in case of failures.
* **Dynamic SQL**: Write a stored procedure named **GetEmployeeByFilter** that allows the user to search for employees based on different filters, such as name, job title, and department. The procedure should use dynamic SQL to construct the query based on the provided filter values.
* **Recursive Stored Procedure**: Implement a stored procedure named **GetEmployeeHierarchy** that takes an employee ID as input and returns the hierarchical structure of employees under the given employee. Assume that the "Employees" table has a self-referencing foreign key to represent the manager-subordinate relationship.
* **Table-Valued Parameters:** Create a stored procedure named **InsertEmployees** that takes a table-valued parameter containing employee data (employee ID, name, department, etc.) and inserts multiple employees into the "Employees" table in a single call.
* **Returning Resultsets:** Write a stored procedure named **GetEmployeeProjects** that takes an employee ID as input and returns two result sets:
  + The first result set should contain the employee's details (name, department, etc.).
  + The second result set should contain all the projects the employee is currently working on.
* **Dynamic Resultsets:** Design a stored procedure named **GetEmployeesByProject** that takes a project ID as input and returns the employees who are working on that project. The number of columns and their data types should be dynamic, based on the number and types of attributes in the "Employees" table.

**User Defined Functions Practice Questions**

1. Create a UDF named **GetFullName** that concatenates the first name and last name of an employee with a space in between and Write a query to get the full names of all employees using the **GetFullName** UDF.
2. Create a UDF named **GetSalaryGrade** that takes an employee's salary as input and returns their salary grade based on the following conditions:
   1. If the salary is less than 50000, return 'Low'.
   2. If the salary is between 50000 and 80000 (inclusive), return 'Medium'.
   3. If the salary is greater than 80000, return 'High'.
   4. Write a query to get the EmployeeID, FullName, and SalaryGrade for all employees using the **GetSalaryGrade** UDF.
3. Create a UDF named **GetAverageSalary** that calculates and returns the average salary of all employees and Write a query to get the average salary of all employees using the **GetAverageSalary** UDF.
4. Create a Scalar UDF to calculate the discounted price for a product. The discount percentage is 10% and Use the Scalar UDF to get the discounted prices for all products.
5. Create a Table-Valued UDF to return all products within a specific price range and Use the Table-Valued UDF to get products within a price range.
6. Create an Inline Table-Valued UDF to get all products in a specific category and Use the Inline Table-Valued UDF to get all products from a specific category.