Advanced SQL

1. Write a SQL query to find the names and salaries of the employees that take the minimal salary in the company. Use a nested SELECT statement.

```
SELECT Salary,
       FirstName + ' ' + LastName AS [Name]
FROM Employees
WHERE Salary =
       (SELECT MIN(Salary) FROM Employees)
```

2. Write a SQL query to find the names and salaries of the employees that have a salary that is up to 10% higher than the minimal salary for the company.

```
SELECT Salary,
       FirstName + ' ' + LastName AS [Name]
FROM Employees
WHERE Salary <
       (SELECT 1.1 * MIN(Salary) FROM Employees)
```

3. Write a SQL query to find the full name, salary and department of the employees that take the minimal salary in their department. Use a nested SELECT statement.

```
SELECT FirstName, LastName, Salary, DepartmentID
FROM Employees e
WHERE Salary =
    (
    SELECT MIN(Salary)
    FROM Employees
    WHERE DepartmentID = e.DepartmentID
    )
```

4. Write a SQL query to find the average salary in the department #1.

```
SELECT AVG(Salary)
FROM Employees
WHERE DepartmentID = 1
```

5. Write a SQL query to find the average salary in the "Sales" department.

```sql
SELECT AVG(Salary)
FROM Employees e
       JOIN Departments d
         ON d.DepartmentID = e.DepartmentID
WHERE d.Name = 'Sales'
```

6. Write a SQL query to find the number of employees in the "Sales" department.

```sql
SELECT Count(*)
FROM Employees e
       JOIN Departments d
         ON d.DepartmentID = e.DepartmentID
WHERE d.Name = 'Sales'
```

7. Write a SQL query to find the number of all employees that have manager.

```sql
SELECT Count(ManagerID)
FROM Employees
```

8. Write a SQL query to find the number of all employees that have no manager.

```sql
SELECT Count(*) - Count(ManagerID)
FROM Employees
```

9. Write a SQL query to find all departments and the average salary for each of them.

```sql
SELECT DepartmentID ,AVG(Salary) AS [Average Salary]
FROM Employees
GROUP BY (DepartmentID)
```

10.Write a SQL query to find the count of all employees in each department and for each town.

```
SELECT COUNT(*) AS [Employees count],
              d.Name AS [Department],
              t.Name AS [Town]
FROM Employees e
       JOIN Departments d
         ON e.DepartmentID = d.DepartmentID
       JOIN Addresses a
         ON e.AddressID = a.AddressID
       JOIN Towns t
         ON a.TownID = t.TownID
GROUP BY t.Name, d.Name
```

11.Write a SQL query to find all managers that have exactly 5 employees. Display their first name and last name.

```
SELECT m.FirstName, m.LastName
FROM Employees e
       JOIN Employees m
         ON e.ManagerID = m.EmployeeID
GROUP BY m.FirstName, m.LastName
HAVING COUNT(*) = 5
```

12.Write a SQL query to find all employees along with their managers. For employees that do not have manager display the value "(no manager)".

```
SELECT e.FirstName + ' ' +  e.LastName AS Employee,
       ISNULL(m.FirstName, 'no') + ' ' + ISNULL(m.LastName, 'manager') AS Manager
FROM Employees e
  LEFT JOIN Employees m
    ON e.ManagerID = m.EmployeeID
```

13.Write a SQL query to find the names of all employees whose last name is exactly 5 characters long. Use the built-in LEN(str) function.

```
SELECT FirstName + ' ' +  LastName AS [Employee]
FROM Employees
WHERE LEN(LastName) = 5
```

14. Write a SQL query to display the current date and time in the following format "day.month.year hour:minutes:seconds:milliseconds". Search in Google to find how to format dates in SQL Server.

```
SELECT FORMAT(GETDATE(), 'dd.MM.yyyy HH:mm:ss:fff');
```

15. Write a SQL statement to create a table Users. Users should have username, password, full name and last login time. Choose appropriate data types for the table fields. Define a primary key column with a primary key constraint. Define the primary key column as identity to facilitate inserting records. Define unique constraint to avoid repeating usernames. Define a check constraint to ensure the password is at least 5 characters long.

```
CREATE TABLE Users
(
    UserId INT PRIMARY KEY IDENTITY(1,1) NOT NULL,
    Username nvarchar(50) UNIQUE NOT NULL,
    Pass nvarchar(50) CHECK(LEN(Pass) > 4) NOT NULL,
    Fullname nvarchar(200) NOT NULL,
    LastLogin smalldatetime
)
```

16. Write a SQL statement to create a view that displays the users from the Users table that have been in the system today. Test if the view works correctly.

```
CREATE VIEW   UsersToday
AS
       SELECT *
       FROM Users
       WHERE DATEDIFF(DAY, LastLogin, GETDATE()) = 0
```

17. Write a SQL statement to create a table Groups. Groups should have unique name (use unique constraint). Define primary key and identity column.

```
CREATE TABLE Groups
(
    GroupId INT PRIMARY KEY IDENTITY(1,1),
    Name nvarchar(100) UNIQUE
)
```

18. Write a SQL statement to add a column GroupID to the table Users. Fill some data in this new column and as well in the Groups table. Write a SQL statement to add a foreign key constraint between tables Users and Groups tables.

```
ALTER TABLE Users
    ADD GroupID INT

ALTER TABLE Users
    ADD CONSTRAINT FK_UsersGroup FOREIGN KEY(GroupID) REFERENCES Groups(GroupID)
```

19. Write SQL statements to insert several records in the Users and Groups tables.

```
INSERT INTO Groups
VALUES
    ('Students'),
    ('Proffessors')

INSERT INTO Users
VALUES
    ('Pesho', 'parolata', 'Pesho Georgiev', GETDATE(), 1),
    ('Gosho', 'parolata', 'Gosho Georgiev', GETDATE(), 1),
    ('Ivan', 'parolata', 'Ivan Georgiev', GETDATE(), 1),
    ('Mimi', 'parolata', 'Mimi Georgieva', GETDATE(), 1),
    ('Daddy', 'parolata', 'Daddy The Prof', GETDATE(), 2)
```

20. Write SQL statements to update some of the records in the Users and Groups tables.

```
UPDATE Users
SET Username = 'ThePesho', Password = 'novataparola'
FROM Users
WHERE Username = 'Pesho'

UPDATE Groups
SET Name = 'Morons'
FROM Groups
WHERE GroupID = 1
```

21. Write SQL statements to delete some of the records from the Users and Groups tables.

```
DELETE FROM Users
WHERE Username = 'ThePesho'

DELETE FROM Groups
WHERE Name = 'Morons'
```

22. Write SQL statements to insert in the Users table the names of all employees from the Employees table. Combine the first and last names as a full name. For username use the first letter of the first name + the last name (in lowercase). Use the same for the password, and NULL for last login time.

*Changed the substring of the first name to be 5 letters – otherwise unique constraint doesn't pass.

```
INSERT INTO Users(Username, Pass, Fullname, GroupID)
          SELECT SUBSTRING(FirstName,0,5) + LOWER(LastName),
                      SUBSTRING(FirstName,0,5) + LOWER(LastName) + 'pass',
                      FirstName + ' ' + LastName,
                      1
          FROM Employees
```

23. Write a SQL statement that changes the password to NULL for all users that have not been in the system since 10.03.2010.

```
UPDATE Users
SET Pass = NULL
FROM Users
WHERE   LastLogin < CONVERT(datetime, '10-03-2010')
```

24. Write a SQL statement that deletes all users without passwords (NULL password).

```
DELETE FROM Users
WHERE Pass IS NULL
```

25. Write a SQL query to display the average employee salary by department and job title.

```
SELECT JobTitle,
          DepartmentID AS [Department],
          AVG(Salary) AS [Average Salary]
FROM Employees
GROUP BY JobTitle, DepartmentID
```

26.Write a SQL query to display the minimal employee salary by department and job title along with the name of some of the employees that take it.

```
SELECT e.JobTitle,
          e.DepartmentID AS [Department],
          e.FirstName + ' ' + e.LastName as [Name],
          e.Salary
FROM Employees e
WHERE e.Salary =
(
       SELECT MIN(Salary)
       FROM Employees emp
       WHERE emp.JobTitle = e.JobTitle AND emp.DepartmentID = e.DepartmentID
)
GROUP BY e.JobTitle, e.DepartmentID, e.FirstName + ' ' + e.LastName, e.Salary
```

27.Write a SQL query to display the town where maximal number of employees work.

```
SELECT TOP(1) t.Name, COUNT(e.EmployeeID) AS WorkingEmployees
FROM Towns t
        JOIN Addresses a
        ON t.TownID = a.TownID
        JOIN Employees e
        ON e.AddressID = a.AddressID
GROUP BY t.Name
ORDER BY WorkingEmployees DESC
```

28.Write a SQL query to display the number of managers from each town.

```
SELECT COUNT(DISTINCT m.EmployeeID) AS [Managers count],
          t.Name AS [Town]
FROM Employees e
       JOIN Employees m
         ON e.ManagerID = m.EmployeeID
       JOIN Addresses a
         ON m.AddressID = a.AddressID
       JOIN Towns t
         ON a.TownID = t.TownID
GROUP BY t.Name
```

29. Write a SQL to create table WorkHours to store work reports for each employee (employee id, date, task, hours, comments). Don't forget to define identity, primary key and appropriate foreign key.

```
CREATE TABLE WorkHours
(
    WorkHoursID INT IDENTITY(1,1) PRIMARY KEY,
    EmployeeID INT FOREIGN KEY(EmployeeID) REFERENCES Employees(EmployeeID) NOT NULL,
    ReportDate smalldatetime NOT NULL,
    ReportTask nvarchar(250) NULL,
    ReportHours INT NULL,
    Comments ntext NULL,
)

CREATE TABLE WorkHoursLog
(
    WorkHoursLogID INT IDENTITY(1,1) PRIMARY KEY,
    Command varchar(6) NOT NULL,
    OldWorkHoursID INT NULL,
    OldEmployeeID INT NULL,
    OldReportDate smalldatetime NULL,
    OldReportTask nvarchar(250) NULL,
    OldReportHours INT NULL,
    OldComments ntext NULL,
    NewWorkHoursID INT NULL,
    NewEmployeeID INT NULL,
    NewReportDate smalldatetime NULL,
    NewReportTask nvarchar(250) NULL,
    NewReportHours INT NULL,
    NewComments ntext NULL,
)
```

Issue few SQL statements to insert, update and delete of some data in the table.

```
DELETE FROM WorkHours WHERE WorkHoursID = 2

INSERT INTO WorkHours
VALUES(2, '2014-08-22', 2, NULL, NULL)

UPDATE WorkHours
SET ReportHours = 8
FROM WorkHours
WHERE WorkHoursID = 1;
```

Define a table WorkHoursLogs to track all changes in the WorkHours table with triggers. For each change keep the old record data, the new record data and the command (insert / update / delete).

```
CREATE TRIGGER TR_WorkHoursDelete ON WorkHours FOR DELETE
AS
    INSERT INTO WorkHoursLog
    SELECT 'DELETE',
           d.WorkHoursID, d.EmployeeID, d.ReportDate, d.ReportTask, d.ReportHours,
d.Comments,
           NULL, NULL, NULL, NULL, NULL, NULL
    FROM deleted d
GO
```

```
CREATE TRIGGER TR_WorkHoursInsert ON WorkHours FOR INSERT
AS
    INSERT INTO WorkHoursLog
    SELECT 'INSERT',
           NULL, NULL, NULL, NULL, NULL, NULL,
           i.WorkHoursID, i.EmployeeID, i.ReportDate, i.ReportTask, i.ReportHours,
i.Comments
    FROM inserted i
GO

CREATE TRIGGER TR_WorkHoursUpdate ON WorkHours FOR UPDATE
AS
    INSERT INTO WorkHoursLog
    SELECT 'UPDATE',
           d.WorkHoursID, d.EmployeeID, d.ReportDate, d.ReportTask, d.ReportHours,
d.Comments,
           i.WorkHoursID, i.EmployeeID, i.ReportDate, i.ReportTask, i.ReportHours,
i.Comments
    FROM deleted d, inserted i
GO
```

30. Start a database transaction, delete all employees from the 'Sales' department along with all dependent records from the pother tables. At the end rollback the transaction. **DONE!**

31. Start a database transaction and drop the table EmployeesProjects. Now how you could restore back the lost table data? BY BACKUP AND RECOVERY **(WITH SCRIPT OR BAK FILE)**

32. Find how to use temporary tables in SQL Server. Using temporary tables backup all records from EmployeesProjects and restore them back after dropping and re-creating the table. **DONE!**