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 FirstName + ' ' + LastName AS Name, Salary
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 FirstName + ' ' + LastName AS Name, Salary
FROM Employees
WHERE Salary > (SELECT MIN(Salary) FROM Employees) + (Salary * 0.1)

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 e.FirstName + ' ' + e.LastName AS FullName, e.Salary, d.Name
FROM Employees e
JOIN Departments d
       ON e.DepartmentID = d.DepartmentID
WHERE Salary = (SELECT MIN(Salary) FROM Employees em
                        WHERE em.DepartmentID = e.DepartmentID)
ORDER BY d.Name


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.

SELECT AVG(e.Salary) AS 'Average Salary (Sales dep.)'
FROM Employees e
JOIN Departments d
       ON e.DepartmentID = d.DepartmentID
       AND d.Name = 'Sales'


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

SELECT COUNT(*) AS 'Employees with manager'
FROM Employees e
JOIN Employees m
       ON e.ManagerID = m.EmployeeID
```

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

```
SELECT Count(*) AS 'Employees without manager'
FROM Employees
WHERE ManagerID IS NULL
```

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

```
SELECT d.Name AS 'Department Name', AVG(e.Salary) AS 'Average Salary'
FROM Employees e
JOIN Departments d
       ON d.DepartmentID = e.DepartmentID
GROUP BY d.Name
ORDER BY AVG(e.Salary)
```

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

```
SELECT COUNT(*) AS 'Number of Employees', d.Name, t.Name
FROM Employees e
       JOIN Departments d
       ON e.DepartmentID = d.DepartmentID
              JOIN Addresses a
              ON a.AddressID = e.AddressID
                     JOIN Towns t
                     ON t.TownID = a.TownID
GROUP BY d.Name, t.Name
```

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

```
SELECT em.FirstName, em.LastName, em.EmployeeID
FROM Employees e
JOIN Employees em
       ON e.ManagerID = em.EmployeeID
       GROUP BY em.FirstName, em.LastName, em.EmployeeID
       HAVING COUNT(e.EmployeeID) = 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((y.FirstName + ' ' +
y.LastName), '(no manager)') AS Manager
FROM Employees e
LEFT OUTER JOIN Employees y
       ON e.ManagerID = y.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
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 CONVERT(varchar(30), GETDATE(), 113)
```

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 IDENTITY PRIMARY KEY,
Username nvarchar(30) NOT NULL,
Password nvarchar(30) NOT NULL CHECK(LEN(Password) > 5),
FullName nvarchar(50) NOT NULL,
LastLogin varchar(30) NOT NULL
)
GO
```

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 [Todays Users] AS
SELECT Username
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 IDENTITY,
Name nvarchar(50) UNIQUE NOT NULL,
CONSTRAINT PK_Groups PRIMARY KEY(GroupID)
)
```

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.

```
ALTER TABLE Users
       ADD GroupID int NOT NULL
GO

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

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

```
INSERT INTO Groups VALUES
('Students'),
('Teachers'),
('Directors'),
('Presidents'),
('Clients')

INSERT INTO Users VALUES
('Vanko23', '123454', 'Ivan Goshev', GETDATE(), 1),
```

```
('Pencho34', '353445', 'Pencho Neshev', GETDATE(), 2),
('Pitar43', '532445', 'Pitar Toshev', GETDATE(), 3),
('Kolio234', '984545', 'Nikolay Iliev', GETDATE(), 4)
```

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

```
UPDATE Users
SET Username = 'VankoVanko'
WHERE UserID = 13

UPDATE Groups
SET Name = 'ExWorkers'
WHERE GroupID = 2
```

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

```
DELETE FROM Users
WHERE UserID = 8

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

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.

```
INSERT INTO Users (Username, [Password], Fullname)
SELECT LOWER(LEFT(FirstName, 3) + LastName),
             LOWER(LEFT(FirstName, 3) + LastName),
             (FirstName + ' ' + LastName)
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 [Password] = NULL
WHERE LastLogin < CONVERT(DATETIME, 2015-10-10)
```

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

```
USE TelerikAcademy
SELECT d.Name AS [Department Name], e.JobTitle, MIN(e.Salary) AS [Average Salary]
FROM Employees e
JOIN Departments d
       ON d.DepartmentID = e.DepartmentID
GROUP BY d.Name, e.JobTitle
```

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 MIN(e.FirstName) AS [Random Employee], d.Name AS [Department Name], e.JobTitle,
MIN(e.Salary) AS [Average Salary]
FROM Employees e
```

```
        JOIN Departments d
        ON d.DepartmentID = e.DepartmentID
GROUP BY d.Name, e.JobTitle

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

SELECT TOP 1 t.Name AS [Town], COUNT(e.EmployeeID) AS [Number of Employees]
FROM Employees e
        JOIN Addresses a
        ON a.AddressID = e.AddressID
                JOIN Towns t
                ON t.TownID = a.TownID
GROUP BY t.Name
ORDER BY COUNT(e.EmployeeID) DESC

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

SELECT t.Name, COUNT(m.EmployeeID)
FROM Employees m
        JOIN Addresses a
    ON m.AddressID = a.AddressID
        JOIN Towns t
    ON a.TownID = t.TownID
 WHERE m.EmployeeID IN(SELECT ManagerID FROM Employees)
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.
Issue few SQL statements to insert, update and delete of some data in the table.
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).

USE TelerikAcademy
CREATE TABLE WorkHours (
        EmployeeId INT IDENTITY,
        [Date] DATETIME,
        Task NVARCHAR(100),
        [Hours] INT,
        Comments NVARCHAR(300)
        CONSTRAINT PK_WorkHours PRIMARY KEY(EmployeeId)
        CONSTRAINT FK_WorkHours_Employees FOREIGN KEY(EmployeeId)
        REFERENCES Employees(EmployeeId)
)
GO

INSERT INTO WorkHours
VALUES (GETDATE(), 'Write homework', 5, 'Homework about advanced SQL'),
                (GETDATE(), 'Go go Lecture', 4, 'Attend to lecture in Telerik Academy'),
                (GETDATE(), 'Rest', 2, 'Rest after hard day')

UPDATE WorkHours
SET Date = '2015-10-05 18:00'
WHERE Task LIKE '%Lecture%'

DELETE FROM WorkHours
WHERE [Hours] < 3

CREATE TABLE WorkHoursLogs(
        LogId INT IDENTITY,
```

```
        OldRecord nvarchar(500),
        NewRecord nvarchar(500),
        Command nvarchar(10),
        EmployeeId INT,
        CONSTRAINT PK_WorkHoursLogs PRIMARY KEY(LogId),
        CONSTRAINT FK_WorkHoursLogs_WorkHours FOREIGN KEY(EmployeeId)
        REFERENCES WorkHours(EmployeeId)
)
GO

CREATE TRIGGER tr_WorkHoursInsert ON WorkHours FOR INSERT
AS
        INSERT INTO WorkHoursLogs(OldRecord, NewRecord, Command, EmployeeId)
        VALUES('',
                (SELECT 'Day: ' + CAST(Date AS nvarchar(50)) + ' ' + ' Task: ' + Task
+ ' ' +
                                    ' Hours: ' + CAST([Hours] AS nvarchar(50)) + ' ' +
Comments
                    FROM Inserted),
                'INSERT',
                (SELECT EmployeeID FROM Inserted))
GO

CREATE TRIGGER tr_WorkHoursUpdate ON WorkHours FOR UPDATE
AS
        INSERT INTO WorkHoursLogs(OldRecord, NewRecord, Command, EmployeeId)
        VALUES((SELECT 'Day: ' + CAST(Date AS nvarchar(50)) + ' ' + ' Task: ' + Task + '
' +
                                     ' Hours: ' + CAST([Hours] AS nvarchar(50)) + ' ' +
Comments FROM Deleted),
                (SELECT 'Day: ' + CAST(Date AS nvarchar(50)) + ' ' + ' Task: ' + Task
+ ' ' +
                                    ' Hours: ' + CAST([Hours] AS nvarchar(50)) + ' ' +
Comments FROM Inserted),
                'UPDATE',
                (SELECT EmployeeID FROM Inserted))
GO

CREATE TRIGGER tr_WorkHoursDelete ON WorkHours FOR DELETE
AS
        INSERT INTO WorkHoursLogs(OldRecord, NewRecord, Command, EmployeeId)
        VALUES((SELECT 'Day: ' + CAST(Date AS nvarchar(50)) + ' ' + ' Task: ' + Task + '
' +
                                    ' Hours: ' + CAST([Hours] AS nvarchar(50)) + ' ' +
Comments FROM Deleted),
                '',
                'DELETE',
                (SELECT EmployeeID FROM Deleted))
GO

INSERT INTO WorkHours
VALUES(GETDATE(), 'Sleep', 8, 'Sleep when its dark outside')

DELETE FROM WorkHours
WHERE Task = 'Rest'

UPDATE WorkHours
SET Task = 'Win Money'
WHERE EmployeeID = 1
```

```
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.
BEGIN TRAN

    ALTER TABLE Departments
        DROP CONSTRAINT FK_Departments_Employees
    GO

    DELETE *
        FROM Employees e
        JOIN Departments d
            ON e.DepartmentID = d.DepartmentID
        WHERE d.Name = 'Sales'

ROLLBACK TRAN

31. Start a database transaction and drop the table EmployeesProjects.
Now how you could restore back the lost table data?
BEGIN TRANSACTION

    DROP TABLE EmployeesProjects

ROLLBACK TRANSACTION

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.
 CREATE TABLE #TemporaryTable (
       EmployeeId INT,
       ProjectId INT
)

INSERT INTO #TemporaryTable
SELECT EmployeeId, ProjectId
FROM EmployeesProjects

DROP TABLE EmployeesProjects

CREATE TABLE EmployeesProjects (
       EmployeeId INT,
       ProjectId INT,
       CONSTRAINT PK_EmployeesProjects PRIMARY KEY(EmployeeID, ProjectID),
       CONSTRAINT FK_EmployeesProjects_Employees FOREIGN KEY(EmployeeID)
       REFERENCES Employees(EmployeeID),
       CONSTRAINT FK_EmployeesProjects_Projects FOREIGN KEY(ProjectID)
       REFERENCES Projects(ProjectID)
)

INSERT INTO EmployeesProjects
SELECT EmployeeId, ProjectId
FROM #TemporaryTable
```