--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 e.FirstName, e.LastName, e.Salary

FROM Employees AS e

WHERE e.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 e.FirstName, e.LastName, e.Salary

FROM Employees AS e

WHERE e.Salary <= 1.1 \* (

        SELECT 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 e.FirstName + ' ' + e.LastName AS [FULL Name], e.Salary, d.Name AS [Department Name]

FROM Employees AS e JOIN Departments d

  ON e.DepartmentID = d.DepartmentID

WHERE e.Salary = (

    SELECT MIN(Salary)

    FROM Employees

        WHERE DepartmentID = d.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.

SELECT AVG(Salary)

FROM Employees

WHERE DepartmentID = (

        SELECT DepartmentID

        FROM Departments

        WHERE Name = 'Sales')

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

SELECT COUNT(\*)

FROM Employees

WHERE DepartmentID = (

        SELECT DepartmentID

        FROM Departments

        WHERE Name = 'Sales')

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

SELECT COUNT(ManagerID)

FROM Employees

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

SELECT COUNT(\*)

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)

FROM Departments AS d JOIN Employees AS e

  ON d.DepartmentID = e.DepartmentID

GROUP BY d.Name

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

SELECT COUNT(\*) AS [Emp COUNT], d.Name AS [Dep], t.Name AS [Town]

FROM Employees AS e JOIN Departments AS d

  ON e.DepartmentID = d.DepartmentID

  JOIN Addresses AS a ON e.AddressID = a.AddressID

  JOIN Towns AS t ON a.TownID = t.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 FirstName, LastName

FROM Employees AS e

WHERE 5 = (

        SELECT COUNT(\*)

        FROM Employees

        WHERE ManagerID = e.EmployeeID)

--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 + ' ' + m.LastName, '(no manager)' ) AS [Manager]

FROM Employees AS e LEFT JOIN Employees AS 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

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 IDENTITY,

                CONSTRAINT PK\_Users PRIMARY KEY(UserId),

        Username VARCHAR(20) NOT NULL,

                CONSTRAINT UK\_Users\_Username UNIQUE(Username),

                CHECK(LEN(Username) >= 3),

        Password VARCHAR(20) NOT NULL,

                CHECK(LEN(Password) >= 5),

        FullName VARCHAR(40) NOT NULL,

        LastLogin DateTime,

)

--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 AllUsersFromToday AS

SELECT \*

FROM Users

INSERT INTO Users

VALUES ('George', 'Georgiev', 'George Georgiev', '02.02.2002');

SELECT \*

FROM AllUsersFromToday

--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,

                CONSTRAINT PK\_Groups PRIMARY KEY(GroupId),

        Name VARCHAR(50) NOT NULL

                CONSTRAINT UK\_Groups\_Name UNIQUE(Name),

)

--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

INSERT INTO Groups(Name)

VALUES

        ('Telerik Academy'),

        ('Trainee'),

        ('Trainers'),

        ('Developers'),

        ('Support'),

        ('QA')

UPDATE Users

SET GroupId = 3

WHERE GroupId IS NULL

INSERT INTO Users (Username, Password, FullName, LastLogin, GroupId)

VALUES

        ('Ivan','Ivanov','Ivan Ivanov', '2005-5-5 12:00:15', 1),

        ('Petar','Petrov','Petar Petrov', '2006-6-6 12:00:15', 1),

        ('Todor','Todorov','Todor Todorov', '2005-3-3 11:00:00', 6),

        ('Kristiqn','qwerty','Kristiqn Kristiqnov', '2011-1-1', 5),

        ('Ivailo','123456','Ivailo Ivailov', '2012-9-9', 2),

        ('Teodora','parol@','Teodora Teodorova', '2013-11-6', 4),

        ('Valentina','ToP!s3cr37','Valentina Valentinova', '2005-5-5', 5),

        ('Viktoriq','B1234C','Some Name', '2012-11-7', 4)

ALTER TABLE Users

ADD CONSTRAINT FK\_Users\_Groups

FOREIGN KEY(GroupId)

REFERENCES Groups(GroupId)

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

INSERT INTO Users(Username, Password, FullName, LastLogin, GroupID)

VALUES

        ('Marian', '@kul@7@', 'Marian Marinov', GETDATE(), 1),

        ('Lyubo', 'kapi7@n@', 'Lyubo Lyubomirov', GETDATE(), 1)

INSERT INTO Groups(Name)

VALUES

        ('Other'),

        ('Fake Group')

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

UPDATE Users

SET GroupID =

        (SELECT GroupId

        FROM Groups

        WHERE name = 'Other')

WHERE username = 'Marian' OR Username = 'Lyubo'

UPDATE Groups

SET Name = 'Really Fake Group'

WHERE Name = 'Fake Group'

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

DELETE FROM Users

WHERE Username = 'Lyubo'

DELETE FROM Groups

WHERE Name LIKE ('%fake%')

--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, LastLogin, GroupId)

SELECT

        -- Check length Constraint with Case-When-Then statements

        -- and resolve unique nicknames problem by adding row number

        CASE WHEN LEN(SUBSTRING(FirstName,1,1) + LOWER(LastName)) >= 5

                THEN SUBSTRING(FirstName,1,1) + LOWER(LastName) + CONVERT(nvarchar(3), ROW\_NUMBER() OVER(ORDER BY FirstName))

                ELSE SUBSTRING(FirstName,1,1) + LOWER(LastName) + '12345' + CONVERT(nvarchar(3), ROW\_NUMBER() OVER(ORDER BY FirstName))

        END AS [Username],

        CASE  WHEN LEN(SUBSTRING(FirstName,1,1) + LOWER(LastName)) >= 5

                THEN SUBSTRING(FirstName,1,1) + LOWER(LastName)

                ELSE SUBSTRING(FirstName,1,1) + LOWER(LastName) + '12345'

        END AS [Password],

        FirstName + ' ' + LastName AS [FullName],

        NULL AS [LastLogin],

        8 AS GroupId

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 < '10.03.2010'

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

DELETE FROM Users

WHERE Password IS NULL

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

SELECT e.JobTitle, d.Name AS [Department], AVG(e.Salary) AS [Average Salary]

FROM Employees AS e JOIN Departments AS d

  ON e.DepartmentID = d.DepartmentID

GROUP BY e.JobTitle, d.Name

--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.Salary) AS [MIN Salary],

        MIN(FirstName + ' ' + LastName) AS [Employee],

        e.JobTitle,

        d.Name AS [Department]

FROM Employees AS e JOIN Departments AS d

  ON e.DepartmentID = d.DepartmentID

GROUP BY d.Name, e.JobTitle

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

SELECT Name, [Manager COUNT]

FROM

        (SELECT COUNT(\*) AS [Manager COUNT], t.Name

        FROM Employees AS e JOIN Addresses AS a

                ON e.AddressID = a.AddressID

                JOIN Towns AS t ON a.TownID = t.TownID

        GROUP BY t.Name) AS [ManagersPerTown]

WHERE [Manager COUNT] = (

        SELECT MAX([Manager COUNT])

        FROM

                (SELECT COUNT(\*) AS [Manager COUNT], t.Name

                FROM Employees AS e JOIN Addresses AS a

                        ON e.AddressID = a.AddressID

                        JOIN Towns AS t ON a.TownID = t.TownID

                GROUP BY t.Name) AS [ManagersPerTown])

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

SELECT COUNT(\*) AS [Managers COUNT], t.Name

FROM Employees AS e JOIN Addresses AS a

  ON e.AddressID = a.AddressID

  JOIN Towns AS t ON a.TownID = t.TownID

GROUP BY t.Name

ORDER BY COUNT(\*)

--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).

CREATE TABLE WorkHours

(

        WorkHourId BIGINT IDENTITY,

                CONSTRAINT PK\_WorkHours PRIMARY KEY (WorkHourId),

        WorkDate DATETIME NOT NULL,

        Task VARCHAR(100) NOT NULL,

        Comments TEXT,

        WorkHours INT,

        EmployeeId INT NOT NULL,

                CONSTRAINT FK\_WorkHours\_Employees FOREIGN KEY(EmployeeId)

                REFERENCES Employees(EmployeeId),

)

INSERT INTO WorkHours (WorkDate, WorkHours,Task, Comments,EmployeeId)

VALUES

        (GETDATE(), 2, 'Databases Lecture', NULL,5),

        (GETDATE(), 2, 'Support Basics Lecture', NULL,4),

        (GETDATE(), 1, 'Databases Homework', NULL,6),

        (GETDATE(), 1, 'Support Basics Homework', 'Bad Homework',10),

        (GETDATE(), 3, 'Linux OS Lecture', 'Nice lecture',1)

UPDATE WorkHours

SET WorkHours.WorkHours = 3

WHERE Task LIKE ('%Databases%')

DELETE FROM WorkHours

WHERE Task LIKE('%LINUX%')

CREATE TABLE WorkHoursLogs

(

        WorkHoursLogId BIGINT IDENTITY

                CONSTRAINT PK\_WorkHoursLogs PRIMARY KEY(WorkHoursLogId),

        CommandDateTime DATETIME NOT NULL,

        CommandType nvarchar(6) NOT NULL,

        WorkHourId BIGINT NOT NULL,

        CONSTRAINT FK\_WorkHoursLogs\_WorkHours FOREIGN KEY(WorkHourId)

                REFERENCES WorkHours(WorkHourId),

        OldWorkHourId BIGINT,

        OldWorkDate DATETIME,

        OldTask VARCHAR(100),

        OldComments TEXT,

        OldWorkHours INT,

        OldEmployeeId INT NOT NULL,

        NewWorkHourId BIGINT,

        NewWorkDate DATETIME,

        NewTask VARCHAR(100),

        NewComments TEXT,

        NewWorkHours INT,

        NewEmployeeId INT NOT NULL,

)

--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

DELETE FROM Employees

WHERE Employees.DepartmentID = (

        SELECT DepartmentID

        FROM Departments

        WHERE Departments.Name = 'Sales')

ROLLBACK

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

BEGIN TRAN

DROP TABLE EmployeesProjects

ROLLBACK

--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 #LocalTempTable(

        EmployeeID INT NOT NULL,

        ProjectID INT NOT NULL,

        CONSTRAINT PK\_EmployeeesProjects PRIMARY KEY (EmployeeID, ProjectID),

)

INSERT INTO #LocalTempTable

SELECT \* FROM EmployeesProjects

DROP TABLE EmployeesProjects

CREATE TABLE EmployeesProjects(

        EmployeeID INT NOT NULL,

        ProjectID INT NOT NULL,

        CONSTRAINT PK\_EmployeeesProjects 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 \* FROM #LocalTempTable