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 [TelerikAcademy].dbo.Employees

WHERE Salary = (SELECT MIN(Salary)

FROM [TelerikAcademy].dbo.Employees);

1. **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 [TelerikAcademy].dbo.Employees

WHERE Salary <= (SELECT 1.1 \* MIN(Salary)

FROM [TelerikAcademy].dbo.Employees);

1. **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 Name, e.Salary,

d.Name

FROM [TelerikAcademy].dbo.Employees e

JOIN [TelerikAcademy].dbo.Departments d

ON d.DepartmentID = e.DepartmentID

WHERE Salary = (SELECT MIN(Salary)

FROM [TelerikAcademy].dbo.Employees

WHERE DepartmentID = e.DepartmentID);

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

SELECT AVG(Salary)

FROM Employees

WHERE DepartmentID = 1;

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

SELECT AVG(Salary) AS [Average Salary in Sales]

FROM Employees e

JOIN Departments d

ON d.DepartmentID = e.DepartmentID

WHERE d.Name = 'Sales';

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

SELECT COUNT(employeeID)

FROM Employees e

JOIN Departments d

ON d.DepartmentID = e.DepartmentID

WHERE d.Name = 'Sales';

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

SELECT COUNT(employeeID) AS [Employees with a Manager]

FROM Employees e

WHERE EXISTS (SELECT EmployeeID

FROM Employees m

WHERE m.EmployeeID = e.ManagerID);

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

SELECT COUNT(employeeID) AS [Employees without a Manager]

FROM Employees e

WHERE e.ManagerID is null;

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

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

FROM Employees e

JOIN Departments d

ON d.DepartmentID = e.DepartmentID

GROUP BY d.Name;

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

SELECT t.Name AS Town, d.Name AS Department,

COUNT(EmployeeID) AS Employees

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

ORDER BY t.Name;

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

SELECT m.FirstName, min(m.LastName),

COUNT(\*) AS EmployeeCount

FROM Employees e

JOIN Employees m

ON e.ManagerID = m.EmployeeID

GROUP BY m.FirstName

HAVING COUNT(\*) = 5

1. **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,

COALESCE(CONVERT(nvarchar(50), ManagerID), 'no manager')

FROM Employees e

order by ManagerID

1. **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;

1. **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(24),GETDATE(),4) + ' '

+ CONVERT(VARCHAR(24),GETDATE(),14);

1. **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,

Username nvarchar(50) NOT NULL,

UserPassword nvarchar(50) NOT NULL,

Fullname nvarchar(50) NOT NULL,

LastLoginTime date NOT NULL,

CONSTRAINT PK\_Users PRIMARY KEY(UserID),

CONSTRAINT Username UNIQUE(Username),

CONSTRAINT [MinLengthConstraint]

CHECK (DATALENGTH([UserPassword]) >= 5)

)

GO

1. **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, GETDATE(), LastLoginTime) < 1

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

CREATE TABLE Groups (

GroupID int IDENTITY,

Name nvarchar(50) NOT NULL,

CONSTRAINT PK\_Groups PRIMARY KEY(GroupID),

CONSTRAINT UniqueName UNIQUE(Name)

)

GO

1. **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 Users and Groups tables.**

ALTER TABLE Users

ADD GroupID int,

CONSTRAINT FK\_Users\_Groups

FOREIGN KEY (GroupID)

REFERENCES Groups(GroupID)

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

BEGIN TRAN

INSERT INTO Groups(Name)

Values('FirstGroup');

INSERT INTO Groups(Name)

Values('SecondGroup');

INSERT INTO Groups(Name)

Values('ThirdGroup')

INSERT INTO Users(Username, UserPassword, Fullname,

GroupID)

VALUES ('Gosho', 'gosho', 'Georgi Ivanov', 1)

INSERT INTO Users(Username, UserPassword, Fullname,

GroupID)

VALUES ('Penka', 'penka', 'Penka Yordanova', 2)

INSERT INTO Users(Username, UserPassword, Fullname,

GroupID)

VALUES ('Ivan', 'ivan34', 'Ivan Petrov', 3)

COMMIT TRAN

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

UPDATE Users

SET Username = 'Pesho23'

WHERE Username = 'Pesho'

UPDATE Groups

SET Name = 'AnotherGroup'

WHERE Name = 'SecondGroup'

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

DELETE FROM Users

WHERE Username = 'Pesho23'

UPDATE Users

SET GroupID = 1

WHERE GroupID = 2

DELETE FROM Groups

WHERE GroupID = 2

1. **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.**

BEGIN TRANSACTION

INSERT INTO Users

(Username, UserPassword, Fullname, GroupID)

SELECT LOWER(LEFT(FirstName,1) + LastName),

LOWER(LEFT(FirstName,1) + LastName),

FirstName + ' ' + LastName,

1

FROM Employees

Where FirstName NOT LIKE 'Annette'

COMMIT TRANSACTION

Because the username does not allow duplication and Annette’s username is already taken.

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

BEGIN TRAN

UPDATE Users

SET UserPassword = null

WHERE LastLoginTime < '10.03.2010'

COMMIT TRAN

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

DELETE FROM Users

WHERE UserPassword is null

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

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

FROM Employees e

JOIN Departments d

ON e.DepartmentID = d.DepartmentID

GROUP BY d.Name, JobTitle

1. **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 d.Name, e.JobTitle, MIN(Salary) AS [Average Salary],

MIN(e.FirstName + ' ' + e.LastName) AS Employee

FROM Employees e

JOIN Departments d

ON e.DepartmentID = d.DepartmentID

GROUP BY d.Name, JobTitle

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

SELECT TOP 1 t.Name AS City,

COUNT(e.EmployeeID) AS NumberOfEmployees

FROM Employees e

JOIN Addresses a

ON e.AddressID = a.AddressID

JOIN Towns t

ON t.TownID = a.TownID

GROUP BY t.Name

ORDER BY NumberOfEmployees DESC

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

SELECT t.Name AS City,

COUNT(m.ManagerID) AS NumberOfManagers,

COUNT(m.EmployeeID) AS NumberOfEmployees

FROM Employees e

JOIN Addresses a

ON e.AddressID = a.AddressID

JOIN Towns t

ON t.TownID = a.TownID

JOIN Employees m

ON e.ManagerID = m.EmployeeID

GROUP BY t.Name

ORDER BY NumberOfEmployees DESC

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

BEGIN TRAN

CREATE TABLE WorkHours (

WorkHoursID int IDENTITY,

EmployeeID int NOT NULL,

WorkingDate date NOT NULL,

Task nvarchar(50) NOT NULL,

WorkedHours int NOT NULL,

Comments nvarchar(50) NULL,

CONSTRAINT PK\_WorkHours PRIMARY KEY(WorkHoursID),

CONSTRAINT FK\_WorkHours\_Employees

FOREIGN KEY(EmployeeID)

REFERENCES Employees(EmployeeID)

)

GO

COMMIT TRAN

BEGIN TRAN

INSERT INTO WorkHours

(EmployeeID, WorkingDate, Task, WorkedHours, Comments)

VALUES(1, Convert(varchar(30),'7/8/2014',102), 'do sth', 6, null)

INSERT INTO WorkHours

(EmployeeID, WorkingDate, Task, WorkedHours, Comments)

VALUES(2, Convert(varchar(30),'7/7/2014',102), 'do sth else', 8, 'good worker')

COMMIT TRAN

BEGIN TRAN

UPDATE WorkHours

SET Comments = 'to do comment'

WHERE Comments is NULL

COMMIT TRAN

DELETE FROM WorkHours

WHERE WorkHoursID = 3

BEGIN TRAN

CREATE TABLE WorkHoursLogs (

WorkHoursLogID int IDENTITY,

OldRecord nvarchar(50) NOT NULL,

NewRecord nvarchar(50) NOT NULL,

Command nvarchar(50) NOT NULL,

WorkHoursID int NOT NULL,

CONSTRAINT PK\_WorkHoursLogs PRIMARY KEY(WorkHoursLogID),

CONSTRAINT FK\_WorkHoursLogs\_WorkHours

FOREIGN KEY(WorkHoursID)

REFERENCES WorkHours(WorkHoursID)

)

GO

COMMIT TRAN

CREATE TRIGGER tr\_WorkHoursInsert

ON WorkHours FOR INSERT

AS

INSERT INTO WorkHoursLogs

(OldRecord, NewRecord, Command, WorkHoursID)

VALUES(' ', (SELECT 'Day: ' + CAST(WorkingDate AS nvarchar(50))

+ ' ' + 'Task: ' + Task + ' ' + 'Hours: ' +

CAST(WorkedHours AS nvarchar(50)) + ' ' + Comments FROM inserted),

'INSERT', (SELECT WorkHoursID FROM inserted))

GO

CREATE TRIGGER tr\_WorkHoursUpdate ON WorkHours FOR UPDATE

AS

INSERT INTO WorkHoursLogs(OldRecord, NewRecord, Command, WorkHoursId)

VALUES((SELECT 'Day: ' + CAST(WorkingDate AS nvarchar(50)) +

' ' + 'Task: ' + Task + ' ' + ' Hours: ' +

CAST(WorkedHours AS nvarchar(50)) + ' ' + Comments FROM Deleted),

(SELECT 'Day: ' + CAST(WorkingDate AS nvarchar(50)) + ' ' + ' Task: ' +

Task + ' ' + 'Hours: ' + CAST(WorkedHours AS nvarchar(50)) + ' ' +

Comments FROM Inserted),

'UPDATE',

(SELECT WorkHoursId FROM Inserted))

GO

CREATE TRIGGER tr\_WorkHoursDeleted ON WorkHours FOR DELETE

AS

INSERT INTO WorkHoursLogs(OldRecord, NewRecord, Command, WorkHoursId)

VALUES((SELECT 'Day: ' + CAST(WorkingDate AS nvarchar(50)) +

' ' + 'Task: ' + Task + ' ' + ' Hours: ' +

CAST(WorkedHours AS nvarchar(50)) + ' ' + Comments FROM Deleted),

' ',

'DELETE',

(SELECT WorkHoursId FROM Deleted))

GO

1. **Start a database transaction, delete all employees from the 'Sales' department along with all dependent records from the other tables. At the end rollback the transaction.**

BEGIN TRAN

ALTER TABLE Departments DROP

CONSTRAINT FK\_Departments\_Employees

DELETE FROM Employees

FROM Employees e

JOIN Departments d

ON e.DepartmentID = d.DepartmentID

WHERE d.Name = 'Sales'

ROLLBACK TRAN

1. **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 TRAN

1. **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.**

BEGIN TRAN

CREATE TABLE TemporaryEmployeesProjects(

EmployeeID int NOT NULL,

ProjectID int NOT NULL

)

INSERT INTO TemporaryEmployeesProjects

SELECT EmployeeID, ProjectID

FROM EmployeesProjects

DROP TABLE EmployeesProjects

CREATE TABLE EmployeesProjects(

EmployeeID int NOT NULL,

ProjectID int NOT NULL,

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

CONSTRAINT FK\_EP\_Employee FOREIGN KEY(EmployeeID)

REFERENCES Employees(EmployeeID),

CONSTRAINT FK\_EP\_Project FOREIGN KEY(ProjectID)

REFERENCES Projects(ProjectID)

)

INSERT INTO EmployeesProjects

SELECT EmployeeID, ProjectID

FROM TemporaryEmployeesProjects

DROP TABLE TemporaryEmployeesProjects

COMMIT TRAN