# Advanced SQL – Homework

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.**

USE TelerikAcademy;

SELECT e.FirstName, e.LastName, e.Salary

FROM Employees AS e

WHERE e.Salary =

(SELECT MIN(Salary)

FROM Employees);

GO

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.**

USE TelerikAcademy;

SELECT e.FirstName, e.LastName, e.Salary

FROM Employees AS e

WHERE e.Salary BETWEEN

(SELECT MIN(Salary)

FROM Employees)

AND

(SELECT MIN(Salary) + (MIN(Salary) \* 10 / 100)

FROM Employees);

GO

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.**

USE TelerikAcademy;

SELECT e.FirstName + e.LastName AS [Full Name], e.Salary, d.Name

FROM Employees AS e, Departments AS d

WHERE e.DepartmentID = d.DepartmentID AND e.Salary =

(SELECT MIN(Salary)

FROM Employees

WHERE DepartmentID = e.DepartmentID);

GO

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

USE TelerikAcademy;

SELECT AVG(e.Salary)

FROM Employees AS e

WHERE e.DepartmentID = 1;

GO

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

USE TelerikAcademy;

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

FROM Employees AS e

INNER JOIN Departments AS d

ON e.DepartmentID = d.DepartmentID

WHERE d.Name = 'Sales'

GROUP BY d.Name;

GO

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

USE TelerikAcademy;

SELECT Count(\*) AS [Number Of Employees], d.Name AS [Department Name]

FROM Employees AS e

INNER JOIN Departments AS d

ON e.DepartmentID = d.DepartmentID

WHERE d.Name = 'Sales'

GROUP BY d.Name;

GO

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

USE TelerikAcademy;

SELECT COUNT(\*) AS [Number Of Employees With Manager]

FROM Employees AS e

WHERE e.ManagerID IS NOT NULL;

GO

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

USE TelerikAcademy;

SELECT COUNT(\*) AS [Number Of Employees Without Manager]

FROM Employees AS e

WHERE e.ManagerID IS NULL;

GO

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

USE TelerikAcademy;

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

FROM Employees AS e

INNER JOIN Departments AS d

ON d.DepartmentID = e.DepartmentID

GROUP BY d.Name;

GO

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

USE TelerikAcademy;

(SELECT 'Department:' + d.Name, COUNT(\*) AS [Number Of Employees]

FROM Departments AS d

INNER JOIN Employees AS e

ON d.DepartmentID = e.DepartmentID

GROUP BY d.Name)

UNION

(SELECT 'Town:' + t.Name, COUNT(\*) AS [Number Of Employees]

FROM Towns AS t

INNER JOIN Addresses AS a

ON t.TownID = a.TownID

INNER JOIN Employees AS e

ON a.AddressID = e.AddressID

GROUP BY t.Name);

GO

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

USE TelerikAcademy;

SELECT m.FirstName, m.LastName, COUNT(\*) AS [Number of Employees]

FROM Employees AS m

INNER JOIN Employees AS e

ON e.ManagerID = m.EmployeeID

GROUP BY m.FirstName, m.LastName

HAVING COUNT(\*) = 5;

GO

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)".**

USE TelerikAcademy;

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;

GO

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.**

USE TelerikAcademy;

SELECT e.FirstName + ' ' + e.LastName AS [Employee]

FROM Employees AS e

WHERE LEN(e.LastName) = 5;

GO

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.**

USE TelerikAcademy;

SELECT CONVERT(VARCHAR, GETDATE(), 104) + ' ' +

CONVERT(VARCHAR, GETDATE(), 114) AS [Current Date];

GO

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.**

USE TelerikAcademy;

-- Delete the table from the database

IF OBJECT\_ID('Users') IS NOT NULL

BEGIN

DROP TABLE Users

END

GO

-- Create the table again

CREATE TABLE Users(

UserID int IDENTITY,

UserName nvarchar(40) NOT NULL,

UserPassword nvarchar(40),

FullName nvarchar(100) NOT NULL,

LastLogin datetime,

CONSTRAINT PK\_Users PRIMARY KEY(UserID),

CONSTRAINT IXU\_Username UNIQUE (UserName),

CONSTRAINT CK\_UserPassword CHECK (DATALENGTH(UserPassword) > 5)

);

GO

-- Insert data to the table Users to test the next task

INSERT INTO Users(UserName, UserPassword, FullName, LastLogin)

VALUES ('pesho', 'peshev', 'pesho peshev', GETDATE()),

('gosho', 'goshev', 'gosho goshev', '01.01.2013'),

('vankata', 'vankov', 'vankata vankov', GETDATE());

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.**

USE TelerikAcademy;

GO

-- Delete the view from the database

IF OBJECT\_ID('Login Users Today') IS NOT NULL

BEGIN

DROP VIEW [Login Users Today];

END

GO

-- Creat the view

CREATE VIEW [Login Users Today] AS

SELECT \* FROM Users

WHERE DATEADD(dd, 0, DATEDIFF(dd, 0, LastLogin)) = DATEADD(dd, 0, DATEDIFF(dd, 0, GETDATE()));

GO

-- Test the view

SELECT \* FROM [Login Users Today];

GO

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

USE TelerikAcademy;

-- Delete the table from the database

IF OBJECT\_ID('Groups') IS NOT NULL

BEGIN

DROP TABLE Groups

END

GO

-- Create the table again

CREATE TABLE Groups(

GroupID int IDENTITY,

Name nvarchar(250) NOT NULL,

CONSTRAINT PK\_Groups PRIMARY KEY(GroupID),

CONSTRAINT IXU\_Name UNIQUE (Name)

);

GO

-- Insert data to the table to test future tasks

INSERT INTO Groups (Name)

VALUES ('All about the football'),

('Think GREEN'),

('Live NOW'),

('Linkin Park');

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 tables Users and Groups tables.**

USE TelerikAcademy;

-- Add column GroupID

ALTER TABLE Users ADD GroupID int

GO

-- Update records

UPDATE Users

SET GroupID = 1

WHERE UserId = 1;

UPDATE Users

SET GroupID = 2

WHERE UserId = 2;

UPDATE Users

SET GroupID = 3

WHERE UserId = 3;

GO

-- Add the foreign key

ALTER TABLE Users

ADD CONSTRAINT FK\_Users\_Groups

FOREIGN KEY (GroupID)

REFERENCES Groups(GroupID);

GO

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

USE TelerikAcademy;

-- Insert data to the Groups

INSERT INTO Groups (Name)

VALUES ('The name of the game'),

('The Simpsons'),

('C#'),

('Databases - MySQL & MSSQL');

GO

-- Insert data to the Users

INSERT INTO Users(UserName, UserPassword, FullName, LastLogin, GroupID)

VALUES ('kircho', 'krichev', 'kricho kirchev', GETDATE(), 5),

('kolio', 'kolchev', 'kolio kolchev', '02.12.2014', 6),

('mincho', 'praznikov', 'mincho praznikov', GETDATE(), 7),

('valkata', 'valentinov', 'valkata valentinov', GETDATE(), 8);

GO

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

USE TelerikAcademy;

-- Update Users

UPDATE Users

SET GroupID = 7

WHERE GroupID = 8;

UPDATE Users

SET GroupID = 2

WHERE GroupID = 1;

GO

-- Update Groups

UPDATE Groups

SET Name = Name + ' OOP & HQC'

WHERE GroupID = 7;

UPDATE Groups

SET Name = Name + ' - FOOTBALL'

WHERE GroupID = 5;

GO

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

USE TelerikAcademy;

-- Update Users

DELETE FROM Users

WHERE GroupID = 5;

GO

-- Update Groups

DELETE FROM Groups

WHERE Name Like 'The name of the game%';

GO

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.**

USE TelerikAcademy;

-- I changed Username and passwor preventing from duplicates

INSERT INTO Users (UserName, UserPassword, FullName)

SELECT

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

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

e.FirstName + ' ' + e.LastName

FROM Employees AS e;

GO

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.**

USE TelerikAcademy;

UPDATE Users

SET UserPassword = NULL

WHERE CAST(LastLogin AS date) <= CONVERT(datetime, '10.03.2010', 104)

OR LastLogin IS NULL;

GO

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

USE TelerikAcademy;

DELETE FROM Users

WHERE UserPassword IS NULL;

GO

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

USE TelerikAcademy;

(SELECT 'Department: ' + d.Name, AVG(e.Salary) AS [Average Salary]

FROM Employees AS e

JOIN Departments AS d

ON e.DepartmentID = d.DepartmentID

GROUP BY d.Name

)

UNION

(SELECT 'Job: ' + e.JobTitle, AVG(e.Salary) AS [Average Salary]

FROM Employees AS e

GROUP BY e.JobTitle

);

GO

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.**

USE TelerikAcademy;

(SELECT MIN(e.FirstName + ' ' + e.LastName) AS [Name],

'Department: '+ d.Name, MIN(e.Salary) AS [Average Salary]

FROM Employees AS e

JOIN Departments AS d

ON e.DepartmentID = d.DepartmentID

GROUP BY d.Name

)

UNION

(SELECT MIN(e.FirstName + ' ' + e.LastName) AS [Name], 'Job: ' + e.JobTitle, MIN(e.Salary) AS [Average Salary]

FROM Employees AS e

GROUP BY e.JobTitle

);

GO

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

USE TelerikAcademy;

SELECT TOP 1 t.Name, COUNT(\*) AS [Employees Number]

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(\*) DESC;

GO

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

USE TelerikAcademy;

SELECT t.Name as Town, COUNT(e.ManagerID) AS [Managers Number]

FROM Employees e

JOIN Addresses a

ON e.AddressID = a.AddressID

JOIN Towns t

ON t.TownID = a.TownID

WHERE e.EmployeeID in

(SELECT DISTINCT ManagerID

FROM Employees)

GROUP BY t.Name

GO

1. **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.**

USE TelerikAcademy;

CREATE TABLE WorkHours(

WorkHoursID int IDENTITY,

EmployeeID int,

WorkDate datetime,

Task nvarchar(50),

WorkHours int,

Comment nvarchar(256),

CONSTRAINT PK\_WorkHours PRIMARY KEY(WorkHoursID),

CONSTRAINT FK\_WorkHours\_Employees FOREIGN KEY(EmployeeID) REFERENCES Employees(EmployeeID)

)

GO

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

USE TelerikAcademy;

INSERT INTO WorkHours(WorkDate, Task, WorkHours, Comment)

VALUES (GETDATE(), 'Fix bug #256', 23, 'No commits before bug fixed'),

(GETDATE(), 'Implement Composite pattern', 3, '15% bonus'),

(GETDATE(), 'Add like button to the form', 6, 'No bonus');

DELETE FROM WorkHours

WHERE Task LIKE '%bug #256%';

UPDATE WorkHours

SET WorkHours = 2

WHERE Task LIKE '%like button%';

GO

**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 WorkHoursLog(

Id int IDENTITY,

OldRecord nvarchar(100) NOT NULL,

NewRecord nvarchar(100) NOT NULL,

Command nvarchar(10) NOT NULL,

CONSTRAINT PK\_WorkHoursLog PRIMARY KEY(Id)

)

GO

DROP TRIGGER tr\_WorkHoursInsert;

GO

CREATE TRIGGER tr\_WorkHoursInsert ON WorkHours FOR INSERT

AS

INSERT INTO WorkHoursLog(OldRecord, NewRecord, Command)

VALUES(' ',

(SELECT 'Day: ' + CAST(WorkDate AS nvarchar(50)) + ' ' + ' Task: ' + Task + ' ' + ' Hours: ' + CAST([WorkHours] AS nvarchar(50)) + ' ' + CAST([WorkHours] AS nvarchar(50))Comment

FROM Inserted),

'INSERT'

);

GO

DROP TRIGGER tr\_WorkHoursUpdate;

GO

CREATE TRIGGER tr\_WorkHoursUpdate ON WorkHours FOR UPDATE

AS

INSERT INTO WorkHoursLog(OldRecord, NewRecord, Command)

VALUES((SELECT 'Day: ' + CAST(WorkDate AS nvarchar(50)) + ' ' + ' Task: ' + Task + ' ' + ' Hours: ' + CAST([WorkHours] AS nvarchar(50)) + ' ' + Comment

FROM Deleted),

(SELECT 'Day: ' + CAST(WorkDate AS nvarchar(50)) + ' ' + ' Task: ' + Task + ' ' + ' Hours: ' + CAST([WorkHours] AS nvarchar(50)) + ' ' + Comment

FROM Inserted),

'UPDATE'

);

GO

DROP TRIGGER tr\_WorkHoursDeleted;

GO

CREATE TRIGGER tr\_WorkHoursDeleted ON WorkHours AFTER DELETE

AS

INSERT INTO WorkHoursLog(OldRecord, NewRecord, Command)

VALUES(

(SELECT 'Day: ' + CAST(WorkDate AS nvarchar(50)) + ' ' + ' Task: ' + Task + ' ' + ' Hours: ' + CAST([WorkHours] AS nvarchar(50)) + ' ' + Comment

FROM Deleted),

' ',

'DELETE'

);

GO

-- Simple test queries

INSERT INTO WorkHours(WorkDate, Task, WorkHours, Comment)

VALUES(GETDATE(), 'Very important task', 56, 'It must be done!');

DELETE FROM WorkHours

WHERE Comment = 'It must be done!';

UPDATE WorkHours

SET Task = Task + ' and Observer pattern'

WHERE Task = 'Implement Composite pattern';

GO

1. **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.**

USE TelerikAcademy;

BEGIN TRAN

ALTER TABLE Departments

DROP CONSTRAINT FK\_Departments\_Employees;

ALTER TABLE Departments

ADD CONSTRAINT FK\_Departments\_Employees

FOREIGN KEY (ManagerID)

REFERENCES Employees (EmployeeID)

ON DELETE CASCADE

ON UPDATE CASCADE;

ALTER TABLE EmployeesProjects

DROP CONSTRAINT FK\_EmployeesProjects\_Employees;

ALTER TABLE EmployeesProjects

ADD CONSTRAINT FK\_EmployeesProjects\_Employees

FOREIGN KEY (EmployeeID)

REFERENCES Employees (EmployeeID)

ON DELETE CASCADE

ON UPDATE CASCADE;

ALTER TABLE WorkHours

DROP CONSTRAINT FK\_WorkHours\_Employees;

ALTER TABLE WorkHours

ADD CONSTRAINT FK\_WorkHours\_Employees

FOREIGN KEY (EmployeeID)

REFERENCES Employees (EmployeeID)

ON DELETE CASCADE

ON UPDATE CASCADE;

DELETE FROM Employees

SELECT d.Name

FROM Employees e

JOIN Departments d

ON e.DepartmentID = d.DepartmentID

WHERE d.Name = 'Sales'

GROUP BY d.Name

ROLLBACK TRAN;

GO

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

USE TelerikAcademy;

BEGIN TRAN

DROP TABLE EmployeesProjects;

-- ROLLBACK of course ;)

ROLLBACK TRAN;

GO

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.**

USE TelerikAcademy;

CREATE TABLE #BackUpEmployeesProjects(

EmployeeID INT NOT NULL,

ProjectID INT NOT NULL,

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

);

INSERT INTO #BackUpEmployeesProjects

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 #BackUpEmployeesProjects;

GO