# Exercises: Functions and Stored Procedures

This document defines the **exercise assignments** for the ["Databases Basics - MSSQL" course @ Software University.](https://softuni.bg/trainings/3491/ms-sql-september-2021)

# Queries for SoftUni Database

## Employees with Salary Above 35000

Create storedprocedure **usp\_GetEmployeesSalaryAbove35000** that returns **all employees’ first and last names** for whose **salary is above 35000**.

#### Example

|  |  |
| --- | --- |
| **First Name** | **Last Name** |
| Roberto | Tamburello |
| David | Bradley |
| Terri | Duffy |
| … | … |

CREATE PROCEDURE usp\_GetEmployeesSalaryAbove35000 AS

BEGIN

SELECT [FirstName] AS [First Name],

[LastName] AS [Last Name]

FROM [Employees]

WHERE [Salary] > 35000

END

## Employees with Salary Above Number

Create a stored procedure **usp\_GetEmployeesSalaryAboveNumber** that **accept a number** (of type **DECIMAL(18,4)**) as parameter and returns **all employees’ first and last names** whose salary is **above or equal** to the given number.

#### Example

Supplied number for that example is 48100.

|  |  |
| --- | --- |
| **First Name** | **Last Name** |
| Terri | Duffy |
| Jean | Trenary |
| Ken | Sanchez |
| … | … |

CREATE PROCEDURE usp\_GetEmployeesSalaryAboveNumber @minSalary DECIMAL(18, 4) AS

BEGIN

SELECT [FirstName] AS [First Name],

[LastName] AS [Last Name]

FROM [Employees]

WHERE [Salary] >= @minSalary

END

## Town Names Starting With

Create a stored procedure **usp\_GetTownsStartingWith** that **accept string as parameter** and returns **all town names starting with that string.**

#### Example

Here is the list of all towns **starting with "b".**

|  |
| --- |
| **Town** |
| Bellevue |
| Bothell |
| Bordeaux |
| Berlin |

CREATE PROCEDURE usp\_GetTownsStartingWith @startLetters NVARCHAR(25) AS

BEGIN

SELECT [Name] AS [Town]

FROM [Towns]

WHERE [NAME] LIKE @startLetters + '%'

END

## Employees from Town

Create a stored procedure **usp\_GetEmployeesFromTown** that accepts **town name** as parameter and return the **employees’ first and last name that live in the given town.**

#### Example

Here it is a list of employees **living in Sofia.**

|  |  |
| --- | --- |
| **First Name** | **Last Name** |
| Svetlin | Nakov |
| Martin | Kulov |
| George | Denchev |

CREATE PROCEDURE usp\_GetEmployeesFromTown @townName VARCHAR(50) AS

BEGIN

SELECT e.[FirstName] AS [First Name],

e.[LastName] AS [Last Name]

FROM [Employees] AS e

LEFT JOIN [Addresses] AS a

ON e.[AddressID] = a.AddressID

LEFT JOIN [Towns] AS t

ON a.[TownID] = t.[TownID]

WHERE t.[Name] = @townName

END

## Salary Level Function

Create a function **ufn\_GetSalaryLevel(@salary DECIMAL(18,4))** that receives **salary of an employee** and returns the **level of the salary**.

* If salary is **< 30000** return **"Low"**
* If salary is **between 30000 and 50000 (inclusive)** return **"Average"**
* If salary is **> 50000** return **"High"**

#### Example

|  |  |
| --- | --- |
| **Salary** | **Salary Level** |
| 13500.00 | Low |
| 43300.00 | Average |
| 125500.00 | High |

CREATE FUNCTION ufn\_GetSalaryLevel(@salary DECIMAL(18,4)) RETURNS VARCHAR(7) AS

BEGIN

DECLARE @salaryLevel VARCHAR(7)

IF @salary < 30000

BEGIN

SET @salaryLevel = 'Low'

END

ELSE IF @salary BETWEEN 30000 AND 50000

BEGIN

SET @salaryLevel = 'Average'

END

ELSE

BEGIN

SET @salaryLevel = 'High'

END

RETURN @salaryLevel

END

## Employees by Salary Level

Create a stored procedure **usp\_EmployeesBySalaryLevel** that receive as **parameter** **level of salary** (low, average, or high) and print the **names of all employees** that have given level of salary. You should use the function - "**dbo.ufn\_GetSalaryLevel(@Salary)** ", which was part of the previous task, inside your "**CREATE PROCEDURE …**" query.

#### Example

Here is the list of all employees with high salary.

|  |  |
| --- | --- |
| **First Name** | **Last Name** |
| Terri | Duffy |
| Jean | Trenary |
| Ken | Sanchez |
| … | … |

CREATE PROCEDURE usp\_EmployeesBySalaryLevel @salaryLevel VARCHAR(7) AS

BEGIN

SELECT [FirstName],

[LastName]

FROM [Employees]

WHERE dbo.ufn\_GetSalaryLevel([Salary]) = @salaryLevel

END

## Define Function

Define a function **ufn\_IsWordComprised(@setOfLetters, @word)** that returns **true** or **false** depending on that if the word is comprised of the given set of letters.

#### Example

|  |  |  |
| --- | --- | --- |
| **SetOfLetters** | **Word** | **Result** |
| oistmiahf | Sofia | 1 |
| oistmiahf | halves | 0 |
| bobr | Rob | 1 |
| pppp | Guy | 0 |

CREATE FUNCTION ufn\_IsWordComprised(@setOfLetters NVARCHAR(50), @word NVARCHAR(50))

RETURNS BIT

AS

BEGIN

DECLARE @i TINYINT = 1;

WHILE LEN(@word) >= @i

BEGIN

DECLARE @currentLetter NVARCHAR(1) = SUBSTRING(@word, @i, 1);

IF(@setOfLetters NOT LIKE '%' + @currentLetter + '%') -- IF(CHARINDEX(@currentLetter, @setOfLetters) = 0)

RETURN 0

SET @i += 1;

END

RETURN 1

END

## \* Delete Employees and Departments

Create a **procedure** with the name **usp\_DeleteEmployeesFromDepartment (@departmentId** INT**) which deletes all Employees** from a **given** **department**. **Delete these departments** from the **Departments table** too. **Finally, SELECT** the **number** of **employees** from the **given department**. If the delete statements are correct the select query should return 0.

After completing that exercise restore your database to revert all changes.

#### Hint:

You may set **ManagerID** column in Departments table to **nullable** (using query "ALTER TABLE …").

CREATE PROCEDURE usp\_DeleteEmployeesFromDepartment (@departmentId INT)

AS

BEGIN

DELETE FROM [EmployeesProjects]

WHERE [EmployeeID] IN (

SELECT [EmployeeID]

FROM [Employees]

WHERE [DepartmentID] = @departmentId

)

UPDATE [Employees]

SET [ManagerID] = NULL

WHERE [ManagerID] IN (

SELECT [EmployeeID]

FROM [Employees]

WHERE [DepartmentID] = @departmentId

)

ALTER TABLE [Departments]

ALTER COLUMN [ManagerID] INT --NULL

UPDATE [Departments]

SET [ManagerID] = NULL

WHERE [ManagerID] IN (

SELECT [EmployeeID]

FROM [Employees]

WHERE [DepartmentID] = @departmentId

)

DELETE FROM [Employees]

WHERE [DepartmentID] = @departmentId

DELETE FROM [Departments]

WHERE [DepartmentID] = @departmentId

SELECT COUNT(\*)

FROM [Employees]

WHERE [DepartmentID] = @departmentId

END

# Queries for Bank Database

## Find Full Name

You are given a database schema with tables **AccountHolders(Id (PK), FirstName, LastName, SSN)** and **Accounts(Id (PK), AccountHolderId (FK), Balance)**. Write a stored procedure **usp\_GetHoldersFullName** that selects the full names of all people.

#### Example

|  |
| --- |
| **Full Name** |
| Susan Cane |
| Kim Novac |
| Jimmy Henderson |
| … |

CREATE PROC usp\_GetHoldersFullName AS

BEGIN

SELECT [FirstName] + ' ' + [LastName] AS [Full Name]

FROM [AccountHolders]

END

## People with Balance Higher Than

Your task is to create a stored procedure **usp\_GetHoldersWithBalanceHigherThan** that accepts a **number as a parameter** and returns all **people who have more money in a total of all their accounts than the supplied number**. Order them by the first name, then by the last name.

#### Example

|  |  |
| --- | --- |
| **First Name** | **Last Name** |
| Monika | Miteva |
| Petar | Kirilov |
| … | … |

CREATE PROC usp\_GetHoldersWithBalanceHigherThan(@inputSalary MONEY) AS

BEGIN

SELECT

ah.FirstName,

ah.LastName

FROM Accounts a

JOIN AccountHolders ah ON ah.Id = a.AccountHolderId

GROUP BY a.AccountHolderId, ah.FirstName, ah.LastName

HAVING SUM(a.Balance) > @inputSalary

ORDER BY ah.FirstName, ah.LastName

END

## Future Value Function

Your task is to create a function **ufn\_CalculateFutureValue** that accepts as parameters – **sum (decimal)**, **yearly interest rate (float)**, and **the number of years(int)**. It should calculate and return the future value of the initial sum rounded to the **fourth digit** after the decimal delimiter. Using the following formula:

* **I** – Initial sum
* **R** – Yearly interest rate
* **T** – Number of years

#### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| **Initial sum:** 1000  **Yearly Interest rate:** 10%  **years:** 5  ufn\_CalculateFutureValue(1000, 0.1, 5) | 1610.5100 |

CREATE FUNCTION ufn\_CalculateFutureValue (@sum DECIMAL(18, 4), @yearlyInterestRate FLOAT, @numberOfyears INT)

RETURNS DECIMAL(18, 4)

AS

BEGIN

DECLARE @x FLOAT = 1 + @yearlyInterestRate;

DECLARE @calculate DECIMAL(18,4) = @sum \* (POWER(@x, @numberOfyears));

RETURN @calculate

END

## Calculating Interest

Your task is to create a stored procedure **usp\_CalculateFutureValueForAccount** that uses the function from the previous problem to give an interest to a person's account **for 5 years**, along with information about his/her **account id, first name, last name, and current balance** as it is shown in the example below. It should take the **AccountId** and the **interest rate** as parameters. Again, you are provided with the **dbo.ufn\_CalculateFutureValue** function which was part of the previous task.

#### Example

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Account Id** | **First Name** | **Last Name** | **Current Balance** | **Balance in 5 years** |
| 1 | Susan | Cane | 123.12 | 198.2860 |

\*Note: for the example above interest rate is 0.1

CREATE PROC usp\_CalculateFutureValueForAccount (@accountID INT ,@interestRate FLOAT)

AS

BEGIN

SELECT

a.Id AS [Account Id],

ah.FirstName AS [First Name],

ah.LastName AS [Last Name],

a.Balance AS [Current Balance],

dbo.ufn\_CalculateFutureValue(a.Balance, @interestRate, 5) AS [Balance in 5 years]

FROM Accounts a

JOIN AccountHolders ah ON ah.Id = a.AccountHolderId

WHERE a.Id = @accountID

END

# Queries for Diablo Database

You are given a **database "Diablo"** holding users, games, items, characters, and statistics available as SQL script. Your task is to write some stored procedures, views, and other server-side database objects and write some SQL queries for displaying data from the database.

**Important:** start with a **clean copy of the "Diablo" database** **on each problem**. Just execute the SQL script again.

## \*Scalar Function: Cash in User Games Odd Rows

Create a **function** **ufn\_CashInUsersGames** that **sums the cash of odd rows**. Rows must be ordered by cash in descending order. The function should take a **game name** as a **parameter** and **return the result as a table**. Submit **only your function** **in**.

Execute the function over the following game names, ordered exactly like: "**Love in a mist**".

#### Output

|  |
| --- |
| **SumCash** |
| 8585.00 |

#### Hint

Use **ROW\_NUMBER** to get the rankings of all rows based on order criteria.

CREATE FUNCTION ufn\_CashInUsersGames (@gameName NVARCHAR(50))

RETURNS TABLE

AS

RETURN SELECT (

SELECT

SUM([Cash]) AS [SumCash]

FROM(

SELECT g.[Name],

ug.[Cash],

ROW\_NUMBER() OVER(PARTITION BY g.[Name] ORDER BY ug.[Cash] DESC) AS [RowNumber]

FROM [UsersGames] AS ug

JOIN [Games] AS g

ON ug.[GameId] = g.[Id]

WHERE g.[Name] = @gameName

) AS [RowNumberSubQuery]

WHERE [RowNumber] % 2 <> 0

) AS [SumCash]