# Exercises: Database Programmability and Transactions

This document defines the **exercise assignments** for the ["Databases Basics - MSSQL" course @ Software University.](https://softuni.bg/courses/databases-basics-ms-sql-server)

# Functions and Procedures

## 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**.

**Solution:**

CREATE PROC usp\_GetEmployeesSalaryAbove35000

AS

SELECT e.FirstName, e.LastName FROM Employees AS e

WHERE e.Salary > 35000 ;

#### Example

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

### Employees with Salary Above Number

Create 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.

**Solution:**

CREATE PROC usp\_GetEmployeesSalaryAboveNumber (@InputNumber DECIMAL(18,4))

AS

SELECT e.FirstName, e.LastName FROM Employees AS e

WHERE e.Salary >= @InputNumber

#### Example

Supplied number for that example is 48100.

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

### Town Names Starting With

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

**Solution:**

CREATE PROC usp\_GetTownsStartingWith (@InputString NVARCHAR(20))

AS

SELECT t.[Name] FROM Towns AS t

WHERE t.[Name] LIKE @InputString + '%'

#### Example

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

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

### Employees from Town

Write 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.**

**Solution:**

CREATE PROC usp\_GetEmployeesFromTown (@TownName NVARCHAR(30))

AS

SELECT e.FirstName AS [First Name], e.LastName AS [Last Name] FROM Employees AS e

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

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

WHERE t.[Name] = @TownName

#### Example

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

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

### Salary Level Function

Write 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”**

**Solution:**

CREATE FUNCTION ufn\_GetSalaryLevel(@Salary DECIMAL(18,4))

RETURNS NVARCHAR(7)

AS

BEGIN

DECLARE @SalaryLevel NVARCHAR(7)

IF @Salary <30000

BEGIN

SET @SalaryLevel = 'Low'

END

ELSE IF @Salary <=50000

BEGIN

SET @SalaryLevel = 'Average'

END

ELSE

BEGIN

SET @SalaryLevel = 'High'

END

RETURN @SalaryLevel;

END

#### Example

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

### Employees by Salary Level

Write 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.

**Solution:**

CREATE PROCEDURE usp\_EmployeesBySalaryLevel(@SalaryLevel VARCHAR(7))

AS

BEGIN

SELECT e.FirstName, e.LastName

FROM Employees AS e

WHERE dbo.ufn\_GetSalaryLevel(e.Salary) = @SalaryLevel

END

#### Example

Here is the list of all employees with high salary.

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

### Define Function

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

**Solution:**

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

RETURNS BIT

AS

BEGIN

DECLARE @result BIT

DECLARE @counter INT = 1

DECLARE @currentLetter CHAR

WHILE(@counter <= LEN( @word))

BEGIN

SET @currentLetter = SUBSTRING (@word, @counter, 1)

DECLARE @charIndex INT = CHARINDEX(@currentLetter,@setOfLetters )

IF( @charIndex <= 0)

BEGIN

RETURN 0

END

SET @counter+= 1

END

RETURN 1

END

#### Example

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

### \* Delete Employees and Departments

Write 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 …").

**Solution:**

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

UPDATE Departments

SET ManagerID = NULL

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.

**Solution:**

CREATE PROCEDURE usp\_GetHoldersFullName

AS

BEGIN

SELECT CONCAT(a.FirstName, ' ', a.LastName) AS [Full Name] FROM AccountHolders AS a

END

#### Example

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

### 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 total of all their accounts than the supplied number**. Order them by first name, then by last name

**Solution:**

CREATE PROCEDURE usp\_GetHoldersWithBalanceHigherThan (@minBalanace DECIMAL(18,4))

AS

BEGIN

SELECT ah.FirstName, ah.LastName FROM AccountHolders AS ah

JOIN Accounts AS a ON ah.Id = a.AccountHolderId

GROUP BY ah.FirstName, ah.LastName

HAVING SUM(a.Balance) > @minBalanace

ORDER BY ah.FirstName, ah.LastName

END

#### Example

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

### Future Value Function

Your task is to create a function **ufn\_CalculateFutureValue** that accepts as parameters – **sum (decimal)**, **yearly interest rate (float)** and **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

**Solution:**

CREATE FUNCTION dbo.ufn\_CalculateFutureValue (@sum DECIMAL(18,2), @yearlyInterest FLOAT, @numberOfYears INT)

RETURNS DECIMAL(18,4)

AS

BEGIN

DECLARE @result DECIMAL(18,4)

SET @result = @sum \* (POWER ((1+@yearlyInterest), @numberOfYears))

RETURN @result

#### END

#### Example

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

### 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 “**dbo.ufn\_CalculateFutureValue**” function which was part of the previous task.

**Solution:**

CREATE PROCEDURE usp\_CalculateFutureValueForAccount (@accountId INT, @yearlyInterest FLOAT)

AS

BEGIN

SELECT a.Id AS [Account Id], -- correct here

ah.FirstName AS [First Name],

ah.LastName AS [Last Name],

a.Balance AS [Current Balance],

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

FROM Accounts AS a

JOIN AccountHolders AS ah ON a.AccountHolderId = ah.Id

WHERE a.Id = @accountId

END

#### 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

## 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 table**. Submit **only your function** **in**.

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

**Solution:**

CREATE FUNCTION ufn\_CashInUsersGames (@gameName VARCHAR(30))

RETURNS @output TABLE (SumCash DECIMAL (18,4))

AS

BEGIN

INSERT INTO @output

SELECT SUM(r.SumCash) AS SumCash FROM

(

SELECT g.[Name],

ug.[Cash] AS SumCash,

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

FROM UsersGames AS ug

JOIN Games AS g ON ug.GameId = g.Id

GROUP BY g.[Name] ,ug.[Cash]

HAVING g.[Name] = @gameName

)

AS r

WHERE r.Ranking % 2 <> 0

GROUP BY r.Name

RETURN

END

#### Output

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

#### Hint

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