# 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**.

CREATE PROC usp\_GetEmployeesSalaryAbove35000

AS

SELECT FirstName, LastName

FROM Employees

WHERE 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.

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

AS

SELECT FirstName, LastName

FROM Employees

WHERE Salary >= @salary

#### 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.**

CREATE PROC usp\_GetTownsStartingWith(@text VARCHAR(MAX))

AS

SELECT [Name]

FROM Towns

WHERE [Name] LIKE @text + '%'

EXEC usp\_GetTownsStartingWith 'b'

#### 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.**

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

AS

SELECT FirstName, LastName

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”**

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

RETURNS VARCHAR(10)

AS

BEGIN

DECLARE @SalaryLevel VARCHAR(10)

BEGIN

IF(@salary < 30000)

BEGIN

SET @SalaryLevel = 'Low'

END

ELSE IF(@salary <= 50000)

BEGIN

SET @SalaryLevel = 'Average'

END

ELSE IF(@salary > 50000)

BEGIN

SET @SalaryLevel = 'High'

END

END

RETURN @SalaryLevel

END

SELECT Salary, dbo.ufn\_GetSalaryLevel(Salary) AS [Salary Level]

FROM Employees

#### 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.

CREATE PROC usp\_EmployeesBySalaryLevel(@SalaryLevel VARCHAR(10))

AS

SELECT FirstName, LastName

FROM Employees

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

EXEC usp\_EmployeesBySalaryLevel 'High'

#### 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.

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

RETURNS BIT

AS

BEGIN

DECLARE @WordLength INT = LEN(@word)

DECLARE @INDEX INT = 1

DECLARE @trueOrFalse BIT = 1

WHILE @WordLength >= @INDEX AND @trueOrFalse > 0

BEGIN

DECLARE @letterFromWord CHAR(1)

DECLARE @charIndex INT

SET @letterFromWord = SUBSTRING(@word, @INDEX, 1)

SET @charIndex = CHARINDEX(@letterFromWord, @setOfLetters)

IF(@charIndex > 0)

BEGIN

SET @trueOrFalse = 1

END

ELSE

BEGIN

SET @trueOrFalse = 0

END

SET @INDEX += 1

END

RETURN @trueOrFalse

END

DROP FUNCTION ufn\_IsWordComprised

SELECT dbo.ufn\_IsWordComprised('oistmiahf', 'Sofia')

SELECT dbo.ufn\_IsWordComprised('oistmiahf', 'halves')

#### 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 …").

CREATE PROC 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.

CREATE PROC usp\_GetHoldersFullName

AS

BEGIN

SELECT CONCAT(FirstName,' ', LastName) AS [Full Name]

FROM AccountHolders

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

CREATE PROC usp\_GetHoldersWithBalanceHigherThan(@money DECIMAL(18,2))

AS

BEGIN

SELECT ah.FirstName, ah.LastName

FROM Accounts AS a

JOIN AccountHolders AS ah

ON a.AccountHolderId = ah.Id

GROUP BY ah.FirstName, ah.LastName

HAVING SUM(a.Balance) > @money

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

CREATE FUNCTION ufn\_CalculateFutureValue(@initialSum DECIMAL(18, 4), @yearlyInterestRate FLOAT, @numberOfYears INT)

RETURNS DECIMAL(18,4)

AS

BEGIN

DECLARE @futureValue DECIMAL (18, 4) = @initialSum \* POWER(1+@yearlyInterestRate, @numberOfYears)

RETURN @futureValue

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.

CREATE PROC usp\_CalculateFutureValueForAccount(@AccountID INT, @InterestRate FLOAT)

AS

BEGIN

SELECT

ah.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 AS a

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

WHERE @AccountID = a.Id

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**”.

CREATE FUNCTION ufn\_CashInUsersGames(@NameGame NVARCHAR(MAX))

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

AS

BEGIN

INSERT INTO @output SELECT(

SELECT SUM(Cash) AS [SumCash]

FROM (

SELECT \*, ROW\_NUMBER() OVER(ORDER BY Cash DESC) AS [RowNum]

FROM UsersGames

WHERE GameId IN (

SELECT Id

FROM Games

WHERE [Name] = @NameGame

)) AS [RowNumTable]

WHERE [RowNum] % 2 <> 0)

RETURN;

END

#### Output

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

#### Hint

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

# Triggers and Transactions

## Queries for Bank Database

### Create Table Logs

Create a table – **Logs** (LogId, AccountId, OldSum, NewSum). Add a **trigger** to the Accounts table that **enters** a new entry into the **Logs** table every time the sum **on** an **account** **changes**. Submit **only** the **query** that **creates** the **trigger**.

#### Example

|  |  |  |  |
| --- | --- | --- | --- |
| **LogId** | **AccountId** | **OldSum** | **NewSum** |
| 1 | 1 | 123.12 | 113.12 |
| … | … | … | … |

### Create Table Emails

Create another table – **NotificationEmails**(Id, Recipient, Subject, Body). Add a **trigger** to logs table and **create new email whenever new record is inserted in logs table.** The following data is required to be filled for each email:

* **Recipient** – AccountId
* **Subject** – “Balance change for account: **{AccountId}**”
* **Body** - “On **{date}** your balance was changed from **{old}** to **{new}.**”

**Submit** your query **only** for the **trigger** action.

#### Example

|  |  |  |  |
| --- | --- | --- | --- |
| **Id** | **Recipient** | **Subject** | **Body** |
| 1 | 1 | Balance change for account: 1 | On Sep 12 2016 2:09PM your balance was changed from 113.12 to 103.12. |
| … | … | … | … |

### Deposit Money

Add stored procedure **usp\_DepositMoney** (**AccountId**, **MoneyAmount**) that deposits money to an existing account. Make sure to guarantee valid positive MoneyAmount with precision up to **fourth sign after decimal point**. The procedure should produce exact results working with the specified precision.

#### Example

Here is the result for **AccountId = 1** and **MoneyAmount = 10.**

|  |  |  |
| --- | --- | --- |
| **AccountId** | **AccountHolderId** | **Balance** |
| 1 | 1 | 133.1200 |

### Withdraw Money

Add stored procedure **usp\_WithdrawMoney** (**AccountId**, **MoneyAmount**) that withdraws money from an existing account. Make sure to guarantee valid positive MoneyAmount with precision up to **fourth sign after decimal point**. The procedure should produce exact results working with the specified precision.

#### Example

Here is the result for **AccountId = 5** and **MoneyAmount = 25.**

|  |  |  |
| --- | --- | --- |
| **AccountId** | **AccountHolderId** | **Balance** |
| 5 | 11 | 36496.2000 |

### Money Transfer

Write stored procedure **usp\_TransferMoney**(SenderId, ReceiverId, Amount) that **transfers money from one account to another**. Make sure to guarantee valid positive MoneyAmount with precision up to **fourth sign after decimal point**. Make sure that the whole procedure **passes without errors** and **if error occurs make** **no change in the database.** You can use both: “**usp\_DepositMoney**”, “**usp\_WithdrawMoney**” (look at previous two problems about those procedures).

#### Example

Here is the result for SenderId **= 5,** ReceiverId = 1and **MoneyAmount = 5000.**

|  |  |  |
| --- | --- | --- |
| **AccountId** | **AccountHolderId** | **Balance** |
| 1 | 1 | 5123.12 |
| 5 | 11 | 31521.2000 |

## 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.

### Trigger

1. Users **should not** be allowed to buy items with **higher level** than **their** **level**. Create a **trigger** that **restricts** that. The trigger should prevent **inserting items** that are above specified level while allowing all others to be inserted.
2. Add bonus cash of **50000** to users: **baleremuda, loosenoise, inguinalself, buildingdeltoid, monoxidecos** in the game **“Bali”.**
3. There are two groups of **items** that you must buy for the above users. The first are items with **id between 251 and 299 including**. Second group are items with **id between 501 and 539 including.  
   Take** off **cash** from each user **for** the bought **items**.
4. Select all users in the current game (“Bali”) with their items. Display **username**, **game name**, **cash** and **item name**. Sort the result by username alphabetically, then by item name alphabetically.

#### Output

|  |  |  |  |
| --- | --- | --- | --- |
| **Username** | **Name** | **Cash** | **Item Name** |
| baleremuda | Bali | 41153.00 | Iron Wolves Doctrine |
| baleremuda | Bali | 41153.00 | Irontoe Mudsputters |
| … | … | … | … |
| buildingdeltoid | Bali | 38800.00 | Alabaster Gloves |
| … | … | … | … |

### \*Massive Shopping

1. User **Stamat** in **Safflower** gamewants to buy some items. He likes all items **from Level 11 to 12** as well as all items **from Level 19 to 21.** As it is a bulk operation you have to **use transactions.**
2. A transaction is the operation of taking out the cash from the user in the current game as well as adding up the items.
3. Write transactions for each level range. If anything goes wrong turn back the changes inside of the transaction.
4. Extract all of **Stamat**’s item names in the given game sorted by name alphabetically

#### Output

|  |
| --- |
| **Item Name** |
| Akarats Awakening |
| Amulets |
| Angelic Shard |
| … |

## Queries for SoftUni Database

### Employees with Three Projects

Create a procedure **usp\_AssignProject(@emloyeeId, @projectID)** that **assigns** **projects** to employee. If the employee has more than **3** project throw **exception** and **rollback** the changes. The exception message must be: "**The employee has too many projects!**" with Severity = 16, State = 1.

### Delete Employees

Create a table Deleted\_Employees(EmployeeId PK, FirstName, LastName, MiddleName, JobTitle, DepartmentId, Salary) that will hold information about fired (deleted) employees from the **Employees** table. Add a trigger to **Employees** table that inserts the corresponding information about the deleted records in Deleted\_Employees.