Exercises: Database Programmability and Transactions

This document defines the **exercise assignments** for the "Databases Basics - MSSQL" course @ Software University.

Section I. Functions and Procedures

Part 1. Queries for SoftUni Database

Problem 1. Employees with Salary Above 35000

Create stored procedure **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
…	…

Problem 2. 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.

Example

Supplied number for that example is 48100.

First Name	Last Name
Terri	Duffy
Jean	Trenary
Ken	Sanchez
…	…

Problem 3. Town Names Starting With

Write 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

Problem 4. 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.**

Example

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

First Name	Last Name
Svetlin	Nakov
Martin	Kulov
George	Denchev

Problem 5. 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"**

Example

Salary	Salary Level
13500.00	Low
43300.00	Average
125500.00	High

Problem 6. 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.

Example

Here is the list of all employees with high salary.

First Name	Last Name
Terri	Duffy
Jean	Trenary
Ken	Sanchez
…	…

Problem 7. 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.

Example

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

Problem 8. * 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 …").

Part 2. Queries for Bank Database

Problem 9. 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
…

Problem 10. 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

Example

First Name	Last Name
Monika	Miteva
Petar	Kirilov
…	…

Problem 11. 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:

$$FV = I \times ((1 + R)^T)$$

- **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`

Problem 12. 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.

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

Part 3. 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.

Problem 13. *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**".

Output

SumCash
8585.00

Hint

Use **ROW_NUMBER** to get the rankings of all rows based on order criteria.

Section II. Triggers and Transactions

Part 1. Queries for Bank Database

Problem 14. 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
…	…	…	…

Problem 15. 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.
…	…	…	…

Problem 16. 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

Problem 17. 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

Problem 18. 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 = 1** and **MoneyAmount = 5000.**

AccountId	AccountHolderId	Balance
1	1	5123.12
5	11	31521.2000

Part 2. 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.

Problem 19. 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
…	…	…	…

Problem 20. *Massive Shopping

1. User **Stamat** in **Safflower** game wants 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
…

Part 3. Queries for SoftUni Database

Problem 21. 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.

Problem 22. 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`**.