# Exercises: Triggers 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)

# PART I – 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 |

# PART II – 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 |
| … |

# Part III – 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.