# Exercises: Functions, Triggers and Transactions

This document defines the **exercise assignments** for the ["Databases Basics - MySQL" course @ Software University.](https://softuni.bg/trainings/1634/databases-basics-mysql-may-2017)

# Part I – Queries for SoftUni Database

## Employees with Salary Above 35000

Create storedprocedure **usp\_get\_employees\_salary\_above\_35000** that returns **all employees’ first and last names** for whose **salary is above 35000**. The result should be sorted by **first\_name** then by **last\_name** alphabetically. Submit your query statement as **Run skeleton, run queries & check DB in Judge.**

### Example

|  |  |
| --- | --- |
| **first\_name** | **last\_name** |
| Amy | Alberts |
| Brian | Welcker |
| Dan | Wilson |
| … | … |

## Employees with Salary Above Number

Create stored procedure **usp\_get\_employees\_salary\_above** that **accept a number** as parameter and return **all employees’ first and last names** whose salary is **above or equal** to the given number. The result should be sorted by **first\_name** then by **last\_name** alphabetically. Submit your query statement as **Run skeleton, run queries & check DB in Judge.**

### Example

Supplied number for that example is 48100.

|  |  |
| --- | --- |
| **first\_name** | **last\_name** |
| Amy | Alberts |
| Brian | Welcker |
| Dylan | Miller |
| … | … |

## Town Names Starting With

Write a stored procedure **usp\_get\_towns\_starting\_with** that **accept string as parameter** and returns **all town names starting with that string.** The result should be sorted by **town name** alphabetically. Submit your query statement as **Run skeleton, run queries & check DB in Judge.**

### Example

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

|  |
| --- |
| **town\_name** |
| Bellevue |
| Berlin |
| Bordeaux |
| Bothell |

## Employees from Town

Write a stored procedure **usp\_get\_employees\_from\_town** that accepts **town name** as parameter and return the **employees’ first and last name that live in the given town.** The result should be sorted by **first\_name** then by **last\_name** alphabetically. Submit your query statement as **Run skeleton, run queries & check DB in Judge.**

### Example

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

|  |  |
| --- | --- |
| **first\_name** | **last\_name** |
| George | Denchev |
| Martin | Kulov |
| Svetlin | Nakov |

## Salary Level Function

Write a function **ufn\_get\_salary\_level** 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”**

Submit your query statement as **Run skeleton, run queries & check DB in Judge.**

### Example

|  |  |
| --- | --- |
| **salary** | **salary\_Level** |
| 13500.00 | Low |
| 43300.00 | Average |
| 125500.00 | High |

## Employees by Salary Level

Write a stored procedure **usp\_get\_employees\_by\_salary\_level** that receive as **parameter** **level of salary** (low, average or high) and print the **names of all employees** that have given level of salary. The result should be sorted by **first\_name** then by **last\_name** both in descending order.

Submit your query statement as **Run skeleton, run queries & check DB in Judge.**

### Example

Here is the list of all employees with **high salary**.

|  |  |
| --- | --- |
| **first\_name** | **last\_name** |
| Terri | Duffy |
| Laura | Norman |
| Ken | Sanchez |
| … | … |

## Define Function

Define a function **ufn\_is\_word\_comprised(set\_of\_letters, word)** that returns **true** or **false** depending on that if the word is a comprised of the given set of letters.

Submit your query statement as **Run skeleton, run queries & check DB in Judge.**

### Example

|  |  |  |
| --- | --- | --- |
| **set\_of\_letters** | **word** | **result** |
| oistmiahf | Sofia | 1 |
| oistmiahf | halves | 0 |
| bobr | Rob | 1 |
| pppp | Guy | 0 |

## \* Delete Employees and Departments

Write a SQL query to delete all **employees** from the **Production** and **Production Control** departments. Also **delete these departments from the departments table.** After that exercise restore your database to revert those changes.

Submit your query statement as **Run skeleton, run queries & check DB in Judge.**

# PART II – Queries for Bank Database

## Find Full Name

You are given a database schema with tables **account\_holders(id (PK), first\_name, last\_name, ssn)** and **accounts(id (PK), account\_holder\_id (FK), balance)**. Write a stored procedure **usp\_get\_holders\_full\_name** that selects the full names of all people. **.** The result should be sorted by **full\_name** alphabetically. Submit your query statement as **Run skeleton, run queries & check DB in Judge.**

### Example

|  |
| --- |
| **full\_name** |
| Bjorn Sweden |
| Jimmy Henderson |
| Kim Novac |
| … |

## People with Balance Higher Than

Your task is to create a stored procedure **usp\_get\_holders\_with\_balance\_higher\_than** 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**.

The result should be sorted by **first\_name** then by **last\_name** alphabetically.

Submit your query statement as **Run skeleton, run queries & check DB in Judge.**

### Example

Supplied number for that example is 7000.

|  |  |
| --- | --- |
| **first\_name** | **last\_name** |
| Monika | Miteva |
| Petar | Kirilov |
| … | … |

## Future Value Function

Your task is to create a function **ufn\_calculate\_future\_value** that accepts as parameters – **sum**, **yearly interest rate** and **number of years**. It should calculate and return the future value of the initial sum. Using the following formula:

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

Submit your query statement as **Run skeleton, run queries & check DB in Judge.**

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| **Initial sum:** 1000  **Yearly Interest rate:** 10%  **years:** 5  ufn\_calculate\_future\_value(1000, 0.1, 5) | 1610.51 |

## Calculating Interest

Your task is to create a stored procedure **usp\_calculate\_future\_value\_for\_account** 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 **account\_id** and the **interest\_rate** as parameters. Interest rate should have precision up to 0.0001, same as the calculated balance after 5 years. **Be extremely careful to achieve the desired precision!**

Submit your query statement as **Run skeleton, run queries & check DB in Judge.**

### Example

Here is the result for **account\_id = 1** and **interest\_rate = 0.1.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **account\_id** | **fist\_name** | **last\_name** | **current\_balance** | **balance\_in\_5\_years** |
| 1 | Susan | Cane | 123.1200 | 198.2860 |

## Deposit Money

Add stored procedure **usp\_deposit\_money** (account\_id, money\_amount) that operate in transactions.

Make sure to guarantee valid positive money\_amount with precision up to fourth sign after decimal point. The procedure should produce exact results working with the specified precision.

Submit your query statement as **Run skeleton, run queries & check DB in Judge.**

### Example

Here is the result for **account\_id = 1** and **money\_amount = 10.**

|  |  |  |
| --- | --- | --- |
| **account\_id** | **account\_holder\_id** | **balance** |
| 1 | 1 | 133.1200 |

## Withdraw Money

Add stored procedures **usp\_withdraw\_money** (account\_id, money\_amount) that operate in transactions.

Make sure to guarantee withdraw is done only when balance is enough and money\_amount is valid positive number. Work with precision up to fourth sign after decimal point. The procedure should produce exact results working with the specified precision.

Submit your query statement as **Run skeleton, run queries & check DB in Judge.**

### Example

Here is the result for **account\_id = 1** and **money\_amount = 10.**

|  |  |  |
| --- | --- | --- |
| **account\_id** | **account\_holder\_id** | **balance** |
| 1 | 1 | 113.1200 |

## Money Transfer

Write stored procedure **usp\_transfer\_money(from\_account\_id, to\_account\_id, amount)** that **transfers money from one account to another**. Consider cases when one of the **account\_ids** is not valid, the amount of **money is negative number, outgoing balance** is enough or transferring **from/to one and the same account.** Make sure that the whole procedure **passes without errors** and **if error occurs make** **no change in the database.**

Make sure to guarantee exact results working with precision up to fourth sign after decimal point.

Submit your query statement as **Run skeleton, run queries & check DB in Judge.**

### Example

Here is the result for **from\_account\_id = 1, to\_account\_id = 2** and **money\_amount = 10.**

|  |  |  |
| --- | --- | --- |
| **account\_id** | **account\_holder\_id** | **balance** |
| 1 | 1 | 113.1200 |
| 2 | 3 | 4364.2300 |

## Log Accounts Trigger

Create another table – **logs** (log\_id, account\_id, old\_sum, new\_sum). 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 your query statement as **Run skeleton, run queries & check DB in Judge.**

### Example

|  |  |  |  |
| --- | --- | --- | --- |
| **log\_id** | **account\_id** | **old\_sum** | **new\_sum** |
| 1 | 1 | 123.12 | 113.12 |
| … | … | … | … |

## Emails Trigger

Create another table – **notification\_emails**(id, recipient, subject, body). Add a trigger to logs table to **create new email whenever new record is inserted in logs table.** The following data is required to be filled for each email:

* **recipient** – account\_id
* **subject** – “Balance change for account: **{account\_id}**”
* **body** - “On **{date}** your balance was changed from **{old}** to **{new}.**”

Submit your query statement as **Run skeleton, run queries & check DB in Judge.**

### Example

|  |  |  |  |
| --- | --- | --- | --- |
| **id** | **recipient** | **subject** | **body** |
| 1 | 1 | Balance change for account: 1 | On Sep 15 2016 at 11:44:06 AM your balance was changed from 133 to 143. |
| … | … | … | … |

# PART III – 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.

## 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.** The user will try to by items from the two groups separately one after another. As it is a bulk operation you have to **use transactions.** The user is not allowed to have negative **cash** in its game, he is only able to by all items in a group, if his cash is enough, or none of the items in the group.
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 item names in the given game sorted by name alphabetically.
5. Extract the remaining cash of the user **Stamat** in **Safflower** game.

Create stored procedure **usp\_massive\_shopping** **()** that combines subtasks from 1 to 5. Submit your procedure as **Run skeleton, Run queries & check DB in Judge.**

### Output

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

|  |
| --- |
| **cash** |
| 1489.0000 |

## Buy Items for User in Game

1. User **Alex** isin theshop in the game “**Edinburgh**” and she wants to buy some items. She likes all items of the list :

**Blackguard**,

**Bottomless Potion of Amplification**,

**Eye of Etlich (Diablo III)**,

**Gem of Efficacious Toxin**,

**Golden Gorget of Leoric,**

**Ziggurat Tooth,**

**The Three Hundredth Spear,**

**The Short Mans Finger,**

**Tzo Krins Gaze,**

**Valtheks Rebuke,**

**Utars Roar,**

**Urn of Quickening,**

**Boots,**

**Bombardiers Rucksack,**

**Cloak of Deception,**

**Hellfire Amulet**.

**Buy all the items in the specified order.** You should add the data in the right tables. Get the money for the items from user in game **cash**. Alex should NOT be allowed to by items with min\_level higher than the current level of her game. Make sure Alex will buy only items she is allowed to buy, according her game level, and items she can afford.

1. Select all users in the current game with their items. Display username, game name, cash and item name. Sort the result by item name.

Create stored procedure **usp\_buy\_items\_for\_alex** **()** that combines subtasks 1 and 2. Submit your procedure as **Run skeleton, Run queries & check DB in Judge.**

### Output

|  |  |  |  |
| --- | --- | --- | --- |
| **Username** | **Name** | **Cash** | **Item Name** |
| Alex | Edinburgh | 3763.0000 | Akanesh, the Herald of Righteousness |
| … | … | … | … |
| corruptpizz | Edinburgh | \*\*\*\*.\*\* | Broken Crown |
| … | … | … | … |
| printerstencils | Edinburgh | \*\*\*\*.\*\* | Envious Blade |