Exercises: Functions, Triggers and Transactions

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

Part I – Queries for SoftUni Database

1. Employees with Salary Above 35000

Create stored procedure **`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, and **`id`** **ascending**. Submit your query statement as Run skeleton, run queries & check DB in Judge.

Example

first_name	last_name
Amy	Alberts
Brian	Welcker
Dan	Wilson
…	…

2. 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** and **`id`** **ascending**. 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
…	…

3. 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

4. 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 and **id ascending**. 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

5. 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

6. 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
...	...

7. Define Function

Define a function **ufn_is_word_comprised(set_of_letters varchar(50), word varchar(50))** 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

PART II – Queries for Bank Database

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

Example

full_name
Bjorn Sweden
Jimmy Henderson
Kim Novac
…

9. 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 and **`account id`** ascending.
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
…	…

10. 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:

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

- **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`

11. 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

12. 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

13. 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

14. 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

15. 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

The following data in logs table is inserted after updating balance of account with **account_id = 1** with 10.

log_id	account_id	old_sum	new_sum
1	1	123.12	113.12
2	1	145.43	155.43

16. 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 (current 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.
…	…	…	…