Tutorial 3: SQL Aggregations, Foreign Keys and Triggers

SQL Aggregations

The goal of this exercise is to practice aggregations and grouping by in the SQL SELECT statement.

1. Review the slides of Week 2 and 3.
2. Login to MySQL and DBeaver and make sure you have the tables `products`, `pcs`, `laptops`, `printers`, `customers` and `sales` in your database.
3. Open the screen on DBeaver to write and execute SQL statements.
4. Write SQL SELECT statements for performing the following queries:
    - a. Find the makers who make exactly three different PC models. List all such makers, and for each of them, list also the average speed of the three PC models they make.
    - b. List the makers who make at least two PC models with different speed.
    - c. List the makers who make at least two different computers (PCs or laptops) with speed of at least 2.80.
    - d. Find the dates on which the shop made total sales (money paid for all products sold on the date) of at least 1000 euro. List these dates and for each date list also the total quantity of products sold.

SQL Foreign Keys

1. Make sure you have the example pcshop schema in your MySQL database.
2. Make sure your tables are stored with the InnoDB engine. MySQL has two alternative engines (physical representation of the tables) called MyISAM and InnoDB. Only InnoDB supports foreign key constraints. Check the structure of your database and if the tables' type is not InnoDB then execute the commands:

```
ALTER TABLE customers ENGINE = InnoDB;
ALTER TABLE laptops ENGINE = InnoDB;
ALTER TABLE pcs ENGINE = InnoDB;
ALTER TABLE printers ENGINE = InnoDB;
ALTER TABLE products ENGINE = InnoDB;
ALTER TABLE sales ENGINE = InnoDB;
```

3. Use the ALTER TABLE statement to specify foreign keys. For example, this is how you can define foreign keys for table Sales:

```
ALTER TABLE sales ADD FOREIGN KEY
    (customer_id) REFERENCES customers(customer_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE;

ALTER TABLE sales ADD FOREIGN KEY
    (model) REFERENCES products(model)
    ON UPDATE CASCADE;
```

4. Test the result of the previous step by:
    - Updating/deleting a customer id in table customers. This operation should result in modifying/deleting the customer id in sales as well.
    - Updating a model number in table products. This operation should result in modifying the model number in sales as well.
    - Deleting a model from products which model is also present in table sales. This operation should be rejected.

SQL Basic Triggers

1. Drop any foreign keys you have defined above as they may conflict with the triggers you will write here.

2. In the SQL window enter the following code:

```
delimiter //

CREATE TRIGGER DeleteCustomer
AFTER DELETE ON sales
FOR EACH ROW BEGIN

IF (OLD.customer_id NOT IN (SELECT customer_id FROM sales)) THEN

DELETE FROM customers WHERE customer_id = OLD.customer_id;

END IF;

END;//
```

3. Can you explain what the code does? Navigate to table sales and delete all sales for a particular customer (e.g., customer 1122334455). Then navigate to table customers and check if that customer (e.g., 1122334455) is still there.

4. Try to add a second trigger for the same event (i.e., AFTER DELETE ON SALES) with a different name. What is the result?

   For example, try to add the trigger:

```
delimiter //
CREATE TRIGGER MarkModel
AFTER DELETE ON sales
FOR EACH ROW BEGIN

IF (OLD.model NOT IN (SELECT model FROM sales)) THEN

UPDATE products SET type='never sold'
WHERE model = OLD.model;

END IF;

END;//
```