Homework Assignment 2

ISCI 410 Introduction to Databases
Fall 2016
Due: 10/23/2016 at 11:59PM*

*Note: For assignments of this scale, expect a week to do them. I am giving more time for this assignment because I want you to use this assignment to prepare for the midterm. You should expect an additional assignment to be given before 10/23/2016, so plan accordingly.

This assignment will use the learning_sql database. The following table shows the primary and foreign keys.

Learning SQL Database Relationships		
Table	**Primary Key**	**Foreign Key**
account	**account_id**	**open_branch_id** *references* **branch.id** **cust_id** *references* **customer.cust_id** **open_emp_id** *references* **employee.emp_id** **product_cd** *references* **product.product_cd**
branch	**branch_id**	
business	**cust_id**	**cust_id** *references* **customer.cust_id**
customer	**cust_id**	
department	**dept_id**	
employee	**emp_id**	**dept_id** *references* **department.dept_id** **assigned_branch_id** *references* **branch.branch_id** **superior_emp_id** *references* **employee.emp_id**
individual		**cust_id** *references* **customer.cust_id**
officer	**officer_id**	**cust_id** *references* **business.cust_id**
product	**product_cd**	**product_type_cd** *references* **product_type.product_type_cd**
transaction	**txn_id**	**execution_branch_id** *references* **branch.branch_id** **account_id** *references* **account.account_id** **teller_emp_id** *references* **employee.emp_id**

Rules:

No collaboration is allowed.

Each of the following requested queries is worth 10 points. Your queries must be logically sound and remain correct even as the database transitions from valid state to another. In other words, if the data in the database were to change, your queries must still provide valid responses. Do not hardcode anything assuming that the data will never change. You can assume that the constraints in the above table will always be satisfied.

For this assignment, the full testing suite will be available on sql-tester.tomchik.net. This means that if you upload your queries and all tests pass you should get a grade of 100%. Note that the *sql-tester* only checks for column names and correct output. I will need to inspect the queries manually to ensure that a subquery was used for query 10. I will also need to make sure that your queries do not contain hardcoded values that would become invalid if the database were to change.

Submission instructions: each query must be in a file named *N*.sql, where *N* is the query number. For example, your answer for query 1 should be in a file named 1.sql. Your final submission should include files 1.sql, 2.sql, …, 10.sql. All .sql files should be submitted in a zip archive with the following naming format: *lastname_firstname_hw2.zip*. For example, my submission would be *tomchik_paul_hw2.zip.*

I recommend that you create a directory on your computer named *lastname_firstname_hw2*. Save all of your queries *(1.sql, 2.sql, …, 10.sql)* in that directory. Then, you can simply create a zip archive of the directory to upload to the sql-tester app and for submission to Blackboard.

Let me add that if you use this naming format for your *sql-tester* uploads, I will be able to find your uploaded scripts on the server. If you have any questions, I will be able to look at your scripts to help you debug them.

For those of you who want a challenge, I encourage you to explore the JOIN syntax

- http://dev.mysql.com/doc/refman/5.7/en/join.html
- https://www.postgresql.org/docs/9.5/static/queries-table-expressions.html

This was not covered in the class lectures, but the concepts were covered in the Relational Algebra lectures and self-teaching is a very valuable skill in programming. The column naming in this database is conducive to that syntax.

Please do not hesitate to send me questions, however please *do not give up on yourself too easily*.

Regretfully, not all of these queries can be performed on the Relax app. I would recommend that you use that app to explore the data, but not much else.

Queries:

1. For the **account** table, show
    a. the total of all avail_balances as **total_avail_balance**,
    b. the average of all avail_balances as **avg_avail_balance**,
    c. the minimum of all avail_balances as **min_avail_balance**,
    d. the maximum of all avail_balances as **max_avail_balance**,
    e. the count of all avail_balances as **count_avail_balance**.

    The result of the query should be a single row with the above specified column names.
2. Show the **account_id** and **avail_balance** for all checking **account**s (i.e. account.product_cd = 'CHK') where the available balance is more than twice the average checking account balance.
3. Show the total number of **transaction**s where the teller_emp_id is null. The column name should be **nonteller_txn_ct**.
4. Show the **name** of every **branch** where no **transaction** took place.
5. Show the **cust_id** and **total** of avail_balances across all **account**s for each customer.
6. Show the **cust_id**, **fname**, **lname**, and the **account** **product_cd** of all **individuals** who have only one account.
7. Show all **business** **name**s where an **officer**'s last name appears in the business name.
8. Show the 6 highest **avail_balance**s in **account** *in descending order*. An avail_balance with duplicates should only count once. For example, consider the following list of numbers: (1,2,3,3,4). The highest three numbers would be (4,3,2). HINT: Look at the slide for "Query 57: Better way to find the 6 highest salaries in faculty". WARNING: phpMyAdmin may complain about non-existent errors.
9. Show the names of the **individual**s with the closest birth_dates. The individual names should be formatted as 'lname, fname'. I.E 'Smith, Joe'. The column name for the older individual's name should be **individual_1**. The column name for the older individual's name should be **individual_2**.

    HINTS:
    1. Look at "Query 37: This solves the original problem". You need to use the idea of leveraging information obtained in a subquery.
    2. Use the subtraction operator, -, to get relative differences between the birth dates.
10. Show the **account_id**, **teller_txn_total**, and **nonteller_txn_total** for each account, where teller_txn_total is sum of all amounts for teller **transaction**s (teller_emp_id is not null), and nonteller_txn_total is the sum of all non-teller transaction amounts. Order by the ratio of teller_txn_total to the total amount of all transactions for that customer. HINT: Use UNIONS. WARNING: again, phpMyAdmin may again complain about non-existent errors.

NOTE:

1. The required column names are in bold black text. The required table names are in bold blue text.
2. phpMyAdmin's syntax checker reports false errors. It may complain about non-existent problems. You may need to ignore them and run the query to get feedback from MySQL itself. I encountered these for queries 8 and 10.
3. Bonus points will be given to bug-finders. If you discover a *significant* problem (such as a class app not working, not a minor typo) and bring it to my attention, I will give you a bonus point on the assignment. It pays to start the assignments early.