# Brandon Zink

# **CSCI 3287: Homework 2**

**Due Date**

9:55p Fri. Oct. 27, 2017

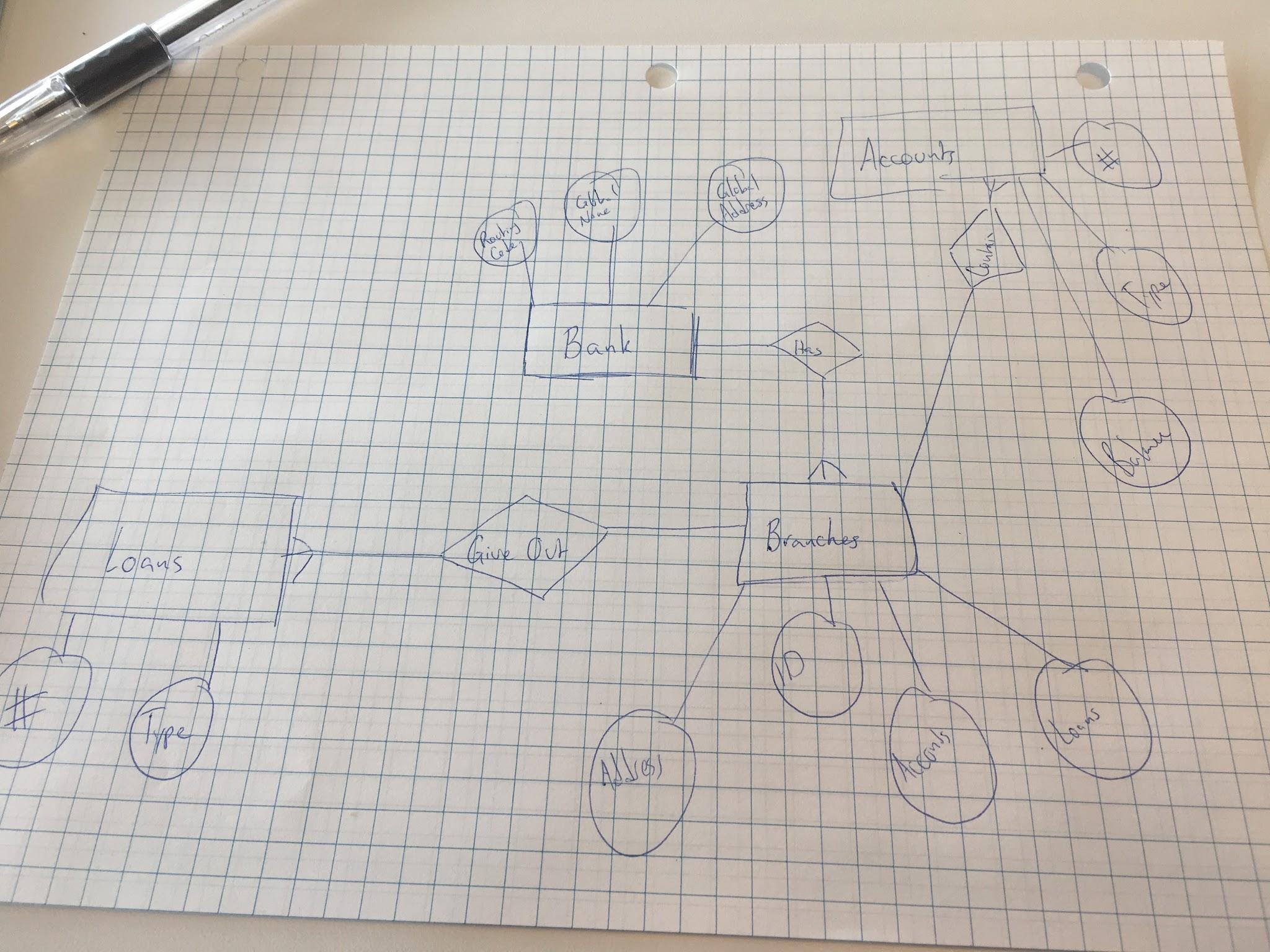
No late submissions will be accepted.

**Delivery Method**

Submit a Microsoft Word or PDF document containing your answers using Moodle.

**Problem 1:** Entity Relationship Diagram

Consider the problem of creating a database for a national bank. The bank has a routing code associated with it. We call the bank by its global name and correspond with it using a global address. Each bank has a number of physical branches that have at least an address. Banks manage accounts which typically have account numbers, types and balances. Loans are given locally by branches for various reasons and are tracked with loan numbers and types. Make sure you indicate the appropriate keys and multiplicity relationships.



The problem was unclear as far as whether the bank or branch manages the account. I attached it to the branch, but if the question was meant to be the main bank, it would be moved there.

**Problem 2**: Checking Functional Dependencies

For each part of this problem you will provide a *single* SQL query to check whether a certain condition holds on a specific instance of a relation in the following way: **your query should return an empty result if and only if the condition holds on the instance.** (If the condition *doesn't hold*, your query should return something non-empty, but it doesn't matter what this is).

Note our language here: the conditions that we specify cannot be proved to hold **in general** without knowing the externally-defined functional dependencies. This means that you are *checking whether they* ***could*** *hold in general for the relation, given any specific set of tuples*.

You may assume that there will be no NULL values in the tables, **and you may assume that the relations are *sets* rather than multisets**, but otherwise your query should work for general instances. We define the schemas of the tables used below for convenience, but in this problem you will need to construct your own test tables if you wish to use them to check your answers!

1. {A,B} -> {C} and {C} -> {A,B} holds for relation R(A,B,C,D,E)

SELECT R1.A, R1.B, R1.C FROM R AS R1, R AS R2 WHERE R2.A=R1.A AND R2.B=R1.B AND R1.C <> R2.C;

Here we take that R1 ad R2 are duplicates of R, and check for any column where A and B are the same but they are not equal to C.

1. {A,B,C} is a **superkey** for a relation R(A,B,C,D,E)

SELECT \* FROM R AS R1, R AS R2 WHERE (R1.A = R2.A AND R1.B =

R2.B AND R1.C = R2.C) AND (R1.D <> R2.D OR R1.E <> R2.E);

Similar to part A, just checking where A and B and C are the same but D and E are not.

1. {A} and {C} are keys for a relation S(A,B,C)

SELECT \* FROM S AS S1, S AS S2 WHERE ((S1.A = S2.A) AND (S1.B <>

S2.B OR S1.C <> S2.C)) OR (S1.C = S2.C) AND (S1.B <> S2.B OR S1.A <> S2.A);

Same as A and B, but instead we check twice for both keys A and C.

**Problem 3:** Double Buffering with IO

This problem explores an optimization often referred to as ***double buffering***, which we'll use to speed up the **external merge sort algorithm**.

Recall that *sequential IO* (i.e. involving reading from / writing to consecutive pages) is generally much faster that *random access IO* (any reading / writing that is not sequential). Additionally, on newer memory technologies like SSD reading data can be faster than writing data.

In other words, for example, if we read 4 consecutive pages from file A, this should be much faster than reading 1 page from A, then 1 page from file B, then the next page from A.

Assume that 3/4 sequential *READS* are "free", i.e. the total cost of 4 sequential reads is 1 IO. We will also assume that the writes are always twice as expensive as a read. Sequential writes are never free, therefore the cost of N writes is always 2N.

* **NO REPACKING:** Consider the external merge sort algorithm using the basic optimizations but do not use the repacking optimization
* **ONE BUFFER PAGE RESERVED FOR OUTPUT:** Assume we use one page for output in a merge, e.g. a B-way merge would require B+1 buffer pages
* **REMEMBER TO ROUND:** Take ceilings (i.e. rounding up to nearest integer values) into account in this problem for full credit! Note that we have sometimes omitted these (for simplicity) in lecture
* **Consider worst case cost:** In other words, if 2 reads *could happen* to be sequential, but in general might not be, consider these random IO

Consider a modification of the external merge sort algorithm where **reads are always read in 4-page chunks (i.e. 4 pages sequentially at a time)** so as to take advantage of sequential reads. Calculate the cost of performing the external merge sort for a setup having B + 1 = 20 buffer pages and an unsorted input file with 160 pages.

Show the steps of your work and make sure to explain your reasoning by writing them as python comments above the final answers.

1. Give the **exact** IO cost of splitting and sorting the files? As is standard we want runs of size B + 1

Read = 160/4 #reading from disk in 4 page chunks

Write = 160\*2 #writing back to the disk

360 IO Cost

1. How many passes of merging are required?

There are 3 passes of merging required. Pass 1 will sort from 8 sorted chunks to 4, then pass 2 from 4 sorted chunks to 2, then pass 3 will sort from 2 sorted chunks into 1 final sorted chunk.

1. What is the IO cost of the first pass of merging? Note: the highest arity merge should always be used.

The first pass of merge will require reading in all data in 40 page increments, then incurring the merge cost, then writing those chunks back out to the disk

160/4 = 40 cost read in

160\*2 = 320 write back out

40+320=360

1. What is the total IO cost of running this external merge sort algorithm? **Do not forget to add in the remaining passes (if any) of merging.**

You take the total cost for each pass of merging and multiply the total number of merging, so (360\*3) = 1080 IO cost.

If you also take into account the sort part, you get (360\*4) = 1440 IO cost, but the CA said that you shouldn’t include this because Frank.