University of Toronto
CSC343, Summer 2019

Assignment 3

Due: Monday, August 5, 2019, before 10:00 pm

Jingqi Zhang, 1004117671

Qingyang Li, 1002741063

You may work with a partner. Note that your submission must be **typed** and not handwritten. More details are in the Submission Instructions section at the end of the handout.

Database Design and SQL DDL

1. Consider the relation $R(A, B, C, D, E, F)$. Let the set of FD's for R be $\{A \rightarrow B, CD \rightarrow A, CB \rightarrow D, CE \rightarrow D, AE \rightarrow F\}$.

   (a) What are all of the keys for R?

   CE

   (b) Do the given FD's form a minimal basis? Prove or disprove.

   To find the minimal basis, we need to check whether the given FDs have redundancy FDs.

   We first check each LHS's closure.
   $A^+ = \{A,B\}$
   $CD^+ = \{C,D,A,B\}$
   $BC^+ = \{B,C,D,A\}$
   $CE^+ = \{C,E,D,A,B,F\}$
   $AE^+ = \{A,E,F,B\}$
   Now we reduce the trivial FDs and split RHS by the closure.
   we get the new set S of FDs $\{A \rightarrow B, CD \rightarrow A,\ CD \rightarrow B, BC \rightarrow A,\ BC \rightarrow D,\ CE \rightarrow B,\ CE \rightarrow A,\ CE \rightarrow F,\ AE \rightarrow B,\ AE \rightarrow F\}$.
   Now we reduce the redundant FDs by FD properties: $\{A \rightarrow B, CD \rightarrow A, BC \rightarrow A,\ BC \rightarrow D,\ CE \rightarrow A,\ CE \rightarrow F,\ AE \rightarrow F\}$.

   (c) Provide a decomposition of R into 3NF-satisfying relations.

   The minimal basis is $A \rightarrow B, CD \rightarrow A, BC \rightarrow A,\ BC \rightarrow D,\ CE \rightarrow A,\ CE \rightarrow F,\ AE \rightarrow F$.

   So we get AB, ACD , ABC, BCD, ACE,CEF. Since CEF is a superkey, we don't have

to add any more relation. Those above are all wanted.

(d) Are any of the relations that you made in part (c) not in BCNF?

Since CE is the key for R, after projecting the FDs on new relations, we found that FDs on AB, ACD , ABC, BCD are not include superkey. So they are not in BCNF.

2. Answer the following questions.

(a) Prove or disprove the following:

Suppose a relation R is decomposed into R1 and R2 with one common attribute between the two new relations

If the common attribute between R1 and R2 forms a key for at least one of R1 or R2, then the decomposition is lossless.

By definition: If a relation R is decomposed into R1 and R2, then we would say that the decomposition is lossless if one of the following happens;

$(R1 \cap R2) \rightarrow R1$

$(R1 \cap R2) \rightarrow R2$

Since
$R1 \cap R2 = key$
$key \rightarrow R1$
$key \rightarrow R2$
Lossless proved.

(b) Can a relation and a set of FDs be in both BCNF and 3NF at the same time? If so, explain what conditions must be met. If not, explain what is preventing this from being possible.

Yes

3NF : FD $X \rightarrow Y$ satisfies 3NF iff X is a superkey or Y is prime.

BCNF : For every nontrivial FD $X \rightarrow Y$ that holds in R, X is a superkey.

So, the condition that "For every nontrivial FD $X \rightarrow Y$ that holds in R, X is a superkey." must hold to satisfy BCNF and 3NF at the same time.

3. Prove or disprove that:

(a) If $A \rightarrow B$ then $B \rightarrow C$

It can be disproved by the following counterexample which only satisfies $A \to B$ but not $B \to C$

A	B	C
a1	b1	c1
a1	b1	c2
a2	b1	c3

(b) If $AB \to C$ then $A \to C$ and $B \to C$

It can be disproved by the following counterexample which only satisfies $AB \to C$ but not $A \to C$ and $B \to C$

A	B	C
a1	b1	c1
a1	b2	c2
a2	b1	c3

Continued on next page...

4. **Design and DDL**

Consider the following domain:

You are running a Fresh Juice business with multiple stores around the country, and you want to keep the information for these stores in a relational database. The following is a list of that information:

- Stores: Each store has a city, phone number, and manager. There is only one store per city.
- Beverages: Each juice beverage has a name (e.g. Kiwi Lime), and a number of calories for a regular and large size. A large size always has 200 more calories than the regular size. Every store should keep track of the number of inventory of each beverage (how much it has left in stock).
- Transactions: When a customer makes an order, that order should have a date, price, and an indication of which loyalty card was used, if applicable. You can assume one beverage is ordered per transaction, and we should know what that beverage was.
- Loyalty card: Customers can have a loyalty card if they like to go to your stores a lot. There needs to be information on how many transactions a customer made with the card, and their home store (the one they go to most frequently).

(a)Define a single relation for this domain that manages to store all of the required information (just write it out R(...), no need for SQL definitions yet). There is not necessarily one correct answer for this relation, but the information should be stored in a practically useful way. It is ok to add attributes that arent explicitly listed in the domain as long as they are useful.

R(city, phone, manager, juice, cal, size+, stock, trans_id, date, price, card_id, #order, home_store)

(b)Write all of the functional dependencies for your relation that would be inferred by the description of this domain. Do not include trivial or redundant FDs (find a minimal basis).

Let the set of FD's for R be F.
Then F = {city →phone; city →manager; juice →cal; city, juice →stock; trans_id →date; trans_id →price; trans_id → card_id; trans_id →juice; trans_id →size+; trans_id →city; trans_id, card_id → #order; card_id →home_store}

(c)Provide a useful instance of your relation that shows all three types of anomalies. Describe the anomalies you have presented as they appear in your particular instance (give an example for each of the three anomalies in your relation).

For example, in the table below:

city	phone	manager	juice	cal	size+	stock	trans_id	date	price	card_id	#order	home_store
Trt	123	Jinny	Lime	180	false	178	1	07-06	1.99	7777	7	Trt
Trt	123	Jinny	Lime	180	false	177	2	07-06	1.99	7777	8	Trt
Trt	123	Jinny	Lime	180	false	176	6	07-06	1.99	7777	12	Trt
Trt	123	Jinny	Lime	180	false	175	7	07-07	1.99	7777	21	Trt
Trt	123	Jinny	Lime	180	false	170	10	07-08	1.99	7777	29	Trt
NY	222	Adam	Grape	204	false	89	12	08-07	3.99	7777	23	Trt

• *Update anomalies*

Changing the genre for the " Trt, 123, Jinny " tuples in one tuple (ex: we update the 1st instence row "Jinny" to "Tom") then we have to update all the other four " Trt, 123, Jinny " tuples (change to " Trt, 123, Tom ") since we have FDs such as: city →phone; city →manager. Otherwise if we don't update all, it will create inconsistent data(each city has only one store).

• *Redundancy anomalies*

There are lots of duplicate information for the " Trt, 123, Jinny "tuples and " Lime, 180" tuples and " 7777, Trt " tuples(ex: there are 5 instance rows in the table have those duplications but indicate the different transcation orders). Those are unnecessary repetition of information since we have FDs such as: city →phone; city →manager; juice →cal.

• *Delation anomalies*

Removing the " Grape " as juice name can remove the " NY, 222, Adam " tuples entirely, which results in unwanted loss of this whole NY store information.

(d) Your relation will likely (read certainly) have some redundancy. Decompose your relation into a set of relations without any BCNF violations. Write all of your steps in full and clearly show why your relations do not violate BCNF.

Let the relation be R(city, phone, manager, juice, cal, size+, stock, trans_id, date, price, card_id, #order, home_store), which we defined in a).

Let the set of FD's for R be F = {city →phone; city →manager; juice →cal; city, juice →stock; trans_id →date; trans_id →price; trans_id → card_id; trans_id →juice; trans_id →size+; trans_id →city; trans_id, card_id → # order; card_id →home_store}, which we showed in b).

Now we check every FD in F by BCNF violation and split our R to smaller schemas, using BCNF Decomposition algorithm.

For city →phone,
obviously it violates BCNF since city is not the superkey of R.
Then compute city$^+$ = {city, phone, manager}.
So now R split into to R_1(city, phone, manager) and R_2(city, juice, cal, size+, stock, trans_id, date, price, card_id, #order, home_store).
And since the project FDs of F onto R_1 is {city →phone; city →manager} and it satisfies the BCNF by definition, we stop recursively split R_1, denote as ***Store****. Now we continue splitting R_2. Notice that the project FDs of F onto R_2 is { juice →cal; city,*

juice →stock; trans_id →date; trans_id →price; trans_id → card_id; trans_id →juice; trans_id →size+; trans_id →city; trans_id, card_id → # order; card_id →home_store}, we denote as F_2.

For juice →cal,
obviously it violates BCNF since juice is not the superkey of R_2.
Then compute $juice^+$ = {juice, cal}.
So now R_2 split into to R_3(juice, cal) and R_4(city, juice, size+, stock, trans_id, date, price, #order, home_store).
*And since the project FDs of F onto R_3 is { juice →cal} and it satisfies the BCNF by definition, we stop recursively split R_3, denote as **JuiceType**. Now we continue splitting R_4. Notice that the project FDs of F onto R_4 is {city, juice →stock; trans_id →date; trans_id →price; trans_id → card_id; trans_id →juice; trans_id →size+; trans_id →city; trans_id, card_id → # order; card_id →home_store}, we denote as F_4.*

For city, juice →stock,
obviously it violates BCNF since city, juice is not the superkey of R_4.
Then compute {city, juice}$^+$ = {juice, city, stock}.
So now R_4 split into to R_5(juice, city, stock) and R_6(juice, city, size+, trans_id, date, price, card_id, #order, home_store).
*And since the project FDs of F onto R_5 is { juice, city →stock} and it satisfies the BCNF by definition, we stop recursively split R_5, denote as **StoreStock**. Now we continue splitting R_6. Notice that the project FDs of F onto R_6 is {trans_id →date; trans_id →price; trans_id → card_id; trans_id →juice; trans_id →size+; trans_id →city; trans_id, card_id → # order; card_id →home_store}, we denote as F_6.*

For card_id → home_store,
obviously it violates BCNF since card_id is not the superkey of R_6.
Then compute $card\_id^+$ = {card_id, home_store}.
So now R_6 split into to R_7(card_id, home_store) and R_8(juice, city, size+, trans_id, date, price, card_id, #order).
*And since the project FDs of F onto R_7 is {card_id, home_store} and it satisfies the BCNF by definition, we stop recursively split R_7, denote as **LoyaltyCard**. Now we continue splitting R_8. Notice that the project FDs of F onto R_8 is {trans_id →date; trans_id →price; trans_id → card_id; trans_id →juice; trans_id →size+; trans_id →city; trans_id, card_id → # order}, and it satisfies the BCNF by definition (both$trans_id^+$ and {trans_id, card_id}$^+$ are {trans_id, date, price, city, size+, juice, card_id, # order}) . So we denote R_8 as **Transactions**.*

*Hence **Store**(city, phone, manager), **JuiceType**(juice, cal), **StoreStock**(juice, city, stock), **LoyaltyCard**(card_id, home_store) and **Transactions**(juice, city, size+, trans_id, date, price, card_id, #order) are the set of decomposed relations required.*

(e) Explain how your new relations prevent the anomalies you pointed out in part (c).

Since we decompose the single relation R into 5 relations, and each relation has guaranteed in BCNF, the functionality and purpose of each relation has been more specific.

Take the example I use in part (c), now the table can be splitted into 5 smaller table:

Store

city	phone	manager
Trt	123	Jinny
NY	222	Adam

JuiceType

juice	cal
Lime	180
Grape	204

LoyaltyCard

card_id	home_store
7777	Trt

StoreStock

city	juice	stock
Trt	Lime	170
NY	Grape	89

Transactions

city	juice	size+	trans_id	date	price	card_id	#order
Trt	Lime	false	1	07-06	1.99	7777	7
Trt	Lime	false	2	07-06	1.99	7777	8
Trt	Lime	false	6	07-06	1.99	7777	12
Trt	Lime	false	7	07-07	1.99	7777	21
Trt	Lime	false	10	07-08	1.99	7777	29
NY	Grape	false	12	08-07	3.99	7777	23

As you can see, the Redundancy anomalies we mentioned in part (c) has been prevented, such as 5 redundant "Trt, 123, Jinny" tuples and " Lime, 180" tuples and " 7777, Trt " tuples are now simplified as only 1 in Store ,JuiceType and LoyaltyCard, without unnecessary duplications.

In addition, now if we update "Trt, 123, Jinny" tuples to "Trt, 123, Tom", we only need to update the tuple in Store relation, for only one row, without any inconsistent information, that is we prevent the Update anomalies.

And it is also clear that now we remove "Grape" as juice name, we either remove from the JuiceType or Transcations, both will not affect the store information "NY, 222, Adam" tuples.

(f) **SQL DDL:**

Using your new relations from part (d), create a schema using the SQL DDL language. They should have proper relation names, attribute names and types, and constraints (including keys, foreign key, unique, not null, etc.).

You should add **comments** above each table and attribute describing what it represents. You can add comments using a double dash –. You should also insert some useful data into each of the relations (directly in the ddl file, not through csv).

Use the DDL files from the lectures and A2 as examples to help you make them. Unlike A2, you do not need to write any SQL queries for this part.

What to hand in for this part: In your A3.pdf file, describe the decisions for any constraints you put in your DDL file. Hand in a file called fruits.ddl containing your schema, as well as a plain text file called fruits-demo.txt that shows you starting postgreSQL, successfully importing fruits.ddl, and exiting posgreSQL.

This is similar to what you did in the preps. You must hand in this demo and the file must be a plain text file or you get zero for this part of the assignment.

Shown in required files.

Submission instructions

Your assignment must be typed; handwritten assignments will not be marked. You may use any word-processing software you like.

For this assignment, hand in a file A3.pdf that contains your answers to the questions above. Also hand in fruits.ddl and fruits-demo.txt.

You must declare your team and hand in your work electronically using the MarkUs online system. Well before the due date, you should declare your team and try submitting with MarkUs.