1. [20 pts]

To expand our business to more universities, we have been asked to add the idea of course sections to our design. Specifically, each course will now have at least one section, and meetings will be associated with sections instead of with courses. Sections will have IDs that uniquely identify them across all courses. In addition, we want all sections to have **different names**. Based on his understanding of these requirements, one of our old school database designers -- one of those "go straight to the tables" folks -- has proposed the following design to **replace** the current SWOOSH course table.

```
CREATE TABLE CourseSection (
    course_id text,
    course_name text,
    course_description text,
    section_id text,
    section_name text,
    PRIMARY KEY (course_id, section_id, section_name)
);
```

(a) [5 pts] Looking at the primary key constraint of this table, what non-trivial functional dependency (or dependencies) can you infer, if any?

course_id, section_id , section_name ⟶ course_name, course_description

(b) [5 pts] An intern on the team -- a CS122A survivor -- says that this design doesn't make sense to her after looking at the proposed CREATE TABLE statement. When you ask her why, she vaguely says that it has several "key problems", but she doesn't want to say more for fear of offending the design's proposer. Your job is to correct the above design by fixing its constraints, e.g., by offering a corrected PRIMARY KEY clause. If any additional constraint(s) are needed to properly capture the described nature of sections after this change, write those additional constraint(s) below as well.

PRIMARY KEY (section_id) UNIQUE(section_name)
or
PRIMARY KEY (section_name) UNIQUE(section_id)

(c) [5 pts] Considering your constraints, and also referring back to the constraints for the original Course entity and its resulting table, what non-trivial functional dependencies does the revised table have?

course_id ⟶ course_name, course_description

section_id ⟶ section_name

section_name ⟶ section_id

section_id → course_id

(d) [5 pts] If CourseSection is not already at least in 3NF, then normalize it into at least 3NF and show the resulting relation(s) and specify their candidate keys. Make sure that your 3NF decomposition is both lossless-join and dependency-preserving. Note: If CourseSection was already in 3NF, then just list the candidate keys of CourseSection. What is the highest normal form that your answer now satisfies?

Course(course_id, course_name, course_description) Candidate key: course_id

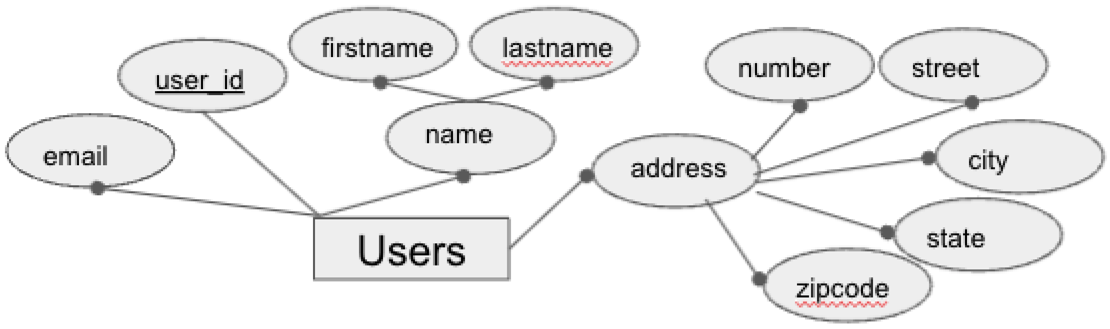
Section(section_id, course_id, section_name) Candidate keys: section_id, section_name

BCNF

2. [25 pts]

Your boss at SWOOSH, not a big tree-lover, wants to send physical greeting letters to all users. You realize that a physical address is thus needed for users in addition to their email address. The physical address should include the usual: number, street, city, state, and zip code. Assume that a zip code can include multiple cities but **not** multiple states[1].

(a) [5 pts] Draw a revised E-R diagram for the Users entity. Your diagram should only show the Users entity set with all of its attributes. You may draw your diagram with any software you prefer, take a screenshot, and paste it below.



(b) [5 pts] Provide appropriate table creation DDL for the revised Users table (which we will be referring to as Users for the rest of this problem):

```
CREATE TABLE cs122a_fall21.Users (
        user_id text,
        email text NOT NULL,
        first_name text NOT NULL,
        last_name text NOT NULL,
        address_number text NOT NULL,
        address_street text NOT NULL,
        address_city text NOT NULL,
        address_state text NOT NULL,
        address_zipcode text NOT NULL,
        PRIMARY KEY(user_id)
);
```

(c) [5 pts] What are the functional dependencies that hold on your Users table?

[1] This is not true in the real world, but you are to assume it here. (Don't argue with your boss. :-))

user_id ⟶ email, first_name, last_name, address_number, address_street, address_city, address_state, address_zipcode
address_zipcode ⟶ address_state

(d) [5 pts] Normalize the Users table into 3NF, show the resulting relations, and specify their candidate keys. Make sure that your 3NF decomposition is both lossless-join and dependency-preserving. What is the highest normal form that your answer now satisfies?
Users(user_id, email, first_name, last_name, address_number, address_street, address_city, address_zipcode) CK: user_id
ZipcodeState(address_zipcode, address_state) CK: address_zipcode
BCNF

(e) [5 pts] Take a thoughtful look at the Users table in the resulting decomposed tables and what they will mean for the SWOOSH application and query team. Is it definitely a good idea to use these tables instead of the one in part (b)? Why or why not? (Identify any tradeoff(s) you see.)
No, decomposing the Users table reduces the redundancy by only a little, but the semantic of the address is broken. We don't want state to be stored in a separate table because that requires join whenever querying on address. It's not worthy to reduce the performance in this case.

3. [15 pts]

Consider the following relation:

F	**E**	**D**
f_5	e_1	d_4
f_8	e_3	d_1
f_1	e_3	d_1
f_4	e_2	d_2
f_4	e_5	d_2
f_1	e_4	d_1

(a) [10 pts] Given the current state of the database, and for each one of the following functional dependencies, answer: a) Does this functional dependency hold in the above relation instance [Yes/No]? b) If your answer to the previous question was "No", explain why by listing a tuple that causes a violation.

i) F ⟶ E

No: (f_4, e_2, d_2), (f_4, e_5, d_2) or (f_1, e_3, d_1), (f_1, e_4, d_1)

ii) E ⟶ D

Yes

iii) D⟶ E

No: (f_4, e_2, d_2), (f_4, e_5, d_2)

iv) E ⟶ F

No: (f_8, e_3, d_1), (f_1, e_3, d_1)

v) F ⟶ D

Yes

(b) [5 pts] List all ***potential*** candidate keys (if there are any) for the above relation that can be inferred from the given relation instance.

(EF)

Normalizing a schema with a set of FDs can be done automatically by computers. Complete questions 4 and 5 with the help of the normalization tool provided by Griffith University - *but try each part by hand first*! Use the problems to cement your understanding and use the tool to check your answers.

4. [20 pts]

Order(order_id(O), product_id(P), store_id(S), customer_id(C), customer_name(N))

(All attributes contain only atomic values.)

FD1: OP $\longrightarrow$ SC

FD2: O $\longrightarrow$ CN

FD3: P $\longrightarrow$ S

FD4: C $\longrightarrow$ N

(a) [5 pts] Compute OP+, the attribute closure of the attribute set (order_id, product_id). Show each step of your work as well as your final result.

OP+ = {OPSCN}

(b) [5 pts] List the candidate keys of Order. Is this related to your answer to (a)? Why?

OP - yes - because my answer to (a) means OP -> everything!

(c) [5 pts] What's the highest normal form that Order satisfies and **why**?

1NF. (It violates 2NF because some non-prime attributes are dependent on part of a candidate key, e.g., O $\longrightarrow$ CN)

(d) [5 pts] If Order is not already at least in 3NF, then normalize Order into 3NF and show the resulting relation(s) and specify their candidate keys. Make sure that your 3NF decomposition is both lossless-join and dependency-preserving. Note: If Order was already in 3NF, then just list the candidate keys of R. What is the highest normal form that your answer now satisfies?

Solution 1:

R0(C, N), candidate key C

R1(O, C), candidate key O

R2(P, S), candidate key P

R3(O, P), candidate key (O, P)
BCNF

5. [20 pts]

Meeting(meeting_id(M), passcode(P), instructor_id(I), instructor_name(N), course_id(C), course_description(D), recording(R))

(All attributes contain only atomic values.)

FD1: M ⟶ P

FD2: M ⟶ ICR

FD3: I ⟶ N

FD4: C ⟶ D

(a) [5 pts] Compute M+, the attribute closure of attribute M. Show your work and final result.

M+ = {MPINCDR}

(b) [5 pts] Is M ⟶ D in FD+, the closure of the set of FDs? Why or why not?

Yes, because D is in M+.

(c) [5 pts] Normalize R into BCNF and show the resulting relation(s) and their candidate keys.

R0(M,P,I,C,R): M

R1(I, N): I

R2(C, D): C

(d) [5 pts] Is the decomposition in part (c) dependency-preserving? Why or why not?

Yes. The closure of the set of all FDs hold on the decomposed relations(FD1,2,3,4) is the same as the closure of the original FDs. Or$\{FD1,\ FD2,\ FD3,\ FD4\}^{+} = \{FD1,\ FD2,\ FD3, FD4\}^{+}$