Stage IV
CSC 315-01
30 March 2020

STAGE IV: Group 9

315: Julian De La Cruz, Alhasssane Traore, Adam Mellan, Amulya Badineni

PART 1: BCNF

Table 1

1. User

eMail	Password	create_date	fname	lname	county	city	zip_code

   - This table is not in BCNF because of data redundancy, a deletion anomaly, and its form is only in 1NF. The table does satisfy these 4 requirements of 1NF: each column has atomic values, the values under each attribute are of the same type, the column names are unique, and the order of values doesn't matter. The first issue to address is data redundancy. The attributes *county*, *city*, and *zip_code* can add repetition of similar data in multiple places and also cause anomalies. A deletion anomaly can occur if we have a user whose county is unique. For example, say the database contains one individual from Mercer County. If the user decides to delete his/her account, then that would cause a deletion in not only the user's account credentials but also his/her residential information that has not been saved in any other table. And so, we would accidentally lose a related dataset in the process of deleting another dataset. To fix the anomaly, we create another table where only the residential information is stored (as shown in the next bullet point).
   - Residence (the table as a result of the normalization of USER):

user_id	county	city	zip_code

   - User (the modified table after normalization of USER):

user_id	eMail	password	create_date	fname	lname

2. User (after part 1)

user_id	eMail	password	create_date	fname	lname

- The table is now in BCNF because there is no data redundancy, partial dependencies have been removed by creating another table, and transitive dependencies do not exist as the attributes depend on the composite key (user_id + eMail) can uniquely identify any row of the USER table.
- FUNCTIONAL DEPENDENCIES:
  - user_id eMail → password, create_date, fname, lname

3. Residence (after and from part 1)

user_id	county	city	zip_code

- The table is in BCNF because there is no data redundancy, partial dependencies do not exist as all the attributes are dependent on the primary key, and transitive dependencies do not exist as the attributes depend on the primary key (user_id) can uniquely identify any row of the Residence table.
- FUNCTIONAL DEPENDENCIES:
  - user_id → county, city, zip_code

Table 2

1. Preferences

pref_regions	notifications	p_eMail	interest_1	interest_2	interest_3

- This table is not in BCNF because all attributes do not depend on the key. It may have data redundancy and we have two separate, but related datasets, in one table. To convert this table to BCNF, we have to separate the two related datasets (as shown in the next bullet points) and create another table.
- Preferences (new table after normalization):

user_id	notification	pref_region

- Interests (the table that resulted from the normalization of PREFERENCES):

user_id	interest_1	interest_2	interest_3

2. Preferences (after part 1)

user_id	notification	pref_region

- This table is in BCNF because all attributes depend on the primary key, which leaves no room for any non-prime key to depend on another non-prime key.
- FUNCTIONAL DEPENDENCIES:
  - user_id → notification, pref_region

3. Interests

user_id	interest_1	interest_2	interest_3

- This table is in BCNF because all attributes depend on the primary key. The different interests refer to the sustainability issues a user has an interest in. The 3 interests, listed as attributes in the table, cannot be NULL because a user will have to choose at least 3 changes to move to the next step of creating an account.
- FUNCTIONAL DEPENDENCIES:
  - user_id → interest_1, interest_2, interest_3

Table 3

1. Newsfeed

user_id	media_type

- This table represents the type of content the user will see in his/her newsfeed. Based on the user's preferences, he/she will see videos, forums, podcasts, and images in his/her newsfeed. This table is not in BCNF because we had assumed that the attribute media_type can have multiple values, but the values of an attribute can only be atomic. We solved this problem by creating another table (number 2).

2. Newsfeed(after normalization)

media_id	user_id	media_type

- This table is in BCNF because the attribute media_type depends on the primary key media_id. The attribute media_id would increment every time a new piece of media, an article, video, or image, is uploaded. So, each piece of media has a unique identification number that will allow for an easy retrieval process. The attribute user_id can be used to determine what piece of media a user has posted. We didn't form user_id and media_id into a composite key because user_id is specific to the user and media_id refers to the content in the whole database.
- FUNCTIONAL DEPENDENCIES:
  - media_id → user_id, media_type
  - user_id → media_type

****The highlighted tables are our final products after normalization and what we will be using going forward****

PART 2: VIEWS

1. To ensure the security of information from unauthorized users, we will be using query authorization statements, GRANT and REVOKE, in order to hide certain tuples. In general, a user will be able to see the basic information of other users: name, zip code, and whatever other content they wish to be made public (county, city, interests). The information that will never be seen by a user is the password, user_id, and media_id. These are for the backend users only.
2. One view that will be needed is the join between User, Preferences, and Newsfeed. This output is important because it ties together a user's preferences with their sign in and newsfeed information.
    a. Data: user's password and ID as well as their preferences and newsfeed information
    b. Transaction: Verification of user's password and ID in order to access their preferences and display the correct newsfeed
    c. Query: for this transaction would be to select the user ID and password to verify they are the correct pair to login with.
3. Since the view is supposed to be always up-to-date, the preferred way of efficiently implementing a view would be through view materialization. The immediate update strategy, for updating any changes made in the base tables appear in the view, seems to be the best decision for our database. Mainly because we assume that most of the information provided by the user will remain constant with the exception of a few changes to per year. Since the changes are few and will most likely happen around the same time, using the immediate update strategy will be less complex, quick, and relatively easy.
4. To implement user interaction, we will allow the user to search for the names of other users, interests, regions, and types of media.
    a. search for a friends in the local area
    b. list other sustainability areas
    c. search for articles, videos, images
    d. a user can delete any media that has been posted by him/her but nothing related to another user
5. The database administrators are the only ones who can INSERT, UPDATE, and DELETE within the database.

UPDATED RELATIONAL SCHEMA: